How to assemble and use the AutoDevKit adaptive front lighting kits

Introduction

The AEKD-AFL001 represents the complete logic and driving hardware for an adaptive front lighting system for prototyping, testing, and development purposes.

It consists of several AutoDevKit boards designed for motor vehicle application development featuring ST automotive-grade components.

The set includes two stepper motor control boards, a four-channel LED driver board, a control board with MCU, a connector board with a FAN switch board and another connector board for wiring configuration.

The adaptive front lighting system firmware runs on the control board automotive-grade SPC5 chorus MCU and allows independent control of all the function boards and their respective loads.

The package also includes sample applications to help users familiarize themselves with the code quickly.

The firmware is available in the STSW-AUTODEVKIT studio. Just import the related project named 'SPC58ECXX_RLA adaptive front lighting (AFL)'.

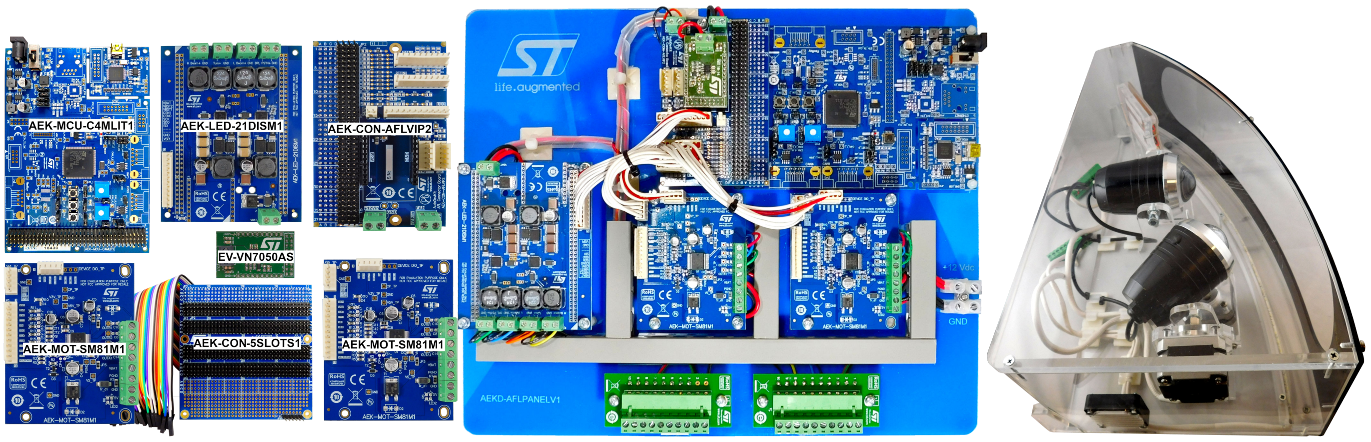
You can also order our fully compatible demo motor vehicle front lighting assembly (AEKD-AFLLIGHT1) with LED lights, stepper motors, and a fan to provide a complete adaptive front lighting tool for application and solution development purposes.

Figure 1. AutoDevKit adaptive front lighting development kits

Left: AEKD-AFL001 kit of individual boards

Right: AEKD-AFLPANEL1 kit of assembled and connected boards

Bottom: AEKD-AFLLIGHT1 kit of typical headlight loads



1 Overview of Adaptive front lighting demonstration kits

The AFL kit consists of an MCU development board, a LED driver board and two motor control boards from the AutoDevKit range designed for automotive application development, as well as a high side driver evaluation board ideal for cooling fan control.

The kit also includes two adapter boards which help manage the various connections between the MCU board and driver boards, and make integrating other boards or components easier with mirrored connectors and breadboard areas.

The AEKD-AFLPANEL1 kit is a convenient alternative to AEKD-AFL001, with the same boards professionally mounted and organized on a Perspex panel, ready for immediate use.

Note: *All the boards in the AEKD-AFL001 adaptive front lighting kit can be ordered separately from the ST website or from authorized ST distributors.*

Table 1. **ST development and functional boards for AFL application**

Kit RPN	Board RPN	Q.ty	Description	Core product
AEKD-AFL001 / AEKD-AFLPANEL1	AEK-MCU-C4MLIT1	1	MCU discovery board for SPC58 Chorus automotive microcontroller with CAN transceivers	SPC58EC80 32-bit power architecture MCU for automotive general purpose applications - Chorus family
	AEK-LED-21DISM1	1	Digitally controlled LED driver board for automotive lighting applications	L99LD21 high power LED driver for automotive applications
	AEK-MOT-SM81M1	2	Stepper Motor driver for automotive applications	L99SM81V programmable stepper motor driver for automotive applications
	EV-VN7050AS	1	Driver board ideal for driving AFL cooling fans	VN7050AS high-side driver with MultiSense analog feedback for automotive applications
	AEK-CON-AFLVIP2	1	Adaptive Front Lighting dedicated connector board with VIPower board slot	-
	AEK-CON-5SLOTS1 (1)	1	5 Slot AutoDevKit connector board and breadboard facilitating pin reassignment	-

1. *AEKD-AFL001 only*

1.1 AFL demonstration kit architecture

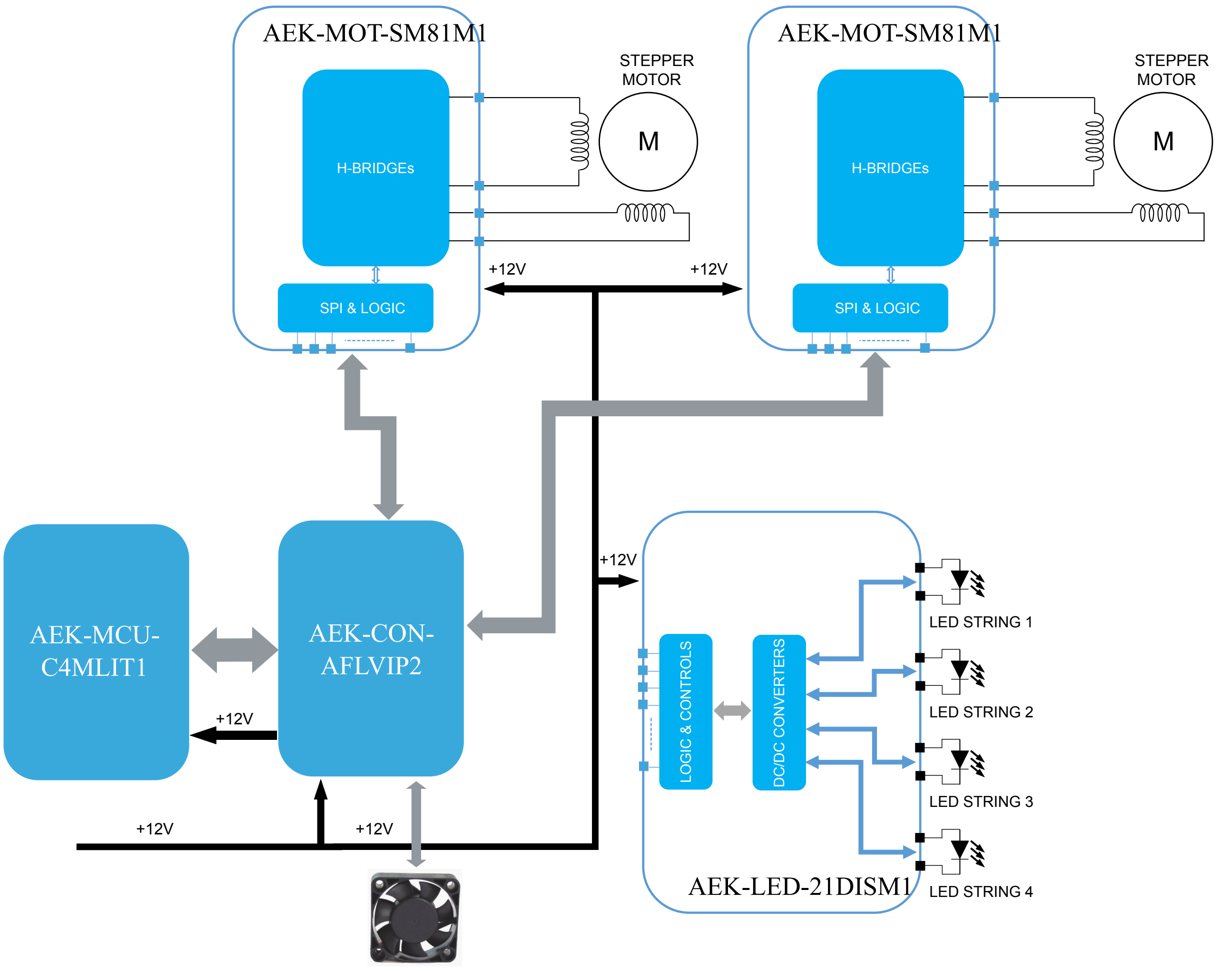
The AEK-MCU-C4MLIT1 control board holds the central microcontroller that monitors and controls overall AFL functionality via SPI communication and GPIOs.

The AEK-CON-AFLVIP2 connector board interfaces the control board with the following headlight function boards:

- an AEK-LED-21DISM1 LED driver board, which drives the following independent LED strings for an automotive headlight: high beam, low beam, daytime running light (DRL) and blinker.
- two AEK-MOT-SM81M1 motor control boards to drive independent stepper motors for position adjustment on respective X and Y axes.
- an EV-VN7050AS cooling fan control board with simple high-side switch.

Note: *ST supplies a family of dedicated EV-VNH7xxx H-bridge boards that provide even greater functionality and control than the board supplied in the kit.*

Figure 2. **Adaptive Front Lighting development system**



All of the boards are 12 V compatible as would be expected in an automotive application powered from a motor vehicle battery. The AEKD-AFLPANEL1 kit, in which all the boards are mounted and connected on a Perspex panel, uses common power cabling and a single connector to supply the entire system.

1.2 AFL software development and solutions

The hardware is fully supported by a software ecosystem that includes a development environment named AutoDevKit Studio (STSW-AUTODEVKIT) for automotive applications, in which you can import the entire AFL firmware code project named 'SPC58ECXX_RLA Adaptive Front Lighting (AFL)'.

1.3 Other hardware

ST also supplies an AEKD-AFLLIGHT1 headlight simulation kit with the typical loads found in AFL headlights: two stepper motors, four LED lights and a cooling fan mounted in a Perspex headlight assembly.

Other hardware required for AFL applications using ST boards or kits include:

- A mini-B USB to USB Type-A cable to connect the MCU programmer
- AC adapter 12V 4A
- Jumper wires (if using the AEKD-AFL001)

2 The AutoDevKit AFL development platform

The basic elements of an adaptive front lighting system are the headlight assembly with LED lights and directional positioning motors, and the control and driver logic to manage the system.

The ST AutoDevKit ecosystem contains these elements in two highly convenient preassembled kits. The first is the AEKD-AFLPANEL1 kit with all the necessary control and function boards and preloaded firmware, and the second is the AEKD-AFLLIGHT1 headlight assembly, which is connected to the panel through two simple connectors.

You can also assemble a control unit from the boards in the AEKD-AFL001 kit. This will require wiring the appropriate power and signal lines and interface connectors, and loading the necessary microcontroller firmware before you proceed.

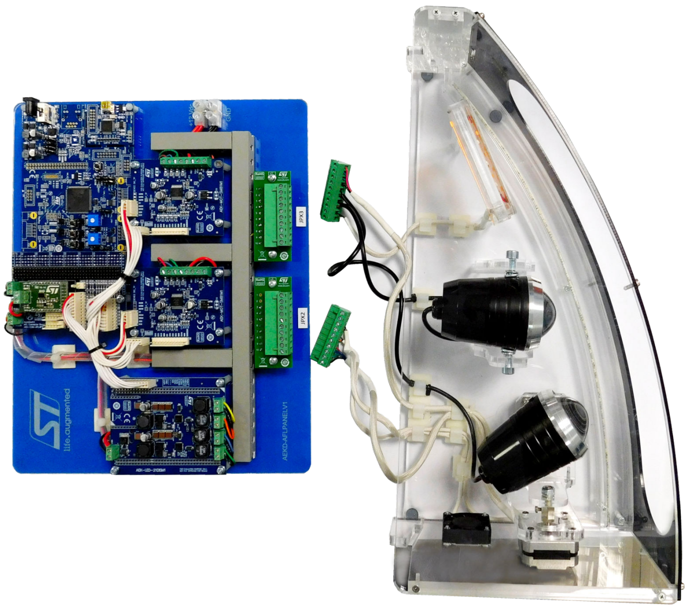
2.1 How to connect the AFL driver kit with the headlight kit

This procedure assumes you are using the AEKD-AFLPANEL1 kit as the control unit and the AEKD-AFLLIGHT1 headlight assembly with typical loads for an adaptive front lighting system.

Step 1. Attach the connectors on the headlight assembly to their counterparts on the control board panel.

Step 2. Supply appropriate DC power at the dedicated connector.
Connect a 12 V, 4 A power supply adapter.

Figure 3. AFL driver kit and headlight connection



As soon as appropriate power is supplied, the AFL demonstration code pre-loaded on the AEKD-AFLPANEL1 central microcontroller (SPC58 MCU) starts turning the headlights on and off and moving them around in a continuous sequence.

Related links

Watch our Adaptive Front Lighting video

Appendix A Steering wheel turn simulation with a potentiometer on page 43

3 AFL source code overview: Automatic mode

The `main()` function in the AFL source code consists of the following blocks:
1. The AFL-Demo Setup, which runs an initialization sequence after a power up or reset event.
2. The AFL-Demo main, which runs a sequence of headlight actions in a continuous loop.

3.1 AFL-Demo setup

The AFL-Demo Setup consists of the following functions:
1. Init components
2. Enable Isr
3. Set up AFL environment

The first two instructions manage the initialization of the components related to MCU standard functions and the low-level drivers for the MCU peripherals, such as Clock, External Interrupt Request Queue (EIRQ), Port Configurations, etc. These two functions are automatically generated by the IDE tool from the configuration information provided during the creation of the AutoDevKit Studio project.

The third function manages the initialization of all AutoDevKit components and MCU functionality implemented by the AFL-Demo engine.

```
/**
* @brief   Setup AFL Environment .
*
*/
void SetupAflEnv()
{
    /* Turn off the second led of the main Board.*/
    pal_lld_setpad(PORT_F, PF_LED2);
    /* Initialize CAN manager */
    CAN_Manager_init(&CAND2, &can_config_mcanconf);
    /* Initialize the AEK-LED-21DISM1 device */
    ClearAndTrigger(AEK_LED_21DISM1_DEV0);
    /* Initialize the AEK-MOTSM81M1 - Motor  x axis  */
    init_AEK_MOT_SM81M1(AEK_MOT_SM81M1_DEV0);
    /* Initialize the AEK-MOTSM81M1 - Motor y axis  */
    init_AEK_MOT_SM81M1(AEK_MOT_SM81M1_DEV1);
    /* Activate DRL and Low Beam */
    ActivateDRL_LowBeam();
    saradc_lld_start(sarAdc, &saradc_config_saradcconf);
    /* Configure and start WKPU Low Level Driver.*/
    wkpu_lld_start(&WKPUD1, &wkpu_config_configuration_name);
    /* Adjust Low Beam to home position */
    Calibration();
    /* Turn on the cooling fan */
    ActiveIN(EV_VNx7xxx_DEV0);
    /* Start the PIT used for triggering the CAN Message send event. */
    pit_lld_channel_start(&PITD1, PIT0_CHANNEL_CH3);
}
```

The `SetupAflEnv()` function source code shows an important feature AutoDevKit Studio: except for the `ActivateDRL_LowBeam()` and `Calibration()` functions, all the others are part of the APIs made available:

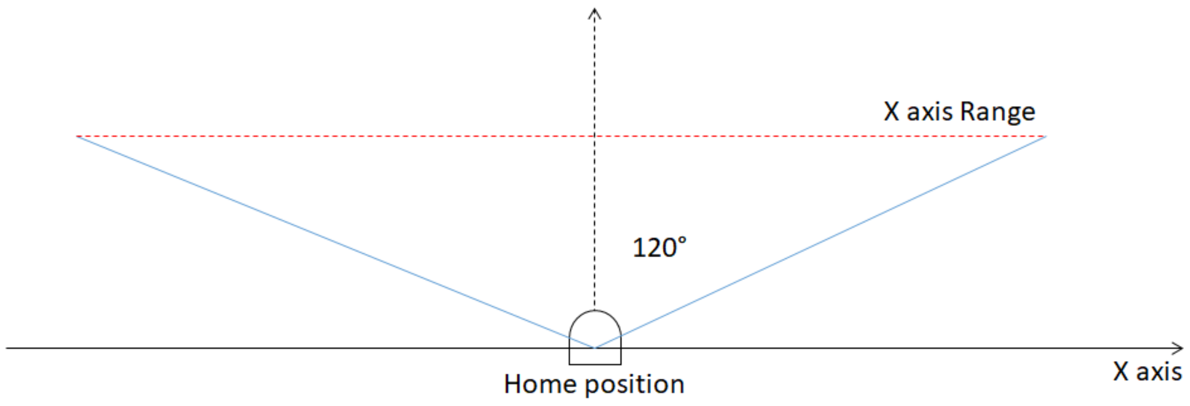
- Low Level Drivers for CAN, ADC, Wake-up, Timer
- `AEK_LED_21DISM1_component_rla`
- `AEK_MOT_SM81M_component_rla`
- `EV_VNx7xxx_component_rla`

These APIs enable direct access to the MCU and its peripheral registers, expose methods to manage the LED driver, stepper motor, and VN actuator components.

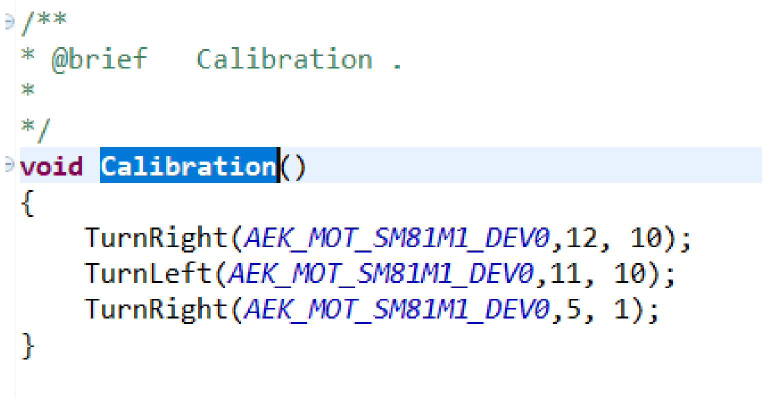
3.1.1 Calibration() function

The aim of the Calibration function after power-on is to ensure that the low beam is in its home position. The low beam is moved by the X-axis stepper motor across its full movement range. During this phase, the stepper motor might trip against the X rail end position several times before completing the procedure and returning to center position.

Figure 4. X-axis movement during calibration



The `Calibration()` function source code shows how this functionality is achieved through the functions in the APIs (i.e., `AEK_ MOT_SM81M_component_rla`) provided.



```
/**
 * @brief   Calibration .
 *
 */
void Calibration()
{
     TurnRight(AEK_MOT_SM81M1_DEV0,12, 10);
     TurnLeft(AEK_MOT_SM81M1_DEV0,11, 10);
     TurnRight(AEK_MOT_SM81M1_DEV0,5, 1);
}
```

The parameters for the `TurnRight()` and `TurnLeft()` functions belonging to the stepper motor APIs are:

- The device (stepper motor) to be driven by the MCU.
- The number of motor steps to perform either on the right or on the left side.
- The delay time in ms between one motor step and the following one.

3.1.2 ActivateDRL_LowBeam() function

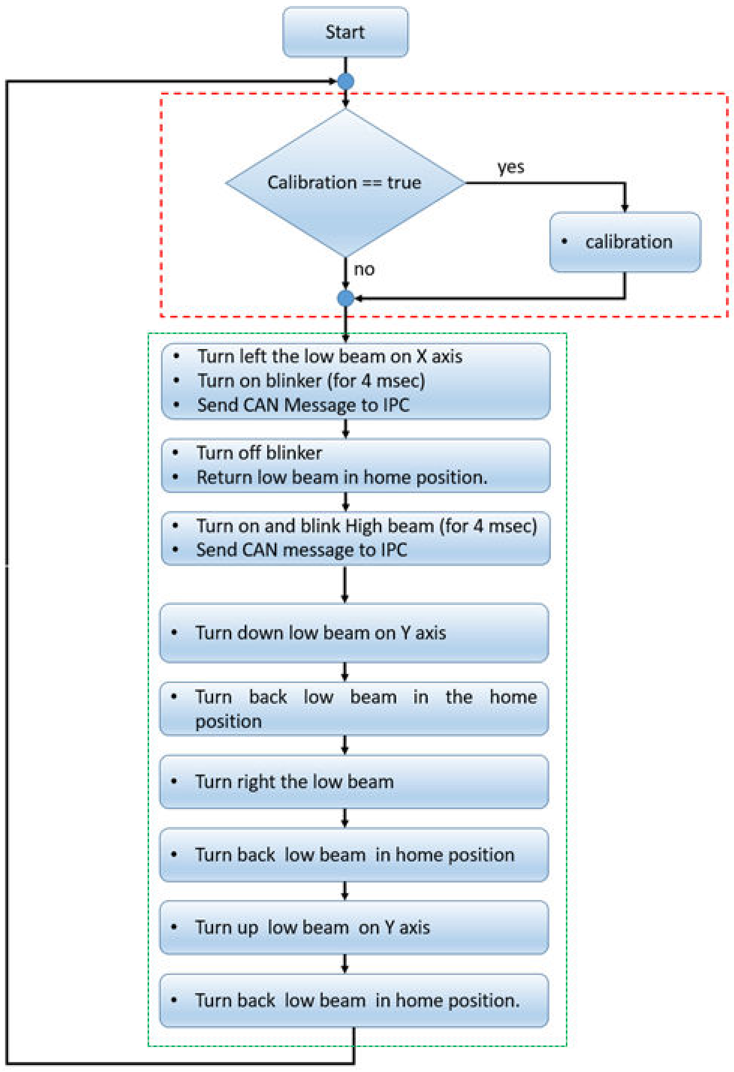
The `ActivateDRL_LowBeam()` function implements a similar API-based approach.

```
void ActivateDRL_LowBeam()
{
    AEK_LED_21DISM1drv_DINActiveBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV1, BUCK2);
    osalThreadDelayMilliseconds(3000);
    AEK_LED_21DISM1drv_DINActiveBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV1, BUCK1);
    CAN_Manager_Activation_LowBeam();
}
```

3.2 AFL-Demo main

The AFL-Demo main engine is implemented by the `Afl()` function. The flow chart below shows an initial block involving calibration, followed by a longer block involving several actions. The calibration block is ignored in Automatic Mode because there is no steering wheel input in this mode.

Figure 5. AFL-Demo main flowchart


Start
Calibration == true
yes
no
• calibration
• Turn left the low beam on X axis
• Turn on blinker (for 4 msec)
• Send CAN Message to IPC
• Turn off blinker
• Return low beam in home position.
• Turn on and blink High beam (for 4 msec)
• Send CAN message to IPC
• Turn down low beam on Y axis
• Turn back low beam in the home position
• Turn right the low beam
• Turn back low beam in home position
• Turn up low beam on Y axis
• Turn back low beam in home position.

The source code below implements several functions.

```
// Turn On Blinker
pit_lld_channel_start(&PITD1, PIT0_CHANNEL_CH2);

//Turn left Motor axis x
TurnLeft(AEK_MOT_SM81M1_DEV0, 5, 1);
```

- Turn the low beam left on x-axis.
- Turn on the blinker (for 4 ms).
- Send CAN message to IPC.

3.2.1 TurnLeft()

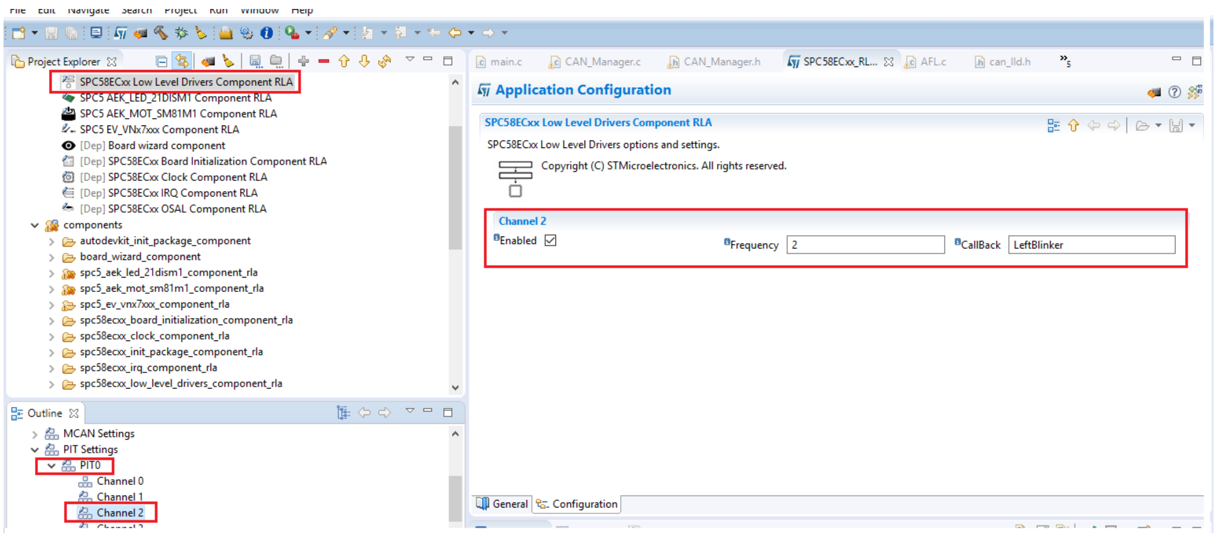
When the `TurnLeft()` function is executed, the MCU sends a driving command via SPI to the x-axis stepper motor to turn the low beam left. This function belongs to the collection of APIs related to `AEK_MOT_SM81M_component_rla` from the AutoDevKit ecosystem.

3.2.2 pit_lld_channel_start()

The `pit_lld_channel_start()` function configures a programmable interrupt timer (PIT) channel to control the blinking frequency of the turn indicator light. The configuration requires timing information expressed as a frequency and the name of a callback function to be executed at the end of each PIT time lapse. The pit_lld_chanell_start belongs to the low level driver APIs provided by AutoDevKit Studio.

The `pit_lld_chanell_start()` function starts a programmable interrupt timer that raises an event at the set frequency to interrupt the MCU and force it to run the `LeftBlinker(void)` callback function which specifically manages this event.

Figure 6. Component configuration for left turn indicator blinking


File Edit Navigate Search Project Run Window Help
Project Explorer
main.c
CAN_Manager.c
CAN_Manager.h
SPC58ECxx_RL...
AFL.c
can_lld.h
Application Configuration
SPC58ECxx Low Level Drivers Component RLA
SPC5 AEK_LED_21DISM1 Component RLA
SPC5 AEK_MOT_SM81M1 Component RLA
SPC5 EV_VNx7xxx Component RLA
[Dep] Board wizard component
[Dep] SPC58ECxx Board Initialization Component RLA
[Dep] SPC58ECxx Clock Component RLA
[Dep] SPC58ECxx IRQ Component RLA
[Dep] SPC58ECxx OSAL Component RLA
components
autodevkit_init_package_component
board_wizard_component
spc5_aek_led_21dism1_component_rla
spc5_aek_mot_sm81m1_component_rla
spc5_ev_vnx7xxx_component_rla
spc58ecxx_board_initialization_component_rla
spc58ecxx_clock_component_rla
spc58ecxx_init_package_component_rla
spc58ecxx_irq_component_rla
spc58ecxx_low_level_drivers_component_rla
Outline
MCAN Settings
PIT Settings
PIT0
Channel 0
Channel 1
Channel 2
General
Configuration

SPC58ECxx Low Level Drivers Component RLA

SPC58ECxx Low Level Drivers options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

Channel 2

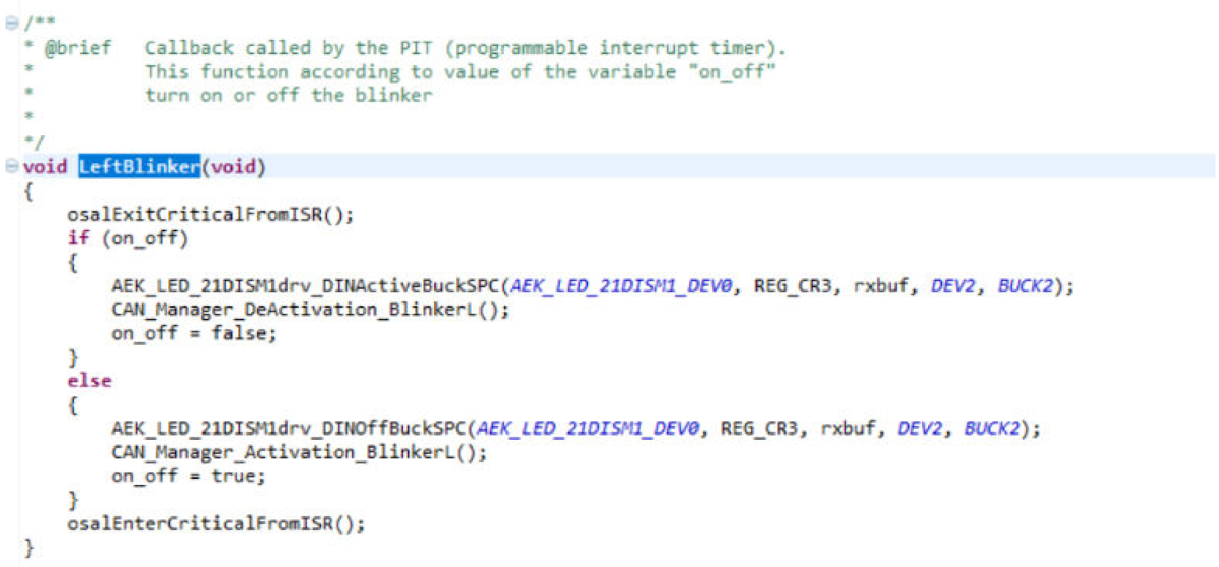
Enabled ☑ Frequency 2 CallBack LeftBlinker

Related links

7.4 Component configuration in AutoDevKit Studio on page 33

3.2.3 LeftBlinker()

The `LeftBlinker()` function switches the turning light on or off according to the on_off variable, which produces the typical blinking effect on a car turn indicator light.



```
/**
 * @brief   Callback called by the PIT (programmable interrupt timer).
 *          This function according to value of the variable "on_off"
 *          turn on or off the blinker
 *
 */
void LeftBlinker(void)
{
    osalExitCriticalFromISR();
    if (on_off)
    {
        AEK_LED_21DISM1drv_DINActiveBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV2, BUCK2);
        CAN_Manager_DeActivation_BlinkerL();
        on_off = false;
    }
    else
    {
        AEK_LED_21DISM1drv_DINOffBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV2, BUCK2);
        CAN_Manager_Activation_BlinkerL();
        on_off = true;
    }
    osalEnterCriticalFromISR();
}
```

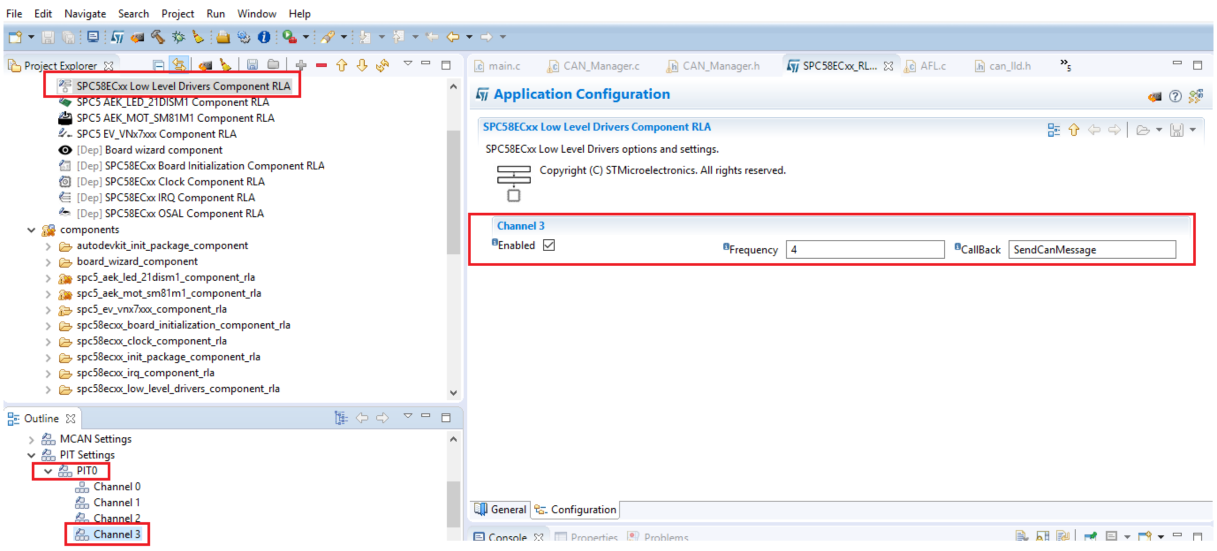
The `LeftBlinker()` source code involves the following API functions:

- `AEK_LED_21DISM1drv_DINActiveBuckSPC()`
- `AEK_LED_21DISM1drv_DINOffBuckSPC()`
- `CAN_Manager_DeActivation_BlinkerL()`
- `CAN_Manager_Activation_BlinkerL()`

The first two APIs send control signals via SPI to activate and deactivate the buck circuit in the LED driver board in order to switch the indicator light on and off.

The other two functions set a hexadecimal code in a CAN message that is transmitted to an instrumental panel control (IPC) to actuate the turn indicator LEDs in the car dashboard. The interrupt that actually sends the CAN message is another dedicated PIT which periodically executes the `CAN_Manager_TransmitMessage()` callback.

Figure 7. **Component configuration for dashboard indicator blinking**



Project Explorer
- SPC58ECxx Low Level Drivers Component RLA
- SPC5 AEK_LED_21DISM1 Component RLA
- SPC5 AEK_MOT_SM81M1 Component RLA
- SPC5 EV_VNx7xxx Component RLA
- [Dep] Board wizard component
- [Dep] SPC58ECxx Board Initialization Component RLA
- [Dep] SPC58ECxx Clock Component RLA
- [Dep] SPC58ECxx IRQ Component RLA
- [Dep] SPC58ECxx OSAL Component RLA
- components
  - autodevkit_init_package_component
  - board_wizard_component
  - spc5_aek_led_21dism1_component_rla
  - spc5_aek_mot_sm81m1_component_rla
  - spc5_ev_vnx7xxx_component_rla
  - spc58ecxx_board_initialization_component_rla
  - spc58ecxx_clock_component_rla
  - spc58ecxx_init_package_component_rla
  - spc58ecxx_irq_component_rla
  - spc58ecxx_low_level_drivers_component_rla

Outline
- MCAN Settings
- PIT Settings
  - PIT0
    - Channel 0
    - Channel 1
    - Channel 2
    - Channel 3

Application Configuration

SPC58ECxx Low Level Drivers Component RLA

SPC58ECxx Low Level Drivers options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

Channel 3

Enabled ✓ Frequency 4 CallBack SendCanMessage

```
void CAN_Manager_TransmitMessage(CANDriver *driver)
{
    CANTxFrame tmp;
    int n_signal  = 0;
    tmp.ID        = can_mess.id;
    tmp.DLC       = can_mess.dlc;
    tmp.OPERATION = CAN_OP_NORMAL;
    tmp.TYPE      = can_mess.type;
    tmp.data32[0] = can_mess.data32[0];
    tmp.data32[1] = can_mess.data32[1];
    can_lld_transmit(driver, CAN_ANY_TXBUFFER, &tmp);
}
```

Related links

7.4 Component configuration in AutoDevKit Studio on page 33

3.2.4 osalThreadDelayMilliseconds()

The `osalThreadDelayMilliseconds()` function is invoked several times in the source code to implement delays, which are fundamental in real-time asynchronous environments without an operating system. This function is part of the operating system abstraction (OSAL) low level driver provided by AutoDevKit Studio.

3.2.5 Other functions

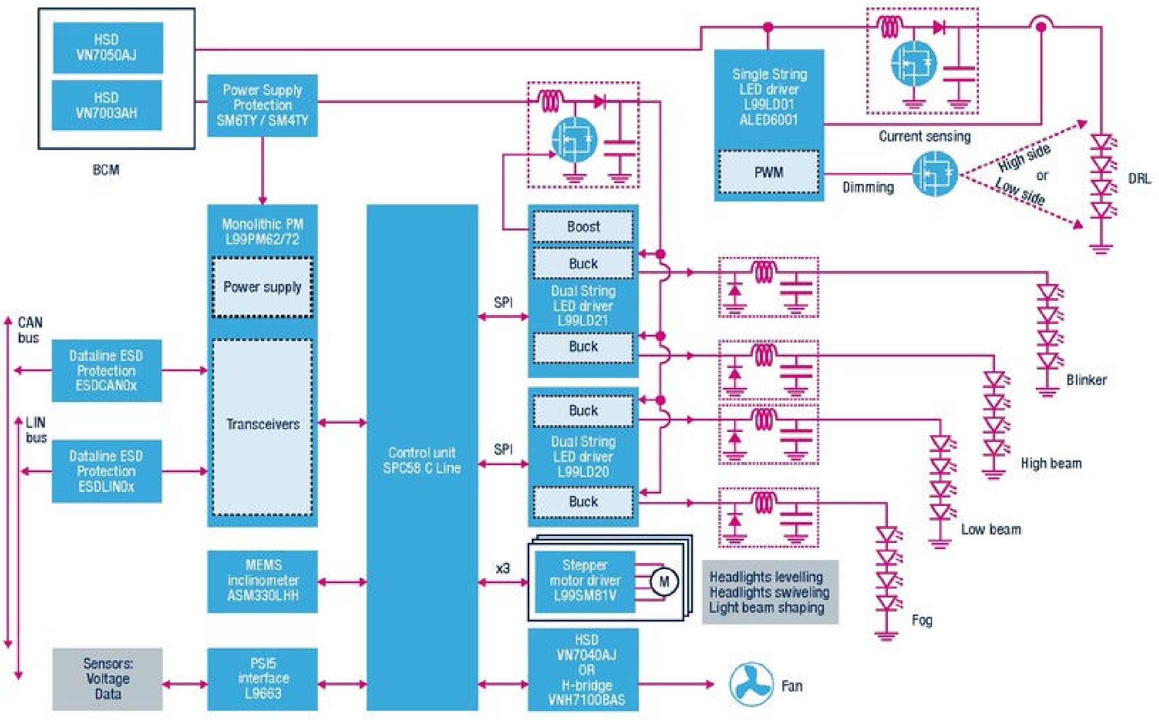
- Low level driver functions:
  - pal_lld_setpad: function used to set the logical state of an output port
  - can_lld_start: function used to configure and to activate the CAN peripheral
  - saradc_lld_start: function used to configure and to activate SAR ADC peripheral
  - wkpu_lld_start: function used to configure and to activate the WKPU signal
- AEK_LED_21DISM1_component_rla:
  - AEK_LED_21DISM1drv_DINActiveBuckSPC: activates the Buck of the LED driver device
  - AEK_LED_21DISM1drv_DINOffBuckSPC: deactivates the Buck of the LED driver device
  - AEK_LED_21DISM1drv_IntensityBuckSPC: sets the current to define the light intensity
  - AEK_LED_21DISM1drv_ClearStausRegister: clears the status LED driver the control register

- AEK_MOT_SM81M_component_rla
  - AEK_MOT_SM81M1drv_WriteSPIRegisterMOV: writes the appropriate register to set the step resolution
  - init_AEK_MOT_SM81M1: initializes the register the stepper motor driver registers

4 Hardware

The image below shows the extent of the range of ST products related to adaptive front lighting applications, including the SPC58 Chorus 4M automotive microcontroller with CAN transceivers, which is able to control the entire system.

Figure 8. AFL block diagram with ST application devices


HSD VN7050AJ
HSD VN7003AH
BCM
Power Supply Protection SM6TY / SM4TY
Monolithic PM L99PM62/72
Power supply
Transceivers
CAN bus
LIN bus
Dataline ESD Protection ESDCANOx
Dataline ESD Protection ESDLINOx
MEMS inclinometer ASM330LHH
Sensors: Voltage Data
PSI5 interface L9663
Control unit SPC58 C Line
SPI
SPI
x3
Boost
Buck
Dual String LED driver L99LD21
Buck
Buck
Dual String LED driver L99LD20
Buck
Stepper motor driver L99SM81V
M
HSD VN7040AJ OR H-bridge VNH7100BAS
Fan
Single String LED driver L99LD01 ALED6001
PWM
Current sensing
Dimming
High side or Low side
DRL
Blinker
High beam
Low beam
Fog
Headlights levelling Headlights swiveling Light beam shaping

In the AEKD-AFL001 and AEKD-AFLPANEL1 kits, the AEK-MCU-C4MLIT1 control board MCU monitors and controls the following function boards:

- an AEK-LED-21DISM1 evaluation board with two L99LD21 LED drivers to drive headlight lamps and turning indicators.
- two AEK-MOT-SM81M1 evaluation boards with L99SM81V stepper motor driver for low beam horizontal swivel adjustment and vertical tilt adjustment.
- EV-VN7050AS evaluation board with VN7050AS high-side driver for fan control.

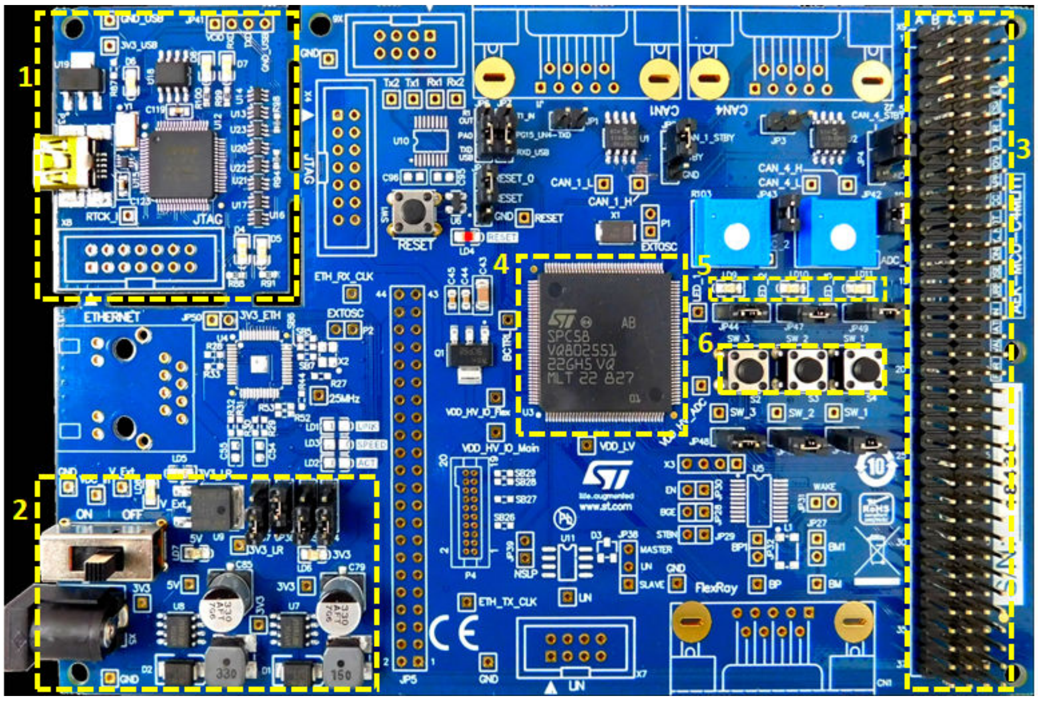
4.1 AEK-MCU-C4MLIT1 system control board

This board provides overall AFL system control via SPI and GPIOs. It is Automotive Safety Integrity Level B (ASIL B) qualified and features a hardware security module (HSM) with security software that is compliant with the EVITA Medium version.

The board has an integrated programmer debugger (supporting both PLS UDE and OpenOCD) for MCU debugging and programming and a USB Virtual COM port for serial output to a PC running a standard terminal program.

Figure 9. **AEK-MCU-C4MLIT1 control board functional blocks**

1. Debugger/programmer
2. 12V input power and DC-DC conversion to 5V and 3.3V
3. Connector (4x37 pins) for function boards
4. MCU SPC58EC80E5 32-bit
5. User defined LEDs
6. User defined buttons



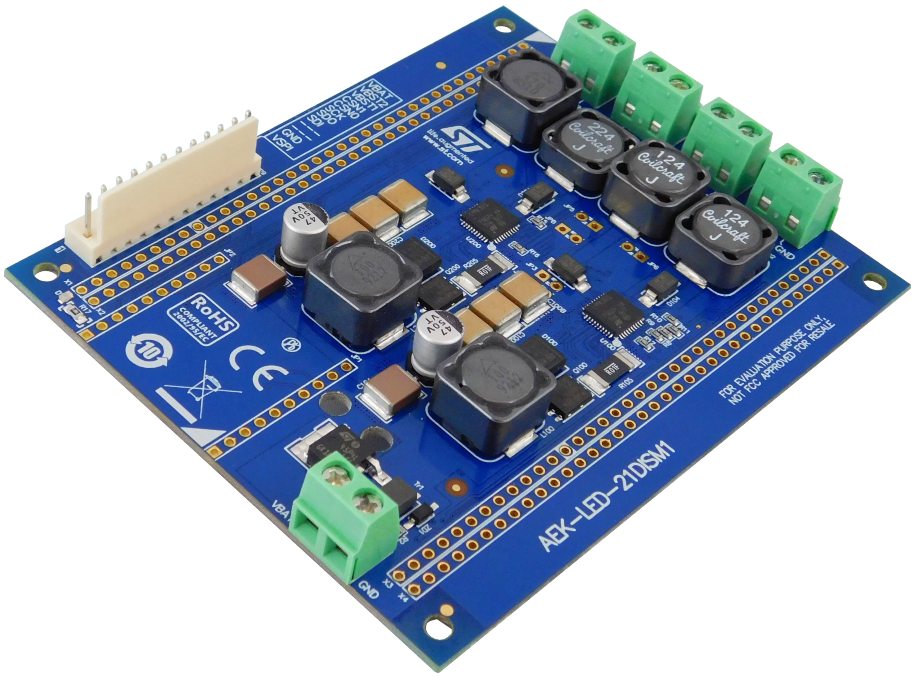
The 32-bit SPC58EC80E5 microcontroller has the following features:

- 2 main dual-issue 32-bit CPUs built on Power Architecture technology, operating up to 180MHz.
- 4224 KB (4096 KB code Flash + 128 KB data Flash) on-chip flash memory.
- 384 KB on-chip general-purpose SRAM (+ 128 KB local data RAM: 64 KB included in each CPU).
- 8 DSPIs available: 3 DSPIs are allocated in AFL-Demo to connect the LED driver board and the two stepper motor driver boards.
- 8 (Programmable Interrupt Timer) PIT channels available: 2 PIT channels are allocated for the AFL-Demo.
- 2 x 32 eMIOS channels available: 1 channel allocated for the AFL-Demo.
- Up to 96 channels SARADC available: 1 channel allocated for steering wheel position.
- 8 MCAN available: 1 MCAN used to connect and control an instrumental dashboard.

4.2 AEK-LED-21DISM1 LED driver board

The AEK-LED-21DISM1 evaluation board is designed to drive four LED strings. In the AFL-Demo, it is used to power the daytime running light (DRL), low beam, high beam and blinker. The board hosts two L99LD21 LED driver ICs, each of which is able to manage a single boost and two buck circuits.

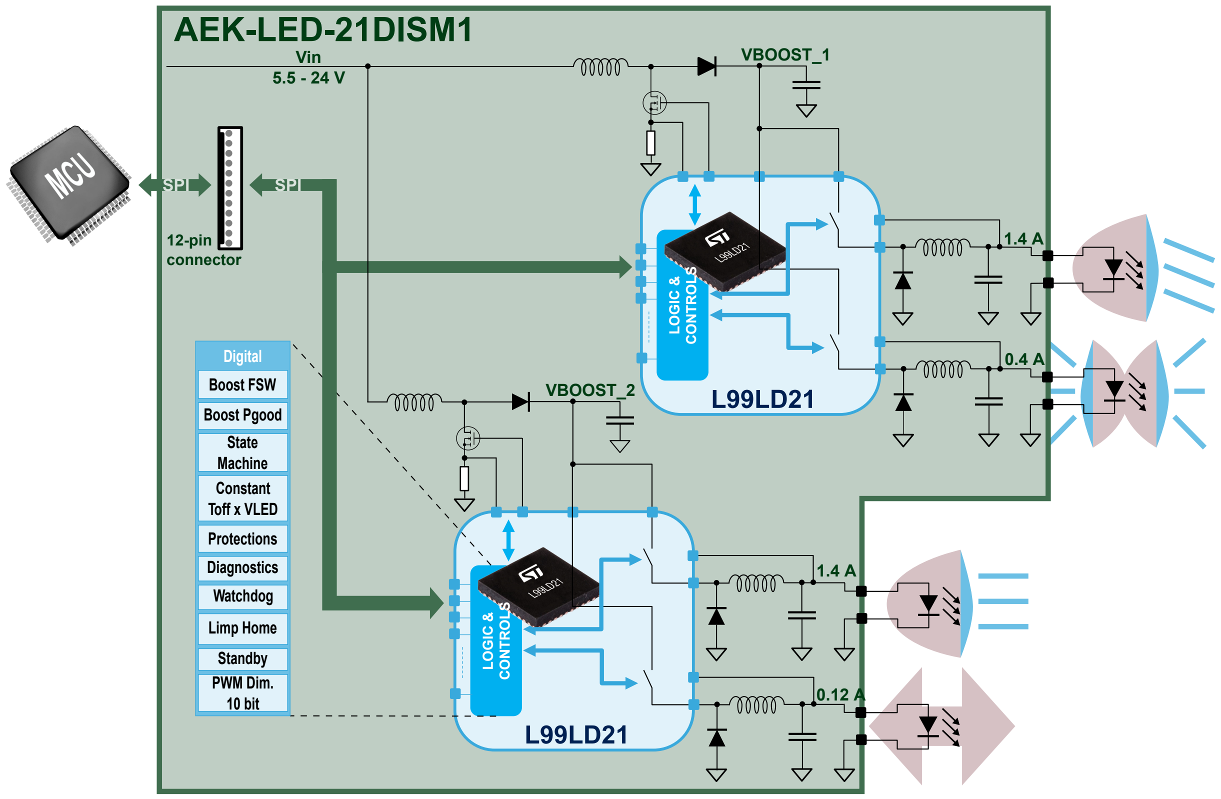
Figure 10. AEK-LED-21DISM1 LED driver board



The boost controller section in each L99LD21 LED driver integrates a high current gate driver for an external N-channel MOSFET and delivers a constant output voltage to two integrated buck converters. The boost controller of the two devices can be stacked to allow dual phase operation for high power applications with an interleaving pattern for improved input current ripple and current load sharing. Each L99LD21 buck converter integrates an N-channel MOSFET driven by a bootstrap circuit.

The overall behavior of the board AEK-LED-21DISM1 is monitored and controlled through L99LD21 dedicated registers, which can be read or set by an external microcontroller through the fast SPI interface via a 12-pin male connector. Finally, a special Limp Home Mode implemented in the L99LD21 driver ensures that active communication with the MCU is regularly monitored.

Figure 11. AEK-LED-21DISM1 four LED load driving block diagram



The AEK-LED-21DISM1 board has the following features:

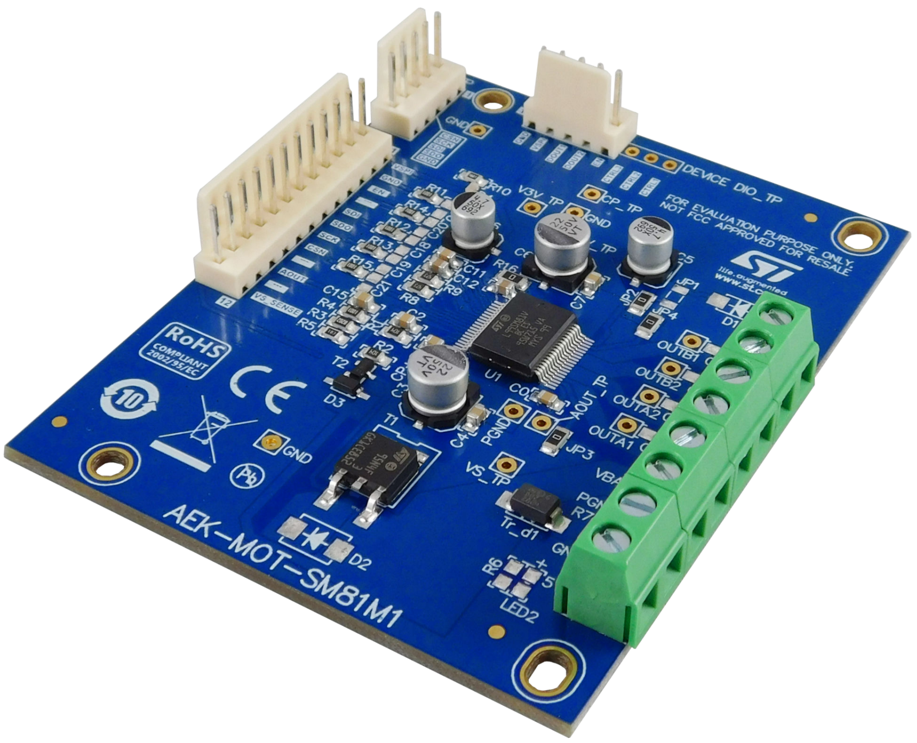
- Two embedded L99LD21 high power LED drivers able to supply four independent channels
- Output current up to 1.69 A each channel
- Input operating voltage range from 5.5 V to 24 V
- Programmable via SPI, allowing accurate LED current setting
- Protections and diagnostics for output short-circuit and open-load, overtemperature, and battery undervoltage.
- Automotive Limp-Home Mode support
- Board size: 81 x 100 mm. Maximum components height: 11 mm
- WEEE and RoHS compliant
- All ST components are qualified Automotive grade

4.3 AEK-MOT-SM81M1 stepper motor driver board

The AEK-MOT-SM81M1 evaluation board targets automotive stepper motor applications. It is designed to drive one bipolar stepper motor with stepping resolution ranging from 4 full steps to 64 micro steps and with coil voltage monitoring for stall detection. The embedded L99SM81V stepper motor driver manages timers, counters, a reference table and status registers that are manipulated by an external MCU via SPI.

The AEKD-AFL001 and AEKD-AFLPANEL1 kits include two boards to drive an x-axis and a y-axis stepper motor.

Figure 12. **AEK-MOT-SM81M1 motor driver board**



The AEK-MOT-SM81M1 board has the following features:

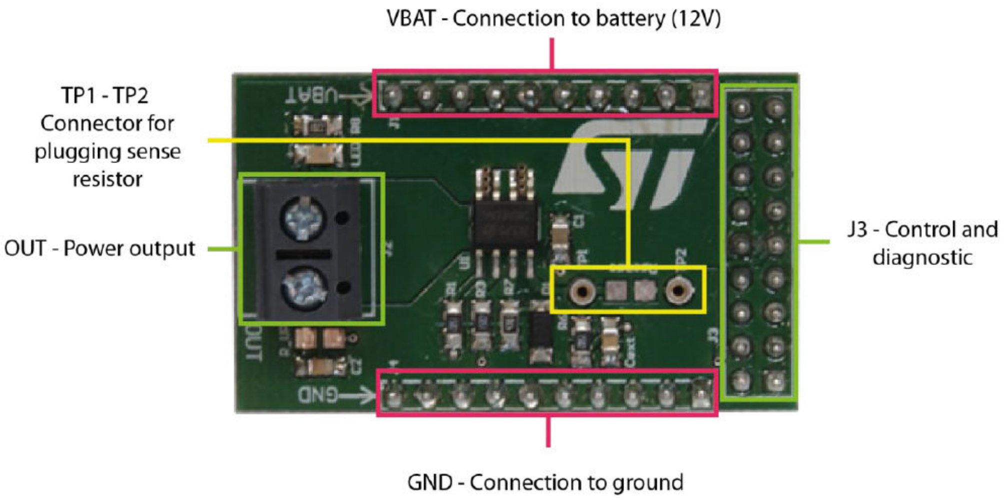
- Board functionality based on L99SM81V programmable stepper motor driver for automotive applications:
  - with micro-stepping and hold functions
  - BEMF monitoring for stall detection
  - programmable configuration via SPI
  - 5 V internal linear voltage regulator (output available on board connector)
- Board reverse battery protection with STD95N4F3 MOSFET, which can be substituted with two optionally mounted diodes and a jumper
- Input operating voltage range from 6 V to 28 V
- Output current up to 1.35 A
- Board size: 65 mm length x 81 mm width x 11 mm maximum component height
- WEEE and RoHS compliant
- All ST components are qualified Automotive grade
- Applications: automotive bipolar stepper motor

4.4 EV-VN7050AS fan control board

The EV-VN7050AS evaluation board in the AFL-Demo is used to drive a cooling fan for the LED headlight. The board is based on the VN7050AS single channel high-side driver featuring ST's VIPower technology. The device is normally powered at 12 V, but has an operating range of 4-28 V to accommodate battery voltage spikes and crank values.

The VN7050AS also integrates advanced protection functions such as load current limitation, overload active management through power limitation and over-temperature shutdown. The device current limitation value is 30 A (typical) and has very low standby power. The board can accommodate a sense resistor for load current sensing.

Figure 13. EV-VN7050AS high-side driver board components


VBAT - Connection to battery (12V)
TP1 - TP2 Connector for plugging sense resistor
OUT - Power output
J3 - Control and diagnostic
GND - Connection to ground

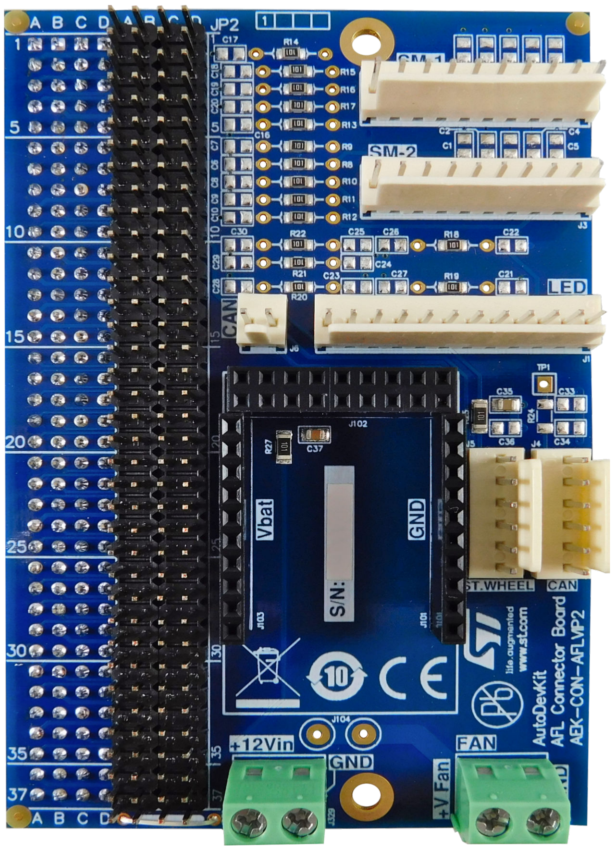
4.5 AEK-CON-AFLVIP2 connector board

The AEK-CON-AFLVIP2 evaluation board is designed to facilitate the connection between the boards in the Adaptive Front Light solution by ST: an MCU control board like the AEK-MCU-C4MLIT1, two AEK-MOT-SM81M1 stepper motor boards, and an AEK-LED-21DISM1 LED driver board, as well as a dedicated slot for EV-VNx boards like the EV-VN7050AS for cooling fan control.

Note: *The female connector for the MCU control board is located on the rear side of the connector board.*

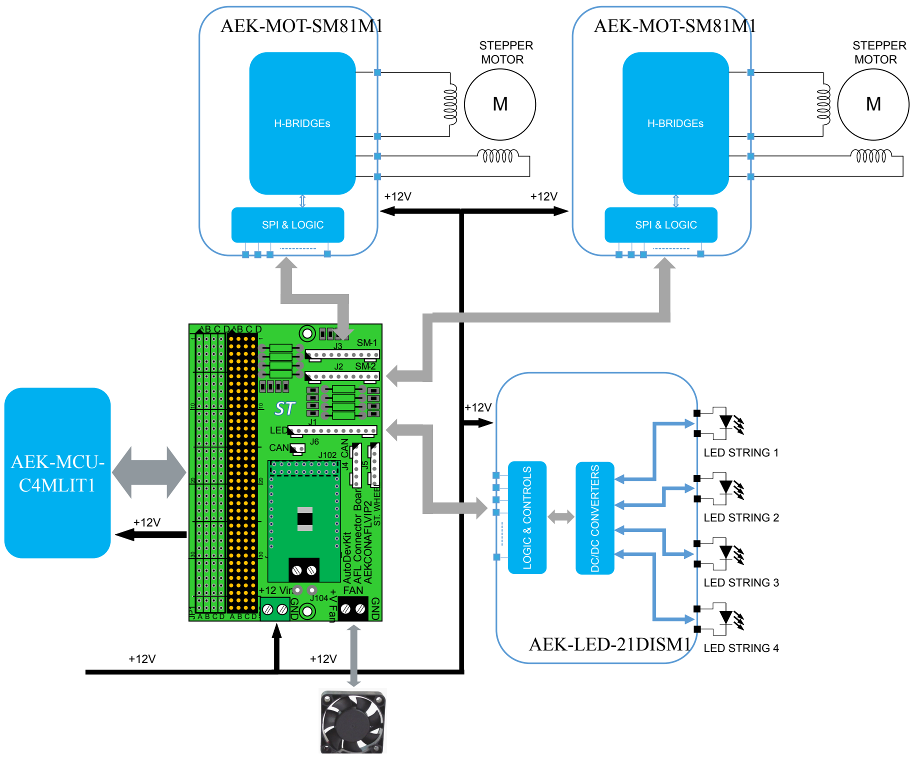
The 12 V supply input for the entire system can also be supplied through this board.

Figure 14. AEK-CON-AFLVIP2 connector board



The connector board is designed to allow the addition of extra functions to those already included in the AFL-Demo through the 4x37 male pin connector, which has the same electrical connections as those on the MCU board.

Figure 15. AEK-CON-AFLVIP2 block diagram



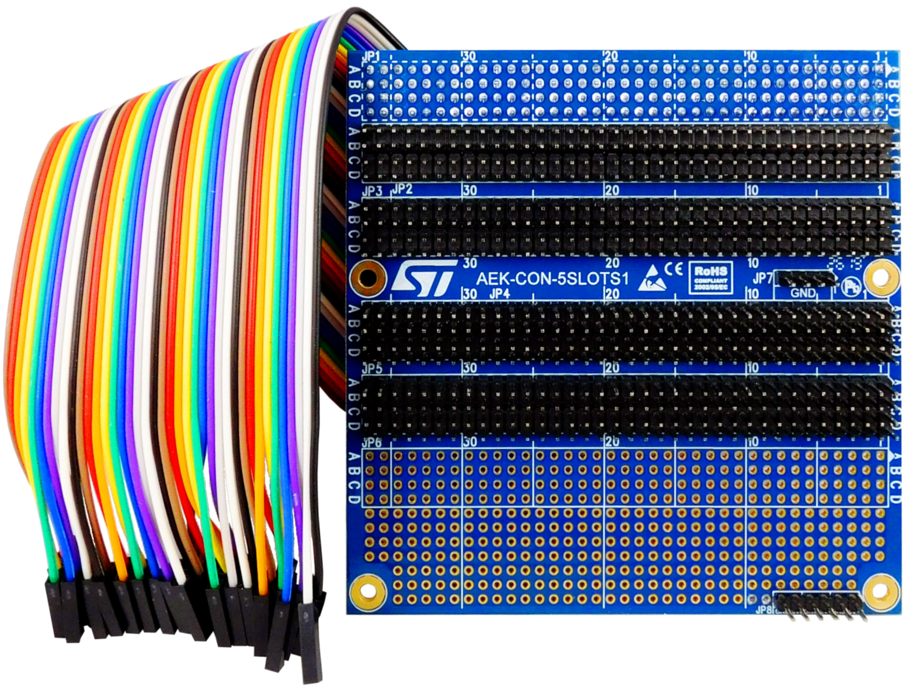
The AEK-CON-AFLVIP2 board has the following features:

- Designed to connect the following elements for a complete automatic front lighting (AFL) adjustment solution for a LED vehicle headlight:
  – An MCU board such as the AEK-MCU-C4MLIT1 with 4x37 connector
  – Two AEK-MOT-SM81M1 stepper motor boards
  – An AEK-LED-21DISM1 LED driver control board with two L99LD21 LED drivers and providing four independent channels
  – An EV-VNx actuator board such as the EV-VN7050AS to switch a headlight colling fan on and off
- Connected boards are controlled via three separate SPIs
- Input voltage from 8 V_{DC} to 15 V_{DC}
- Board size: 65 x 100 mm. Maximum component height: 27 mm
- Cables included for two stepper motor boards and one LED driver board
- WEEE and RoHS compliant

4.6 AEK-CON-5SLOTS1 connector board

The AEK-CON-5SLOTS1 connector board is designed to facilitate connections and reduce development time while prototyping automotive applications with an SPC5x Discovery board. The set of 4x37 extension connectors allow developers to reconfigure pin assignments for ADCs, Timers, GPIOs, DSPIs and other peripherals on SPC5 Discovery boards and generate different pin assignment patterns on the extension connectors.

Figure 16. **AEK-CON-5SLOTS1 connector board**

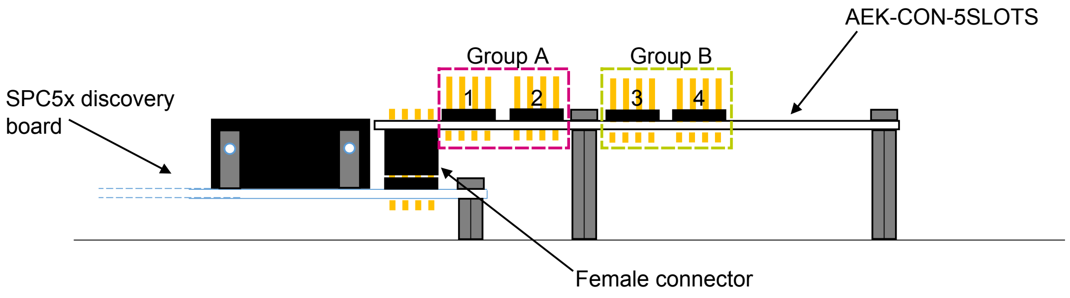


The AEK-CON-5SLOTS1 board has the following features:

- Fast and easy extension of 4x37 connector on SPC5x discovery boards
- Single female connector to plug the board onto an SPC5x 4x37 connector
- Two male connectors with same outputs as 4x37 connector
- Two configurable male connectors 4x37 whose pins can be reassigned to connect different functional boards
- Additional ground and 5V pins
- Bare section available on the board for specific application purposes
- Female-to-female jump wire set included

Regarding the following image showing the connector Groups, the two Group A connectors, 1 and 2, share a direct connection, as do the two Group B connectors, 3 and 4; the AEK-CON-5SLOTS1 is plugged directly on the SPC5x Discovery board via the 4x37 female connector on the bottom side of the board.

Figure 17. **AEK-CON-5SLOTS1 connector groups**



The configuration of the MCU pins is transferred to the 4x37 male connectors 1 and 2 of the AEK-CON-5SLOTS1 board. The pins on connector 2 can then be reassigned using jumper wires to new pins on connector 3. As connectors 3 and 4 are electrically connected, an external functional board compatible with the new pin assignment can be plugged to connector 4.

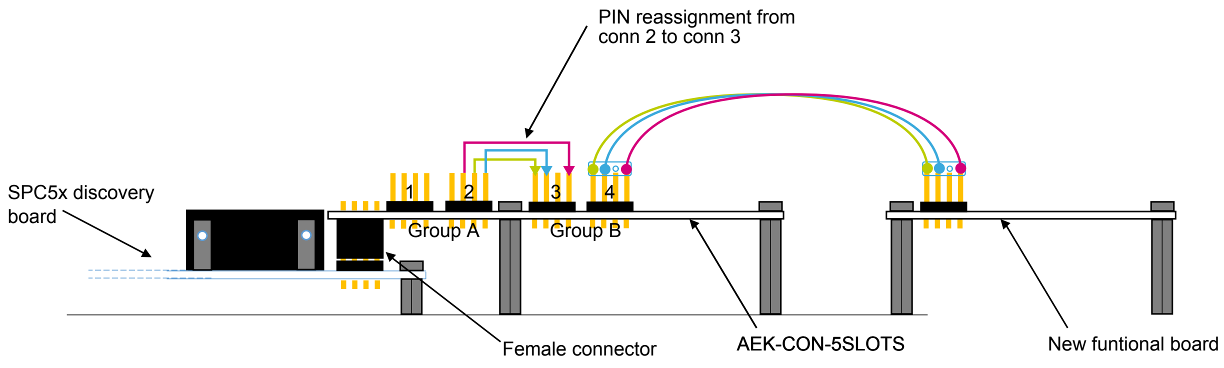


Figure 18. **AEK-CON-5SLOTS1 pin reassignment for external function boards**

The unused connector 1 in group A offers the possibility to stack more extension connector boards on top of each other.

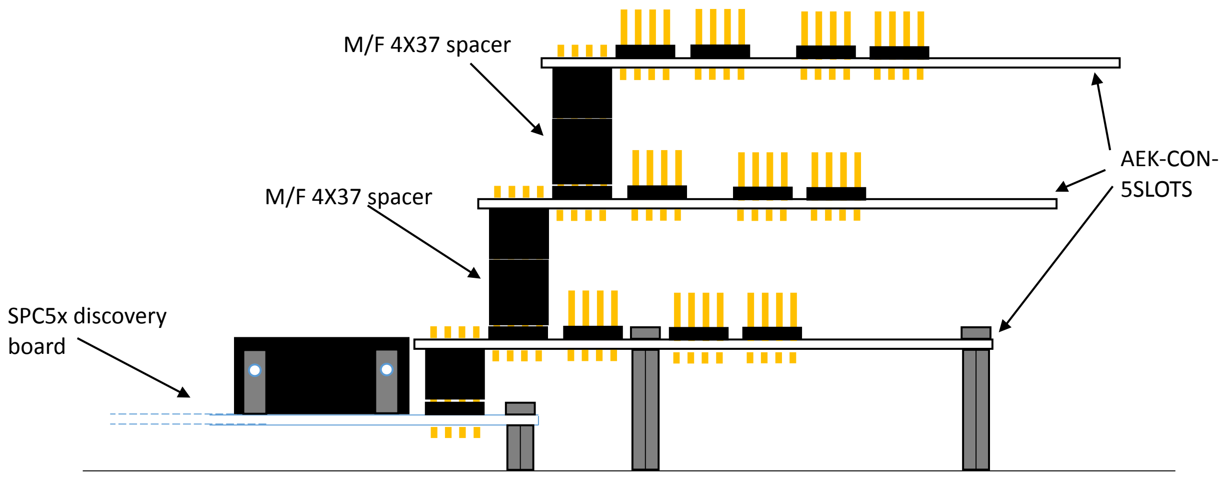


Figure 19. **Multiple AEK-CON-5SLOTS1 extension boards mounted on SPC5x Discovery board**

The AEK-CON-5SLOTS board also has a breadboard area to add simple circuits necessary to complete the prototype application.

5 AFL-Demo software package

The AFL-Demo software package consists of the following components:

- The AutoDevKit Studio contained firmware 'SPC58ECXX_RLA Adaptive Front Lighting (AFL)' to be loaded on the SPC58EC80 microcontroller.
- SPC58Cxx_RLA Adaptive Front Lighting(AFL) project and driver files, including:
  - SPC5 AEK_LED21DISIM1 Component RLA
  - SPC5 AEK_MOT_SM81M1 Component RLA
  - SPC5 EV_VNx7xxx Component RLA
  - SPC58Cxx Low Level Driver

Related links

Discover the full range of available components from the AutoDevKit page on the ST website

6 How to program the SPC58EC80 microcontroller

Follow this procedure to load the latest version of the firmware on the AEK-MCU-C4MLIT1 discovery board. You may skip this procedure if you have the AEKD-AFLPANEL1 kit, as the software is already loaded on the discovery board included in the kit.

Step 1. Download and install SPC5-UDESTK-SW debugger (an alternative is to use the pre-installed AutoDevKit Studio OpenOCD).

Step 2. Connect the mini-B USB cable between your PC and the AEK-MCU-C4MLIT1 discovery board.

Step 3. Turn-on the discovery board.

Step 4. Run the UDE application on your PC.

Step 5. In the UDE program selcect, [**File**]>[**Open**] ‘afl.wsx’.

afl.wsx is found in the UDE directory of the code zip; it is a hook file for the Universal Debug Engine (UDE) to burn the AFL-Demo code in the SPC58 microcontroller Flash memory.

As soon as the file is loaded a new window appears.

Step 6. Click [**Program All**] in the new window.

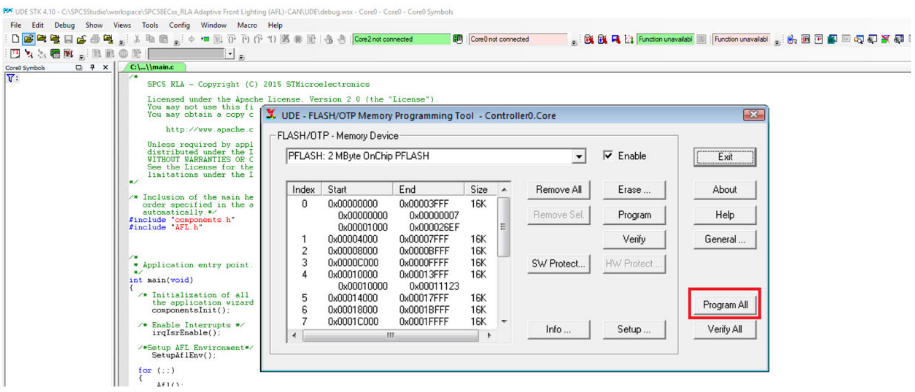
A confirmation message appears in the same window then the operation is completed.

The new firmware is now flashed in the microcontroller.

Step 7. Exit the window.

Step 8. Reset the AEK-MCU-C4MLIT1 discovery board to run the updated firmware.

Figure 20. Flashing the AFL firmware using UDE STK



```
/*
    SPC5 RLA - Copyright (C) 2015 STMicroelectronics

    Licensed under the Apache License, Version 2.0 (the "License").
    You may not use this fi
    You may obtain a copy c

        http://www.apache.c

    Unless required by appl
    distributed under the I
    WITHOUT WARRANTIES OR C
    See the License for the
    limitations under the I
*/

/* Inclusion of the main he
   order specified in the a
   automatically.*/
#include "components.h"
#include "AFL.h"


/*
 * Application entry point.
 */
int main(void)
{
  /* Initialization of all
     the application wizard
     componentsInit();

  /* Enable Interrupts */
     irqIsrEnable();

  /*Setup AFL Environment*/
     SetupAflEnv();

  for (;;)
  {
     Afl();
```

Index	Start	End	Size
0	0x00000000	0x00003FFF	16K
	0x00000000	0x00000007	
	0x00001000	0x000026EF	
1	0x00004000	0x00007FFF	16K
2	0x00008000	0x0000BFFF	16K
3	0x0000C000	0x0000FFFF	16K
4	0x00010000	0x00013FFF	16K
	0x00010000	0x00011123	
5	0x00014000	0x00017FFF	16K
6	0x00018000	0x0001BFFF	16K
7	0x0001C000	0x0001FFFF	16K

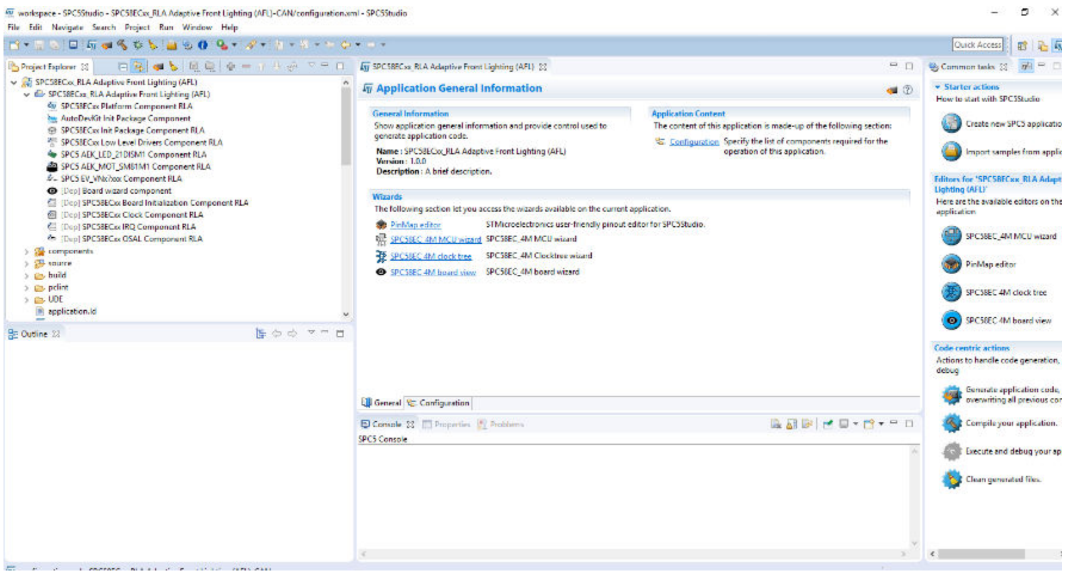
7 AutoDevKit Studio overview

AutoDevKit Studio is an integrated development environment (IDE) based on Eclipse. It contains a standard workspace and an extensible plug-in system environment customization.

The aim of the IDE is to maximize developer productivity of embedded applications based automotive-grade microcontrollers with a single tool for evaluation, development, design and production.

AutoDevKit Studio includes an application wizard to simplify project creation and configuration; it automatically solves component dependencies and generates support files.

Figure 21. AutoDevKit Studio AFL project workspace



workspace - SPC5Studio - SPC58ECxx_RLA Adaptive Front Lighting (AFL)-CAN/configuration.xml - SPC5Studio

File Edit Navigate Search Project Run Window Help

Quick Access

Project Explorer

- SPC58ECxx_RLA Adaptive Front Lighting (AFL)
  - SPC58ECxx_RLA Adaptive Front Lighting (AFL)
    - SPC58ECxx Platform Component RLA
    - AutoDevKit Init Package Component
    - SPC58ECxx Init Package Component RLA
    - SPC58ECxx Low Level Drivers Component RLA
    - SPC5 AEK_LED_21DISM1 Component RLA
    - SPC5 AEK_MOT_SM81M1 Component RLA
    - SPC5 EV_VNx7xxx Component RLA
    - [Dep] Board wizard component
    - [Dep] SPC58ECxx Board Initialization Component RLA
    - [Dep] SPC58ECxx Clock Component RLA
    - [Dep] SPC58ECxx IRQ Component RLA
    - [Dep] SPC58ECxx OSAL Component RLA
  - components
  - source
  - build
  - pclint
  - UDE
  - application.ld

Outline

SPC58ECxx_RLA Adaptive Front Lighting (AFL)

Application General Information

General Information

Show application general information and provide control used to generate application code.

Name : SPC58ECxx_RLA Adaptive Front Lighting (AFL)
Version : 1.0.0
Description : A brief description.

Application Content

The content of this application is made-up of the following section:

Configuration Specify the list of components required for the operation of this application.

Wizards

The following section let you access the wizards available on the current application.

PinMap editor	STMicroelectronics user-friendly pinout editor for SPC5Studio.
SPC58EC 4M MCU wizard	SPC58EC_4M MCU wizard
SPC58EC 4M clock tree	SPC58EC_4M Clocktree wizard
SPC58EC 4M board view	SPC58EC_4M board wizard

General | Configuration

Console | Properties | Problems

SPC5 Console

Common tasks

Starter actions

How to start with SPC5Studio

- Create new SPC5 applicatio
- Import samples from applic

Editors for 'SPC58ECxx_RLA Adapt Lighting (AFL)'

Here are the available editors on the application

- SPC58EC_4M MCU wizard
- PinMap editor
- SPC58EC 4M clock tree
- SPC58EC 4M board view

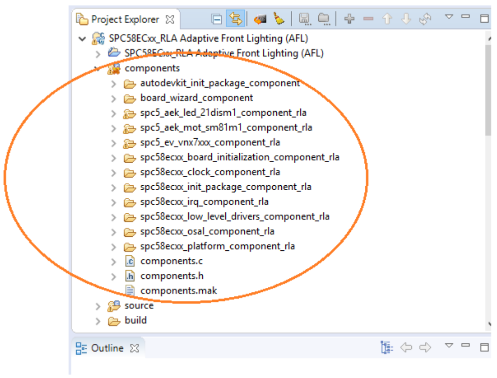
Code centric actions

Actions to handle code generation, debug

- Generate application code, overwriting all previous co
- Compile your application.
- Execute and debug your ap
- Clean generated files.

The application wizard integrates the initial components into the project, with the key elements employed by the tool to generate the final application source code. Typical of layered architecture, the services in one component are provided to other components. Furthermore, the configuration of each component is supported by an intuitive GUI.

Figure 22. AutoDevKit Studio AFL project components



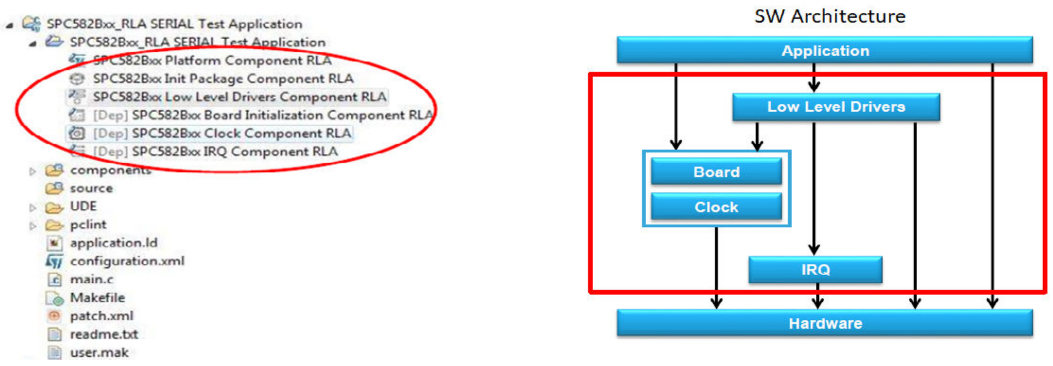
Project Explorer

- SPC58ECxx_RLA Adaptive Front Lighting (AFL)
  - SPC58ECxx_RLA Adaptive Front Lighting (AFL)
  - components
    - autodevkit_init_package_component
    - board_wizard_component
    - spc5_aek_led_21dism1_component_rla
    - spc5_aek_mot_sm81m1_component_rla
    - spc5_ev_vnx7xxx_component_rla
    - spc58ecxx_board_initialization_component_rla
    - spc58ecxx_clock_component_rla
    - spc58ecxx_init_package_component_rla
    - spc58ecxx_irq_component_rla
    - spc58ecxx_low_level_drivers_component_rla
    - spc58ecxx_osal_component_rla
    - spc58ecxx_platform_component_rla
    - components.c
    - components.h
    - components.mak
  - source
  - build

Outline

Register Level Access (RLA) components are low level drivers with direct access to the MCU and peripherals such as CAN, Ethernet, DSPI, ADC, PIT and GTM. The RLA components can be added and configured through a time-saving GUI.

Figure 23. **AutoDevKit Studio AFL RLA drivers**



```
SPC582Bxx_RLA SERIAL Test Application
  SPC582Bxx_RLA SERIAL Test Application
    SPC582Bxx Platform Component RLA
    SPC582Bxx Init Package Component RLA
    SPC582Bxx Low Level Drivers Component RLA
    [Dep] SPC582Bxx Board Initialization Component RLA
    [Dep] SPC582Bxx Clock Component RLA
    [Dep] SPC582Bxx IRQ Component RLA
  components
  source
  UDE
  pclint
  application.ld
  configuration.xml
  main.c
  Makefile
  patch.xml
  readme.txt
  user.mak
```

The FreeRTOS open source real time operating system is available on request as a separate component that is compatible with the rest of the environment.

AutoDevKit Studio also contains straightforward software examples for each peripheral in the MCU, which developers can use to become familiar with the specific code involved.

Other advantages include:

- the ability to integrate other software products from the Eclipse standard market place
- the availability of a free license GCC GNU C Compiler component
- the possibility to support industry-standard compilers
- support of multi-core microcontrollers
- a PinMap editor to facilitate MCU pin connections

7.1 How to create a new project in AutoDevKit Studio

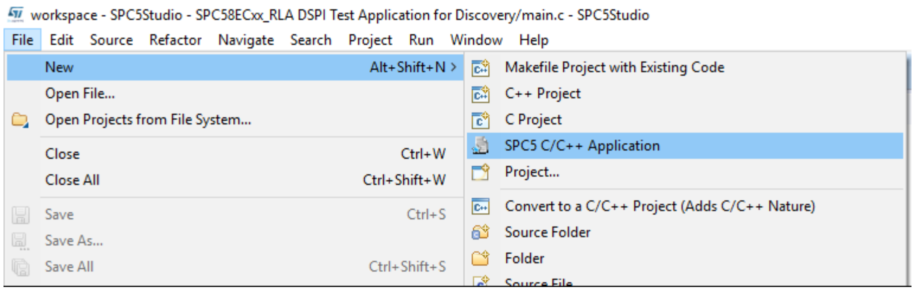
The following procedure provides step-by-step guide to create a new project in AutoDevKit Studio.

Step 1. Install AutoDevKit Studio (downloadable from www.st.com/autodevkitsw).

Step 2. Create a new SPC5 application:

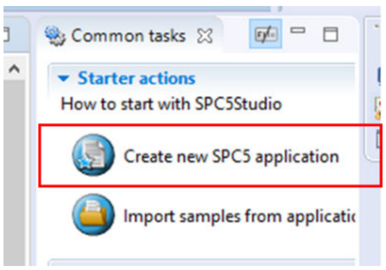
– Select [**File**] [**New**] [**SPC5 C/C++ Application**]

Figure 24. **New Project menu**



– Or select the icon “Create a new SPC5 application” present in the Starter actions tab.

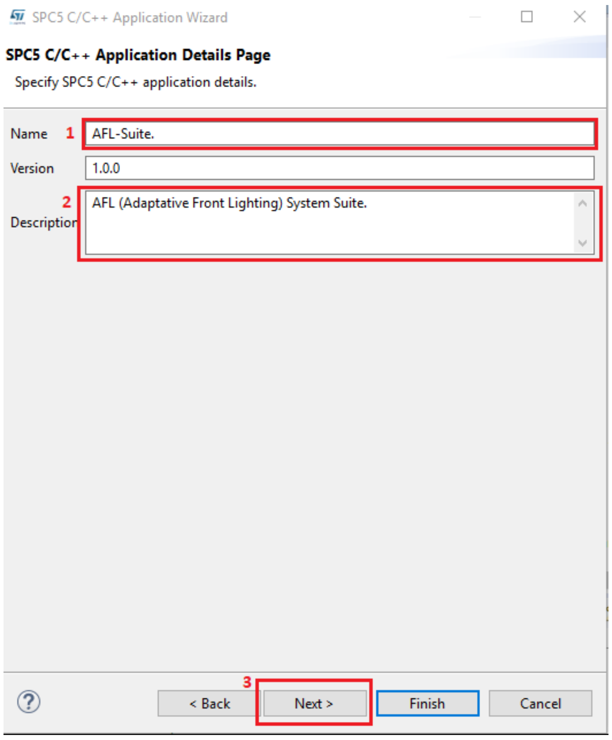
Figure 25. **Starter actions tab**


Common tasks
Starter actions
How to start with SPC5Studio
Create new SPC5 application
Import samples from applicati

A window appears, prompting for application details.

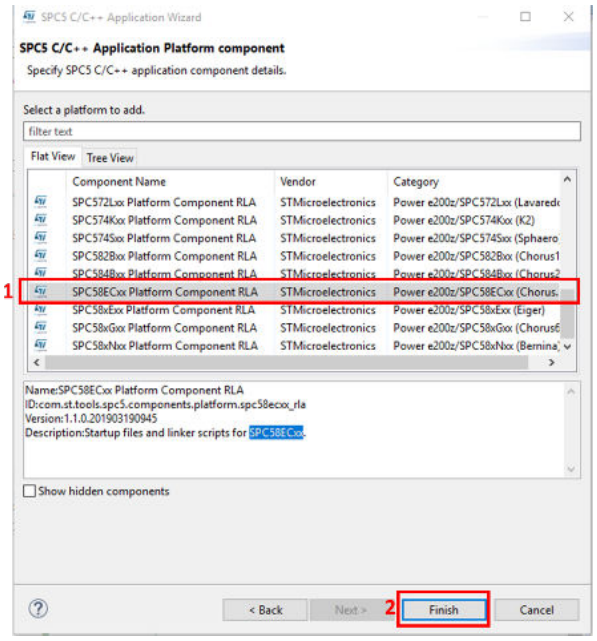
Step 3. Fill in the application details and then click on the [**Next**] Button.

Figure 26. **New application details page**


SPC5 C/C++ Application Wizard
SPC5 C/C++ Application Details Page
Specify SPC5 C/C++ application details.
1
2
3
< Back
Next >
Finish
Cancel

Name: AFL-Suite.
Version: 1.0.0
Description: AFL (Adaptative Front Lighting) System Suite.

Step 4. Select the SPC58Cxx component and select the [**Finish**] button.


SPC5 C/C++ Application Wizard

SPC5 C/C++ Application Platform component

Specify SPC5 C/C++ application component details.

Select a platform to add.

	Component Name	Vendor	Category
	SPC572Lxx Platform Component RLA	STMicroelectronics	Power e200z/SPC572Lxx (Lavaredc
	SPC574Kxx Platform Component RLA	STMicroelectronics	Power e200z/SPC574Kxx (K2)
	SPC574Sxx Platform Component RLA	STMicroelectronics	Power e200z/SPC574Sxx (Sphaero
	SPC582Bxx Platform Component RLA	STMicroelectronics	Power e200z/SPC582Bxx (Chorus1
	SPC584Bxx Platform Component RLA	STMicroelectronics	Power e200z/SPC584Bxx (Chorus2
1	SPC58ECxx Platform Component RLA	STMicroelectronics	Power e200z/SPC58ECxx (Chorus.
	SPC58xExx Platform Component RLA	STMicroelectronics	Power e200z/SPC58xExx (Eiger)
	SPC58xGxx Platform Component RLA	STMicroelectronics	Power e200z/SPC58xGxx (Chorus6
	SPC58xNxx Platform Component RLA	STMicroelectronics	Power e200z/SPC58xNxx (Bernina)

Name:SPC58ECxx Platform Component RLA
ID:com.st.tools.spc5.components.platform.spc58ecxx_rla
Version:1.1.0.201903190945
Description:Startup files and linker scripts for SPC58ECxx.

Figure 27. New application MCU platform

Step 5. Click on the [**Generate application code**] button.
A base C project is created in the workspace.

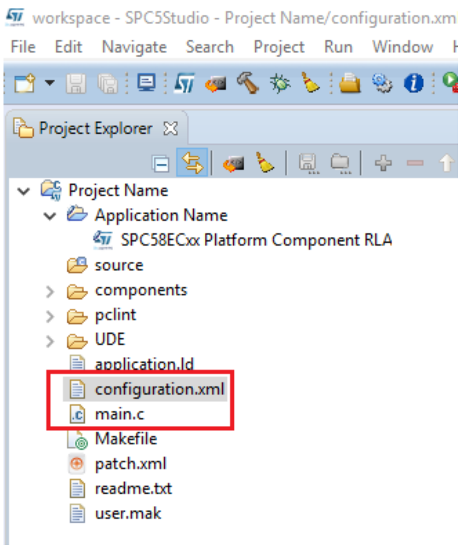
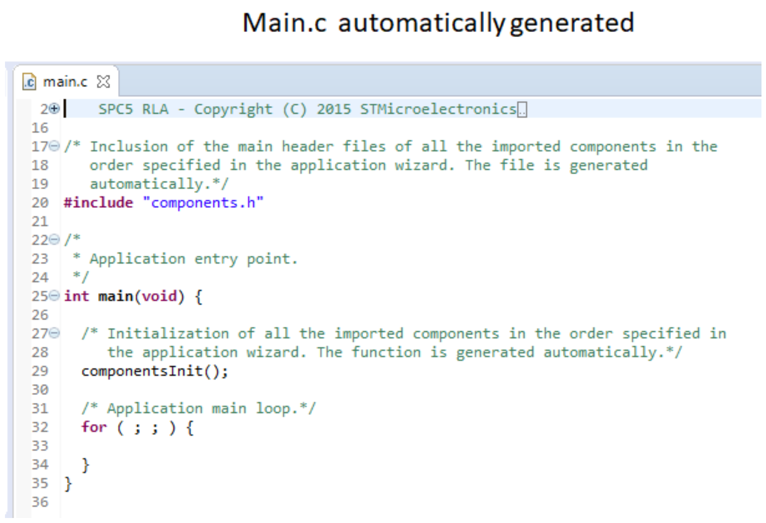


Figure 28. New SPC5 application files in outline view

There are two fundamental files visible in the application tree view:

- configuration.xml with project configuration information that is updated every time the project is changed.
- main.c in which the actual application is implemented. The main.c file only contains a basic initialization section and an infinite loop when it is first created.

Figure 29. **main.c application file**



```
  SPC5 RLA - Copyright (C) 2015 STMicroelectronics

/* Inclusion of the main header files of all the imported components in the
   order specified in the application wizard. The file is generated
   automatically.*/
#include "components.h"

/*
 * Application entry point.
 */
int main(void) {

  /* Initialization of all the imported components in the order specified in
     the application wizard. The function is generated automatically.*/
  componentsInit();

  /* Application main loop.*/
  for ( ; ; ) {

  }
}

```

7.2 How to add components to an AutoDevKit Studio project

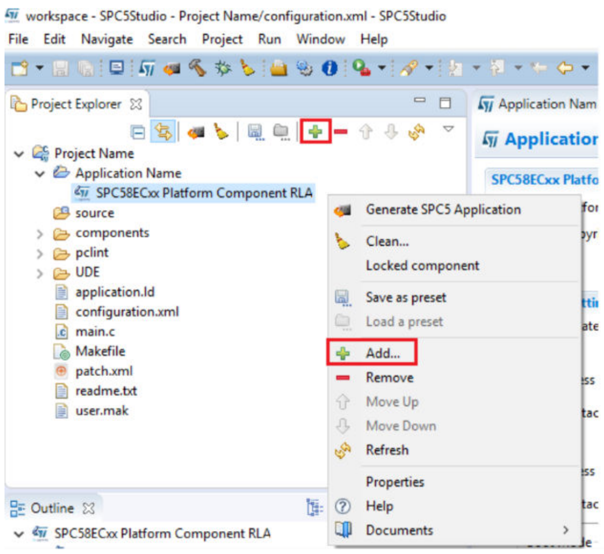
The following procedure shows how to add an available component to a project.

Step 1. In the Project Explorer tab, select the SPC58Cxx Platform component RLA.

Step 2. View the available components for the chosen platform:

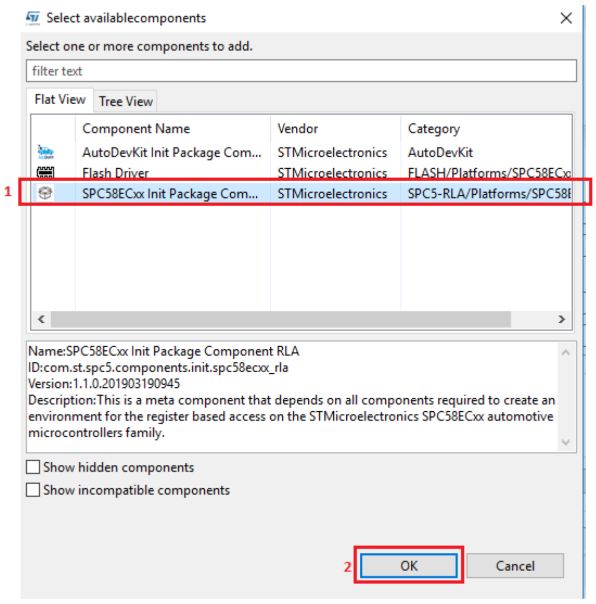
- right click on the mouse and select add, or
- select the + icon in the project explorer

Figure 30. **Open available components list**



Step 3. Select the components you want to add and click the [**OK**] button.
You must at least add the following two components to your AutoDevKit Studio Project:
– SPC58Cxx Init Package Component
– Low Level Driver

Figure 31. Select available components for platform


Select availablecomponents
Select one or more components to add.
filter text
Flat View Tree View

Component Name	Vendor	Category
AutoDevKit Init Package Com...	STMicroelectronics	AutoDevKit
Flash Driver	STMicroelectronics	FLASH/Platforms/SPC58ECx
SPC58ECxx Init Package Com...	STMicroelectronics	SPC5-RLA/Platforms/SPC58E

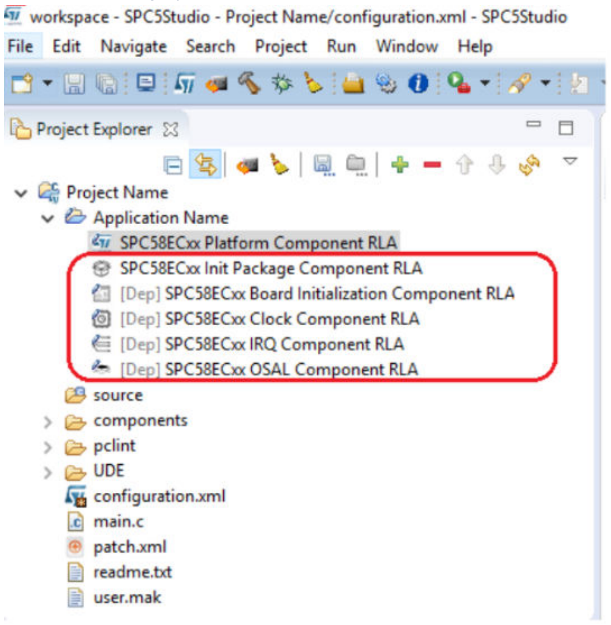
Name:SPC58ECxx Init Package Component RLA
ID:com.st.spc5.components.init.spc58ecxx_rla
Version:1.1.0.201903190945
Description:This is a meta component that depends on all components required to create an environment for the register based access on the STMicroelectronics SPC58ECxx automotive microcontrollers family.

☐ Show hidden components
☐ Show incompatible components

The added components appear in the Application Folder:

Figure 32. New components visible in project tree

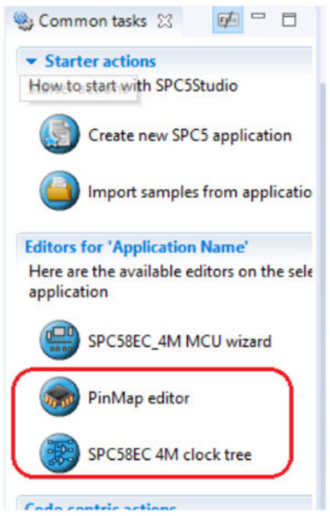
SPC58Cxx Board Initialization Component RLA: initialization and the configuration of the selected board.
SPC58Cxx Clock Component RLA: for the configuration of the MCU clock tree
SPC58Cxx IRQ Component RLA: to set and configure interrupt Request QUEUE
<SPC58Cxx> OSAL Component RLA: operating system abstraction


workspace - SPC5Studio - Project Name/configuration.xml - SPC5Studio

```
Project Name
  Application Name
    SPC58ECxx Platform Component RLA
    SPC58ECxx Init Package Component RLA
    [Dep] SPC58ECxx Board Initialization Component RLA
    [Dep] SPC58ECxx Clock Component RLA
    [Dep] SPC58ECxx IRQ Component RLA
    [Dep] SPC58ECxx OSAL Component RLA
  source
  components
  pclint
  UDE
  configuration.xml
  main.c
  patch.xml
  readme.txt
  user.mak
```

The PinMap Editor icon also becomes selectable in the Editors tab:

Figure 33. **PinMap editor icon in Editors tab**


Common tasks
Starter actions
How to start with SPC5Studio
Create new SPC5 application
Import samples from applicatio
Editors for 'Application Name'
Here are the available editors on the sele application
SPC58EC_4M MCU wizard
PinMap editor
SPC58EC 4M clock tree

Repeat the procedure to add other components to your project.

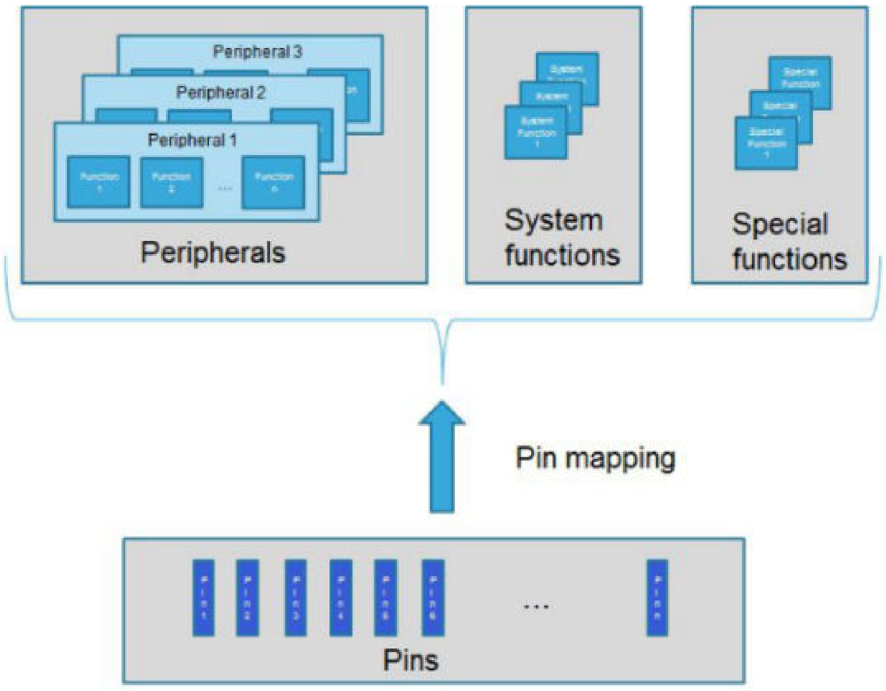
7.3 PinMap Editor

Pins in SPC5 microcontrollers are identified as Ports because they can have different functions depending on the configuration settings in certain MCU registers:

- Peripheral related pins such as CAN, ADC, eMIOS.
- System function pins such as RESET (cannot be modified).
- Special function pins such as supply voltage pins.

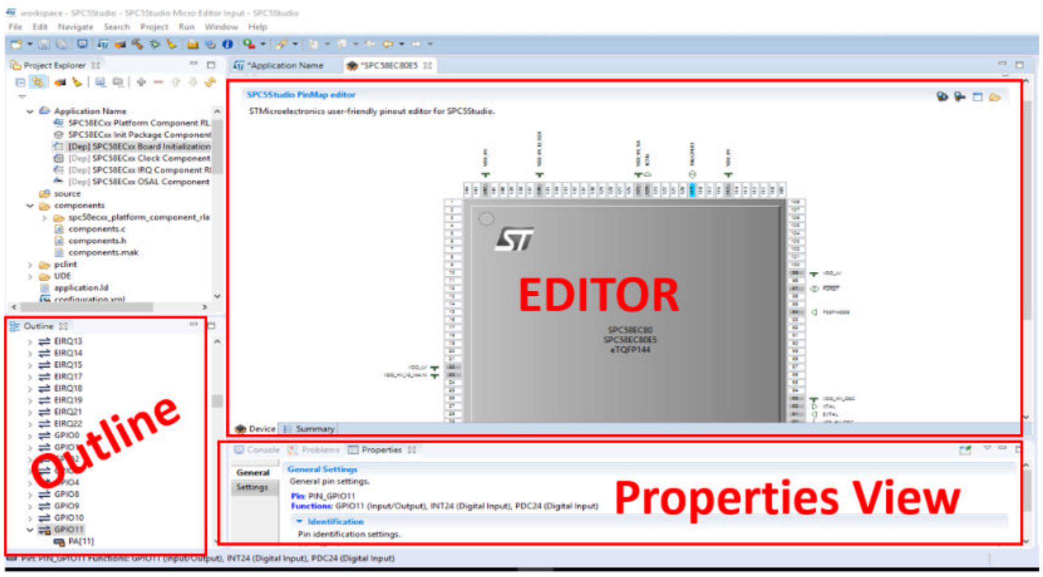
The PinMap editor supports pin functionality selection by providing the possible configurations for each selected pin.

Figure 34. **SPC5 MCU pin functions**



The PinMap editor features Editor, Outline and Properties windows.

Figure 35. **PinMap editor interface windows**



STMicroelectronics user-friendly pinout editor for SPC5Studio.

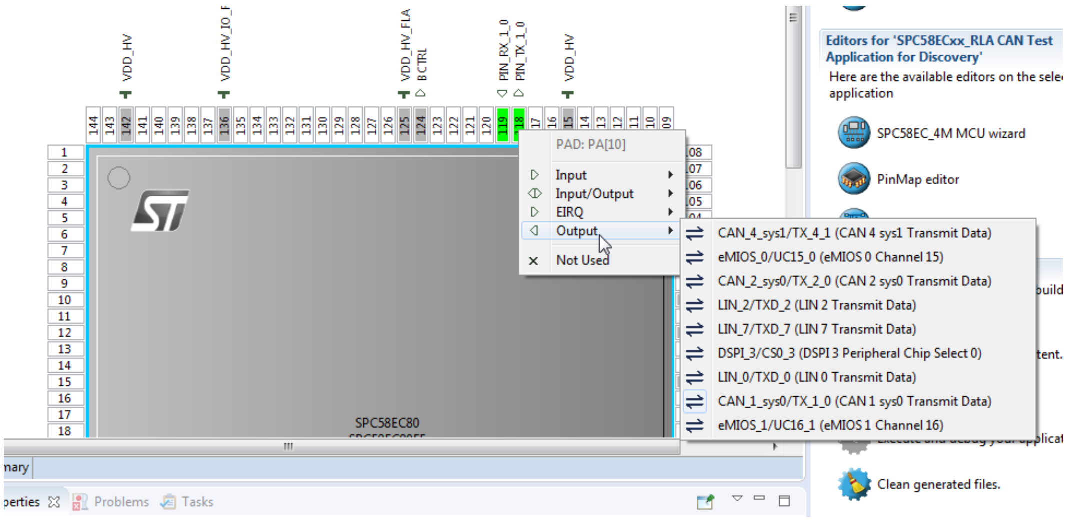
General Settings
General pin settings.
Pin: PIN_GPIO11
Functions: GPIO11 (Input/Output), INT24 (Digital Input), PDC24 (Digital Input)
Identification
Pin identification settings.

7.3.1 How to use the Editor view

The Editor view is the largest window, where the pin configuration can be performed with the help of a graphical representation of the selected MCU. The following procedure shows how to configure a pin in the Editor view.

Step 1. In the main editor window of the PinMap editor, right click on the pin you wish to edit.

Step 2. Select a signal direction and a corresponding function from the list of pre-loaded configurations that may be associated with the pin.



CAN_4_sys1/TX_4_1 (CAN 4 sys1 Transmit Data)
eMIOS_0/UC15_0 (eMIOS 0 Channel 15)
CAN_2_sys0/TX_2_0 (CAN 2 sys0 Transmit Data)
LIN_2/TXD_2 (LIN 2 Transmit Data)
LIN_7/TXD_7 (LIN 7 Transmit Data)
DSPI_3/CS0_3 (DSPI 3 Peripheral Chip Select 0)
LIN_0/TXD_0 (LIN 0 Transmit Data)
CAN_1_sys0/TX_1_0 (CAN 1 sys0 Transmit Data)
eMIOS_1/UC16_1 (eMIOS 1 Channel 16)

Figure 36. **Pin signal function selection**

7.3.2 How to use the Outline view

The Outline view is presented in a Device → Function → Pin hierarchy, which lists all the eligible pins for a particular peripheral function.

Step 1. In the Outline window select the select the DSPI peripheral.

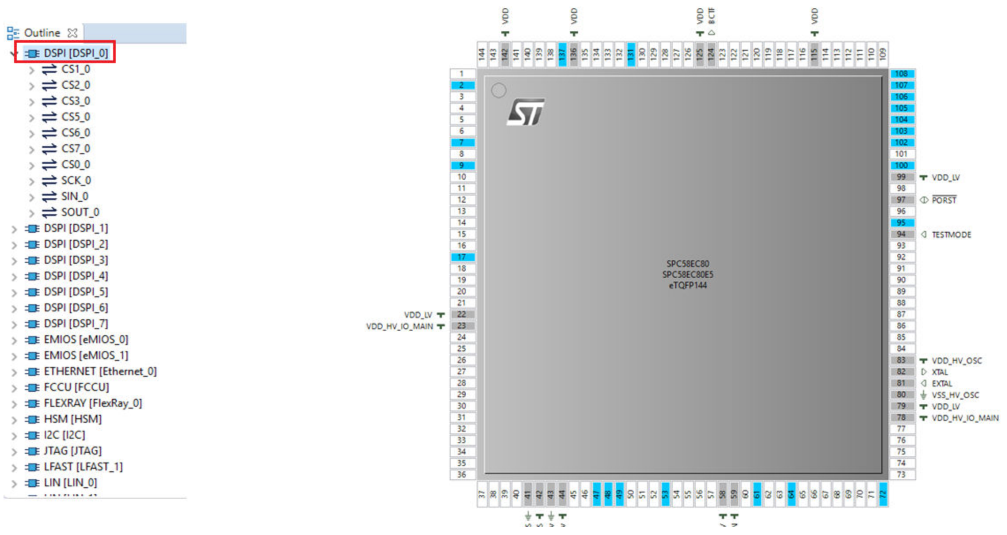


Figure 37. **Selected peripheral in Outline view and eligible pins in Editor view**

Step 2. Select an appropriate signal for the peripheral.

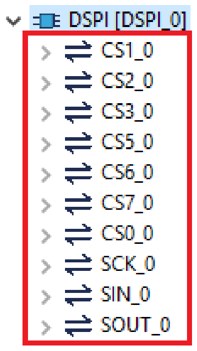

```
DSPI [DSPI_0]
  CS1_0
  CS2_0
  CS3_0
  CS5_0
  CS6_0
  CS7_0
  CS0_0
  SCK_0
  SIN_0
  SOUT_0
```

Figure 38. **Signals for DSPI peripheral**

Step 3. Select one of the pins available for the chosen signal.
The corresponding MCU ports for each pin are shown in square brackets.

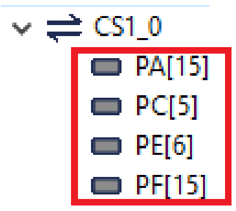
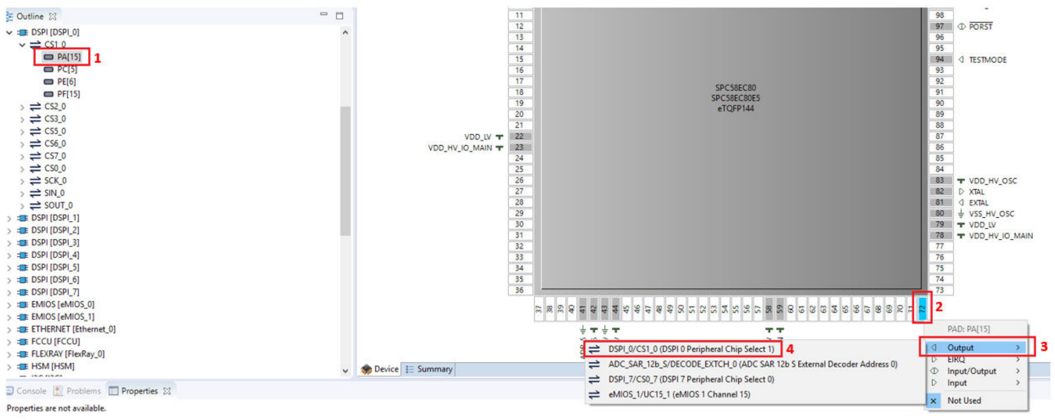

```
CS1_0
  PA[15]
  PC[5]
  PE[6]
  PF[15]
```

Figure 39. **Pins for selected peripheral signal**

Step 4. In the Editor window, right-click on the selected pin → select the signal direction → select preloaded configuration.

Step 5.


```
Outline
DSPI [DSPI_0]
  CS1_0
    PA[15]   1
    PC[5]
    PE[6]
    PF[15]
  CS2_0
  CS3_0
  CS5_0
  CS6_0
  CS7_0
  CS0_0
  SCK_0
  SIN_0
  SOUT_0
DSPI [DSPI_1]
DSPI [DSPI_2]
DSPI [DSPI_3]
DSPI [DSPI_4]
DSPI [DSPI_5]
DSPI [DSPI_6]
DSPI [DSPI_7]
EMIOS [eMIOS_0]
EMIOS [eMIOS_1]
ETHERNET [Ethernet_0]
FCCU [FCCU]
FLEXRAY [FlexRay_0]
HSM [HSM]

Console  Problems  Properties
Properties are not available.
```

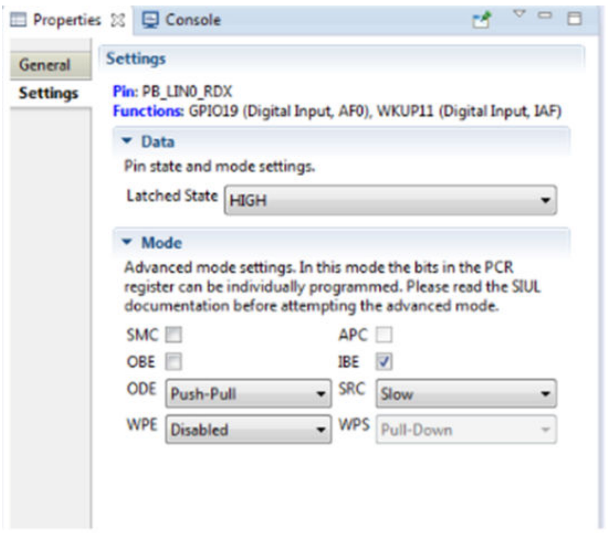
```
PAD: PA[15]
Output          >   3
EIRQ            >
Input/Output    >
Input           >
Not Used

DSPI_0/CS1_0 (DSPI 0 Peripheral Chip Select 1)   4
ADC_SAR_12b_S/DECODE_EXTCH_0 (ADC SAR 12b S External Decoder Address 0)
DSPI_7/CS0_7 (DSPI 7 Peripheral Chip Select 0)
eMIOS_1/UC15_1 (eMIOS 1 Channel 15)
```

Figure 40. **Pin configuration in Editor view**

7.3.3 The Properties view

This view provides further information for a selected pin, including name, pin state and mode settings.

Figure 41. PinMap Properties view



Properties | Console

General
Settings

Settings

Pin: PB_LIN0_RDX
Functions: GPIO19 (Digital Input, AF0), WKUP11 (Digital Input, IAF)

Data
Pin state and mode settings.
Latched State: HIGH

Mode
Advanced mode settings. In this mode the bits in the PCR register can be individually programmed. Please read the SIUL documentation before attempting the advanced mode.

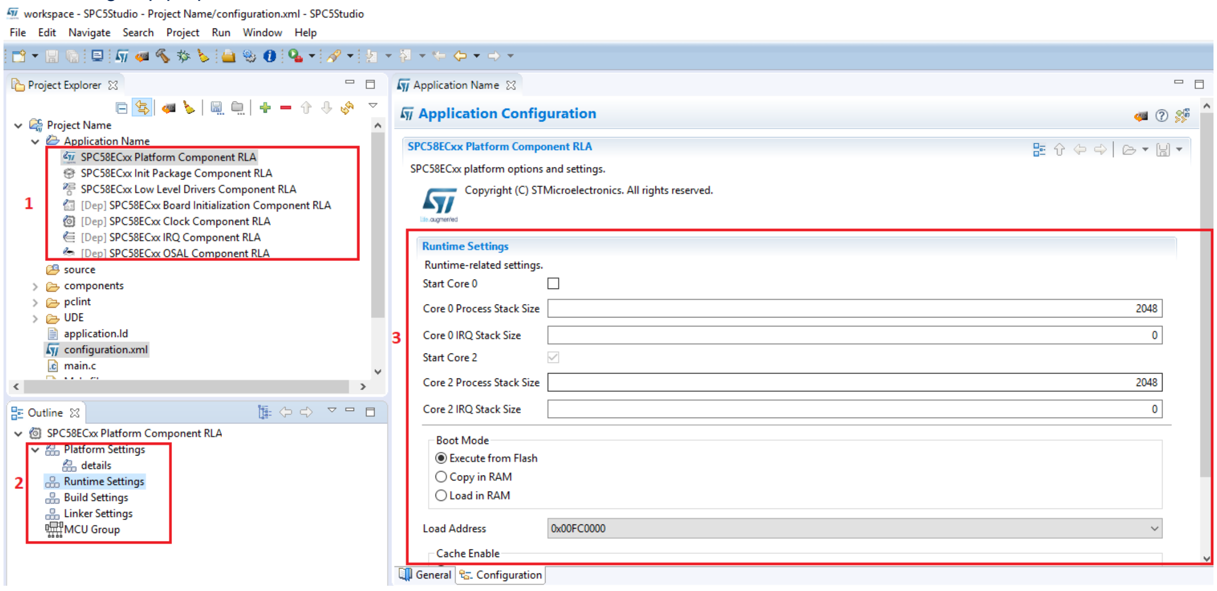
SMC ☐ APC ☐
OBE ☐ IBE ☑
ODE: Push-Pull SRC: Slow
WPE: Disabled WPS: Pull-Down

7.4 Component configuration in AutoDevKit Studio

AutoDevKit Studio includes a Configuration viewer with default component configuration values that may or may not be editable.

Figure 42. Component configuration sequence

1. Select a component to configure from your project
2. Select a group in the Outline view
3. Edit the group properties



workspace - SPC5Studio - Project Name/configuration.xml - SPC5Studio

Project Explorer
- Project Name
  - Application Name
    - 1
      - SPC58ECxx Platform Component RLA
      - SPC58ECxx Init Package Component RLA
      - SPC58ECxx Low Level Drivers Component RLA
      - [Dep] SPC58ECxx Board Initialization Component RLA
      - [Dep] SPC58ECxx Clock Component RLA
      - [Dep] SPC58ECxx IRQ Component RLA
      - [Dep] SPC58ECxx OSAL Component RLA
    - source
    - components
    - pclint
    - UDE
    - application.ld
    - configuration.xml
    - main.c

Outline
- SPC58ECxx Platform Component RLA
  - 2
    - Platform Settings
      - details
    - Runtime Settings
    - Build Settings
    - Linker Settings
    - MCU Group

Application Name

Application Configuration

SPC58ECxx Platform Component RLA
SPC58ECxx platform options and settings.
Copyright (C) STMicroelectronics. All rights reserved.

3

Runtime Settings
Runtime-related settings.

Start Core 0	☐
Core 0 Process Stack Size	2048
Core 0 IRQ Stack Size	0
Start Core 2	✓
Core 2 Process Stack Size	2048
Core 2 IRQ Stack Size	0

Boot Mode
- [✓ Execute from Flash]
- Copy in RAM
- Load in RAM

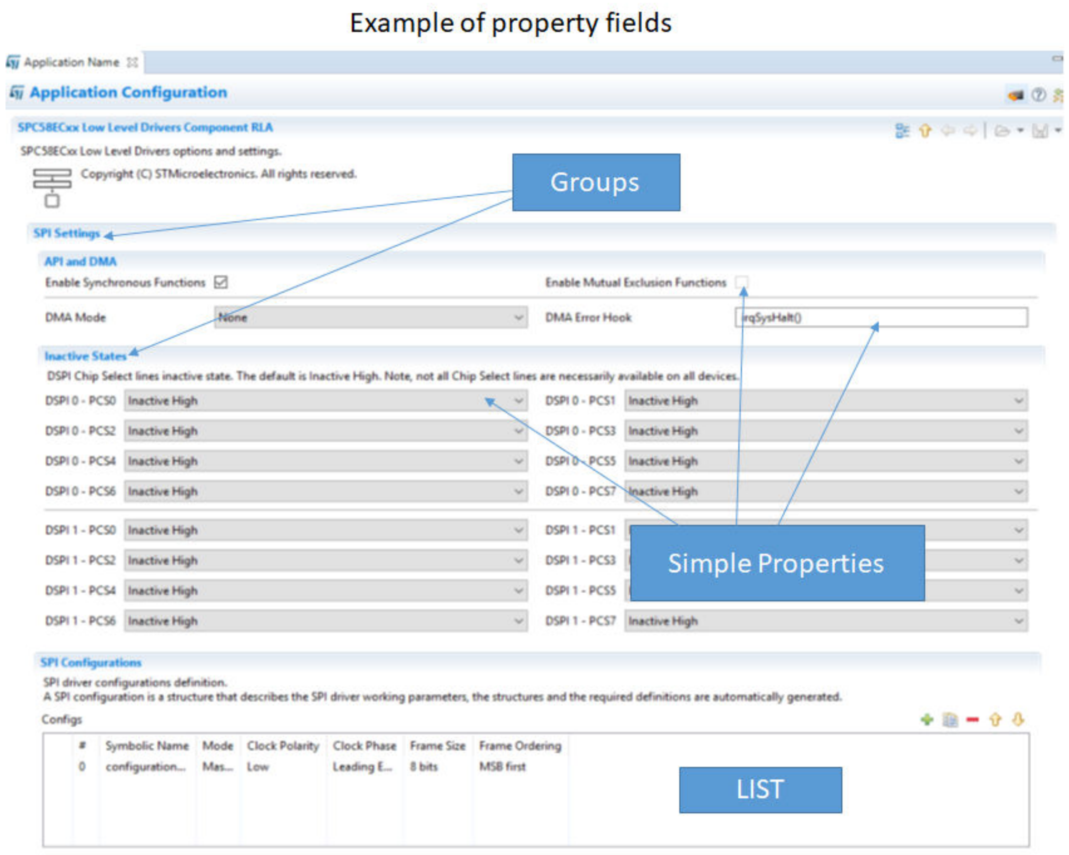
Load Address: 0x00FC0000

Cache Enable

General | Configuration

Note: *Be sure to save the project whenever you set or change a property in order to keep the changes.*

Figure 43. **Simple, group and list property fields**



Application Configuration

SPC58ECxx Low Level Drivers Component RLA

SPC58ECxx Low Level Drivers options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

SPI Settings

API and DMA

Enable Synchronous Functions ☑ Enable Mutual Exclusion Functions ☐

DMA Mode: None DMA Error Hook: irqSysHalt()

Inactive States

DSPI Chip Select lines inactive state. The default is Inactive High. Note, not all Chip Select lines are necessarily available on all devices.

DSPI 0 - PCS0	Inactive High	DSPI 0 - PCS1	Inactive High
DSPI 0 - PCS2	Inactive High	DSPI 0 - PCS3	Inactive High
DSPI 0 - PCS4	Inactive High	DSPI 0 - PCS5	Inactive High
DSPI 0 - PCS6	Inactive High	DSPI 0 - PCS7	Inactive High
DSPI 1 - PCS0	Inactive High	DSPI 1 - PCS1	
DSPI 1 - PCS2	Inactive High	DSPI 1 - PCS3	
DSPI 1 - PCS4	Inactive High	DSPI 1 - PCS5	
DSPI 1 - PCS6	Inactive High	DSPI 1 - PCS7	Inactive High

SPI Configurations

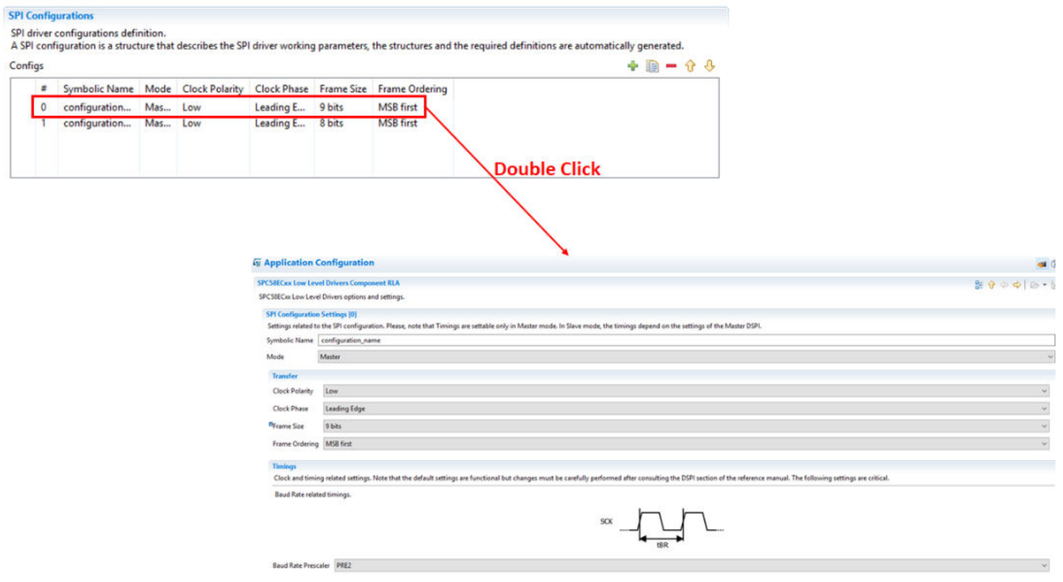
SPI driver configurations definition.
A SPI configuration is a structure that describes the SPI driver working parameters, the structures and the required definitions are automatically generated.

Configs

#	Symbolic Name	Mode	Clock Polarity	Clock Phase	Frame Size	Frame Ordering
0	configuration...	Mas...	Low	Leading E...	8 bits	MSB first

You can edit the content of a list item by double-clicking on it.

Figure 44. **Editing list items**



SPI Configurations

SPI driver configurations definition.
A SPI configuration is a structure that describes the SPI driver working parameters, the structures and the required definitions are automatically generated.

Configs

#	Symbolic Name	Mode	Clock Polarity	Clock Phase	Frame Size	Frame Ordering
0	configuration...	Mas...	Low	Leading E...	9 bits	MSB first
1	configuration...	Mas...	Low	Leading E...	8 bits	MSB first

Application Configuration

SPC58ECxx Low Level Drivers Component RLA

SPC58ECxx Low Level Drivers options and settings.

SPI Configuration Settings [0]

Settings related to the SPI configuration. Please, note that Timings are settable only in Master mode. In Slave mode, the timings depend on the settings of the Master DSPI.

Symbolic Name: configuration_name

Mode: Master

Transfer

Clock Polarity: Low

Clock Phase: Leading Edge

Frame Size: 9 bits

Frame Ordering: MSB first

Timings

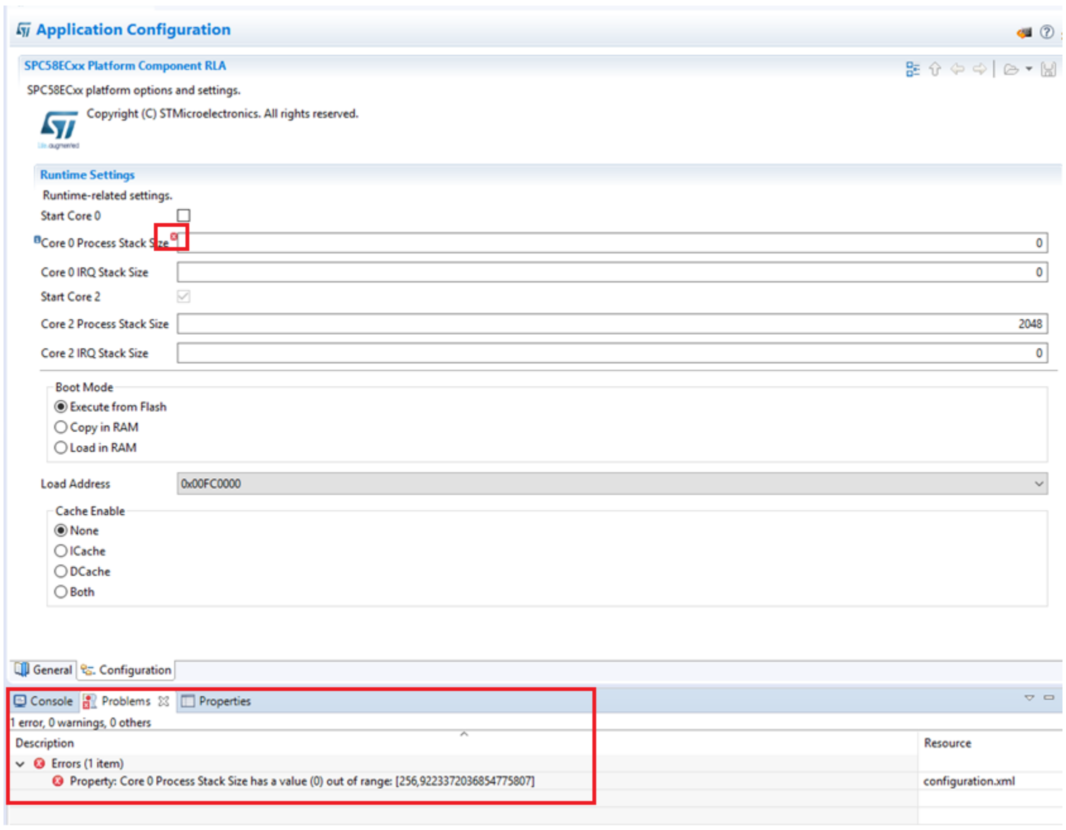
Clock and timing related settings. Note that the default settings are functional but changes must be carefully performed after consulting the DSPI section of the reference manual. The following settings are critical.

Baud Rate related timings.

Baud Rate Prescaler: PRE2

Values assigned to a component property are automatically validated and a red “x” appears in the top left corner of the name if an error is detected; a corresponding explanation is provided in the Problems tab.

Figure 45. **Validation of property values**



Application Configuration

SPC58ECxx Platform Component RLA

SPC58ECxx platform options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

Runtime Settings

Runtime-related settings.

Start Core 0	☐
Core 0 Process Stack Size	0
Core 0 IRQ Stack Size	0
Start Core 2	✓
Core 2 Process Stack Size	2048
Core 2 IRQ Stack Size	0

Boot Mode
- [✓ Execute from Flash]
- Copy in RAM
- Load in RAM

Load Address: 0x00FC0000

Cache Enable
- [✓ None]
- ICache
- DCache
- Both

General | Configuration

Console | Problems | Properties

1 error, 0 warnings, 0 others

Description	Resource
Errors (1 item)	
Property: Core 0 Process Stack Size has a value (0) out of range: [256,9223372036854775807]	configuration.xml

When the configuration of the embedded system is completed and there are no errors in the Problem tab, you can select the [**Code generation**] command to output the configuration files.

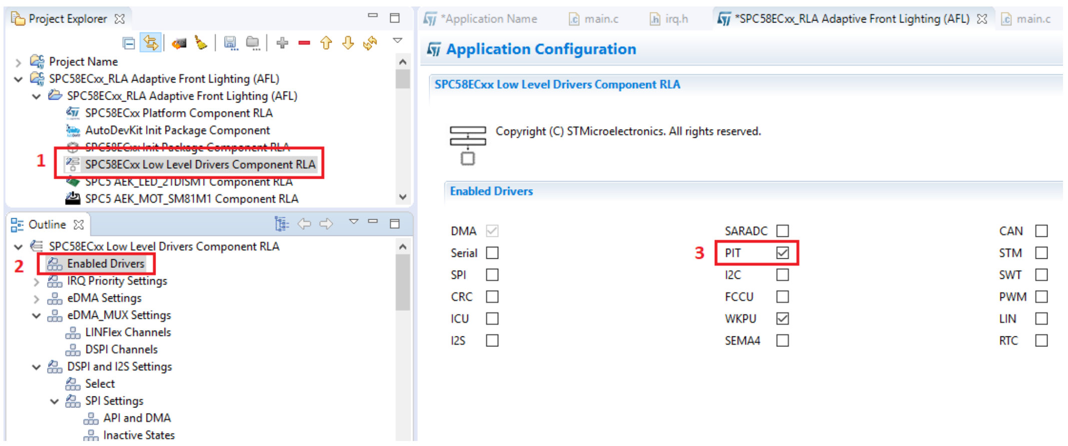
7.5 How to set low level driver IRQ priority settings

The SPC58 Chorus MCU used in the AFL-Demo kit has several interrupts of different types, which can be associated with events linked to peripherals such as DSPI, LIN, PIT, or to other components. These interrupts should be prioritized in order to avoid potential conflicts.

Step 1. Select the low level driver component.

Step 2. In the Outline tab, select Enabled Drivers.

Step 3. Check the drivers to be enabled.



Project Explorer

- Project Name
- SPC58ECxx_RLA Adaptive Front Lighting (AFL)
  - SPC58ECxx_RLA Adaptive Front Lighting (AFL)
    - SPC58ECxx Platform Component RLA
    - AutoDevKit Init Package Component
    - SPC58ECxx Init Package Component RLA
    - 1 SPC58ECxx Low Level Drivers Component RLA
    - SPC5 AEK_LED_21DISM1 Component RLA
    - SPC5 AEK_MOT_SM81M1 Component RLA

Outline

- SPC58ECxx Low Level Drivers Component RLA
  - 2 Enabled Drivers
  - IRQ Priority Settings
  - eDMA Settings
  - eDMA_MUX Settings
    - LINFlex Channels
    - DSPI Channels
  - DSPI and I2S Settings
    - Select
    - SPI Settings
      - API and DMA
      - Inactive States

Application Configuration

SPC58ECxx Low Level Drivers Component RLA

Copyright (C) STMicroelectronics. All rights reserved.

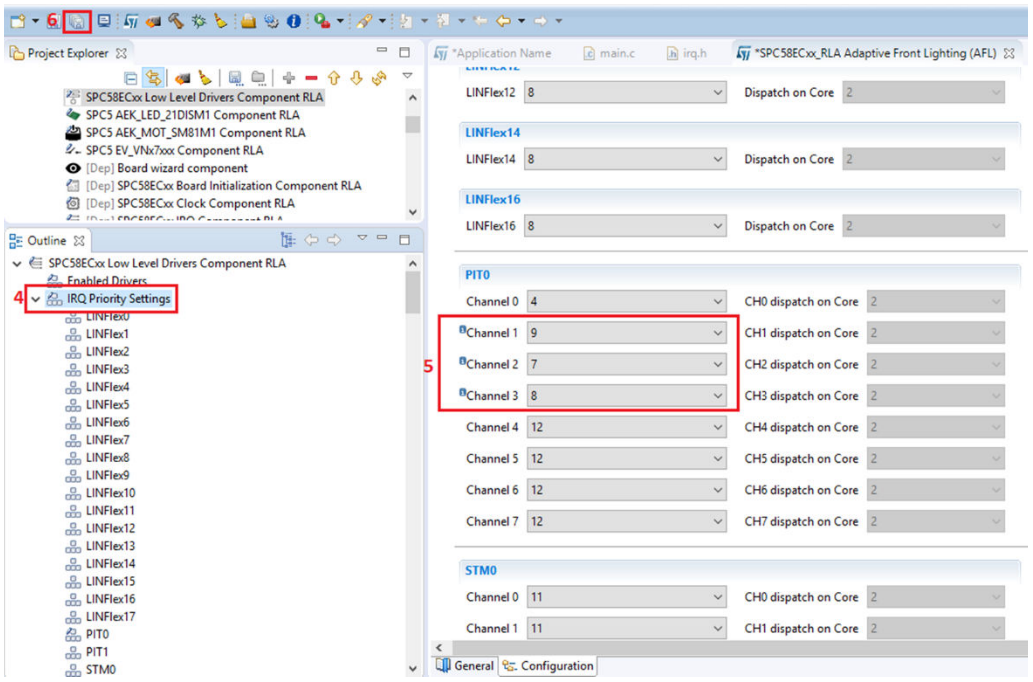
Enabled Drivers

DMA	✓	SARADC	☐	CAN	☐
Serial	☐	3 PIT	✓	STM	☐
SPI	☐	I2C	☐	SWT	☐
CRC	☐	FCCU	☐	PWM	☐
ICU	☐	WKPU	✓	LIN	☐
I2S	☐	SEMA4	☐	RTC	☐

Figure 46. Enabling low level drivers

Step 4. In the Outline tab, select IRQ Property Settings

Step 5. In the configuration Editor, search the group that matches the driver and change the priority according to your needs.



Project Explorer

- SPC58ECxx Low Level Drivers Component RLA
- SPC5 AEK_LED_21DISM1 Component RLA
- SPC5 AEK_MOT_SM81M1 Component RLA
- SPC5 EV_VNx7xxx Component RLA
- [Dep] Board wizard component
- [Dep] SPC58ECxx Board Initialization Component RLA
- [Dep] SPC58ECxx Clock Component RLA

Outline

- SPC58ECxx Low Level Drivers Component RLA
  - Enabled Drivers
  - 4 IRQ Priority Settings
    - LINFlex0, LINFlex1, LINFlex2, LINFlex3, LINFlex4, LINFlex5, LINFlex6, LINFlex7, LINFlex8, LINFlex9, LINFlex10, LINFlex11, LINFlex12, LINFlex13, LINFlex14, LINFlex15, LINFlex16, LINFlex17, PIT0, PIT1, STM0

LINFlex12	8	Dispatch on Core	2
LINFlex14			
LINFlex14	8	Dispatch on Core	2
LINFlex16			
LINFlex16	8	Dispatch on Core	2
PIT0			
Channel 0	4	CH0 dispatch on Core	2
Channel 1	9	CH1 dispatch on Core	2
Channel 2	7	CH2 dispatch on Core	2
Channel 3	8	CH3 dispatch on Core	2
Channel 4	12	CH4 dispatch on Core	2
Channel 5	12	CH5 dispatch on Core	2
Channel 6	12	CH6 dispatch on Core	2
Channel 7	12	CH7 dispatch on Core	2
STM0			
Channel 0	11	CH0 dispatch on Core	2
Channel 1	11	CH1 dispatch on Core	2

Figure 47. IRQ priority settings

7.6 How to generate and compile application source code

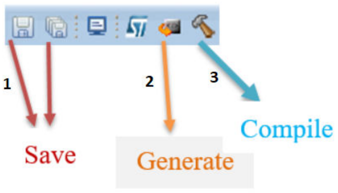
To generate and compile the application source code, follow the procedure below described:

Step 1. Save the project.

Step 2. Click on the [**Generate**] icon to update the configuration files.

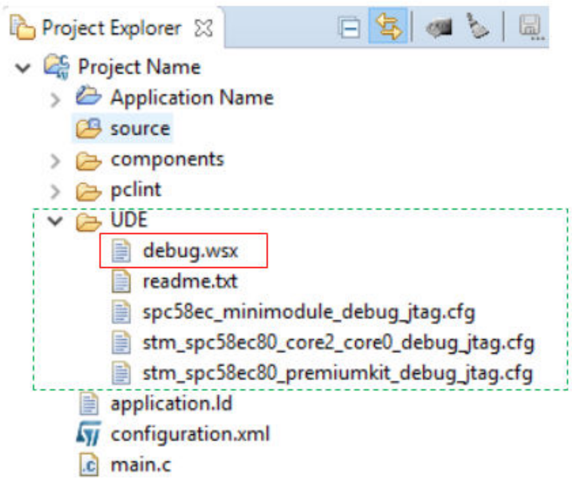
Step 3. Click on the [**Compile**] icon to compile the project source and produce the MCU Flash image.

Figure 48. **AutoDevKit Studio save, generate and compile icons**



Once the source code is compiled and linked, a file named debug.wsx will appear in the UDE folder.

Figure 49. **Compiled project files**



```
Project Explorer
Project Name
  Application Name
  source
  components
  pclint
  UDE
    debug.wsx
    readme.txt
    spc58ec_minimodule_debug_jtag.cfg
    stm_spc58ec80_core2_core0_debug_jtag.cfg
    stm_spc58ec80_premiumkit_debug_jtag.cfg
  application.ld
  configuration.xml
  main.c
```

Note: Users implement their application in the `main()` *function contained in the main.c file, which is automatically created by AutoDevKit Studio in the project creation phase.*

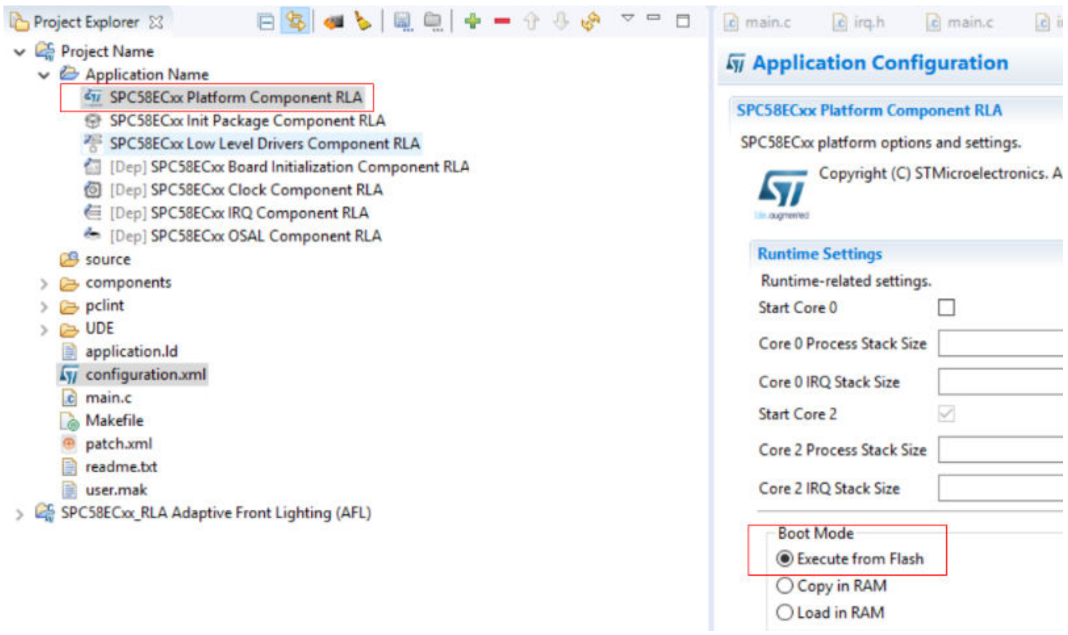
7.7 How to debug an application generated with AutoDevKit Studio

Before you begin debugging, you should install Universal Debug Engine (UDE) tool, which you can download from the SPC5-UDESTK-SW web page on the ST website. An alternative is represented by the OpenOCD programmer/debugger integrated in AutoDevKit Studio.

The procedure below shows how to debug the code generated in AutoDevKit Studio using the (UDE) tool.

Step 1. In AutoDevKit Studio, check the [**Execute from Flash**] radio button in the Boot Mode section of the Application Configuration view.

Figure 50. **Boot Mode radio button in Application Configuration view**



Project Explorer

- Project Name
  - Application Name
    - SPC58ECxx Platform Component RLA
    - SPC58ECxx Init Package Component RLA
    - SPC58ECxx Low Level Drivers Component RLA
    - [Dep] SPC58ECxx Board Initialization Component RLA
    - [Dep] SPC58ECxx Clock Component RLA
    - [Dep] SPC58ECxx IRQ Component RLA
    - [Dep] SPC58ECxx OSAL Component RLA
  - source
  - components
  - pclint
  - UDE
  - application.ld
  - configuration.xml
  - main.c
  - Makefile
  - patch.xml
  - readme.txt
  - user.mak
- SPC58ECxx_RLA Adaptive Front Lighting (AFL)

main.c | irq.h | main.c

Application Configuration

SPC58ECxx Platform Component RLA

SPC58ECxx platform options and settings.

Copyright (C) STMicroelectronics. A

Runtime Settings

Runtime-related settings.

Start Core 0	☐
Core 0 Process Stack Size	
Core 0 IRQ Stack Size	
Start Core 2	✓
Core 2 Process Stack Size	
Core 2 IRQ Stack Size	

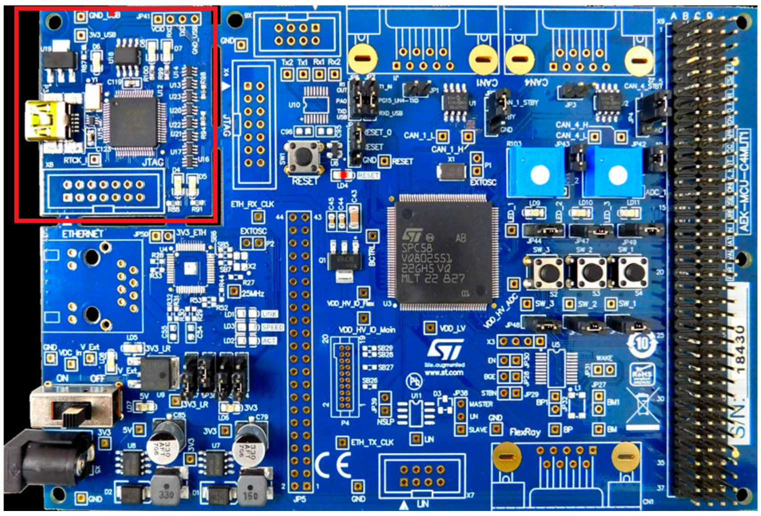
Boot Mode

[✓ Execute from Flash] Copy in RAM Load in RAM

Step 2. Compile the project in AutoDevKit Studio.

Step 3. Connect a USB cable between the mini-USB connector of the PLS on the AEK-MCU-C4MLIT1 board and the PC.

Figure 51. **mini-USB port on AEK-MCU-C4MLIT1 board**

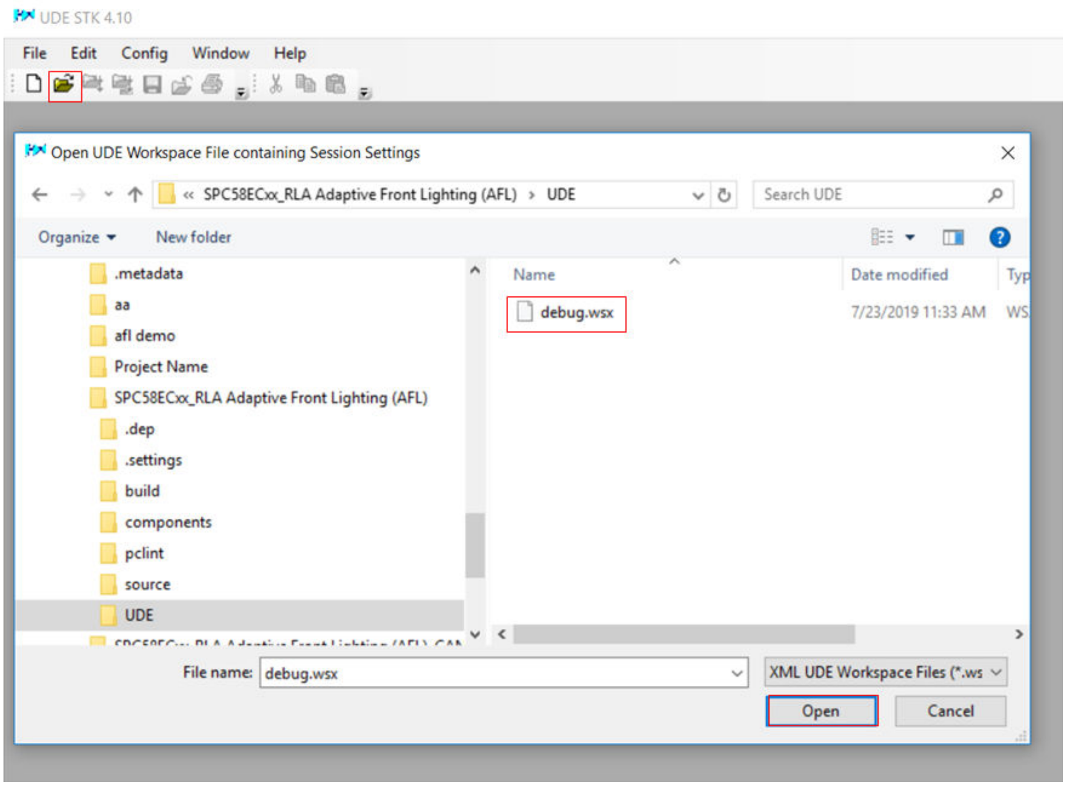


Step 4. Run the UDE tool.

Step 5. Open the project workspace folder and select debug.wsx from the UDE folder.

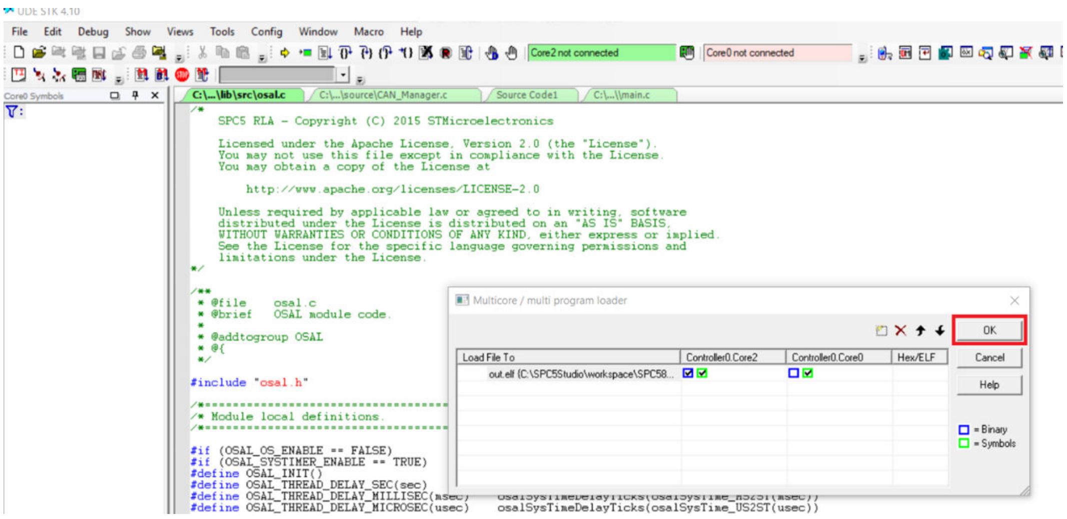
Step 6. Click the [**Open**] button.

Figure 52. Open debug.wsx in the UDE debugging tool



Step 7. Click [**Ok**] in the popup window.

Figure 53. UDE program loader

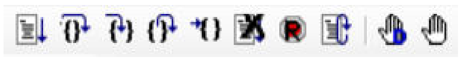


```
/*
    SPC5 RLA - Copyright (C) 2015 STMicroelectronics

    Licensed under the Apache License, Version 2.0 (the "License").
    You may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
*/

/**
 * @file    osal.c
 * @brief   OSAL module code.
 *
 * @addtogroup OSAL
 * @{
 */

#include "osal.h"

/*==================================
/* Module local definitions.
/*==================================

#if (OSAL_OS_ENABLE == FALSE)
#if (OSAL_SYSTIMER_ENABLE == TRUE)
#define OSAL_INIT()
#define OSAL_THREAD_DELAY_SEC(sec)
#define OSAL_THREAD_DELAY_MILLISEC(msec)     osalSysTimeDelayTicks(osalSysTime_MS2ST(msec))
#define OSAL_THREAD_DELAY_MICROSEC(usec)     osalSysTimeDelayTicks(osalSysTime_US2ST(usec))
```

Load File To	Controller0.Core2	Controller0.Core0	Hex/ELF
out.elf (C:\SPC5Studio\workspace\SPC58...	🟦 ☑	🟦 ☑	

Step 8. Proceed to debug using the command buttons in the Debug task bar or under the Debug menu.

Figure 54. UDE debug command buttons



7.8 Adding functional board components to an AutoDevKit project

Each functional board component only appears in the list of the MCUs able to support it, and there is a Release Note component for all MCU platforms, which provides a summary of the components installed for the functional boards.

Related links

7.4 Component configuration in AutoDevKit Studio on page 33

Check the AutoDevKit page on the ST website for the latest updates

7.9 How to configure a functional board component

The configuration of a functional board component is very similar to configuring other components in AutoDevKit Studio, the difference being two new features in the Configuration View for Pin Allocation:

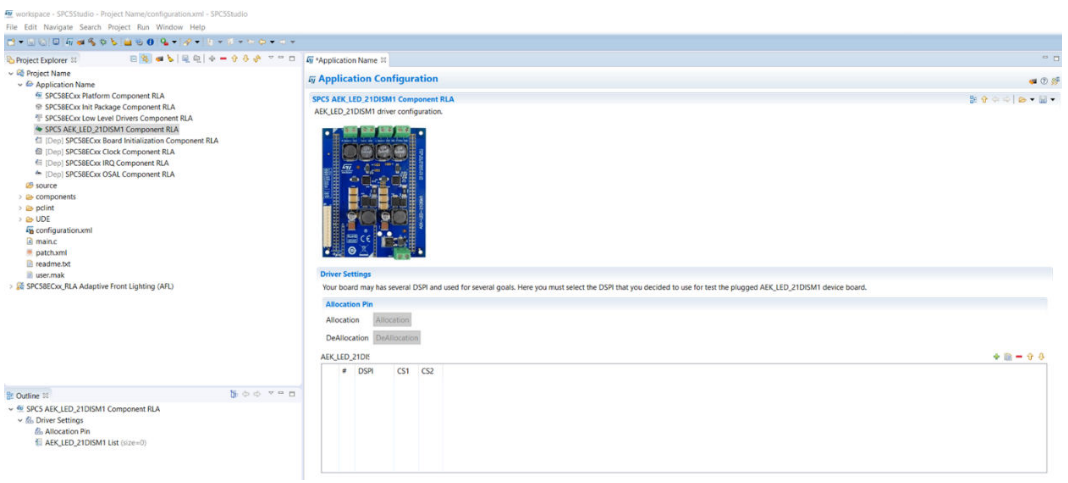
- Allocation
- Deallocation

These features automatically allocate or deallocate available MCU pins to allow the use of the MCU peripherals by newly added components.

To enable these two features, you must first configure the newly added component. The procedure is best illustrated with the following example, in which we add and configure a LED Driver component AEK-LED-21DISM1 that manages four LED strings.

Step 1. Add the AEK_LED_21DISM1 component to the project.

Figure 55. Add LED driver component to project


workspace - SPC5Studio - Project Name/configuration.xml - SPC5Studio
File Edit Navigate Search Project Run Window Help
Project Explorer
*Application Name
Application Configuration
Outline

```
Project Name
  Application Name
    SPC58ECxx Platform Component RLA
    SPC58ECxx Init Package Component RLA
    SPC58ECxx Low Level Drivers Component RLA
    SPC5 AEK_LED_21DISM1 Component RLA
    [Dep] SPC58ECxx Board Initialization Component RLA
    [Dep] SPC58ECxx Clock Component RLA
    [Dep] SPC58ECxx IRQ Component RLA
    [Dep] SPC58ECxx OSAL Component RLA
  source
  components
  pclint
  UDE
  configuration.xml
  main.c
  patch.xml
  readme.txt
  user.mak
SPC58ECxx_RLA Adaptive Front Lighting (AFL)
```

SPC5 AEK_LED_21DISM1 Component RLA
AEK_LED_21DISM1 driver configuration.

Driver Settings
Your board may has several DSPI and used for several goals. Here you must select the DSPI that you decided to use for test the plugged AEK_LED_21DISM1 device board.

Allocation Pin
Allocation: Allocation
DeAllocation: DeAllocation

AEK_LED_21DI!

#	DSPI	CS1	CS2

```
SPC5 AEK_LED_21DISM1 Component RLA
  Driver Settings
    Allocation Pin
    AEK_LED_21DISM1 List (size=0)
```

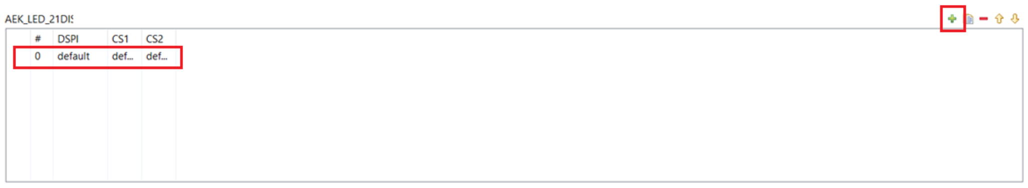
Step 2. Follow the steps below to configure the component directly.

This is different to the normal handling of other AutoDevKit Studio components where we usually proceed to verify pin allocation in the PinMap Editor.

Step 3. Click on the [**+**] button available at the top of the AEK_LED_21DISM1 table.

Step 4. Double-click on new DSPI object to open its configuration view.

Figure 56. Add new DSPI object

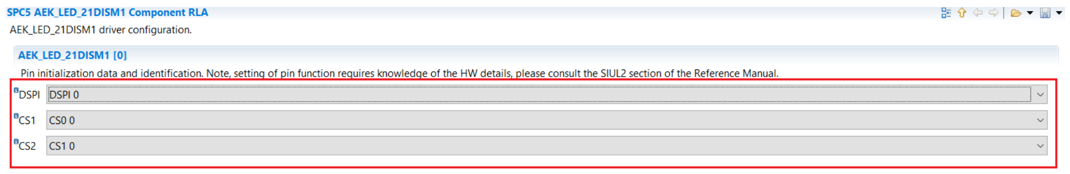


AEK_LED_21DI!

#	DSPI	CS1	CS2
0	default	def...	def...

Step 5. Select a value from the drop-down lists according to your requirements.

Figure 57. LED driver board pin configuration



SPC5 AEK_LED_21DISM1 Component RLA

AEK_LED_21DISM1 driver configuration.

AEK_LED_21DISM1 [0]

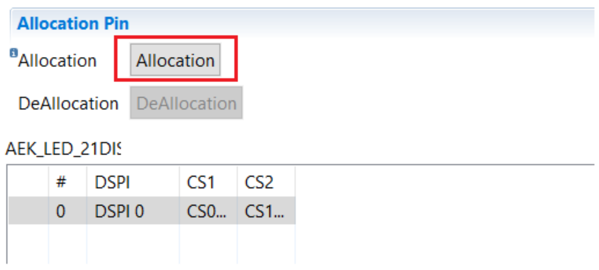
Pin initialization data and identification. Note, setting of pin function requires knowledge of the HW details, please consult the SIUL2 section of the Reference Manual.

DSPI	DSPI 0
CS1	CS0 0
CS2	CS1 0

Step 6. Select the button to return to the parent screen.

Step 7. Click the [**Allocation**] button to delegate the automatic pin allocation to AutoDevKit.

Figure 58. Pin allocation button



Allocation Pin

Allocation: Allocation

DeAllocation: DeAllocation

AEK_LED_21DIS

#	DSPI	CS1	CS2
0	DSPI 0	CS0...	CS1...

Step 8. Click [**OK**] in the confirmation window.
You can open the PinMap editor to check that the pins have been allocated correctly.

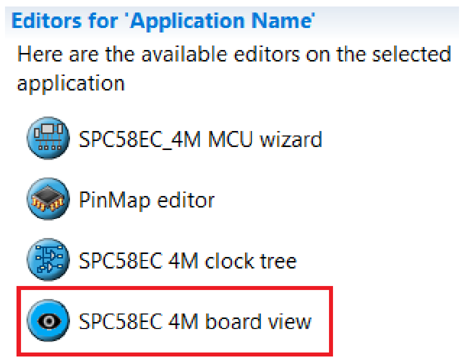
Related links

7.4 Component configuration in AutoDevKit Studio on page 33

7.10 Board View tab

The Board View tab introduced with the AutoDevKit Init Package Component shows how MCU pins are mapped on the available board connectors. The view also provides dedicated tables to represent how the MCU allocated pins are connected to functional boards added from the AutoDevKit component library. Each table shows the mapping between functional board connectors and pins and corresponding MCU board connectors and pins. This information can render the wiring phase less time-consuming and error-prone.

Figure 59. Board View button

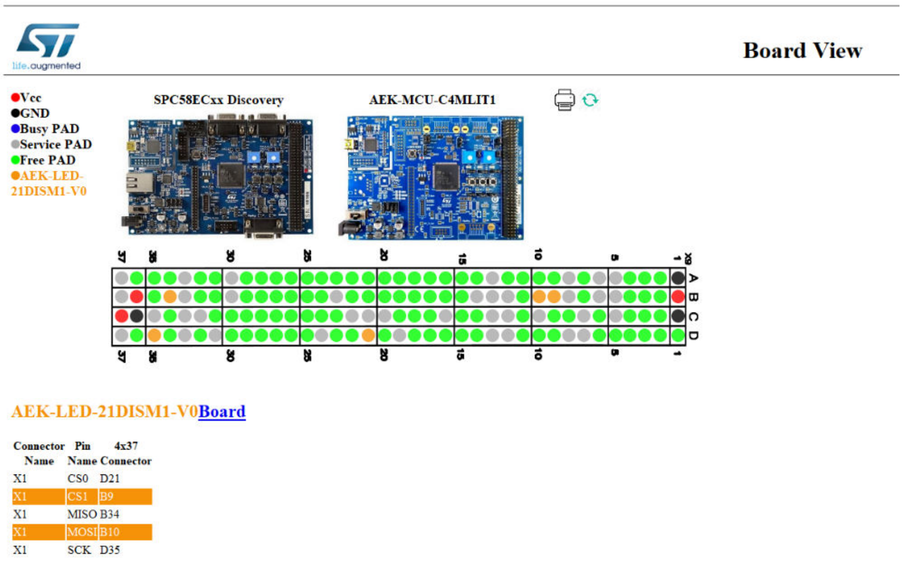


Editors for 'Application Name'

Here are the available editors on the selected application

Note: You must add the AutoDevKit Init Package Component to your project to render the Board View button visible and to see other AutoDevKit components (functional boards).

Figure 60. **Board View editor window**



●Vcc
●GND
●Busy PAD
●Service PAD
●Free PAD
●AEK-LED-21DISM1-V0

AEK-LED-21DISM1-V0Board

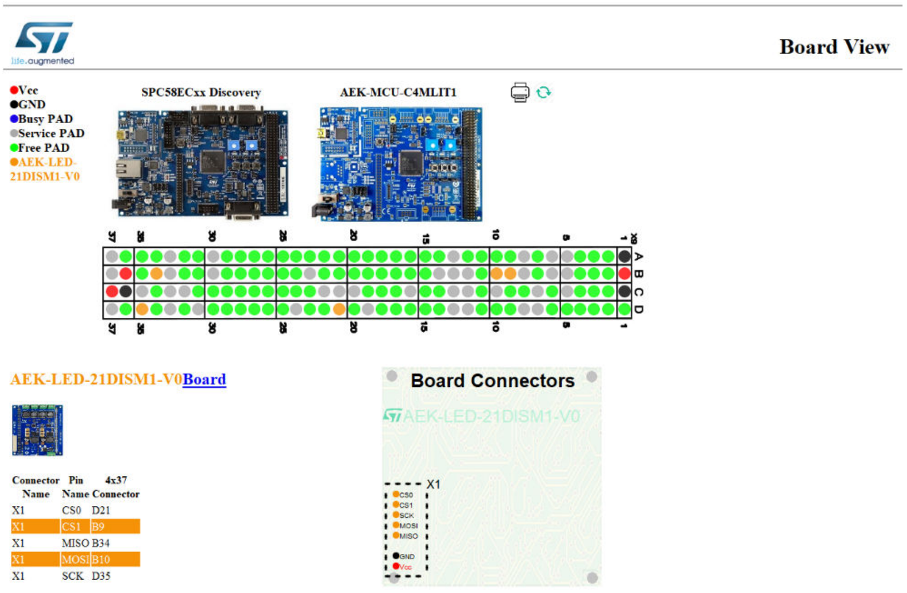
Connector Name	Pin Name	4x37 Connector
X1	CS0	D21
X1	CS1	B9
X1	MISO	B34
X1	MOSI	B10
X1	SCK	D35

In the example below, the colored dots in the 4x37 connector image help identify the pins required to wire the functional board to the MCU board. Some pins are required for the AEK-LED-21DISM1 board instance V0, while others are unused or reserved.

Note: *If pins are allocated directly in the PinMap Editor without using the component automatic allocation feature, these pins appear as Busy in the Board View.*

Clicking the ‘Board’ link next to each table reveals the functional board connector and pin locations.

Figure 61. **Board View editor window with board connectors**



●Vcc
●GND
●Busy PAD
●Service PAD
●Free PAD
●AEK-LED-21DISM1-V0

AEK-LED-21DISM1-V0Board

Connector Name	Pin Name	4x37 Connector
X1	CS0	D21
X1	CS1	B9
X1	MISO	B34
X1	MOSI	B10
X1	SCK	D35

The top rows are frozen, so the MCU board 4x37 connector always remains visible.

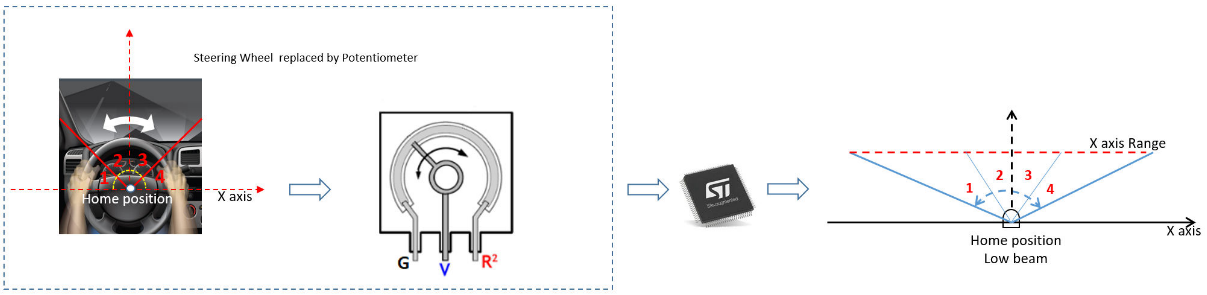
Use the refresh logo to update the view after project modifications.

Note: *To improve the view, you may need to resize the editor window or generate a printable version by clicking the printer icon.*

Appendix A Steering wheel turn simulation with a potentiometer

A potentiometer (POT) can be used to simulate steering wheel rotation in an adaptive front lighting platform where the headlight low beam direction is adjusted accordingly.

Figure 62. Representing a steering wheel with a potentiometer in an AFL system



The Potentiometer (POT) acts as a position transducer and its output voltage is used to control the low beam position via the MCU. The POT output voltage signal is fed to the MCU through one of its ADC converter channels, and then manipulated by the MCU firmware to rotate the low-beam headlight accordingly.

If we define four POT output voltage ranges to represent four low-beam positions, then the motor will have a span of five steps in total. The number of voltage ranges and consequently the number of steps can be increased to raise the sensitivity of the system.

A DC motor can be used to form a continuous system instead of a discrete stepper motor system, as long as appropriate hysteresis is introduced to reduce headlight wobble.

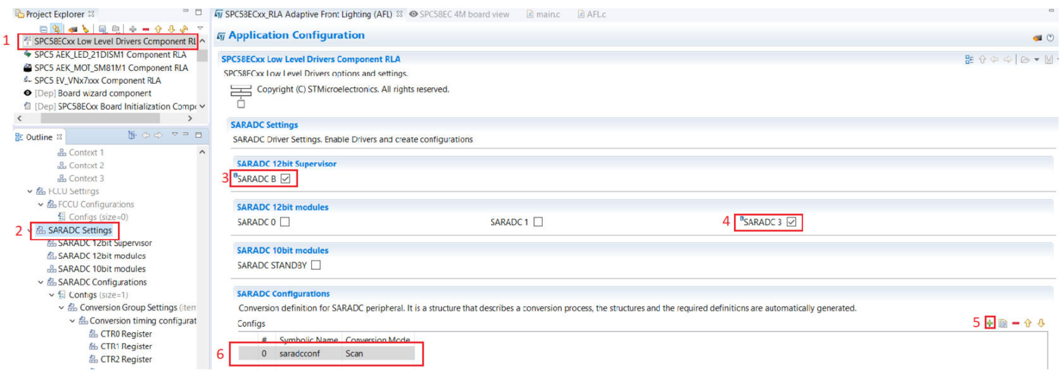
A.1 How to edit the AFL-Demo code to accept potentiometer inputs: Manual Mode

Step 1. Load the AFL-Demo source code and open the PinMap editor.

Step 2. Connect the potentiometer output voltage to an MCU ADC channel pin and set it in the PinMap editor. In the AFL-Demo, pin 51 is the designated ADC channel.

Step 3. Configure the ADC sampling frequency.

Figure 63. ADC channel configuration


Project Explorer
1
SPC58ECxx Low Level Drivers Component RL
SPC5 AEK_LED_21DISM1 Component RLA
SPC5 AEK_MOT_SM81M1 Component RLA
SPC5 EV_VNx7xxx Component RLA
[Dep] Board wizard component
[Dep] SPC58ECxx Board Initialization Compo
Outline
Context 1
Context 2
Context 3
FCCU Settings
FCCU Configurations
Configs (size=0)
2
SARADC Settings
SARADC 12bit Supervisor
SARADC 12bit modules
SARADC 10bit modules
SARADC Configurations
Configs (size=1)
Conversion Group Settings (item
Conversion timing configurat
CTR0 Register
CTR1 Register
CTR2 Register
SPC58ECxx_RLA Adaptive Front Lighting (AFL)
SPC58EC 4M board view
main.c
AFL.c
Application Configuration
5
6

SPC58ECxx Low Level Drivers Component RLA
SPC58ECxx Low Level Drivers options and settings.
Copyright (C) STMicroelectronics. All rights reserved.

SARADC Settings
SARADC Driver Settings. Enable Drivers and create configurations

SARADC 12bit Supervisor
3 SARADC B ☑

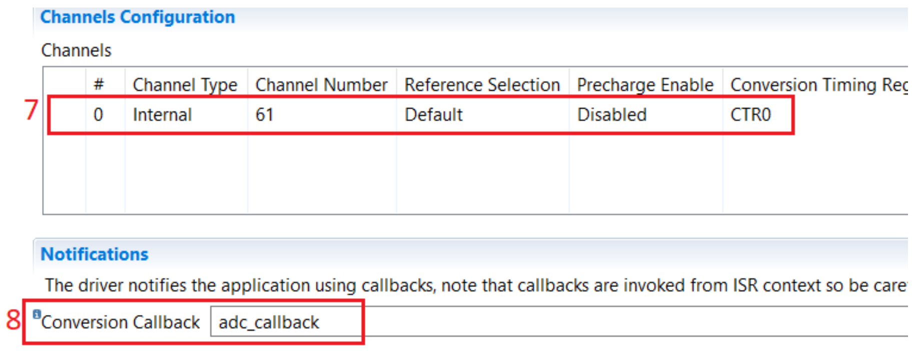
SARADC 12bit modules
SARADC 0 ☐ SARADC 1 ☐ 4 SARADC 3 ☑

SARADC 10bit modules
SARADC STANDBY ☐

SARADC Configurations
Conversion definition for SARADC peripheral. It is a structure that describes a conversion process, the structures and the required definitions are automatically generated.
Configs

#	Symbolic Name	Conversion Mode
0	saradcconf	Scan

Step 4. Double-click the “saradconf” config channel and insert the Callback name to be invoked whenever an ADC value is ready to be acquired.



#	Channel Type	Channel Number	Reference Selection	Precharge Enable	Conversion Timing Req
0	Internal	61	Default	Disabled	CTR0

The driver notifies the application using callbacks, note that callbacks are invoked from ISR context so be care

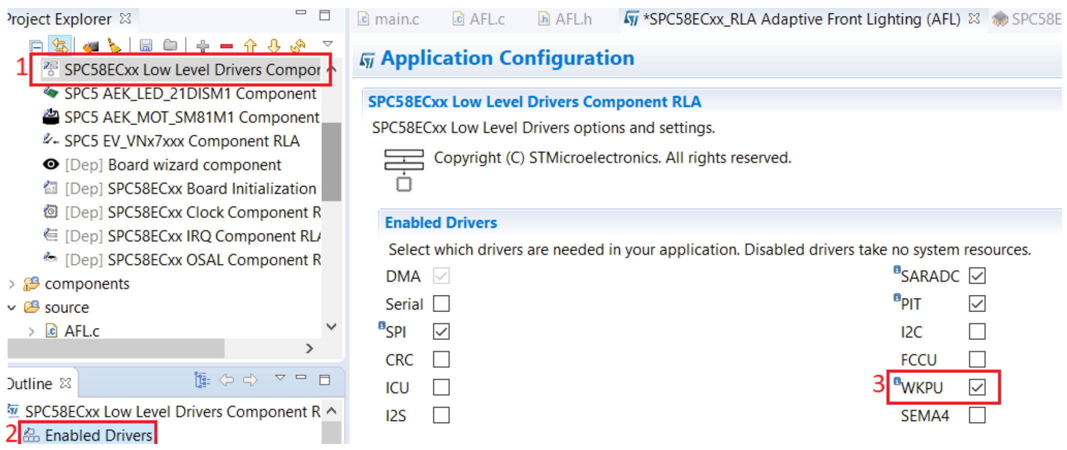
Conversion Callback: adc_callback

Figure 64. **ADC callback nomination**

Step 5. Use the Board view to connect the potentiometer to pin 5 of the ST. WHEEL connector on the AEK-CON-AFLVIP2 connector board.

Step 6. In the PinMap, select and associate a pin to the SW1 button on the MCU board.
SW1 will be set up to switch between Manual and Automatic Modes.

Step 7. Enable the WKPU driver.


Project Explorer
1
SPC58ECxx Low Level Drivers Compor
SPC5 AEK_LED_21DISM1 Component
SPC5 AEK_MOT_SM81M1 Component
SPC5 EV_VNx7xxx Component RLA
[Dep] Board wizard component
[Dep] SPC58ECxx Board Initialization
[Dep] SPC58ECxx Clock Component R
[Dep] SPC58ECxx IRQ Component RL
[Dep] SPC58ECxx OSAL Component R
components
source
AFL.c
Outline
SPC58ECxx Low Level Drivers Component R
2
Enabled Drivers
main.c
AFL.c
AFL.h
*SPC58ECxx_RLA Adaptive Front Lighting (AFL)
SPC58E
Application Configuration
SPC58ECxx Low Level Drivers Component RLA
Enabled Drivers
3

SPC58ECxx Low Level Drivers options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

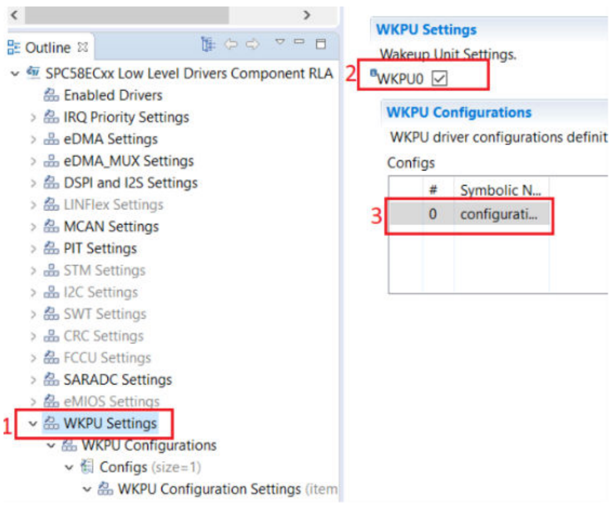
Select which drivers are needed in your application. Disabled drivers take no system resources.

DMA	✓	SARADC	✓
Serial	☐	PIT	✓
SPI	✓	I2C	☐
CRC	☐	FCCU	☐
ICU	☐	WKPU	✓
I2S	☐	SEMA4	☐

Figure 65. **Wakeup Unit (WKPU) driver enable**

Step 8. Associate and configure the Wakeup signal to the SW1 button.

Figure 66. **Wakeup Unit association with SW1**



Outline
- SPC58ECxx Low Level Drivers Component RLA
  - Enabled Drivers
  - IRQ Priority Settings
  - eDMA Settings
  - eDMA_MUX Settings
  - DSPI and I2S Settings
  - LINFlex Settings
  - MCAN Settings
  - PIT Settings
  - STM Settings
  - I2C Settings
  - SWT Settings
  - CRC Settings
  - FCCU Settings
  - SARADC Settings
  - eMIOS Settings
  - 1 WKPU Settings
    - WKPU Configurations
      - Configs (size=1)
        - WKPU Configuration Settings (item

WKPU Settings
Wakeup Unit Settings.
2 WKPU0 ☑

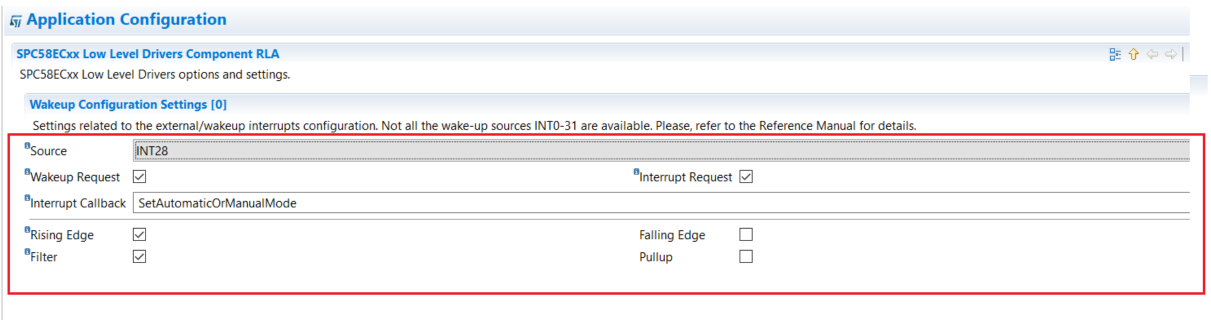
WKPU Configurations
WKPU driver configurations definit
Configs

#	Symbolic N...
3 0	configurati...

Step 9. Insert ‘SetAutomaticOrManualMode’ in the Interrupt Callback field.

This function changes the state of a global variable used to switch between Automatic and Manual Mode code execution.

Figure 67. **Callback nomination for SW1 interrupt**



Application Configuration

SPC58ECxx Low Level Drivers Component RLA
SPC58ECxx Low Level Drivers options and settings.

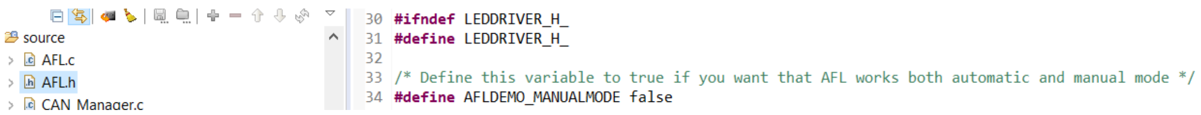
Wakeup Configuration Settings [0]
Settings related to the external/wakeup interrupts configuration. Not all the wake-up sources INT0-31 are available. Please, refer to the Reference Manual for details.

Source	INT28		
Wakeup Request	☑	Interrupt Request	☑
Interrupt Callback	SetAutomaticOrManualMode		
Rising Edge	☑	Falling Edge	☐
Filter	☑	Pullup	☐

Step 10. Set the variable `#define AFLDEMO_MANUALMODE` from false to true in the file AFL.h file.

The code for Manual Mode is already nested in the AFL-Demo project.

Figure 68. **Define to enable Manual Mode**



source
- AFL.c
- AFL.h
- CAN Manager.c

```
#ifndef LEDDRIVER_H_
#define LEDDRIVER_H_

/* Define this variable to true if you want that AFL works both automatic and manual mode */
#define AFLDEMO_MANUALMODE false
```

Appendix B CAN protocol in automotive applications

The car area network (CAN) protocol is widely used in Automotive Body and Convenience applications where the main body control module (BCM) connects with several electrical control units (ECUs).

The CAN protocol is implemented in the AFL-Demo software to communicate with an instrument panel control (IPC) so that actions performed by the headlight, such as blinking indicator light, can be reflected in the dashboard.

In-vehicle CAN messages belong to the following categories:

1. Messages triggered by specific events.
2. Message sent cyclically for safety reasons.

The ECU normally manages cyclic messages with a scheduler to ensure appropriate timing of transmission. The AFL-Demo source code only implements basic communication with the IPC; for example, only one type of message is sent to actuate the blinking effect on a car dashboard, so we have associated a programmable interrupt timer (PIT) to trigger the CAN message send event instead of a scheduler, which would normally be the case.

Revision history

Table 2. Document revision history

Date	Version	Changes
09-Oct-2019	1	Initial release.
03-Jun-2024	2	Updated Section Description, , Section 1: Overview of Adaptive front lighting demonstration kits, Section 2: The AutoDevKit AFL development platform, Section 3: AFL source code overview: Automatic mode, Section 4: Hardware, Section 5: AFL-Demo software package, Section 6: How to program the SPC58EC80 microcontroller, Section 7: AutoDevKit Studio overview and Appendix A.1: How to edit the AFL-Demo code to accept potentiometer inputs: Manual Mode.

Contents

List of figures

List of tables