Smart Code Debugging Techniques
• Introduction
• Simulator and C-SPY macros
• Breakpoints
• SWD/SWO and features
• Debugging a crash
• Debugging external images and Bootloader
• RTOS Kernel Awareness Plugin
• Demo
• Summary
• Q&A
• WIKIPEDIA: ”**Debugging** is a methodical process of finding and reducing the number of bugs …
Simulator and C-SPY Macros
The C-SPY Simulator simulates the functions of the target processor entirely in software.

The C-SPY Simulator supports:

- Instruction-level simulation
- Memory configuration and validation
- Interrupt simulation
- Peripheral simulation (using the C-SPY macros).
C-SPY macros enable you to build complex debug functions suited to your needs. They can use functions such as:

- File operations
- Memory read/write
- Breakpoint setting/clearing
- Interrupt enable/disable
- JTAG commands
- SFR read/write

Examples

- Prepare the device for debugging
- Simulate HW/SW not yet ready
- Store the content of a memory range to file
- Use some really complex breakpoints
Breakpoints
CORTEX-M3/M4 DEBUG ARCHITECTURE

**DWT**
- 4 watchpoints
- PC sampler
- ETM trigger
- interrupt trace
- CPU statistics

**FPB**
- 6 breakpoints

**ITM**
- software trace
- 32 channels
- time stamping

**ETM**
- instruction trace

**DAP**
- live core access
- JTAG
- SWD/SWO
- trace port
• On a Cortex M device the debugger (C-SPY) instructs the "Flash Patch and Breakpoint Unit" to halt the core on a specific instruction
  • This unit only works in a specified address range.
  • Limit in number of breakpoints

<table>
<thead>
<tr>
<th></th>
<th>ARM7</th>
<th>ARM9</th>
<th>CM0</th>
<th>CM3</th>
<th>CM4</th>
<th>CR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakpoints</td>
<td>2</td>
<td>2</td>
<td>≤ 4</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Watchpoints (DWT)</td>
<td>≤ 2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Software breakpoints
  • Here the debugger replaces the instruction where we want the breakpoint with a bkpt instruction. After stopping the instruction is inserted again.
    • Require the ability to program the memory where the code is placed.

• Complex conditional code and data breakpoints
• Log and data log breakpoints
SWD and SWO
ARM CORE COMPARISON

- All cores have different interface and debug capabilities

<table>
<thead>
<tr>
<th></th>
<th>ARM7</th>
<th>ARM9</th>
<th>CM0</th>
<th>CM3</th>
<th>CM4</th>
<th>CR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SWD</td>
<td></td>
<td></td>
<td>(X)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SWO</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ITM</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ETM</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>ETB</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
SWD, a modern JTAG

- JTAG was not intended for debugging, SWD is.
- Fewer wires, high speed

Trace, very handy for finding bugs.

- ETM
- SWO Trace
- ETB

SWO

- Lots and lots and lots of info to be found here
SWO

• One wire data transfer, asynchronous bit stream
• USART
  • Wrong speed setting makes data impossible to interpret, as a RS232 UART
  • Speed derived from mcu-clock -> sensitive to speed changes of cpu
  • 1Mb/s
• Manchester encoded
  • The clock can be derived from the signal -> No speed setting necessary
    • Changing speed of mcu is OK
  • 100Mb/s

• printf
  • High speed terminal IO
  • stdio will use SWO to send data to Terminal IO in CSPY
    • Use during development while your special hw is being constructed
    • Old fashioned printf-debugging
    • No use of breakpoints, print diagnostic messages
  • Simple to use, just include stdio.h
Another similar feature is the ITM events. By including these macros in the code data is sent to, displayed in and logged by C-SPY.

- Simple way of sending small packets of data
- 4 channels
- Different sizes
- Tracking data
  - ITM_EVENT8(channel, value)
  - ITM_EVENT16(channel, value)
  - ITM_EVENT32(channel, value)
Interrupt Logging
• Everytime an interrupt enters or exits a timestamp is sent to C-SPY
• Great for verifying timing of interrupts
• Verify how interrupts are interrupted by others with higher priority.
Code Coverage and Function Profiling
• Code coverage use trace data to analyze what parts of the included functions are really executed in a run.
• Can be a requirement that all code is really tested, i.e. executed in a test session
• Test cases can be added to verify code, or untested code can be removed
FUNCTION PROFILING

- Function profiling using sampled trace data
- Function profiling using ETM trace data
  - More accurate, still based on how many hits you get in each function
- Function profiling using function calls as derived from ETM data
  - Here the number of times a function is called is measured, not number of times multiplied with the length of the function
Debugging a Crashed Application
DEBUGGING A CRASHED APPLICATION

On Cortex M devices a lot of bugs can cause the execution to be shifted into predefined exception handlers

- UsageFault_Handler()
- BusFault_Handler()
- MemMang_Handler()
- HardFault_Handler()

Of these, the HardFault handler is probably the most common, as it is used by unrecoverable system failures
External Images and Debugging the Bootloader
• In C-SPY there is an ability to connect to a target without stopping the running program.

• Useful if you have a system that after running a while acts funny. Connect to it when it is behaving strange
  • Run the device without debugger -> Debugger will not affect normal execution.
  • No trace of course, but possible to find clues of a problem that is not evident at startup
EXTERNAL DEBUG INFORMATION

• Add extern debug information to your session
  • If the device already is partially programmed
  • Bootloader – Application
  • Large binary that you don’t want to program again, but want the debug information

• Debug only session, using a ELF file as input
  • The device is programmed, you only want to debug the device.
  • Program only
  • Simpler debug, no changes to code
  • Project -> Create new project -> Externally built executable

• Usefull when debugging the bootloader
RTOS Kernel Awareness Plugin
The C-SPY Debugger of IAR Embedded Workbench has a native RTOS awareness plugin. This makes it possible to monitor:

- Threads
- Message Queues
- Semaphores
- Mutexes
- Pools
- Timers
- Even Flag Groups
- Thread Performance Metrics
- Timer Performance Metrics

It will show you how your Application and tasks are behaving.
Demo
Summary
• IAR comprehensive debugging environments feature a wide range of different capabilities to perform smart debugging of your code

• In this presentation and demo we had an overview of available debug features, e.g. complex breakpoints, macros, stack checking, event logging, live data, and profiling

• We also demonstrated on how these available features can be used in a smart way for comprehensive debugging of your code
Want to learn more?

- Book an IAR Academy course
- Get a demo of our latest news

THANK YOU FOR YOUR ATTENTION!