Hello, and welcome to this presentation of the STM32 debug interface. It covers the debug capabilities offered by STM32F7 devices.
The debug interface of STM32 products provides access to MCU internal resources. This interface is used to program the MCU and debug applications using basic debug features. In addition to the basic debugging features, applications benefit from the trace capability used to quickly identify possible malfunctioning parts of the application and to create coverage and profiling reports used for application tests, optimizations and certifications.
The STM32F7 offers several basic debugging features which are supported by all hardware and software debug tool sets. Debugging hardware can interface with the STM32F7 through the 5-wire standard JTAG interface or the 2-wire serial wire debug port. It provides debugger access to a view of memory that is consistent with that observed by load/store instructions acting on the processor. It is able to access all internal data buses, or directly read or write to all registers and memories, including programming the Flash memory.

The user can control the execution of the application through breakpoints and code stepping. When the program reaches a breakpoint, internal peripherals like timers can be frozen in their current state or can be left running.
The STM32F7 also supports more advanced debugging features. These features require additional support by the debugging toolset – both hardware and software – as well as support on the MCU side for features requiring additional pins like the embedded trace macrocell (ETM) or the trace point unit interface (TPUI).
The Flash patch and breakpoint unit provides hardware breakpoints for debugging the application or possibly correcting software bugs in the code memory space. The full implementation option offers two literal comparators and six instruction comparators.
The embedded data watchpoint trigger provides four comparators configurable as a hardware watchpoint, ETM trigger, PC sampler or data address sampler. It provides the necessary information for data tracing and system profiling analysis, for which it embeds counters for counting the number of clock cycles, load and store operations, sleep cycles, clocks per instruction and also information about interrupt overhead. It can also generate reports about the application profile.
The instrumentation trace macrocell (ITM) supports printf style debugging information for diagnostics. Packets can be invoked by software – a direct write to the ITM or by hardware – triggered from the data watchpoint trigger (DWT).
It also provides a timestamp from the 21-bit counter.
The embedded trace macrocell (ETM) provides information about the execution flow of the application by tracing data through the DWT or ITM and tracing instructions through the ETM. This information is then sent to the debugger host for processing. This information allows the debugger to completely reconstruct the execution flow. It is very useful to quickly identify bugs and also generate code coverage and profiling reports, which are used for test purposes and certifications.
The trace port interface unit formats information from the on-chip trace units – ITM and ETM – and sends them to the debugger host. It supports Asynchronous mode with one pin used for communication in Single-wire mode, or Synchronous mode with up to 5 pins working in both JTAG and Single-wire modes. Synchronous mode provides better data throughput. After a device reset, these pins are not assigned and must be configured by the debugger host.
The STM32F7 supports flexible debug pin assignments. After a reset, all five debug pins are assigned to the debug interface. The application may reconfigure them back for GPIO operation and release these pins for application use.

Please note that these debug pins have internal pull-up or pull-down resistors enabled after a reset to prevent any uncontrolled IO levels on these pins.
The debug interface operates in all low-power modes. For Sleep, Stop and Standby modes, related bits must be configured in the DBGMCU_CR register in order to prevent the clock and regulators from stopping when entering a low-power mode.

<table>
<thead>
<tr>
<th>Mode</th>
<th>I/O Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Active.</td>
</tr>
<tr>
<td>Sleep</td>
<td>Active. DBG_SLEEP must be previously set by debugger.</td>
</tr>
<tr>
<td>Stop</td>
<td>Active. DBG_STOP must be previously set by debugger.</td>
</tr>
<tr>
<td>Standby</td>
<td>Active. DBG_STANDBY must be previously set by debugger.</td>
</tr>
</tbody>
</table>
References

• For more details, please refer to the following sources:
  • Cortex®-M7 r1p0 technical reference manual
    • Search for “Cortex®-M7 technical reference manual” at http://infocenter.arm.com
  • ARM® debug interface

For more technical details, visit the infocenter.arm.com website.