Chapter 2

The Cortex-M Series: Hardware and Software

Introduction

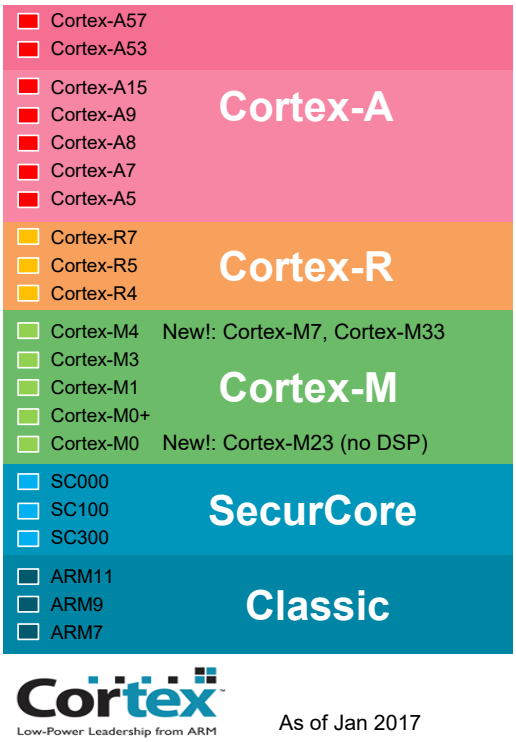
In this chapter the real-time DSP platform of primary focus for the course, the Cortex M4, will be introduced and explained. in terms of hardware, software, and development environments. Beginning topics include:

- ARM Architectures and Processors
  - What is ARM Architecture
  - ARM Processor Families
  - ARM Cortex-M Series
  - Cortex-M4 Processor
  - ARM Processor vs. ARM Architectures
- ARM Cortex-M4 Processor
  - Cortex-M4 Processor Overview
  - Cortex-M4 Block Diagram
  - Cortex-M4 Registers

What is ARM Architecture

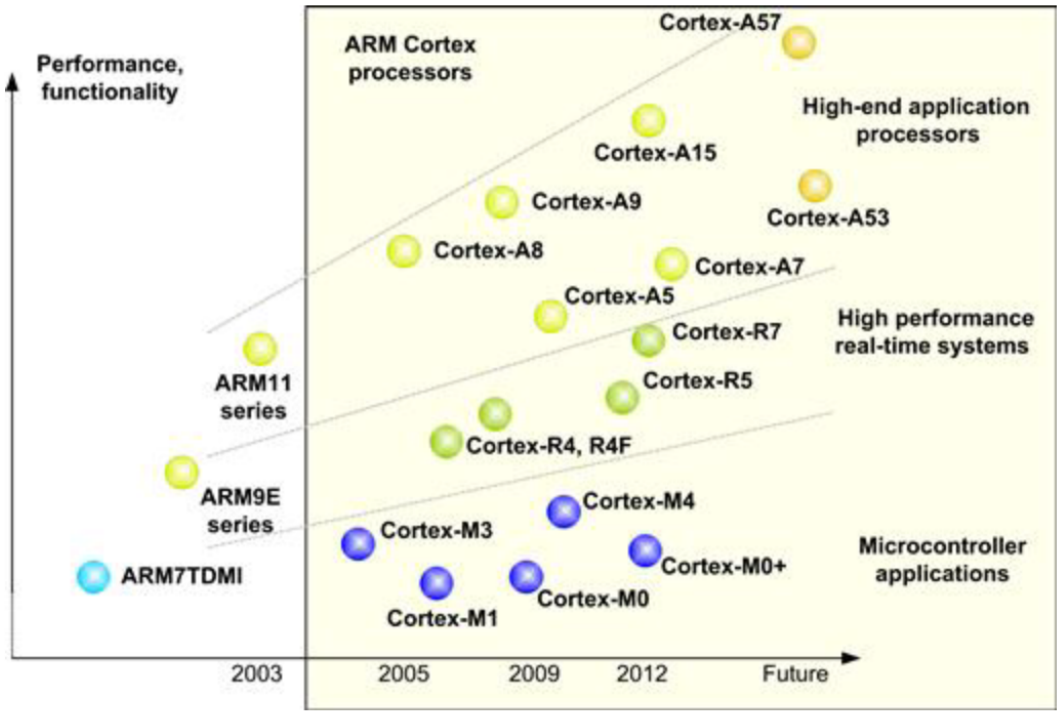
- ARM architecture is a family of RISC-based processor architectures
  - Well-known for its power efficiency;
  - Hence widely used in mobile devices, such as smart phones and tablets
  - Designed and licensed to a wide eco-system by ARM
- ARM Holdings
  - The company designs ARM-based processors;
  - Does not manufacture, but licenses designs to semiconductor partners who add their own Intellectual Property (IP) on top of ARM's IP, fabricate and sell to customers;
  - Also offer other IP apart from processors, such as physical IPs, interconnect IPs, graphics cores, and development tools

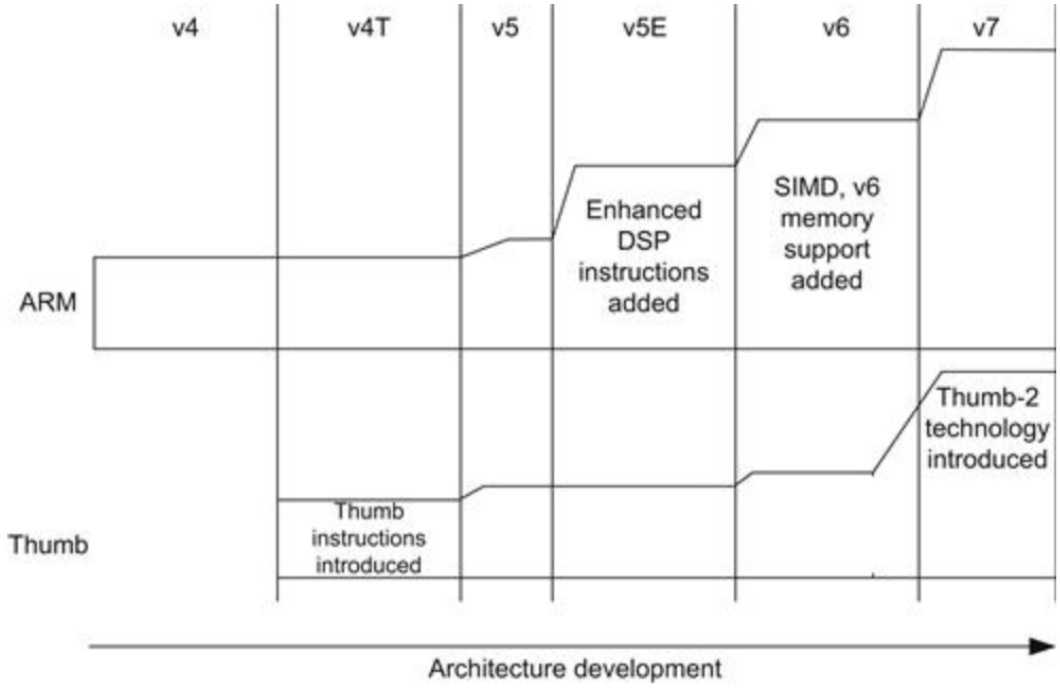
ARM Processor Families



- Cortex-A series (Application)
  - High performance processors capable of full Operating System (OS) support;
  - Applications include smartphones, digital TV, smart books, home gateways etc.
- Cortex-R series (Real-time)
  - High performance for real-time applications;
  - High reliability
  - Applications include automotive braking system, powertrains etc.
- Cortex-M series (Microcontroller)
  - Cost-sensitive solutions for deterministic microcontroller applications;
  - Applications include microcontrollers, mixed signal devices, smart sensors, automotive body electronics and airbags; more recently IoT
- SecurCore series
  - High security applications.
- Previous classic processors: Include ARM7, ARM9, ARM11 families

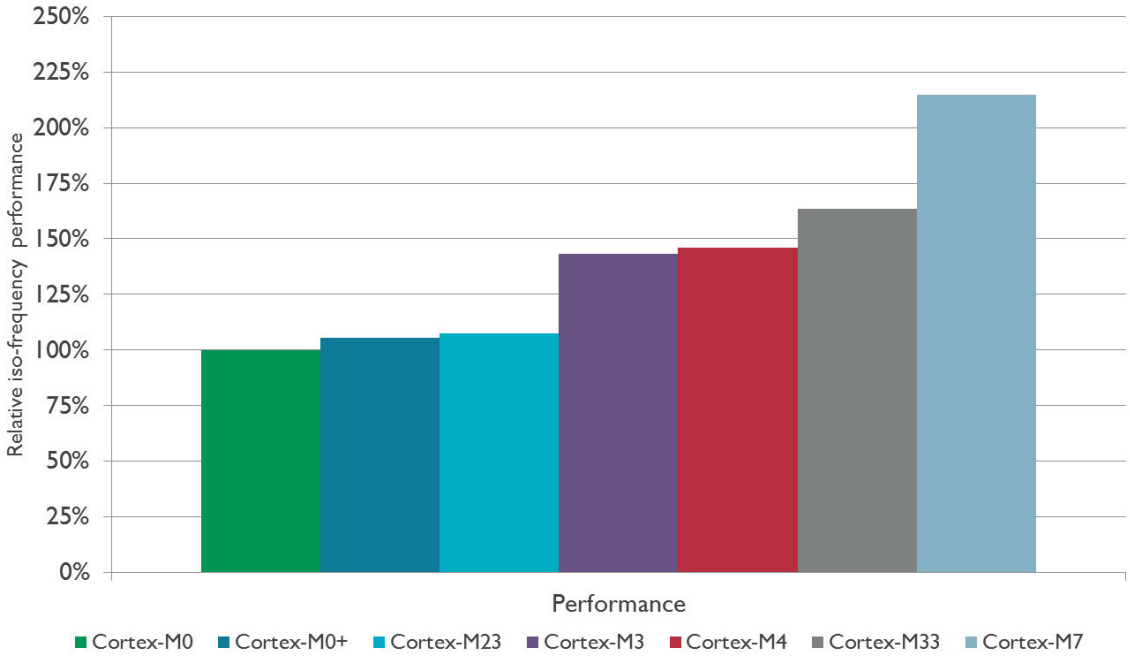
ARM Families and Architecture Over Time[1]





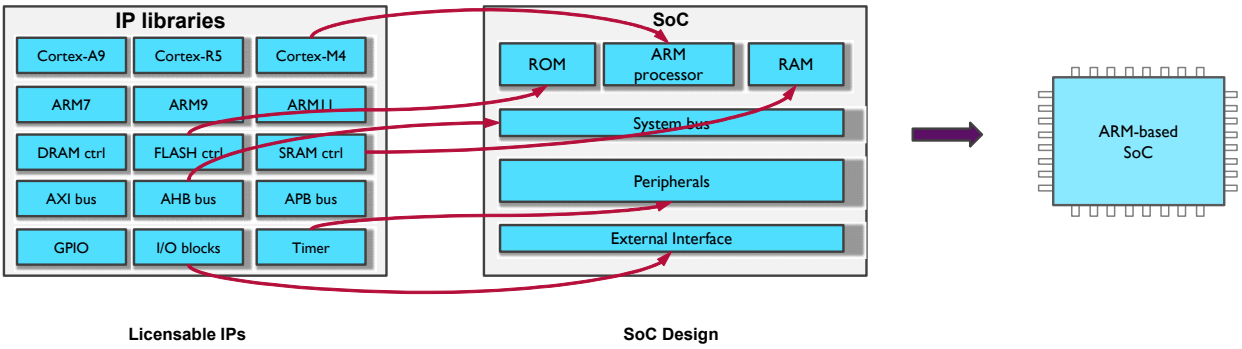
1. J. Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, 3rd edition, Newnes 2014.

Relative Performance[1]



Design an ARM-based SoC

- Select a set of IP cores from ARM and/or other third-party IP vendors
- Integrate IP cores into a single chip design
- Give design to semiconductor foundries for chip fabrication



1. https://www.arm.com/products/processors/cortex-m

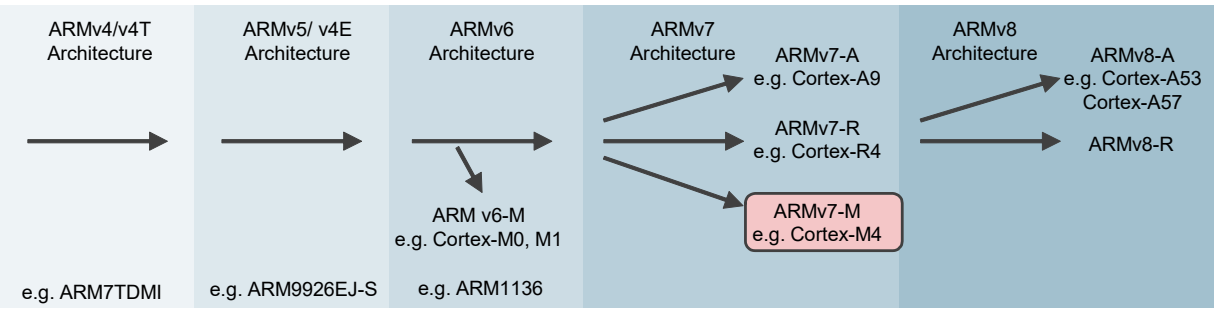
ARM Cortex-M Series

- Cortex-M series: Cortex-M0, M0+, M1, M3, M4, M7, M33.
- Energy-efficiency
  - Lower energy cost, longer battery life
- Smaller code
  - Lower silicon costs
- Ease of use
  - Faster software development and reuse
- Embedded applications
  - Smart metering, human interface devices, automotive and industrial control systems, white goods, consumer products and medical instrumentation, IoT

ARM Processors vs. ARM Architectures

- ARM architecture
  - Describes the details of instruction set, programmer's model, exception model, and memory map
  - Documented in the Architecture Reference Manual
- ARM processor
  - Developed using one of the ARM architectures
  - More implementation details, such as timing information

– Documented in processor's Technical Reference Manual



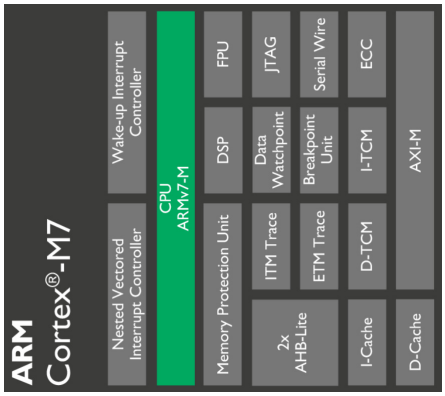
As of Dec 2013

Companies Making ARM Chips

- Apple, AppliedMicro, Microchip (Atmel), Broadcom, Cypress Semiconductor, Nvidia, NXP, Samsung Electronics, ST Microelectronics, and Texas Instruments (http://en.wikipedia.org/wiki/ARM_architecture)
- More — Xilinx (Zynq), ...

ARM Cortex-M Series Family

ARM Architecture	Core Architecture	Thumb®	Thumb®-2	Hardware Multiply	Hardware Divide	Saturated Math	DSP Extensions	Floating Point
ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	Software	No
ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	Software	No
ARMv6-M	Von Neumann	Most	Subset	3 or 33 cycle	No	No	Software	No
ARMv7-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Software	No
ARMv7E-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Hardware	Optional

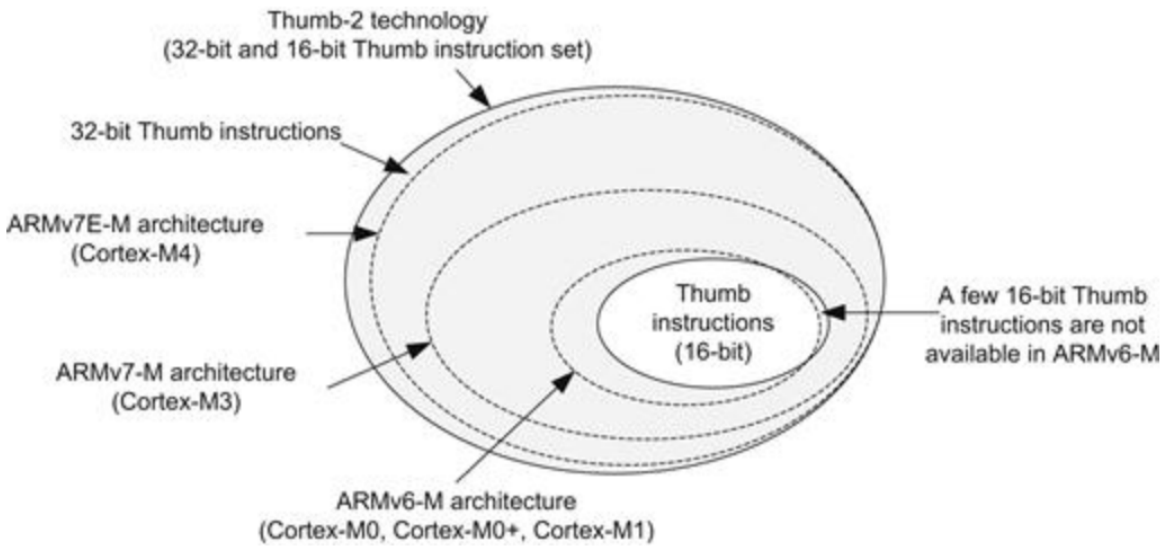


Cortex-M4 Processor Overview

- Cortex-M4 Processor
  - Introduced in 2010
  - Designed with a large variety of highly efficient signal processing features
  - Features extended single-cycle multiply accumulate instructions, optimized SIMD arithmetic, saturating arithmetic and an optional Floating Point Unit.
- High Performance Efficiency
  - 1.25 DMIPS/MHz (Dhrystone Million Instructions Per Second / MHz) at the order of µWatts / MHz
- Low Power Consumption
  - Longer battery life – especially critical in mobile products
- Enhanced Determinism
  - The critical tasks and interrupt routines can be served quickly in a known number of cycles

Cortex-M4 Processor Features

- 32-bit Reduced Instruction Set Computing (RISC) processor
- Harvard architecture
  - Separated data bus and instruction bus
- Instruction set
  - Include the entire Thumb®-1 (16-bit) and Thumb®-2 (16/32-bit) instruction sets[1]



- 3-stage + branch speculation pipeline
- Performance efficiency
  - 1.25 – 1.95 DMIPS/MHz (Dhrystone Million Instructions Per Second / MHz)

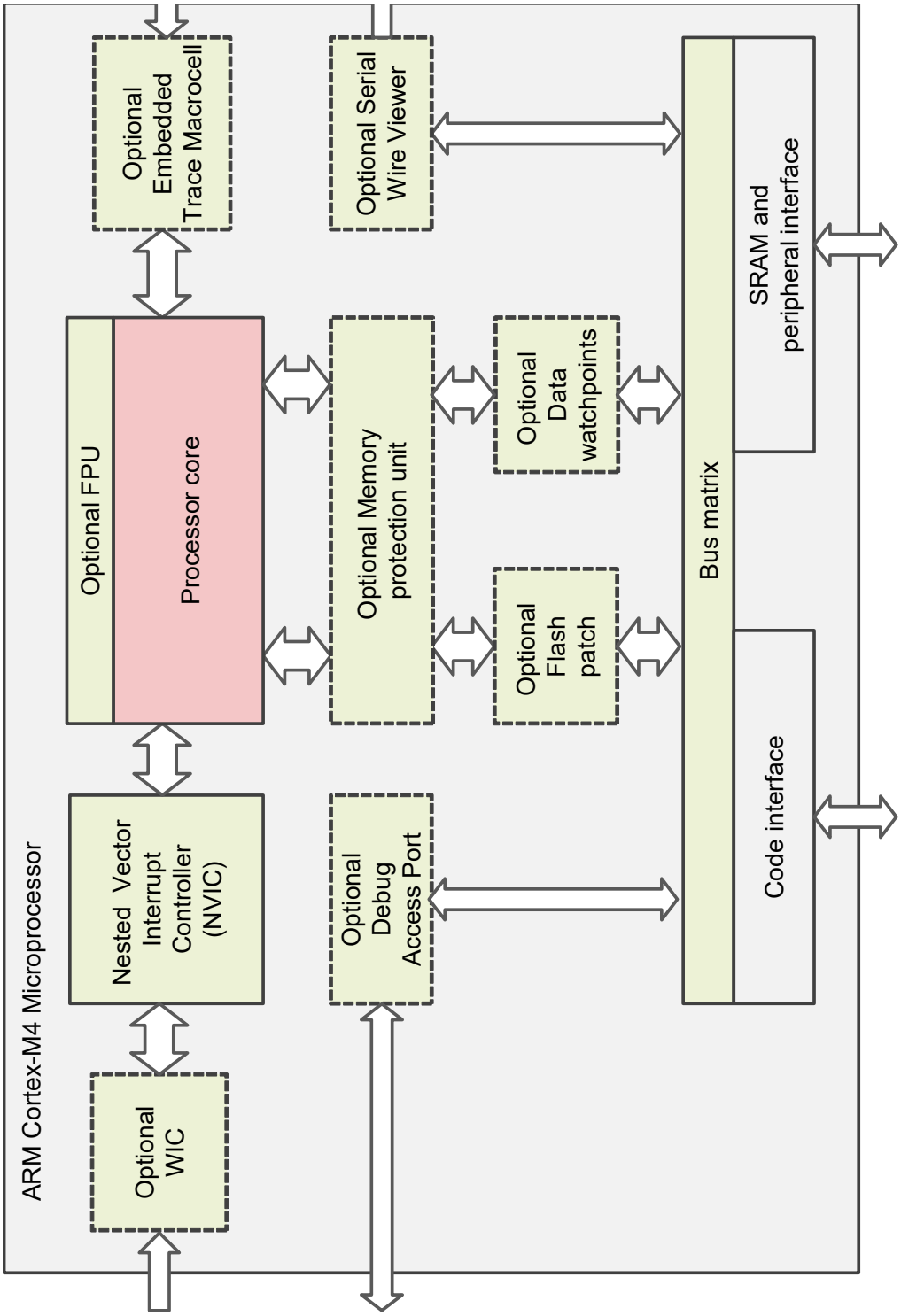
1. J. Yiu, *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors*, 3rd edition, Newnes, 2014.

- Supported Interrupts
  - Non-maskable Interrupt (NMI) + 1 to 240 physical interrupts
  - 8 to 256 interrupt priority levels
- Supports Sleep Modes
  - Up to 240 Wake-up Interrupts
  - Integrated WFI (Wait For Interrupt) and WFE (Wait For Event) Instructions and Sleep On Exit capability (to be covered in more detail later)
  - Sleep & Deep Sleep Signals
  - Optional Retention Mode with ARM Power Management Kit
- Enhanced Instructions
  - Hardware Divide (2-12 Cycles)
  - Single-Cycle 16, 32-bit MAC, Single-cycle dual 16-bit MAC
  - 8, 16-bit SIMD arithmetic
- Debug
  - Optional JTAG & Serial-Wire Debug (SWD) Ports
  - Up to 8 Breakpoints and 4 Watchpoints
- Memory Protection Unit (MPU)
  - Optional 8 region MPU with sub regions and background region

- Cortex-M4 processor is designed to meet the challenges of low dynamic power constraints while retaining light footprints
  - 180 nm ultra low power process –157 μW/MHz
  - 90 nm low power process – 33 μW/MHz
  - 40 nm G process – 8 μW/MHz

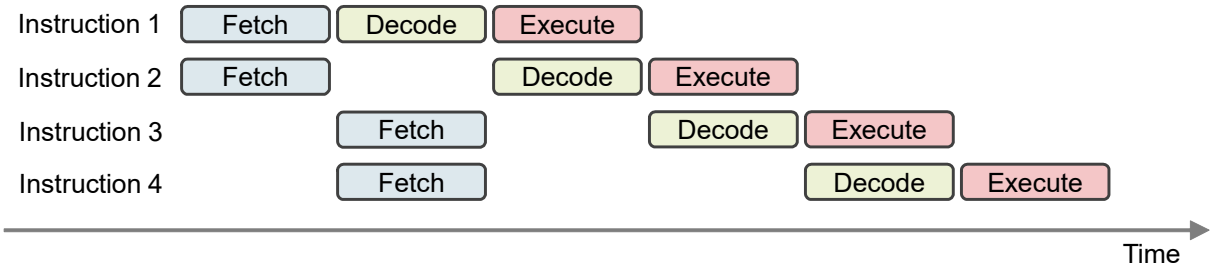
ARM Cortex-M4 Implementation Data			
Process	180ULL (7-track, typical 1.8v, 25C)	90LP (7-track, typical 1.2v, 25C)	40G 9-track, typical 0.9v, 25C)
Dynamic Power	157 μW/MHz	33 μW/MHz	8 μW/MHz
Floorplanned Area	0.56 mm^2	0.17 mm^2	0.04 mm^2

Cortex-M4 Block Diagram



Cortex-M4 Block Diagram (cont.)

- Processor core
  - Contains internal registers, the ALU, data path, and some control logic
  - Registers include sixteen 32-bit registers for both general and special usage
- Processor pipeline stages
  - Three-stage pipeline: fetch, decode, and execution
  - Some instructions may take multiple cycles to execute, in which case the pipeline will be stalled
  - The pipeline will be flushed if a branch instruction is executed
  - Up to two instructions can be fetched in one transfer (16-bit instructions)



- Nested Vectored Interrupt Controller (NVIC)
  - Up to 240 interrupt request signals and a non-maskable interrupt (NMI)

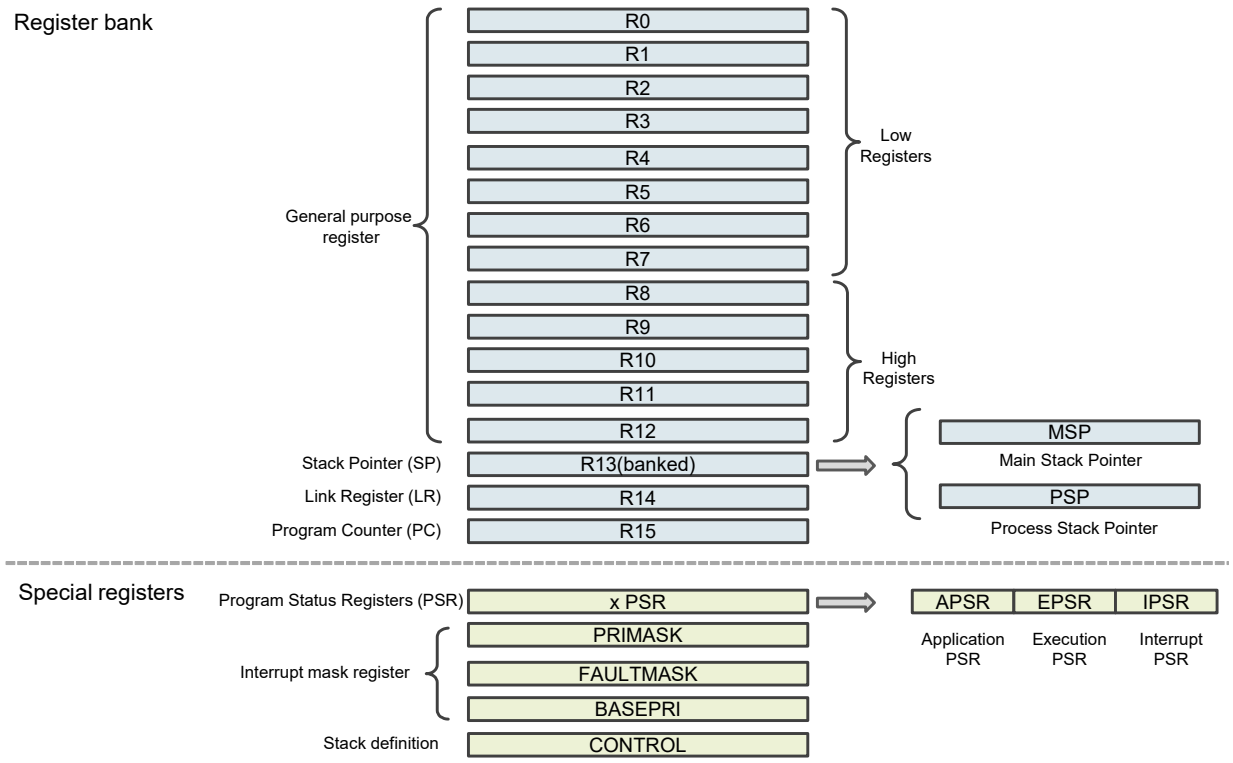
  - Automatically handles nested interrupts, such as comparing priorities between interrupt requests and the current priority level
- Wakeup Interrupt Controller (WIC)
  - For low-power applications, the microcontroller can enter sleep mode by shutting down most of the components.
  - When an interrupt request is detected, the WIC can inform the power management unit to power up the system.
- Memory Protection Unit (optional)
  - Used to protect memory content, e.g. make some memory regions read-only or preventing user applications from accessing privileged application data
- Bus interconnect
  - Allows data transfer to take place on different buses simultaneously
  - Provides data transfer management, e.g. a write buffer, bit-oriented operations (bit-band)
  - May include bus bridges (e.g. AHB-to-APB bus bridge) to connect different buses into a network using a single global memory space
  - Includes the internal bus system, the data path in the processor core, and the AHB LITE interface unit

- Debug subsystem
  - Handles debug control, program breakpoints, and data watchpoints
  - When a debug event occurs, it can put the processor core in a halted state, where developers can analyse the status of the processor at that point, such as register values and flags

Cortex-M4 Registers

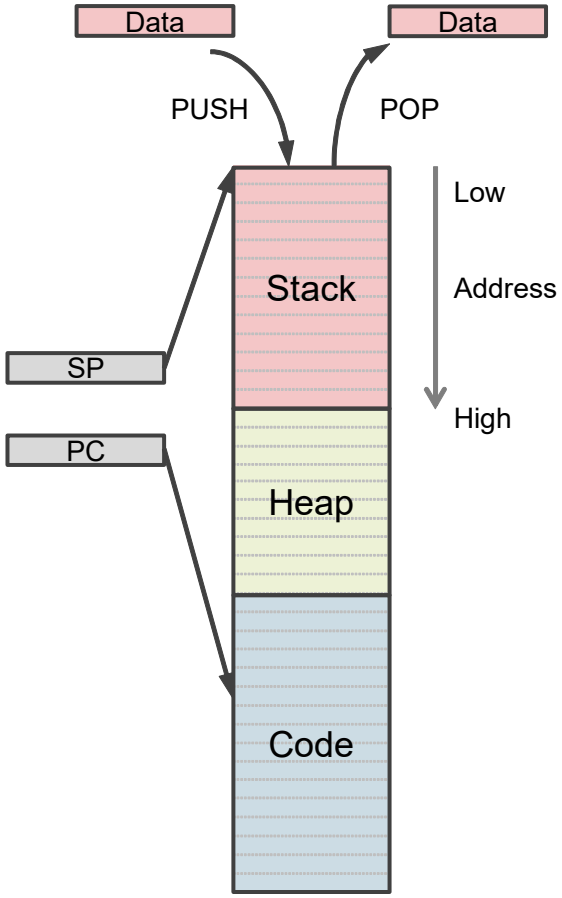
- Processor registers
  - The internal registers are used to store and process temporary data within the processor core
  - All registers are inside the processor core, hence they can be accessed quickly
  - Load-store architecture
  - To process memory data, they have to be first loaded from memory to registers, processed inside the processor core using register data only, and then written back to memory if needed
- Cortex-M4 registers
  - Register bank
  - Sixteen 32-bit registers (thirteen are used for general-purpose);
  - Special registers

Cortex-M4 Registers (cont.)

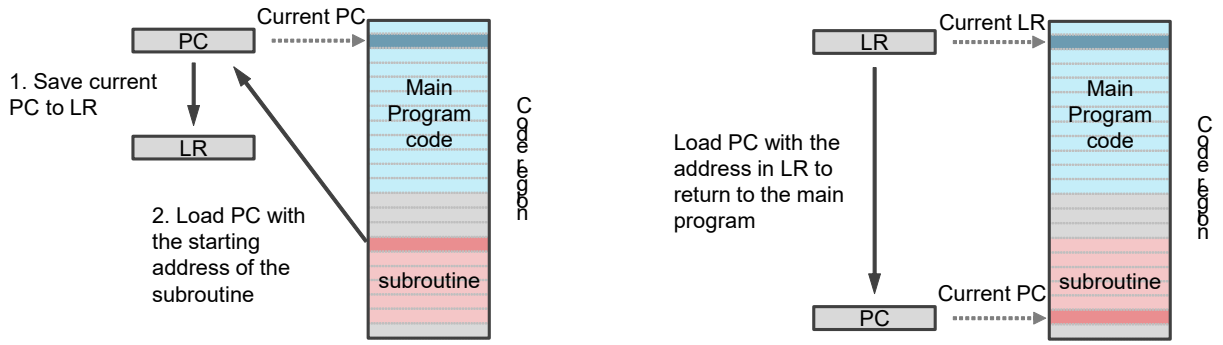


- R0 – R12: general purpose registers
  - Low registers (R0 – R7) can be accessed by any instruction
  - High registers (R8 – R12) sometimes cannot be accessed e.g. by some Thumb (16-bit) instructions
- R13: Stack Pointer (SP)
  - Records the current address of the stack
  - Used for saving the context of a program while switching between tasks

  - Cortex-M4 has two SPs: Main SP, used in applications that require privileged access e.g. OS kernel, and exception handlers, and Process SP, used in base-level application code (when not running an exception handler)
- Program Counter (PC)
  - Records the address of the current instruction code
  - Automatically incremented by 4 at each operation (for 32-bit instruction code), except branching operations
  - A branching operation, such as function calls, will change the PC to a specific address, meanwhile it saves the current PC to the Link Register (LR)

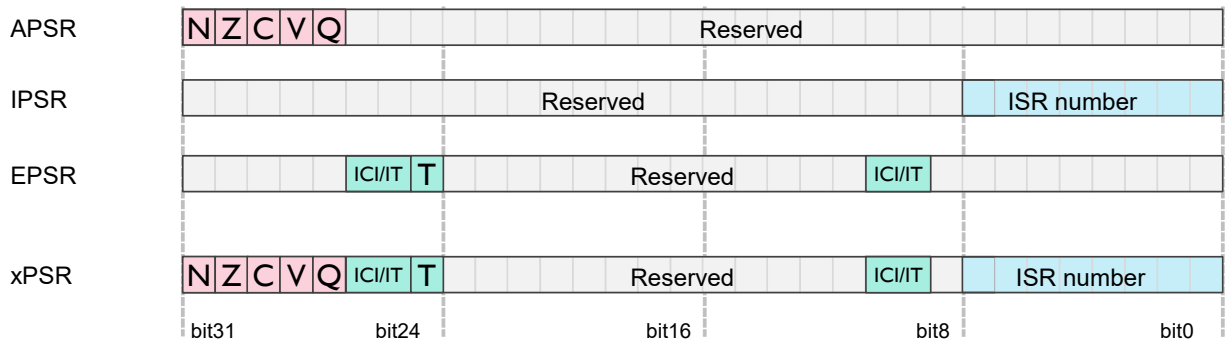


- R14: Link Register (LR)
  - The LR is used to store the return address of a subroutine or a function call
  - The program counter (PC) will load the value from LR after a function is finished



Call a subroutine **Return from a subroutine to the main program**

- xPSR, combined Program Status Register
  - Provides information about program execution and ALU flags
  - Application PSR (APSR)
  - Interrupt PSR (IPSR)
  - Execution PSR (EPSR)

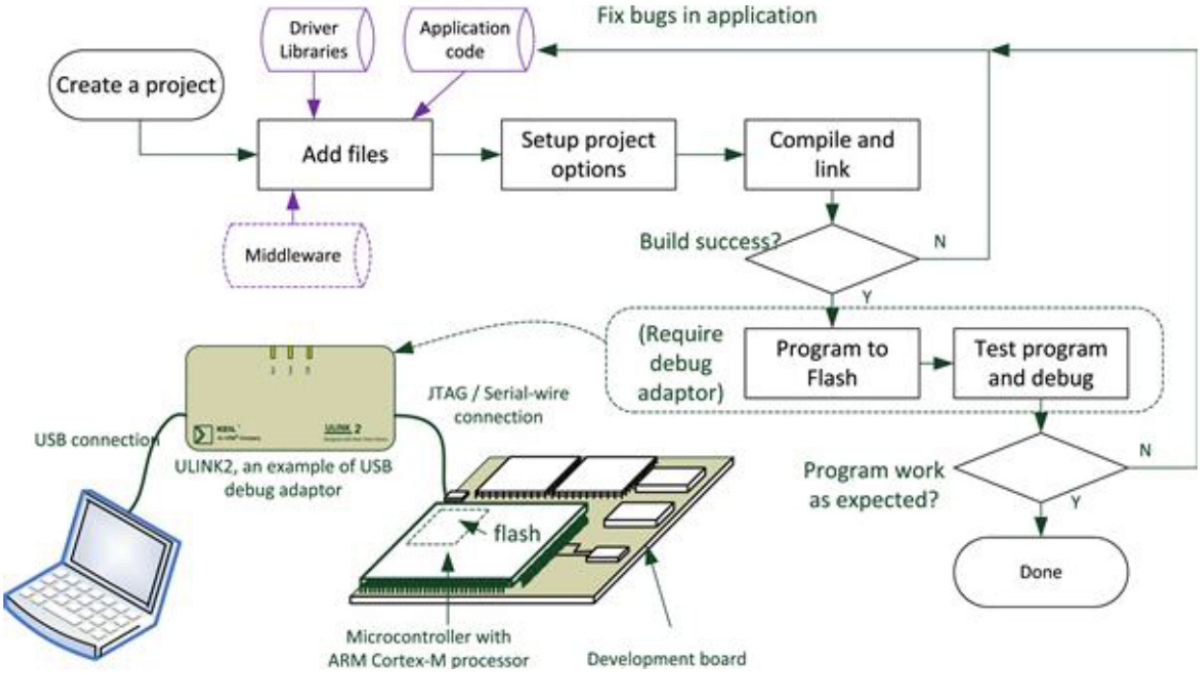


- APSR
  - N: negative flag – set to one if the result from ALU is negative
  - Z: zero flag – set to one if the result from ALU is zero
  - C: carry flag – set to one if an unsigned overflow occurs
  - V: overflow flag – set to one if a signed overflow occurs
  - Q: sticky saturation flag – set to one if saturation has occurred in saturating arithmetic instructions, or overflow has occurred in certain multiply instructions
- IPSR
  - ISR number – current executing interrupt service routine number
- EPSR
  - T: Thumb state – always one since Cortex-M4 only supports the Thumb state (more on processor states in the next module)
  - IC/IT: Interrupt-Continuable Instruction (ICI) bit, IF-THEN instruction status bit
- Interrupt mask registers
  - 1-bit PRIMASK
  - Set to one will block all the interrupts apart from nonmaskable interrupt (NMI) and the hard fault exception
  - 1-bit FAULTMASK

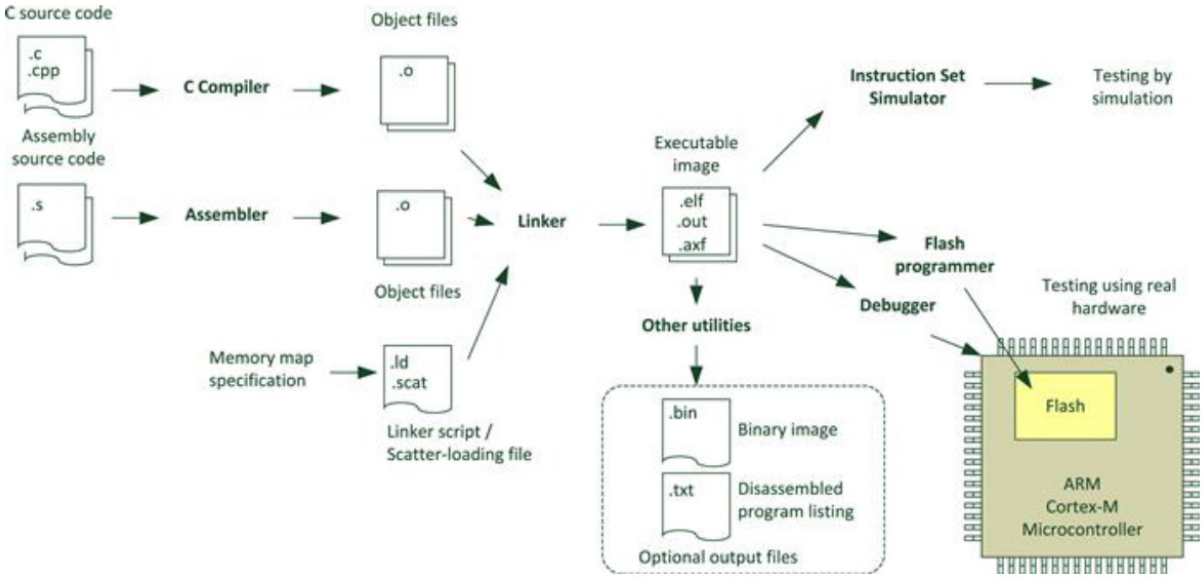
  - Set to one will block all the interrupts apart from NMI
  - 1-bit BASEPRI
  - Set to one will block all interrupts of the same or lower level (only allow for interrupts with higher priorities)
- CONTROL: special register
  - 1-bit stack definition
  - Set to one: use the process stack pointer (PSP)
  - Clear to zero: use the main stack pointer (MSP)

Software Development Overview

- The software development flow[1]:

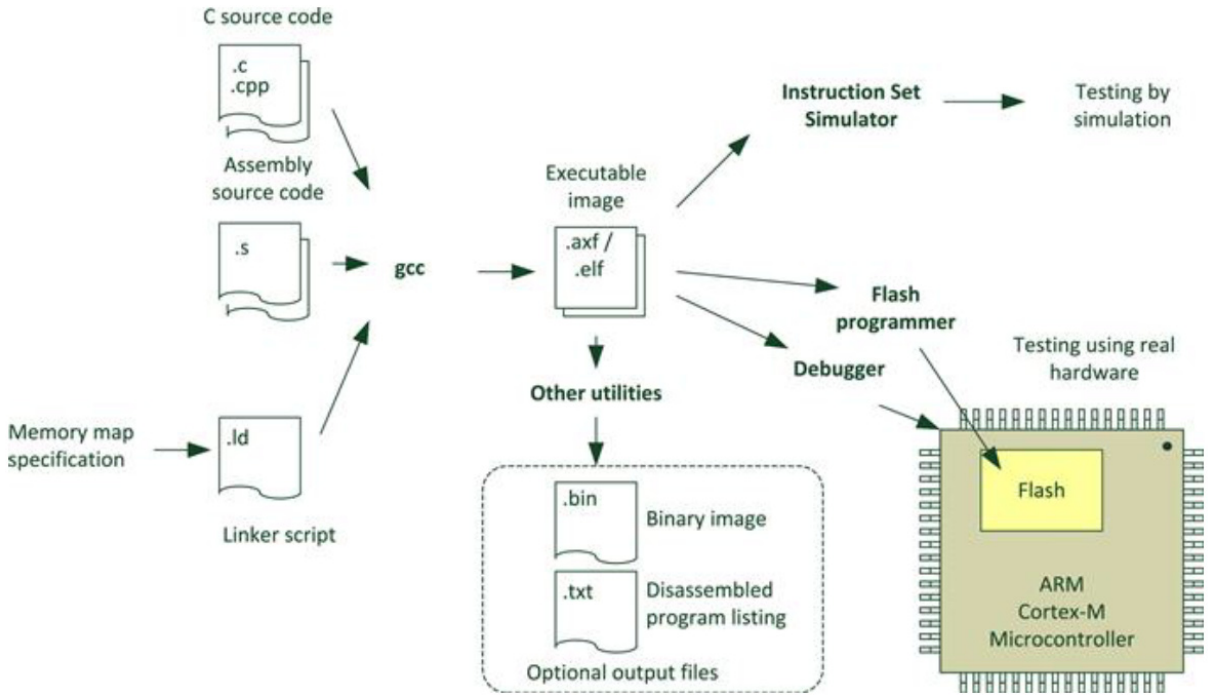


- At the file level the build process for Keil MDK is:

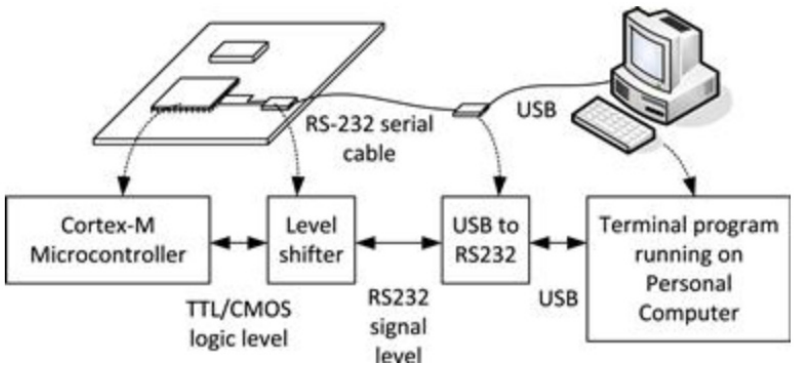


1. J. Yiu, *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors*, 3rd edition, Newnes, 2014.

- When using the GNU tool chain compilation and linking are merged[1]



- A debugging process that we will follow with the limited capability of the STM32F4 on-board emulator, is the use of a UART



  – The level shifter is not required with the USB to serial rx/tx wire device found in the lab

1. J. Yiu, *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors*, 3rd edition, Newnes, 2014.

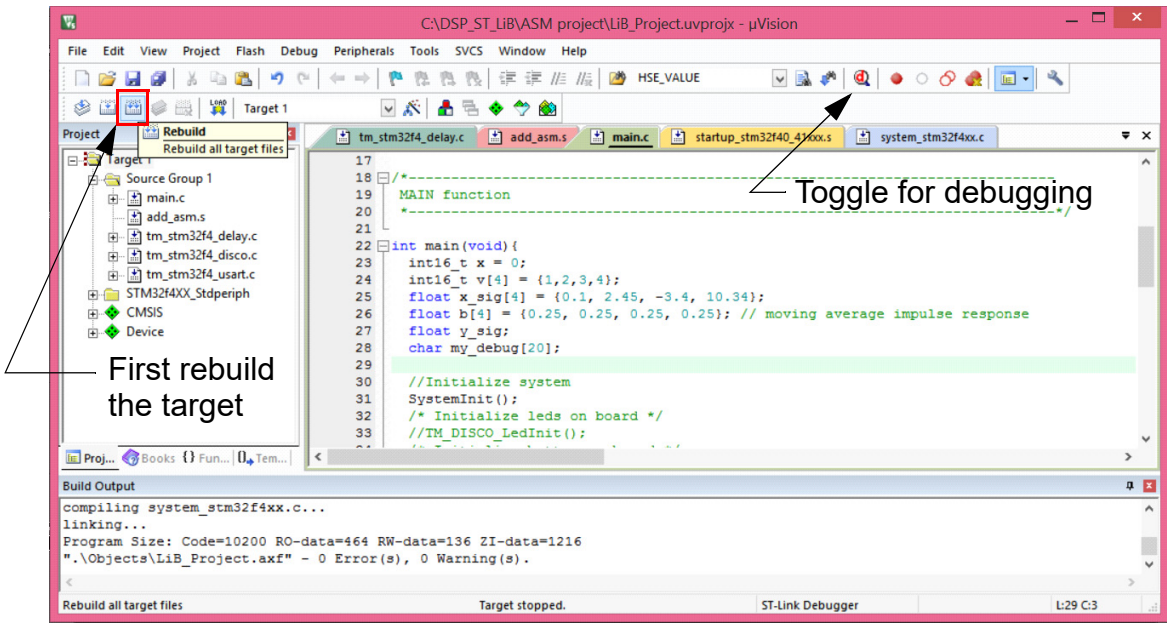
- The preferred IDE for this first offering of the Cortex-M4 version of the course, we will focus on the use of the ARM Keil integrated development environment

Keil Specific Debugging Examples

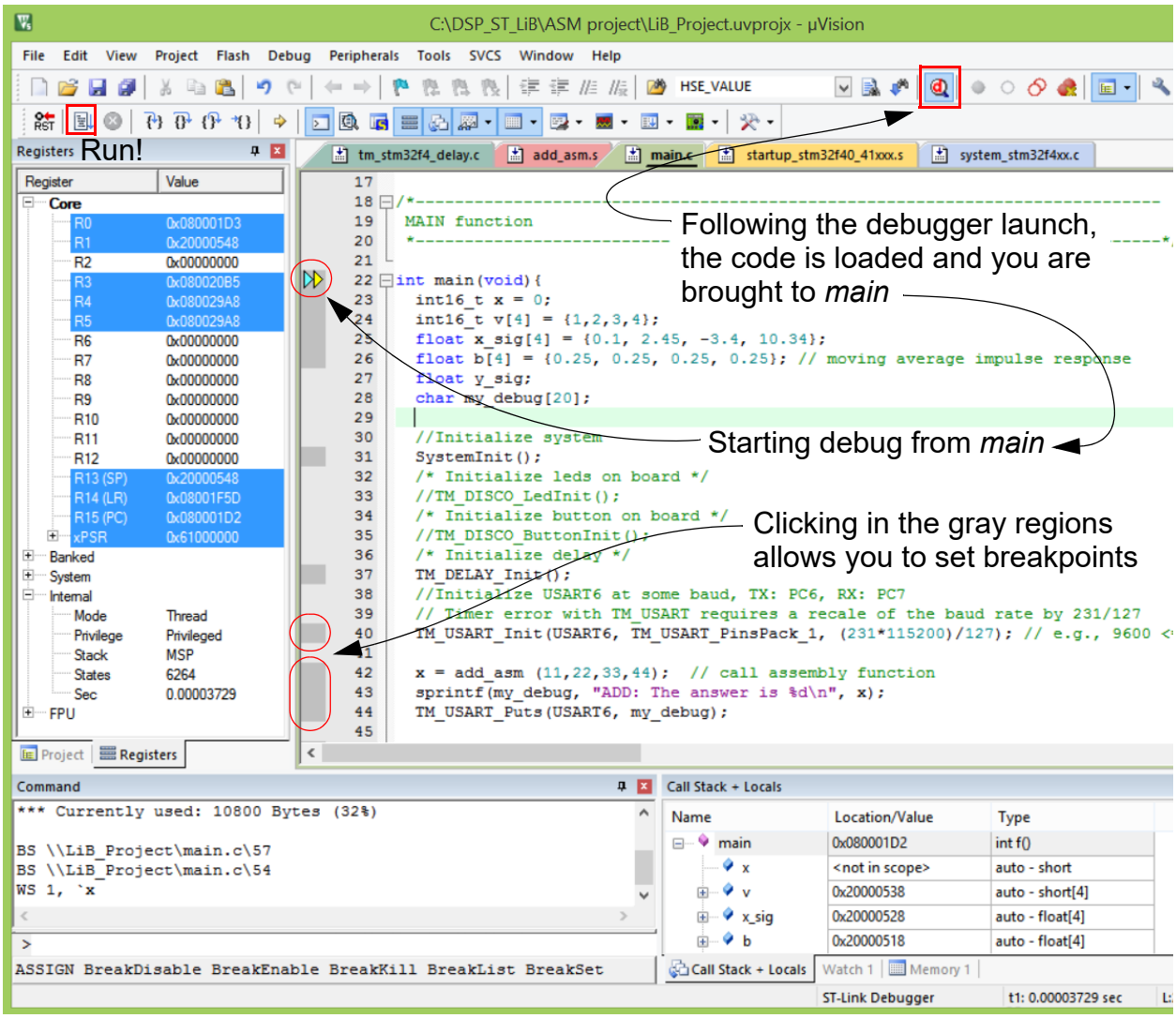
- In this section a few simple debugging examples are provided to help you get started with writing and debugging code
- The absolute starting point for working with hardware software IDE combination is Lab0 on the course Web Site

Setting Breakpoints

- First build the code in your project and launch the debugger

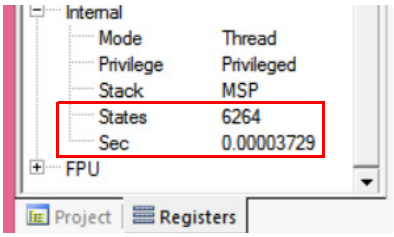


- The red d button is toggle for starting and stopping the debugger

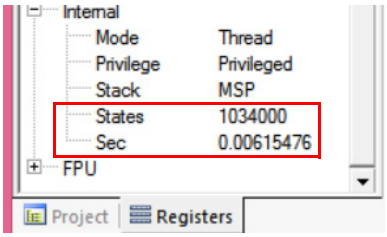


- Set break points at code lines 54 and 55
- Now click the run button and verify that the debugger stops at line 54
- The State (cycle) counter and Sec (total session run time stopwatch) will advance as you move from the main entry point down to line 54

Before



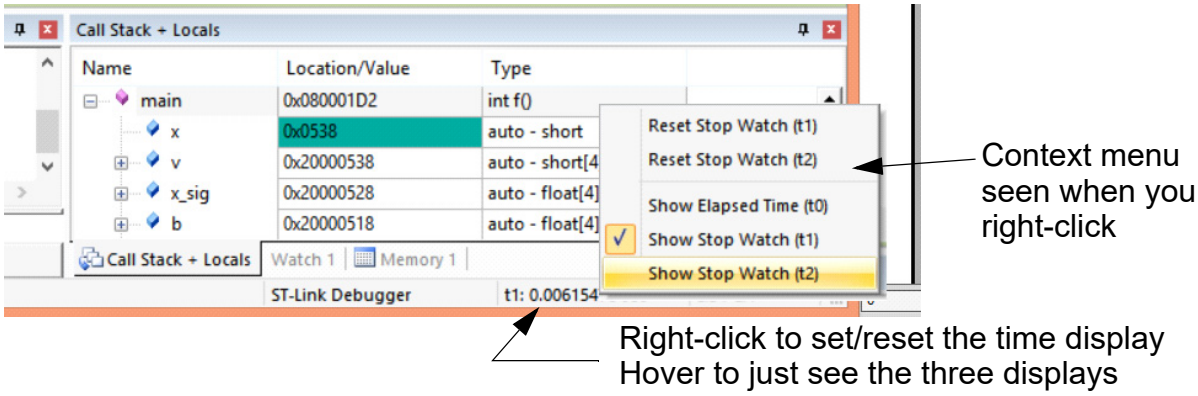
After



```
54    y_sig = dot_p(x_sig, b, 4);
55    sprintf(my_debug, "Dot Prod: The answer is %4.4f\n", y_sig);
56    TM_USART_Puts(USART6, my_debug);
57    while (1)
58      ;
59  }
```

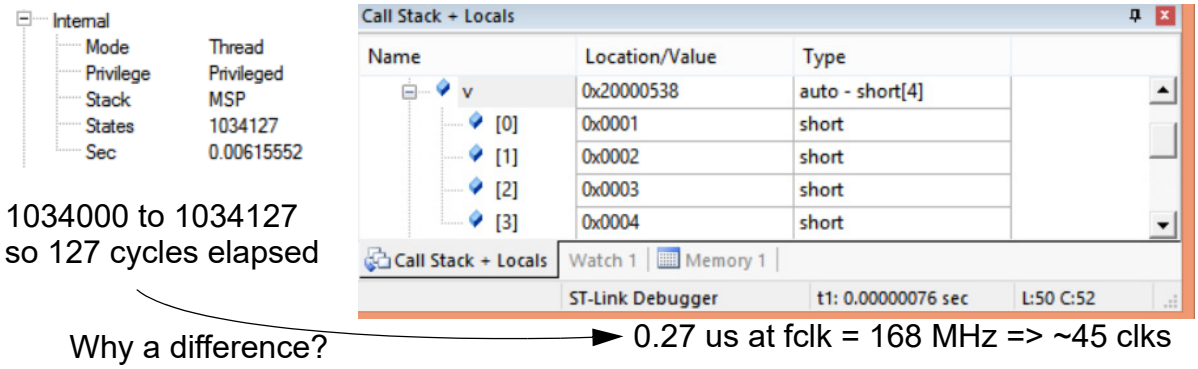
Profiling or Code Timing

- You just saw the code run from main to the first breakpoint
- You also saw that CPU cycle count and clock stopwatch each increment
- Being able to count cycle and/or observe run time in seconds is very valuable in characterizing the performance of real-time code
- If you now clock the run button again, the debugger will advance to line 55 and the cycle count and CPU time will advance by some amount; **don't do it just yet**
- The middle field of the lower right status bar of the Keil app contains two resettable stop watches:



- Reset the t1 stop watch and now click run
- Stop watch now displays the elapsed time in running the `dot_p` function
- The *States* in the registers view has also incremented

- The debugger has knowledge of the CPU clock frequency, so the time increment is a true estimate of the elapsed processing time to *enter-compute-return* from `dot_p`; NICE!



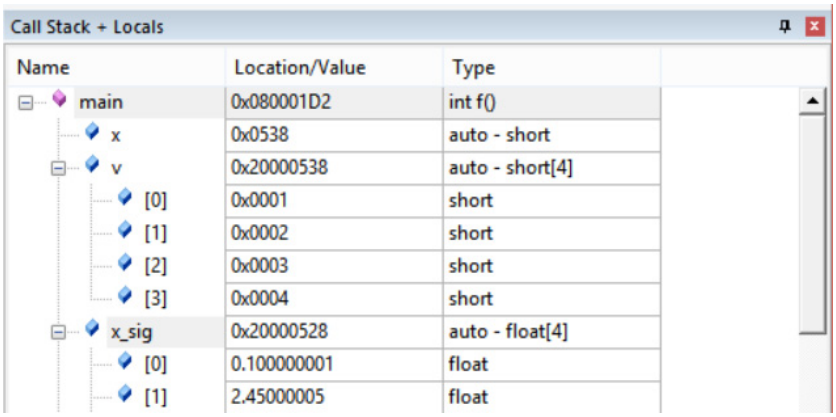
1034000 to 1034127
so 127 cycles elapsed

Why a difference?

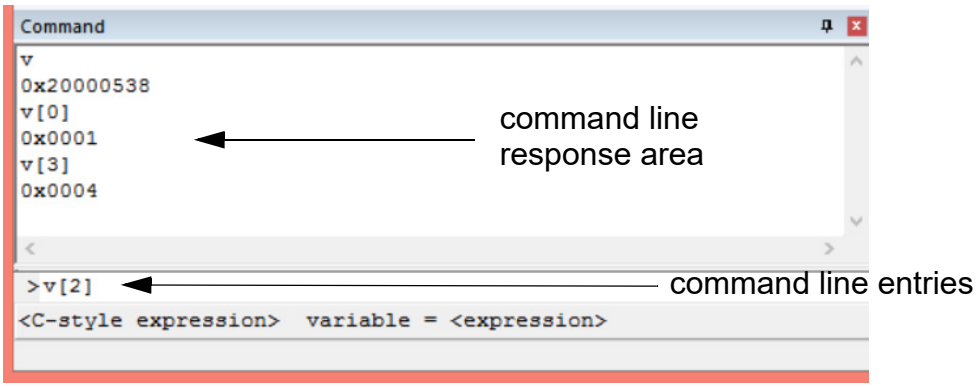
0.27 us at fclk = 168 MHz => ~45 clks

Using the Command Line

- By hovering over variables or looking at the Command Stack + Locals view (lower right above the stop watch display), you can look at variables that are within scope of where the debugger has stopped
- You can expand array variables for example

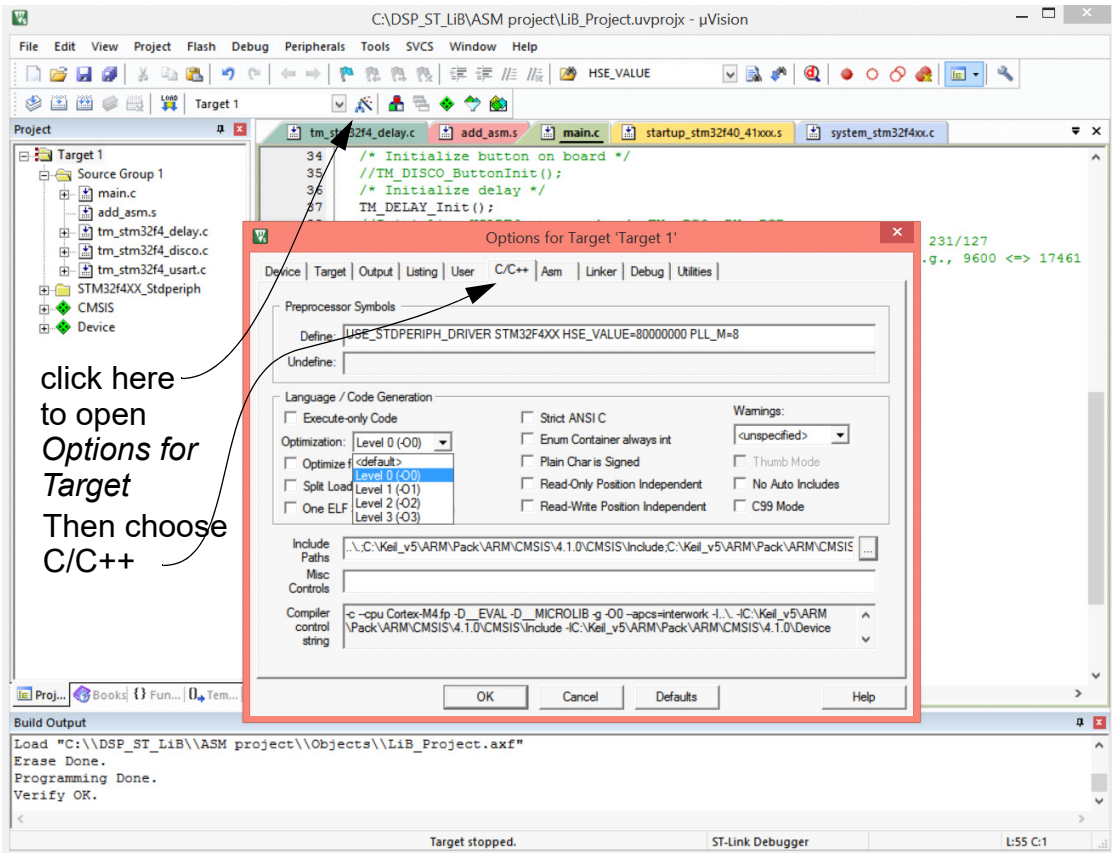


- The command line also allows you to type commands and obtain debug information



Setting the Compiler Optimization Level

- With the debugger stopped you can go to the *Options for Target* button



-

Writing to GPIO for Real-Time Timing

- Writing to a GPIO for scope/logic analyzer timing measurements

Useful Resources

- Architecture Reference Manual:

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403c/index.html

- Cortex-M4 Technical Reference Manual:

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439d/DDI0439D_cortex_m4_processor_r0p1_trm.pdf

- Cortex-M4 Devices Generic User Guide:

http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf

- The Designer's Guide to the Cortex-M Processor Family: A Tutorial Approach (Martin)

http://www.amazon.com/Designers-Guide-Cortex-M-Processor-Family/dp/0080982964/ref=sr_1_4?ie=UTF8&qid=1423511335&sr=8-4&keywords=Cortex-M

- Embedded Systems: Real-Time Operating Systems for Arm Cortex M Microcontrollers (Valvano)

http://www.amazon.com/Embedded-Systems-Real-Time-Operating-Microcontrollers/dp/1466468866/ref=sr_1_1?s=books&ie=UTF8&qid=1423511893&sr=1-1&keywords=Real-Time+Operating+Systems+for+Cortex-M