Introduction

STM32CubeMX is a graphical tool for STM32 microcontrollers. It is part of the STM32Cube™ initiative (see Section 1) and is available either as a standalone application or as an Eclipse plug-in for integration in integrated development environments (IDEs).

STM32CubeMX has the following key features:

- **Easy microcontroller selection** covering the whole STM32 portfolio
- **Board selection** from a list of STMicroelectronics boards
- **Easy microcontroller configuration** (pins, clock tree, peripherals, middleware) and generation of the corresponding initialization C code
- **Easy switching to another microcontroller** by importing a previously-saved configuration to a new MCU project
- **Easy exporting of current configuration to a compatible MCU**
- **Generation of configuration reports**
- **Generation of embedded C projects** for a selection of integrated development environment tool chains. STM32CubeMX projects include the generated initialization C code, MISRA 2004 compliant STM32 HAL drivers, the middleware stacks required for the user configuration, and all the relevant files for opening and building the project in the selected IDE.
- **Power consumption calculation** for a user-defined application sequence
- **Self-updates** allowing the user to keep STM32CubeMX up-to-date
- Download and update of STM32Cube embedded software required for user application development (see Appendix E for details on the STM32Cube embedded software offer)

Although STM32CubeMX offers a user interface and generates C code compliant with STM32 MCU design and firmware solutions, users need to refer to the product technical documentation for details on actual implementations of microcontroller peripherals and firmware.

The following documents are available from [www.st.com](http://www.st.com):

- STM32 microcontroller reference manuals and datasheets
- STM32Cube HAL/LL driver user manuals for STM32F0 (UM1785), STM32F1 (UM1850), STM32F2 (UM1940), STM32F3 (UM1786), STM32F4 (UM1725), STM32F7 (UM1905), STM32G0 (UM2303), STM32G4 (UM2570), STM32H7 (UM2217), STM32L0 (UM1749), STM32L1 (UM1816), STM32L4/L4+ (UM1884), STM32MP1 (https://wiki.st.com/stm32mpu) and STM32WB (UM2442).
# Contents

1  **STM32Cube overview** ................................................. 18

2  **Getting started with STM32CubeMX** .............................. 19
   2.1 Principles .......................................................... 19
   2.2 Key features ......................................................... 21
   2.3 Rules and limitations ............................................. 22

3  **Installing and running STM32CubeMX** .......................... 23
   3.1 System requirements ............................................... 23
      3.1.1 Supported operating systems and architectures ............ 23
      3.1.2 Memory prerequisites ....................................... 23
      3.1.3 Software requirements .................................... 23
   3.2 Installing/uninstalling STM32CubeMX standalone version .... 23
      3.2.1 Installing STM32CubeMX standalone version ............... 23
      3.2.2 Installing STM32CubeMX from command line ............... 25
      3.2.3 Uninstalling STM32CubeMX standalone version ............ 27
   3.3 Launching STM32CubeMX .......................................... 28
      3.3.1 Running STM32CubeMX as standalone application ........... 28
      3.3.2 Running STM32CubeMX in command-line mode ............... 28
   3.4 Getting updates using STM32CubeMX ............................ 30
      3.4.1 Updater configuration ....................................... 32
      3.4.2 Installing STM32 MCU packages ............................. 34
      3.4.3 Installing STM32 MCU package patches .................... 35
      3.4.4 Installing embedded software packs ....................... 35
      3.4.5 Removing already installed embedded software packages ... 40
      3.4.6 Checking for updates ....................................... 42

4  **STM32CubeMX user interface** .................................... 43
   4.1 Home page ......................................................... 44
      4.1.1 File menu .................................................... 45
      4.1.2 Window menu and Outputs tabs .............................. 46
      4.1.3 Help menu ................................................... 48
      4.1.4 Social links ................................................. 48
   4.2 New Project window ............................................... 49
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 MCU selector</td>
<td>50</td>
</tr>
<tr>
<td>4.2.2 Board selector</td>
<td>54</td>
</tr>
<tr>
<td>4.2.3 Cross selector</td>
<td>54</td>
</tr>
<tr>
<td>4.3 Project page</td>
<td>57</td>
</tr>
<tr>
<td>4.4 Pinout &amp; Configuration view</td>
<td>60</td>
</tr>
<tr>
<td>4.4.1 Component list</td>
<td>61</td>
</tr>
<tr>
<td>4.4.2 Component Mode panel</td>
<td>63</td>
</tr>
<tr>
<td>4.4.3 Pinout view</td>
<td>64</td>
</tr>
<tr>
<td>4.4.4 Pinout menu and shortcuts</td>
<td>65</td>
</tr>
<tr>
<td>4.4.5 Pinout view advanced actions</td>
<td>67</td>
</tr>
<tr>
<td>4.4.6 Keep Current Signals Placement</td>
<td>68</td>
</tr>
<tr>
<td>4.4.7 Pinning and labeling signals on pins</td>
<td>69</td>
</tr>
<tr>
<td>4.4.8 System view</td>
<td>70</td>
</tr>
<tr>
<td>4.4.9 Component Configuration panel</td>
<td>73</td>
</tr>
<tr>
<td>4.4.10 User Constants configuration window</td>
<td>75</td>
</tr>
<tr>
<td>4.4.11 GPIO Configuration window</td>
<td>80</td>
</tr>
<tr>
<td>4.4.12 DMA Configuration window</td>
<td>82</td>
</tr>
<tr>
<td>4.4.13 NVIC Configuration window</td>
<td>84</td>
</tr>
<tr>
<td>4.4.14 FreeRTOS configuration panel</td>
<td>91</td>
</tr>
<tr>
<td>4.4.15 Graphics frameworks and simulator</td>
<td>96</td>
</tr>
<tr>
<td>4.4.16 Setting HAL timebase source</td>
<td>98</td>
</tr>
<tr>
<td>4.5 Pinout &amp; Configuration view for STM32MP1 Series</td>
<td>102</td>
</tr>
<tr>
<td>4.5.1 Run time configuration</td>
<td>103</td>
</tr>
<tr>
<td>4.5.2 Boot stages configuration</td>
<td>104</td>
</tr>
<tr>
<td>4.6 Pinout &amp; Configuration view for STM32H7 dual-core product lines</td>
<td>105</td>
</tr>
<tr>
<td>4.7 Clock Configuration view</td>
<td>107</td>
</tr>
<tr>
<td>4.7.1 Clock tree configuration functions</td>
<td>108</td>
</tr>
<tr>
<td>4.7.2 Recommendations</td>
<td>112</td>
</tr>
<tr>
<td>4.7.3 STM32F43x/42x power-over drive feature</td>
<td>113</td>
</tr>
<tr>
<td>4.7.4 Clock tree glossary</td>
<td>114</td>
</tr>
<tr>
<td>4.8 Project Manager view</td>
<td>115</td>
</tr>
<tr>
<td>4.8.1 Project tab</td>
<td>116</td>
</tr>
<tr>
<td>4.8.2 Code Generator tab</td>
<td>121</td>
</tr>
<tr>
<td>4.8.3 Advanced Settings tab</td>
<td>125</td>
</tr>
<tr>
<td>4.9 Import Project window</td>
<td>127</td>
</tr>
<tr>
<td>4.10 Set unused / Reset used GPIOs windows</td>
<td>133</td>
</tr>
</tbody>
</table>
4.11 Update Manager windows ........................................... 135
4.12 Additional software component selection window ............... 136
  4.12.1 Introduction on software components ...................... 136
  4.12.2 Filter panel ............................................... 138
  4.12.3 Software component table ................................ 138
  4.12.4 Software component conditions .......................... 139
  4.12.5 Updating the tree view for additional software components 142
4.13 About window ................................................... 143

5 STM32CubeMX tools .................................................. 144
  5.1 Power Consumption Calculator view .............................. 144
    5.1.1 Building a power consumption sequence .................. 145
    5.1.2 Configuring a step in the power sequence ............... 150
    5.1.3 Managing user-defined power sequence and reviewing results 153
    5.1.4 Power sequence step parameters glossary ................ 156
    5.1.5 Battery glossary ........................................... 158
    5.1.6 SMPS feature ............................................. 158
    5.1.7 BLE support (STM32WB Series only) ....................... 164
    5.1.8 Example feature (STM32MP1 and STM32H7 dual-core only) 165
  5.2 DDR Suite (for STM32MP1 Series only) ........................ 167
    5.2.1 DDR configuration ........................................ 168
    5.2.2 Connection to the target and DDR register loading ..... 172
    5.2.3 DDR testing ............................................. 175
    5.2.4 DDR tuning ............................................... 177

6 STM32CubeMX C Code generation overview .......................... 181
  6.1 STM32Cube code generation using only HAL drivers (default mode) 181
  6.2 STM32Cube code generation using Low Layer drivers ............ 183
  6.3 Custom code generation ......................................... 189
    6.3.1 STM32CubeMX data model for FreeMarker user templates 189
    6.3.2 Saving and selecting user templates ........................ 190
    6.3.3 Custom code generation ................................. 190
  6.4 Additional settings for C project generation .................... 192

7 Code generation for dual-core MCUs (STM32H7 dual-core product lines only) .................... 196
8 Device tree generation (STM32MP1 Series only) .................. 198
  8.1 Device tree overview ............................................. 198
  8.2 STM32CubeMX Device tree generation ......................... 200
    8.2.1 Device tree generation for Linux kernel ................... 201
    8.2.2 Device tree generation for U-boot ............................. 202
    8.2.3 Device tree generation for TF-A ............................. 203

9 Support of additional software components using CMSIS-Pack standard ................................................. 204

10 Tutorial 1: From pinout to project C code generation using an MCU of the STM32F4 Series ....................... 207
  10.1 Creating a new STM32CubeMX Project ......................... 207
  10.2 Configuring the MCU pinout ..................................... 209
  10.3 Saving the project .............................................. 212
  10.4 Generating the report .......................................... 213
  10.5 Configuring the MCU clock tree ................................ 213
  10.6 Configuring the MCU initialization parameters .............. 216
    10.6.1 Initial conditions ........................................ 216
    10.6.2 Configuring the peripherals ................................ 217
    10.6.3 Configuring the GPIOs ....................................... 219
    10.6.4 Configuring the DMAs ........................................ 221
    10.6.5 Configuring the middleware ................................ 222
  10.7 Generating a complete C project ............................... 226
    10.7.1 Setting project options .................................... 226
    10.7.2 Downloading firmware package and generating the C code 227
  10.8 Building and updating the C code project .................... 232
  10.9 Switching to another MCU ...................................... 237

11 Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board ......................... 238

12 Tutorial 3 - Using the Power Consumption Calculator to optimize the embedded application consumption and more ... 246
  12.1 Tutorial overview ............................................. 246
  12.2 Application example description ................................ 247
12.3 Using the Power Consumption Calculator ............................ 247
12.3.1 Creating a power sequence ...................................... 247
12.3.2 Optimizing application power consumption ................. 249

13 Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board .............................. 256
13.1 Tutorial overview ..................................................... 256
13.2 Creating a new STM32CubeMX project and selecting the Nucleo board ............................................... 256
13.3 Selecting the features from the Pinout view ..................... 258
13.4 Configuring the MCU clock tree from the Clock Configuration view ...................................................... 260
13.5 Configuring the peripheral parameters from the Configuration view ...................................................... 261
13.6 Configuring the project settings and generating the project .......................................................... 264
13.7 Updating the project with the user application code ........... 265
13.8 Compiling and running the project .................................. 266
13.9 Configuring Tera Term software as serial communication client on the PC .............................................. 266

14 Tutorial 5: Exporting current project configuration to a compatible MCU ................................................. 268

15 Tutorial 6 – Adding embedded software packs to user projects .... 272

16 Tutorial 7 – Using the X-Cube-BLE1 software pack .............. 275

17 Tutorial 8 – Using STemWin Graphics framework ................ 287
17.1 Step 1: Selecting an MCU for Graphics ......................... 287
17.2 Step 2: Enabling STemWin from the pinout view ............ 287
17.3 Step 3: Configuring STemWin parameters from the configuration window ............................................ 288
17.4 Step 4: Using STemWin GUIBuilder tool from the configuration window ............................................ 289
17.5 Step 5: Generating the embedded C project and updates .......................................................... 290


19 Tutorial 10: Using ST-TouchGFX framework ...................... 294
UM1718 Contents

19.1 Step 1: Selecting an MCU for Graphics. ........................................... 294
19.2 Step 2: Enabling ST-TouchGFX from the pinout view ......................... 294
19.3 Step 3: Configuring ST-TouchGFX parameters from the configuration window ........................................... 295
19.4 Step 4: Generating the C project and using ST-TouchGFXDesigner ........ 297
19.5 Step 5: Generated project files ......................................................... 298

20 FAQ ........................................................................................................... 300
20.1 On the Pinout configuration panel, why does STM32CubeMX move some functions when I add a new peripheral mode? ........... 300
20.2 How can I manually force a function remapping? ................................. 300
20.3 Why are some pins highlighted in yellow or in light green in the Pinout view? Why cannot I change the function of some pins (when I click some pins, nothing happens)? ............... 300
20.4 Why do I get the error “Java 7 update 45” when installing “Java 7 update 45” or a more recent version of the JRE? ......................... 300
20.5 Why does the RTC multiplexer remain inactive on the Clock tree view? 301
20.6 How can I select LSE and HSE as clock source and change the frequency? ................................................................. 302
20.7 Why STM32CubeMX does not allow me to configure PC13, PC14, PC15 and PI8 as outputs when one of them is already configured as an output? ................................................................. 302
20.8 Ethernet Configuration: why cannot I specify DP83848 or LAN8742A in some cases? ......................................................... 303

Appendix A STM32CubeMX pin assignment rules ........................................ 304
A.1 Block consistency ................................................................................. 304
A.2 Block inter-dependency ....................................................................... 308
A.3 One block = one peripheral mode ........................................................ 310
A.4 Block remapping (STM32F10x only) .................................................... 310
A.5 Function remapping ........................................................................... 311
A.6 Block shifting (only for STM32F10x and when "Keep Current Signals placement" is unchecked) .................................................. 312
A.7 Setting and clearing a peripheral mode ................................................ 313
A.8 Mapping a function individually .......................................................... 313
A.9 GPIO signals mapping ....................................................................... 313
## Appendix B STM32CubeMX C code generation design choices and limitations

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 STM32CubeMX generated C code and user sections</td>
<td>314</td>
</tr>
<tr>
<td>B.2 STM32CubeMX design choices for peripheral initialization</td>
<td>314</td>
</tr>
<tr>
<td>B.3 STM32CubeMX design choices and limitations for middleware initialization</td>
<td>315</td>
</tr>
<tr>
<td>B.3.1 Overview</td>
<td>315</td>
</tr>
<tr>
<td>B.3.2 USB Host</td>
<td>316</td>
</tr>
<tr>
<td>B.3.3 USB Device</td>
<td>316</td>
</tr>
<tr>
<td>B.3.4 FatFs</td>
<td>316</td>
</tr>
<tr>
<td>B.3.5 FreeRTOS</td>
<td>317</td>
</tr>
<tr>
<td>B.3.6 LwiP</td>
<td>318</td>
</tr>
<tr>
<td>B.3.7 Libjpeg</td>
<td>320</td>
</tr>
<tr>
<td>B.3.8 Mbed TLS</td>
<td>321</td>
</tr>
<tr>
<td>B.3.9 TouchSensing</td>
<td>324</td>
</tr>
<tr>
<td>B.3.10 PDM2PCM</td>
<td>327</td>
</tr>
<tr>
<td>B.3.11 Graphics</td>
<td>327</td>
</tr>
<tr>
<td>B.3.12 STM32WPAN BLE/Thread (STM32WB Series only)</td>
<td>329</td>
</tr>
<tr>
<td>B.3.13 OpenAmp and RESMGR_UTILITY (STM32MP Series only)</td>
<td>333</td>
</tr>
</tbody>
</table>

## Appendix C STM32 microcontrollers naming conventions

## Appendix D STM32 microcontrollers power consumption parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.1 Power modes</td>
<td>338</td>
</tr>
<tr>
<td>D.1.1 STM32L1 Series</td>
<td>338</td>
</tr>
<tr>
<td>D.1.2 STM32F4 Series</td>
<td>339</td>
</tr>
<tr>
<td>D.1.3 STM32L0 Series</td>
<td>340</td>
</tr>
<tr>
<td>D.2 Power consumption ranges</td>
<td>341</td>
</tr>
<tr>
<td>D.2.1 STM32L1 Series features three VCORE ranges</td>
<td>341</td>
</tr>
<tr>
<td>D.2.2 STM32F4 Series features several VCORE scales</td>
<td>342</td>
</tr>
<tr>
<td>D.2.3 STM32L0 Series features three VCORE ranges</td>
<td>342</td>
</tr>
</tbody>
</table>

## Appendix E STM32Cube embedded software packages

## Revision history

8/361 UM1718 Rev 29
List of tables

Table 1. Command line summary ......................................................... 29
Table 2. Home page shortcuts ............................................................ 45
Table 3. Window menu ................................................................. 46
Table 4. Help menu shortcuts ............................................................ 48
Table 5. Component list, mode icons and color schemes .................... 62
Table 6. Pinout menu and shortcuts .................................................. 65
Table 7. Configuration states .......................................................... 71
Table 8. Peripheral and Middleware Configuration window buttons and tooltips .................................................. 74
Table 9. Clock configuration view widgets ........................................ 111
Table 10. Voltage scaling versus power over-drive and HCLK frequency 114
Table 11. Relations between power over-drive and HCLK frequency ........ 114
Table 12. Glossary ................................................................. 114
Table 13. Software components ..................................................... 138
Table 14. Software component conditions icons ............................... 139
Table 15. LL versus HAL code generation: drivers included in STM32CubeMX projects 184
Table 16. LL versus HAL code generation: STM32CubeMX generated header files .................................................. 184
Table 17. LL versus HAL: STM32CubeMX generated source files .............. 185
Table 18. LL versus HAL: STM32CubeMX generated functions and function calls .................................................. 185
Table 19. Connection with hardware resources .................................. 281
Table 20. Document revision history ................................................. 344
# List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Overview of STM32CubeMX C code generation flow.</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Example of STM32CubeMX installation in interactive mode</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3</td>
<td>STM32Cube Installation Wizard</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Auto-install command line.</td>
<td>27</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Displaying Windows default proxy settings.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Updater Settings window.</td>
<td>32</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Connection Parameters tab - Manual Configuration of Proxy Server.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Embedded Software Packages Manager window</td>
<td>34</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Managing embedded software packages - Help menu.</td>
<td>36</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Managing embedded software packages - Adding new url.</td>
<td>37</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Checking the validity of vendor pack.pdsc file url.</td>
<td>37</td>
</tr>
<tr>
<td>Figure 12</td>
<td>User-defined list of software packs.</td>
<td>38</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Selecting an embedded software pack release.</td>
<td>38</td>
</tr>
<tr>
<td>Figure 14</td>
<td>License agreement acceptance.</td>
<td>39</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Embedded software pack release - Successful installation.</td>
<td>40</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Removing libraries.</td>
<td>41</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Removing library confirmation message.</td>
<td>41</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Library deletion progress window.</td>
<td>41</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Help menu: checking for updates.</td>
<td>42</td>
</tr>
<tr>
<td>Figure 20</td>
<td>STM32CubeMX Home page.</td>
<td>44</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Window menu</td>
<td>47</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Output view.</td>
<td>47</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Link to social platforms.</td>
<td>48</td>
</tr>
<tr>
<td>Figure 24</td>
<td>New Project window shortcuts.</td>
<td>49</td>
</tr>
<tr>
<td>Figure 25</td>
<td>New Project window - MCU selector.</td>
<td>50</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Enabling graphics choice in MCU selector.</td>
<td>51</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Marking an MCU as favorite.</td>
<td>52</td>
</tr>
<tr>
<td>Figure 28</td>
<td>New Project window - MCU list with close function.</td>
<td>53</td>
</tr>
<tr>
<td>Figure 29</td>
<td>New Project window - List showing close MCUs.</td>
<td>53</td>
</tr>
<tr>
<td>Figure 30</td>
<td>New Project window - Board selector.</td>
<td>54</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Cross selector - Data refresh prerequisite.</td>
<td>55</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Cross selector - Part number selection per vendor.</td>
<td>55</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Cross selector - Partial part number selection completion.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Cross selector - Compare cart.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Cross selector - Part number selection for a new project.</td>
<td>57</td>
</tr>
<tr>
<td>Figure 36</td>
<td>STM32CubeMX Main window upon MCU selection.</td>
<td>58</td>
</tr>
<tr>
<td>Figure 37</td>
<td>STM32CubeMX Main window upon board selection (peripherals not initialized).</td>
<td>59</td>
</tr>
<tr>
<td>Figure 38</td>
<td>STM32CubeMX Main window upon board selection (peripherals initialized with default configuration).</td>
<td>60</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Contextual Help window (default).</td>
<td>61</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Contextual Help detailed information.</td>
<td>62</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Pinout view.</td>
<td>64</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Modifying pin assignments from the Pinout view.</td>
<td>67</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Example of remapping in case of block of pins consistency.</td>
<td>68</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Pins/Signals Options window.</td>
<td>70</td>
</tr>
<tr>
<td>Figure 45</td>
<td>System view.</td>
<td>71</td>
</tr>
<tr>
<td>Figure 46</td>
<td>Configuration window tabs (GPIO, DMA and NVIC settings for STM32F4 Series).</td>
<td>72</td>
</tr>
<tr>
<td>Figure 47</td>
<td>Peripheral Mode and Configuration view.</td>
<td>73</td>
</tr>
</tbody>
</table>
Figure 48. Formula when input parameter is set in No Check mode. 75
Figure 49. User Constants tab. 75
Figure 50. Extract of the generated main.h file. 76
Figure 51. Using constants for peripheral parameter settings. 76
Figure 52. Specifying user constant value and name. 77
Figure 53. Deleting an user constant is not allowed when the constant is already used for another constant definition. 78
Figure 54. Deleting an user constant used for parameter configuration - Confirmation request. 78
Figure 55. Deleting a user constant used for peripheral configuration - Consequence on peripheral configuration. 78
Figure 56. Searching for a name in a user constant list. 79
Figure 57. Searching for a value in a user constant list. 79
Figure 58. GPIO Configuration window - GPIO selection. 80
Figure 59. GPIO configuration grouped by peripheral. 81
Figure 60. Multiple Pins Configuration. 81
Figure 61. Adding a new DMA request. 82
Figure 62. DMA configuration. 83
Figure 63. DMA MemToMem configuration. 84
Figure 64. NVIC Configuration tab - FreeRTOS disabled. 85
Figure 65. NVIC Configuration tab - FreeRTOS enabled. 86
Figure 66. I2C NVIC Configuration window. 86
Figure 67. NVIC Code generation – All interrupts enabled. 88
Figure 68. NVIC Code generation – IRQ Handler generation. 90
Figure 69. FreeRTOS configuration view. 91
Figure 70. FreeRTOS: configuring tasks and queues. 92
Figure 71. FreeRTOS: creating a new task. 93
Figure 72. FreeRTOS - Configuring timers, mutexes and semaphores. 94
Figure 73. FreeRTOS Heap usage. 96
Figure 74. Enabling STemWin framework. 97
Figure 75. Saving changes. 98
Figure 76. Selecting a HAL timebase source (STM32F407 example). 99
Figure 77. TIM1 selected as HAL timebase source. 99
Figure 78. NVIC settings when using SysTick as HAL timebase, no FreeRTOS. 100
Figure 79. NVIC settings when using FreeRTOS and SysTick as HAL timebase. 101
Figure 80. NVIC settings when using FreeRTOS and TIM2 as HAL timebase. 102
Figure 81. STM32MP1 boot devices and runtime contexts. 103
Figure 82. STM32MP1 Series: assignment options for GPIOs. 103
Figure 83. Select peripherals as boot devices. 104
Figure 84. STM32H7 dual-core: peripheral and middleware context assignment. 105
Figure 85. STM32H7 dual-core: GPIOs context assignment. 106
Figure 86. STM32F469NIHx clock tree configuration view. 107
Figure 87. Clock tree configuration view with errors. 108
Figure 88. Clock tree configuration: enabling RTC, RCC clock source and outputs from Pinout view. 112
Figure 89. Clock tree configuration: RCC peripheral advanced parameters. 113
Figure 90. Project Settings window. 115
Figure 91. Project folder. 116
Figure 92. Selecting a basic application structure. 118
Figure 93. Selecting an advanced application structure. 119
Figure 94. OpenSTLinux settings (STM32MP1 Series only). 119
Figure 95. Selecting a different firmware location. 120
List of figures

Figure 96. Firmware location selection error message ............................................. 120
Figure 97. Recommended new firmware repository structure ..................................... 120
Figure 98. Project Settings code generator ............................................................. 123
Figure 99. Template Settings window ....................................................................... 124
Figure 100. Generated project template .................................................................... 125
Figure 101. Advanced Settings window .................................................................... 126
Figure 102. Generated init functions without C language “static” keyword ................. 126
Figure 103. Automatic project import ................................................................. 128
Figure 104. Manual project import ......................................................................... 129
Figure 105. Import Project menu - Try import with errors ........................................... 131
Figure 106. Import Project menu - Successful import after adjustments ....... 132
Figure 107. Set unused pins window ..................................................................... 133
Figure 108. Reset used pins window ..................................................................... 133
Figure 109. Set unused GPIO pins with Keep Current Signals Placement checked .... 134
Figure 110. Set unused GPIO pins with Keep Current Signals Placement unchecked .... 135
Figure 111. Additional software components - collapsed view showing packs and bundles . 137
Figure 112. Additional software components - expanded view showing component details . 137
Figure 113. Dependency solving: solution found ..................................................... 139
Figure 114. Dependency solving: no solution found ............................................... 140
Figure 115. Software components conditions - Solution proposals and guidance ...... 140
Figure 116. Software components conditions resolved ......................................... 141
Figure 117. Selection of additional software components ....................................... 142
Figure 118. Additional software components - Updated tree view .................... 142
Figure 119. About window .................................................................................... 143
Figure 120. Power Consumption Calculator default view ......................................... 145
Figure 121. Battery selection ............................................................................. 146
Figure 122. Step management functions ................................................................. 146
Figure 123. Power consumption sequence: New Step default view ....................... 147
Figure 124. Enabling the transition checker option on an already configured sequence - All transitions valid ................................................................. 148
Figure 125. Enabling the transition checker option on an already configured sequence - At least one transition invalid .......................................................... 148
Figure 126. Transition checker option - Show log ..................................................... 149
Figure 127. Interpolated power consumption .......................................................... 151
Figure 128. ADC selected in Pinout view ............................................................... 152
Figure 129. Power Consumption Calculator Step configuration window: ADC enabled using import pinout ................................................................. 153
Figure 130. Power Consumption Calculator view after sequence building ............. 154
Figure 131. Sequence table management functions .......................................... 154
Figure 132. Power Consumption: Peripherals consumption chart ................. 155
Figure 133. Description of the Results area .......................................................... 155
Figure 134. Overall peripheral consumption .......................................................... 157
Figure 135. Selecting SMPS for the current project ................................................. 159
Figure 136. SMPS database - Adding new SMPS models .................................... 160
Figure 137. SMPS database - Selecting a different SMPS model ...................... 160
Figure 138. Current project configuration updated with new SMPS model ............. 161
Figure 139. SMPS database management window with new model selected .......... 161
Figure 140. SMPS transition checker and state diagram helper window ............... 162
Figure 141. Configuring the SMPS mode for each step ...................................... 163
Figure 142. RF related consumption (STM32WB Series only) ............................. 164
Figure 143. RF BLE mode configuration (STM32WB Series only) ....................... 165
Figure 144. Power Consumption Calculator – Example set .................................. 166
<table>
<thead>
<tr>
<th>Figure number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>196</td>
<td>Simple pinout configuration</td>
<td>211</td>
</tr>
<tr>
<td>197</td>
<td>Save Project As window</td>
<td>212</td>
</tr>
<tr>
<td>198</td>
<td>Generate Project Report - New project creation</td>
<td>213</td>
</tr>
<tr>
<td>199</td>
<td>Generate Project Report - Project successfully created</td>
<td>213</td>
</tr>
<tr>
<td>200</td>
<td>Clock tree view</td>
<td>214</td>
</tr>
<tr>
<td>201</td>
<td>HSI clock enabled</td>
<td>215</td>
</tr>
<tr>
<td>202</td>
<td>HSE clock source disabled</td>
<td>215</td>
</tr>
<tr>
<td>203</td>
<td>HSE clock source enabled</td>
<td>215</td>
</tr>
<tr>
<td>204</td>
<td>External PLL clock source enabled</td>
<td>215</td>
</tr>
<tr>
<td>205</td>
<td>Pinout &amp; Configuration view</td>
<td>217</td>
</tr>
<tr>
<td>206</td>
<td>Case of Peripheral and Middleware without configuration parameters.</td>
<td>217</td>
</tr>
<tr>
<td>207</td>
<td>Timer 3 configuration window</td>
<td>218</td>
</tr>
<tr>
<td>208</td>
<td>Timer 3 configuration</td>
<td>218</td>
</tr>
<tr>
<td>209</td>
<td>Enabling Timer 3 interrupt</td>
<td>219</td>
</tr>
<tr>
<td>210</td>
<td>GPIO configuration color scheme and tooltip</td>
<td>219</td>
</tr>
<tr>
<td>211</td>
<td>GPIO mode configuration</td>
<td>220</td>
</tr>
<tr>
<td>212</td>
<td>DMA parameters configuration window</td>
<td>221</td>
</tr>
<tr>
<td>213</td>
<td>Middleware tooltip</td>
<td>222</td>
</tr>
<tr>
<td>214</td>
<td>USB Host configuration</td>
<td>222</td>
</tr>
<tr>
<td>215</td>
<td>FatFs over USB mode enabled</td>
<td>223</td>
</tr>
<tr>
<td>216</td>
<td>System view with FatFs and USB enabled</td>
<td>224</td>
</tr>
<tr>
<td>217</td>
<td>FatFs define statements</td>
<td>225</td>
</tr>
<tr>
<td>218</td>
<td>Project Settings and toolchain selection</td>
<td>226</td>
</tr>
<tr>
<td>219</td>
<td>Project Manager menu - Code Generator tab</td>
<td>227</td>
</tr>
<tr>
<td>220</td>
<td>Missing firmware package warning message</td>
<td>227</td>
</tr>
<tr>
<td>221</td>
<td>Error during download</td>
<td>228</td>
</tr>
<tr>
<td>222</td>
<td>Updater settings for download</td>
<td>228</td>
</tr>
<tr>
<td>223</td>
<td>Updater settings with connection</td>
<td>229</td>
</tr>
<tr>
<td>224</td>
<td>Downloading the firmware package</td>
<td>229</td>
</tr>
<tr>
<td>225</td>
<td>Unzipping the firmware package</td>
<td>230</td>
</tr>
<tr>
<td>226</td>
<td>C code generation completion message</td>
<td>230</td>
</tr>
<tr>
<td>227</td>
<td>C code generation output folder</td>
<td>231</td>
</tr>
<tr>
<td>228</td>
<td>C code generation output: Projects folder</td>
<td>232</td>
</tr>
<tr>
<td>229</td>
<td>C code generation for EWARM</td>
<td>233</td>
</tr>
<tr>
<td>230</td>
<td>STM32CubeMX generated project open in IAR™ IDE</td>
<td>234</td>
</tr>
<tr>
<td>231</td>
<td>IAR™ options</td>
<td>235</td>
</tr>
<tr>
<td>232</td>
<td>SWD connection</td>
<td>235</td>
</tr>
<tr>
<td>233</td>
<td>Project building log</td>
<td>235</td>
</tr>
<tr>
<td>234</td>
<td>User Section 2</td>
<td>236</td>
</tr>
<tr>
<td>235</td>
<td>User Section 4</td>
<td>236</td>
</tr>
<tr>
<td>236</td>
<td>Import Project menu</td>
<td>237</td>
</tr>
<tr>
<td>237</td>
<td>Board peripheral initialization dialog box</td>
<td>238</td>
</tr>
<tr>
<td>238</td>
<td>Board selection</td>
<td>239</td>
</tr>
<tr>
<td>239</td>
<td>SDIO peripheral configuration</td>
<td>239</td>
</tr>
<tr>
<td>240</td>
<td>FatFs mode configuration</td>
<td>240</td>
</tr>
<tr>
<td>241</td>
<td>RCC peripheral configuration</td>
<td>240</td>
</tr>
<tr>
<td>242</td>
<td>Clock tree view</td>
<td>240</td>
</tr>
<tr>
<td>243</td>
<td>FATFS tutorial - Project settings</td>
<td>241</td>
</tr>
<tr>
<td>244</td>
<td>C code generation completion message</td>
<td>241</td>
</tr>
<tr>
<td>245</td>
<td>IDE workspace</td>
<td>242</td>
</tr>
<tr>
<td>246</td>
<td>Power Consumption Calculation example</td>
<td>248</td>
</tr>
<tr>
<td>247</td>
<td>VDD and battery selection menu</td>
<td>248</td>
</tr>
</tbody>
</table>
Figure 248. Sequence table .................................................. 249
Figure 249. sequence results before optimization .............................................. 249
Figure 250. Step 1 optimization ................................................. 250
Figure 251. Step 5 optimization .................................................. 251
Figure 252. Step 6 optimization .................................................. 252
Figure 253. Step 7 optimization .................................................. 253
Figure 254. Step 8 optimization .................................................. 254
Figure 255. Step 10 optimization ............................................... 255
Figure 256. Power sequence results after optimizations ........................................ 255
Figure 257. Selecting NUCLEO_L053R8 board ............................................ 257
Figure 258. Selecting debug pins .................................................. 258
Figure 259. Selecting TIM2 clock source ............................................. 258
Figure 260. Selecting asynchronous mode for USART2 ....................................... 259
Figure 261. Checking pin assignment ................................................ 259
Figure 262. Configuring the MCU clock tree ............................................ 260
Figure 263. Configuring USART2 parameters ........................................... 261
Figure 264. Configuring TIM2 parameters ............................................. 262
Figure 265. Enabling TIM2 interrupt ................................................ 263
Figure 266. Project Settings menu ................................................. 264
Figure 267. Generating the code ................................................... 265
Figure 268. Checking the communication port ......................................... 266
Figure 269. Setting Tera Term port parameters ....................................... 267
Figure 270. Setting Tera Term port parameters ....................................... 267
Figure 271. Existing or new project pinout ............................................. 268
Figure 272. List of pinout compatible MCUs - Partial match with hardware compatibility .......................................................... 269
Figure 273. List of Pinout compatible MCUs - Exact and partial match .................... 269
Figure 274. Selecting a compatible MCU and importing the configuration ................. 270
Figure 275. Configuration imported to the selected compatible MCU .................... 270
Figure 276. Additional software components enabled for the current project .......... 272
Figure 277. Pack software components - no configurable parameters .................. 273
Figure 278. Pack tutorial - project settings ........................................... 273
Figure 279. Generated project with third party pack components .......................... 274
Figure 280. Hardware prerequisites .................................................. 275
Figure 281. Embedded software packages ............................................. 276
Figure 282. Mobile application ..................................................... 276
Figure 283. Installing Embedded software packages ...................................... 277
Figure 284. Starting a new project - selecting the NUCLEO-L053R8 board .......... 278
Figure 285. Starting a new project - initializing all peripherals ........................... 278
Figure 286. Selecting X-Cube-BLE1 components ....................................... 279
Figure 287. Configuring peripherals and GPIOs ......................................... 280
Figure 288. Configuring NVIC interrupts ............................................. 281
Figure 289. Enabling X-Cube-BLE1 .................................................. 282
Figure 290. Configuring the SensorDemo project ........................................... 283
Figure 291. Open SensorDemo project in the IDE toolchain .................................. 283
Figure 292. Launching the SensorDemo project in Atollie® TrueStudio® ................. 284
Figure 293. Viewing the SensorDemo project in Atollie® TrueStudio® ................ 284
Figure 294. Configuring the SensorDemo project in Atollie® TrueStudio® ............. 285
Figure 295. Testing the SensorDemo application ......................................... 286
Figure 296. Tutorial - Selecting an MCU for Graphics .................................... 287
Figure 297. Graphics frameworks tooltip ............................................. 287
Figure 298. Enabling STemWin framework ............................................. 288
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>299</td>
<td>STemWin Graphics framework configuration window</td>
<td>289</td>
</tr>
<tr>
<td>300</td>
<td>STemWin GUIBuilder configuration panel</td>
<td>290</td>
</tr>
<tr>
<td>301</td>
<td>STemWin GUIBuilder</td>
<td>290</td>
</tr>
<tr>
<td>302</td>
<td>STemWin generated project and files</td>
<td>291</td>
</tr>
<tr>
<td>303</td>
<td>GFXSIMULATOR in Configuration view</td>
<td>292</td>
</tr>
<tr>
<td>304</td>
<td>Graphics simulator user interface</td>
<td>293</td>
</tr>
<tr>
<td>305</td>
<td>Graphics simulator - Current Configuration fields</td>
<td>293</td>
</tr>
<tr>
<td>306</td>
<td>Enabling ST-TouchGFX</td>
<td>294</td>
</tr>
<tr>
<td>307</td>
<td>TouchGFXDesigner: installer location</td>
<td>295</td>
</tr>
<tr>
<td>308</td>
<td>TouchGFX incompatible LTDC Pixel format</td>
<td>295</td>
</tr>
<tr>
<td>309</td>
<td>Setting TouchGFX compatible LTDC pixel format</td>
<td>296</td>
</tr>
<tr>
<td>310</td>
<td>TouchGFXDesigner path missing</td>
<td>296</td>
</tr>
<tr>
<td>311</td>
<td>ST-TouchGFXDesigner executable file</td>
<td>297</td>
</tr>
<tr>
<td>312</td>
<td>Generate project before ST-TouchGFXDesigner launch</td>
<td>297</td>
</tr>
<tr>
<td>313</td>
<td>Launching ST-TOUCHGFXDesigner tool</td>
<td>297</td>
</tr>
<tr>
<td>314</td>
<td>ST-TOUCHGFXDesigner user interface</td>
<td>298</td>
</tr>
<tr>
<td>315</td>
<td>ST-TOUCHGFX generated project and files</td>
<td>299</td>
</tr>
<tr>
<td>316</td>
<td>Java™ Control Panel</td>
<td>301</td>
</tr>
<tr>
<td>317</td>
<td>Pinout view - Enabling the RTC</td>
<td>301</td>
</tr>
<tr>
<td>318</td>
<td>Pinout view - Enabling LSE and HSE clocks</td>
<td>302</td>
</tr>
<tr>
<td>319</td>
<td>Pinout view - Setting LSE/HSE clock frequency</td>
<td>302</td>
</tr>
<tr>
<td>320</td>
<td>Block mapping</td>
<td>305</td>
</tr>
<tr>
<td>321</td>
<td>Block remapping</td>
<td>306</td>
</tr>
<tr>
<td>322</td>
<td>Block remapping - Example 1</td>
<td>307</td>
</tr>
<tr>
<td>323</td>
<td>Block remapping - Example 2</td>
<td>307</td>
</tr>
<tr>
<td>324</td>
<td>Block inter-dependency - SPI signals assigned to PB3/4/5</td>
<td>308</td>
</tr>
<tr>
<td>325</td>
<td>Block inter-dependency - SPI1_MISO function assigned to PA6</td>
<td>309</td>
</tr>
<tr>
<td>326</td>
<td>One block = one peripheral mode - I2C1_SMBA function assigned to PB5</td>
<td>310</td>
</tr>
<tr>
<td>327</td>
<td>Block remapping - Example 2</td>
<td>311</td>
</tr>
<tr>
<td>328</td>
<td>Function remapping example</td>
<td>311</td>
</tr>
<tr>
<td>329</td>
<td>Block shifting not applied</td>
<td>312</td>
</tr>
<tr>
<td>330</td>
<td>Block shifting applied</td>
<td>313</td>
</tr>
<tr>
<td>331</td>
<td>FreeRTOS HOOK functions to be completed by user</td>
<td>317</td>
</tr>
<tr>
<td>332</td>
<td>LwIP 1.4.1 configuration</td>
<td>318</td>
</tr>
<tr>
<td>333</td>
<td>LwIP 1.5 configuration</td>
<td>319</td>
</tr>
<tr>
<td>334</td>
<td>Libjpeg configuration window</td>
<td>321</td>
</tr>
<tr>
<td>335</td>
<td>Mbed TLS without LwIP</td>
<td>322</td>
</tr>
<tr>
<td>336</td>
<td>Mbed TLS with LwIP and FreeRTOS</td>
<td>323</td>
</tr>
<tr>
<td>337</td>
<td>Mbed TLS configuration window</td>
<td>324</td>
</tr>
<tr>
<td>338</td>
<td>Enabling the TouchSensing peripheral</td>
<td>325</td>
</tr>
<tr>
<td>339</td>
<td>Touch-sensing sensor selection panel</td>
<td>326</td>
</tr>
<tr>
<td>340</td>
<td>TouchSensing configuration panel</td>
<td>327</td>
</tr>
<tr>
<td>341</td>
<td>Graphics application architecture</td>
<td>328</td>
</tr>
<tr>
<td>342</td>
<td>BLE and Thread middleware support in STM32CubeMX</td>
<td>329</td>
</tr>
<tr>
<td>343</td>
<td>STM32CubeWB Package download</td>
<td>330</td>
</tr>
<tr>
<td>344</td>
<td>STM32CubeWB BLE applications folder</td>
<td>331</td>
</tr>
<tr>
<td>345</td>
<td>BLE Server profile selection</td>
<td>332</td>
</tr>
<tr>
<td>346</td>
<td>BLE Client profile selection</td>
<td>332</td>
</tr>
<tr>
<td>347</td>
<td>Thread application selection</td>
<td>333</td>
</tr>
<tr>
<td>348</td>
<td>Enabling OpenAmp for STM32MP1 devices</td>
<td>334</td>
</tr>
<tr>
<td>349</td>
<td>Enabling the Resource Manager for STM32MP1 devices</td>
<td>334</td>
</tr>
<tr>
<td>350</td>
<td>Resource Manager: peripheral assignment view</td>
<td>335</td>
</tr>
</tbody>
</table>
Figure 351. STM32 microcontroller part numbering scheme .................. 337
Figure 352. STM32Cube Embedded Software package ............................ 343
1 STM32Cube overview

STM32Cube is an STMicroelectronics original initiative to make developers’ lives easier by reducing development effort, time and cost. STM32Cube covers the whole portfolio of STM32 microcontrollers, based on 32-bit Arm® Cortex® cores.

STM32Cube includes:

• STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.

• A comprehensive embedded software platform, delivered per Series (such as STM32CubeF2 for STM32F2 Series and STM32CubeF4 for STM32F4 Series)
  – The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio
  – Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  – A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  – All embedded software utilities, delivered with a full set of examples.

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
2 Getting started with STM32CubeMX

2.1 Principles

Customers need to quickly identify the MCU that best meets their requirements (core architecture, features, memory size, performance...). While board designers main concerns are to optimize the microcontroller pin configuration for their board layout and to fulfill the application requirements (choice of peripherals operating modes), embedded system developers are more interested in developing new applications for a specific target device, and migrating existing designs to different microcontrollers.

The time taken to migrate to new platforms and update the C code to new firmware drivers adds unnecessary delays to the project. STM32CubeMX was developed within STM32Cube initiative which purpose is to meet customer key requirements to maximize software reuse and minimize the time to create the target system:

- Software reuse and application design portability are achieved through STM32Cube firmware solution proposing a common Hardware Abstraction Layer API across STM32 portfolio.
- Optimized migration time is achieved thanks to STM32CubeMX built-in knowledge of STM32 microcontrollers, peripherals and middleware (LwIP and USB communication protocol stacks, FatFs file system for small embedded systems, FreeRTOS).

STM32CubeMX graphical interface performs the following functions:

- Fast and easy configuration of the MCU pins, clock tree and operating modes for the selected peripherals and middleware
- Generation of pin configuration report for board designers
- Generation of a complete project with all the necessary libraries and initialization C code to set up the device in the user defined operating mode. The project can be directly open in the selected application development environment (for a selection of supported IDEs) to proceed with application development (see Figure 1).

During the configuration process, STM32CubeMX detects conflicts and invalid settings and highlights them through meaningful icons and useful tool tips.
Figure 1. Overview of STM32CubeMX C code generation flow
2.2 Key features

STM32CubeMX comes with the following features:

- **Project management**
  STM32CubeMX allows creating, saving and loading previously saved projects:
  - When STM32CubeMX is launched, the user can choose to create a new project or to load a previously saved project.
  - Saving the project saves user settings and configuration performed within the project in an .ioc file that will be used the next time the project will be loaded in STM32CubeMX.
  STM32CubeMX also allows importing previously saved projects in new projects.
  STM32CubeMX projects come in two flavors:
  - MCU configuration only: .ioc file is saved in a dedicated project folder.
  - MCU configuration with C code generation: in this case .ioc files are saved in a dedicated project folder along with the generated source C code. There can be only one .ioc file per project.

- **Easy MCU and STMicroelectronics board selection**
  When starting a new project, a dedicated window opens to select either a microcontroller or an STMicroelectronics board from STM32 portfolio. Different filtering options are available to ease the MCU and board selection.

- **Easy pinout configuration**
  - From the Pinout view, the user can select the peripherals from a list and configure the peripheral modes required for the application. STM32CubeMX assigns and configures the pins accordingly.
  - For more advanced users, it is also possible to directly map a peripheral function to a physical pin using the Pinout view. The signals can be locked on pins to prevent STM32CubeMX conflict solver from moving the signal to another pin.
  - Pinout configuration can be exported as a .csv file.

- **Complete project generation**
  The project generation includes pinout, firmware and middleware initialization C code for a set of IDEs. It is based on STM32Cube embedded software libraries. The following actions can be performed:
  - Starting from the previously defined pinout, the user can proceed with the configuration of middleware, clock tree, services (RNG, CRC, etc...) and peripheral parameters. STM32CubeMX generates the corresponding initialization C code. The result is a project directory including generated main.c file and C header files for configuration and initialization, plus a copy of the necessary HAL and middleware libraries as well as specific files for the selected IDE.
  - The user can modify the generated source files by adding user-defined C code in user dedicated sections. STM32CubeMX ensures that the user C code is preserved upon next C code generation (the user C code is commented if it is no longer relevant for the current configuration).
  - STM32CubeMX can generate user files by using user-defined freemarker .ftl template files.
  - From the Project settings menu, the user can select the development toolchain (IDE) for which the C code has to be generated. STM32CubeMX ensures that the IDE relevant project files are added to the project folder so that the project can be
directly imported as a new project within third party IDE (IAR™ EWARM, Keil™ MDK-ARM, Atollic® TrueSTUDIO® and AC6 System Workbench for STM32).

- **Power consumption calculation**
  Starting with the selection of a microcontroller part number and a battery type, the user can define a sequence of steps representing the application life cycle and parameters (choice of frequencies, enabled peripherals, step duration). STM32CubeMX Power Consumption Calculator returns the corresponding power consumption and battery life estimates.

- **Clock tree configuration**
  STM32CubeMX offers a graphic representation of the clock tree as it can be found in the device reference manual. The user can change the default settings (clock sources, prescaler and frequency values). The clock tree is then updated accordingly. Invalid settings and limitations are highlighted and documented with tool tips. Clock tree configuration conflicts can be solved by using the solver feature. When no exact match is found for a given user configuration, STM32CubeMX proposes the closest solution.

- **Automatic updates of STM32CubeMX and STM32Cube MCU packages**
  STM32CubeMX comes with an updater mechanism that can be configured for automatic or on-demand check for updates. It supports STM32CubeMX self-updates as well as STM32Cube firmware library package updates. The updater mechanism also allows deleting previously installed packages.

- **Report generation**
  .pdf and .csv reports can be generated to document user configuration work.

- **Graphics simulator**
  For graphics-capable microcontrollers, STM32CubeMX allows the user to simulate a graphics configuration and adjust graphics parameters to optimize the performance. Once the results are satisfactory, the current project configuration can be adjusted accordingly.

- **Support of embedded software packages in CMSIS-Pack format**
  STM32CubeMX allows getting and downloading updates of embedded software packages delivered in CMSIS-Pack format. Selected software components belonging to these new releases can then be added to the current project.

- **Contextual help**
  Contextual help windows can be displayed by hovering the mouse over Cores, Series, Peripherals And Middleware. They provide a short description and links to the relevant documentation corresponding to the selected item.

### 2.3 Rules and limitations

- **C code generation covers only peripheral and middleware initialization. It is based on STM32Cube HAL firmware libraries.**

- **STM32CubeMX C code generation covers only initialization code for peripherals and middleware components that use the drivers included in STM32Cube embedded software packages. The code generation of some peripherals and middleware components is not yet supported.**

- **Refer to Appendix A for a description of pin assignment rules.**

- **Refer to Appendix B for a description of STM32CubeMX C code generation design choices and limitations.**
3 Installing and running STM32CubeMX

3.1 System requirements

3.1.1 Supported operating systems and architectures
- Windows® 7: 32-bit (x86), 64-bit (x64)
- Windows® 8: 32-bit (x86), 64-bit (x64)
- Windows® 10: 32-bit (x86), 64-bit (x64)
- Linux®: 32-bit (x86) and 64-bit (x64) (tested on RedHat, Ubuntu and Fedora)
  Since STM32CubeMX is a 32-bit application, some versions of Linux 64-bit distributions require to install 32-bit compliant packages such as ia32-libs.
- macOS®: 64-bit (x64) (tested on OS X El Capitan and Sierra)

3.1.2 Memory prerequisites
- Recommended minimum RAM: 2 Gbytes.

3.1.3 Software requirements
The Java™ Run Time Environment 1.8 must be installed.
Note that Java 9 and Java 10 are not supported and there is limited validation done with Java 11.
After Oracle announcement related to ‘End of Public Updates for Oracle JDK 8’, you can access OpenJDK 8 via https://adoptopenjdk.net/.

3.2 Installing/uninstalling STM32CubeMX standalone version

3.2.1 Installing STM32CubeMX standalone version
To install STM32CubeMX, follow the steps below:
2. Extract (unzip) stm32cubemx.zip whole package into the same directory.
3. Check your access rights and launch the installation wizard:
   On Windows®:
   a) Make sure you have administrators rights.
   b) Double-click the SetupSTM32CubeMX-VERSION.exe file to launch the installation wizard.
   On Linux®:
   a) Make sure you have access rights to the target installation directory. You can run the installation as root (or sudo) to install STM32CubeMX in shared directories.
   b) Do “chmod 777 SetupSTM32CubeMX-5.0.0.0.linux” to change the properties, so that the file is executable.
c) Double-click on the SetupSTM32CubeMX-VERSION.linux file, or launch it from
the console window.

On macOS®:
  a) Make sure you have administrators rights.
  b) Double-click SetupSTM32CubeMX-VERSION application file to launch the
      installation wizard.
      In case of error, launch the exe file with the following command:

4. Upon successful installation of STM32CubeMX on Windows, STM32CubeMX icon is
displayed on your desktop and STM32CubeMX application is available from the
Program menu. STM32CubeMX .ioc files are displayed with a cube icon. Double-click
them to open up them using STM32CubeMX.

5. Delete the content of the zip from your disk.

Note: If the proper version of the Java™ Runtime Environment (version 1.7_45 or newer) is not
installed, the wizard will propose to download it and stop. Restart STM32CubeMX
installation once Java™ installation is complete. Refer to Section 20: FAQ for issues when
installing the JRE.

When working on Windows, only the latest installation of STM32CubeMX will be enabled in
the Program menu. Previous versions can be kept on your PC (not recommended) when
different installation folders have been specified. Otherwise, the new installation overwrites
the previous ones.
3.2.2 Installing STM32CubeMX from command line

There are two ways to launch an installation from a console window: either in console interactive mode or via a script.

Interactive mode

To perform interactive installation, type the following command:

```
java -jar SetupSTM32CubeMX-4.14.0.exe -console
```

At each installation step, an answer is requested (see Figure 2).

Figure 2. Example of STM32CubeMX installation in interactive mode
Auto-install mode

At end of an installation, performed either using STM32CubeMX graphical wizard or console mode, it is possible to generate an auto-installation script containing user installation preferences (see Figure 3):

Figure 3. STM32Cube Installation Wizard

You can then launch the installation by typing the following command:

```
java -jar SetupSTM32CubeMX-4.14.0.exe auto-install.xml
```
3.2.3 Uninstalling STM32CubeMX standalone version

Uninstalling STM32CubeMX on macOS®

To uninstall STM32CubeMX on macOS use the following command line:

```
java -jar <STM32CubeMX installation path>/Uninstaller/uninstaller.jar.
```

Uninstalling STM32CubeMX on Linux®

There are three different ways to uninstall STM32CubeMX on Linux:

- By using the following command line
  
  ```
  java -jar <STM32CubeMX installation path>/Uninstaller/uninstaller.jar.
  ```

- Through a Windows Explorer window:
  a) Use a file explorer.
  b) Go to the Uninstaller directory of STM32CubeMX installation.
  c) Double-click the start uninstall desktop shortcut.

Uninstalling STM32CubeMX on Windows®

There are three different ways to uninstall STM32CubeMX on Windows:

- By using the following command line
  
  ```
  java -jar <STM32CubeMX installation path>/Uninstaller/uninstaller.jar.
  ```

- Through a Windows Explorer window:
  a) Use a file explorer.
  b) Go to the Uninstaller directory of STM32CubeMX installation.
  c) Double-click the start uninstall desktop shortcut.

- Through the Windows Control Panel:
  a) Select Programs and Features from the Windows Control Panel to display the list of programs installed on your computer.
  b) Right-click STM32CubeMX and select uninstall.
3.3 Launching STM32CubeMX

3.3.1 Running STM32CubeMX as standalone application

To run STM32CubeMX as a standalone application on Windows:
- select STM32CubeMX from Program Files > ST Microelectronics > STM32CubeMX.
- or double-click STM32CubeMX icon on your desktop.

To run STM32CubeMX as a standalone application on Linux, launch the STM32CubeMX executable from STM32CubeMX installation directory.

To run STM32CubeMX as a standalone application on macOS®, launch the STM32CubeMX application from the launchpad.

*Note:* There is no STM32CubeMX desktop icon on macOS®.

3.3.2 Running STM32CubeMX in command-line mode

To facilitate its integration with other tools, STM32CubeMX provides a command-line mode. Using a set of commands, you can:
- load an MCU
- load an existing configuration
- save a current configuration
- set project parameters and generate corresponding code
- generate user code from templates.

Three command-line modes are available:
- To run STM32CubeMX in interactive command-line mode, use the following command line:
  - On Windows:
    java -jar STM32CubeMX.exe -i
  - On Linux® and macOS®:
    java -jar STM32CubeMX -i
  The “MX>” prompt is then displayed to indicate that the application is ready to accept commands.
- To run STM32CubeMX in command-line mode getting commands from a script, use the following command line:
  - On Windows:
    java -jar STM32CubeMX.exe -s <script filename>
  - On Linux and macOS:
    java -jar STM32CubeMX -s <script filename>
  All the commands to be executed must be listed in the script file. An example of script file content is shown below:
  ```
  load STM32F417VETx
  project name MyFirstMXGeneratedProject
  project toolchain "MDK-ARM v4"
  project path C:\STM32CubeProjects\STM32F417VETx
  project generate
  ```
exit

- To run STM32CubeMX in command-line mode getting commands from a script and without UI, use the following command line:
  - On Windows:
    ```bash
    java -jar STM32CubeMX.exe -q <script filename>
    ```
  - On Linux and macOS:
    ```bash
    java -jar STM32CubeMX -q <script filename>
    ```

Here again, the user can enter commands when the MX prompt is displayed.

Table 1 for available commands.

<table>
<thead>
<tr>
<th>Command line</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td>Displays the list of available commands.</td>
<td>help</td>
</tr>
<tr>
<td>load &lt;mcu&gt;</td>
<td>Loads the selected MCU.</td>
<td>load STM32F101RCTx&lt;br&gt;load STM32F101Z(F-G)Tx</td>
</tr>
<tr>
<td>config load &lt;filename&gt;</td>
<td>Loads a previously saved configuration.</td>
<td>config load C:\Cube\ccmram\ccmram.ioc</td>
</tr>
<tr>
<td>config save &lt;filename&gt;</td>
<td>Saves the current configuration.</td>
<td>config save C:\Cube\ccmram\ccmram.ioc</td>
</tr>
<tr>
<td>config saveext &lt;filename&gt;</td>
<td>Saves the current configuration with all parameters, including those for which values have been kept to default (unchanged by the user).</td>
<td>config saveext C:\Cube\ccmram\ccmram.ioc</td>
</tr>
<tr>
<td>config saveas &lt;filename&gt;</td>
<td>Saves the current project under a new name.</td>
<td>config saveas C:\Cube\ccmram2\ccmram2.ioc</td>
</tr>
<tr>
<td>csv pinout &lt;filename&gt;</td>
<td>Exports the current pin configuration as a csv file. This file can be (later) imported into a board layout tool.</td>
<td>Csv pinout mypinout.csv</td>
</tr>
<tr>
<td>script &lt;filename&gt;</td>
<td>Runs all commands in the script file. There must be one command per line.</td>
<td>script myscript.txt</td>
</tr>
<tr>
<td>project couplefilesbyip &lt;0</td>
<td>1&gt;</td>
<td>This code generation option allows the user to choose between 0 (to generating the peripheral initializations in the main) or 1 (to generate each peripheral initialization in dedicated .c/.h files).</td>
</tr>
<tr>
<td>setDriver &lt;Peripheral Name&gt; &lt;HAL</td>
<td>LL&gt;</td>
<td>For supported Series, STM32CubeMX can generate peripheral initialization code based on LL drivers or on HAL drivers. This command line allows the user to choose, for each peripheral, between HAL-based and LL-based code generation. By default code generation is based on HAL drivers.</td>
</tr>
</tbody>
</table>
3.4 Getting updates using STM32CubeMX

STM32CubeMX implements a mechanism to access the Internet and to:

- download embedded software packages: STM32Cube MCU packages (full releases and patches) and third-party packages (.pack) based on the Arm® CMIS pack format
- manage a user-defined list of third-party packs
- check for STM32CubeMX and embedded software packages updates
- perform self-updates of STM32CubeMX
- refresh STM32 MCUs descriptions and documentation offer.

Installation and update related submenus are available under the Help menu.

Off-line updates can also be performed on computers without Internet access (see Section 3.4.2). This is done by browsing the filesystem and selecting available STM32Cube MCU packages.
If the PC on which STM32CubeMX runs is connected to a computer network using a proxy server, STM32CubeMX needs to connect to that server to access the Internet, get self-updates and download firmware packages. Refer to Section 3.4.1 for a description of this connection configuration.

To view Windows default proxy settings, select Internet options from the Control panel and select LAN settings from the Connections tab (see Figure 5).

**Figure 5. Displaying Windows default proxy settings**

Several proxy types exist and different computer network configurations are possible:
- Without proxy: the application directly accesses the web (Windows default configuration).
- Proxy without login/password
- Proxy with login/password: when using an Internet browser, a dialog box opens and prompts the user to enter its login/password.
- Web proxies with login/password: when using an Internet browser, a web page opens and prompts the user to enter its login/password.

If needed, contact your IT administrator for proxy information (proxy type, http address, port).

STM32CubeMX does not support web proxies. In this case, the user will not be able to benefit from the update mechanism and will need to manually copy the STM32Cube MCU packages from http://www.st.com/stm32cube to the repository. To do it, follow the sequence below:
1. Go to http://www.st.com/stm32cube and download the relevant STM32Cube MCU package from the Associated Software section.

2. Unzip the zip package to your STM32Cube repository. Find out the default repository folder location in the Updater settings tab as shown in Figure 6 (you might need to update it to use a different location or name).

### 3.4.1 Updater configuration

To perform STM32Cube new library package installation or updates, the tool must be configured as follows:

1. Select Help > Updater Settings to open the Updater Settings window.
2. From the Updater Settings tab (see Figure 6)
   a) Specify the repository destination folder where the downloaded packages will be stored.
   b) Enable/Disable the automatic check for updates.

**Figure 6. Updater Settings window**

3. In the Connection Parameters tab, specify the proxy server settings appropriate for your network configuration by selecting a proxy type among the following possibilities (see Figure 7):
   - No Proxy
   - Use System Proxy Parameters
     
     On Windows, proxy parameters will be retrieved from the PC system settings.
Uncheck “Require Authentication” if a proxy server without login/password configuration is used.

– Manual Configuration of Proxy Server

Enter the Proxy server http address and port number. Enter login/password information or uncheck “Require Authentication” if a proxy server without login/password configuration is used.

4. Optionally uncheck Remember my credentials to prevent STM32CubeMX to save encrypted login/password information in a file. This implies reentering login/password information each time STM32CubeMX is launched.

5. Click the Check Connection button to verify if the connection works. A green check mark appears to confirm that the connection operates correctly.

6. Select Help > Install New Libraries submenu to select among a list of possible packages to install.

7. If the tool is configured for manual checks, select Help > Check for Updates to find out about new tool versions or firmware library patches available to install.
### 3.4.2 Installing STM32 MCU packages

To download new STM32 MCU packages, follow the steps below:

1. Select **Help > Manage embedded software packages** to open the **Embedded Software Packages Manager** (see **Figure 8**), or use Install/Remove button from the Home page.

   - **Expand/collapse buttons** expands/collapses the list of packages, respectively.

   If the installation was performed using STM32CubeMX, all the packages available for download are displayed along with their version including the version currently installed on the user PC (if any), and the latest version available from [www.st.com](http://www.st.com).

   If no Internet access is available at that time, choose “From Local ...”, then browse to select the zip file of the desired STM32Cube MCU package that has been previously downloaded. An integrity check is performed on the file to ensure that it is fully supported by STM32CubeMX.

   The package is marked in green when the version installed matches the latest version available from [www.st.com](http://www.st.com).

2. Click the checkbox to select a package then “Install Now” to start the download.

   See **Figure 8** for an example.

---

**Figure 8. Embedded Software Packages Manager window**

---
3.4.3 Installing STM32 MCU package patches

Use the procedure described in Section 3.4.2 to download STM32 MCU package patches. A library patch, such as STM32Cube_FW_F7_1.4.1, can be easily identified by its version number which third digit is non-null (e.g. ‘1’ for the 1.4.1 version).

The patch is not a complete library package but only the set of library files that need to be updated. The patched files go on top of the original package (e.g. STM32Cube_FW_F7_1.4.1 complements STM32Cube_FW_F7_1.4.0 package).

Prior to 4.17 version, STM32CubeMX copies the patches within the original baseline directory (e.g. STM32Cube_FW_F7_V1.4.1 patched files are copied within the directory called STM32Cube_FW_F7_V1.4.0).

Starting with STM32CubeMX 4.17, downloading a patch leads to the creation of a dedicated directory. As an example, downloading STM32Cube_FW_F7_V1.4.1 patch creates the STM32Cube_FW_F7_V1.4.1 directory that contains the original STM32Cube_FW_F7_V1.4.0 baseline plus the patched files contained in STM32Cube_FW_F7_V1.4.1 package.

Users can then choose to go on using the original package (without patches) for some projects and upgrade to a patched version for others projects.

3.4.4 Installing embedded software packs

Starting from the release 4.24, STM32CubeMX offers the possibility to select third-party embedded software packages coming in the Arm® Keil™ CMSIS-Pack format (.pack), which contents are described thanks to the pack description (.pdsc) file. Reference documentation is available from http://www.keil.com.

1. Select Help > Manage embedded software packages to open the New Libraries Manager window (see Figure 9), or use Install/Remove button from the Home page.

Use Expand/collapse buttons to expand/collapse the list of packages, respectively.
2. Click **From Local** … button to browse the computer filesystem and select an embedded software package. STM32Cube MCU packages come as zip archives and embedded software packs come as .pack archives. This action is required in the following cases:
   - No Internet access is possible but the embedded software package is available locally on the computer.
   - The embedded software package is not public and hence not available on Internet. For such packages, STM32CubeMX cannot detect and propose updates.
3. Click **From URL**… button to specify the download location from Internet for either one of the pack .pdsc file or the vendor pack index (.pidx). Proceed as follow:
   a) Choose **From URL** … and click **New** (see **Figure 10**).
   b) Specify the .pdsc file url. As an example, the url of Oryx-Embedded middleware pack is https://www.oryx-embedded.com/download/pack/Oryx-Embedded.Middleware.pdsc (see **Figure 11**).
c) Click the **Check** button to verify that the url provided is valid (see **Figure 11**).

**Figure 11. Checking the validity of vendor pack.pdsc file url**
d) Click OK. The pack pdsc information is now available in the user defined pack list (see Figure 12).

To delete a url from the list, select the url checkbox and click Remove.

Figure 12. User-defined list of software packs

![User Defined Packs Manager](image)

e) Click OK to close the window and start retrieving pdsc information. Upon successful completion, the available pack versions are shown in the list of libraries that can be installed. Use the corresponding checkbox to select a given release.

Figure 13. Selecting an embedded software pack release

![Embedded Software Packages Manager](image)
f) Click **Install Now** to start downloading the software pack. A progress bar opens to indicate the installation progress. If the pack comes with a license agreement, a window pops up to ask for user's acceptance (see Figure 14). When the installation is successful, the check box turns green (see Figure 15). The user will then be able to add software components from this pack to its projects.

**Figure 14. License agreement acceptance**
3.4.5 Removing already installed embedded software packages

Proceed as follows (see figures 16 to 18) to clean up the repository from old library versions, thus saving disk space:

1. Select Help > Manage embedded software packages to open the Embedded Software Packages Manager, or use Install/Remove button from the Home page.
2. Click a green checkbox to select a package available in stm32cube repository.
3. Click the Remove Now button and confirm. A progress window then opens to show the deletion status.

Figure 15. Embedded software pack release - Successful installation
Figure 16. Removing libraries

![Embedded Software Packages Manager](image1)

STM32CubeF4 Firmware Package V1.19.0 / 29-December-2017

**Main Changes**

- HAL CAN driver has been redesigned with new API's.
- Support latest mbedTLS, LwIP and Fatf's stacks.

Figure 17. Removing library confirmation message

![Removing library confirmation message](image2)

Figure 18. Library deletion progress window

![Library deletion progress window](image3)
3.4.6 Checking for updates

STM32CubeMX can check if updates are available for STM32CubeMX currently installed version or for the embedded software packages installed in the repository folder (Figure 19).

When the updater is configured for automatic checks, it regularly verifies if updates are available.

When automatic checks have been disabled in the updater settings window, the user can manually check if updates are available:

1. Click the icon to open the Update Manager window or Select Help > Check for Updates. All the updates available for the user current installation are listed.
2. Click the check box to select a package, and then Install Now to download the update.

**Figure 19. Help menu: checking for updates**
4 STM32CubeMX user interface

STM32CubeMX user interface comes with three main views the user can navigate through using convenient breadcrumbs:

1. the Home page
2. the New project window
3. the project page

They come with panels, buttons and menus allowing users to take actions and make configuration choices with a single click.

The user interface is detailed in the following sections.

For C code generation, although the user can switch back and forth between the different configuration views, it is recommended to follow the sequence below:

1. From the Project Manager view, configure the project settings.
2. From the Mode panel in the Pinout & Configuration view, configure the RCC peripheral by enabling the external clocks, master output clocks, audio input clocks (when relevant for your application). This automatically displays more options on the Clock configuration view (see Figure 86). Then, select the features (peripherals, middlewares) and their operating modes relevant to the application.
3. If necessary, adjust the clock tree configuration from the clock configuration view.
4. From the Configuration panel in the Pinout & Configuration view configure the parameters required to initialize the peripherals and middleware operating modes.
5. Generate the initialization C code by clicking GENERATE CODE.
4.1 Home page

The Home page is the first window that opens up when launching STM32CubeMX program (see Figure 20). Closing it closes down the application. It offers shortcuts for some top level menus and access to social networks sites. Top-level menus and social network links remain accessible from the subsequent project page and are detailed in the following sections.

Figure 20. STM32CubeMX Home page
4.1.1 File menu

Refer to Table 2 for a description of the File menu and shortcuts.

<table>
<thead>
<tr>
<th>Name</th>
<th>Keyboard shortcut</th>
<th>Description</th>
<th>Home page shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Project...</td>
<td>Ctrl-N</td>
<td>Opens a new project window showing all supported MCUs and a set of STMicroelectronics boards to choose from(1).</td>
<td>To create a new project starting from a board click ACCESS TO BOARD SELECTOR. To create a new project starting from an MCU click ACCESS TO MCU SELECTOR.</td>
</tr>
<tr>
<td>Load Project...</td>
<td>Ctrl-L</td>
<td>Loads an existing STM32CubeMX project configuration by selecting an STM32CubeMX configuration .ioc file (see Caution:).</td>
<td>Under Other project, click browse icon</td>
</tr>
<tr>
<td>Import Project...</td>
<td>Ctrl-I</td>
<td>Opens a new window to select the configuration file to be imported as well as the import settings. The import is possible only if you start from an empty MCU configuration. Otherwise, the menu is disabled(2).</td>
<td>None</td>
</tr>
<tr>
<td>Save Project</td>
<td>Ctrl-S</td>
<td>Saves current project configuration (pinout, clock tree, peripherals, middlewares, Power Consumption Calculator) as a new project. This action creates a project folder including an .ioc file, according to user defined project settings.</td>
<td>None</td>
</tr>
<tr>
<td>Save Project as…</td>
<td>Ctrl-A</td>
<td>Saves the current project.</td>
<td>None</td>
</tr>
<tr>
<td>Close Project</td>
<td>Ctrl-C</td>
<td>Closes the current project and switches back to the welcome page.</td>
<td>None</td>
</tr>
<tr>
<td>Recent Projects</td>
<td>none</td>
<td>Displays the list of the five most recently saved projects.</td>
<td>Under Recent Project, click icon next to the project name.</td>
</tr>
<tr>
<td>Generate Report</td>
<td>Ctrl-R</td>
<td>Saves the project current configuration as two documents (pdf and text formats).</td>
<td>None</td>
</tr>
<tr>
<td>Exit</td>
<td>Ctrl-X</td>
<td>Proposes to save the project (if needed), then closes the application.</td>
<td>To close the window and the application click on .</td>
</tr>
</tbody>
</table>

1. On New project: to avoid any popup error messages at this stage, make sure an Internet connection is available (Connection Parameters tab under Help > Updater settings menu) or that Data Auto-refresh settings are set to No Auto-Refresh at application start (Updater Settings tab under Help > Updater Settings menu).

2. On Import, a status window displays the warnings or errors detected when checking for import conflicts. The user can then decide to cancel the import.
Caution: On project load: STM32CubeMX detects if the project was created with an older version of the tool and if this is the case, it proposes the user to either migrate to use the latest STM32CubeMX database and STM32Cube firmware version, or to continue.
Prior to STM32CubeMX 4.17, clicking Continue still upgrades to the latest database “compatible” with the STM32Cube firmware version used by the project.
Starting from STM32CubeMX 4.17, clicking Continue keeps the database used to create the project untouched. If the required database version is not available on the computer, it will be automatically downloaded.
When upgrading to a new version of STM32CubeMX, make sure to always backup your projects before loading the new project (especially when the project includes user code).

4.1.2 Window menu and Outputs tabs

The Window menu allows the user to access the Outputs function.

Table 3. Window menu

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td>Selecting/deselecting Outputs from the Window menu hides/shows the following Outputs tabs at the bottom of STM32CubeMX project page (see Figure 21)</td>
</tr>
<tr>
<td></td>
<td>– MCUs selection tab that lists the MCUs of a given family matching the user criteria (Series, peripherals, package,...) when an MCU was selected last(1);</td>
</tr>
<tr>
<td></td>
<td>– Outputs tab that displays a non-exhaustive list of the actions performed, raised errors and warnings (see Figure 22) found upon user actions.</td>
</tr>
<tr>
<td>Font size</td>
<td>Makes possible to change STM32CubeMX font size settings. STM32CubeMX must be re-launched for changes to take effect.</td>
</tr>
</tbody>
</table>

1. Selecting a different MCU from the list resets the current project configuration and switches to the new MCU. The user will be prompted to confirm this action before proceeding.
Figure 21. Window menu

Figure 22. Output view
4.1.3 Help menu

Refer to Table 4 for a description of the Help menu and shortcuts.

Table 4. Help menu shortcuts

<table>
<thead>
<tr>
<th>Name</th>
<th>Keyboard shortcut</th>
<th>Description</th>
<th>Home page shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help</td>
<td>F1</td>
<td>Opens the STM32CubeMX user manual.</td>
<td>None</td>
</tr>
<tr>
<td>About</td>
<td>Alt-A</td>
<td>Shows version information.</td>
<td>None</td>
</tr>
<tr>
<td>Docs &amp; Resources</td>
<td>Alt-D</td>
<td>Displays the official documentation available for the MCU used in the current project.</td>
<td>None</td>
</tr>
<tr>
<td>Refresh Data</td>
<td>Alt-R</td>
<td>Opens a dialog window that proposes to refresh STM32CubeMX database with STM32 MCU latest information (description and list of official documents), and allows the user to download all official documentation in one shot.</td>
<td>None</td>
</tr>
<tr>
<td>Check for Updates</td>
<td>Alt-C</td>
<td>Shows the software and firmware release updates available for download.</td>
<td>Click</td>
</tr>
<tr>
<td>Manage embedded software packages</td>
<td>Alt-U</td>
<td>Shows all the embedded software packages available for installation. A green check box indicates that the package is already installed in the user repository folder (the repository folder location is specified under Help &gt; Updater Settings menu).</td>
<td>Click</td>
</tr>
<tr>
<td>Updater Settings...</td>
<td>Alt-S</td>
<td>Opens the updater settings window to configure manual versus automatic updates, proxy settings for Internet connections, repository folder where the downloaded software and firmware releases will be stored.</td>
<td>None</td>
</tr>
<tr>
<td>User Preferences</td>
<td></td>
<td>Opens the user preference window to enable or disable collect of features usage statistics.</td>
<td>None</td>
</tr>
</tbody>
</table>

4.1.4 Social links

Developer communities on popular social platforms such as Facebook™, Twitter™, STM32 YouTube™ channel, as well as ST Community can be accessed from the STM32CubeMX toolbar (see Figure 23).

Figure 23. Link to social platforms
4.2 New Project window

The New Project window is accessible through the File Menu or directly through shortcuts from the Home page (see Figure 24).

Figure 24. New Project window shortcuts

The main purpose here is to select from the STM32 portfolio a microcontroller or board part number that best fits the user application needs.

This window shows two tabs to choose from:
- the MCU selector tab (offering a list of target processors)
- a Board selector tab (showing a list of STMicroelectronics boards)
- A Cross selector tab (allow the user to find, for a given MCU/MPU part number and for a set of criteria, the best replacement within the STM32 portfolio).
4.2.1 MCU selector

MCU selection

The MCU selector allows filtering on a combination of criteria: series, lines, packages, peripherals, or additional characteristics such as price, memory size or number of I/Os (see Figure 25), and on their graphics capabilities as well.

![Figure 25. New Project window - MCU selector](image)

MCU selection based on graphics criteria

Selecting the checkbox to enable the Graphic Choice refreshes the MCU selector view (as shown in Figure 26) with:

1. A set of Graphics specific filtering criteria
2. The list of MCUs, meeting these criteria along with their graphics performance score. The graphics performance score is an indicative estimation of the graphics performance that can be achieved using the MCU for the selected graphics system configuration: the higher the score, better is the performance. It is shown in the GFX column. Moreover, selecting an MCU from this list will allow to use graphical stacks in the project.
3. A graphics summary panel, showing the minimum requirements for pixel clock and graphics RAM size to meet the selected graphics criteria.

It also displays the performance ranges (maximum system clock and Graphics performance score) that can be achieved with the current list of MCUs.

Parameters descriptions are provided in tooltips (to display: hover the mouse over the parameter name).
Export to Excel feature

Clicking on the icon allows the user to save the MCU table information to an Excel file.
Show favorite MCUs feature

Clicking the star icon for an MCU from the list marks it as favorite, see Figure 27.

Figure 27. Marking an MCU as favorite

MCU close selector feature

When the number of MCUs found is lower than 50, the selector offers to list the MCUs with close features (see Figure 28). Clicking the Display similar items button displays them (see Figure 29): by default, MCUs are sorted first by matching ratio, then by part number. For close MCUs (those with a matching ratio lower than 100%) rows are shown in gray and non matching cells are highlighted in dark gray.
Note: A matching percentage is computed for each user selected criteria, for example:
- when requesting four instances of the CAN peripheral, the MCUs with only three instances will reach a 75% match on the CAN criteria.
- if the maximum price criteria is selected, the matching ratio for a given MCU will be the
maximum requested price divided by the actual MCU price. In the case of a minimum price criteria, the matching ratio will be the MCU price divided by the minimum requested price. Finally, all criteria ratios are averaged to give the Match column percentage value.

4.2.2 Board selector

The **Board selector** allows filtering on STM32 board types, Series and peripherals (see **Figure 30**). Only the default board configuration is proposed. Alternative board configurations obtained by reconfiguring jumpers or by using solder bridges are not supported.

When a board is selected, the **Pinout** view is initialized with the relevant MCU part number along with the pin assignments for the LCD, buttons, communication interfaces, LEDs, and other functions. Optionally, the user can choose to initialize it with the default peripheral modes.

When a board configuration is selected, the signals change to 'pinned', i.e. they cannot be moved automatically by STM32CubeMX constraint solver (user action on the peripheral tree, such as the selection of a peripheral mode, will not move the signals). This ensures that the user configuration remains compatible with the board.

![Figure 30. New Project window - Board selector](image)

4.2.3 Cross selector

**Part number selection**

The **Cross selector** allows users to find products of the STM32 portfolio that best replace the MCU or MPU they are currently using (from ST or other silicon vendors).

To access this functionality, STM32CubeMX data must be up to date. This is ensured by clicking Refresh Data from the Refresh Data window accessible through under the Help menu (see **Figure NewFigure_CrossSelector_1.png**).
Clicking “ACCESS TO CROSS SELECTOR” under the “Start my project from Cross Selector” section of the main page opens the New Project window on the Cross selector tab. Two drop downs menus allow the user to select the vendor and the part number of the product to be compared to (see Figure 32). A part number can also be entered partially: STM32CubeMX proposes a list of matching products (see Figure 33).

Figure 31. Cross selector - Data refresh prerequisite

Figure 32. Cross selector - Part number selection per vendor
Compare cart

Once a part number is selected, a list of matching ST part number candidates is displayed along with their matching ratio in the Matching ST candidates panel.

By default, the three closest matches are selected and added to the compare cart along with the part number to be compared to (see Figure 34).

This selection can be changed anytime in the Matching ST candidates panel.

The comparison can be customized: the features to be used for comparison can be unselected when considered as irrelevant and their level of importance can be adjusted. These choices will affect the computed matching ratio.

The comparison is disabled for features that are not supported on the part number to be compared with, or when the feature information is unavailable.
Buttons are available to manipulate and save a copy of the compare cart view:
- to hide criteria that are not used for the comparison or show all criteria.
- to come back to default STM32CubeMX comparison settings
- to copy and paste the current cart view in a document or email.

**MCU/MPU selection for a new project**

Clicking an STM32 part number from the compare cart, selects it in the MCU/MPU Selector tab. Then, clicking will create a new project for that part number. (see Figure 35).

**Figure 35. Cross selector - Part number selection for a new project**

Clicking the Cross Selector Tab allows the user to go back to the cart and change the current selection for another part number.

### 4.3 Project page

Once an STM32 part number or a board has been selected or a previously saved project has been loaded, the project page opens, showing the following set of views (refer to dedicated section for detailed description):
- Pinout & Configuration
- Clock Configuration
- Project Manager
- Tools

The user can move across them without impacting his currently saved configuration.

A **GENERATE CODE** button is always accessible for the user to click and allows to generate the code corresponding to the current project configuration.

Moreover, thanks to convenient navigation breadcrumbs (see Figure 36), the user can detect what its current location is in STM32CubeMX user interface, and can move to other...
locations:

- to the home page by clicking the Home breadcrumb
- to the new project window by clicking the part number
- back to the project page by clicking the project name (or Untitled if the project does not have a name yet).

Figure 36. STM32CubeMX Main window upon MCU selection
Selecting a board, then answering **No** in the dialog window requesting to initialize all peripherals to their default mode, automatically sets the pinout for this board. However, only the pins set as GPIOs are marked as configured, i.e. highlighted in green, while no peripheral mode is set. The user can then manually select from the peripheral tree the peripheral modes required for its application (see Figure 37).

**Figure 37. STM32CubeMX Main window upon board selection (peripherals not initialized)**
Selecting a board, then accepting to initialize all peripherals to their default mode, automatically sets both the pinout and the default modes for the peripherals available on the board. This means that STM32CubeMX will generate the C initialization code for all the peripherals available on the board and not only for those relevant to the user application (see Figure 38).

Figure 38. STM32CubeMX Main window upon board selection (peripherals initialized with default configuration)

4.4 Pinout & Configuration view

The Pinout & Configuration view comes with the following main panels, function and menu:

- A Component list that can be visualized in alphabetical order and per categories. By default, it consists of the list of peripheral and middleware that the selected MCU supports. Selecting a component from that list will open two additional panels (Mode and Configuration) that allow to set its functional mode and configure its initialization parameters that will be included in the generated code.
- A Pinout view that shows a graphic representation of the pinout for the selected package (e.g. BGA, QFP) where each pin is represented with its name (e.g. PC4) and its current alternate function assignment, if any.
- A System view that gives an overview of all the software configurable components: GPIOs, peripherals, middleware and additional software components. Clickable
buttons allow opening the configuration options for the given component (Mode and Configuration panels). The button icon color reflects the status of the configuration status.

- An **Additional Software** function that allows to select, for the current project, software components that are not available by default. Selecting an additional software component will update the **Pinout & Configuration** view accordingly.

- A **Pinout** menu that allows to perform pinout related actions such as clear pinout configuration, export pinout configuration as csv file, etc...

**Tips**

- You can re-dimension the different panels at will: hovering the mouse over a panel border display a two-ended arrow: right-click and pull in a direction to either extend or reduce the panel.

- You can show/hide the Configuration, Mode, Pinout and System views using the open and close arrows.

### 4.4.1 Component list

The component list shows all the components available for the project. Selecting a component from the component list, opens the Mode and Configuration panels.

**Contextual help**

The **Contextual Help** window is displayed when hovering the mouse over a peripheral or a middleware short name.

By default, the window displays the extended name and source of configuration conflicts if any (see **Figure 39**).

![Figure 39. Contextual Help window (default)](image)

Clicking the **details and documentation** link (or CTRL+d) provides additional information such as summary and reference documentation links (see **Figure 40**). For a given peripheral, clicking **Datasheet** or **Reference manual** opens the corresponding document, stored in STM32CubeMX repository folder, at the relevant chapter. Since microcontrollers
Datasheets and reference manuals are downloaded to STM32CubeMX repository only upon users’ request, a functional Internet connection is required:

- To check your Internet connection, open the **Connection** tab from the **Help > Updater Settings** menu.
- To request the download of reference documentation for the currently selected microcontroller, click **Refresh** from the **Help > Refresh Data** menu window.

**Figure 40. Contextual Help detailed information**

**Icons and color schemes**

*Table 5* shows the icons and color scheme used in the component list view and the corresponding color scheme in the Mode panel.

**Table 5. Component list, mode icons and color schemes**

<table>
<thead>
<tr>
<th>Display</th>
<th>Component status</th>
<th>Corresponding Mode view / Tooltips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain black text</td>
<td>The peripheral is not configured (no mode is set) and all modes are available.</td>
<td></td>
</tr>
<tr>
<td>Example: <strong>UART5</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gray italic text</td>
<td>Peripheral is not available because some constraints are not solved. See tooltip.</td>
<td></td>
</tr>
<tr>
<td>Example: <strong>LWIP</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.4.2 Component Mode panel

Select a component from the component list on the left panel to open the **Mode** panel.

The **Mode** panel helps the user configuring the MCU pins based on a selection of peripherals and of their operating modes. Since STM32 MCUs allow a same pin to be used by different peripherals and for several functions (alternate functions), the tool searches for the pinout configuration that best fits the set of peripherals selected by the user. STM32CubeMX highlights the conflicts that cannot be solved automatically (see Table 5).

The **Mode** panel also allows to enable middleware and other software components for the project.

**Note:** For some middleware (USB, FATS, LwIP), a peripheral mode must be enabled before activating the middleware mode. Tooltips guide the user through the configuration. For FatFs, a user-defined mode has been introduced. This allows STM32CubeMX to generate

<table>
<thead>
<tr>
<th>Display</th>
<th>Component status</th>
<th>Corresponding Mode view / Tooltips</th>
</tr>
</thead>
<tbody>
<tr>
<td>![green] ![red] <strong>ETH</strong></td>
<td>The peripheral is configured (at least one mode is set) and all other modes are available. The green check mark indicates that all parameters are properly configured, a cross indicates they are not.</td>
<td>![Mode] ![Activate Rx Err signal]</td>
</tr>
<tr>
<td>![yellow] ![triangle] <strong>USB_OTG_HS</strong></td>
<td>The peripheral is not configured (no mode is set) and at least one of its modes is unavailable.</td>
<td>![Mode] ![Enable] ![Disable] ![Configure] ![Activate] ![Activate Only] ![Configure Only] ![Configure Only]</td>
</tr>
<tr>
<td>![red] ![triangle] <strong>USB_OTG_HS</strong></td>
<td>The peripheral is configured (one mode is set) and at least one of its other modes is unavailable.</td>
<td>![Mode] ![Enable] ![Disable] ![Configure] ![Activate] ![Activate Only] ![Configure Only] ![Configure Only]</td>
</tr>
<tr>
<td>![red] ![circle] <strong>I2C2</strong></td>
<td>The peripheral is not configured (no mode is set) and no mode is available. Move the mouse over the peripheral name to display the tooltip describing the conflict.</td>
<td>![Mode] ![Enable] ![Disable] ![Configure] ![Activate] ![Activate Only] ![Configure Only] ![Configure Only]</td>
</tr>
<tr>
<td>![red] <strong>IRTIM</strong></td>
<td>Peripheral is not available because of constraints.</td>
<td>![IRTIM] ![Detailed Interface] ![Not available] ![Configuration] ![Configuration] ![Configuration] ![Configuration] ![Configuration] ![Configuration]</td>
</tr>
</tbody>
</table>
FatFs code without a predefined peripheral mode. Then, it will be up to the user to connect the middleware with a user-defined peripheral by updating the generated user_diskio.c/.h driver files with the necessary code.

### 4.4.3 Pinout view

Select **Pinout view** to show for the selected part number, a graphic representation of the pinout for the selected package (e.g. BGA, QFP...) where each pin is represented with its name (e.g. PC4), its configuration state and its current alternate function assignment if any (e.g. ETH_MII_RXD0), see Figure 41 for an example.

![Pinout view](image)

The **Pinout** view is automatically refreshed to match the user’s component configuration performed in the **Mode** panel.

Assigning pins directly through the Pinout view instead of the Mode panel requires a good knowledge of the MCU since each individual pin can be assigned to a specific function.
Tips and tricks
See Table 2: Home page shortcuts for list of menus and shortcuts.
- Use the mouse wheel to zoom in and out.
- Click and drag the chip diagram to move it.
- Click best fit to reset it to best suited position and size.
- Use Pinout > Export pinout menus to export the pinout configuration as .csv text format.
- Some basic controls, such as insuring blocks of pins consistency, are built-in. See Appendix A: STM32CubeMX pin assignment rules for details.

4.4.4 Pinout menu and shortcuts

Table 6. Pinout menu and shortcuts

<table>
<thead>
<tr>
<th>Name or Icon</th>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep Current Signals Placement</td>
<td>Ctrl-K</td>
<td>Prevents moving pin assignments to match a new peripheral operating mode. It is recommended to use the new pinning feature that can block each pin assignment individually and leave this checkbox unchecked.</td>
</tr>
<tr>
<td>Show User Label</td>
<td>None</td>
<td>Displays user defined labels in the Pinout view.</td>
</tr>
<tr>
<td>Undo Mode and pinout</td>
<td>Ctrl-Z</td>
<td>Undoes last configuration steps (one by one).</td>
</tr>
<tr>
<td>Redo Mode and pinout</td>
<td>Ctrl-Y</td>
<td>Redoes steps that have been undone (one by one).</td>
</tr>
<tr>
<td>Disable All Modes</td>
<td>Ctrl-D</td>
<td>Resets to “Disabled” all peripherals and middleware modes that have been enabled. The pins configured in these modes (green color) are consequently reset to “Unused” (gray color). Peripheral and middleware labels change from green to black (when unused) or gray (when not available).</td>
</tr>
<tr>
<td>Clear Pinouts</td>
<td>Ctrl-P</td>
<td>Clears user pinout configuration in the Pinout view. Note that this action puts all configured pins back to their reset state and disables all the peripheral and middleware modes previously enabled (whether they were using signals on pins or not).</td>
</tr>
<tr>
<td>Pins/Signals Option</td>
<td>Ctrl-O</td>
<td>Opens a window showing the list of all the configured pins together with the name of the signal on the pin and a Label field allowing the user to specify a label name for each pin of the list. For this menu to be active, at least one pin must have been configured. Click the pin icon to pin/unpin signals individually. Select multiple rows then right click to open contextual menu and select action to pin or unpin all selected signals at once. Click column header names to sort alphabetically by name or according to placement on MCU.</td>
</tr>
<tr>
<td>Clear Single Mapped Signals</td>
<td>Ctrl-M</td>
<td>Clears signal assignments to pins for signals that have no associated mode (highlighted in orange and not pinned).</td>
</tr>
</tbody>
</table>
### Table 6. Pinout menu and shortcuts (continued)

<table>
<thead>
<tr>
<th>Name or Icon</th>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
</table>
| List Pinout Compatible MCUs | Alt-L | Provides a list of MCUs that best match the pin configuration of the current project. The matching can be:  
– An exact match  
– A partial match with hardware compatibility: pin locations are the same, pin names may have been changed  
– A partial match without hardware compatibility: all signals could be mapped but not all at the same pin location  
Refer to Section 14: Tutorial 5: Exporting current project configuration to a compatible MCU. |
| Export pinout with Alt. Functions | - | Generates pin configuration as a .csv text file including alternate functions information. |
| Export pinout without Alt. Functions | Ctrl-U | Generates pin configuration as a .csv text file excluding alternate functions information. |
| Reset used GPIOs | Alt-G | Opens a window to specify the number of GPIOs to be freed among the total number of GPIO pins that are configured. |
| Set unused GPIOs | Ctrl-G | Opens a window to specify the number of GPIOs to be configured among the total number of GPIO pins that are not used yet.  
Specify their mode: Input, Output or Analog (recommended configuration to optimize power consumption).  
**Caution:** Before using this menu, make sure the debug pins (available under SYS peripheral) are set to access microcontroller debug facilities. |
| Layout reset | - | - |
|  | - | Zooms-in the pinout view. |
|  | - | Adjusts the chip pinout diagram to the best fit size. |
|  | - | Zooms-out the pinout view. |
|  | - | Rotates 90 degrees clock wise. |
|  | - | Rotate 90 degrees counter-clock wise. |
|  | - | Flips horizontally between bottom view and top view. |
|  | - | Flips vertically between bottom view and top view. |
|  | - | This Search field allows the user to search for a pin name, signal name or signal label in the Pinout view.  
When it is found, the pin or set of pins that matches the search criteria blinks on the Pinout view.  
Click the Pinout view to stop blinking. |
4.4.5 Pinout view advanced actions

Manually modifying pin assignments

To manually modify a pin assignment, follow the sequence below:
1. Click the pin in the Pinout view to display the list of all other possible alternate functions together with the current assignment highlighted in blue (see Figure 42).
2. Click to select the new function to assign to the pin.

![Figure 42. Modifying pin assignments from the Pinout view](image)

Manually remapping a function to another pin

To manually remap a function to another pin, follow the sequence below:
1. Press the CTRL key and click the pin in the Pinout view. Possible pins for relocation, if any, are highlighted in blue.
2. Drag the function to the target pin.

Caution: A pin assignment performed from the Pinout view overwrites any previous assignment.

Manual remapping with destination pin ambiguity

For MCUs with block of pins consistency (STM32F100x / F101x / F102x / F103x and STM32F105x / F107x), the destination pin can be ambiguous, e.g. there can be more than one destination block including the destination pin. To display all the possible alternative remapping blocks, move the mouse over the target pin.

Note: A “block of pins” is a group of pins that must be assigned together to achieve a given peripheral mode. As shown in Figure 43, two blocks of pins are available on a STM32F107xx MCU to configure the Ethernet peripheral in RMII synchronous mode: \{PC1, PA1, PA2, PA7, PC4, PC5, PB11, PB12, PB13, PB5\} and \{PC1, PA1, PA2, PD10, PD9, PD8, PB11, PB12, PB13, PB5\}. 
Resolving pin conflicts

To resolve the pin conflicts that may occur when some peripheral modes use the same pins, STM32CubeMX attempts to reassign the peripheral mode functions to other pins. The peripherals for which pin conflicts cannot be solved are highlighted in fuchsia with a tooltip describing the conflict.

If the conflict cannot be solved by remapping the modes, the user can try the following:

- If the Keep Current Signals Placement box is checked, try to select the peripherals in a different sequence.
- Uncheck the Keep Current Signals Placement box and let STM32CubeMX try all the remap combinations to find a solution.
- Manually remap a mode of a peripheral when you cannot use it because there is no pin available for one of the signals of that mode.

4.4.6 Keep Current Signals Placement

This checkbox is available from the Pinout menu. It can be selected or deselected at any time during the configuration. It is unselected by default.

It is recommended to keep the checkbox unchecked for an optimized placement of the peripherals (maximum number of peripherals concurrently used).

The Keep Current Signals Placement checkbox should be selected when the objective is to match a board design.

Keep Current Signals Placement is unchecked

This allows STM32CubeMX to remap previously mapped blocks to other pins in order to serve a new request (selection of a new peripheral mode or a new peripheral mode function) which conflicts with the current pinout configuration.
Keep Current Signals Placement is checked

This ensures that all the functions corresponding to a given peripheral mode remain allocated (mapped) to a given pin. Once the allocation is done, STM32CubeMX cannot move a peripheral mode function from one pin to another. New configuration requests are served if feasible within current pin configuration.

This functionality is useful to:

- lock all the pins corresponding to peripherals that have been configured using the Peripherals panel
- maintain a function mapped to a pin while doing manual remapping from the Pinout view.

Tip

If a mode becomes unavailable (highlighted in fuchsia), try to find another pin remapping configuration for this mode by following the steps below:

1. From the Pinout view, deselect the assigned functions one by one until the mode becomes available again.
2. Then, select the mode again and continue the pinout configuration with the new sequence (see Appendix A: STM32CubeMX pin assignment rules for a remapping example). This operation being time consuming, it is recommended to deselect the Keep Current Signals Placement checkbox.

Note: Even if Keep Current Signals Placement is unchecked, GPIO_ functions (excepted GPIO_EXTI functions) are not moved by STM32CubeMX.

4.4.7 Pinning and labeling signals on pins

STM32CubeMX comes with a feature allowing the user to selectively lock (or pin) signals to pins. This prevents STM32CubeMX from automatically moving pinned signals to other pins when resolving conflicts. Labels, that are used for code generation, can also be assigned to the signals (see Section 6.1 for details).

There are several ways to pin, unpin and label the signals:

1. From the Pinout view, right-click a pin with a signal assignment. This opens a contextual menu:
   a) For unpinned signals, select Signal Pinning to pin the signal. A pin icon is then displayed on the relevant pin. The signal can no longer be moved automatically (for example when resolving pin assignment conflicts).
   b) For pinned signals, select Signal Unpinning to unpin the signal. The pin icon is removed. From now on, to resolve a conflict (such as peripheral mode conflict), this signal can be moved to another pin, provided the Keep user placement option is unchecked.
   c) Select Enter User Label to specify a user defined label for this signal. The new label will replace the default signal name in the Pinout view.
2. From the **Pinout** menu, select **Pins/Signals Options**
   The Pins/Signals Options window (see **Figure 44**) lists all configured pins.

   **Figure 44. Pins/Signals Options window**

   ![Pins/Signals Options window](image)

   a) Click the first column to individually pin/unpin signals.
   b) Select multiple rows and right-click to open the contextual menu and select **Signal(s) Pinning** or **Unpinning**.
   c) Select the User Label field to edit the field and enter a user-defined label.
   d) Order list alphabetically by Pin or Signal name by clicking the column header. Click once more to go back to default i.e. to list ordered according to pin placement on MCU.

   **Note**: *Even if a signal is pinned, it is still possible however to manually change the pin signal assignment from the Pinout view: click the pin to display other possible signals for this pin and select the relevant one.*

### 4.4.8 System view

Select **System view** to show all the software configurable components: GPIOs, peripherals and middleware. Clickable buttons allow the user to open the mode and configuration options of the component. The button icon reflects the component configuration status (see **Table 7** for configuration states and Figure System view).

When the user changes the component configuration from the Configuration panel, the system view is automatically refreshed with the new configuration state.

If the user disables the component from the Mode panel, the system view is automatically refreshed and there is no longer a button showing for that component.
Table 7. Configuration states

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Green Checkmark]</td>
<td>Configuration is complete and correct.</td>
</tr>
<tr>
<td>![Exclamation Mark]</td>
<td>Configuration is correct but some parts remain to be configured (may be optional).</td>
</tr>
<tr>
<td>![X Mark]</td>
<td>Configuration is invalid and needs to be fixed for the generated C project to be functional.</td>
</tr>
</tbody>
</table>
GPIO, DMA and NVIC settings can be accessed either via a dedicated button (like other peripherals or via a tab in the Configuration panel (see Figure 46).

Figure 46. Configuration window tabs (GPIO, DMA and NVIC settings for STM32F4 Series)
4.4.9 Component Configuration panel

This panel appears when clicking on a component name in the left panel. It allows the user to configure the functional parameters required to initialize the peripheral or the middleware in the selected operating mode (see Figure 47). STM32CubeMX will use these settings to generate the corresponding initialization C code.

Figure 47. Peripheral Mode and Configuration view

The configuration window includes several tabs:

- **Parameter settings** to configure library dedicated parameters for the selected peripheral or middleware,
- **NVIC, GPIO and DMA settings** to set the parameters for the selected peripheral (see Section 4.4.13, Section 4.4.11 and Section 4.4.12 for configuration details).
- **User constants** to create one or several user defined constants, common to the whole project (see Section 4.4.10 for user constants details).

Invalid settings are detected and are:

- reset to minimum / maximum valid value if user choice is, respectively, smaller / larger than minimum / maximum threshold
- reset to previous valid value if the previous value is neither a maximum nor a minimum threshold value
- highlighted in fuchsia.
Table 8 describes peripheral and middleware configuration buttons and messages.

Table 8. Peripheral and Middleware Configuration window buttons and tooltips

<table>
<thead>
<tr>
<th>Buttons and messages</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shows and Hides the description panel.</td>
<td></td>
</tr>
<tr>
<td><strong>Tooltip</strong></td>
<td>Guides the user through the settings of parameters with valid min-max range. To display it, move the mouse over a parameter value from a list of possible values.</td>
</tr>
<tr>
<td><strong>I2C Clock Speed (Hz)</strong></td>
<td>Clicking on the gear icon allows to select whether to display hexadecimal or decimal values, or any value unchecked (No check option).</td>
</tr>
<tr>
<td><strong>Search (Ctrl+F)</strong></td>
<td>Resets the component back to its default configuration (initial settings from STM32CubeMX).</td>
</tr>
</tbody>
</table>

No check option

By default, STM32CubeMX checks that the parameter values entered by the user are valid. You can bypass this check by selecting the option No Check for a given parameter. This allows entering you any value (such as a constant) that might not be known by STM32CubeMX configuration.

The validity check can be bypassed only on parameters whose values are of integer type (either hexadecimal or decimal). It cannot be bypassed on parameters coming from a predefined list of possible values or on those which are of non-integer or text type.

To go back to the default mode (decimal or hexadecimal values with validity check enabled), enter a decimal or hexadecimal value and check the relevant option (hexadecimal or decimal check).

Caution: When a parameter depends upon another parameter that is set to No Check:

- Case of a parameter depending on another parameter for the evaluation of its minimum or maximum possible value: If the other parameter is set to No Check, the minimum or maximum value is no longer evaluated and checked.

- Case of a parameter depending on another parameter for the evaluation of its current value: If the other parameter is set to No Check, the value is no longer automatically derived. Instead, it is replaced with the formula text showing as variable the string of the parameter set to No check (see Figure 48).
4.4.10 User Constants configuration window

An **User Constants** tab is available to define user constants (see **Figure 49**). Constants are automatically generated in the STM32CubeMX user project within the main.h file (see **Figure 50**). Once defined, they can be used to configure peripheral and middleware parameters (see **Figure 51**).
Figure 50. Extract of the generated main.h file

```c
/* Includes -----------------------------------------------*/

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private define ------------------------------------------*/
#define CONSTANT_1 10
#define CONSTANT_2 0xff
#define CONSTANT_3 CONSTANT_1
#define CONSTANT_4 (CONSTANT_3+CONSTANT_1)*100/CONSTANT_1
#define CONSTANT_5 (CONSTANT_3 - CONSTANT_1)

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */
```

Figure 51. Using constants for peripheral parameter settings
Creating/editing user constants

Click the Add button to open the User Constants tab and create a new user-defined constant (see Figure 52).

A constant consists of:

- A name that must comply with the following rules:
  - It must be unique.
  - It shall not be a C/C++ keyword.
  - It shall not contain a space.
  - It shall not start with digits.

- A value
  The constant value can be (see Figure 49 for examples):
  - a simple decimal or hexadecimal value
  - a previously defined constant
  - a formula using arithmetic operators (subtraction, addition, division, multiplication, and remainder) and numeric value or user-defined numeric constants as operands
  - a character string: the string value must be between double quotes (example: “constant_for_usart”).

Once a constant is defined, its name and/or its value can still be changed: double-click the row that specifies the user constant to be modified. This opens the User Constants tab for edition. The change of constant name is applied wherever the constant is used. This does not affect the peripheral or middleware configuration state. However changing the constant value impacts the parameters that use it and might result in invalid settings (e.g. exceeding a maximum threshold). Invalid parameter settings will be highlighted in fuchsia.

Figure 52. Specifying user constant value and name
Deleting user constants

Click the **Remove** button to delete an existing user-defined constant.

The user constant is then automatically removed except in the following cases:

- When the constant is used for the definition of another constant. In this case, a popup window displays an explanatory message (see **Figure 53**).

  **Figure 53. Deleting an user constant is not allowed when the constant is already used for another constant definition**

- When the constant is used for the configuration of a peripheral or middleware library parameter. In this case, the user is requested to confirm the deletion since the constant removal will result in an invalid peripheral or middleware configuration (see **Figure 54**).

  **Figure 54. Deleting an user constant used for parameter configuration - Confirmation request**

Clicking **Yes** leads to an invalid peripheral configuration (see **Figure 55**)

  **Figure 55. Deleting a user constant used for peripheral configuration - Consequence on peripheral configuration**
Searching for user constants

The **Search Constants** field allows searching for a constant name or value in the complete list of user constants (see Figure 56 and Figure 57).

**Figure 56. Searching for a name in a user constant list**

**Figure 57. Searching for a value in a user constant list**
4.4.11 GPIO Configuration window

Click GPIO in the System view panel to open the GPIO configuration window that allows you to configure the GPIO pin settings (see Figure 58). The configuration is populated with default values that might not be adequate for some peripheral configurations. In particular, check if the GPIO speed is sufficient for the peripheral communication speed and select the internal pull-up whenever needed.

Note: GPIO settings can also be accessed for a specific peripheral instance via the dedicated window in the peripheral instance configuration window. In addition, GPIOs can be configured in output mode (default output level). The generated code will be updated accordingly.

Figure 58. GPIO Configuration window - GPIO selection

Click on a row or select a set of rows to display the corresponding GPIO parameters:

- **GPIO PIN state**
  It changes the default value of the GPIO Output level. It is set to low by default and can be changed to high.

- **GPIO mode** (analog, input, output, alternate function)
  Selecting a peripheral mode in the Pinout view automatically configures the pins with the relevant alternate function and GPIO mode.

- **GPIO pull-up/pull-down**
  It is set to a default value and can be configured when other choices are possible.

- **GPIO maximum output speed** (for communication peripherals only)
  It is set to Low by default for power consumption optimization and can be changed to a higher frequency to fit application requirements.

- **User Label**
  It changes the default name (e.g. GPIO_input) into a user defined name. The Pinout view is updated accordingly. The GPIO can be found under this new name via the Find menu.
The **Group by Peripherals** checkbox allows the user to group all instances of a peripheral under the same window (see **Figure 59**).

**Figure 59. GPIO configuration grouped by peripheral**

As shown in **Figure 60**, row multi-selection can be performed to change a set of pins to a given configuration at the same time.

**Figure 60. Multiple Pins Configuration**
4.4.12 DMA Configuration window

Click DMA in the System view to open the DMA configuration window.

This window is used to configure the generic DMA controllers available on the MCU. The DMA interfaces allow to perform data transfers between memories and peripherals while the CPU is running, and memory to memory transfers (if supported).

**Note:** Some peripherals (such as USB or Ethernet) have their own DMA controller, which is enabled by default or via the Peripheral Configuration window.

Clicking Add in the DMA configuration window adds a new line at the end of the DMA configuration window with a combo box proposing to choose between possible DMA requests to be mapped to peripherals signals (see Figure 61).

![Figure 61. Adding a new DMA request](image)

Selecting a DMA request automatically assigns a stream among all the streams available, a direction and a priority. When the DMA channel is configured, it is up to the application code to fully describe the DMA transfer run-time parameters such as the start address, etc....

The DMA request (called channel for STM32F4 MCUs) is used to reserve a stream to transfer data between peripherals and memories (see Figure 62). The stream priority will be used to decide which stream to select for the next DMA transfer.

DMA controllers support a dual priority system using the software priority first, and in case of equal software priorities, a hardware priority that is given by the stream number.
Additional DMA configuration settings can be done through the **DMA configuration** window:

- **Mode**: regular mode, circular mode, or peripheral flow controller mode (only available for the SDIO peripheral).
- **Increment Add**: the type of peripheral address and memory address increment (fixed or post-incremented in which case the address is incremented after each transfer). Click the checkbox to enable the post-incremented mode.
- **Peripheral data width**: 8, 16 or 32 bits
- Switching from the default direct mode to the **FIFO mode** with programmable **threshold**:
  a) Click the **Use FIFO** checkbox.
  b) Then, configure the **peripheral and memory data width** (8, 16 or 32 bits).
  c) Select between **single transfer** and **burst transfer**. If you select burst transfer, choose a burst size (1, 4, 8 or 16).

In case of memory-to-memory transfer (MemToMem), the DMA configuration applies to a source memory and to a destination memory.
4.4.13 NVIC Configuration window

Click NVIC in the System view to open the Nested Vector interrupt controller configuration window (see Figure 64).

Interrupt unmasking and interrupt handlers are managed within two tabs:

- The NVIC tab allows enabling peripheral interrupts in the NVIC controller and setting their priorities.
- The Code generation tab allows selecting options for interrupt related code generation.

Enabling interruptions using the NVIC tab view

The NVIC view (see Figure 64) does not show all possible interrupts but only the ones available for the peripherals selected in the Pinout & Configuration panels. System interrupts are displayed but can never be disabled.

Check/Uncheck the Show only enabled interrupts box to filter or not enabled interrupts.

Use the search field to filter out the interrupt vector table according to a string value. As an example, after enabling UART peripherals from the Pinout panel, type UART in the NVIC search field and click the green arrow close to it: all UART interrupts are then displayed.

Enabling a peripheral interrupt will generate of NVIC function calls HAL_NVIC_SetPriority and HAL_NVIC_EnableIRQ for this peripheral.
When FreeRTOS is enabled, an additional column is shown (see Figure 65).

In this case, all the interrupt service routines (ISRs) that are calling the interrupt safe FreeRTOS APIs must have a priority lower than the priority defined in the LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY parameter (the highest the value, the lowest the priority). The check in the corresponding checkbox guarantees that the restriction is applied.

If an ISR does not use such functions, the checkbox can be unchecked and any priority level can be set. It is possible to check/uncheck multiple rows (see rows highlighted in blue in Figure 65).
Peripheral dedicated interrupts can also be accessed through the NVIC window in the Peripheral Configuration window (see Figure 66).

Figure 65. NVIC Configuration tab - FreeRTOS enabled

Figure 66. I2C NVIC Configuration window
STM32CubeMX NVIC configuration consists in selecting a priority group, enabling/disabling interrupts and configuring interrupts priority levels (preemption and sub-priority levels):

1. Select a **priority group**

   Several bits allow to define NVIC priority levels. These bits are divided in two priority groups corresponding to two priority types: preemption priority and sub-priority. For example, in the case of STM32F4 MCUs, the NVIC priority group 0 corresponds to 0-bit preemption and 4-bit sub-priority.

2. In the interrupt table, click one or more rows to select one or more interrupt vectors. Use the widgets below the interrupt table to configure the vectors one by one or several at a time:

   - **Enable checkbox**: check/uncheck to enable/disable the interrupt.
   - **Preemption priority**: select a priority level. The preemption priority defines the ability of one interrupt to interrupt another.
   - **Sub-priority**: select a priority level. The sub-priority defines the interrupt priority level.

**Code generation options for interrupt handling**

The **Code Generation** view allows customizing the code generated for interrupt initialization and interrupt handlers:

- **Selection/Deselection of all interrupts for sequence ordering and IRQ handler code generation**

  Use the checkboxes in front of the column names to configure all interrupts at a time (see Figure 67). Note that system interrupts are not eligible for init sequence reordering as the software solution does not control it.
• **Default initialization sequence of interrupts**

  By default, the interrupts are enabled as part of the peripheral MSP initialization function, after the configuration of the GPIOs and the enabling of the peripheral clock. This is shown in the CAN example below, where `HAL_NVIC_SetPriority` and `HAL_NVIC_EnableIRQ` functions are called within `stm32xxx_hal_msp.c` file inside the peripheral `msp_init` function.

  Interrupt enabling code is shown in bold:

  ```c
  void HAL_CAN_MspInit(CAN_HandleTypeDef* hcan)
  {
    GPIO_InitTypeDef GPIO_InitStruct;
    if(hcan->Instance==CAN1)
    {
      /* Peripheral clock enable */
      __CAN1_CLK_ENABLE();
      /**CAN1 GPIO Configuration
      PD0     ------> CAN1_RX
      PD1     ------> CAN1_TX
      */
      GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
      GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  }
  ```
For EXTI GPIOs only, interrupts are enabled within the `MX_GPIO_Init` function:

```c
/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

For some peripherals, the application still needs to call another function to actually activate the interruptions. Taking the timer peripheral as an example, the `HAL_TIM_IC_Start_IT` function needs to be called to start the Timer input capture (IC) measurement in interrupt mode.

**Configuration of interrupts initialization sequence**
Checking **Select for Init sequence ordering** for a set of peripherals moves the `HAL_NVIC` function calls for each peripheral to a same dedicated function, named `MX_NVIC_Init`, defined in the main.c. Moreover, the `HAL_NVIC` functions for each peripheral are called in the order specified in the **Code generation** view bottom part (see Figure 68).

As an example, the configuration shown in Figure 68 generates the following code:

```c
/** NVIC Configuration */
void MX_NVIC_Init(void)
{

  /* CAN1_TX_IRQHandler interrupt configuration */
  HAL_NVIC_SetPriority(CAN1_TX_IRQn, 2, 2);
  HAL_NVIC_EnableIRQ(CAN1_TX_IRQn);

  /* PVD_IRQHandler interrupt configuration */
  HAL_NVIC_SetPriority(PVD_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(PVD_IRQn);

  /* FLASH_IRQHandler interrupt configuration */
  HAL_NVIC_SetPriority(FLASH_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(FLASH_IRQn);

  /* RCC_IRQn interrupt configuration */
```
Interrupts handler code generation
By default, STM32CubeMX generates interrupt handlers within the stm32xxx_it.c file. As an example:

```c
void NMI_Handler(void)
{
    HAL_RCC_NMI_IRQHandler();
}

void CAN1_TX_IRQHandler(void)
{
    HAL_CAN_IRQHandler(&hcan1);
}
```

The column Generate IRQ Handler allows controlling whether the interrupt handler function call shall be generated or not. Deselecting CAN1_TX and NMI interrupts from the Generate IRQ Handler column as shown in Figure 68 removes the code mentioned earlier from the stm32xxx_it.c file.

![Figure 68. NVIC Code generation – IRQ Handler generation](image)
### 4.4.14 FreeRTOS configuration panel

Through STM32CubeMX FreeRTOS configuration window, the user can configure all the resources required for a real-time OS application and reserve the corresponding heap. FreeRTOS elements are defined and created in the generated code using CMSIS-RTOS API functions. Follow the sequence below:

1. In the **Pinout & Configuration** tab, click FreeRTOS to reveal the Mode and configuration panels (see Figure 69).
2. Enable FreeRTOS in the Mode panel.
3. Go to the configuration panel to proceed with configuring FreeRTOS native parameters and objects, such as tasks, timers, queues, and semaphores. In the Config tab, configure Kernel and Software settings. In the Include parameters tab, select the API functions required by the application and this way, optimize the code size. Both Config and Include parameters will be part of the FreeRTOSConfig.h file.

![Figure 69. FreeRTOS configuration view](image-url)
Tasks and Queues Tab

As any RTOS, FreeRTOS allows structuring a real-time application into a set of independent tasks, with only one task being executed at a given time. Queues are meant for inter-task communications: they allow to exchange messages between tasks or between interrupts and tasks.

In STM32CubeMX, the **FreeRTOS Tasks and Queues** tab allows creating and configuring such tasks and queues (see **Figure 70**). The corresponding initialization code will be generated within main.c or freeRTOS.c if the option “generate code as pair of .c/.h files per peripherals and middleware” is set in the **Project Settings** menu.

The corresponding initialization code will be generated within main.c by default or within freeRTOS.c if the option “generate code as pair of .c/.h files per peripherals and middleware” is set in the **Project Manager** menu.

**Figure 70. FreeRTOS: configuring tasks and queues**

- **Tasks**
  - Under the **Tasks** section, click the **Add** button to open the **New Task** window where task **name**, **priority**, **stack size** and **entry function** can be configured (see **Figure 71**). These settings can be updated at any time: double-clicking a task row opens again the new task window for editing.
  - The entry function can be generated as weak or external:
    - When the task is generated as **weak**, the user can propose another definition than the one generated by default.
    - When the task is **extern**, it is up to the user to provide its function definition. By default, the function definition is generated including user sections to allow customization.

- **Queues**
  - Under the **Queues** section, click the **Add** button to open the **New Queue** window where the queue **name**, **size** and **item size** can be configured (see **Figure 71**). The queue size corresponds to the maximum number of items that the queue can hold at a
time, while the item size is the size of each data item stored in the queue. The item size can be expressed either in number of bytes or as a data type:

- 1 byte for uint8_t, int8_t, char and portCHAR types
- 2 bytes for uint16_t, int16_t, short and portSHORT types
- 4 bytes for uint32_t, int32_t, int, long and float
- 8 bytes for uint64_t, int64_t and double

By default, the FreeRTOS heap usage calculator uses 4 bytes when the item size cannot be automatically derived from user input.

These settings can be updated at any time: double-clicking a queue row opens again the new queue window for editing.

Figure 71. FreeRTOS: creating a new task
The following code snippet shows the generated code corresponding to Figure 70.

```c
/* Create the thread(s) */
/* definition and creation of defaultTask */
osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

/* definition and creation of Task_A */
osThreadDef(Task_A, StartTask_A, osPriorityHigh, 0, 128);
Task_AHandle = osThreadCreate(osThread(Task_A), NULL);

/* definition and creation of Task_B */
osThreadDef(Task_B, StartTask_B, osPriorityLow, 0, 256);
Task_BHandle = osThreadCreate(osThread(Task_B), NULL);

/* Create the queue(s) */
/* definition and creation of myQueue_1 */
osMessageQDef(myQueue_1, 16, 4);
myQueue_1Handle = osMessageCreate(osMessageQ(myQueue_1), NULL);

/* definition and creation of myQueue_2 */
osMessageQDef(myQueue_2, 32, 2);
myQueue_2Handle = osMessageCreate(osMessageQ(myQueue_2), NULL);
```

Timers, Mutexes and Semaphores

FreeRTOS timers, mutexes and semaphores can be created via the FreeRTOS Timers and Semaphores tab. They first need to be enabled from the Config tab (see Figure 72).

![FreeRTOS Configuration](image)

Figure 72. FreeRTOS - Configuring timers, mutexes and semaphores
Under each object dedicated section, clicking the Add button to open the corresponding New <object> window where the object specific parameters can be specified. Object settings can be modified at any time: double-clicking the relevant row opens again the New <object> window for edition.

**Note:** Expand the window if the newly created objects are not visible.

- **Timers**
  Prior to creating timers, their usage (USE_TIMERS definition) must be enabled in the software timer definitions section of the Configuration parameters tab. In the same section, timer task priority, queue length and stack depth can be also configured. The timer can be created to be one-shot (run once) or auto-reload (periodic). The timer name and the corresponding callback function name must be specified. It is up to the user to fill the callback function code and to specify the timer period (time between the timer being started and its callback function being executed) when calling the CMSIS-RTOS osTimerStart function.

- **Mutexes / Semaphores**
  Prior to creating mutexes, recursive mutexes and counting semaphores, their usage (USE_MUTEXES, USE_RECURSIVE_MUTEXES, USE_COUNTING_SEMAPHORES definitions) must be enabled within the Kernel settings section of the Configuration parameters tab.
  The following code snippet shows the generated code corresponding to Figure 72).

```c
/* Create the semaphores(s) */
/* definition and creation of myBinarySem01 */
osSemaphoreDef(myBinarySem01);
myBinarySem01Handle = osSemaphoreCreate(osSemaphore(myBinarySem01), 1);

/* definition and creation of myCountingSem01 */
osSemaphoreDef(myCountingSem01);
myCountingSem01Handle = osSemaphoreCreate(osSemaphore(myCountingSem01), 7);

/* Create the timer(s) */
/* definition and creation of myTimer01 */
osTimerDef(myTimer01, Callback01);
myTimer01Handle = osTimerCreate(osTimer(myTimer01), osTimerPeriodic, NULL);

/* definition and creation of myTimer02 */
osTimerDef(myTimer02, Callback02);
myTimer02Handle = osTimerCreate(osTimer(myTimer02), osTimerOnce, NULL);

/* Create the mutex(es) */
/* definition and creation of myMutex01 */
osMutexDef(myMutex01);
myMutex01Handle = osMutexCreate(osMutex(myMutex01));
```
/* Create the recursive mutex(es) */
/* definition and creation of myRecursiveMutex01 */
osMutexDef(myRecursiveMutex01);
myRecursiveMutex01Handle =
osRecursiveMutexCreate(osMutex(myRecursiveMutex01));

FreeRTOS heap usage

The FreeRTOS Heap usage tab displays the heap currently used and compares it to the TOTAL_HEAP_SIZE parameter set in the Config Parameters tab. When the total heap used crosses the TOTAL_HEAP_SIZE maximum threshold, it is shown in fuchsia and a cross of the same color appears on the tab (see Figure 73).

Figure 73. FreeRTOS Heap usage

4.4.15 Graphics frameworks and simulator

After selecting a graphics capable microcontroller (see MCU selection based on graphics criteria), the user can select one between STemWin and TouchGFX graphics frameworks.

For more details, refer to associated documentation and license terms available within STM32Cube MCU embedded software packages.

Using the Graphics simulator, the user will be able to compare the Graphics current configuration performance with simulated settings. These latter can be imported to be used in the project.

Enabling Graphics frameworks and prerequisites

The necessary peripherals and middleware must be correctly configured for a Graphics framework to be fully available. In the left panel (component list), hover the mouse over the GRAPHICS middleware component to show the tooltip detailing the prerequisites (see Figure 74).
Configuring Graphics frameworks

After selecting in the Mode panel a Graphics framework and the relevant Display interface, the GRAPHICS configuration panel becomes available to configure Graphics common parameters, framework specific parameters and launching framework specific tools. See the tutorials dedicated to each framework for details.

Using the Graphics simulator

Clicking the GFXSIMULATOR from the left panel opens the Graphics simulator user interface (see Figure 75).

- In the left panel the user can edit parameters to change the simulator default configuration.
- In the right panel the user can visualize the Graphics configuration of the current project.
- In the bottom panels the user can compare both configurations performance results for a set of Graphics use cases.

If satisfied with simulator results, the user can click the Import Settings button to import simulator settings in the current configuration.

Clicking Reset Configuration restores STM32CubeMX default settings for the Simulator Configuration.
4.4.16 Setting HAL timebase source

By default, the STM32Cube HAL is built around a unique timebase source, the Arm® Cortex® system timer (SysTick).

However, HAL-timebase related functions are defined as weak so that they can be overloaded to use another hardware timebase source. This is strongly recommended when the application uses an RTOS, since this middleware has full control on the SysTick configuration (tick and priority) and most RTOSs force the SysTick priority to be the lowest.

Using the SysTick remains acceptable if the application respects the HAL programming model, that is, does not perform any call to HAL timebase services within an Interrupt Service Request context (no dead lock issue).

To change the HAL timebase source, go to the SYS peripheral in the Component list panel and select a clock among the available sources: SysTick, TIM1, TIM2,... (see Figure 76).

![Figure 75. Saving changes](image)
When used as timebase source, a given peripheral is grayed and can no longer be selected (see Figure 77).

**Figure 77. TIM1 selected as HAL timebase source**
As illustrated in the following examples, the selection of the HAL timebase source and the use of FreeRTOS influence the generated code.

Example of configuration using SysTick without FreeRTOS

As illustrated in Figure 78, the SysTick priority is set to 0 (High) when using the SysTick without FreeRTOS.

**Figure 78. NVIC settings when using SysTick as HAL timebase, no FreeRTOS**

Interrupt priorities (in main.c) and handler code (in stm32f4xx_it.c) are generated accordingly:

- **main.c file**
  ```c
  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
  
  /* USER CODE BEGIN SysTick_IRQn 0 */
  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */
  /* USER CODE END SysTick_IRQn 1 */
  ```

- **stm32f4xx_it.c file**
  ```c
  /*@brief This function handles System tick timer. */
  void SysTick_Handler(void)
  {
    /* USER CODE BEGIN SysTick_IRQHandler 0 */
    /* USER CODE END SysTick_IRQHandler 0 */
    HAL_IncTick();
    HAL_SYSTICK_IRQHandler();
    /* USER CODE BEGIN SysTick_IRQHandler 1 */
    /* USER CODE END SysTick_IRQHandler 1 */
  }
  ```
Example of configuration using SysTick and FreeRTOS

As illustrated in Figure 79, the SysTick priority is set to 15 (Low) when using the SysTick with FreeRTOS.

Figure 79. NVIC settings when using FreeRTOS and SysTick as HAL timebase

As shown in the code snippets below, the SysTick interrupt handler is updated to use CMSIS-os osSystickHandler function.

- main.c file
  
  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);

- stm32f4xx_it.c file
  
  /**
   * @brief This function handles System tick timer.
   */
  void SysTick_Handler(void)
  {
    /* USER CODE BEGIN SysTick_IRQn 0 */

    /* USER CODE END SysTick_IRQn 0 */
    HAL_IncTick();
    osSystickHandler();
    /* USER CODE BEGIN SysTick_IRQn 1 */

    /* USER CODE END SysTick_IRQn 1 */
  }
Example of configuration using TIM2 as HAL timebase source

When TIM2 is used as HAL timebase source, a new stm32f4xx_hal_timebase_TIM.c file is generated to overload the HAL timebase related functions, including the `HAL_InitTick` function that configures the TIM2 as the HAL time-base source.

The priority of TIM2 timebase interrupts is set to 0 (High). The SysTick priority is set to 15 (Low) if FreeRTOS is used, otherwise is set to 0 (High).

![Figure 80. NVIC settings when using FreeRTOS and TIM2 as HAL timebase](image)

The stm32f4xx_it.c file is generated accordingly:
- SysTick_Handler calls osSystickHandler when FreeRTOS is used, otherwise it calls `HAL_SYSTICK_IRQHandler`.
- TIM2_IRQHandler is generated to handle TIM2 global interrupt.

4.5 Pinout & Configuration view for STM32MP1 Series

For the STM32MP1 Series the Pinout & Configuration view allows the user to:
- assign components to one or several run time contexts
- configure peripherals as boot devices
- select the peripherals to be managed by boot loaders
- Assign GPIOs to one runtime or boot context (see Figure 82).

These possibilities are offered in two different panels (see Figure 81)
1. from the component tree panel, that lists all supported peripherals and middleware (the “Show contexts” option must be enabled)
2. from each component mode panel, opened by clicking the component name.
4.5.1 Run time configuration

The STM32MP1 devices are multi-core (Arm® Cortex®-A7 dual-core and Cortex®-M4) and multi-firmware, each firmware executing on one of the cores. The association between firmware and core defines a runtime context where the firmware executes its code.

Three runtime contexts are available:
1. Cortex-A7 Non Secure running the Linux kernel
2. Cortex-A7 Secure running the SP_min
3. Cortex-M4 running the STM32Cube firmware.

Assigning a component to a runtime context means specifying which context(s) will control the component at runtime. Assignments to a Cortex-A7 context will be reflected in the device tree code generation, while assignments to the Cortex-M4 context will be reflected in STM32Cube based C code generation (refer to code generation sections for more details).

The component assignment to a context is done in the context dedicated column.
4.5.2 Boot stages configuration

Boot ROM peripherals selection

Several execution stages are needed by the microprocessor to be up and running.

The binary code embedded in the ROM is the first to be executed. It uses a default configuration to initialize the clock tree and all peripherals involved in the boot detection.

The peripherals managed by the boot ROM program can be selected as boot devices. This choice is done in the bootROM column (see Figure 83).

Figure 83. Select peripherals as boot devices

When a peripheral is set as boot device, it imposes a specific pinout: some signals have to be mapped exclusively on pins visible by the boot ROM and only these signals/pins will be taken into account by the boot ROM program.

When a functional mode of a ROM-bootable peripheral is set, the pinout linked to this mode is the same of that for a runtime context except for the signals imposed on specific pins by the boot ROM code.

During the boot step (boot ROM code execution), the peripheral is running only with the sub-set of bootable signals and pins. After boot, during runtime, the peripheral runs with all signals necessary to the selected functional mode.

Boot loader (A7BL) peripherals selection

When the board starts, the launching of each of the Cortex-A7 runtime contexts (Secure and Non Secure) on which a firmware executes (SP_min for Cortex-A7 Secure, Linux kernel for Cortex-A7 Non Secure) preceded by an early boot execution stage, that is before U-Boot relocation in DDR.

The Boot loader (A7BL) column is used to define which devices can be managed during this Boot loader Stage.

This assignment will be reflected in the different Device-Trees generated (refer to code generation sections for more details).
4.6 Pinout & Configuration view for STM32H7 dual-core product lines

Some STM32H7 product lines come with an Arm Cortex-M7 core, an Arm Cortex-M4 core and three power domains.

For such products, the **Pinout & Configuration** view allows the user to:

- For each peripheral and middleware: assign it to one core context or both, whenever possible. In case both contexts are selected, assign an “initializer” core to indicate on which core the peripheral or middleware initialization function shall be called.
- For each peripheral: view the power domain it belongs to.
- For GPIOs: assign it to a core or leave it free for other components that may require it. In this last case the GPIO initialization will be performed on the same core as the component reserving it (code will be generated accordingly).

For peripherals and middleware, these possibilities are offered in two different panels:

1. from the component tree panel, which lists all supported peripherals and middleware (clicking the gear icon enables the “Show contexts” option), see **Figure 84**
2. from each component mode panel, opened by clicking the component name.

**Figure 84. STM32H7 dual-core: peripheral and middleware context assignment**

For GPIOs (see **Figure 85**), assignment is done through the **Pinout** view directly or later and automatically through its selection in the platform settings panel of a middleware.
Figure 85. STM32H7 dual-core: GPIOs context assignment

Select a GPIO for a pin
Right-click the pin
Select Pin Reservation and
Either assign it immediately to a core
Or leave it free for later use by other resources
4.7 Clock Configuration view

STM32CubeMX Clock Configuration window (see Figure 86) provides a schematic overview of the clock paths, clock sources, dividers, and multipliers. Drop-down menus and buttons can be used to modify the actual clock tree configuration, to meet the application requirements.

Figure 86. STM32F469NIHx clock tree configuration view

Actual clock speeds are displayed and active. The use clock signals are highlighted in blue.
Out-of-range configured values are highlighted as shown in Figure 87 to flag potential issues. A solver feature is proposed to automatically resolve such configuration issues.

Figure 87. Clock tree configuration view with errors

Reverse path is supported: just enter the required clock speed in the blue field and STM32CubeMX will attempt to reconfigure multipliers and dividers to provide the requested value. The resulting clock value can then be locked by right clicking the field to prevent modifications.

STM32CubeMX generates the corresponding initialization code:
- main.c with relevant HAL_RCC structure initializations and function calls
- stm32xxxx_hal_conf.h for oscillator frequencies and \( V_{DD} \) values.

4.7.1 Clock tree configuration functions

External clock sources

When external clock sources are used, the user must previously enable them from the Pinout view available under the RCC peripheral.
Peripheral clock configuration options

Other paths, corresponding to clock peripherals, are grayed out. To become active, the peripheral must be properly configured in the Pinout view (e.g. USB). This view allows to:

- **Enter a frequency value for the CPU Clock (HCLK), buses or peripheral clocks**
  STM32CubeMX tries to propose a clock tree configuration that reaches the desired frequency while adjusting prescalers and dividers and taking into account other peripheral constraints (such as USB clock minimum value). If no solution can be found, STM32CubeMX proposes to switch to a different clock source or can even conclude that no solution matches the desired frequency.

- **Lock the frequency fields for which the current value should be preserved**
  Right click a frequency field and select Lock to preserve the value currently assigned when STM32CubeMX will search for a new clock configuration solution.
  The user can unlock the locked frequency fields when the preservation is no longer necessary.

- **Select the clock source that will drive the system clock (SYSCLK)**
  - External oscillator clock (HSE) for a user defined frequency.
  - Internal oscillator clock (HSI) for the defined fixed frequency.
  - Main PLL clock

- **Select secondary sources (as available for the product)**
  - Low-speed internal (LSI) or external (LSE) clock
  - I2S input clock
  - ...

- **Select prescalers, dividers and multipliers values.**

- **Enable the Clock Security system (CSS) on HSE when it is supported by the MCU**
  This feature is available only when the HSE clock is used as the system clock source directly or indirectly through the PLL. It allows detecting HSE failure and inform the software about it, thus allowing the MCU to perform rescue operations.

- **Enable the CSS on LSE when it is supported by the MCU**
  This feature is available only when the LSE and LSI are enabled and after the RTC or LCD clock sources have been selected to be either LSE or LSI.

- **Reset the Clock tree default settings by using the toolbar Reset button:**
  This feature reloads STM32CubeMX default clock tree configuration.

- **Undo/Redo user configuration steps by using the toolbar Undo/Redo buttons**

- **Detect and resolve configuration issues**
  Erroneous clock tree configurations are detected prior to code generation. Errors are highlighted in fuchsia and the Clock Configuration view is marked with a fuchsia cross (see Figure 87).
  Issues can be resolved manually or automatically by clicking the Resolve Clock Issue button that is enabled only if issues have been detected.
  The underlying resolution process follows a specific sequence:
  - Setting HSE frequency to its maximum value (optional).
  - Setting HCLK frequency then peripheral frequencies to a maximum or minimum value (optional).
  - Changing multiplexers inputs (optional).
d) Finally, iterating through multiplier/dividers values to fix the issue. The clock tree is cleared from fuchsia highlights if a solution is found. Otherwise an error message is displayed.

Note: To be available from the clock tree, external clocks, I2S input clock, and master clocks shall be enabled in RCC configuration in the Pinout view. This information is also available as tooltips.

The tool will automatically perform the following operations:
- Adjust bus frequencies, timers, peripherals and master output clocks according to user selection of clock sources, clock frequencies and prescalers/multipliers/dividers values.
- Check the validity of user settings.
- Highlight invalid settings in fuchsia and provide tooltips to guide the user to achieve a valid configuration.

The Clock Configuration view is adjusted according to the RCC settings (configured in RCC Pinout & Configuration views) and vice versa:
- If in RCC Pinout view, the external and output clocks are enabled, they become configurable in the Clock Configuration view.
- If in RCC Configuration view, the Timer prescaler is enabled, the choice of Timer clocks multipliers will be adjusted.

Conversely, the clock tree configuration may affect some RCC parameters in the configuration view:
- Flash latency: number of wait states automatically derived from $V_{DD}$ voltage, HCLK frequency, and power over-drive state.
- Power regulator voltage scale: automatically derived from HCLK frequency.
- Power over-drive is enabled automatically according to HCLK frequency. When the power drive is enabled, the maximum possible frequency values for AHB and APB domains are increased. They are displayed in the Clock Configuration view.

The default optimal system settings that is used at startup are defined in the system_stm32f4xx.c file. This file is copied by STM32CubeMX from the STM32CubeF4 MCU package. The switch to user defined clock settings is done afterwards in the main function.
Figure 86 gives an example of Clock tree configuration for an STM32F429x MCU and Table 9 describes the widgets that can be used to configure each clock.

Table 9. Clock configuration view widgets

<table>
<thead>
<tr>
<th>Format</th>
<th>Configuration status of the Peripheral Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Format" /></td>
<td>Active clock sources</td>
</tr>
</tbody>
</table>
| ![Format](image) | Unavailable settings are blurred or grayed out (clock sources, dividers,…)
| ![Format](image) | Gray drop down lists for prescalers, dividers, multipliers selection. |
| ![Format](image) | Multiplier selection |
| ![Format](image) | User defined frequency values |
| ![Format](image) | Automatically derived frequency values |
| ![Format](image) | User-modifiable frequency field |
| ![Format](image) | Right click blue border rectangles to lock/unlock a frequency field. Lock to preserve the frequency value during clock tree configuration updates. |
### 4.7.2 Recommendations

The **Clock Configuration** view is not the only entry for clock configuration, RCC and RTC peripherals can also be configured.

1. From the **Pinout & Configuration** view, go to the RCC mode panel to enable the clocks as needed: external clocks, master output clocks and Audio I2S input clock when available. Then go to the RCC configuration panel, and adjust the default settings if needed. Changes will be reflected in the **Clock Configuration** view. The defined settings may change the settings in the RCC configuration as well (see Figure 88).

![Figure 88. Clock tree configuration: enabling RTC, RCC clock source and outputs from Pinout view](image-url)
2. Go to the **RCC configuration** in the **Pinout & Configuration** view. The settings defined there for advanced configurations will be reflected in the **Clock configuration** view. The defined settings may change the settings in the RCC configuration.

**Figure 89. Clock tree configuration: RCC peripheral advanced parameters**

---

**4.7.3 STM32F43x/42x power-over drive feature**

STM32F42x/43x MCUs implement a power over-drive feature allowing to work at the maximum AHB/APB bus frequencies (e.g., 180 MHz for HCLK) when a sufficient $V_{DD}$ supply voltage is applied (e.g. $V_{DD} > 2.1$ V).

*Table 10* lists the different parameters linked to the power over-drive feature and their availability in STM32CubeMX user interface.
Table 10. Voltage scaling versus power over-drive and HCLK frequency

<table>
<thead>
<tr>
<th>Parameter</th>
<th>STM32CubeMX panel</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{DD}$ voltage</td>
<td></td>
<td>User-defined within a predefined range. Impacts power over-drive.</td>
</tr>
<tr>
<td>Power regulator voltage scaling</td>
<td></td>
<td>Automatically derived from HCLK frequency and power over-drive</td>
</tr>
<tr>
<td></td>
<td>Configuration (RCC)</td>
<td>(see Table 11).</td>
</tr>
<tr>
<td>Power Over Drive</td>
<td></td>
<td>This value is conditioned by HCLK and $V_{DD}$ values (see Table 11).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It can be enabled only if $V_{DD} \geq 2.2$ V.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When $V_{DD} \geq 2.2$ V it is either automatically derived from HCLK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or it can be configured by the user if multiple choices are possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. $HCLK = 130$ MHz).</td>
</tr>
<tr>
<td>$HCLK/ AHB$ clock maximum frequency value</td>
<td>Clock Configuration</td>
<td>Displayed in blue to indicate the maximum possible value. For example: maximum value is 168 MHz for HCLK when power over-drive cannot be activated (when $V_{DD} \leq 2.1$ V), otherwise it is 180 MHz.</td>
</tr>
<tr>
<td>$APB1/ APB2$ clock maximum frequency value</td>
<td></td>
<td>Displayed in blue to indicate maximum possible value</td>
</tr>
</tbody>
</table>

Table 11 gives the relations between power-over drive mode and HCLK frequency.

Table 11. Relations between power over-drive and HCLK frequency

<table>
<thead>
<tr>
<th>$V_{DD}$ &gt; 2.1 V required to enable power over-drive (POD)</th>
<th>Corresponding voltage scaling and power over-drive (POD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤120 MHz</td>
<td>Scale 3</td>
</tr>
<tr>
<td></td>
<td>POD is disabled</td>
</tr>
<tr>
<td>120 to 144 MHz</td>
<td>Scale 2</td>
</tr>
<tr>
<td></td>
<td>POD can be either disabled or enabled</td>
</tr>
<tr>
<td>144 to 168 MHz</td>
<td>Scale 1 when POD is disabled</td>
</tr>
<tr>
<td></td>
<td>Scale 2 when POD is enabled</td>
</tr>
<tr>
<td>168 to 180 MHz</td>
<td>POD must be enabled</td>
</tr>
<tr>
<td></td>
<td>Scale 1 (otherwise frequency range not supported)</td>
</tr>
</tbody>
</table>

4.7.4 Clock tree glossary

Table 12. Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSI</td>
<td>High speed Internal oscillator: enabled after reset, lower accuracy than HSE.</td>
</tr>
<tr>
<td>HSE</td>
<td>High speed external oscillator: requires an external clock circuit.</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase locked loop: used to multiply above clock sources.</td>
</tr>
<tr>
<td>LSI</td>
<td>Low speed Internal clock: low power clocks usually used for watchdog timers.</td>
</tr>
</tbody>
</table>
4.8 Project Manager view

This view (see Figure 90) comes with three tabs:

- General project setting: to specify the project name, location, toolchain, and firmware version.
- Code generation: to set code generation options such as the location of peripheral initialization code, library copy/link options, and to select templates for customized code.
- Advanced settings: dedicated to ordering STM32CubeMX initialization function calls.

The code will be generated in the project folder tree shown in Figure 91.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSE</td>
<td>Low speed external clock: powered by an external clock.</td>
</tr>
<tr>
<td>SYSCLK</td>
<td>System clock</td>
</tr>
<tr>
<td>HCLK</td>
<td>Internal AHB clock frequency</td>
</tr>
<tr>
<td>FCLK</td>
<td>Cortex free running clock</td>
</tr>
<tr>
<td>AHB</td>
<td>Advanced high performance bus</td>
</tr>
<tr>
<td>APB1</td>
<td>Low speed advanced peripheral bus</td>
</tr>
<tr>
<td>APB2</td>
<td>High speed advanced peripheral bus</td>
</tr>
</tbody>
</table>

Table 12. Glossary (continued)
Note: Some project settings options become read-only once the project is saved. To modify these options, the project must be saved as a new project using the File> Save Project as menu.

4.8.1 Project tab

The Project tab of the Project Settings window allows configuring the following options (see Figure 90):

- Project settings:
  - Project name: name used to create the project folder and the .ioc file name at a given project location
  - Project location: directory where the project folder is stored.
  - Application structure: select between Basic and Advanced options.
    - Basic structure: recommended for projects using one or no middleware. This structure consists in placing the IDE configuration folder at the same level as the sources, organized in sources and includes subfolders (see Figure 92)
    - Advanced structure: recommended when several middleware components are used in the project. It makes the integration of middleware applications easier (see Figure 93)
  - Toolchain folder location: by default, it is located in the project folder at the same level as the .ioc file.
  - Toolchain/IDE: selected toolchain
  - For the STM32MP1 Series only, OpenSTLinux settings: location of generated device tree and manifest version and contents for current project (see Figure 94). These information enable the synchronization of the right SW components versions with STM32CubeMP1 for Cortex® M and Linux, tf-a, u-boot for Cortex® A. It is important to take them into account especially to ensure one Cube firmware version is aligned with SW components for Cortex® A around OpenAMP / RPM link and resource management API.

Selecting Makefile under Toolchain/IDE leads to the generation of a generic gcc-based makefile.
Selecting *Other Toolchains (GPDSC)* generates a gpdsc file. The gpdsc file provides a generic description of the project, including the list and paths of drivers and other files (such as startup files) that are required for building the project. This allows extending STM32CubeMX project generation to any toolchain supporting gpdsc since the toolchain will be able to load a STM32CubeMX generated C project by processing the gpdsc file information. To standardize the description of embedded projects, the gpdsc solution is based on CMSIS-Pack.

- **Additional project settings for SW4STM32 and Atollic® TrueSTUDIO® toolchain:**
  Select the optional *Generate under root* checkbox to generate the toolchain project files in STM32CubeMX user project root folder or deselect it to generate them under a dedicated toolchain folder.
  
  STM32CubeMX project generation under the root folder allows to benefit from the following Eclipse features when using Eclipse-based IDEs such as SW4STM32 and TrueStudio®:
  
  – Optional copy of the project into the Eclipse workspace when importing a project.
  – Use of source control systems such as GIT or SVN from the Eclipse workspace.
  
  However, it shall be noted that choosing to copy the project into workspace will prevent any further synchronization between changes done in Eclipse and changes done in STM32CubeMX as there will be two different copies of the project.

- **Linker settings:** value of minimum heap and stack sizes to be allocated for the application. The default values proposed are 0x200 and 0x400 for heap and stack sizes, respectively. These values may need to be increased when the application uses middleware stacks.

- **Firmware package selection when more than one version is available (this is the case when successive versions implement the same API and support the same MCUs).** By default, the latest available version is used.

- **Firmware location selection option**

  The default location is the location specified under the *Help > updater settings* menu.
  
  Deselecting the *Use Default Firmware Location* checkbox allows the user to specify a different path for the firmware that will be used for the project (see *Figure 95*).
Figure 92. Selecting a basic application structure
Figure 93. Selecting an advanced application structure

![Advanced Application Structure Diagram]

Figure 94. OpenSTLinux settings (STM32MP1 Series only)

![OpenSTLinux Settings Table]

<table>
<thead>
<tr>
<th>OpenSTLinux Settings</th>
<th>DeviceTree Root Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C:\STM32CubeMX_Projects\DiscoMP1_project\DeviceTree\</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manifest Version</th>
<th>openstlinux-4.19-thud-mp1-19-01-11</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Manifest Content:</th>
<th>Community Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firmware Name</td>
<td>Community Version</td>
</tr>
<tr>
<td>TF-A</td>
<td>2.0</td>
</tr>
<tr>
<td>Linux</td>
<td>4.19</td>
</tr>
<tr>
<td>Cube</td>
<td>STM32Cube FW_MP1 V1.0.0</td>
</tr>
<tr>
<td>U-Boot</td>
<td>2018.11</td>
</tr>
</tbody>
</table>
The new location must contain at least a *Drivers* directory containing the HAL and CMSIS drivers from the relevant STM32Cube MCU package. An error message pops up if the folders cannot be found (see Figure 96).

To reuse the same *Drivers* folder across all projects that use the same firmware location, select the *Add the library files as reference* from the *Code generator* tab allows (see Figure 97).
Caution: STM32CubeMX manages firmware updates solely for this default location. Choosing another location will prevent the user from benefiting from automatic updates. The user must manually copy new driver versions to its project folder.

4.8.2 Code Generator tab

The Code Generator tab allows specifying the following code generation options (see Figure 98):

- STM32Cube Firmware Library Package option
- Generated files options
- HAL settings options
- Custom code template options

STM32Cube Firmware Library Package option

The following actions are possible:

- Copy all used libraries into the project folder
  STM32CubeMX will copy to the user project folder, the drivers libraries (HAL, CMSIS) and the middleware libraries relevant to the user configuration (e.g. FatFs, USB, ..).
- Copy only the necessary library files:
  STM32CubeMX will copy to the user project folder only the library files relevant to the user configuration (e.g., SDIO HAL driver from the HAL library, ..).
- Add the required library as referenced in the toolchain project configuration file
  By default, the required library files are copied to the user project. Select this option for the configuration file to point to files in STM32CubeMX repository instead: the user project folder will not hold a copy of the library files but only a reference to the files in STM32CubeMX repository.

Generated files options

This area allows the user to define the following options:

- Generate peripheral initialization as a pair of .c/.h files or keep all peripheral initializations in the main.c file.
- Backup previously generated files in a backup directory
  The .bak extension is added to previously generated .c/.h files.
  Keep user code when regenerating the C code.
  This option applies only to user sections within STM32CubeMX generated files. It does not apply to the user files that might have been added manually or generated via ftl templates.
- Delete previously generated files when these files are no longer needed by the current configuration. For example, uart.c/.h file are deleted if the UART peripheral, that was enabled in previous code generation, is now disabled in current configuration.
HAL settings options

This area allows selection one HAL settings options among the following:

- Set all free pins as analog to optimize power consumption
- Enable/disable Use the Full Assert function: the Define statement in the stm32xx_hal_conf.h configuration file will be commented or uncommented, respectively.

Custom code template options

To generate custom code, click the Settings button under Template Settings, to open the Template Settings window (see Figure 99).

The user will then be prompted to choose a source directory to select the code templates from, and a destination directory where the corresponding code will be generated.

The default source directory points to the extra_template directory, within STM32CubeMX installation folder, which is meant for storing all user defined templates. The default destination folder is located in the user project folder.

STM32CubeMX will then use the selected templates to generate user custom code (see Section 6.3: Custom code generation).
Figure 100 shows the result of the template configuration shown on Figure 99: a sample.h file is generated according to sample_h.ftl template definition.

Figure 98. Project Settings code generator
Figure 99. Template Settings window
4.8.3 Advanced Settings tab

Figure 101 shows the peripheral and/or middleware selected for the project.

Ordering initialization function calls

By default, the generated code calls the peripheral/middleware initialization functions in the order in which peripherals and middleware have been enabled in STM32CubeMX. The user can then choose to re-order them by modifying the Rank number using the up and down arrow buttons.

The reset button allows switching back to alphabetical order.

Disabling calls to initialization functions

If the “Not to be generated” checkbox is checked, STM32CubeMX does not generate the call to the corresponding peripheral initialization function. It is up to the user code to do it.

Choosing between HAL and LL based code generation for a given peripheral instance

Starting from STM32CubeMX 4.17 and STM32L4 Series, STM32CubeMX offers the possibility for some peripherals to generate initialization code based on Low Layer (LL)
drivers instead of HAL drivers: the user can choose between LL and HAL driver in the Driver Selector section. The code will be generated accordingly (see Section 6.2: STM32Cube code generation using Low Layer drivers).

Figure 101. Advanced Settings window

Unselecting the Visibility (Static) option, as shown for MX_I2C1_init function in Figure 101, allows the generation of the function definition without the static keyword and hence extends its visibility outside the current file (see Figure 102).

Figure 102. Generated init functions without C language “static” keyword

/* Private function prototypes --------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_LPTIM1_Init(void);
static void MX_LPTIM2_Init(void);
void MX_I2C1_Init(void);
static void MX_I2C2_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_Init(void);*/
Caution: For the STM32MP1 Series only
By default the SystemClock_Config function is called in STM32Cube Cube firmware main() function since the 'Not generate Function call' box in Project Manager/Advanced Settings panel is not activated by default (see Figure 101).
This configuration is valid for running STM32Cube firmware in engineering mode (Cortex-M4 stand-alone mode).
This configuration is not valid for running STM32Cube firmware in production mode: the 'Not generate Function call' box must be checked under Project Manager/Advanced Settings panel so that there is no call to SystemClock_Config() in the main() function.

4.9 Import Project window

The Import Project menu eases the porting of a previously-saved configuration to another MCU. By default the following settings are imported:

- Pinout tab: MCU pins and corresponding peripheral modes. The import fails if the same peripheral instances are not available in the target MCU.
- Clock configuration tab: clock tree parameters.
- Configuration tab: peripherals and middleware libraries initialization parameters.
- Project settings: choice of toolchain and code generation options.

To import a project, proceed as follows:

1. Select the Import project icon that appears under the File menu after starting a New Project and once an MCU has been selected.
   The menu remains active as long as no user configuration settings are defined for the new project, that is just after the MCU selection. It is disabled as soon as a user action is performed on the project configuration.

2. Select File > Import Project for the dedicated Import project window to open. This window allows to specify the following options:
   - The STM32CubeMX configuration file (.ioc) pathname of the project to import on top of current empty project.
   - Whether to import the configuration defined in the Power Consumption Calculator tab or not.
   - Whether to import the project settings defined through the Project > Settings menu: IDE selection, code generation options and advanced settings.
   - Whether to import the project settings defined through the Project > Settings menu: IDE selection and code generation options.
Whether to attempt to import the whole configuration (automatic import) or only a subset (manual import).

a) Automatic project import (see Figure 103)

Figure 103. Automatic project import
b) Manual project import

In this case, checkboxes allow the user to manually select the set of peripherals (see *Figure 104*).

Select the **Try Import** option to attempt importing.

*Figure 104. Manual project import*
The Peripheral List indicates:

- The peripheral instances configured in the project to be imported
- The peripheral instances, if any exists for the MCU currently selected, to which the configuration has to be imported. If several peripheral instances are candidate for the import, the user needs to choose one.

Conflicts can occur when importing a smaller package with less pins or a lower-end MCU with less peripheral options.

Click the **Try Import** button to check for such conflicts: the Import Status window and the Peripheral list get refreshed to indicate errors (see Figure 105), warnings and whether the import has been successful or not:

- Warning icons indicate that the user has selected a peripheral instance more than once and that one of the import requests will not be performed.
- A cross sign indicates that there is a pinout conflict and that the configuration cannot be imported as such.

The manual import can be used to refine import choices and resolve the issues raised by the import trial. Figure 106 gives an example of successful import trial, that has been obtained by deselecting the import request for some peripherals.

The **Show View** function allows switching between the different configuration tabs (pinout, clock tree, peripheral configuration) for checking influence of the "Try Import" action before actual deployment on current project (see Figure 106).
Figure 105. Import Project menu - Try import with errors
3. Choose **OK** to import with the current status or **Cancel** to go back to the empty project without importing.

Upon import, the Import icon gets grayed since the MCU is now configured and it is no more possible to import a non-empty configuration.
4.10 Set unused / Reset used GPIOs windows

These windows are used to configure several pins at the same time in the same GPIO mode.

To open them:

- Select Pinout > Set unused GPIOs from the STM32CubeMX menu bar.

Note: The user selects the number of GPIOs and lets STM32CubeMX choose the actual pins to be configured or reset, among the available ones.

Figure 107. Set unused pins window

- Select Pinout > Reset used GPIOs from the STM32CubeMX menu bar.

Depending whether the Keep Current Signals Placement option is checked or not on the toolbar, STM32CubeMX conflict solver will be able to move or not the GPIO signals to other unused GPIOs:

- When Keep Current Signals Placement is off (unchecked), STM32CubeMX conflict solver can move the GPIO signals to unused pins in order to fit in another peripheral mode.
- When Keep Current Signals Placement is on (checked), GPIO signals will not be moved and the number of possible peripheral modes becomes limited.

Refer to Figure 109 and Figure 110 and check the limitation(s) in available peripheral modes.

Figure 108. Reset used pins window
Figure 109. Set unused GPIO pins with Keep Current Signals Placement checked
4.11 Update Manager windows

Three windows can be accessed through the Help menu available from STM32CubeMX menu bar:

1. Select Help > Check for updates to open the Check Update Manager window and find out about the latest software versions available for download.

2. Select Help > Manage embedded software packages to open the Embedded Software Package Manager window and find out about the embedded software packages available for download. It also allows checking for package updates and removing previously installed software packages.

3. Select Help > Updater settings to open the Updater settings window and configure update mechanism settings (proxy settings, manual versus automatic updates, repository folder where embedded software packages are stored).

Refer to Section 3.4: Getting updates using STM32CubeMX for a detailed description of these windows.
4.12 Additional software component selection window

This window can be opened at any time when working on the project. It allows the user to select additional software components for the current project. This feature is currently not supported for multi-core products.

It includes three panels:

- **Software component** panel
  
  It is located on the top right side of the window. It displays the list of the software components that can be selected for the project.

- **Filter** panel
  
  It is located on the left side of the window and provides a set of criteria to filter the pack component list.

- **Condition Resolution** panel
  
  It is located on the bottom right side of the window, and is displayed by clicking a condition to be resolved. It provides a detailed description of the condition and proposes solutions to solve it if any can be found in the software component list.

The Additional Software Component selection window can be opened by clicking Additional Software from the Pinout & Configuration tab.

See Section 9: Support of additional software components using CMSIS-Pack standard for more details on how to handle additional software components through STM32CubeMX CMSIS-Pack integration.

4.12.1 Introduction on software components

Arm® Keil™ CMSIS-Pack standard defines a software component as a list of files. The component or each of the corresponding individual files can optionally refer to a condition that must resolve to true, otherwise the component or file is not applicable in the given context.

There are no component names. Instead, each component is uniquely identified for a given vendor pack by the combination of class name, group name and a version. Additional categories, such as sub-group and variant can be assigned.

By default, the window shows a component list grouped by vendor name (first column) and pack or bundle version (second column), see Figure 111 for an example. Clicking the button expands the view and displays all software components details (see Figure 112).
Figure 111. Additional software components - collapsed view showing packs and bundles

Figure 112. Additional software components - expanded view showing component details
4.12.2 Filter panel

To filter the software component list, choose pack vendor names and software component classes or enter a text string in the search field.

The resulting software component table is collapsed. Click the button to expand it and display all the components that match the filtering criteria.

Click to clear all filter checkboxes or to clear the filter for a given category (pack vendors or component classes).

4.12.3 Software component table

STM32CubeMX parses the pack .pdc file to extract the list of embedded software components and the corresponding features.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>Pack vendor name, as found in the pack name.</td>
</tr>
<tr>
<td>Pack / Bundle</td>
<td>Pack or bundle name.</td>
</tr>
<tr>
<td>Class(1)</td>
<td>Component class the component belongs to.</td>
</tr>
<tr>
<td>Pack Action</td>
<td>Click to install the pack, means that the pack is already installed.</td>
</tr>
<tr>
<td>Group/SubGroup</td>
<td>Group and optionally the subgroup the software component belongs to.</td>
</tr>
<tr>
<td>Variant</td>
<td>Optional field for software components that come in different variants (such as debug and release).</td>
</tr>
<tr>
<td>Condition</td>
<td>Short name describing the condition (such as CMSIS-RTOS dependencies).</td>
</tr>
<tr>
<td>Status</td>
<td>Status of the condition to be met (see Table 14 for the different condition statuses). Clicking the status opens the condition description in the bottom panel.</td>
</tr>
<tr>
<td>Description</td>
<td>Software component short description.</td>
</tr>
</tbody>
</table>

1. The ARM® Keil™ CMIS-Pack website, http://www.keil.com, lists the following classes:
   - Data Exchange: Software components for data exchange
   - File System: File drive support and file system
   - Graphics: Graphic libraries for user interfaces
   - Network: Network stack using Internet protocols
   - RTOS: Real-time operating systems
   - Safety: Components for testing application software against safety standards
   - Security: Encryption for secure communication or storage
   - USB: Universal serial bus stack
   - Wireless: Communication stacks such as Bluetooth®, WiFi®, and ZigBee®.
4.12.4 Software component conditions

The conditions are dependency rules applying to a given software component. The software component condition status is displayed using the icons shown in Table 14.

Table 14. Software component conditions icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>All software component conditions, if any, are resolved.</td>
</tr>
<tr>
<td>✗</td>
<td>One or more software component conditions cannot be resolved with current pack list and for currently selected MCU.</td>
</tr>
<tr>
<td>⚡</td>
<td>One or more conditions must be resolved using another software component of the same pack.</td>
</tr>
<tr>
<td>⚡</td>
<td>One or more conditions must be resolved using software components from another pack.</td>
</tr>
</tbody>
</table>

1. Click the condition status icon to display the conditions to be solved in the bottom panel, together with their description and the software component that could solve the issue, if any (see Figure 113 and Figure 114).

Figure 113. Dependency solving: solution found
2. Clicking a proposed solution in the bottom panel highlights the software component selection checkbox in the component table above (see Figure 115).

Figure 114. Dependency solving: no solution found

Figure 115. Software components conditions - Solution proposals and guidance
3. Selecting the recommended software component resolves the condition. The condition status changes to resolved, ✔, (see Figure 116).

Figure 116. Software components conditions resolved
4.12.5 Updating the tree view for additional software components

Once the selection of the software components required for the application is complete (see Figure 117), click OK to refresh STM32CubeMX window: the selected component appears in the tree view under Additional Software (see Figure 118).

Figure 117. Selection of additional software components

The current selection of additional software components appears in the tree view (see Figure 118).

Figure 118. Additional software components - Updated tree view
4.13 About window

This window displays STM32CubeMX version information.

To open it, select Help > About from the STM32CubeMX menu bar.

Figure 119. About window
STM32CubeMX tools

5 STM32CubeMX tools

5.1 Power Consumption Calculator view

For an ever-growing number of embedded systems applications, power consumption is a major concern. To help minimizing it, STM32CubeMX offers the Power Consumption Calculator tab (see Figure 120), which, given a microcontroller, a battery model and a user-defined power sequence, provides the following results:

- Average current consumption
  Power consumption values can either be taken from the datasheet or interpolated from a user specified bus or core frequency.
- Battery life
- Average DMIPs
  DMIPs values are directly taken from the MCU datasheet and are neither interpolated nor extrapolated.
- Maximum ambient temperature (T\text{\text{MAX}})
  According to the chip internal power consumption, the package type and a maximum junction temperature of 105 °C, the tool computes the maximum ambient temperature to ensure good operating conditions.

Current T\text{\text{MAX}} implementation does not account for I/O consumption. For an accurate T\text{\text{MAX}} estimate, I/O consumption must be specified using the Additional Consumption field. The formula for I/O dynamic current consumption is specified in the microcontroller datasheet.

The Power Consumption Calculator view allows developers to visualize an estimate of the embedded application consumption and lower it further at each power sequence step:

- Make use of low power modes when any available
- Adjust clock sources and frequencies based on the step requirements.
- Enable the peripherals necessary for each phase.

For each step, the user can choose VBUS as possible power source instead of the battery. This will impact the battery life estimation. If power consumption measurements are available at different voltage levels, STM32CubeMX will also propose a choice of voltage values (see Figure 123).

An additional option, the transition checker, is available for STM32L0, STM32L1, STM32L4 and STM32L4+ Series. When it is enabled, the transition checker detects invalid transitions within the currently configured sequence. It ensures that only possible transitions are proposed to the user when a new step is added.
5.1.1 Building a power consumption sequence

The default starting view is shown in Figure 120.

Figure 120. Power Consumption Calculator default view

Selecting a $V_{DD}$ value

From this view and when multiple choices are available, the user must select a $V_{DD}$ value.
Selecting a battery model (optional)

Optionally, the user can select a battery model. This can also be done once the power consumption sequence is configured.

The user can select a predefined battery or choose to specify a new battery that best matches its application (see Figure 121).

**Figure 121. Battery selection**

Power sequence default view

The user can now proceed and build a power sequence.

Managing sequence steps

Steps can be reorganized within a sequence (Add new, Delete a step, Duplicate a step, move Up or Down in the sequence) using the set of Step buttons (see Figure 122).

The user can undo or redo the last configuration actions by clicking the Undo button in the Power Consumption Calculator view or the Undo icon from the main toolbar.

**Figure 122. Step management functions**
Adding a step

There are two ways to add a new step:

- Click Add in the Power Consumption panel. The New Step window opens with empty step settings.
- Or, select a step from the sequence table and click Duplicate. A New Step window opens duplicating the step settings (see Figure 123).

Figure 123. Power consumption sequence: New Step default view

Once a step is configured, resulting current consumption and $T_{AMAX}$ values are provided in the window.
Editing a step
To edit a step, double-click it in the sequence table, this will open the Edit Step window.

Moving a step
By default, a new step is added at the end of a sequence. Click the step in the sequence table to select it and use the Up and Down buttons to move it elsewhere in the sequence.

Deleting a step
Select the step to be deleted and click the Delete button.

Using the transition checker
Not all transitions between power modes are possible. The Power Consumption Calculator power menu proposes a transition checker to detect invalid transitions or restrict the sequence configuration to only valid transitions.

Enabling the transition checker option prior to sequence configuration ensures that the user will be able to select only valid transition steps.

Enabling the transition checker option on an already configured sequence will highlight the sequence with a green frame if all transitions are valid (see Figure 124), or in fuchsia if at least one transition is invalid (fuchsia frame with description of invalid step highlighted in fuchsia, see Figure 125). In the latter case, the user can click the Show log button to find out how to solve the transition issue (see Figure 126).

Figure 124. Enabling the transition checker option on an already configured sequence - All transitions valid

![Sequence Table - All transitions valid](image)

Figure 125. Enabling the transition checker option on an already configured sequence - At least one transition invalid

![Sequence Table - At least one transition invalid](image)
Figure 126. Transition checker option - Show log
5.1.2 Configuring a step in the power sequence

The step configuration is performed from the Edit Step and New Step windows. The graphical interface guides the user by forcing a predefined order for setting parameters.

Their naming may differ according to the selected MCU Series. For details on each parameter, refer to glossary in Section 5.1.4 and to Appendix D: STM32 microcontrollers power consumption parameters, or to the electrical characteristics section of the datasheet.

The parameters are set automatically by the tool when there is only one possible value (in this case, the parameter cannot be modified and is grayed out). The tool proposes only the configuration choices relevant to the selected MCU.

To configure a new step:
1. Click Add or Duplicate to open the New step window or double-click a step from the sequence table to open the Edit step window.
2. Within the open step window, select in the following order:
   - The Power Mode
     Changing the Power Mode resets the whole step configuration.
   - The Peripherals
     Peripherals can be selected/deselected at any time after the Power Mode is configured.
   - The Power scale
     The power scale corresponds to the power consumption range (STM32L1) or the power scale (STM32F4).
     Changing the Power Mode or the Power Consumption Range discards all subsequent configurations.
   - The Memory Fetch Type
   - The VDD value if multiple choices available
   - The voltage source (battery or VBUS)
   - A Clock Configuration
     Changing the Clock Configuration resets the frequency choices further down.
     When multiple choices are available, the CPU Frequency (STM32F4) and the AHB Bus Frequency/CPU Frequency (STM32L1) or, for active modes, a user specified frequency. In this case, the consumption value will be interpolated (see Using interpolation).
3. Optionally set
   - A step duration (1 ms is the default value)
   - An additional consumption value (expressed in mA) to reflect, for example, external components used by the application (external regulator, external pull-up, LEDs or other displays). This value added to the microcontroller power consumption will impact the step overall power consumption.
4. Once the configuration is complete, the Add button becomes active. Click it to create the step and add it to the sequence table.
Using interpolation

For steps configured for active modes (Run, Sleep), frequency interpolation is supported by selecting CPU frequency as User Defined and entering a frequency in Hz (see Figure 127).

Figure 127. Interpolated power consumption
Importing pinout

*Figure 128* illustrates the example of the ADC configuration in the *Pinout* view: clicking **Enable IPs from Pinout** in the Power Consumption Calculator view selects the ADC peripheral and GPIO A (*Figure 129*).

The **Enable IPs from Pinout** button allows the user to automatically select the peripherals that have been configured in the *Pinout* view.

*Figure 128. ADC selected in Pinout view*
Selecting/deselecting all peripherals
Clicking **Enable All IPs** allows the user to select all peripherals at once.
Clicking **Disable All IPs** removes them as contributors to the consumption.

**Figure 129. Power Consumption Calculator Step configuration window:**
ADC enabled using import pinout

5.1.3 Managing user-defined power sequence and reviewing results
The configuration of a power sequence leads to an update of the Power Consumption Calculator view (see **Figure 130**):
- The sequence table shows all steps and step parameters values. A category column indicates whether the consumption values are taken from the datasheet or are interpolated.
- The sequence chart area shows different views of the power sequence according to a display type (e.g. plot all steps, plot low power versus run modes, ..)
- The results summary provides the total sequence time, the maximum ambient temperature ($T_{\text{MAX}}$), plus an estimate of the average power consumption, DMIPS, and battery lifetime provided a valid battery configuration has been selected.
Managing the whole sequence (load, save and compare)

From the power menu (see Figure 131), the current sequence can be saved, deleted or compared to a previously saved sequence that will be displayed in a dedicated popup window.

Figure 131. Sequence table management functions
Managing the results charts and display options

In the Display area, select the type of chart to display (sequence steps, pie charts, consumption per peripherals, ...). You can also click External Display to open the charts in dedicated windows (see Figure 132).

Right-click on the chart to access the contextual menus: Properties, Copy, Save as png picture file, Print, Zoom menus, and Auto Range to reset to the original view before zoom operations. Zooming can also be achieved by mouse selecting from left to right a zone in the chart and Zoom reset by clicking the chart and dragging the mouse to the left.

Figure 132. Power Consumption: Peripherals consumption chart

Overview of the Results summary area

This area provides the following information (see Figure 133):

- Total sequence time, as the sum of the sequence steps durations.
- Average consumption, as the sum of each step consumption weighed by the step duration.
- The average DMIPS (Dhrystone million instructions per second) based on Dhrystone benchmark, highlighting the CPU performance for the defined sequence.
- Battery life estimation for the selected battery model, based on the average power consumption and the battery self-discharge.
- \( T_{\text{MAX}} \): highest maximum ambient temperature value found during the sequence.

Figure 133. Description of the Results area
5.1.4 Power sequence step parameters glossary

The parameters that characterize power sequence steps are the following (refer to Appendix D: STM32 microcontrollers power consumption parameters for more details):

- Power modes
  To save energy, it is recommended to switch the microcontroller operating mode from running mode, where a maximum power is required, to a low-power mode requiring limited resources.

- $V_{CORE}$ range (STM32L1) or Power scale (STM32F4)
  These parameters are set by software to control the power supply range for digital peripherals.

- Memory Fetch Type
  This field proposes the possible memory locations for application C code execution. It can be either RAM, FLASH or FLASH with ART ON or OFF (only for families that feature a proprietary Adaptive real-time (ART) memory accelerator which increases the program execution speed when executing from Flash memory).
  The performance achieved thanks to the ART accelerator is equivalent to 0 wait state program execution from Flash memory. In terms of power consumption, it is equivalent to program execution from RAM. In addition, STM32CubeMX uses the same selection choice to cover both settings, RAM and Flash with ART ON.

- Clock Configuration
  This operation sets the AHB bus frequency or the CPU frequency that will be used for computing the microcontroller power consumption. When there is only one possible choice, the frequencies are automatically configured.
  The clock configuration drop-down list allows to configure the application clocks:
  - the internal or external oscillator sources: MSI, HSI, LSI, HSE or LSE
  - the oscillator frequency
  - other determining parameters, among them PLL ON, LSE Bypass, AHB prescaler value, LCD with duty

- Peripherals
  The peripheral list shows the peripherals available for the selected power mode. The power consumption is given assuming that peripherals are only clocked (e.g. not in use by a running program). Each peripheral can be enabled or disabled. Peripherals individual power consumptions are displayed in a tooltip. An overall consumption due to peripheral analog and digital parts is provided in the step Results area (see Figure 134).
The user can select the peripherals relevant for the application:

- None (*Disable All*),
- Some (using peripheral dedicated checkbox),
- All (*Activate All*),
- Or all from the previously defined pinout configuration (*Import Pinout*).

Only the selected and enabled peripherals are taken into account when computing the power consumption.

- **Step duration**
  The user can change the default step duration value. When building a sequence, the user can either create steps according to the application actual power sequence or define them as a percentage spent in each mode. For example, if an application
spends 30% in Run mode, 20% in Sleep and 50% in Stop, the user must configure a 3-step sequence consisting in 30 ms in Run, 20 ms in Sleep and 50 ms in Stop.

- Additional Consumption
  This field allows entering an additional consumption resulting from specific user configuration (e.g. MCU providing power supply to other connected devices).

5.1.5 Battery glossary

- Capacity (mAh)
  Amount of energy that can be delivered in a single battery discharge.

- Self-discharge (% / month)
  This percentage, over a specified period, represents the loss of battery capacity when the battery is not used (open-circuit conditions), as a result of internal leakage.

- Nominal voltage (V)
  Voltage supplied by a fully charged battery.

- Max. continuous current (mA)
  This current corresponds to the maximum current that can be delivered during the battery lifetime period without damaging the battery.

- Max. pulse current (mA)
  This is the maximum pulse current that can be delivered exceptionally, for instance when the application is switched on during the starting phase.

5.1.6 SMPS feature

Some microcontrollers (e.g. STM32L496xxxxP) allow the user to connect an external switched mode power supply (SMPS) to further reduce power consumption.

For such microcontrollers, the Power Consumption Calculator tool offers the following features:

- Selection of SMPS for the current project
  From the left panel, check the **Use SMPS** box to use SMPS (see Figure 135). By default, ST SMPS model is used.

- Selection of another SMPS model by clicking the **Change** button
  This opens the SMPS database management window in which the user can add a new SMPS model (see Figure 136). The user can then select a different SMPS model for the current sequence (see Figure 137, Figure 138 and Figure 139).

- Check for invalid SMPS transitions in the current sequence by enabling the SMPS checker
  To do this, select the checkbox to enable the checker and click the **Help** button to open the reference state diagram (see Figure 140).

- Configuration of SMPS mode for each step (see Figure 141)
  If the SMPS checker is enabled, only the SMPS modes valid for the current step are proposed.
Figure 135. Selecting SMPS for the current project
Figure 136. SMPS database - Adding new SMPS models

Figure 137. SMPS database - Selecting a different SMPS model
Figure 138. Current project configuration updated with new SMPS model

Figure 139. SMPS database management window with new model selected
Figure 140. SMPS transition checker and state diagram helper window
Figure 141. Configuring the SMPS mode for each step
5.1.7 **BLE support (STM32WB Series only)**

The Power Consumption tool allows the user to take into account the consumption related to the RF peripheral and corresponding BLE functional mode, combined with the usage of the SMPS feature.

![Figure 142. RF related consumption (STM32WB Series only)](image)
The BLE mode can be selected from the left panel and configured to reflect the user’s application relevant settings.

**Figure 143. RF BLE mode configuration (STM32WB Series only)**

5.1.8 Example feature (STM32MP1 and STM32H7 dual-core only)

Under the section “Sequence Examples”, the PCC tool allows to access examples: each example come with an explanatory slide-set and a ready-made sequence to be loaded in PCC (see **Figure 144**).
Figure 144. Power Consumption Calculator – Example set

Figure 145. Power Consumption Calculator – Example sequence loading

Clicking “Load Example N” loads the sequence corresponding to the example N (see Figure 145).

Clicking “Example N Presentation” displays the explanations for that example.
The example can be changed anytime: the new sequence can be either added to the current sequence, or replace it (see Figure 146).

**Figure 146. Power Consumption Calculator – Example sequence new selection**

![Figure 146](image)

**Note:** The examples are provided for a given part number and may require adjustments when used for a different part number. Also, after loading, it is recommended to edit each step and check settings.

### 5.2 DDR Suite (for STM32MP1 Series only)

DDR SDRAMs are complex high speed devices that need careful PCB design.

The STM32MP15 devices support the following DDR types:
- LPDDR2
- LPDDR3
- DDR3 / DDR3L

They are specified by the JEDEC standard (standardization of interfaces, commands, timings, packages and ballout).

STM32CubeMX has been extended to provide an exhaustive tool suite for the STM32MP1 DDR subsystem. It proposes the following key features.
- **Configuration of DDR** controller and PHY registers is managed automatically based on reduced set of editable parameters.
- **DDR testing** is offered based on a rich tests list. Tests go from basic to stress tests. User can also develop its own tests.
- **DDR tuning** of byte lanes delays is proposed to compensate board design imperfections.
**DDR configuration** is accessible like the other peripherals in the Pinout & Configuration view: clicking the DDR from the component panel, opens the mode and configuration panels.

**DDR Test suite testing and tuning** features are available from the Tools view.

The DDR suite relies on two important concepts:
- the **DDR timings** as key inputs for the configuration of the DDR Controller and PHY
- the tuning of DDR signals to compensate board design imperfections.

### 5.2.1 DDR configuration

STM32CubeMX allows to set DDR system parameters and JEDEC core timings. The timing parameters are available in the DDR datasheet.

**DDR type, width and density**

The DDR type, width and density parameter settings are required to proceed with the DDR configuration step. This can be done in the mode panel after selecting the DDR in the Pinout & Configuration view.

See Figure 147 for an example of LPDDR2 settings.

![Figure 147. DDR pinout and configuration settings](image)

Another example: for a configuration with two “DDR3 16 bits 2 Gb” chips, settings are “DDR3/DDR3L”, “32 bits” and 4 Gb”.

**Note:** Contexts for DDR IP cannot be changed, DDR is tied to “Cortex-A7 Non-Secure” identified as “Cortex-A7 NS” in the tool.
**DDR configuration**

Clicking on a parameter will show additional details in the DDR configuration footer.

- The DDR frequency is taken from the ‘Clock configuration’ tab, it cannot be changed in the DDR configuration.
- The ‘Relaxed Timing’ mode is used during bring-up phase for trying relaxed key DDR timings value (one t_{CK} added to t_{RC}, t_{RCD} and t_{RP} timings)
- Other parameters must be retrieved from the user DDR datasheet.
- Some parameters are read-only: they are for information only and depend on the DDR type.

Clicking “generate code” automatically computes the DDR node of the device tree (DDR Controller and DDR PHY registers values) based on these parameters.

**DDR3 configuration**

For DDR3, the configuration is made easier with the selection of a **Speed Bin / Grade** combination, instead of manually editing timing parameters.
The Speed Bin / Grade combination has to match the selected DDR. If the exact combination is not in the pick-list, “1066E / 6-6-6” must be selected for faster DDR Speed bin / Grade, whereas “1066G / 8-8-8” can be used as a relaxed configuration.
Timing edition is then optional and reserved for advanced users: select Show Advanced parameters to display the list.

**DDR tuning tab (read-only)**

Users can check modifications to tuning parameters via the tuning tab. These parameters are read-only in the DDR configuration panel (see Figure 149), are modified after tuning operations, and are related to DQS position and DQ line delay:

- ‘Slave DDL Phase’, ‘DQS delay fine tuning’ and ‘DQS# delay fine tuning’ defines the position of the DQS strobe signal for a particular byte. This position is the best one regarding DQ line eye diagram.
- ‘DQ bit x lane delay fine tuning’ defines the delay to apply on bit x of particular byte to compensate potential line length variation for this particular bit

![Figure 149. DDR tuning parameter](image)
5.2.2 Connection to the target and DDR register loading

To manage DDR tests and tuning, STM32CubeMX must establish a connection with the target and more specifically with U-Boot SPL using the DDR interactive protocol:

- the DDR interactive protocol is only available in the Basic boot scheme U-Boot SPL binary and supported over the UART4 peripheral instance
- when U-Boot SPL detects a connection to STM32CubeMX on UART4, it stops its initialization process and accepts commands from STM32CubeMX.

There are two connection options:
1. the U-Boot SPL binary is available in Flash memory
2. the U-Boot SPL needs to be loaded in SYSRAM because the DDR has not yet been tested nor tuned (and, consequently, is not fully functional yet).

Prerequisites

- Installation of ST-Link USB driver to perform firmware upgrades: for Windows, latest version of STSW-LINK009 must be used. For Linux, the STSW-LINK007 driver must be used. Both can be downloaded from www.st.com.

Connection to the target

The COM port must be selected to connect to the target, as indicated in Figure 150.
If U-Boot SPL loading in SysRAM is required, it can be performed through UART or USB using the STM32CubeProgrammer tool. If not automatically detected by STM32CubeMX, the STM32CubeProgrammer tool location must be specified in the Connection settings window: click to open it. U-Boot SPL file has to be manually selected in the build image folder.

Once up, the connection gives the various services and target information (see Figure 151).

**Output/Log messages**

STM32CubeMX outputs DDR suite related activity logs (see Figure 152) and interactive protocol communication logs (see Figure 153). They are displayed by enabling outputs from the Window menu.
Figure 152. DDR activity logs

Creating: STM32MP157CAx  
Initializing: STM32MP157CAx  
SYSRAM successfully loaded with: C:\u-boot-spl-stm32mp157c-dk2-basic.stm32  
DDR Test Suite connected to target board  
Target board configuration name: DDR3-1066/888 bin G 1x4Gb 533MHz v1.41  
Target board DDR size: 4 GBits  
Target board DDR frequency: 533.00MHz

Figure 153. DDR interactive logs

Host > Target info  
Target > Host info  
Target > Host step = 0 : DDR_RESET  
Target > Host name = DDR3-1066/888 bin G 1x4Gb 533MHz v1.41  
Target > Host size = 0x20000000  
Target > Host speed = 533000 kHz  
Host > Target step 3  
Target > Host step to 3:DDR READY  
Target > Host 1:DDR_CTRL_INIT_DONE  
Target > Host 2:DDR_PHY_INIT_DONE  
Target > Host 3:DDR READY  
Host > Target print mtr  
Target > Host mtr= 0x00001101  
Host > Target tuning help  
Target > Host tuning:5  
Target > Host 0:Read DOS gating:software read DOS Gating:  
Target > Host 1:Bit de-skew::  
Target > Host 2:Eye Training:or DQS training:  
Target > Host 3:Display registers::

DDR register loading (optional)

Once connected in DDR interactive mode, user can load the current DDR configuration in SYSRAM.

Figure 154. DDR register loading
This step is optional if the used U-Boot SPL already contains the required DDR configuration. It trigs the DDR Controller and PHY initialization with those registers, and allows the user to quickly test a configuration without generating the device tree and dedicated U-Boot SPL binary file.

5.2.3 DDR testing

Prerequisites
To proceed with DDR testing:
- The DDR suite must be in connected state
- The DDR configuration must be available in memory, either with the U-Boot SPL (with DDR register file in Device Tree) or in the DDR registers (see Section 5.2.2).

DDR test list
DDR tests are part of the U-Boot SPL (see Figure 155).

Figure 155. DDR test list from U-Boot SPL
New tests can be added by modifying the U-boot SPL.

Most of the tests come with parameters to be set prior to execution, such as:

- **Address**: the memory address where the test is executed. All writes and reads are performed on this address. The given address has to be located in the DDR memory region \([\text{DDR base address}, \text{DDR base address} + \text{DDR size}]\).
- On STM32MP15, DDR base address is 0xC0000000 (as an example, DDR size for 4 Gbits is 0x20000000).
- **Loop**: number of test iterations before verdict. Same test is repeated [Loop] times. Verdict OK if all tests are OK, KO otherwise.
- **Size**: the byte size of the region to test. Size must be a multiple of 4 (read/writes are performed on 32-bit unsigned integers) with minimal value equal to 4. Size can be up to DDR size.
- **Pattern**: the 32-bit pattern to be used for read / write operations.

The DDR Suite embeds an auto-correction feature preventing users to specify wrong values.

All tests are performed with Data cache disabled and Instruction cache enabled.

**DDR test results**

The test verdict is reported by the U-Boot SPL: the parameters used for the tests are recalled, along with Pass/Fail status and results details (see Figure 156). The test history is available in the output and Logs panels (see Figure 157).

![Figure 156. DDR test suite results](image)
5.2.4 DDR tuning

Prerequisites

The prerequisites to proceed with DDR tuning are:

- The DDR suite is in connected state
- A valid DDR configuration is available in memory, either with the U-Boot SPL (with DDR register file in Device Tree) or in the DDR registers (see DDR register loading (optional)).

Thanks to DDR tuning it is possible to compensate hardware design slight imperfections for best operations (see AN5122, available on www.st.com, for DDR design routing guidelines).

Tunable signals

The tunable signals are

- DQS signals: position for each data byte
- the 8 DQ bits: delay for each data byte.
Some DDR registers are dedicated to store the corresponding tuned settings:

- ‘Slave DDL Phase’, ‘DQS delay fine tuning’ and ‘DQS# delay fine tuning’ define the position of the DQS strobe signal for a particular byte: this position is the best one regarding DQ line eye diagram
- ‘DQ bit x lane delay fine tuning’ defines the delay to apply on bit x of particular byte to compensate potential line length variation for this particular bit.

**Note:** It is recommended to perform tuning on several boards to make sure that the tuned parameter variation is limited.

### Tuning process

Tuning is done in three consecutive steps (see **Figure 159**):

1. DQS gating
2. Bit deskew
3. Eye training

**Figure 159. DDR tuning process**

#### Bit deskew

The Bit deskew panel (see **Figure 160**) gives a graphical representation of

- the best DQS signal position for the given byte in order to adjust DQ line delay
- the delay to apply for each DQ line of the considered byte. The unit delay value is 20.56 ps. There are four steps. Bit lane delay is thus tunable from 0 to 61.68 ps.
Eye training (centering)

The Eye training (centering) panel (Figure 161) gives the final optimum position of the DQS signal in the half-period for each byte:

- DQS position can vary coarsely from 36 to 144 degrees (quarter period is 90 degrees)
- DQS position can then vary finely about the coarse position with 8 steps, from -61.68 to +82.24 ps

Propagating tuning results

Once tuning is complete, the DDR suite allows the user to propagate the result (tuned parameters) to the current DDR configuration (see Figure 162). The DDR Tuning tab is refreshed accordingly (see Figure 163).
**Figure 163. DDR configuration update after tuning**

The image shows a table and diagram illustrating DDR configuration before and after tuning. The table outlines various parameters such as Boot time, Boot ROM, Boot loader, and different DDR modes like DDR3/DDR4, Width, and Density for configurations like 16b, 16b, and 2b disable. The diagram compares the configuration settings before and after tuning, highlighting changes in values for different byte lanes.
6 STM32CubeMX C Code generation overview

6.1 STM32Cube code generation using only HAL drivers (default mode)

During the C code generation process, STM32CubeMX performs the following actions:

1. If it is missing, it downloads the relevant STM32Cube MCU package from the user repository. STM32CubeMX repository folder is specified in the Help > Updater settings menu.

2. It copies from the firmware package, the relevant files in Drivers/CMSIS and Drivers/STM32F4_HAL_Driver folders and in the Middleware folder if a middleware was selected.

3. It generates the initialization C code (.c/.h files) corresponding to the user MCU configuration and stores it in the Inc and Src folders. By default, the following files are included:

   - **stm32f4xx_hal_conf.h** file: this file defines the enabled HAL modules and sets some parameters (e.g. External High Speed oscillator frequency) to predefined default values or according to user configuration (clock tree).
   - **stm32f4xx_hal_msp.c** (MSP = MCU Support package): this file defines all initialization functions to configure the peripheral instances according to the user configuration (pin allocation, enabling of clock, use of DMA and Interrupts).
   - **main.c** is in charge of:
     - Resetting the MCU to a known state by calling the `HAL_init()` function that resets all peripherals, initializes the Flash memory interface and the SysTick.
     - Configuring and initializing the system clock.
     - Configuring and initializing the GPIOs that are not used by peripherals.
     - Defining and calling, for each configured peripheral, a peripheral initialization function that defines a handle structure that will be passed to the corresponding peripheral HAL init function which in turn will call the peripheral HAL MSP initialization function. Note that when LwIP (respectively USB) middleware is used, the initialization C code for the underlying Ethernet (respectively USB peripheral) is moved from main.c to LwIP (respectively USB) initialization C code itself.

   - **main.h** file:
     - This file contains the define statements corresponding to the pin labels set from the Pinout tab, as well as the user project constants added from the Configuration tab (refer to Figure 164 and Figure 165 for examples):
     
```c
#define MyTimeOut 10
#define LD4_Pin GPIO_PIN_12
#define LD4_GPIO_Port GPIOD
#define LD3_Pin GPIO_PIN_13
#define LD3_GPIO_Port GPIOD
#define LD5_Pin GPIO_PIN_14
#define LD5_GPIO_Port GPIOD
#define LD6_Pin GPIO_PIN_15
#define LD6_GPIO_Port GPIOD
```
In case of duplicate labels, a unique suffix, consisting of the pin port letter and the pin index number, is added and used for the generation of the associated define statements.

In the example of a duplicate I2C1 labels shown in Figure 166, the code generation produces the following code, keeping the I2C1 label on the original port B pin 6 define statements and adding B7 suffix on pin 7 define statements:

```c
#define I2C1_Pin GPIO_PIN_6
#define I2C1_GPIO_Port GPIOB
#define I2C1B7_Pin GPIO_PIN_7
#define I2C1B7_GPIO_Port GPIOB
```
In order for the generated project to compile, define statements shall follow strict naming conventions. They shall start with a letter or an underscore as well as the corresponding label. In addition, they shall not include any special character such as minus sign, parenthesis or brackets. Any special character within the label will be automatically replaced by an underscore in the define name.

If the label contains character strings between "[" or "]", only the first string listed is used for the define name. As an example, the label "LD6 [Blue Led]" corresponds the following define statements:

```c
#define LD6_Pin GPIO_PIN_15
#define LD6_GPIO_Port GPIOD
```

The define statements are used to configure the GPIOs in the generated initialization code. In the following example, the initialization of the pins labeled Audio_RST_Pin and LD4_Pin is done using the corresponding define statements:

```c
/*Configure GPIO pins : LD4_Pin Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin | Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

4. Finally it generates a Projects folder that contains the toolchain specific files that match the user project settings. Double-clicking the IDE specific project file launches the IDE and loads the project ready to be edited, built and debugged.

### 6.2 STM32Cube code generation using Low Layer drivers

For all STM32 Series except STM32H7 and STM32WB Series, STM32CubeMX allows the user to generate peripheral initialization code based either on the peripheral HAL driver or on the peripheral Low Layer (LL) driver.

The choice is made through the Project Manager view (see Section 4.8.3: Advanced Settings tab).

The LL drivers are available only for the peripherals which require an optimized access and do not have a complex software configuration. The LL services allow performing atomic operations by changing the relevant peripheral registers content:

- Examples of supported peripherals: RCC, ADC, GPIO, I2C, SPI, TIM, USART,…
- Examples of peripherals not supported by LL drivers: USB, SDMMC, FSMC.
The LL drivers are available within the STM32CubeL4 package:
- They are located next to the HAL drivers (stm32l4_hal_<peripheral_name>) within the Inc and Src directory of the STM32Cube_FW_L4_V1.6\Drivers\STM32L4xx_HAL_Driver folder.
- They can be easily recognizable by their naming convention: stm32l4_ll_<peripheral_name>

For more details on HAL and LL drivers refer to the STM32L4 HAL and Low-layer drivers user manual (UM1884).

As the decision to use LL or HAL drivers is made on a peripheral basis, the user can mix both HAL and LL drivers within the same project.

The following tables shows the main differences between the three possible STM32CubeMX project generation options: HAL-only, LL-only, and mix of HAL and LL code.

### Table 15. LL versus HAL code generation: drivers included in STM32CubeMX projects

<table>
<thead>
<tr>
<th>Project configuration and drivers to be included</th>
<th>HAL only</th>
<th>LL only</th>
<th>Mix of HAL and LL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>STM32xxx_HAL_Driver</td>
<td>Only HAL driver files</td>
<td>Only LL driver files</td>
<td>Mix of HAL and LL driver files</td>
<td>Only the driver files required for a given configuration (selection of peripherals) are copied when the project settings option is set to &quot;Copy only the necessary files&quot;. Otherwise (&quot;all used libraries&quot; option) the complete set of driver files is copied.</td>
</tr>
</tbody>
</table>

### Table 16. LL versus HAL code generation: STM32CubeMX generated header files

<table>
<thead>
<tr>
<th>Generated header files</th>
<th>HAL only</th>
<th>LL only</th>
<th>Mix of HAL and LL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.h</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>This file contains the include statements and the generated define statements for user constants (GPIO labels and user constants).</td>
</tr>
<tr>
<td>stm32xxx_hal_conf.h</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>This file enables the HAL modules necessary to the project.</td>
</tr>
<tr>
<td>stm32xxx_it.h</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Header file for interrupt handlers</td>
</tr>
<tr>
<td>stm32xx_assert.h</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>This file contains the assert macros and the functions used for checking function parameters.</td>
</tr>
</tbody>
</table>
### Table 17. LL versus HAL: STM32CubeMX generated source files

<table>
<thead>
<tr>
<th>Generated source files</th>
<th>HAL only</th>
<th>LL only</th>
<th>Mix of HAL and LL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| stm32xxx_hal_msp.c     | Yes      | No      | Yes               | This file contains the following functions:  
|                        |          |         |                   | – HAL_MspInit  
|                        |          |         |                   | – for peripherals using HAL drivers:  
|                        |          |         |                   |   HAL_<Peripheral>_MspInit,  
|                        |          |         |                   |   HAL_<Peripheral>_MspDeInit,  
|                        |          |         |                   | These functions are available only for the peripherals that use HAL drivers. |
| stm32xxx_it.c          | Yes      | Yes     | Yes               | Source file for interrupt handlers |

### Table 18. LL versus HAL: STM32CubeMX generated functions and function calls

<table>
<thead>
<tr>
<th>Generated source files</th>
<th>HAL only</th>
<th>LL only</th>
<th>Mix of HAL and LL</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Hal_init()             | Called in main.c | Not used | Called in main.c | This file performs the following functions:  
|                        |          |         |                   | – Configuration of Flash memory prefetch and instruction and data caches  
|                        |          |         |                   | – Selection of the SysTick timer as timebase source  
|                        |          |         |                   | – Setting of NVIC group priority  
|                        |          |         |                   | – MCU low-level initialization. |
| Hal_msp_init()         | Generated in stm32xxx_hal_msp.c and called by HAL_init() | Not used | Generated in stm32xxx_hal_msp.c And called by HAL_init() | This function performs the peripheral resources configuration(1). |
| MX_<Peripheral>_Init() | [1]: Peripheral configuration and call to HAL_<Peripheral>_Init() | [2]: Peripheral and peripheral resource configuration(1) using LL functions Call to LL_Peripheral_Init() | – When HAL driver is selected for the <Peripheral>, function generation and calls are done following [1]:  
|                        |          |         |                   | Peripheral configuration and call to  
|                        |          |         |                   | HAL_<Peripheral>_Init()  
|                        |          |         |                   | – When LL driver selected for the <Peripheral>, function generation and calls are done following [2]:  
|                        |          |         |                   | Peripheral and peripheral resource configuration using LL functions |
|                        |          |         |                   | This file takes care of the peripherals configuration.  
|                        |          |         |                   | When the LL driver is selected for the <Peripheral>, it also performs the peripheral resources configuration(1). |
### Table 18. LL versus HAL: STM32CubeMX generated functions and function calls (continued)

<table>
<thead>
<tr>
<th>Generated source files</th>
<th>HAL only</th>
<th>LL only</th>
<th>Mix of HAL and LL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_&lt;Peripheral&gt;_MspInit()</td>
<td>[3]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the &lt;Peripheral&gt;</td>
<td>Not used</td>
<td>Only HAL driver can be selected for the &lt;Peripheral&gt;: function generation and calls are done following [3]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the &lt;Peripheral&gt;</td>
<td>Peripheral resources configuration(1)</td>
</tr>
<tr>
<td>HAL_&lt;Peripheral&gt;_MspDeInit()</td>
<td>[4]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the &lt;Peripheral&gt;</td>
<td>Not used</td>
<td>Only HAL driver can be selected for the &lt;Peripheral&gt;: function generation and calls are done following [4]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the &lt;Peripheral&gt;</td>
<td>This function can be used to free peripheral resources.</td>
</tr>
</tbody>
</table>

1. Peripheral resources include:
   - peripheral clock
   - pinout configuration (GPIOs)
   - peripheral DMA requests
   - peripheral interrupt requests and priorities.
Figure 167. HAL-based peripheral initialization: usart.c code snippet

```c
// Peripheral Configuration
void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_7B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    ...  
    if (HAL_UART_Init(&huart1) != HAL_OK)
    {  
        Error_Handler();
    }
}

// Peripheral Resources Configuration
void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    if(uartHandle->Instance==USART1)
    {
        /* Peripheral clock enable */
        __HAL_RCC_USART1_CLK_ENABLE();
        /* USART1 GPIO Configuration */
        GPIO_InitStruct.Pin = GPIO_PIN_10;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    }
}

// Peripheral Resources Release
void HAL_UART_MspDeInit(UART_HandleTypeDef* uartHandle)
{
    if(uartHandle->Instance==USART1)
    {
        /* Peripheral clock disable */
        __HAL_RCC_USART1_CLK_DISABLE();
        /* USART1 GPIO Configuration */
        HAL_GPIO_DeInit(GPIOA, GPIO_PIN_10);
        HAL_GPIO_DeInit(GPIOB, GPIO_PIN_6);
    }
}
```
Figure 168. LL-based peripheral initialization: USART.c code snippet

```c
void MX_USART1_UART_Init(void)
{
    LL_USART_InitTypeDef USART_InitStruct;
    LL_GPIO_InitTypeDef GPIO_InitStruct;
    /* Peripheral clock enable */
    LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_USART1);

    /* USART1 GPIO Configuration */
    PA10 -------> USART1_RX
    PB6 -------> USART1_TX

    GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQVERYHIGH;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
    LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = LL_GPIO_PIN_6;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQVERYHIGH;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
    LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    USART_InitStruct.BaudRate = 115200;
    USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_7B;
    USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
    USART_InitStruct.Parity = LL_USART_PARITY_NONE;
    USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
    USART_InitStruct.HardwareFlowControl = LL_USART_HWFLOWCONTROL_NONE;
    USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;

    LL_USART_Init(USART1, &USART_InitStruct);
    LL_USART_ConfigAsyncMode(USART1);
}
```

Figure 169. HAL versus LL: main.c code snippet

```c
/* Includes */
#include "main.h"
#include "stm32f4xx_hal.h"
#include "usart.h"
#include "gpio.h"

void SystemClock_Config(void);
void Error_Handler(void);

int main(void)
{
    /* Reset of all peripherals, initializes the Flash Interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART1_UART_Init();
}

/* Includes */
#include "main.h"
#include "usart.h"
#include "gpio.h"

void SystemClock_Config(void);
void Error_Handler(void);

int main(void)
{
    /* Reset of all peripherals, initializes the Flash Interface and the Systick. */
    LL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART1_UART_Init();
}
```
6.3 **Custom code generation**

STM32CubeMX supports custom code generation by means of a FreeMarker template engine (see http://www.freemarker.org).

6.3.1 **STM32CubeMX data model for FreeMarker user templates**

STM32CubeMX can generate a custom code based on a FreeMarker template file (.ftl extension) for any of the following MCU configuration information:

- List of MCU peripherals used by the user configuration
- List of parameters values for those peripherals
- List of resources used by these peripherals: GPIO, DMA requests and interrupts.

The user template file must be compatible with STM32CubeMX data model. This means that the template must start with the following lines:

```ftl
[#ftl]
[#list configs as dt]
[#assign data = dt]
[#assign peripheralParams = dt.peripheralParams]
[#assign peripheralGPIOParams = dt.peripheralGPIOParams]
[#assign usedIPs = dt.usedIPs]
[/#list]
```

A sample template file is provided for guidance (see Figure 170).

STM32CubeMX will also generate user-specific code if any is available within the template. As shown in the below example, when the sample template is used, the ftl commands are provided as comments next to the data they have generated:

FreeMarker command in template:

```
${peripheralParams.get("RCC").get("LSI_VALUE")}
```

Resulting generated code:

```
LSI_VALUE : 32000 [peripheralParams.get("RCC").get("LSI_VALUE")]
```

Figure 170. extra_templates folder - Default content
6.3.2 Saving and selecting user templates

The user can either place the FreeMarker template files under STM32CubeMX installation path within the db/extra_templates folder or in any other folder.

Then for a given project, the user will select the template files relevant for its project via the Template Settings window accessible from the Code Generator Tab in the Project Manager view menu (see Section 4.8).

6.3.3 Custom code generation

To generate custom code, the user must place the FreeMarker template file under STM32CubeMX installation path within the db/extra_templates folder (see Figure 171).

The template filename must follow the naming convention <user filename>_<file extension>.ftl in order to generate the corresponding custom file as <user filename>_<file extension>.

By default, the custom file is generated in the user project root folder, next to the .ioc file (see Figure 172).

To generate the custom code in a different folder, the user shall match the destination folder tree structure in the extra_templates folder (see Figure 173).

Figure 171. extra_templates folder with user templates
Figure 172. Project root folder with corresponding custom generated files

Figure 173. User custom folder for templates
6.4 Additional settings for C project generation

STM32CubeMX allows specifying additional project settings through the .extSettings file. This file must be placed in the same project folder and at the same level as the .ioc file.

As an example, additional settings can be used when external tools call STM32CubeMX to generate the project and require specific project settings.

Possible entries and syntax

All entries are optional. They are organized under the followings three categories: ProjectFiles, Groups or Others.

- **[ProjectFiles]**: section where to specify additional include directories
  
  Syntax
  
  HeaderPath = <include directory 1 path>;< include directory 2 path >

  Example
  
  HeaderPath=../../IIR_Filter_int32/Inc;

- **[Groups]**: section where to create new groups of files and/or add files to a group
  
  Syntax
  
  <Group name> = <file pathname1>;< file pathname2>

  Example
  
  Doc=$ PROJ_DIR$\..eadme.txt
  LibC:\libraries\mylib1.lib; C:\libraries\mylib2.lib;
  Drivers/BSP/MyRefBoard = C:\MyRefBoard\BSP\board_init.c;
  C:\MyRefBoard\BSP\board_init.h;

- **[Others]** section where to enable HAL modules and/or specify preprocessor define statements
  
  – Enabling pre-processor define statements
  
  Preprocessor define statements can be specified using the following syntax after the [Others] line:

  Syntax
  
  Define = <define1_name>;<define2_name>

  Example
Define = USE_STM32F429I_DISCO

- Enabling HAL modules in generated stm32f4xx_hal_conf.h

HAL modules can be enabled using the following syntax after the [Others] line:

**Syntax**

```
HALModule = <ModuleName1>; <ModuleName1>
```

**Example**

```
HALModule=I2S;I2C
```

**.extSettings file example and generated outcomes**

For the purpose of the example, a new project is created by selecting the STM32F429I-DISCO board from STM32CubeMX board selector. The EWARM toolchain is selected in the Project tab of the Project Manager view. The project is saved as `MyF429IDiscoProject`. In the project folder, next to the generated .ioc file, a .extSettings text file is placed with the following contents:

**[Groups]**

```
[Groups]

Drivers/BSP/STM32F429IDISCO=C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Drivers\BSP\STM32F429I-Discovery\stm32f429i_discovery.c;
C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Drivers\BSP\STM32F429I-Discovery\stm32f429i_discovery.h
Lib=C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Middlewares\Third_Party\FreeRTOS\Source\portable\IAR\ARM_CM4F\portasm.s
Doc=$PROJ_DIR$\..\..\readme.txt
```

**[Others]**

```
[Others]

Define = USE_STM32F429I_DISCO
HALModule = UART;SPI
```

Upon project generation, the presence of this .extSettings file triggers the update of:

- the project MyF429IDiscoProject.ewp file in EWARM folder (see *Figure 175*),
- the stm32f4xx_hal_conf.h file in the project Inc folder (see *Figure 176*)
- the project view within EWARM user interface as shown in *Figure 177* and *Figure 178*. 
Figure 175. Update of the project .ewp file (EWARM IDE) for preprocessor define statements

```xml
<settings>
  <name>IOC1ARM</name>
  <archiveVersion>2</archiveVersion>
  <data>
    <version>28</version>
    <wantNonLocal>1</wantNonLocal>
    <debug>1</debug>
    <option>
      <name>CCDefines</name>
      <state>USE_HAL_DRIVER</state>
      <state>STM32F429xx</state>
      <state>USE_STM32F429I_DISCO</state>
    </option>
  </data>
</settings>
```

Figure 176. Update of stm32f4xx_hal_conf.h file to enable selected modules

```c
/*
 * define HAL_RTC_MODULE_ENABLED */
/*
 * define HAL_RCC_MODULE_ENABLED */
/*
 * define HAL_SPI_MODULE_ENABLED */
/*
 * define HAL_I2C_MODULE_ENABLED */
/*
 * define HAL_DMA_MODULE_ENABLED */
/*
 * define HAL_TIM_MODULE_ENABLED */
/*
 * define HAL_UART_MODULE_ENABLED */
/*
 * define HAL_USART_MODULE_ENABLED */
/*
 * define HAL_WDT_MODULE_ENABLED */
/*
 * define HAL_CRC_MODULE_ENABLED */
/*
 * define HAL_TDC_MODULE_ENABLED */
```

Figure 177. New groups and new files added to groups in EWARM IDE
Figure 178. Preprocessor define statements in EWARM IDE
For working with Arm Cortex-M dual-core products, STM32CubeMX generates code for both cores automatically according to the context assignment and initializer choices made in the user interface (see Section 4.6: Pinout & Configuration view for STM32H7 dual-core product lines for details).

Figure 179. Code generation for STM32H7 dual-core devices

Generated initialization code

The code is generated in CM4, CM7 and Common folders. The Common folder holds the system_stm32h7xx.c, that contains the clock tree settings.

When a peripheral or middleware is assigned to both contexts, the function MX_<name>_init will be generated for both contexts but will be called only from the initializer side.
Generated startup and linker files

Each configuration (\_M4 or \_M7) of the project shall come with a startup file and a linker file, each suffixed with \_M4 or \_M7 respectively.

**Figure 180. Startup and linker files for STM32H7 dual-core devices**

Generated boot mode code

STM32CubeMX supports only one mode of boot for now, where both ARM Cortex-M cores boot at once.

The other boot modes will be introduced later as a project option in the project manager view:
- Arm Cortex-M7 core booting, Arm Cortex-M4 gated
- Arm Cortex-M4 core booting, Arm Cortex-M7 gated
- A first core booting executing from flash, loads the second core code to the SRAM then enables the second core to boot.

STM32CubeMX uses template files delivered with STM32CubeH7 MCU packages as reference.
8 Device tree generation (STM32MP1 Series only)

The Device tree in Linux is used to provide a way to describe non-discoverable hardware. STMicroelectronics is widely using the device tree for all the platform configuration data, including DDR configuration.

Linux developers can manually edit device tree source files (dts) but as an alternative STM32CubeMX offers a partial device-trees generation service to reduce effort and to ease new comers hands-on. STM32CubeMX intends to generate partially device trees corresponding to board level configuration. Partial means that the entire (board level) device-trees are not generated but only main sections that usually implies huge efforts and can cause compilation errors and dysfunction:

- Folders structure and files to folders distribution
- dtsi and headers inclusions
- pinCtrl and clocks generation
- System-On-Chip device nodes positioning
- Multi-core related configurations (Etzpc binding, resources manager binding, peripherals assignment)

8.1 Device tree overview

To run properly, any piece of software needs to get the hardware description of the platform on which it is executed, including the kind of CPU, the memory size and the pin configuration. Current Linux kernels and U-boot have put such non-discoverable hardware description in a separate binary, the device tree blob (dtb). The device tree blob is compiled from the device tree source files (dts) using the dtc compiler provided with the OpenSTLinux distribution.

The device tree structure consist of a board level file (.dts) that includes two device tree source include files (.dtsi): a soc level file and a –pinctrl file, that lists the pin muxing configurations.

The device tree structure is very close to C language multiple level structures with the “root” (/) being the highest level then “peripherals” being sub-nodes described further in the hierarchy (see figures 181, 182 and 183).

STM32CubeMX generation uses widely overloading mechanisms to complete or change some SOC devices definitions when user configurations require it.
Figure 181. STM32CubeMX generated DTS – Extract 1

System and Board information
model = "STMicroelectronics custom STM32CubeMX board";
compatible = "st,stm32mp157c-project2-mx", "st,stm32mp157t";

memory=0x00000000 {
...
};
/* USER CODE BEGIN root */
/* USER CODE END root */

User customization

Full clock configuration

clocks {
  clk_esi, clk-esi {
    clock-cells = <0>;
    compatible = "fixed-clock";
    clock-frequency = <320000>;
    u-boot, dm-pre-teloc;
  };
...;
}
/*root*/

Pin control configuration, including GPIO configuration
u-boot, dm-pre-teloc;
tim1_pins_max: tim1_mx-0 {
  pins {
    pinnmax = <STM32_PINMUX('A', 8, API)>, /* TIM1_CH1 */
    <STM32_PINMUX('A', 9, API)>, /* TIM1_CH2 */
    bias-disable;
    drive-push-pull;
    slew-rate = <0>;
  };
};

Figure 182. STM32CubeMX generated DTS – Extract 2

Multi-core management

m4_system_resources{
  status = "okay";
  /* USER CODE BEGIN m4_system_resources */
  /* USER CODE END m4_system_resources */
};

status = "okay";
/* USER CODE BEGIN m4_rproc */
/* USER CODE END m4_rproc */

Peripheral assignment to Cortex-M4 run time context
m4_rproc-names = "rproc_default", "rproc_sleep";
pinctrl-0 = <tim1_pins_max>;
pinctrl-1 = <tim1_sleep_pins_max>;
status = "okay";
/* USER CODE BEGIN m4_timers1 */
/* USER CODE END m4_timers1 */

For more information about STM32MP1 Series device tree specificities, refer to ST Wiki https://wiki.st.com/stm32mpu.

8.2 STM32CubeMX Device tree generation

For STM32MP1 Series, STM32CubeMX code generation feature has been extended to generate Device trees (DT) targeting the supported firmware:

- a single DT for configuring both TF-A and SP_min
- a DT for configuring U-Boot
- a DT for configuring Linux kernel

DTS generation is accessible through the same GENERATE CODE button.
The DT generation path can be configured from the Project Manager view, in the Advanced Settings tab, under OpenSTLinux Settings (see Figure 184). For each Device tree STM32CubeMX generates Device tree source (DTS) files.

**Figure 184. Project settings for configuring Device tree path**

The Device tree structure consists of:
- a complete clock-tree
- a complete pin control
- a complete multi-cores references definition
- a set of device nodes and sub-nodes
- user sections that can be filled to have complete and bootable Device trees (contents will not be lost at next generation).

The generated DTS files reflect the user configuration, such as the assignment of peripherals to runtime contexts and boot loaders, or clock tree settings.

STM32CubeMX DT generation ensures the coherency between the different DTs. Additionally, it generates the DDR configuration file as a part of the TF-A and U-Boot Device trees.

These files along with the files they include will be compiled to create the device tree blob for the targeted firmware.

### 8.2.1 Device tree generation for Linux kernel

STM32CubeMX only generates the “board” file for Linux. This file includes the “soc” file and the “pinctrl” file corresponding to the selected package.

The device tree nodes generated by STM32CubeMX can be completed by filling the user sections, following the device tree bindings available in the Linux kernel source code Documentation/devicetree/bindings/ folder.
8.2.2 Device tree generation for U-boot

STM32CubeMX does a copy of Linux dts file for U-Boot and completes it with two new files: one file for the “ddr” configuration and one file for U-Boot add-ons that mainly consists in using the “u-boot,dm-pre-reloc” property whenever needed.

The device tree nodes generated by STM32CubeMX can be completed by filling the user sections, following the device tree bindings available in U-Boot source code Documentation/devicetree/bindings/ folder.

Figure 186. STM32CubeMX Device tree generation for U-boot
8.2.3 Device tree generation for TF-A

STM32CubeMX generates a “board” dts file for TF-A that is a lighter version of the Linux “board” dts file, in order to save space. This file itself includes the already lighter dti files versions on “soc” and “pinctrl” sides, that comes with TF-A.

Then, the same “ddr” configuration file generated for U-Boot is reused for TF-A.

The device tree nodes generated by STM32CubeMX can be completed by filling the user sections, following the device tree bindings available in TF-A source code docs/devicetree/bindings/ folder.
The CMSIS-Pack standard describes a delivery mechanism for software components, device parameters, and evaluation board support.

The XML-based package description (pdsc) file describes the content of a software pack (file collection). It includes source code, header files, software libraries, documentation and source code templates. A software pack consists of the complete file collection along with the pdsc file, shipped in ZIP-format. After installing a software pack, all the included software components are available to the development tools.

A software component is a collection of source modules, header and configuration files as well as libraries. Packs containing software components can also include example projects and user code templates.

Refer to http://www.keil.com website for more details.

STM32CubeMX supports third-party and other STMicroelectronics embedded software solutions, delivered as software packs. STM32CubeMX enables to:

1. Install Software Packs and check for updates (see Section 3.4.4).
2. Select software components for the current project (see Section 4.12). Once this is done, the selected components appear in the tree view (see Figure 188).
3. Enable the software component from the tree view (see Figure 189). Use contextual help to get more details on the selection.
4. Configure software components (see Figure 189). This function is possible only for components coming with files in STM32CubeMX proprietary format.
5. Generate the C project for selected toolchains (see Figure 190).
   a) Software components files are automatically copied to the project.
   b) Software component configuration and initialization code are automatically generated. This function is possible only for components coming with files in STM32CubeMX proprietary format.

![Figure 188. Selecting a CMSIS-Pack software component](image)
Figure 189. Enabling and configuring a CMSIS-Pack software component
Figure 190. Project generated with CMSIS-Pack software component
10 Tutorial 1: From pinout to project C code generation using an MCU of the STM32F4 Series

This section describes the configuration and C code generation process. It takes as an example a simple LED toggling application running on the STM32F4DISCOVERY board.

10.1 Creating a new STM32CubeMX Project

1. Select File > New project from the main menu bar or New project from the Home page.
2. Select the MCU Selector tab and filter down the STM32 portfolio by selecting STM32F4 as ‘Series’, STM32F407 as ‘Lines’, and LQFP100 as ‘Package’ (see Figure 191).
3. Select the STM32F407VGTx from the MCU list and click OK.

STM32CubeMX views are then populated with the selected MCU database (Figure 192). Optionally, remove the MCUs Selection bottom window by deselecting Window> Outputs submenu (see Figure 193).
Tutorial 1: From pinout to project C code generation using an MCU of the STM32F4 Series

Figure 192. Pinout view with MCUs selection

Figure 193. Pinout view without MCUs selection window
10.2 Configuring the MCU pinout

For a detailed description of menus, advanced actions and conflict resolutions, refer to Section 4 and Appendix A.

1. By default, STM32CubeMX shows the Pinout view.
2. By default, [ ] Keep Current Signals Placement is unchecked allowing STM32CubeMX to move the peripheral functions around and to find the optimal pin allocation, that is the one that accommodates the maximum number of peripheral modes.

Since the MCU pin configurations must match the STM32F4DISCOVERY board, enable [ ] Keep Current Signals Placement for STM32CubeMX to maintain the peripheral function allocation (mapping) to a given pin.

This setting is saved as a user preference in order to be restored when reopening the tool or when loading another project.

3. Select the required peripherals and peripheral modes:
   a) Configure the GPIO to output the signal on the STM32F4DISCOVERY green LED by right-clicking PD12 from the Pinout view, then select GPIO_output:

   ![Figure 194. GPIO pin configuration](image)
b) Enable a timer to be used as timebase for toggling the LED. This is done by selecting Internal Clock as TIM3 clock source from the peripheral tree (see Figure 195).

Figure 195. Timer configuration
c) You can also configure the RCC to use an external oscillator as potential clock source (see Figure 196).

![Figure 196. Simple pinout configuration](image)

This completes the pinout configuration for this example.

**Note:** Starting with STM32CubeMX 4.2, the user can skip the pinout configuration by directly loading ST Discovery board configuration from the **Board selector** tab.
10.3 Saving the project

1. Click \(\) to save the project.
   When saving for the first time, select a destination folder and filename for the project. The .ioc extension is added automatically to indicate this is an STM32CubeMX configuration file.

   **Figure 197. Save Project As window**

2. Click \(\) to save the project under a different name or location.
10.4 Generating the report

Reports can be generated at any time during the configuration:

1. Click \( \text{Generate Report} \) to generate .pdf and .txt reports.

   If a project file has not been created yet, a warning prompts the user to save the project first and requests a project name and a destination folder (see Figure 198). An .ioc file is then generated for the project along with a .pdf and .txt reports with the same name.

   **Figure 198. Generate Project Report - New project creation**

   ![Generate Project Report - New project creation](image)

   Answering No will require to provide a name and location for the report only.

   As shown in Figure 199, a confirmation message is displayed when the operation is successful.

   **Figure 199. Generate Project Report - Project successfully created**

   ![Generate Project Report - Project successfully created](image)

2. Open the .pdf report using Adobe Reader or the .txt report using your favorite text editor. The reports summarize all the settings and MCU configuration performed for the project.

10.5 Configuring the MCU clock tree

The following sequence describes how to configure the clocks required by the application based on an STM32F4 MCU.

STM32CubeMX automatically generates the system, CPU and AHB/APB bus frequencies from the clock sources and prescalers selected by the user. Wrong settings are detected
and highlighted in fuchsia through a dynamic validation of minimum and maximum conditions. Useful tooltips provide a detailed description of the actions to undertake when the settings are unavailable or wrong. User frequency selection can influence some peripheral parameters (e.g. UART baud rate limitation).

STM32CubeMX uses the clock settings defined in the Clock tree view to generate the initialization C code for each peripheral clock. Clock settings are performed in the generated C code as part of RCC initialization within the project main.c and in stm32f4xx_hal_conf.h (HSE, HSI and external clock values expressed in Hertz).

Follow the sequence below to configure the MCU clock tree:

1. Click the **Clock Configuration** tab to display the clock tree (see Figure 200).

   The internal (HSI, LSI), system (SYSCLK) and peripheral clock frequency fields cannot be edited. The system and peripheral clocks can be adjusted by selecting a clock source, and optionally by using the PLL, prescalers and multipliers.

   ![Figure 200. Clock tree view](image_url)
2. First select the clock source (HSE, HSI or PLLCLK) that will drive the system clock of the microcontroller.

In the example taken for the tutorial, select HSI to use the internal 16 MHz clock (see Figure 201).

**Figure 201. HSI clock enabled**

![HSI clock enabled](image)

To use an external clock source (HSE or LSE), the RCC peripheral shall be configured in the Pinout view since pins will be used to connect the external clock crystals (see Figure 202).

**Figure 202. HSE clock source disabled**

![HSE clock source disabled](image)

Other clock configuration options for the STM32F4DISCOVERY board:

- Select the external HSE source and enter 8 in the HSE input frequency box since an 8 MHz crystal is connected on the discovery board:

**Figure 203. HSE clock source enabled**

![HSE clock source enabled](image)

- Select the external PLL clock source and the HSI or HSE as the PLL input clock source.

**Figure 204. External PLL clock source enabled**

![External PLL clock source enabled](image)
3. Keep the core and peripheral clocks to 16 MHz using HSI, no PLL and no prescaling.

**Note:** Optionally, further adjust the system and peripheral clocks using PLL, prescalers and multipliers:

Other clock sources independent from the system clock can be configured as follows:

- USB OTG FS, Random Number Generator and SDIO clocks are driven by an independent output of the PLL.
- I2S peripherals come with their own internal clock (PLLI2S), alternatively derived by an independent external clock source.
- USB OTG HS and Ethernet Clocks are derived from an external source.

4. Optionally, configure the prescaler for the Microcontroller Clock Output (MCO) pins that allow to output two clocks to the external circuit.

5. Click to save the project.

6. Go to the **Configuration** tab to proceed with the project configuration.

### 10.6 Configuring the MCU initialization parameters

**Caution:** The C code generated by STM32CubeMX covers the initialization of the MCU peripherals and middlewares using the STM32Cube firmware libraries.

#### 10.6.1 Initial conditions

From the **Pinout & Configuration** tab, select and configure (one by one) every component (peripheral, middleware, additional software) required by the application using the **Mode** and **Configuration** panels (see **Figure 205**).

Tooltips and warning messages are displayed when peripherals are not properly configured (see **Section 4: STM32CubeMX user interface** for details).

**Note:** The **RCC** peripheral initialization will use the parameter configuration done in this view as well as the configuration done in the **Clock tree** view (clock source, frequencies, prescaler values, etc…).
10.6.2 Configuring the peripherals

Each peripheral instance corresponds to a dedicated button in the main panel. Some peripheral modes have no configurable parameters, as illustrated below.

Figure 206. Case of Peripheral and Middleware without configuration parameters
Follow the steps below to proceed with peripheral configuration:

1. Click the peripheral button to open the corresponding configuration window.
   In our example
   a) click TIM3 to open the timer configuration window.

   **Figure 207. Timer 3 configuration window**

   ![Timer 3 configuration window](image)

   b) with a 16 MHz APB clock (Clock tree view), set the prescaler to 16000 and the counter period to 1000 to make the LED blink every millisecond.

   **Figure 208. Timer 3 configuration**

   ![Timer 3 configuration](image)
2. Optionally, and when available, select:
   - The **NVIC Settings** tab to display the NVIC configuration and enable interruptions for this peripheral.
   - The **DMA Settings** tab to display the DMA configuration and to configure DMA transfers for this peripheral.

   In the tutorial example, the DMA is not used and the GPIO settings remain unchanged. The interrupt is enabled, as shown in Figure 209.
   - The **GPIO Settings** tab to display the GPIO configuration and to configure the GPIOs for this peripheral.
   - Insert an item:
     - The **User Constants** tab to specify constants to be used in the project.

**Figure 209. Enabling Timer 3 interrupt**

![Figure 209. Enabling Timer 3 interrupt](image)

### 10.6.3 Configuring the GPIOs

The user can adjust all pin configurations from this window. A small icon along with a tooltip indicates the configuration status.

**Figure 210. GPIO configuration color scheme and tooltip**

![Figure 210. GPIO configuration color scheme and tooltip](image)
Follow the sequence below to configure the GPIOs:

1. Click the **GPIO button** in the Configuration view to open the **Pin Configuration** window below.

2. The first tab shows the pins that have been assigned a GPIO mode but not for a dedicated peripheral and middleware. Select a Pin Name to open the configuration for that pin.

   In the tutorial example, select PD12 and configure it in output push-pull mode to drive the STM32F4DISCOVERY LED (see **Figure 211**).

![Figure 211. GPIO mode configuration](image-url)
### 10.6.4 Configuring the DMAs

This is not required for this example. It is recommended to use DMA transfers to offload the CPU. The DMA Configuration window provides a fast and easy way to configure the DMAs (see Figure 212):

1. add a new DMA request and select among a list of possible configurations.
2. select among the available streams.
3. select the Direction: Memory to Peripheral or Peripheral to Memory.
4. select a Priority.
5. enable the FIFO.

**Note:** Configuring the DMA for a given peripheral and middleware can also be performed using the Peripheral and Middleware configuration window.

**Figure 212. DMA parameters configuration window**
10.6.5 Configuring the middleware

This is not required for the example taken for the tutorial.

If a peripheral is required for a middleware mode, the peripheral must be configured in the Pinout view for the middleware mode to become available. A tooltip can guide the user as shown below.

1. Configure the USB peripheral from the Pinout view.

2. Select MSC_FS class from USB Host middleware.

3. Select the checkbox to enable FatFs USB mode in the tree panel.
Figure 215. FatFs over USB mode enabled
4. Select the **Configuration** view. FatFs and USB buttons are then displayed.

**Figure 216. System view with FatFs and USB enabled**
5. FatFs and USB using default settings are already marked as configured. Click FatFs and USB buttons to display default configuration settings. You can also change them by following the guidelines provided at the bottom of the window.

Figure 217. FatFs define statements
10.7 Generating a complete C project

10.7.1 Setting project options

Default project settings can be adjusted prior to C code generation as shown in Figure 218.

1. Select the Project Manager view to update project settings and generation options.
2. Select the Project Tab and choose a Project name, location and a toolchain to generate the project (see Figure 218).

Figure 218. Project Settings and toolchain selection

3. Select the Code Generator tab to choose various C code generation options:
   - The library files copied to Projects folder.
   - C code regeneration (e.g. what is kept or backed up during C code regeneration).
   - HAL specific action (e.g. set all free pins as analog I/Os to reduce MCU power consumption).

In the tutorial example, select the settings as displayed in Figure 219 and click OK.

Note: A dialog window appears when the firmware package is missing. Go to next section for explanation on how to download the firmware package.
10.7.2 Downloading firmware package and generating the C code

1. Click **GENERATE CODE** to generate the C code.

   During C code generation, STM32CubeMX copies files from the relevant STM32Cube MCU package into the project folder so that the project can be compiled. When generating a project for the first time, the firmware package is not available on the user PC and a warning message is displayed:

   **Figure 220. Missing firmware package warning message**

2. STM32CubeMX offers to download the relevant firmware package or to go on. Click **Download** to obtain a complete project, that is a project ready to be used in the selected IDE.

   By clicking **Continue**, only **Inc** and **Src** folders will be created, holding STM32CubeMX generated initialization files. The necessary firmware and middleware libraries will have to be copied manually to obtain a complete project.
If the download fails, an error message is displayed.

Figure 221. Error during download

To solve this issue, execute the next two steps. Skip them otherwise.

3. Select Help > Updater settings menu and adjust the connection parameters to match your network configuration.

Figure 222. Updater settings for download

4. Click Check connection. The check mark turns green once the connection is established.
5. Once the connection is functional, click **GENERATE CODE** to generate the C code. The C code generation process starts and progress is displayed (see next figures).

![Updater settings with connection](image1)

![Downloading the firmware package](image2)
6. Finally, a confirmation message is displayed to indicate that the C code generation has been successful.

**Figure 226. C code generation completion message**

![C code generation completion message](image)
7. Click **Open Folder** to display the generated project contents or click **Open Project** to open the project directly in your IDE. Then proceed with **Section 10.8**.

**Figure 227. C code generation output folder**

The generated project contains:

- The STM32CubeMX .ioc project file located in the root folder. It contains the project user configuration and settings generated through STM32CubeMX user interface.
- The **Drivers** and **Middlewares** folders hold copies of the firmware package files relevant for the user configuration.
- The **Projects** folder contains IDE specific folders with all the files required for the project development and debug within the IDE.
- The **Inc** and **Src** folders contain STM32CubeMX generated files for middleware, peripheral and GPIO initialization, including the main.c file. The STM32CubeMX generated files contain user-dedicated sections allowing to insert user-defined C code.

**Caution:** C code written within the user sections is preserved at next C code generation, while C code written outside these sections is overwritten.

User C code will be lost if user sections are moved or if user sections delimiters are renamed.
10.8 **Building and updating the C code project**

This example explains how to use the generated initialization C code and complete the project, within IAR™ EWARM toolchain, to have the LED blink according to the TIM3 frequency.

A folder is available for the toolchains selected for C code generation: the project can be generated for more than one toolchain by choosing a different toolchain from the Project Manager menu and clicking Generate code once again.

1. Open the project directly in the IDE toolchain by clicking Open Project from the dialog window or by double-clicking the relevant IDE file available in the toolchain folder under STM32CubeMX generated project directory (see Figure 226).

![Figure 228. C code generation output: Projects folder](image)
2. As an example, select .eww file to load the project in the IAR™ EWARM IDE.

Figure 229. C code generation for EWARM
3. Select the main.c file to open in editor.

**Figure 230. STM32CubeMX generated project open in IAR™ IDE**

The htim3 structure handler, system clock, GPIO and TIM3 initialization functions are defined. The initialization functions are called in the main.c. For now the user C code sections are empty.
4. In the IAR™ IDE, right-click the project name and select **Options**.

**Figure 231. IAR™ options**

5. Click the ST-LINK category and make sure SWD is selected to communicate with the STM32F4DISCOVERY board. Click **OK**.

**Figure 232. SWD connection**

6. Select **Project > Rebuild all**. Check if the project building has succeeded.

**Figure 233. Project building log**

<table>
<thead>
<tr>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>stm32H7xx_hal_tim.c</td>
</tr>
<tr>
<td>stm32H7xx_hal_tim_ex.c</td>
</tr>
<tr>
<td>stm32H7xx_it.c</td>
</tr>
<tr>
<td>stm32H7xx_it.c</td>
</tr>
<tr>
<td>system_stm32H7xx.c</td>
</tr>
</tbody>
</table>

**Total number of errors:** 0

**Total number of warnings:** 0
7. Add user C code in the dedicated user sections only.

Note: The main while(1) loop is placed in a user section.

For example:

a) Edit the main.c file.

b) To start timer 3, update User Section 2 with the following C code:

```c
HAL_Init();
/* Configure the system clock */
SystemClock_Config();
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM3_Init();

/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(htim3);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
}
```

c) Then, add the following C code in User Section 4:

```c
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if ( htim->Instance == htim3.Instance )
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
  }
}
/* USER CODE END 4 */
```

This C code implements the weak callback function defined in the HAL timer driver (stm32f4xx_hal_tim.h) to toggle the GPIO pin driving the green LED when the timer counter period has elapsed.

8. Rebuild and program your board using 

Make sure the SWD ST-LINK option is checked as a Project options otherwise board programming will fail.

9. Launch the program using 

The green LED on the STM32F4DISCOVERY board will blink every second.

10. To change the MCU configuration, go back to STM32CubeMX user interface, implement the changes and regenerate the C code. The project will be updated, preserving the C code in the user sections if Keep User Code when re-generating option in Project Manager’s Code Generator tab is enabled.
10.9 Switching to another MCU

STM32CubeMX allows loading a project configuration on an MCU of the same Series.

Proceed as follows:

1. Select File > New Project.

2. Select an MCU belonging to the same Series. As an example, you can select the STM32F429ZITx that is the core MCU of the 32F429IDISCOVERY board.

3. Select File > Import project. In the Import project window, browse to the .ioc file to load. A message warns you that the currently selected MCU (STM32F429ZITx) differs from the one specified in the .ioc file (STM32F407VGTx). Several import options are proposed (see Figure 236).

4. Click the Try Import button and check the import status to verify if the import has been successful.

5. Click OK to really import the project. An output tab is then displayed to report the import results.

6. The green LED on 32F429IDISCOVERY board is connected to PG13: CTRL+ right click PD12 and drag and drop it on PG13.

7. From Project Manager project tab configure the new project name and folder location. Click Generate icon to save the project and generate the code.

8. Select Open the project from the dialog window, update the user sections with the user code, making sure to update the GPIO settings for PG13. Build the project and flash the board. Launch the program and check that LED blinks once per second.

Figure 236. Import Project menu
11 Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board

The tutorial consists in creating and writing to a file on the STM32429I-EVAL1 SD card using the FatFs file system middleware.

To generate a project and run tutorial 2, follow the sequence below:

1. Launch STM32CubeMX.
3. Click the Board Selector Tab to display the list of ST boards.
4. Select EvalBoard as type of Board and STM32F4 as Series to filter down the list.
5. Answer Yes to Initialize all peripherals with their default mode so that the code is generated only for the peripherals used by the application.
6. Select the STM32429I-EVAL board and click OK. Answer No in the dialog box asking to initialize all peripherals to their default modes (see Figure 237). The Pinout view is loaded, matching the MCU pinout configuration on the evaluation board (see Figure 238).

Figure 237. Board peripheral initialization dialog box
From the Peripheral tree on the left, expand the SDIO peripheral and select the SD 4 bits wide bus (see Figure 239).
8. Under the Middlewares category, check **SD Card** as FatFs mode (see *Figure 240*).

*Figure 240. FatFs mode configuration*

![FatFs mode configuration](image)

9. Configure the clocks as follows:
   a) Select the RCC peripheral from the **Pinout** view (see *Figure 241*).

*Figure 241. RCC peripheral configuration*

![RCC peripheral configuration](image)

   b) Configure the clock tree from the clock tab (see *Figure 242*).

*Figure 242. Clock tree view*
10. In the Project tab, specify the project name and destination folder. Then, select the EWARM IDE toolchain.

Figure 243. FATFS tutorial - Project settings

11. Click Ok. Then, on the toolbar menu, click GENERATE CODE to generate the project.

12. Upon code generation completion, click Open Project in the Code Generation dialog window (see Figure 244). This opens the project directly in the IDE.

Figure 244. C code generation completion message
13. In the IDE, check that heap and stack sizes are sufficient: right click the project name and select **Options**, then select **Linker**. Check **Override default** to use the icf file from STM32CubeMX generated project folder. If not already done through CubeMX User interface (under Linker Settings from Project Manager’s project tab), adjust the heap and stack sizes (see **Figure 245**).

![Figure 245. IDE workspace](image)

**Note:** When using the MDK-Arm toolchain, go to the Application/MDK-ARM folder and double-click the `startup_xx.s` file to edit and adjust the heap and stack sizes there.

15. The tutorial consists in creating and writing to a file on the evaluation board SD card using the FatFs file system middleware:
   a) At startup all LEDs are OFF.
   b) The red LED is turned ON to indicate that an error occurred (FatFs initialization, file read/write access errors..).
   c) The orange LED is turned ON to indicate that the FatFs link has been successfully mounted on the SD driver.
   d) The blue LED is turned ON to indicate that the file has been successfully written to the SD Card.
   e) The green LED is turned ON to indicate that the file has been successfully read from file the SD Card.

16. For use case implementation, update main.c with the following code:
   a) Insert main.c private variables in a dedicated user code section:

   ```c
   /* USER CODE BEGIN PV */
   /* Private variables ------------------------------------------*/
   FATFS SDFatFs; /* File system object for SD card logical drive */
   FIL MyFile; /* File object */
   const char wtext[] = "Hello World!";
   const uint8_t image1_bmp[] = {
       0x42,0x4d,0x36,0x84,0x03,0x00,0x00,0x00,0x00,0x00,0x36,0x00,0x00,0x00,0x00,
       0x28,0x00,0x00,0x00,0x4d,0x01,0x00,0x00,0x00,0x01,0x00,0x00,0x01,0x00,
       0x13,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   };
   /* USER CODE END PV */
   
   b) Insert main functional local variables:

   ```
   int main(void)
   {
   /* USER CODE BEGIN 1 */
   FRESULT res; /* FatFs function common result code */
   uint32_t byteswritten, bytesread; /* File write/read counts */
   char rtext[256]; /* File read buffer */
   /* USER CODE END 1 */

   /* MCU Configuration----------------------------------------*/
   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
   HAL_Init();
   
   c) Insert user code in the main function, after initialization calls and before the while loop, to perform actual read/write from/to the SD card:

   ```
   int main(void)
   {
MX_FATFS_Init();

    /* USER CODE BEGIN 2 */
    /**@-0- Turn all LEDs off(red, green, orange and blue) */
    HAL_GPIO_WritePin(GPIOG, (GPIO_PIN_10 | GPIO_PIN_6 | GPIO_PIN_7 |
    GPIO_PIN_12), GPIO_PIN_SET);
    /**@-1- FatFs: Link the SD disk I/O driver */
    if(retSD == 0){
        /* success: set the orange LED on */
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET);
        /**@-2- Register the file system object to the FatFs module */
        if(f_mount(&SDFatFs, (TCHAR const*)SD_Path, 0) != FR_OK){
            /* FatFs Initialization Error : set the red LED on */
            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
            while(1);
        } else {
            /**@-3- Create a FAT file system (format) on the logical drive */
            if(f_mkfs((TCHAR const*)SD_Path, 0, 0) != FR_OK){
                /* FatFs Format Error : set the red LED on */
                HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
                while(1);
            } else {
                /**@-4- Create & Open a new text file object with write access */
                if(f_open(&MyFile, "Hello.txt", FA_CREATE_ALWAYS | FA_WRITE) !=
                    FR_OK){
                    /* 'Hello.txt' file Open for write Error : set the red LED on */
                    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
                    while(1);
                } else {
                    /**@-5- Write data to the text file */
                    res = f_write(&MyFile, wtext, sizeof(wtext), (void *)
                    &byteswritten);
                    if((byteswritten == 0) || (res != FR_OK)){
                        /* 'Hello.txt' file Write or EOF Error : set the red LED on */
                        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
                        while(1);
                    } else {
                        /**@-6- Successful open/write : set the blue LED on */
                        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);
                        f_close(&MyFile);
                    }
                }
            }
        }
    }
    /*@-7- Open the text file object with read access */
    if(f_open(&MyFile, "Hello.txt", FA_READ) != FR_OK){
        /* 'Hello.txt' file Open for read Error : set the red LED on */
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
        while(1);
    } else {
        /**@-8- Read data from the text file */
        res = f_read(&MyFile, rtext, sizeof(wtext), &bytesread);
    }
if((strcmp(rtext,wtext)!=0)|| (res != FR_OK)){
  /* 'Hello.txt' file Read or EOF Error : set the red LED on */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
  while(1);
} else {
  /* Successful read : set the green LED On */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
  /*##-9- Close the open text file *****************************/
  f_close(&MyFile);
  }}}))})
  /*##-10- Unlink the micro SD disk I/O driver *****************************/
  FATFS_UnLinkDriver(SD_Path);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
Tutorial 3 - Using the Power Consumption Calculator to optimize the embedded application consumption and more

12.1 Tutorial overview

This tutorial focuses on STM32CubeMX Power Consumption Calculator (Power Consumption Calculator) feature and its benefits to evaluate the impacts of power-saving techniques on a given application sequence.

The key considerations to reduce a given application power consumption are:

- Reducing the operating voltage
- Reducing the time spent in energy consuming modes
  It is up to the developer to select a configuration that gives the best compromise between low-power consumption and performance.
- Maximizing the time spent in non-active and low-power modes
- Using the optimal clock configuration
  The core should always operate at relatively good speed, since reducing the operating frequency can increase energy consumption if the microcontroller has to remain for a long time in an active operating mode to perform a given operation.
- Enabling only the peripherals relevant for the current application state and clock-gating the others
- When relevant, using the peripherals with low-power features (e.g. waking up the microcontroller with the I2C)
- Minimizing the number of state transitions
- Optimizing memory accesses during code execution
  - Prefer code execution from RAM to Flash memory
  - When relevant, consider aligning CPU frequency with Flash memory operating frequency for zero wait states.

The following tutorial shows how the STM32CubeMX Power Consumption Calculator feature can help to tune an application to minimize its power consumption and extend the battery life.

Note: The Power Consumption Calculator does not account for I/O dynamic current consumption and external board components that can also affect current consumption. For this purpose, an “additional consumption” field is provided for the user to specify such consumption value.
12.2 Application example description

The application is designed using the NUCLEO-L476RG board based on a STM32L476RGTx device and supplied by a 2.4 V battery.

The main purpose of this application is to perform ADC measurements and transfer the conversion results over UART. It uses:

- Multiple low-power modes: Low-power run, Low-power sleep, Sleep, Stop and Standby
- Multiple peripherals: USART, DMA, Timer, COMP, DAC and RTC
  - The RTC is used to run a calendar and to wake up the CPU from Standby when a specified time has elapsed.
  - The DMA transfers ADC measurements from ADC to memory
  - The USART is used in conjunction with the DMA to send/receive data via the virtual COM port and to wake up the CPU from Stop mode.

The process to optimize such complex application is to start describing first a functional only sequence then to introduce, on a step by step basis, the low-power features provided by the STM32L476RG microcontroller.

12.3 Using the Power Consumption Calculator

12.3.1 Creating a power sequence

Follow the steps below to create the sequence (see Figure 246):

1. Launch STM32CubeMX.
2. Click new project and select the Nucleo-L476RG board from the Board tab.
3. Click the Power Consumption Calculator tab to select the Power Consumption Calculator view. A first sequence is then created as a reference.
4. Adapt it to minimize the overall current consumption. To do this:
   a) Select 2.4 V $V_{DD}$ power supply. This value can be adjusted on a step by step basis (see Figure 247).
   b) Select the Li-MnO2 (CR2032) battery. This step is optional. The battery type can be changed later on (see Figure 247).
Tutorial 3 - Using the Power Consumption Calculator to optimize the embedded application con-

Figure 246. Power Consumption Calculation example

Figure 247. $V_{DD}$ and battery selection menu
5. Enable the Transition checker to ensure the sequence is valid (see Figure 247). This option allows verifying that the sequence respects the allowed transitions implemented within the STM32L476RG.

6. Click the Add button to add steps that match the sequence described in Figure 247.
   - By default the steps last 1 ms each, except for the wakeup transitions that are preset using the transition times specified in the product datasheet (see Figure 248).
   - Some peripherals for which consumption is unavailable or negligible are highlighted with "*" (see Figure 248).

7. Click the Save button to save the sequence as SequenceOne.

The application consumption profile is the generated. It shows that the overall sequence consumes an average of 2.01 mA for 9 ms, and the battery lifetime is only 4 days (see Figure 249).

12.3.2 Optimizing application power consumption

Let us now take several actions to optimize the overall consumption and the battery lifetime. These actions are performed on step 1, 4, 5, 6, 7, 8 and 10.
The next figures show on the left the original step and on the right the step updated with several optimization actions.

**Step 1 (Run)**

- **Findings**
  All peripherals are enabled although the application requires only the RTC.

- **Actions**
  - Lower the operating frequency.
  - Enable solely the RTC peripheral.
  - To reduce the average current consumption, reduce the time spent in this mode.

- **Results**
  The current is reduced from 9.05 mA to 2.16 mA (see Figure 250).

**Figure 250. Step 1 optimization**

**Step 4 (Run, RTC)**

- **Action**
  Reduce the time spent in this mode to 0.1 ms.
Step 5 (Run, ADC, DMA, RTC)

- **Actions**
  - Change to Low-power run mode.
  - Lower the operating frequency.

- **Results**
  The current consumption is reduced from 6.17 mA to 271 µA (see Figure 251).

**Figure 251. Step 5 optimization**
Step 6 (Sleep, DMA, ADC, RTC)

- **Actions**
  - Switch to Lower-power sleep mode (BAM mode)
  - Reduce the operating frequency to 2 MHz.

- **Results**
  
The current consumption is reduced from 703 µA to 93 µA (see Figure 252).

**Figure 252. Step 6 optimization**
Step 7 (Run, DMA, RTC, USART)

- **Actions**
  - Switch to Lower-power run mode.
  - Use the power-efficient LPUART peripheral.
  - Reduce the operating frequency to 1 MHz using the interpolation feature.

- **Results**
  The current consumption is reduced from 1.92 µA to 42 µA (see Figure 253).

![Figure 253. Step 7 optimization](image-url)
Step 8 (Stop 0, USART)

- **Actions**
  - Switch to Stop1 low-power mode.
  - Use the power-efficient LPUART peripheral.

- **Results**
  The current consumption is reduced (see Figure 254).

![Figure 254. Step 8 optimization](image-url)
Step 10 (RTC, USART)

- **Actions**
  - Use the power-efficient LPUART peripheral.
  - Reduce the operating frequency to 1 MHz.

- **Results**
  The current consumption is reduced from 1.89 mA to 234 µA (see Figure 255).
  The example given in Figure 256 shows an average current consumption reduction of 155 µA.

See Figure 256 for the sequence overall results: 7 ms duration, about 2 month battery life, and an average current consumption of 165.25 µA.

Use the compare button to compare the current results to the original ones saved as SequenceOne.pcs.

Figure 256. Power sequence results after optimizations
13 Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board

This tutorial aims at demonstrating how to use STM32CubeMX to create a UART serial communication application for a NUCLEO-L053R8 board.

A Windows PC is required for the example. The ST-Link USB connector is used both for serial data communications, and firmware downloading and debugging on the MCU. A Type-A to mini-B USB cable must be connected between the board and the computer. The USART2 peripheral uses PA2 and PA3 pins, which are wired to the ST-Link connector. In addition, USART2 is selected to communicate with the PC via the ST-Link Virtual COM Port. A serial communication client, such as Tera Term, needs to be installed on the PC to display the messages received from the board over the virtual communication Port.

13.1 Tutorial overview

Tutorial 4 will take you through the following steps:
1. Selection of the NUCLEO-L053R8 board from the New Project menu.
2. Selection of the required features (debug, USART, timer) from the Pinout view: peripheral operating modes as well as assignment of relevant signals on pins.
3. Configuration of the MCU clock tree from the Clock Configuration view.
4. Configuration of the peripheral parameters from the Configuration view
5. Configuration of the project settings in the Project Manager menu and generation of the project (initialization code only).
6. Project update with the user application code corresponding to the UART communication example.
7. Compilation, and execution of the project on the board.
8. Configuration of Tera Term software as serial communication client on the PC.
9. The results are displayed on the PC.

13.2 Creating a new STM32CubeMX project and selecting the Nucleo board

To do this, follow the sequence below:
1. Select File > New project from the main menu bar. This opens the New Project window.
2. Go to the Board selector tab and filter on STM32L0 Series.
3. Select NUCLEO-L053R8 and click OK to load the board within the STM32CubeMX user interface (see Figure 257).
Figure 257. Selecting NUCLEO_L053R8 board
13.3 Selecting the features from the Pinout view

1. Select Debug Serial Wire under SYS (see Figure 258).

Figure 258. Selecting debug pins

2. Select Internal Clock as clock source under TIM2 peripheral (see Figure 259).

Figure 259. Selecting TIM2 clock source
3. Select the Asynchronous mode for the USART2 peripheral (see Figure 260).

Figure 260. Selecting asynchronous mode for USART2

4. Check that the signals are properly assigned on pins (see Figure 261):
   - SYS_SWDIO on PA13
   - TCK on PA14
   - USART_TX on PA2
   - USART_RX on PA3

Figure 261. Checking pin assignment
13.4 Configuring the MCU clock tree from the Clock Configuration view

1. Go to the Clock Configuration tab and leave the configuration untouched, in order to use the MSI as input clock and an HCLK of 2.097 MHz (see Figure 262).

**Figure 262. Configuring the MCU clock tree**
13.5 Configuring the peripheral parameters from the Configuration view

1. From the Configuration tab, click USART2 to open the peripheral Parameter Settings window and set the baud rate to 9600. Make sure the Data direction is set to “Receive and Transmit” (see Figure 263).
2. Click OK to apply the changes and close the window.

Figure 263. Configuring USART2 parameters
3. Click **TIM2** and change the prescaler to 16000, the Word Length to 8 bits and the Counter Period to 1000 (see Figure 264).

Figure 264. Configuring TIM2 parameters
4. Enable TIM2 global interrupt from the **NVIC Settings** tab (see *Figure 265*).

*Figure 265. Enabling TIM2 interrupt*
13.6 Configuring the project settings and generating the project

1. In the **Project Settings** menu, specify the project name, destination folder, and select the EWARM IDE toolchain (see Figure 266).

![Figure 266. Project Settings menu](image)

If the firmware package version is not already available on the user PC, a progress window opens to show the firmware package download progress.
2. In the **Code Generator** tab, configure the code to be generated as shown in **Figure 267**, and click **OK** to generate the code.

![Figure 267. Generating the code](image)

### 13.7 Updating the project with the user application code

Add the user code as follows:

```c
/* USER CODE BEGIN 0 */
#include "stdio.h"
#include "string.h"
/* Buffer used for transmission and number of transmissions */
char aTxBuffer[1024];
int nbtime=1;
/* USER CODE END 0 */
```

Within the main function, start the timer event generation function as follows:

```c
/* USER CODE BEGIN 2 */
```
/* Start Timer event generation */
    HAL_TIM_Base_Start_IT(&htim2);
/* USER CODE END 2 */

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    sprintf(aTxBuffer,"STM32CubeMX rocks %d times \t", ++nbtime);
    HAL_UART_Transmit(&huart2,(uint8_t *) aTxBuffer, strlen(aTxBuffer), 5000);
}
/* USER CODE END 4 */

13.8 Compiling and running the project
1. Compile the project within your favorite IDE.
2. Download it to the board.
3. Run the program.

13.9 Configuring Tera Term software as serial communication client on the PC
1. On the computer, check the virtual communication port used by ST Microelectronics from the Device Manager window (see Figure 268).

Figure 268. Checking the communication port
2. To configure Tera Term to listen to the relevant virtual communication port, adjust the parameters to match the USART2 parameter configuration on the MCU (see Figure 269).

Figure 269. Setting Tera Term port parameters

3. The Tera Term window displays a message coming from the board at a period of a few seconds (see Figure 270).

Figure 270. Setting Tera Term port parameters
14 Tutorial 5: Exporting current project configuration to a compatible MCU

When List pinout compatible MCUs is selected from the Pinout menu, STM32CubeMX retrieves the list of the MCUs which are compatible with the current project configuration, and offers to export the current configuration to the newly selected compatible MCU.

This tutorial shows how to display the list of compatible MCUs and export your current project configuration to a compatible MCU:

1. Load an existing project, or create and save a new project:

![Figure 271. Existing or new project pinout](image)

2. Go to the Pinout menu and select List Pinout Compatible MCUs. The Pinout compatible window pops up (see Figure 272 and Figure 273).

If needed, modify the search criteria and the filter options and restart the search process by clicking the Search button.

The color shading and the Comments column indicate the level of matching:

- Exact match: the MCU is fully compatible with the current project (see Figure 273 for an example).
- Partial match with hardware compatibility: the hardware compatibility can be ensured but some pin names could not be preserved. Hover the mouse over the desired MCU to display an explanatory tooltip (see Figure 272 for an example).
Partial match without hardware compatibility: not all signals can be assigned to the exact same pin location and a remapping will be required. Hover the mouse over the desired MCU to display an explanatory tooltip (see Figure 273 for an example).

Figure 272. List of pinout compatible MCUs - Partial match with hardware compatibility

Figure 273. List of Pinout compatible MCUs - Exact and partial match
3. Then, select an MCU to import the current configuration to, and click OK, Import:

**Figure 274. Selecting a compatible MCU and importing the configuration**

The configuration is now available for the selected MCU:

**Figure 275. Configuration imported to the selected compatible MCU**
4. To see the list of compatible MCUs at any time, select **Outputs** under the **Window** menu. To load the current configuration to another compatible MCU, double-click the list of compatible MCUs.

5. To remove some constraints on the search criteria, several solutions are possible:
   - Select the **Ignore Pinning Status** checkbox to ignore pin status (locked pins).
   - Select the **Ignore Power Pins** checkbox not to take into account the power pins.
   - Select the **Ignore System Pins** not take into account the system pins. Hover the mouse over the checkbox to display a tooltip that lists the system pins available on the current MCU.
In this tutorial, the Oryx-Embedded.Middleware.1.7.8. pack is taken as an example to demonstrate how to add pack software components to STM32CubeMX projects. The use of this package shall not be understood as an STMicroelectronics recommendation.

To add embedded software packs to your project, proceed as follows:

1. Install Oryx-Embedded.Middleware.1.7.8.pack using the .pdsc file available from http://www.oryx-embedded.com (see Section 3.4.4: Installing embedded software packs).
2. Select New project.
3. Select STM32F01CCFx from the MCU selector.
4. Select Additional Software from the Pinout & Configuration view to open the additional software component window and choose the following software components: Compiler Support, RTOS Port/None and Date Time Helper Routines from the CycloneCommon bundle (see Section 4.12: Additional software component selection window).
5. Click OK to display the selected components on the tree view and click the checkbox to enable the software components for the current project (see Figure 276).

The pack name highlighted in green indicates that all conditions for the selected software components resolve to true. If at least one condition is not resolved, the pack name is highlighted in orange.
6. Check that no parameters can be configured in the Configuration tab (see Figure 277).

Figure 277. Pack software components - no configurable parameters

7. Select the Project manager project tab to specify project parameters (see Figure 278), and choose IAR™ EWARM as IDE.

Figure 278. Pack tutorial - project settings
8. Generate your project by clicking **Generate Code**. Accept to download the STM32CubeF4 MCU package if it is not present in STM32Cube repository.

9. Click **Open project**. The Oryx software components are displayed in the generated project (see **Figure 279**).

*Figure 279. Generated project with third party pack components*
This tutorial demonstrates how to achieve a functional project using the X-Cube-BLE1 software pack.

Below the prerequisites to run this tutorial:

- **Hardware:** NUCLEO-L053R8, X-NUCLEO-IDB05A1 and mini-USB cable (see Figure 280)
- **Tools:** STM32CubeMX, IDE (Atollic® or any other toolchain supported by STM32CubeMX)
- **Embedded software package:** STM32CubeL0 (version 1.10.0 or higher), X-Cube-BLE1 1.1.0 (see Figure 281).
- **Mobile application** (see Figure 282): STMicroelectronics BlueNRG application for iOS® or Android™
Figure 281. Embedded software packages

Figure 282. Mobile application
Proceed as follows to install and run the tutorial:

1. Check STM32CubeMX Internet connection:
   a) Select the Help > Updater settings menu to open the updater window.
   b) Verify in the Connection tab that the Internet connection is configured and up.

2. Install the required embedded software packages (see Figure 283):
   a) Select the Help > Manage Embedded software packages menu to open the embedded software package manager window.
   b) Click the Refresh button to refresh the list with the latest available package versions.
   c) Select the STM32Cube MCU Package tab and check that the STM32CubeL0 firmware package version 1.10.0 or higher is installed (the checkbox must be green). Otherwise select the checkbox and click Install now.
   d) Select the STMicroelectronics tab and check that the X-Cube-BLE1 software pack version 1.0.0 is installed (checkbox must be green). Otherwise, select the checkbox and click Install now.

3. Start a new project:
   a) Select New Project to open the new project window.
   b) Select the Board selector tab.
   c) Select Nucleo64 as board type and STM32L0 as MCU Series.
   d) Select the NUCLEO-L053R8 from the resulting board list (see Figure 284).
   e) Answer No when prompted to initialize all peripherals in their default mode (see Figure 285).
4. Add X-Cube-BLE1 components to the project:
   a) Click Additional Software from Pinout & Configuration view to open the Additional Software component Selection window.
   b) Select the relevant components (see Figure 286)

The Application group comes with a list of applications: the C files implement the application loop, that is the Process() function. From the Application group, select the SensorDemo application.

Select the Controller and Utilis components

Select the Basic variant for the HCI_TL component. The Basic variant provides the STMicroelectronics implementation of the HCI_TL API while the template option requires the user to implement his own code.

Select the UserBoard variant as HCI_TL_INTERFACE component. Using the UserBoard option generates the <boardname>_bus.c file, that is nucleo_l053r8_bus.c for this tutorial, while the template option generates the custom_bus.c file and requires the user to provide his own implementation.

Refer to the X-Cube-BLE1 pack documentation for more details on software components.
c) Click **OK** to apply the selection to the project and close the window. The left panel **Additional Software** section is updated accordingly.

![Figure 286. Selecting X-Cube-BLE1 components](image)

5. Enable peripherals and GPIOs from the **Pinout** tab (see **Figure 287**):
   a) Configure USART2 in **Asynchronous** mode.
   b) Configure SPI1 in **Full-duplex master** mode.
   c) Left-click the following pins and configure them for the required GPIO settings:
      - PA0: GPIO_EXT10
      - PA1: GPIO_Output
      - PA8: GPIO_Output
   d) Enable **Debug Serial Wire** under **SYS** peripheral.
6. Configure the peripherals from the Configuration tab:
   a) Click the NVIC button under the System section to open the NVIC configuration window. Enable EXTI line 0 and line 1 interrupts and click OK (see Figure 288).
   b) Click the SPI button under the Connectivity section to open the SPI configuration window. Check that the data size is set to 8 bits and the prescaler value to 16 so that HCLK divided by the prescaler value is less or equal to 8 MHz.
   c) Click USART2 under the Connectivity section to open the Configuration window and check the following parameter settings:
      Under Parameter Settings:
      - Baud rate: 115200 bits/s
      - Word length: 8 bits (including parity)
      - Parity: none
      - Stop bits: 1
      Under GPIO Settings:
      - User labels: USART_TX and USART_RX
7. Enable and configure X-Cube-BLE1 pack components from the Pinout & Configuration view:
   a) Click the pack items from the left panel to show the mode and configuration tabs.
   b) Click the check boxes from the Mode panel to enable X-Cube-BLE1, the configuration panel appears showing the parameters to configure. An orange triangle indicates that some parameters are not configured. It turns into a green check mark once all parameters are correctly configured (see Figure 289).
   c) Leave the Parameter Settings Tab unchanged.
   d) Go the Platform settings tab, configure the connection with the hardware resources as indicated in Figure 289 and Table 19.

Table 19. Connection with hardware resources

<table>
<thead>
<tr>
<th>Name</th>
<th>IPs or components</th>
<th>Found solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUS IO driver</td>
<td>SPI in Full-duplex master mode</td>
<td>SPI1</td>
</tr>
<tr>
<td>EXTI Line</td>
<td>GPIO:EXTI</td>
<td>PA0</td>
</tr>
<tr>
<td>CS Line</td>
<td>GPIO:output</td>
<td>PA1</td>
</tr>
<tr>
<td>Reset Line</td>
<td>GPIO:output</td>
<td>PA8</td>
</tr>
<tr>
<td>BSP LED</td>
<td>GPIO:output</td>
<td>PA5</td>
</tr>
<tr>
<td>BSP Button</td>
<td>GPIO:EXTI</td>
<td>PC13</td>
</tr>
<tr>
<td>BSP USART</td>
<td>USART in Asynchronous mode</td>
<td>USART2</td>
</tr>
</tbody>
</table>

Check that the icon turns to ✔️. Click OK to close the Configuration window.
8. Generate the SensorDemo project:
   a) Click \(\text{GENERATE CODE}\) to generate the code. The Project settings window opens if the project has not yet been saved.
   b) Click \(\text{GENERATE CODE}\) to generate the code once the project settings have been properly configured (see Figure 290). When the generation is complete, a dialog window requests to open the project folder (Open Folder) or to open the project in IDE toolchain (Open Project). Select Open Project (see Figure 291).
   c) If .cproject files are associated to Atollic® TrueStudio®, TrueStudio® is automatically launched by clicking Open Project: from the TrueStudio launch window, create or select an existing workspace (see Figure 292) and click OK. STM32CubeMX generated project appears in the TrueStudio® Project explorer panel (see Figure 293).
Figure 290. Configuring the SensorDemo project

Figure 291. Open SensorDemo project in the IDE toolchain
**Figure 292. Launching the SensorDemo project in Atollic® TrueStudio®**

![Eclipse Launcher](image1)

**Figure 293. Viewing the SensorDemo project in Atollic® TrueStudio®**

![Project Explorer](image2)
9. Build and Run the SensorDemo application from the Atollic® TrueStudio®:
   a) Configure the project properties (see Figure 294)
      From the Project explorer panel, right-click the project name (SensorDemo) and select Properties to open the Properties window.
      Select C/C++ Build and enable parallel build from the Behavior tab to speed the build process up.
   b) Click the build icon, , to build the project.
   c) Connect your computer to the Nucleo board ST-link connector via the USB cable.
   d) Click from the Run menu to run the project on the board.

   Figure 294. Configuring the SensorDemo project in Atollic® TrueStudio®

10. Test the STM32 SensorDemo application by launching the BlueNRG application on the phone:
   e) Scan for nearby devices.
   f) Select the BlueNRG device.
   g) Since there is no MEMs sensing elements on the hardware, press the Blue Button to simulate MEMs data: the ST cube rotates by a fixed value each time the button is pressed (see Figure 295).
Figure 295. Testing the SensorDemo application
17 Tutorial 8 – Using STemWin Graphics framework

17.1 Step 1: Selecting an MCU for Graphics

First, Launch STM32CubeMX and click New Project to open the New Project window (see Figure 296)

1. On the left panel, select the Enable checkbox under Graphics choice to display Graphics related parameters, the Graphics summary panel and the list of suitable MCUs.
2. Select the STM32F469NIHx from the MCUs list.
3. Click Start Project to open STM32CubeMX project main view.

Figure 296. Tutorial - Selecting an MCU for Graphics

17.2 Step 2: Enabling STemWin from the pinout view

To enable STemWin, the conflicts must be solved. Hover the mouse over the Graphics framework selection field to display conflict details in tooltips (see Figure 297).

Figure 297. Graphics frameworks tooltip
From the pinout view left panel (see Figure 298)

1. Enable the necessary peripherals: CRC, DMA2D (optional, for Graphics acceleration), FMC SDRAM mode, RCC HSE, LTDC in DSI mode and DSI Host in Adapted Command Mode with TE pin (for connection with Display interface). Under Graphics, STemWin framework is now available for selection.

2. Select STemWin as the Graphics framework and Display Parallel interface using LTDC as the Display Interface.

![Figure 298. Enabling STemWin framework](image)

17.3 Step 3: Configuring STemWin parameters from the configuration window

1. Select the configuration view and click the GRAPHICS button to show STemWin graphics framework configuration (see Figure 299)

2. Use the Parameter Settings panel to adjust Graphics parameters: under External Tools group, enable the use of GUIBuilder tool before proceeding with next step.
17.4 Step 4: Using STemWin GUIBuilder tool from the configuration window

1. Make sure the use of GUIBuilder tool is enabled on the parameter settings panel.
2. Select the STemWin panel and configure the Application Category to be either Window or WindowFrame (see Figure 300).
3. Click Execute to open the GUIBuilder user interface (see Figure 301).

Note: The project settings window pops-up when the project settings are not specified. TrueStudio®, Makefile and other (GPDSC) toolchains are not supported with STemWin and are grayed out in the user interface.

Note: The STM32CubeF4 embedded software package will be downloaded (if not available) in the user’s STM32Cube repository when STM32CubeMX launches the GUIBuilder executable file from this package.

Note: STM32CubeMX user interface is not accessible when the GUIBuilder tool is open.

4. Complete the GUI design work using GUIBuilder user interface and click File > Save to generate the corresponding C file, respectively WindowDLG.c for Window type, FramewinDLG.c for WindowFrame type.
5. Close GUIBuilder to return to STM32CubeMX user interface.
17.5 Step 5: Generating the embedded C project and updates

Click to generate the project (see Figure 302).

- Application main files are generated under Core/Inc and Src folders.
- STemWin library configuration files are generated under STemWin/App folder.
- Hardware initialization and wrapper files under STemWin/Target folder.

Update the generated code by placing user code in dedicated sections, i.e. between //USER START and //USER END tags.
Figure 302. StemWin generated project and files
Tutorial 9: Using STM32CubeMX Graphics simulator

Click GFXSIMULATOR from the configuration view to open the Graphics simulator user interface (see Figure 303).

Figure 303. GFXSIMULATOR in Configuration view

1. Check in the top panel the Simulator’s current working assumptions and the mandatory configuration settings, e.g. if LTDC is not enabled, some results will show as NA (not available). See Figure 304.
2. Adjust the simulator parameter values from the left center panel, compare to current configuration settings in the right panel (see Figure 305).
3. Check in the bottom panel the results for a set of supported use cases.
4. Optionally, click Import Settings to update current project configuration with simulator settings to reach same performance results.
   - A confirmation message pops up.
   - When relevant, a warning message pops up with the settings that could not be imported (unused peripherals, out of range clock frequency value,...).
5. Close the Graphics simulator window to go back to STM32CubeMX configuration view.
Figure 304. Graphics simulator user interface

Figure 305. Graphics simulator - Current Configuration fields
19 Tutorial 10: Using ST-TouchGFX framework

19.1 Step 1: Selecting an MCU for Graphics.


19.2 Step 2: Enabling ST-TouchGFX from the pinout view

See Section 17: Tutorial 8 – Using STemWin Graphics framework to enable a graphic framework from the pinout view.

1. Enable ST-TouchGFX required peripherals and middleware (see Figure 297 ST-TouchGFX tooltip for details): FreeRTOS, DMA2D, TIM, FMC (SDRAM or SRAM), LTDC in DSI mode, DSIHOST (requires RCC HSE).
2. Enable ST-TouchGFX and set the Display Interface (see Figure 306).

Figure 306. Enabling ST-TouchGFX
19.3 Step 3: Configuring ST-TouchGFX parameters from the configuration window

**Note:** Prerequisite: ST-TouchGFX framework requires ST-TouchGFXDesigner.

To install ST-TouchGFXDesigner, double-click the .msi installer file located within the STM32CubeF4 embedded software package (as shown in **Figure 307**) to launch the installation wizard.

**Figure 307. TouchGFXDesigner: installer location**

1. From the Configuration view, click GRAPHICS to open the configuration window.
2. STM32CubeMX highlights in red when the selected LTDC pixel format is not compatible with TouchGFX, as exemplified in **Figure 308**. Adjust LTDC settings to solve the issue (**Figure 309**).

**Figure 308. TouchGFX incompatible LTDC Pixel format**
3. STM32CubeMX highlights in red (as exemplified in Figure 310) when the path to TouchGFXDesigner.exe is missing.

![Figure 310. TouchGFXDesigner path missing](image)

Select the “ST-TouchGFXDesigner.exe not found” text in red to reveal the icon. Click on it and browse to TouchGFX installation folder: select ST-TouchGFXDesigner.exe file from the designer folder (see Figure 311).
19.4 Step 4: Generating the C project and using ST-TouchGFXDesigner

1. First, click on the project generation icon to generate the project using STM32CubeMX.

Note: An error message pops-up on clicking ST-TouchGFXDesigner Execute button before the project is generated (see Figure 312).

2. Then, click on the “Execute Button” to open ST-TouchGFXDesigner user interface (see Figure 313).
3. Proceed with design work using ST-TOUCHGFXDesigner (see Figure 314).
5. Close the tool to return to STM32CubeMX user interface.

Figure 314. ST-TOUCHGFXDesigner user interface

19.5 Step 5: Generated project files

For generated project files, see Figure 315:
- Application main files are generated under Core/Inc and Src folders.
- Under ST-TouchGFX folder:
  - Hardware initialization and wrapper files are generated under the target folder.
  - ST-TouchGFXDesigner generated code is found in the other folders.

Update the generated code by placing user code in dedicated sections, i.e. between //USER START and //USER END tags.
Figure 315. ST-TOUCHGFX generated project and files
20 FAQ

20.1 On the Pinout configuration panel, why does STM32CubeMX move some functions when I add a new peripheral mode?

You may have deselected . In this case, the tool performs an automatic remapping to optimize your placement.

20.2 How can I manually force a function remapping?

Use the Manual Remapping feature.

20.3 Why are some pins highlighted in yellow or in light green in the Pinout view? Why cannot I change the function of some pins (when I click some pins, nothing happens)?

These pins are specific pins (such as power supply or BOOT) which are not available as peripheral signals.

20.4 Why do I get the error “Java 7 update 45” when installing “Java 7 update 45” or a more recent version of the JRE?

The problem generally occurs on 64-bit Windows operating system, when several versions of Java™ are installed on your computer and the 64-bit Java™ installation is too old. During STM32CubeMX installation, the computer searches for a 64-bit installation of Java™.

- If one is found, the ‘Java 7 update 45’ minimum version prerequisite is checked. If the installed version is older, an error is displayed to request the upgrade.
- If no 64-bit installation is found, STM32CubeMX searches for a 32-bit installation. If one is found and the version is too old, the ‘Java 7 update 45’ error is displayed. The user must update the installation to solve the issue.

To avoid this issue from occurring, it is recommended to perform one of the following actions:

1. Remove all Java™ installations and reinstall only one version (32 or 64 bits) (Java 7 update 45 or more recent).
2. Keep 32-bit and 64-bit installations but make sure that the 64-bit version is at least Java 7 update 45.

Note: Some users (Java developers for example) may need to check the PC environment variables defining hard-coded Java paths (e.g. JAVA_HOME or PATH) and update them so that they point to the latest Java installation.

On Windows 7 you can check the Java installation using the Control Panel. To do this, double-click the icon from Control Panel\All Control Panel to open the Java™ settings window (see Figure 316).
20.5 Why does the RTC multiplexer remain inactive on the Clock tree view?

To enable the RTC multiplexer, the user shall enable the RTC peripheral in the Pinout view as indicated below.

Figure 317. Pinout view - Enabling the RTC

You can also enter `java –version` as an MS-DOS command to check the version of your latest Java installation (the Java program called here is a copy of the program installed under C:\Windows\System32):

```java
java version "1.7.0_45"
Java(TM) SE Runtime Environment (build 1.7.0_45-b18)
Java HotSpot(TM) 64-Bit Server VM (build 24.45-b08, mixed mode)```

Figure 316. Java™ Control Panel
20.6 How can I select LSE and HSE as clock source and change the frequency?

The LSE and HSE clocks become active once the RCC is configured as such in the Pinout view. See Figure 318 for an example.

Figure 318. Pinout view - Enabling LSE and HSE clocks

The clock source frequency can then be edited and the external source selected, see Figure 319.

Figure 319. Pinout view - Setting LSE/HSE clock frequency

20.7 Why STM32CubeMX does not allow me to configure PC13, PC14, PC15 and PI8 as outputs when one of them is already configured as an output?

STM32CubeMX implements the restriction documented in the reference manuals as a footnote in table Output Voltage characteristics:

"PC13, PC14, PC15 and PI8 are supplied through the power switch. Since the switch only sinks a limited amount of current (3 mA), the use of GPIOs PC13 to PC15 and PI8 in output mode is limited: the speed should not exceed 2 MHz with a maximum load of 30 pF and these I/Os must not be used as a current source (e.g. to drive a LED)."
20.8 Ethernet Configuration: why cannot I specify DP83848 or LAN8742A in some cases?

For most Series, STM32CubeMX will adjust the list of possible PHY component drivers according to the Ethernet mode that has been selected:

- When the Ethernet MII mode is selected, the user will be able to choose between the DP83848 component driver or a “User Phy”.
- When the Ethernet RMII mode is selected, the user will be able to choose between the LAN8742A component driver or a “User Phy”.

When “User Phy” is selected, the user will have to manually include the component drivers to be used in its project.

*Note:* For STM32H7 Series, the PHY is seen as an external component and is no longer specified under the Ethernet peripheral configuration. The user can select the PHY under LwIP Platform settings tab. However, since the STM32H7 firmware package provides only the driver code for the LAN8742A component that is available on all STM32H7 evaluation and Nucleo boards, STM32CubeMX user interface offers only the choice between “User Phy” and LAN8742.

When LAN8742 is selected, the BSP driver code is copied into the generated project.
Appendix A  STM32CubeMX pin assignment rules

The following pin assignment rules are implemented in STM32CubeMX:
- Rule 1: Block consistency
- Rule 2: Block inter-dependency
- Rule 3: One block = one peripheral mode
- Rule 4: Block remapping (only for STM32F10x)
- Rule 5: Function remapping
- Rule 6: Block shifting (only for STM32F10x)
- Rule 7: Setting or clearing a peripheral mode
- Rule 8: Mapping a function individually (if Keep Current Placement is unchecked)
- Rule 9: GPIO signals mapping

A.1 Block consistency

When setting a pin signal (provided there is no ambiguity about the corresponding peripheral mode), all the pins/signals required for this mode are mapped and pins are shown in green (otherwise the configured pin is shown in orange).

When clearing a pin signal, all the pins/signals required for this mode are unmapped simultaneously and the pins turn back to gray.

Example of block mapping with a STM32F107x MCU

If the user assigns I2C1_SMBA function to PB5, then STM32CubeMX configures pins and modes as follows:
- I2C1_SCL and I2C1_SDA signals are mapped to the PB6 and PB7 pins, respectively (see Figure 320).
- I2C1 peripheral mode is set to SMBus-Alert mode.
Example of block remapping with a STM32F107x MCU

If the user assigns GPIO_Output to PB6, STM32CubeMX automatically disables I2C1 SMBus-Alert peripheral mode from the peripheral tree view and updates the other I2C1 pins (PB5 and PB7) as follows:

- If they are unpinned, the pin configuration is reset (pin grayed out).
- If they are pinned, the peripheral signal assigned to the pins is kept and the pins are highlighted in orange since they no longer match a peripheral mode (see Figure 321).
Figure 321. Block remapping

For STM32CubeMX to find an alternative solution for the I2C peripheral mode, the user will need to unpin I2C1 pins and select the I2C1 mode from the peripheral tree view (see Figure 322 and Figure 323).
Figure 322. Block remapping - Example 1

Figure 323. Block remapping - Example 2
A.2 Block inter-dependency

On the Pinout view, the same signal can appear as an alternate function for multiple pins. However it can be mapped only once.

As a consequence, for STM32F1 MCUs, two blocks of pins cannot be selected simultaneously for the same peripheral mode: when a block/signal from a block is selected, the alternate blocks are cleared.

Example of block remapping of SPI in full-duplex master mode with a STM32F107x MCU

If SPI1 full-duplex master mode is selected from the tree view, by default the corresponding SPI signals are assigned to PB3, PB4 and PB5 pins (see Figure 324).

If the user assigns to PA6 the SPI1_MISO function currently assigned to PB4, STM32CubeMX clears the PB4 pin from the SPI1_MISO function, as well as all the other pins configured for this block, and moves the corresponding SPI1 functions to the relevant pins in the same block as the PB4 pin (see Figure 325).

(by pressing CTRL and clicking PB4 to show PA6 alternate function in blue, then drag and drop the signal to pin PA6)

Figure 324. Block inter-dependency - SPI signals assigned to PB3/4/5
Figure 325. Block inter-dependency - SPI1_MISO function assigned to PA6
A.3 One block = one peripheral mode

When a block of pins is fully configured in the **Pinout** view (shown in green), the related peripheral mode is automatically set in the Peripherals tree.

**Example of STM32F107x MCU**

Assigning the I2C1_SMBA function to PB5 automatically configures I2C1 peripheral in SMBus-Alert mode (see Peripheral tree in *Figure 326*).

**Figure 326. One block = one peripheral mode - I2C1_SMBA function assigned to PB5**

![Diagram showing I2C1_SMBA function assigned to PB5]

A.4 Block remapping (STM32F10x only)

To configure a peripheral mode, STM32CubeMX selects a block of pins and assigns each mode signal to a pin in this block. In doing so, it looks for the first free block to which the mode can be mapped.

When setting a peripheral mode, if at least one pin in the default block is already used, STM32CubeMX tries to find an alternate block. If none can be found, it either selects the functions in a different sequence, or unchecks and remaps all the blocks to find a solution.
Example

STM32CubeMX remaps USART3 hardware-flow-control mode to the (PD8-PD9-PD11-PD12) block, because PB14 of USART3 default block is already allocated to the SPI2_MISO function (see Figure 327).

Figure 327. Block remapping - Example 2

A.5 Function remapping

To configure a peripheral mode, STM32CubeMX assigns each signal of the mode to a pin. In doing so, it will look for the first free pin the signal can be mapped to.

Example using STM32F415x

When configuring USART3 for the Synchronous mode, STM32CubeMX discovered that the default PB10 pin for USART3_TX signal was already used by SPI. It thus remapped it to PD8 (see Figure 328).

Figure 328. Function remapping example
A.6 Block shifting (only for STM32F10x and when “Keep Current Signals placement” is unchecked)

If a block cannot be mapped and there are no free alternate solutions, STM32CubeMX tries to free the pins by remapping all the peripheral modes impacted by the shared pin.

Example

With the Keep current signal placement enabled, if USART3 synchronous mode is set first, the Asynchronous default block (PB10-PB11) is mapped and Ethernet becomes unavailable (shown in red) (see Figure 329).

Unchecking allows STM32CubeMX shifting blocks around and freeing a block for the Ethernet MII mode. (see Figure 330).

Figure 329. Block shifting not applied
A.7 Setting and clearing a peripheral mode

The Peripherals panel and the Pinout view are linked: when a peripheral mode is set or cleared, the corresponding pin functions are set or cleared.

A.8 Mapping a function individually

When STM32CubeMX needs a pin that has already been assigned manually to a function (no peripheral mode set), it can move this function to another pin, only if the keep current signals placement is unchecked and the function is not pinned (no pin icon).

A.9 GPIO signals mapping

I/O signals (GPIO_Input, GPIO_Output, GPIO_Analog) can be assigned to pins either manually through the Pinout view or automatically through the Pinout menu. Such pins can no longer be assigned automatically to another signal: STM32CubeMX signal automatic placement does not take into account this pin anymore since it does not shift I/O signals to other pins.

The pin can still be manually assigned to another signal or to a reset state.
Appendix B  STM32CubeMX C code generation design choices and limitations

This section summarizes STM32CubeMX design choices and limitations.

B.1  STM32CubeMX generated C code and user sections

The C code generated by STM32CubeMX provides user sections as illustrated below. They allow user C code to be inserted and preserved at next C code generation.

User sections shall neither be moved nor renamed. Only the user sections defined by STM32CubeMX are preserved. User created sections will be ignored and lost at next C code generation.

/* USER CODE BEGIN 0 */

(...)

/* USER CODE END 0 */

Note: STM32CubeMX may generate C code in some user sections. It will be up to the user to clean the parts that may become obsolete in this section. For example, the while(1) loop in the main function is placed inside a user section as illustrated below:

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)
{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

}

/* USER CODE END 3 */

B.2  STM32CubeMX design choices for peripheral initialization

STM32CubeMX generates peripheral _Init functions that can be easily identified thanks to the MX_ prefix:

static void MX_GPIO_Init(void);
static void MX_<Peripheral Instance Name>_Init(void);
static void MX_I2S2_Init(void);

An MX_<peripheral instance name>_Init function exists for each peripheral instance selected by the user (e.g, MX_I2S2_Init). It performs the initialization of the relevant handle structure (e.g, &hi2s2 for I2S second instance) that is required for HAL driver initialization (e.g., HAL_I2S_Init) and the actual call to this function:

void MX_I2S2_Init(void)
{

hi2s2.Instance = SPI2;
hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
hi2s2.Init.Standard = I2S_STANDARD_PHILLIPS;
hi2s2.Init.DataFormat = I2S_DATAFORMAT_16B;
hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_192K;
hi2s2.Init.CPOL = I2S_CPOL_LOW;
hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_ENABLE;
HAL_I2S_Init(&hi2s2);
}

By default, the peripheral initialization is done in main.c. If the peripheral is used by a middleware mode, the peripheral initialization can be done in the middleware corresponding .c file.

Customized HAL_<Peripheral Name>_MspInit() functions are created in the stm32f4xx_hal_msp.c file to configure the low-level hardware (GPIO, CLOCK) for the selected peripherals.

B.3 STM32CubeMX design choices and limitations for middleware initialization

B.3.1 Overview

STM32CubeMX does not support C user code insertion in Middleware stack native files although stacks such as LwIP might require it in some use cases.

STM32CubeMX generates middleware init functions that can be easily identified thanks to the MX_ prefix:

MX_LWIP_Init(); // defined in lwip.h file
MX_USB_HOST_Init(); // defined in usb_host.h file
MX_FATFS_Init(); // defined in fatfs.h file

Note however the following exceptions:

- No init function is generated for FreeRTOS unless the user chooses, from the Project settings window, to generate init functions as pairs of .c/.h files. Instead, a StartDefaultTask function is defined in the main.c file and CMSIS-RTOS native function (osKernelStart) is called in the main function.
- If FreeRTOS is enabled, the init functions for the other middlewares in use are called from the StartDefaultTask function in the main.c file.

Example:

```c
void StartDefaultTask(void const * argument)
{
    /* init code for FATFS */
    MX_FATFS_Init();
    /* init code for LWIP */
    MX_LWIP_Init();
    /* init code for USB_HOST */
    MX_USB_HOST_Init();
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
```
for(;;)
{
    osDelay(1);
}
/* USER CODE END 5 */
}

B.3.2 USB Host

USB peripheral initialization is performed within the middleware initialization C code in the usbh_conf.c file, while USB stack initialization is done within the usb_host.c file.

When using the USB Host middleware, the user is responsible for implementing the USBH_UserProcess callback function in the generated usbh_conf.c file.

From STM32CubeMX user interface, the user can select to register one class or all classes if the application requires switching dynamically between classes.

B.3.3 USB Device

USB peripheral initialization is performed within the middleware initialization C code in the usbd_conf.c file, while USB stack initialization is done within the usbd_device.c file.

USB VID, PID and String standard descriptors are configured via STM32CubeMX user interface and available in the usbd_desc.c generated file. Other standard descriptors (configuration, interface) are hard-coded in the same file preventing support of USB composite devices.

When using the USB Device middleware, the user is responsible for implementing the functions in the usbd_<classname>_if.c class interface file for all device classes (e.g., usbd_storage_if.c).

USB MTP and CCID classes are not supported.

B.3.4 FatFs

FatFs is a generic FAT/exFAT file system solution well suited for small embedded systems.

FatFs configuration is available in ffconf.h generated file.

The initialization of the SDIO peripheral for the FatFs SD Card mode and of the FMC peripheral for the FatFs External SDRAM and External SRAM modes are kept in the main.c file.

Some files need to be modified by the user to match user board specificities (BSP in STM32Cube embedded software package can be used as example):
• bsp_driver_sd.c/.h generated files when using FatFs SD Card mode
• bsp_driver_sram.c/.h generated files when using FatFs External SRAM mode
• bsp_driver_sdram.c/.h generated files when using FatFs External SDRAM mode.

Multi-drive FatFs is supported, which means that multiple logical drives can be used by the application (External SDRAM, External SRAM, SD Card, USB Disk, User defined). However support of multiple instances of a given logical drive is not available (e.g. FatFs using two instances of USB hosts or several RAM disks).
NOR and NAND Flash memory are not supported. In this case, the user shall select the FatFs user-defined mode and update the user_diskio.c driver file generated to implement the interface between the middleware and the selected peripheral.

B.3.5 FreeRTOS

FreeRTOS is a free real-time embedded operating system well suited for microcontrollers. FreeRTOS configuration is available in FreeRTOSConfig.h generated file.

When FreeRTOS is enabled, all other selected middleware modes (e.g., LwIP, FatFs, USB) will be initialized within the same FreeRTOS thread in the main.c file.

When GENERATE_RUN_TIME_STATS, CHECK_FOR_STACK_OVERFLOW, USE_IDLE_HOOK, USE_TICK_HOOK and USE_MALLOC_FAILED_HOOK parameters are activated, STM32CubeMX generates freertos.c file with empty functions that the user shall implement. This is highlighted by the tooltip (see Figure 331).

Figure 331. FreeRTOS HOOK functions to be completed by user
B.3.6 LwIP

LwIP is a small independent implementation of the TCP/IP protocol suite: its reduced RAM usage makes it suitable for use in embedded systems with tens of kilobytes of free RAM.

LwIP initialization function is defined in \textit{lwip.c}, while LwIP configuration is available in \textit{lwipopts.h} generated file.

STM32CubeMX supports LwIP over Ethernet only. The Ethernet peripheral initialization is done within the middleware initialization C code.

STM32CubeMX does not support user C code insertion in stack native files. However, some LwIP use cases require modifying stack native files (e.g., \textit{cc.h}, \textit{mib2.c}): user modifications shall be backed up since they will be lost at next STM32CubeMX generation.

Starting with LwIP release 1.5, STM32CubeMX LwIP supports IPv6 (see \textit{Figure 333}).

DHCP must be disabled, to configure a static IP address.

\textbf{Figure 332. LwIP 1.4.1 configuration}
STM32CubeMX generated C code will report compilation errors when specific parameters are enabled (disabled by default). The user must fix the issues with a stack patch (downloaded from Internet) or user C code. The following parameters generate an error:

- MEM_USE_POOLS: user C code to be added either in `lwipopts.h` or in `cc.h` (stack file).
- PPP_SUPPORT, PPPOE_SUPPORT: user C code required
- MEMP_SEPARATE_POOLS with MEMP_OVERFLOW_CHECK > 0: a stack patch required
- MEM_LIBC_MALLOC & RTOS enabled: stack patch required
- LWIP_EVENT_API: stack patch required

In STM32CubeMX, the user must enable FreeRTOS in order to use LwIP with the netconn and sockets APIs. These APIs require the use of threads and consequently of an operating system. Without FreeRTOS, only the LwIP event-driven raw API can be used.
B.3.7 Libjpeg

Libjpeg is a widely used C-library that allows reading and writing JPEG files. It is delivered within STM32CubeF7, STM32CubeH7, STM32CubeF2 and STM32CubeF4 embedded software packages.

STM32CubeMX generates the following files, whose content can be configured by the user through STM32CubeMX user interface:

- **libjpeg.c/.h**
  The `MX_LIBJPEG_Init()` initialization function is generated within the libjpeg.c file. It is empty. It is up to the user to enter in the user sections the code and the calls to the libjpeg functions required for the application.

- **jdata_conf.c**
  This file is generated only when FatFs is selected as data stream management type.

- **jdata_conf.h**
  The content of this file is adjusted according to the datastream management type selected.

- **jconfig.h**
  This file is generated by STM32CubeMX. but cannot be configured.

- **jmorecfg.h**
  Some but not all the define statements contained in this file can be modified through the STM32CubeMX libjpeg configuration menu.
B.3.8 Mbed TLS

Mbed TLS is a C-library that allows including cryptographic capabilities to embedded products. It handles Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, that are used for establishing a secure, encrypted and authenticated link between two parties over an insecure network. Mbed TLS comes with an intuitive API and minimal coding footprint. Visit https://tls.mbed.org/ for more details.

Mbed TLS is delivered within STM32CubeF2, STM32CubeF4, STM32CubeF7 and STM32CubeH7 embedded software packages.

Mbed TLS can work without LwIP stack (see Figure 335: Mbed TLS without LwIP).

If LwIP stack is used, FreeRTOS must be enabled as well (see Figure 336: Mbed TLS with LwIP and FreeRTOS).
STM32CubeMX generates the following files, whose contents can be modified by the user through STM32CubeMX user interface (see Figure 337: Mbed TLS configuration window) and/or using user sections in the code itself:

- `mbedtls_config.h`
- `mbedtls.h`
- `net_sockets.c` (generated only if LwIP is enabled)
- `mbedtls.c`

**Figure 335. Mbed TLS without LwIP**
Figure 336. Mbed TLS with LwIP and FreeRTOS
B.3.9 TouchSensing

The STM32 TouchSensing library is a C-library that allows the creation of higher-end human interfaces by replacing conventional electromechanical switches by capacitive sensors with STM32 microcontrollers.

It requires the touch-sensing peripheral to be configured on the microcontroller.

STM32CubeMX generates the following files, whose contents can be modified by the user through STM32CubeMX user interface (see Figure 338: Enabling the TouchSensing peripheral, Figure 339: Touch-sensing sensor selection panel and Figure 340: TouchSensing configuration panel) and/or using user sections in the code itself:

- `touchsensing.c/.h`
- `tsl_user.c/.h`
- `tsl_conf.h`
Figure 338. Enabling the TouchSensing peripheral
Figure 339. Touch-sensing sensor selection panel
B.3.10 PDM2PCM

The PDM2PCM library is a C-library that allows converting a pulse density modulated (PDM) data output into a 16-bit pulse-code modulation (PCM) format. It requires the CRC peripheral to be enabled.

STM32CubeMX generates the following files, whose contents can be modified by the user through STM32CubeMX user interface and/or using user sections in the code itself:

- `pdm2pcm.h/.c`

B.3.11 Graphics

STMicroelectronics has selected the STemWin framework for the STM32 portfolio, see the dedicated tutorial sections on how to create projects for this framework using STM32CubeMX.
The generated projects follow the embedded software architecture shown in Figure 341 with a hardware initialization file and a wrapper file to address the graphical stack specificity.

**Figure 341. Graphics application architecture**

The graphical stack comes with a specific UI designer tool that can be called from STM32CubeMX user interface to perform advanced design work:

- **GUIBuilder** for **STemWin** framework

All C, C++ or other files generated by the tool end up in the STM32CubeMX project folder.
B.3.12 STM32WPAN BLE/Thread (STM32WB Series only)

STM32WPAN BLE and Thread middleware are now supported in STM32CubeMX.

Figure 342. BLE and Thread middleware support in STM32CubeMX

They are exclusive in a given project and configuration with FreeRTOS is not yet supported.
Application projects generated with STM32CubeMX can be found in the project folder of the STM32CubeWB MCU package.

Figure 343. STM32CubeWB Package download
This package can be installed through STM32CubeMX following the standard procedure described in Section 3.4.2: Installing STM32 MCU packages.

Figure 344. STM32CubeWB BLE applications folder

BLE configuration

To enable BLE some peripherals (RTC, HSEM, RF) must be activated first.

Then, an application type must be selected, it can be one among Transparent mode, Server profile, Router profile or Client profile.

Finally, the mode and other parameters relevant to this application type must be configured.

Note: The BLE Transparent mode and all Thread applications require either the USART or the LPUART peripheral to be configured as well.
Figure 345. BLE Server profile selection

Figure 346. BLE Client profile selection
Thread configuration

To enable Thread some peripherals (RTC, HSEM, RF) must be activated first. Then, an application type must be selected and the relevant parameters configured.

Figure 347. Thread application selection

B.3.13 OpenAmp and RESMGR_UTILITY (STM32MP Series only)

New software and hardware have been introduced on STM32MP1 series to enable multi-core cooperation

- The inter-processor communication controller (IPCC) used to exchange data between two processor instances: it relies on the fact that shared memory buffers are allocated in the MCU SRAM and that each processor owns specific register bank and interrupts.
- The OpenAMP middleware for intercommunication between Cortex-A and Cortex-M cores: it implements the RPMsg messaging protocol (see Figure 348).
- The resource manager library (RESMGR_UTILITY) for system resource management: multi-processor devices give the possibility to run independent firmware on several cores (see Figure 349). This implies a core could use some peripherals without knowledge of the usage of these same peripherals: the role of the resource management library is to control the assignment of a peripheral to a dedicated core and to provide a method to configure the system resources used to operate that peripheral (see Figure 350).
Figure 348. Enabling OpenAmp for STM32MP1 devices

Figure 349. Enabling the Resource Manager for STM32MP1 devices
Figure 350. Resource Manager: peripheral assignment view

For more details visit STM32MP1 dedicated wiki site at https://wiki.st.com/stm32mpu.
Appendix C  STM32 microcontrollers naming conventions

STM32 microcontroller part numbers are codified following the below naming conventions:

- **Device subfamilies**
  The higher the number, the more features available.
  For example STM32L0 line includes STM32L051, L052, L053, L061, L062, L063 subfamilies where STM32L06x part numbers come with AES while STM32L05x do not.
  The last digit indicates the level of features. In the above example:
  - 1 = Access line
  - 2 = with USB
  - 3 = with USB and LCD.

- **Pin counts**
  - F = 20 pins
  - G = 28 pins
  - K = 32 pins
  - T = 36 pins
  - S = 44 pins
  - C = 48 pins
  - R = 64 (or 66) pins
  - M = 80 pins
  - O = 90 pins
  - V = 100 pins
  - Q = 132 pins (e. g. STM32L162QDH6)
  - Z = 144 pins
  - I = 176 (+25) pins
  - B = 208 pins (e. g. STM32F429BIT6)
  - N = 216 pins

- **Flash memory sizes**
  - 4 = 16 Kbytes of Flash memory
  - 6 = 32 Kbytes of Flash memory
  - 8 = 64 Kbytes of Flash memory
  - B = 128 Kbytes of Flash memory
  - C = 256 Kbytes of Flash memory
  - D = 384 Kbytes of Flash memory
  - E = 512 Kbytes of Flash memory
  - F = 768 Kbytes of Flash memory
  - G = 1024 Kbytes of Flash memory
  - I = 2048 Kbytes of Flash memory

- **Packages**
  - B = SDIP
  - H = BGA
UM1718 STM32 microcontrollers naming conventions

- M = SO
- P = TSSOP
- T = LQFP
- U = VFQFPN
- Y = WLCSP

*Figure 351* shows an example of STM32 microcontroller part numbering scheme.

**Figure 351. STM32 microcontroller part numbering scheme**

<table>
<thead>
<tr>
<th>Example:</th>
<th>STM32</th>
<th>F</th>
<th>439</th>
<th>V</th>
<th>I</th>
<th>T</th>
<th>6</th>
<th>xxx</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device family</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STM32 = ARM-based 32-bit microcontroller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Product type</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F = general-purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Device subfamily</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>437: STM32F437xx, USB OTG FS/HS, camera interface, Ethernet, cryptographic acceleration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>439: STM32F439xx, USB OTG FS/HS, camera interface, Ethernet, LCD-TFT, cryptographic acceleration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pin count</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V = 100 pins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z = 144 pins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = 169 pins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I = 176 pins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B = 208 pins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N = 216 pins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Flash memory size</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G = 1024 Kbytes of Flash memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I = 2048 Kbytes of Flash memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Package</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T = LQFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H = BGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y = WLCSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Temperature range</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 = Industrial temperature range, -40 to 85 °C.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 = Industrial temperature range, -40 to 105 °C.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xxx = programmed parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR = tape and reel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D  STM32 microcontrollers power consumption parameters

This section provides an overview on how to use STM32CubeMX Power Consumption Calculator.

Microcontroller power consumption depends on chip size, supply voltage, clock frequency and operating mode. Embedded applications can optimize STM32 MCU power consumption by reducing the clock frequency when fast processing is not required and choosing the optimal operating mode and voltage range to run from. A description of STM32 power modes and voltage range is provided below.

D.1 Power modes

STM32 MCUs support different power modes (refer to STM32 MCU datasheets for full details).

D.1.1 STM32L1 Series

STM32L1 microcontrollers feature up to 6 power modes, including 5 low-power modes:

- **Run** mode
  This mode offers the highest performance using HSE/HSI clock sources. The CPU runs up to 32 MHz and the voltage regulator is enabled.

- **Sleep** mode
  This mode uses HSE or HSI as system clock sources. The voltage regulator is enabled and the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.

- **Low-power run** mode
  This mode uses the multispeed internal (MSI) RC oscillator set to the minimum clock frequency (131 kHz) and the internal regulator in low-power mode. The clock frequency and the number of enabled peripherals are limited.

- **Low-power sleep** mode
  This mode is achieved by entering Sleep mode. The internal voltage regulator is in low-power mode. The clock frequency and the number of enabled peripherals are limited. A typical example would be a timer running at 32 kHz.

  When the wakeup is triggered by an event or an interrupt, the system returns to the Run mode with the regulator ON.

- **Stop** mode
  This mode achieves the lowest power consumption while retaining RAM and register contents. Clocks are stopped. The real-time clock (RTC) an be backed up by using LSE/LSI at 32 kHz/37 kHz. The number of enabled peripherals is limited. The voltage regulator is in low-power mode.

  The device can be woken up from Stop mode by any of the EXTI lines.

- **Standby** mode
  This mode achieves the lowest power consumption. The internal voltage regulator is switched off so that the entire \( V_{\text{CORE}} \) domain is powered off. Clocks are stopped and the real-time clock (RTC) can be preserved up by using LSE/LSI at 32 kHz/37 kHz.
RAM and register contents are lost except for the registers in the Standby circuitry. The number of enabled peripherals is even more limited than in Stop mode.

The device exits Standby mode upon reset, rising edge on one of the three WKUP pins, or if an RTC event occurs (if the RTC is ON).

Note: When exiting Stop or Standby modes to enter the Run mode, STM32L1 MCUs go through a state where the MSI oscillator is used as clock source. This transition can have a significant impact on the global power consumption. For this reason, the Power Consumption Calculator introduces two transition steps: WU_FROM_STOP and WU_FROM_STANDBY. During these steps, the clock is automatically configured to MSI.

D.1.2 STM32F4 Series

STM32F4 microcontrollers feature a total of 5 power modes, including 4 low-power modes:

- **Run mode**
  This is the default mode at power-on or after a system reset. It offers the highest performance using HSE/HSI clock sources. The CPU can run at the maximum frequency depending on the selected power scale.

- **Sleep mode**
  Only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/even occurs. The clock source is the clock that was set before entering Sleep mode.

- **Stop mode**
  This mode achieves a very low power consumption using the RC oscillator as clock source. All clocks in the 1.2 V domain are stopped as well as CPU and peripherals. PLL, HSI RC and HSE crystal oscillators are disabled. The content of registers and internal SRAM are kept.
  
  The voltage regulator can be put either in normal Main regulator mode (MR) or in Low-power regulator mode (LPR). Selecting the regulator in low-power regulator mode increases the wakeup time.
  
  The Flash memory can be put either in Stop mode to achieve a fast wakeup time or in Deep power-down to obtain a lower consumption with a slow wakeup time.

  The Stop mode features two sub-modes:
  
  - **Stop in Normal mode (default mode)**
    In this mode, the 1.2 V domain is preserved in nominal leakage mode and the minimum V12 voltage is 1.08 V.
  
  - **Stop in Under-drive mode**
    In this mode, the 1.2 V domain is preserved in reduced leakage mode and V12 voltage is less than 1.08 V. The regulator (in Main or Low-power mode) is in under-drive or low-voltage mode. The Flash memory must be in Deep-power-down mode. The wakeup time is about 100 µs higher than in normal mode.

- **Standby mode**
  This mode achieves very low power consumption with the RC oscillator as a clock source. The internal voltage regulator is switched off so that the entire 1.2 V domain is powered off: CPU and peripherals are stopped. The PLL, the HSI RC and the HSE crystal oscillators are disabled. SRAM and register contents are lost except for registers in the backup domain and the 4-byte backup SRAM when selected. Only RTC and LSE oscillator blocks are powered. The device exits Standby mode when an
external reset (NRST pin), an IWDG reset, a rising edge on the WKUP pin, or an RTC alarm/ wakeup/ tamper/time stamp event occurs.

- **V\textsubscript{BAT}** operation
  It allows to significantly reduced power consumption compared to the Standby mode. This mode is available when the V\textsubscript{BAT} pin powering the Backup domain is connected to an optional standby voltage supplied by a battery or by another source. The V\textsubscript{BAT} domain is preserved (RTC registers, RTC backup register and backup SRAM) and RTC and LSE oscillator blocks powered. The main difference compared to the Standby mode is external interrupts and RTC alarm/events do not exit the device from V\textsubscript{BAT} operation. Increasing V\textsubscript{DD} to reach the minimum threshold does.

**D.1.3 STM32L0 Series**

STM32L0 microcontrollers feature up to 8 power modes, including 7 low-power modes to achieve the best compromise between low-power consumption, short startup time and available wakeup sources:

- **Run mode**
  This mode offers the highest performance using HSE/HSI clock sources. The CPU can run up to 32 MHz and the voltage regulator is enabled.

- **Sleep mode**
  This mode uses HSE or HSI as system clock sources. The voltage regulator is enabled and only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.

- **Low-power run mode**
  This mode uses the internal regulator in low-power mode and the multispeed internal (MSI) RC oscillator set to the minimum clock frequency (131 kHz). In Low-power run mode, the clock frequency and the number of enabled peripherals are both limited.

- **Low-power sleep mode**
  This mode is achieved by entering Sleep mode with the internal voltage regulator in low-power mode. Both the clock frequency and the number of enabled peripherals are limited. Event or interrupt can revert the system to Run mode with regulator on.

- **Stop mode with RTC**
  The Stop mode achieves the lowest power consumption with, while retaining the RAM, register contents and real time clock. The voltage regulator is in low-power mode. LSE or LSI is still running. All clocks in the V\textsubscript{CORE} domain are stopped, the PLL, MSI RC, HSE crystal and HSI RC oscillators are disabled.
  Some peripherals featuring wakeup capability can enable the HSI RC during Stop mode to detect their wakeup condition. The device can be woken up from Stop mode by any of the EXTI line, in 3.5 μs, and the processor can serve the interrupt or resume the code.

- **Stop mode without RTC**
  This mode is identical to “Stop mode with RTC “, except for the RTC clock which is stopped here.

- **Standby mode with RTC**
  The Standby mode achieves the lowest power consumption with the real time clock running. The internal voltage regulator is switched off so that the entire V\textsubscript{CORE} domain
is powered off. The PLL, MSI RC, HSE crystal and HSI RC oscillators are also switched off. The LSE or LSI is still running.

After entering Standby mode, the RAM and register contents are lost except for registers in the Standby circuitry (wakeup logic, IWDG, RTC, LSI, LSE Crystal 32 KHz oscillator, RCC_CSR register).

The device exits Standby mode in 60 µs when an external reset (NRST pin), an IWDG reset, a rising edge on one of the three WKUP pins, RTC alarm (Alarm A or Alarm B), RTC tamper event, RTC timestamp event or RTC Wakeup event occurs.

- **Standby mode without RTC**
  This mode is identical to Standby mode with RTC, except that the RTC, LSE and LSI clocks are stopped.
  The device exits Standby mode in 60 µs when an external reset (NRST pin) or a rising edge on one of the three WKUP pin occurs.

*Note: The RTC, the IWDG, and the corresponding clock sources are not stopped automatically by entering Stop or Standby mode. The LCD is not stopped automatically by entering Stop mode.*

### D.2 Power consumption ranges

STM32 MCUs power consumption can be further optimized thanks to the dynamic voltage scaling feature: the main internal regulator output voltage V12 that supplies the logic (CPU, digital peripherals, SRAM and Flash memory) can be adjusted by software by selecting a power range (STM32L1 and STM32L0) or power scale (STM32 F4).

Power consumption range definitions are provided below (refer to STM32 MCU datasheets for full details).

#### D.2.1 STM32L1 Series features three $V_{CORE}$ ranges

- **High Performance Range 1** ($V_{DD}$ range limited to 2.0-3.6 V), with the CPU running at up to 32 MHz
  The voltage regulator outputs a 1.8 V voltage (typical) as long as the $V_{DD}$ input voltage is above 2.0 V. Flash program and erase operations can be performed.

- **Medium Performance Range 2** (full $V_{DD}$ range), with a maximum CPU frequency of 16 MHz
  At 1.5 V, the Flash memory is still functional but with medium read access time. Flash program and erase operations are still possible.

- **Low Performance Range 3** (full $V_{DD}$ range), with a maximum CPU frequency limited to 4 MHz (generated only with the multispeed internal RC oscillator clock source)
  At 1.2 V, the Flash memory is still functional but with slow read access time. Flash Program and erase operations are no longer available.
D.2.2 STM32F4 Series features several $V_{\text{CORE}}$ scales

The scale can be modified only when the PLL is OFF and when HSI or HSE is selected as system clock source.

- **Scale 1** (V12 voltage range limited to 1.26-1.40 V), default mode at reset
  
  HCLK frequency range = 144 MHz to 168 MHz (180 MHz with over-drive).
  
  This is the default mode at reset.

- **Scale 2** (V12 voltage range limited to 1.20 to 1.32 V)
  
  HCLK frequency range is up to 144 MHz (168 MHz with over-drive)

- **Scale 3** (V12 voltage range limited to 1.08 to 1.20 V), default mode when exiting Stop mode
  
  HCLK frequency ≤ 120 MHz.

The voltage scaling is adjusted to $f_{\text{HCLK}}$ frequency as follows:

- **STM32F429x/39x MCUs**:
  
  - Scale 1: up to 168 MHz (up to 180 MHz with over-drive)
  
  - Scale 2: from 120 to 144 MHz (up to 168 MHz with over-drive)
  
  - Scale 3: up to 120 MHz.

- **STM32F401x MCUs**:
  
  No Scale 1
  
  - Scale 2: from 60 to 84 MHz
  
  - Scale 3: up to 60 MHz.

- **STM32F40x/41x MCUs**:
  
  - Scale 1: up to 168 MHz
  
  - Scale 2: up to 144 MHz

D.2.3 STM32L0 Series features three $V_{\text{CORE}}$ ranges

- Range 1 ($V_{\text{DD}}$ range limited to 1.71 to 3.6 V), with CPU running at a frequency up to 32 MHz

- Range 2 (full $V_{\text{DD}}$ range), with a maximum CPU frequency of 16 MHz

- Range 3 (full $V_{\text{DD}}$ range), with a maximum CPU frequency limited to 4.2 MHz.
Appendix E   STM32Cube embedded software packages

Along with STM32CubeMX C code generator, embedded software packages are part of STM32Cube initiative (refer to DB2164 databrief): these packages include a low-level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards (see Figure 352). This set of components is highly portable across the STM32 Series. The packages are fully compatible with STM32CubeMX generated C code.

Figure 352. STM32Cube Embedded Software package

Note: STM32CubeF0, STM32CubeF1, STM32CubeF2, STM32CubeF3, STM32CubeF4, STM32CubeL0 and STM32CubeL1 embedded software packages are available on st.com. They are based on STM32Cube release v1.1 (other Series will be introduced progressively) and include the embedded software libraries used by STM32CubeMX for initialization C code generation.

The user should use STM32CubeMX to generate the initialization C code and the examples provided in the package to get started with STM32 application development.
## Revision history

Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>17-Feb-2014</td>
<td>1</td>
<td>4.1</td>
<td>Initial release. Added support of STM32CubeF2 and STM32F2 Series in cover page, Section 2.2: Key features, Section 5.14.1: Peripherals and Middleware Configuration window, and Appendix E: STM32Cube embedded software packages. Updated Section 10.1: Creating a new STM32CubeMX Project, Section 10.2: Configuring the MCU pinout, Section 10.6: Configuring the MCU initialization parameters. Section &quot;Generating GPIO initialization C code move to Section 8: Tutorial 3- Generating GPIO initialization C code (STM32F1 Series only) and content updated. Added Section 20.4: Why do I get the error &quot;Java 7 update 45&quot; when installing &quot;Java 7 update 45&quot; or a more recent version of the JRE?.</td>
</tr>
<tr>
<td>04-Apr-2014</td>
<td>2</td>
<td>4.2</td>
<td>Added support of STM32CubeL0 and STM32L0 Series in cover page, Section 2.2: Key features, Section 2.3: Rules and limitations and Section 5.14.1: Peripherals and Middleware Configuration window Added board selection in Table 13: File menu functions, Section 5.7.3: Pinout menu and Section 4.2: New Project window. Updated Table 15: Pinout menu. Updated Figure 120: Power Consumption Calculator default view and added battery selection in Section 5.1.1: Building a power consumption sequence. Updated note in Section 5.1: Power Consumption Calculator view Updated Section 10.1: Creating a new STM32CubeMX Project. Added Section 20.5: Why does the RTC multiplexer remain inactive on the Clock tree view?, Section 20.6: How can I select LSE and HSE as clock source and change the frequency?, and Section 20.7: Why STM32CubeMX does not allow me to configure PC13, PC14, PC15 and PI8 as outputs when one of them is already configured as an output?.</td>
</tr>
<tr>
<td>24-Apr-2014</td>
<td>3</td>
<td>4.3</td>
<td>Added support of STM32CubeF2 and STM32F2 Series in cover page, Section 2.2: Key features, Section 5.14.1: Peripherals and Middleware Configuration window Added board selection in Table 13: File menu functions, Section 5.7.3: Pinout menu and Section 4.2: New Project window. Updated Table 15: Pinout menu. Updated Figure 120: Power Consumption Calculator default view and added battery selection in Section 5.1.1: Building a power consumption sequence. Updated note in Section 5.1: Power Consumption Calculator view Updated Section 10.1: Creating a new STM32CubeMX Project. Added Section 20.5: Why does the RTC multiplexer remain inactive on the Clock tree view?, Section 20.6: How can I select LSE and HSE as clock source and change the frequency?, and Section 20.7: Why STM32CubeMX does not allow me to configure PC13, PC14, PC15 and PI8 as outputs when one of them is already configured as an output?.</td>
</tr>
</tbody>
</table>
Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Jun-2014</td>
<td>4</td>
<td>4.4</td>
<td>Added support of STM32CubeF0, STM32CubeF3, STM32F0 and STM32F3 Series in cover page, Section 2.2: Key features, Section 2.3: Rules and limitations, Added board selection capability and pin locking capability in Section 2.2: Key features, Table 2: Home page shortcuts, Section 4.2: New Project window, Section 5.7: Toolbar and menus, Section 4.10: Set unused / Reset used GPIOs windows, Section 4.8: Project Manager view, and Section 5.15: Pinout view. Added Section 5.15.1: Pinning and labeling signals on pins. Updated Section 5.16: Configuration view and Section 4.7: Clock Configuration view and Section 5.1: Power Consumption Calculator view. Updated Figure 36: STM32CubeMX Main window upon MCU selection, Figure 90: Project Settings window, Figure 119: About window, Figure 140: STM32CubeMX Pinout view, Figure 120: Chip view, Figure 121: Power Consumption Calculator default view, Figure 121: Battery selection, Figure 87: Building a power consumption sequence, Figure 123: Power consumption sequence: New Step default view, Figure 130: Power Consumption Calculator view after sequence building, Figure 131: Sequence table management functions, Figure 88: PCC Edit Step window, Figure 83: Power consumption sequence: new step configured (STM32F4 example), Figure 128: ADC selected in Pinout view, Figure 129: Power Consumption Calculator Step configuration window: ADC enabled using import pinout, Figure 133: Description of the Results area, Figure 100: Peripheral power consumption tooltip, Figure 246: Power Consumption Calculation example, Figure 155: Sequence table and Figure 156: Power Consumption Calculation results. Updated Figure 142: STM32CubeMX Configuration view and Figure 39: STM32CubeMX Configuration view - STM32F1 Series titles. Added STM32L1 in Section 5.1: Power Consumption Calculator view. Removed Figure Add a new step using the PCC panel from Section 8.1.1: Adding a step. Removed Figure Add a new step to the sequence from Section 5.1.2: Configuring a step in the power sequence. Updated Section 8.2: Reviewing results. Updated appendix B.3.4: FatFs and Appendix D: STM32 microcontrollers power consumption parameters. Added Appendix D.1.3: STM32L0 Series and D.2.3: STM32L0 Series features three VCORE ranges.</td>
</tr>
</tbody>
</table>
### Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Sep-2014</td>
<td>5</td>
<td>4.5</td>
<td>Added support of STM32CubeL1 Series in cover page, Section 2.2: Key features, Section 2.3: Rules and limitations. Updated Section 3.2.3: Uninstalling STM32CubeMX standalone version. Added off-line updates in Section 3.4: Getting updates using STM32CubeMX, modified Figure 8: Embedded Software Packages Manager window, and Section 3.4.2: Installing STM32 MCU packages. Updated Section 4: STM32CubeMX user interface introduction, Table 2: Home page shortcuts and Section 4.2: New Project window. Added Figure 30: New Project window - Board selector. Updated Figure 98: Project Settings code generator. Modified step 3 in Section 4.8: Project Manager view. Updated Figure 39: STM32CubeMX Configuration view - STM32F1 Series. Added STM32L1 in Section 5.14.1: Peripherals and Middleware Configuration window. Updated Figure 58: GPIO Configuration window - GPIO selection; Section 4.4.11: GPIO Configuration window and Figure 63: DMA MemToMem configuration. Updated introduction of Section 4.7: Clock Configuration view. Updated Section 4.7.1: Clock tree configuration functions and Section 4.7.2: Recommendations, Section 5.1: Power Consumption Calculator view, Figure 123: Power consumption sequence: New Step default view, Figure 130: Power Consumption Calculator view after sequence building, Figure 83: Power consumption sequence: new step configured (STM32F4 example), and Figure 129: Power Consumption Calculator Step configuration window: ADC enabled using import pinout. Added Figure 132: Power Consumption: Peripherals consumption chart and updated Figure 100: Peripheral power consumption tooltip. Updated Section 5.1.4: Power sequence step parameters glossary. Updated Section 6: STM32CubeMX C Code generation overview. Updated Section 10.1: Creating a new STM32CubeMX Project and Section 10.2: Configuring the MCU pinout. Added Section 11: Tutorial 2 - Example of FatFs on an SD card using STM32429i-EVAL evaluation board and updated Section 8: Tutorial 3- Generating GPIO initialization C code (STM32F1 Series only). Updated Section 5.1.2: Configuring a step in the power sequence.</td>
</tr>
</tbody>
</table>
Complete project generation, power consumption calculation and clock tree configuration now available on all STM32 Series.
Updated Section 2.2: Key features and Section 2.3: Rules and limitations.
Updated Eclipse IDEs in Section 3.1.3: Software requirements.
Updated Figure 6: Updater Settings window, Figure 8: Embedded Software Packages Manager window and Figure 30: New Project window - Board selector, Updated Section 4.8: Project Manager view and Section 4.11: Update Manager windows.
Updated Figure 119: About window.
Removed Figure STM32CubeMX Configuration view - STM32F1 Series.
Updated Table 17: STM32CubeMX Chip view - Icons and color scheme.
Updated Section 5.14.1: Peripherals and Middleware Configuration window.
Updated Figure 61: Adding a new DMA request and Figure 63: DMA MemToMem configuration.
Updated Section 4.7.1: Clock tree configuration functions.
Updated Figure 121: Battery selection, Figure 87: Building a power consumption sequence, Figure 88: PCC Edit Step window.
Added Section 6.3: Custom code generation.
Updated Figure 200: Clock tree view and Figure 205: Pinout & Configuration view.
Updated peripheral configuration sequence and Figure 207: Timer 3 configuration window in Section 10.6.2: Configuring the peripherals.
Removed Tutorial 3: Generating GPIO initialization C code (STM32F1 Series only).
Updated Figure 211: GPIO mode configuration.
Updated Figure 246: Power Consumption Calculation example and Figure 155: Sequence table.
Updated Appendix A.1: Block consistency, A.2: Block interdependency and A.3: One block = one peripheral mode.
Appendix A.4: Block remapping (STM32F10x only): updated Section : Example.
Appendix A.6: Block shifting (only for STM32F10x and when “Keep Current Signals placement” is unchecked): updated Section : Example.
Updated Appendix A.8: Mapping a function individually.
Updated Appendix B.3.1: Overview.
Updated Appendix D.1.3: STM32L0 Series.

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 19-Jan-2015| 6        | 4.6                        | Complete project generation, power consumption calculation and clock tree configuration now available on all STM32 Series.
Updated Section 2.2: Key features and Section 2.3: Rules and limitations.
Updated Eclipse IDEs in Section 3.1.3: Software requirements.
Updated Figure 6: Updater Settings window, Figure 8: Embedded Software Packages Manager window and Figure 30: New Project window - Board selector, Updated Section 4.8: Project Manager view and Section 4.11: Update Manager windows.
Updated Figure 119: About window.
Removed Figure STM32CubeMX Configuration view - STM32F1 Series.
Updated Table 17: STM32CubeMX Chip view - Icons and color scheme.
Updated Section 5.14.1: Peripherals and Middleware Configuration window.
Updated Figure 61: Adding a new DMA request and Figure 63: DMA MemToMem configuration.
Updated Section 4.7.1: Clock tree configuration functions.
Updated Figure 121: Battery selection, Figure 87: Building a power consumption sequence, Figure 88: PCC Edit Step window.
Added Section 6.3: Custom code generation.
Updated Figure 200: Clock tree view and Figure 205: Pinout & Configuration view.
Updated peripheral configuration sequence and Figure 207: Timer 3 configuration window in Section 10.6.2: Configuring the peripherals.
Removed Tutorial 3: Generating GPIO initialization C code (STM32F1 Series only).
Updated Figure 211: GPIO mode configuration.
Updated Figure 246: Power Consumption Calculation example and Figure 155: Sequence table.
Updated Appendix A.1: Block consistency, A.2: Block interdependency and A.3: One block = one peripheral mode.
Appendix A.4: Block remapping (STM32F10x only): updated Section : Example.
Appendix A.6: Block shifting (only for STM32F10x and when “Keep Current Signals placement” is unchecked): updated Section : Example.
Updated Appendix A.8: Mapping a function individually.
Updated Appendix B.3.1: Overview.
Updated Appendix D.1.3: STM32L0 Series.
<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Mar-2015</td>
<td>7</td>
<td>4.7</td>
<td><strong>Section 2.2: Key features</strong>: removed Pinout initialization C code generation for STM32F1 Series from; updated Complete project generation. Updated Figure 8: Embedded Software Packages Manager window, Figure 30: New Project window - Board selector. Updated IDE list in Section 4.8: Project Manager view and modified Figure 90: Project Settings window. Updated Section 4.7.1: Clock tree configuration functions. Updated Figure 86: STM32F469NIHx clock tree configuration view. Section 5.1: Power Consumption Calculator view: added transition checker option. Updated Figure 120: Power Consumption Calculator default view, Figure 121: Battery selection and Figure 87: Building a power consumption sequence. Added Figure 124: Enabling the transition checker option on an already configured sequence - All transitions valid, Figure 125: Enabling the transition checker option on an already configured sequence - At least one transition invalid and Figure 126: Transition checker option - Show log. Updated Figure 130: Power Consumption Calculator view after sequence building. Updated Section : Managing sequence steps, Section : Managing the whole sequence (load, save and compare). Updated Figure 88: PCC Edit Step window and Figure 133: Description of the Results area. Updated Figure 246: Power Consumption Calculation example, Figure 155: Sequence table, Figure 156: Power Consumption Calculation results and Figure 158: Power consumption results - IP consumption chart. Updated Appendix B.3.1: Overview and B.3.5: FreeRTOS.</td>
</tr>
<tr>
<td>28-May-2015</td>
<td>8</td>
<td>4.8</td>
<td>Added <strong>Section 3.2.2: Installing STM32CubeMX from command line</strong> and <strong>Section 3.3.2: Running STM32CubeMX in command-line mode</strong>. Added STLM32F7 and STM32L4 microcontroller Series. Added Import project feature. Added Import function in Table 13: File menu functions. Added Section 4.9: Import Project window. Updated Figure 123: Power consumption sequence: New Step default view, Figure 88: PCC Edit Step window, Figure 83: Power consumption sequence: new step configured (STM32F4 example), Figure 129: Power Consumption Calculator Step configuration window: ADC enabled using import pinout and Figure 87: Peripheral power consumption tooltip. Updated command line to run STM32CubeMX in Section 3.3.2: Running STM32CubeMX in command-line mode. Updated note in Section 5.16: Configuration view. Added new clock tree configuration functions in Section 4.7.1. Updated Figure 213: Middleware tooltip. Modified code example in Appendix B.1: STM32CubeMX generated C code and user sections. Updated Appendix B.3.1: Overview. Updated generated .h files in Appendix B.3.4: FatFs.</td>
</tr>
<tr>
<td>09-Jul-2015</td>
<td>9</td>
<td>4.9</td>
<td>Added STLM32F7 and STM32L4 microcontroller Series. Added Import project feature. Added Import function in Table 13: File menu functions. Added Section 4.9: Import Project window. Updated Figure 123: Power consumption sequence: New Step default view, Figure 88: PCC Edit Step window, Figure 83: Power consumption sequence: new step configured (STM32F4 example), Figure 129: Power Consumption Calculator Step configuration window: ADC enabled using import pinout and Figure 87: Peripheral power consumption tooltip. Updated command line to run STM32CubeMX in Section 3.3.2: Running STM32CubeMX in command-line mode. Updated note in Section 5.16: Configuration view. Added new clock tree configuration functions in Section 4.7.1. Updated Figure 213: Middleware tooltip. Modified code example in Appendix B.1: STM32CubeMX generated C code and user sections. Updated Appendix B.3.1: Overview. Updated generated .h files in Appendix B.3.4: FatFs.</td>
</tr>
<tr>
<td>Date</td>
<td>Revision</td>
<td>STM32CubeMX release number</td>
<td>Changes</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>27-Aug-2015</td>
<td>10</td>
<td>4.10</td>
<td>Replace UM1742 by UM1940 in Section : Introduction. Updated command line to run STM32CubeMX in command-line mode in Section 3.3.2: Running STM32CubeMX in command-line mode. Modified Table 1: Command line summary. Updated board selection in Section 4.2: New Project window. Updated Section 5.16: Configuration view overview. Updated Section 5.14.1: Peripherals and Middleware Configuration window, Section 4.4.11: GPIO Configuration window and Section 4.4.12: DMA Configuration window. Added Section 4.4.10: User Constants configuration window. Updated Section 4.7: Clock Configuration view and added reserve path. Updated Section 10.1: Creating a new STM32CubeMX Project, Section 10.5: Configuring the MCU clock tree, Section 10.6: Configuring the MCU initialization parameters, Section 10.7.2: Downloading firmware package and generating the C code, Section 10.8: Building and updating the C code project. Added Section 10.9: Switching to another MCU. Updated Section 11: Tutorial 2 - Example of FatFs on an SD card using STM32F429I-EVAL evaluation board and replaced STM32F429I-EVAL by STM32429I-EVAL.</td>
</tr>
<tr>
<td>16-Oct-2015</td>
<td>11</td>
<td>4.11</td>
<td>Updated Figure 8: Embedded Software Packages Manager window and Section 3.4.6: Checking for updates. Character string constant supported in Section 4.4.10: User Constants configuration window. Updated Section 4.7: Clock Configuration view. Updated Section 5.1: Power Consumption Calculator view. Modified Figure 246: Power Consumption Calculation example. Updated Section 12: Tutorial 3 - Using the Power Consumption Calculator to optimize the embedded application consumption and more. Added Eclipse Mars in Section 3.1.3: Software requirements</td>
</tr>
<tr>
<td>03-Dec-2015</td>
<td>12</td>
<td>4.12</td>
<td>Code generation options now supported by the Project settings menu. Updated Section 3.1.3: Software requirements. Added project settings in Section 4.9: Import Project window. Updated Figure 103: Automatic project import; modified Manual project import step and updated Figure 104: Manual project import and Figure 105: Import Project menu - Try import with errors; modified third step of the import sequence. Updated Figure 83: Clock Tree configuration view with errors. Added mxconstants.h in Section 6.1: STM32Cube code generation using only HAL drivers (default mode). Updated Figure 246: Power Consumption Calculation example to Figure 255: Step 10 optimization. Updated Figure 256: Power sequence results after optimizations.</td>
</tr>
</tbody>
</table>
Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 03-Feb-2016| 13       | 4.13                       | Updated Section 2.2: Key features:  
– Information related to .ioc files.  
– Clock tree configuration  
– Automatic updates of STM32CubeMX and STM32Cube.  
Updated limitation related to STM32CubeMX C code generation in Section 2.3: Rules and limitations.  
Added Linux in Section 3.1.1: Supported operating systems and architectures. Updated Java Run Time Environment release number in Section 3.1.3: Software requirements.  
Updated Section 3.2.1: Installing STM32CubeMX standalone version, Section 3.2.3: Uninstalling STM32CubeMX standalone version and Section 3.3.1: Downloading STM32CubeMX plug-in installation package.  
Updated Section 3.3.1: Running STM32CubeMX as standalone application.  
Updated Section 4.8: Project Manager view and Section 4.11: Update Manager windows.  
Updated Section 5.15.1: Pinning and labeling signals on pins.  
Added Section 4.4.16: Setting HAL timebase source.  
Updated Figure 143: Configuration window tabs for GPIO, DMA and NVIC settings (STM32F4 Series).  
Added note related to GPIO configuration in output mode in Section 4.4.11: GPIO Configuration window; updated Figure 58: GPIO Configuration window - GPIO selection.  
Modified Figure 120: Power Consumption Calculator default view, Figure 86: Building a power consumption sequence, Figure 122: Step management functions, Figure 124: Enabling the transition checker option on an already configured sequence - All transitions valid, Figure 125: Enabling the transition checker option on an already configured sequence - At least one transition invalid.  
Added import pinout button icon in Section: Importing pinout.  
Added Section: Selecting/deselecting all peripherals. Modified Figure 130: Power Consumption Calculator view after sequence building, Updated Section: Managing the whole sequence (load, save and compare). Updated Figure 133: Description of the Results area and Figure 100: Peripheral power consumption tooltip.  
Updated Figure 246: Power Consumption Calculation example and Figure 248: Sequence table.  
Updated Section 6.3: Custom code generation.  
Updated Figure 192: Pinout view with MCUs selection and Figure 193: Pinout view without MCUs selection window in Section 10.1: Creating a new STM32CubeMX Project.  
Updated Section 10.6.2: Configuring the peripherals.  
Updated Figure 218: Project Settings and toolchain selection and Figure 219: Project Manager menu - Code Generator tab in Section 10.7.1: Setting project options, and Figure 220: Missing firmware package warning message in Section 10.7.2: Downloading firmware package and generating the C code.  
|
## Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-Mar-2016</td>
<td>14</td>
<td>4.14</td>
<td>Upgraded STM32CubeMX released number to 4.14.0. Added import of previously saved projects and generation of user files from templates in Section 2.2: Key features. Added MacOS in Section 3.1.1: Supported operating systems and architectures, Section 3.2.1: Installing STM32CubeMX standalone version, Section 3.2.3: Uninstalling STM32CubeMX standalone version and Section 3.4.3: Running STM32CubeMX plug-in from Eclipse IDE. Added command lines allowing the generation of user files from templates in Section 3.3.2: Running STM32CubeMX in command-line mode. Updated new library installation sequence in Section 3.4.1: Updater configuration. Updated command lines in Section 3.3.2: Running STM32CubeMX in command-line mode. Table 16: Window menu. Updated Section 5.7: Output windows. Updated Figure 90: Project Settings window and Section 4.8.1: Project tab. Updated Figure 78: NVIC settings when using SysTick as HAL timebase, no FreeRTOS and Figure 79: NVIC settings when using FreeRTOS and SysTick as HAL timebase in Section 4.4.16: Setting HAL timebase source. Updated Figure 49: User Constants tab and Figure 50: Extract of the generated main.h file in Section 4.4.10: User Constants configuration window. Section 4.4.11: GPIO Configuration window: updated Figure 58: GPIO Configuration window - GPIO selection, Figure 59: GPIO configuration grouped by peripheral and Figure 60: Multiple Pins Configuration. Updated Section 4.4.13: NVIC Configuration window.</td>
</tr>
<tr>
<td>18-May-2016</td>
<td>15</td>
<td>4.15</td>
<td>Import project function is no more limited to MCUs of the same Series (see Section 2.2: Key features, Section 5.7.1: File menu and Section 4.9: Import Project window ). Updated command lines in Section 3.3.2: Running STM32CubeMX in command-line mode. Table 1: Command line summary: modified all examples related to config commands as well as set dest_path &lt;path&gt; example. Added caution note for Load Project menu in Table 13: File menu functions. Updated Generate Code menu description in Table 14: Project menu. Updated Set unused GPIOs menu in Table 15: Pinout menu. Added case where FreeRTOS in enabled in Section 4.4.14: FreeRTOS configuration panel. Updated Appendix B.3.5: FreeRTOS and B.3.6: LwIP.</td>
</tr>
</tbody>
</table>
## Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>23-Sep-2016</td>
<td>16</td>
<td>4.17</td>
<td>Replaced <em>mxconstants.h</em> by <em>main.h</em> in the whole document. Updated <em>Introduction</em>, <em>Section 3.1.1: Supported operating systems and architectures</em> and <em>Section 3.1.3: Software requirements</em>. Added <em>Section 3.4.3: Installing STM32 MCU package patches</em>. Updated Load project description in <em>Table 2: Home page shortcuts</em>. Updated Clear Pinouts function in <em>Table 15: Pinout menu</em>. Updated <em>Section 4.8.3: Advanced Settings tab</em> to add Low Layer driver. Added <em>No check</em> and <em>Decimal and hexadecimal check</em> options in <em>Table 17: Peripheral and Middleware Configuration window buttons and tooltips</em>. Updated <em>Section: Tasks and Queues Tab</em> and <em>Figure 73: FreeRTOS Heap usage</em>. Updated <em>Figure 58: GPIO Configuration window - GPIO selection</em>. Replaced PCC by Power Consumption Calculator in the whole document. Added <em>Section 6.2: STM32Cube code generation using Low Layer drivers</em>; updated <em>Table 17: LL versus HAL: STM32CubeMX generated source files</em> and <em>Table 18: LL versus HAL: STM32CubeMX generated functions and function calls</em>. Updated <em>Figure 317: Pinout view - Enabling the RTC</em>. Added <em>Section 13: Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board</em>. Added correspondence between STM32CubeMX release number and document revision.</td>
</tr>
<tr>
<td>21-Nov-2016</td>
<td>17</td>
<td>4.18</td>
<td>Removed Windows XP and added Windows 10 in <em>Section 3.1.3: Software requirements</em>. Updated <em>Section 3.2.3: Uninstalling STM32CubeMX standalone version</em>. Added setDriver command line in <em>Table 1: Command line summary</em>. Added List pinout compatible MCUs feature: – Updated <em>Table 15: Pinout menu</em>. – Added <em>Section 14: Tutorial 5: Exporting current project configuration to a compatible MCU</em> Added Firmware location selection option in <em>Section 4.8.1: Project tab</em> and <em>Figure 90: Project Settings window</em>. Added Restore Default feature: – Updated <em>Table 8: Peripheral and Middleware Configuration window buttons and tooltips</em> – Updated <em>Figure 51: Using constants for peripheral parameter settings</em>.</td>
</tr>
</tbody>
</table>
### Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-Jan-2017</td>
<td>18</td>
<td>4.19</td>
<td>Project import no more limited to microcontrollers belonging to the same Series: updated <em>Introduction, Figure 103: Automatic project import, Figure 104: Manual project import, Figure 105: Import Project menu - Try import with errors and Figure 106: Import Project menu - Successful import after adjustments.</em> Modified Appendix B.3.4: FatFs, B.3.5: FreeRTOS and B.3.6: LwIP. Added Appendix B.3.7: Libjpeg.</td>
</tr>
</tbody>
</table>
| 02-Mar-2017 | 19       | 4.20                        | *Table 17: STM32CubeMX Chip view - Icons and color scheme:*  
  – Updated list of alternate function example.  
  – Updated example and description corresponding to function mapping on a pin.  
  – Added example and description for analog signals sharing the same pin.  
Updated Figure 87: Peripheral Configuration window (STM32F4 Series), Figure 49: User Constants tab, Figure 55: Deleting a user constant used for peripheral configuration - Consequence on peripheral configuration, Figure 56: Searching for a name in a user constant list and Figure 57: Searching for a value in a user constant list.  
Added Section 5.1.6: SMPS feature.  
Added Section 6.4: Additional settings for C project generation.  
Added STM32CubeF4 to the list of packages that include Libjpeg in Appendix B.3.7: Libjpeg. |
| 05-May-2017 | 20       | 4.21                        | Minor modifications in Section 1: STM32Cube overview.  
Updated Figure 25: New Project window - MCU selector and Figure 90: Project Settings window.  
Updated description of Project settings in Section 4.8.1: Project tab.  
Updated Figure 101: Advanced Settings window.  
In Appendix B.3.7: Libjpeg, added STM32CubeF2 and STM32CubeH7 in the list of software packages in which Libjpeg is embedded.  
Modified Figure 352: STM32Cube Embedded Software package look-and-feel. |
<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-Jul-2017</td>
<td>21</td>
<td>4.22</td>
<td>Added STM32H7 to the list of supported STM32 Series. Added MCU data and documentation refresh capability in Section 3.4: Getting updates using STM32CubeMX and updated Figure 6: Updater Settings window. Added capability to identify close MCUs in Section 4.2: New Project window, updated Figure 25: New Project window - MCU selector, added Figure 28: New Project window - MCU list with close function and Figure 29: New Project window - List showing close MCUs., updated Figure 191: MCU selection. Updated Figure 36: STM32CubeMX Main window upon MCU selection. Added Rotate clockwise/Counter clockwise and Top/Bottom view in Table 15: Pinout menu. Added Section 4.1.4: Social links. Updated Figure 141: Configuring the SMPS mode for each step. Updated Section 6.2: STM32Cube code generation using Low Layer drivers. Updated Figure 218: Project Settings and toolchain selection.</td>
</tr>
<tr>
<td>05-Sep-2017</td>
<td>22</td>
<td>4.22.1</td>
<td>Added STM32L4+ Series in Introduction, Section 5.1: Power Consumption Calculator view and Section 6.2: STM32Cube code generation using Low Layer drivers. Added guidelines to run STM32CubeMX on MacOS in Section 3.3.1: Running STM32CubeMX as standalone application. Removed MacOS from Section 3.4.3: Running STM32CubeMX plug-in from Eclipse IDE. Added Section 20.8: Ethernet Configuration: why cannot I specify DP83848 or LAN8742A in some cases?</td>
</tr>
</tbody>
</table>
Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-Jan-2018</td>
<td>24</td>
<td>4.24</td>
<td>Replaced “STM32Cube firmware package” by “STM32Cube MCU package”. Updated Section 1: STM32Cube overview. Updated MacOS in Section 3.1.1: Supported operating systems and architectures. Updated Eclipse requirements in Section 3.1.3: Software requirements. Section 3.4: Getting updates using STM32CubeMX: – updated section introduction – updated Figure 13: Connection Parameters tab - No proxy – Section 3.4.2 renamed into “Installing STM32 MCU packages” and updated. – renamed Section 3.4.3 into “Installing STM32 MCU package patches” – added Section 3.4.4: Installing embedded software packs – updated Section 3.4.6: Checking for updates Updated Figure 30: New Project window - Board selector. Updated Figure 37: STM32CubeMX Main window upon board selection (peripherals not initialized) and introductory sentence. Updated Figure 38: STM32CubeMX Main window upon board selection (peripherals initialized with default configuration) and introductory sentence. Added “Select additional software components” menu in Table 14: Project menu. “Install new libraries” menu renamed “Manage embedded software packages” and corresponding description updated in Table 17: Help menu. Updated Section 3.4.5: Removing already installed embedded software packages. Updated Section 4.11: Update Manager windows Added Section 4.12: Additional software component selection window. Added pin stacking function in Table 17: STM32CubeMX Chip view - Icons and color scheme. Section 6.2: STM32Cube code generation using Low Layer drivers: added STM32F0, STM32F3, STM32L0 in the list of product Series supporting low-level drivers. Section 11: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board: updated Figure 238: Board selection and modified step 6 of the sequence for generating a project and running tutorial 2. Section 13: Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board: updated Figure 257: Selecting NUCLEO_L053R8 board. Added Section 15: Tutorial 6 – Adding embedded software packs to user projects.</td>
</tr>
</tbody>
</table>
### Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-Jan-2018</td>
<td>24 (cont’d)</td>
<td>4.24</td>
<td>Added Appendix B.3.9: TouchSensing and B.3.10: PDM2PCM. Section 4.4.13: NVIC Configuration window/Default initialization sequence of interrupts: changed color corresponding to interrupt enabling code from green to black bold.</td>
</tr>
<tr>
<td>07-Mar-2018</td>
<td>25</td>
<td>4.25</td>
<td>Updated Introduction, Section 1: STM32Cube overview, Section 2.3: Rules and limitations, Section 3.2.1: Installing STM32CubeMX standalone version, Section 4: STM32CubeMX user interface, Section 4.8.1: Project tab and Section 5.13.1: Peripheral and Middleware tree panel. Minor text edits across the whole document. Updated Table 13: File menu functions and Table 11: Relations between power over-drive and HCLK frequency. Updated Figure 25: New Project window - MCU selector, Figure 26: Enabling graphics choice in MCU selector, Figure 90: Project Settings window, Figure 95: Selecting a different firmware location, Figure 74: Enabling STemWin framework, Figure 116: Configuration view for Graphics, Figure 318: Pinout view - Enabling LSE and HSE clocks and Figure 319: Pinout view - Setting LSE/HSE clock frequency. Added Export to Excel feature, Show favorite MCUs feature and Section 4.4.15: Graphics frameworks and simulator. Added Section 17: Tutorial 8 – Using STemWin Graphics framework, Section 18: Tutorial 9: Using STM32CubeMX Graphics simulator and their subsections. Added Section B.3.11: Graphics.</td>
</tr>
<tr>
<td>Date</td>
<td>Revision</td>
<td>STM32CubeMX release number</td>
<td>Changes</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>05-Sep-2018</td>
<td>26</td>
<td>4.27</td>
<td>Updated STM32Cube logo on cover page. Replaced STMCube™ by STM32Cube™ in the whole document. Updated Section 1: STM32Cube overview. Updated Figure 1: Overview of STM32CubeMX C code generation flow. Updated Section 2.2: Key features to add new features: graphic simulator feature, Support of embedded software packages in CMSIS-Pack format and Contextual Help. Changed Section 3.4 title into “Getting updates using STM32CubeMX”. Suppressed figures Connection Parameters tab - No proxy and Connection Parameters tab - Use System proxy parameters. Updated Figure 9: Managing embedded software packages - Help menu. In Section 3.4.4: Installing embedded software packs, updated step 3f of the embedded software pack installation sequence and added Figure 14: License agreement acceptance. Section 4.2: New Project window: updated Figure 25: New Project window - MCU selector, Figure 27: Marking an MCU as favorite and Figure 30: New Project window - Board selector. Section 5.7.1: File menu: added caution note for New Project in Table 13: File menu functions. Updated Figure 107: Pinout menus (Pinout tab selected) and Figure 108: Pinout menus (Pinout tab not selected). Section 4.8: Project Manager view: – Added note related to project saving (step 3). – Updated Figure 90: Project Settings window – Updated Section 4.8.1: Project tab and Figure 95: Selecting a different firmware location. Added Section 4.12.4: Software component conditions, Contextual help, Section 9: Support of additional software components using CMSIS-Pack standard and Section 16: Tutorial 7 – Using the X-Cube-BLE1 software pack.</td>
</tr>
<tr>
<td>12-Nov-2018</td>
<td>27</td>
<td>4.28</td>
<td>Updated Section 3.4.2: Installing STM32 MCU packages, Section 3.4.4: Installing embedded software packs, Section 3.4.5: Removing already installed embedded software packages, Section 3.4.6: Checking for updates and the figures in it. Updated Section 4: STM32CubeMX user interface, its subsections and the figures and the tables in them. Updated Section 9: Support of additional software components using CMSIS-Pack standard, sections 10.6.1 to 10.6.5, Section 10.7.1: Setting project options, Section 10.7.2: Downloading firmware package and generating the C code, Section 10.8: Building and updating the C code project, Section 10.9: Switching to another MCU, Section 11: Tutorial 2 - Example of FatFs on an SD card using STM32429i-EVAL evaluation board and the figures in it, Section 14: Tutorial 5: Exporting current project configuration to a compatible MCU and the figures in it, Section 15: Tutorial 6 – Adding embedded software packs to user projects and Section 16: Tutorial 7 – Using the X-Cube-BLE1 software pack.</td>
</tr>
</tbody>
</table>

Updated Table 18: LL versus HAL: STM32CubeMX generated functions and function calls.

Removed former Figure 164: Enabling and configuring a CMSIS-Pack software component, Figure 192: FatFs peripheral instances, Figure 213: Project Import status, Figure 254: Saving software component selection as user preferences and Figure 268: Configuring X-Cube-BLE1.

Updated Figure 1: Overview of STM32CubeMX C code generation flow, Figure 3: STM32Cube Installation Wizard, Figure 7: Closing STM32CubeMX perspective, Figure 9: Opening Eclipse plug-in, Figure 10: STM32CubeMX perspective, Figure 134: Overall peripheral consumption, Figure 165: User constant generating define statements, Figure 188: Selecting a CMSIS-Pack software component, Figure 189: Enabling and configuring a CMSIS-Pack software component, Figure 190: Project generated with CMSIS-Pack software component, Figure 191: MCU selection, Figure 192: Pinout view with MCUs selection, Figure 193: Pinout view without MCUs selection window, Figure 195: Timer configuration, Figure 196: Simple pinout configuration, Figure 197: Save Project As window, Figure 198: Generate Project Report - New project creation, Figure 199: Generate Project Report - Project successfully created, Figure 200: Clock tree view, Figure 205: Pinout & Configuration view, Figure 206: Case of Peripheral and Middleware without configuration parameters, Figure 207: Timer 3 configuration window, Figure 208: Timer 3 configuration, Figure 209: Enabling Timer 3 interrupt, Figure 210: GPIO configuration color scheme and tooltip, Figure 211: GPIO mode configuration, Figure 212: DMA parameters configuration window, Figure 213: Middleware tooltip, Figure 214: USB Host configuration, Figure 214: USB Host configuration, Figure 215: FatFs over USB mode enabled, Figure 216: System view with FatFs and USB enabled, Figure 217: FatFs define statements, Figure 218: Project Settings and toolchain selection, Figure 219: Project Manager menu - Code Generator tab, Figure 220: Missing firmware package warning message, Figure 222: Updater settings for download, Figure 223: Updater settings with connection, Figure 224: Downloading the firmware package, Figure 225: Unzipping the firmware package, Figure 226: C code generation completion message, Figure 236: Import Project menu, Figure 266: Project Settings menu, Figure 276: Additional software components enabled for the current project, Figure 277: Pack software components - no configurable parameters, Figure 278: Pack tutorial - project settings, Figure 281: Embedded software packages, Figure 283: Installing Embedded software packages, Figure 284: Starting a new project - selecting the NUCLEO-L053R8 board, Figure 285: Starting a new project - initializing all peripherals, Figure 286: Selecting X-Cube-BLE1 components, Figure 287: Configuring peripherals and GPIOs, Figure 288: Configuring NVIC interrupts, Figure 289: Enabling X-Cube-BLE1, Figure 289: Enabling X-Cube-BLE1, Figure 290: Configuring the SensorDemo project and Figure 304: Graphics simulator user interface.
Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Feb-2019</td>
<td>28</td>
<td>4.29</td>
<td>Updated Introduction, Section 1: STM32Cube overview, Section 2.2: Key features, Section 3.1.3: Software requirements, Section 3.4.2: Installing STM32 MCU packages, Section 4: STM32CubeMX user interface, Resolving pin conflicts, Section 4.4.9: Component Configuration panel, Section 4.7: Clock Configuration view, Section 4.8: Project Manager view, Section 4.8.1: Project tab, Section 4.8.3: Advanced Settings tab, Using the transition checker, Section 8.2: STM32CubeMX Device tree generation, Section 6.3.2: Saving and selecting user templates, .extSettings file example and generated outcomes and Section 10.6.4: Configuring the DMAs. Added Section 4.5: Pinout &amp; Configuration view for STM32MP1 Series, Section 4.5.2: Boot stages configuration, Section 5: STM32CubeMX tools, Section 8: Device tree generation (STM32MP1 Series only), Section B.3.12: STM32WPAN BLE/Thread (STM32WB Series only), Section B.3.13: OpenAmp and RESMGR_UTILITY (STM32MP Series only) and their subsections. Removed former Section 1: General information. Updated Table 2: Home page shortcuts, Table 5: Component list, mode icons and color schemes, Table 6: Pinout menu and shortcuts and title of Table 9: Clock configuration view widgets. Updated Figure 90: Project Settings window, Figure 91: Project folder, Figure 95: Selecting a different firmware location, Figure 103: Automatic project import, Figure 104: Manual project import, Figure 105: Import Project menu - Try import with errors, Figure 106: Import Project menu - Successful import after adjustments, Figure 107: Set unused pins window, Figure 108: Reset used pins window, Figure 119: About window, Figure 183: STM32CubeMX generated DTS – Extract 3, Figure 188: Selecting a CMSIS-Pack software component, Figure 189: Enabling and configuring a CMSIS-Pack software component, Figure 243: FATFS tutorial - Project settings and Figure 244: C code generation completion message.</td>
</tr>
<tr>
<td>16-Apr-2019</td>
<td>29</td>
<td>4.30</td>
<td>Updated Introduction, Section 3.1.3: Software requirements, Section 4.2: New Project window, MCU close selector feature, External clock sources, Importing pinout, Selecting/deselecting all peripherals, Section 4.5: Pinout &amp; Configuration view for STM32MP1 Series, Section 4.12: Additional software component selection window, Section 5.2.1: DDR configuration, Section 6.2: STM32Cube code generation using Low Layer drivers, BLE configuration and Section B.3.13: OpenAmp and RESMGR_UTILITY (STM32MP Series only). Added Section 4.2.1: MCU selector, Section 4.2.2: Board selector, Section 4.2.3: Cross selector, Section 4.6: Pinout &amp; Configuration view for STM32H7 dual-core product lines, Section 5.1.8: Example feature (STM32MP1 and STM32H7 dual-core only) and Section 7: Code generation for dual-core MCUs (STM32H7 dual-core product lines only). Removed former Section 3.3: Installing STM32CubeMX plug-in version and its subsections, and former Section 3.4.3: Running STM32CubeMX plug-in from Eclipse IDE.</td>
</tr>
</tbody>
</table>
Table 20. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>STM32CubeMX release number</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-Apr-2019 29 (cont'd)</td>
<td>4.30</td>
<td>Updated Table 3: Window menu. Updated figures 26 to 30, Figure 101: Advanced Settings window, figures 120 to 127, 129 to 132 and 134 to 143, Figure 218: Project Settings and toolchain selection and figures 246 to 256, Added Figure 24: New Project window shortcuts, Figure 82: STM32MP1 Series: assignment options for GPIOs, Figure 350: Resource Manager; peripheral assignment view and Figure 352: STM32Cube Embedded Software package.</td>
<td></td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved