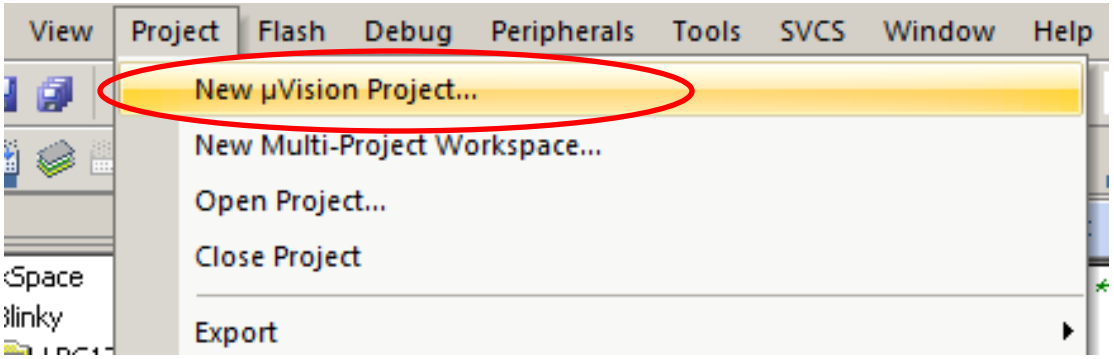
Introduction to Keil IDE and MCB1700

Irene Huang

KEIL IDE INTRODUCTION

Creating a New Project

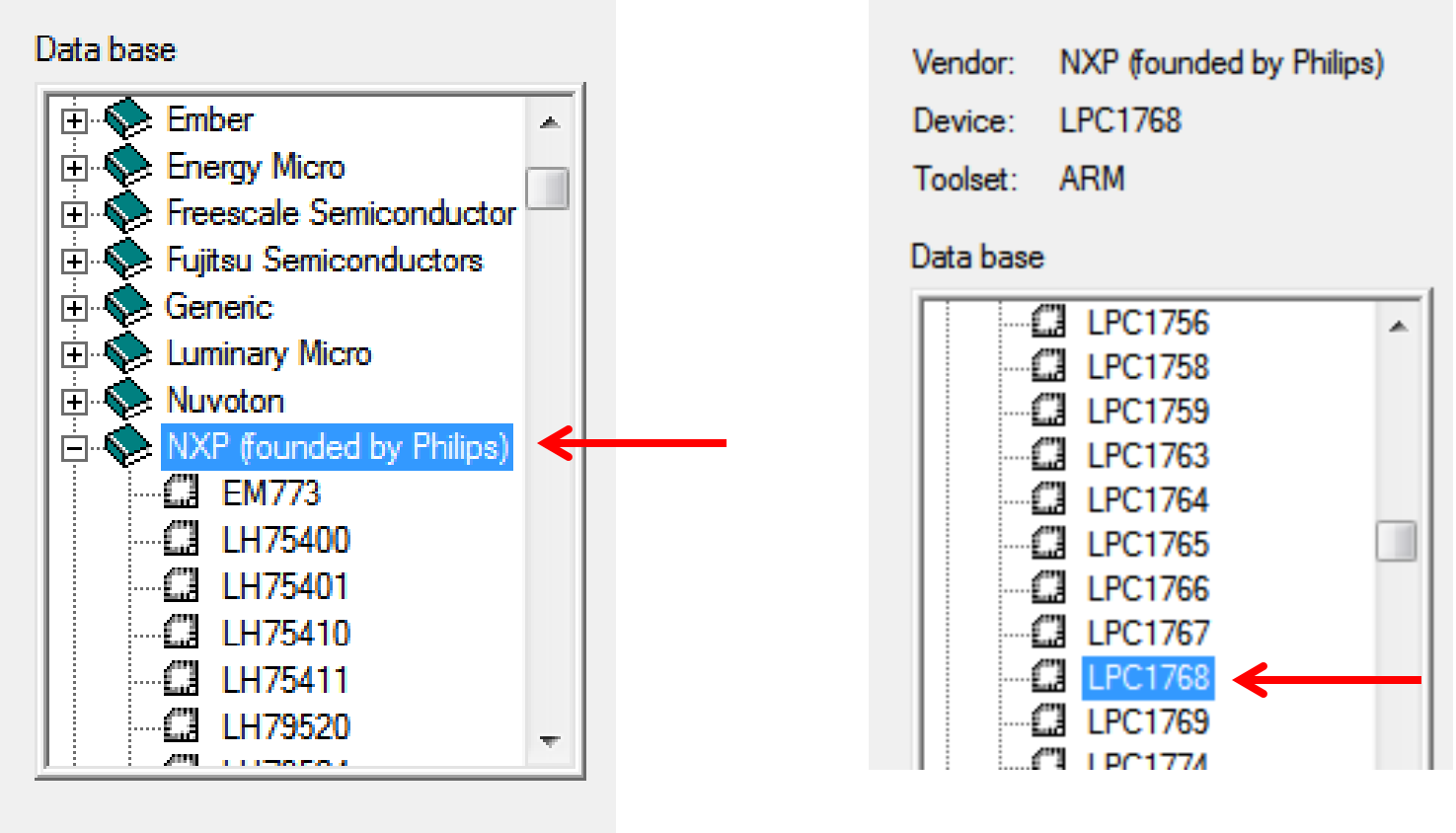
- Create a new folder and name it “Hello”
- Copy the following files to the “Hello” folder
  - `manaul_code\Startup\system_LPC17xx.c`
  - `manual_code\UART_polling\src\uart_polling.c`
  - `manual_code\UART_polling\src\uart_polling.h`
- Create a new µVision by click
  - Project → New µVision Project



No Space allowed in the project folder path name on Nexus!

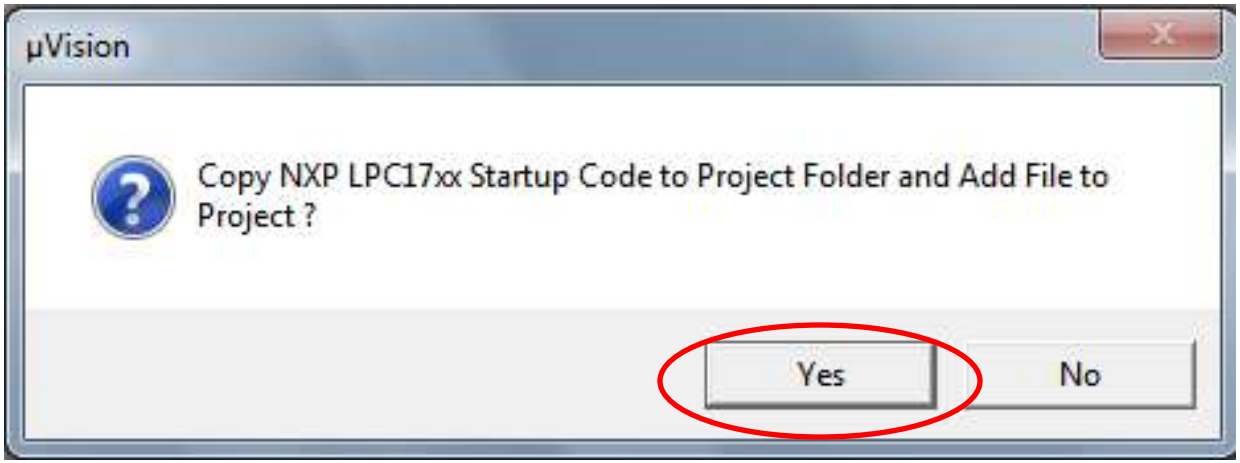
Choosing the Device

- Choose NXP(Founded by Philips) → LPC1768



Copying the startup Code

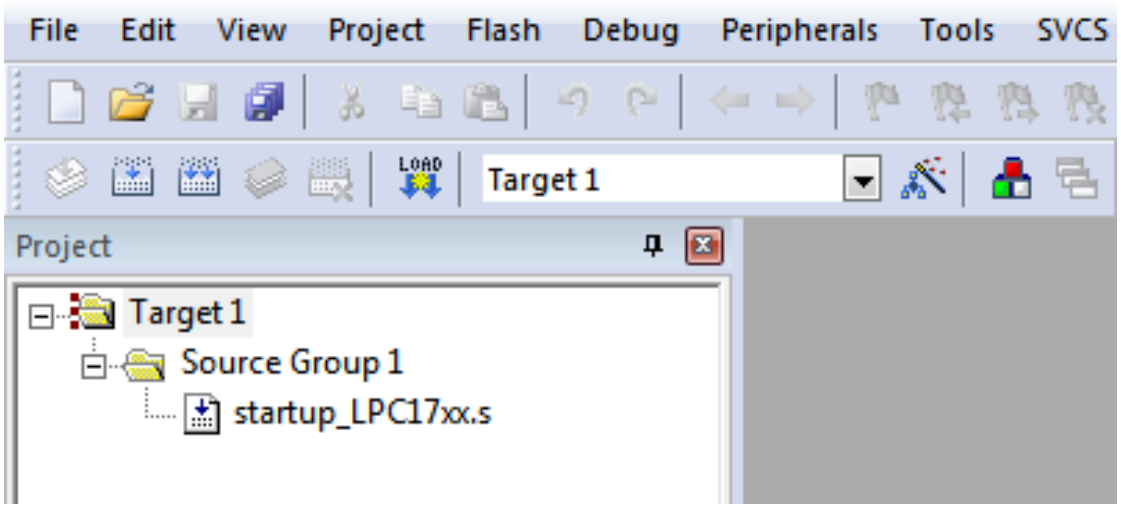
- Answer "**Yes**" to copy the startup code



- If you answer "No", you need to copy the file manually from the example code folder.
  - `manual_code\Startup\startup_LPC17xx.s`

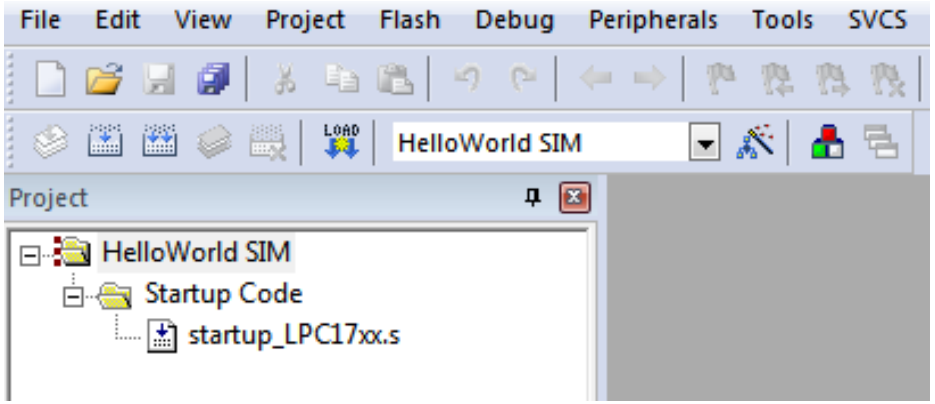
Default Project Components

- You will see the default project setup as follows:



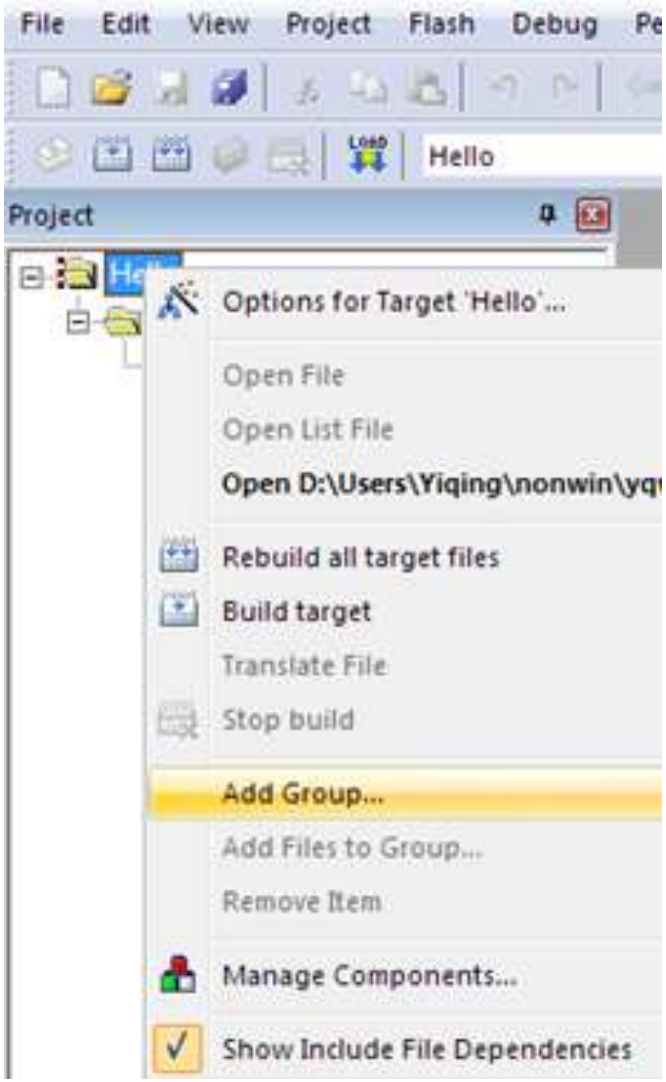
Renaming Project Components

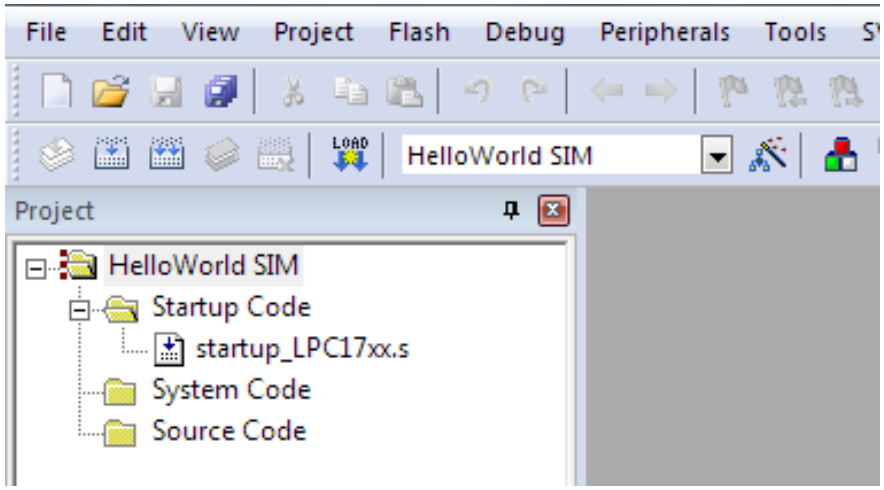
- Rename the default target name
  - Click the target name to highlight it
  - Click the highlighted name to input a new target name, say “HelloWorld SIM”
- Rename the default source group name
  - Click the default source group to highlight it
  - Click the highlighted name to input a new name, say “Startup Code”



Adding New Source Groups

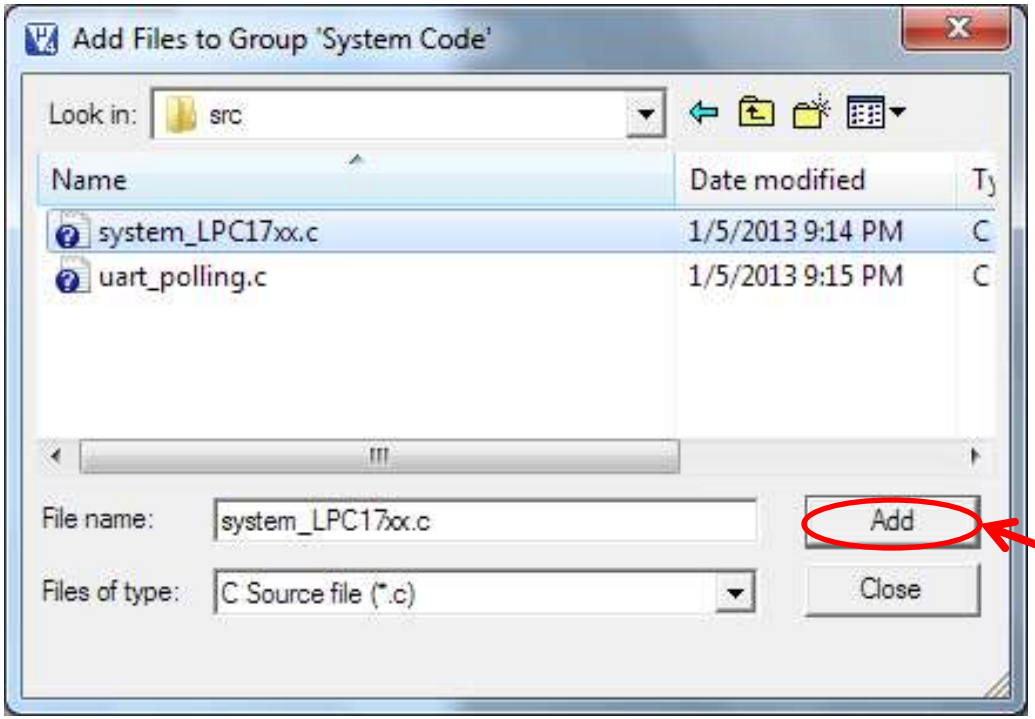
- Right click the target name and add the following source groups
  - System Code
  - Source Code
- You will see the following





Adding Files to Source Groups

- Double click the “System Code” group and add the following files to it.
  - **system_LPC17xx.c**

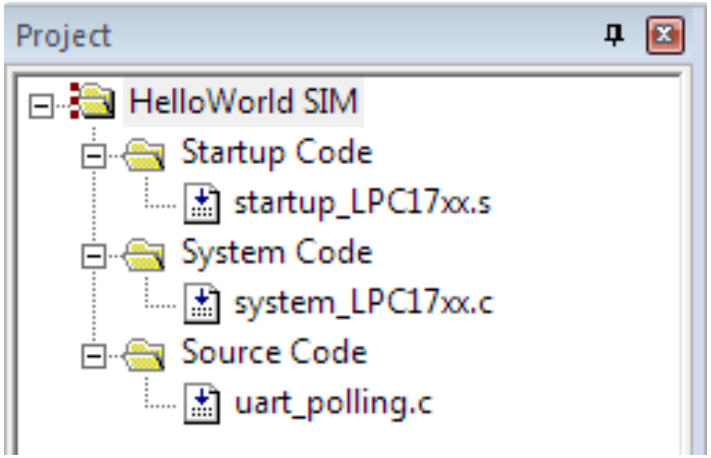


Double click file names to add
or
highlight file names and click “Add” button.

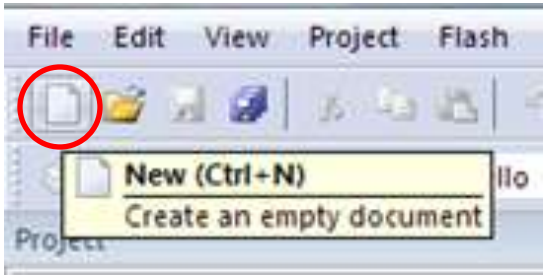
- Add “uart_polling.c” to “Source Code” Group

Creating Your Own Source Code

- Your project would now look like this.

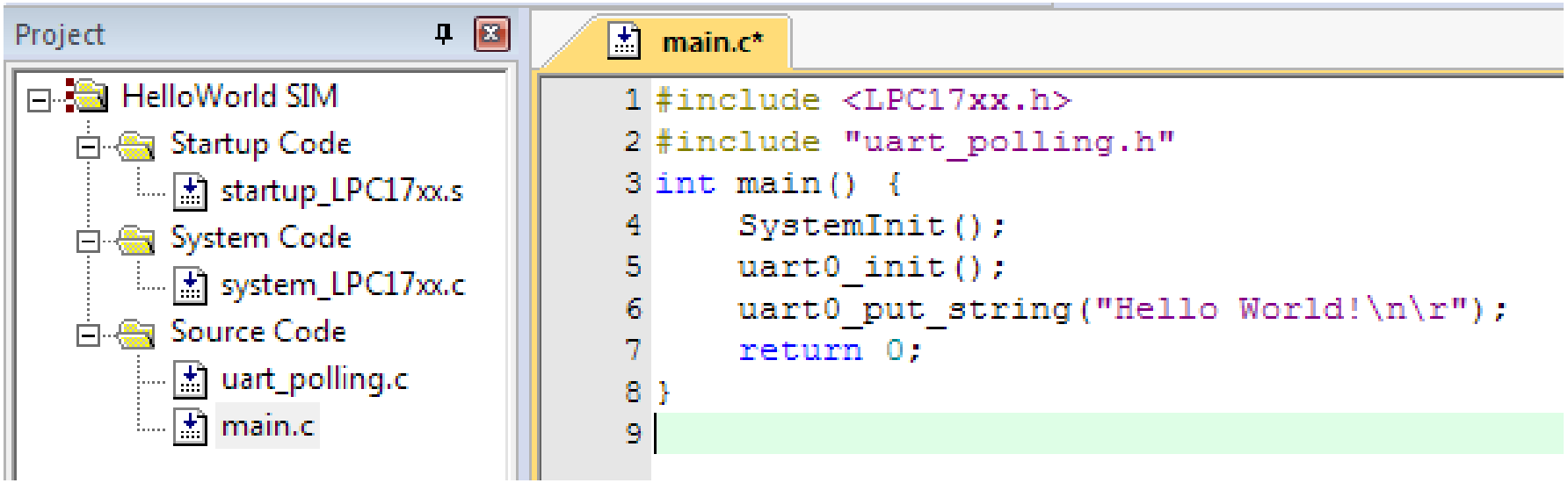


- Click New button to Create a `main.c` file.



Adding main.c to Project

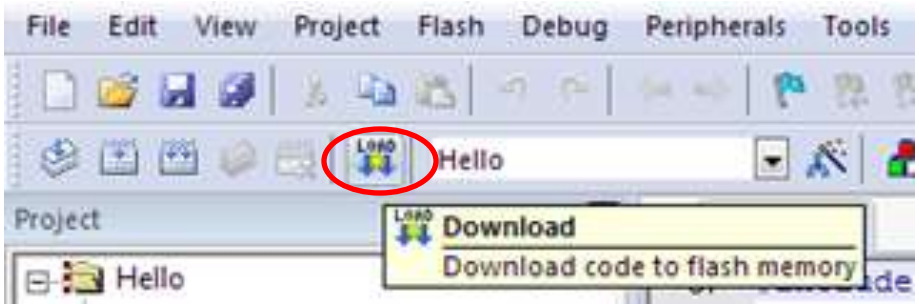
- Add the `main.c` to the "Source Code" group.



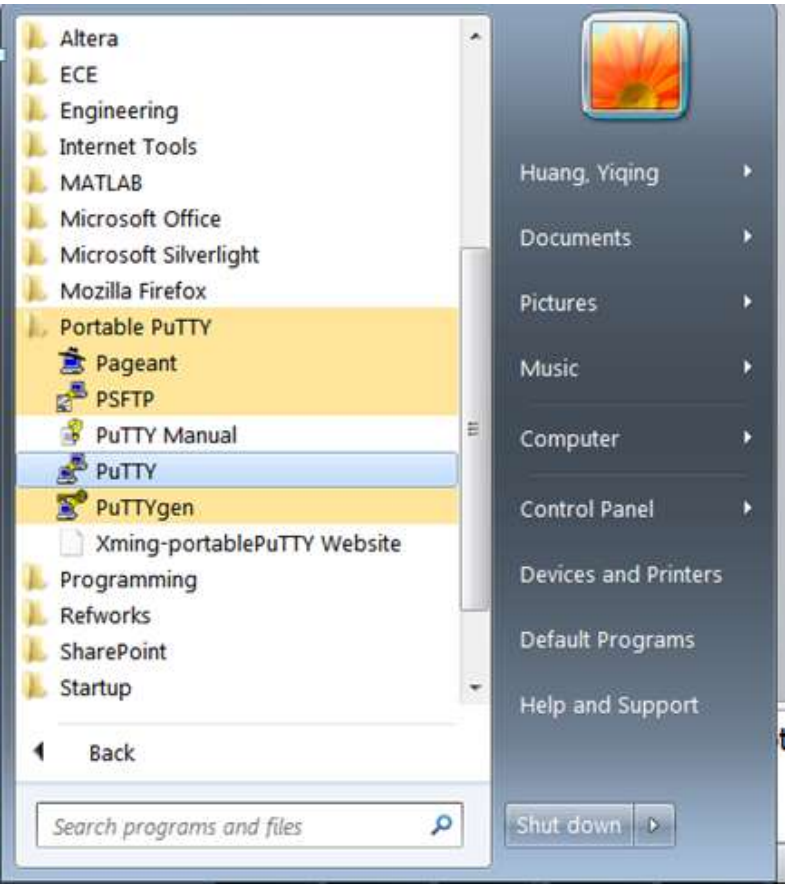
Build and Download

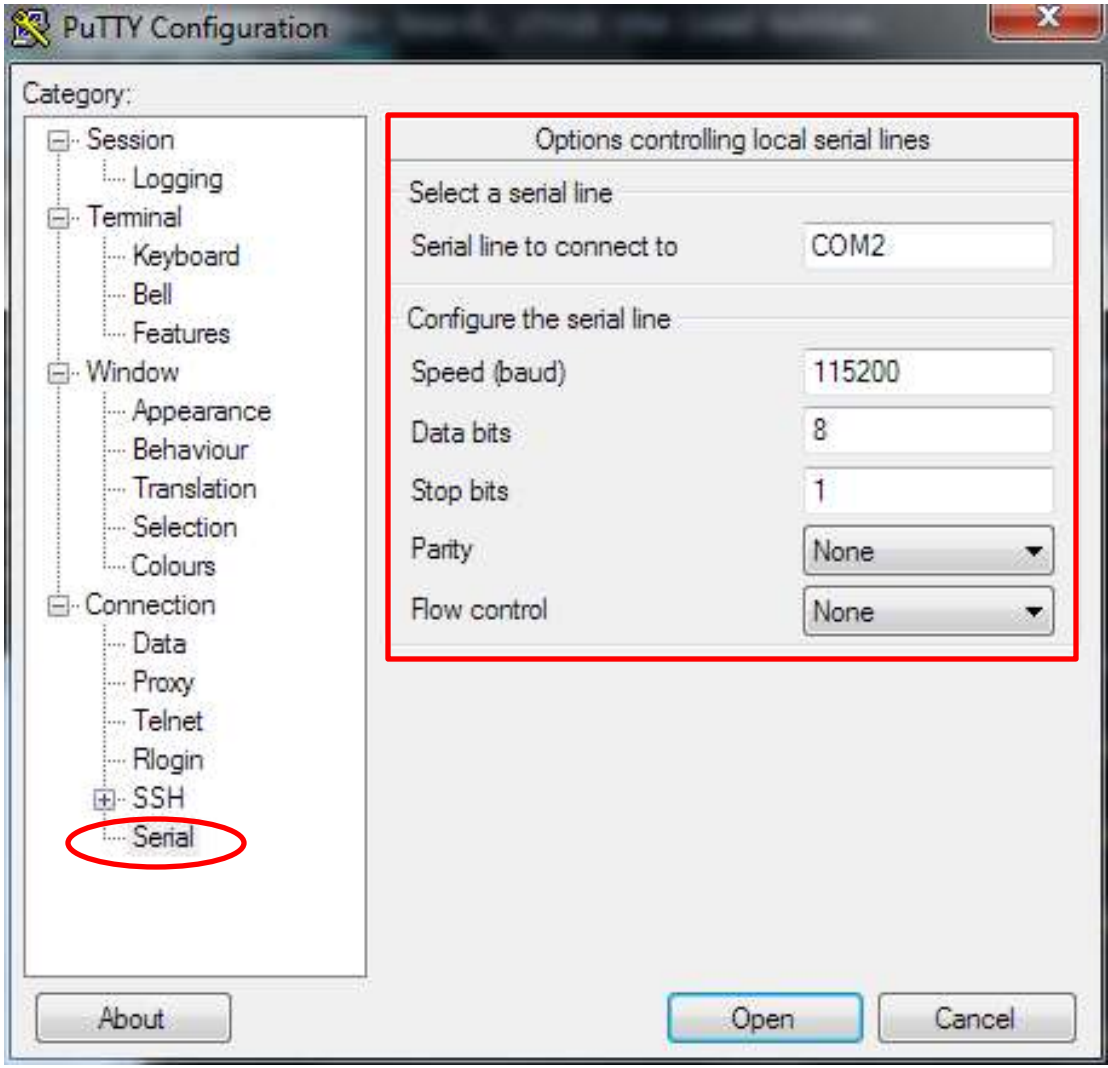
- Click the Build button

- Click the Download button



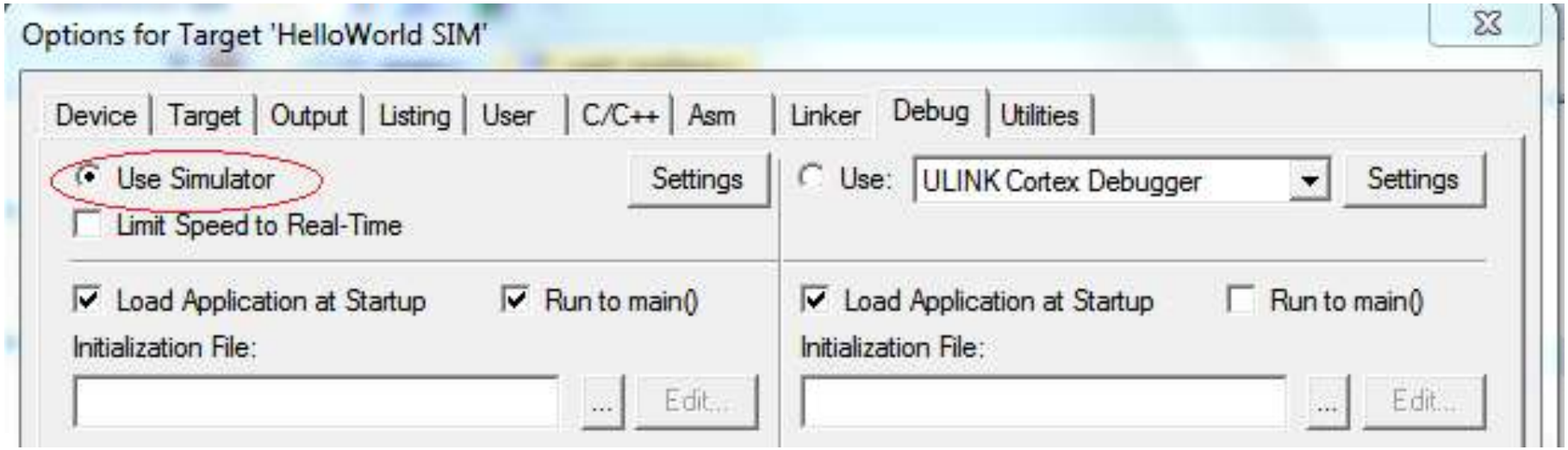
RUN on the Board

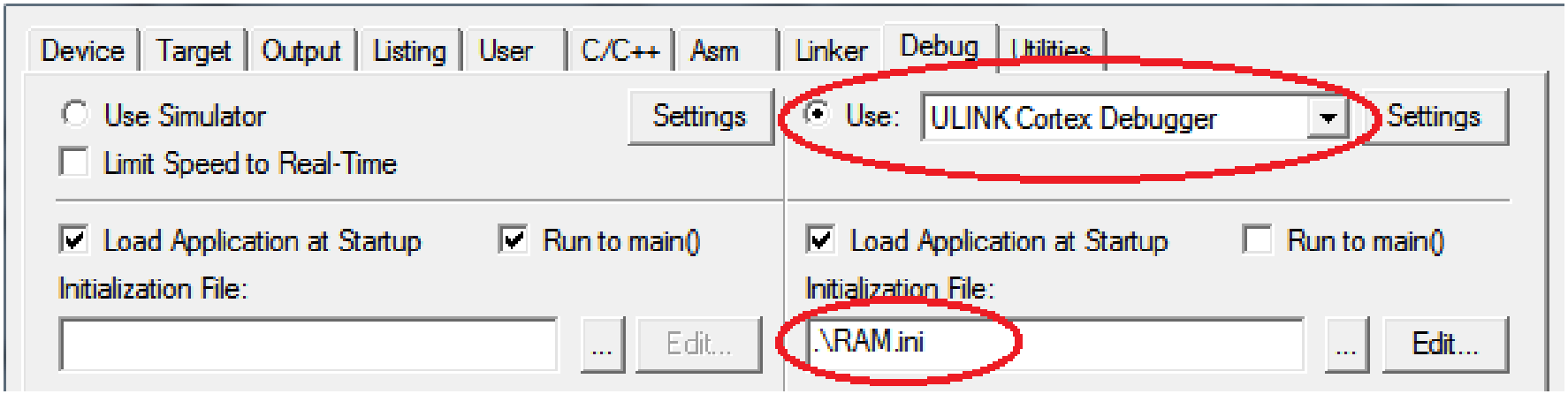




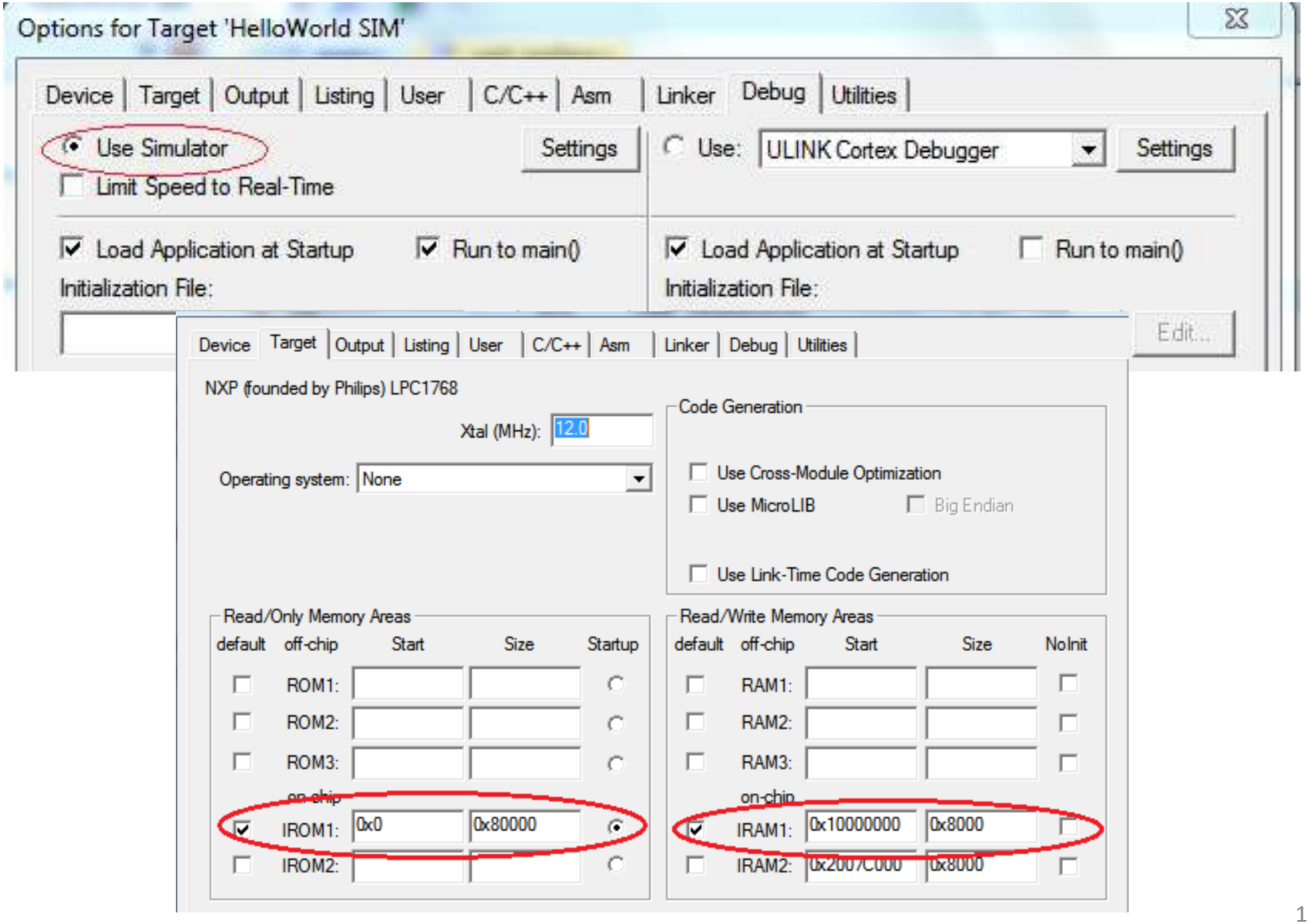
Press the “Reset” button on the board to run

Debug Configuration

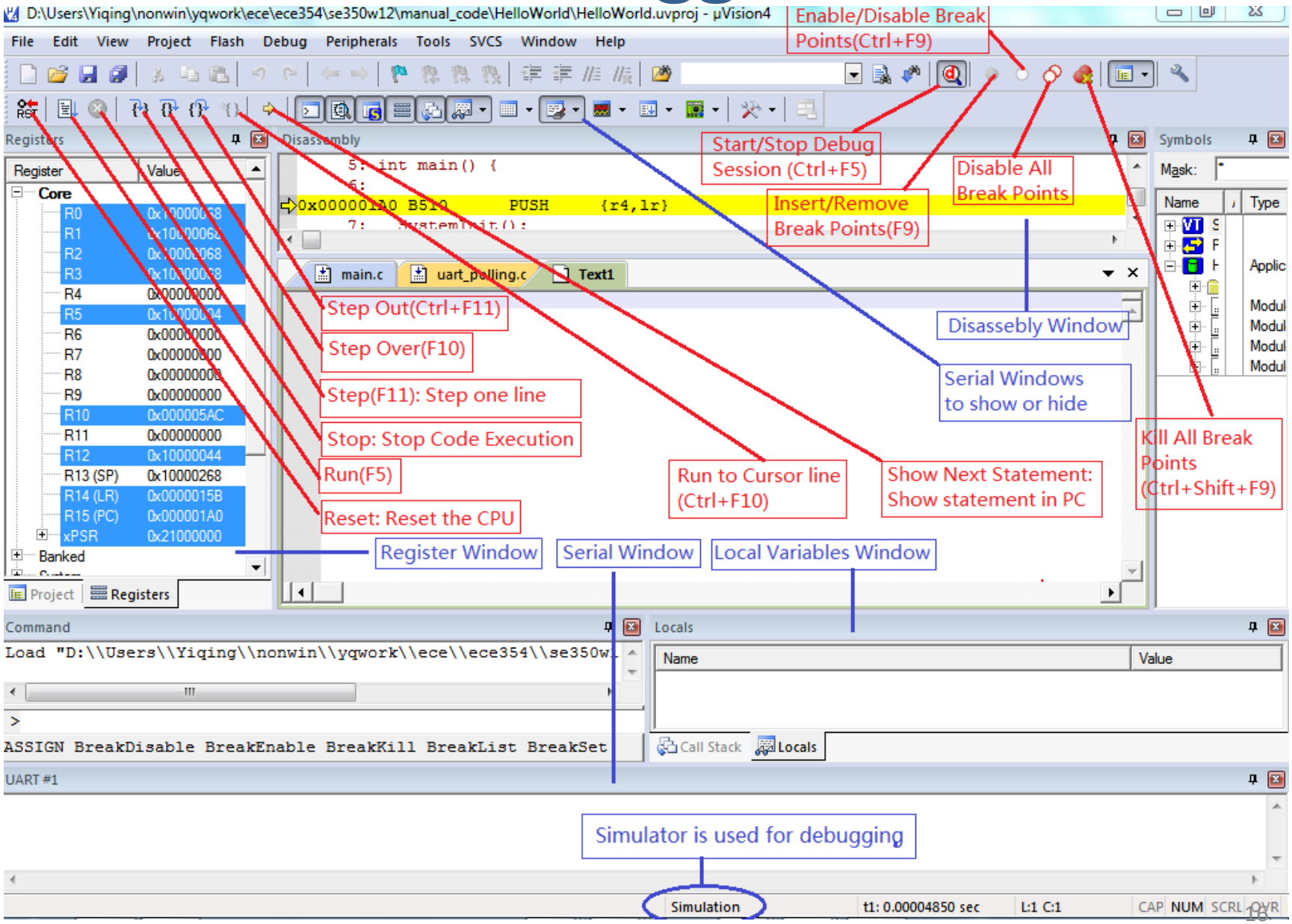


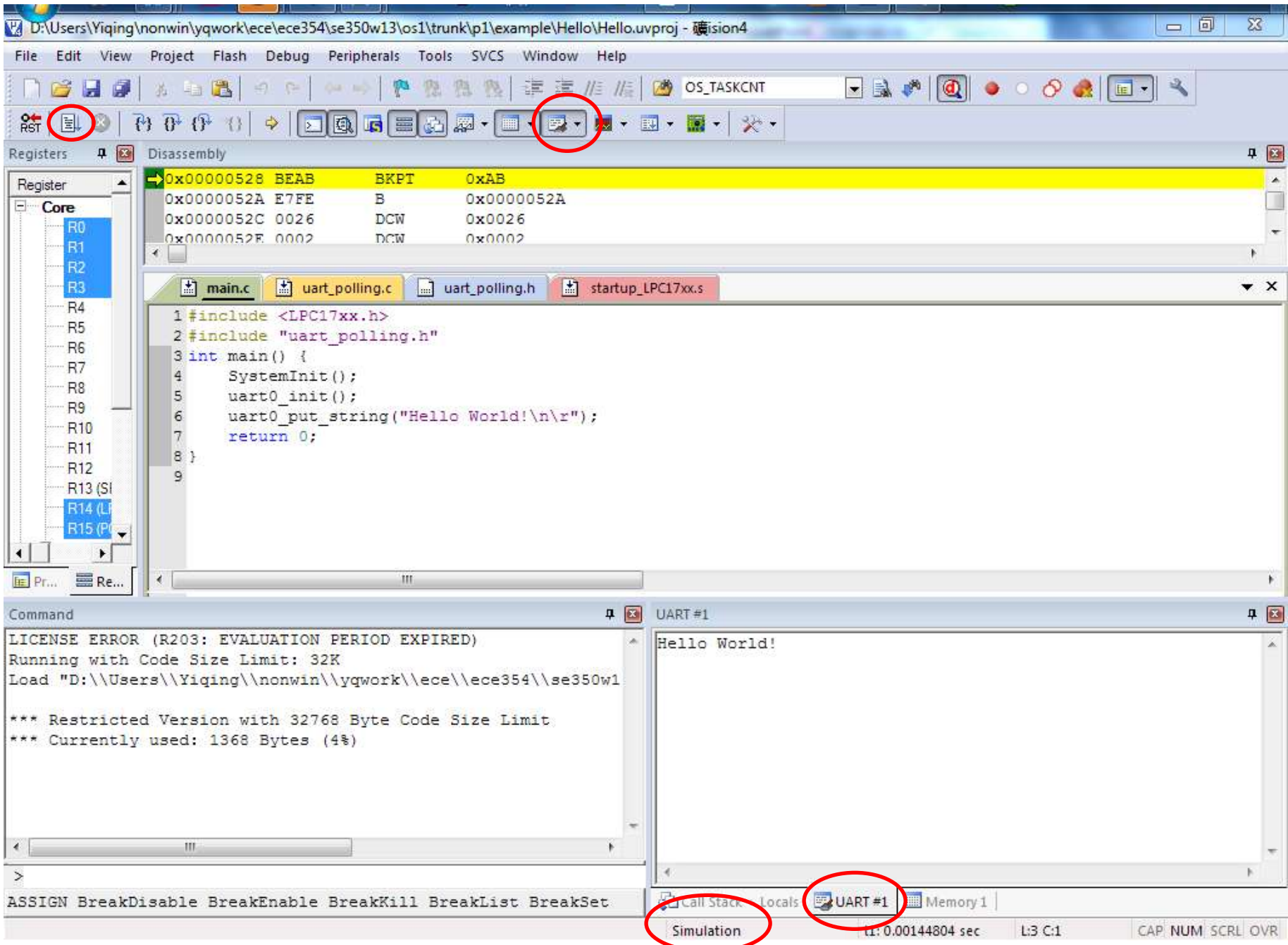


Debug by Simulator

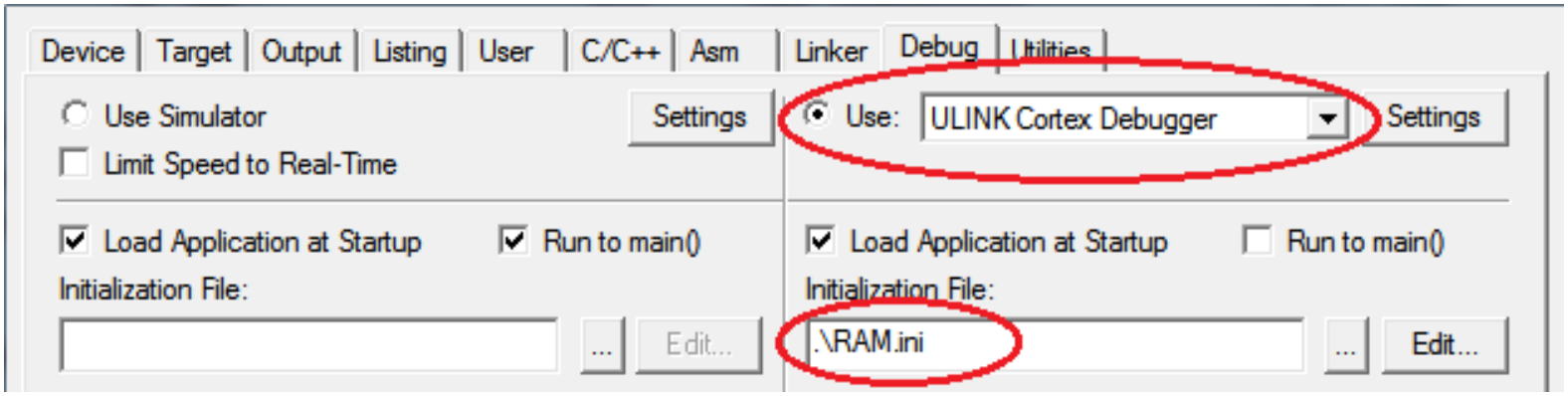


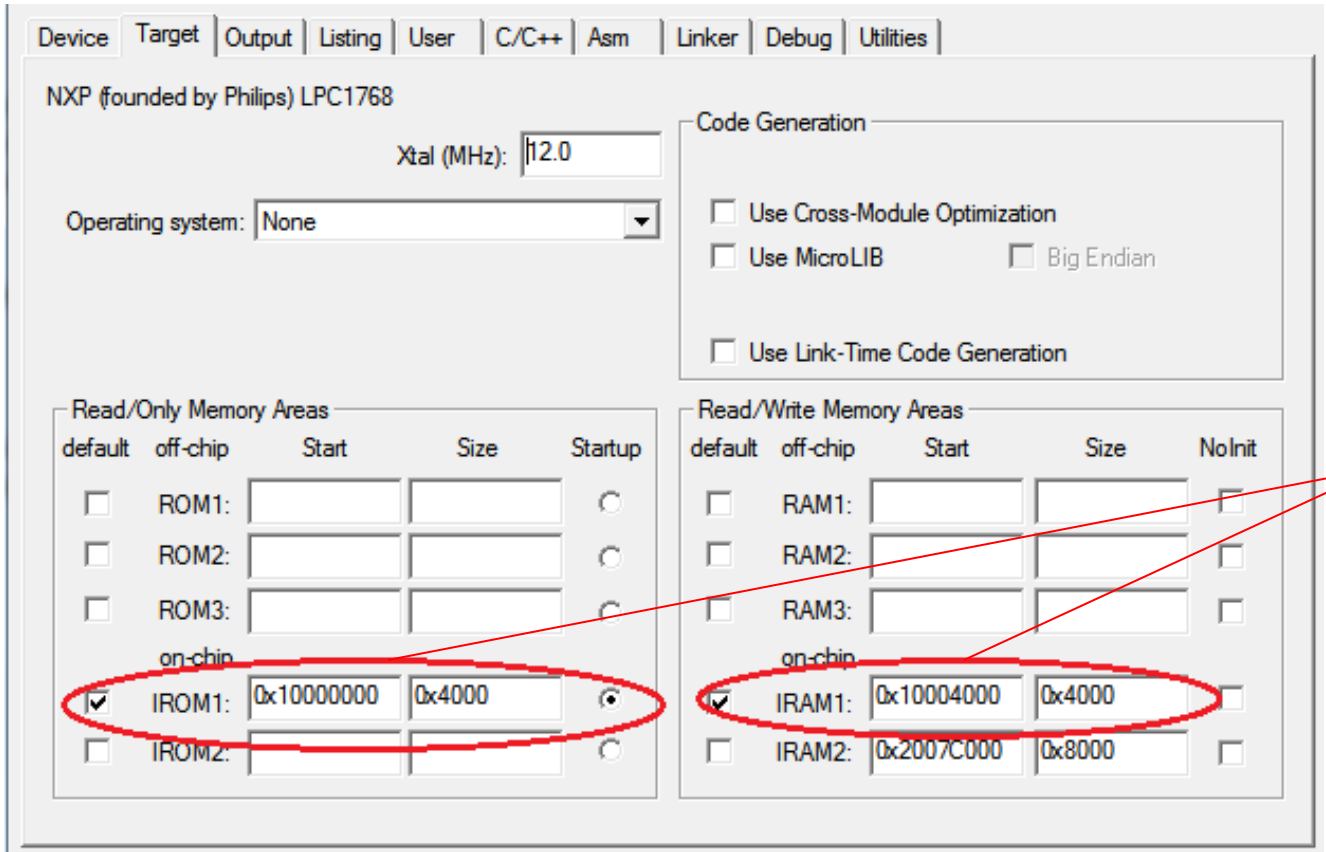
Debugger


Enable/Disable Break Points(Ctrl+F9)
Start/Stop Debug Session (Ctrl+F5)
Disable All Break Points
Insert/Remove Break Points(F9)
Step Out(Ctrl+F11)
Step Over(F10)
Step(F11): Step one line
Stop: Stop Code Execution
Run(F5)
Reset: Reset the CPU
Disassebly Window
Serial Windows to show or hide
Kill All Break Points (Ctrl+Shift+F9)
Run to Cursor line (Ctrl+F10)
Show Next Statement: Show statement in PC
Register Window
Serial Window
Local Variables Window
Simulator is used for debugging


D:\Users\Yiqing\nonwin\yqwork\ece\ece354\se350w13\os1\trunk\p1\example\Hello\Hello.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
OS_TASKCNT
Registers
Register
Core
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (SI
R14 (LI
R15 (PC
Disassembly
0x00000528 BEAB BKPT 0xAB
0x0000052A E7FE B 0x0000052A
0x0000052C 0026 DCW 0x0026
0x0000052E 0002 DCW 0x0002
main.c
uart_polling.c
uart_polling.h
startup_LPC17xx.s
1 #include <LPC17xx.h>
2 #include "uart_polling.h"
3 int main() {
4 SystemInit();
5 uart0_init();
6 uart0_put_string("Hello World!\n\r");
7 return 0;
8 }
9
Command
LICENSE ERROR (R203: EVALUATION PERIOD EXPIRED)
Running with Code Size Limit: 32K
Load "D:\\Users\\Yiqing\\nonwin\\yqwork\\ece\\ece354\\se350w1
*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 1368 Bytes (4%)
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
UART #1
Hello World!
Call Stack
Locals
UART #1
Memory 1
Simulation
t1: 0.00144804 sec
L:3 C:1
CAP NUM SCRL OVR

Debug by ULINK Cortex Debugger





Configure In-Memory Execution

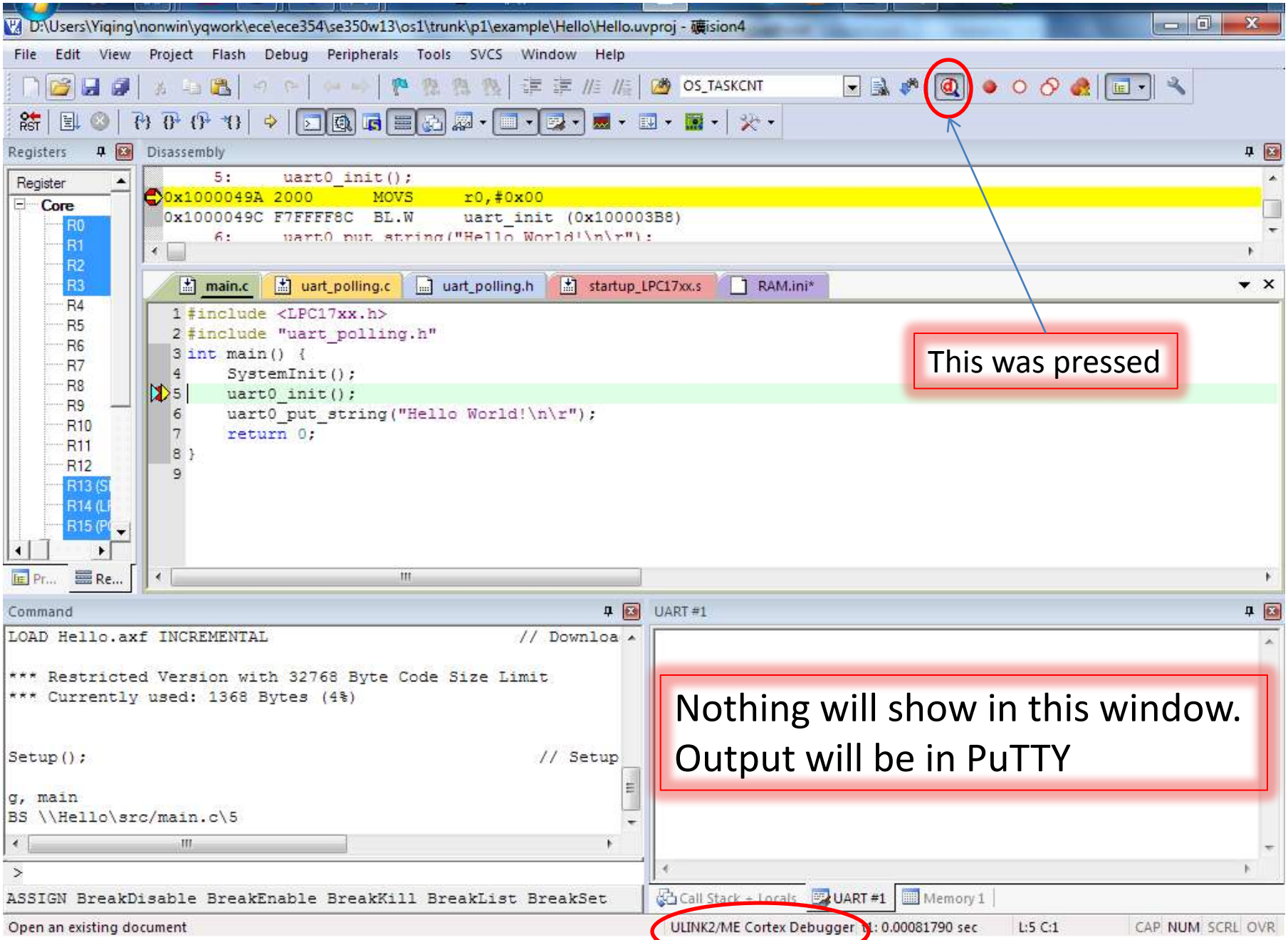
Example RAM.ini File

```
FUNC void Setup (void) {
  SP = _RDWORD(0x10000000);  // Setup Stack Pointer
  PC = _RDWORD(0x10000004);  // Setup Program Counter

  // Setup Vector Table Offset Register
  _WDWORD(0xE000ED08, 0x10000000);

}

LOAD Hello.axf INCREMENTAL   // Download
Setup();                     // Setup for Running
g, main
```


D:\Users\Yiqing\nonwin\yqwork\ece\ece354\se350w13\os1\trunk\p1\example\Hello\Hello.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
OS_TASKCNT
Registers
Register
Core
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
Disassembly
5: uart0_init();
0x1000049A 2000 MOVS r0,#0x00
0x1000049C F7FFFF8C BL.W uart_init (0x100003B8)
main.c
uart_polling.c
uart_polling.h
startup_LPC17xx.s
RAM.ini*
1 #include <LPC17xx.h>
2 #include "uart_polling.h"
3 int main() {
4 SystemInit();
5 uart0_init();
6 uart0_put_string("Hello World!\n\r");
7 return 0;
8 }
9
This was pressed
Command
LOAD Hello.axf INCREMENTAL // Downloa
*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 1368 Bytes (4%)
Setup(); // Setup
g, main
BS \\Hello\src/main.c\5
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
UART #1
Nothing will show in this window.
Output will be in PuTTY
Call Stack + Locals
UART #1
Memory 1
Open an existing document
ULINK2/ME Cortex Debugger
L:5 C:1
CAP NUM SCRL OVR

RTX PROJECT P1 REQUIREMENTS

P1 Requirements : API

- Memory Management: a memory pool which has fixed size of memory block and fixed number of memory blocks.

```
void *request_memory_block()
int release_memory_block(void *MemoryBlock)
```

- Processor Management

```
int release_processor()
```

- Process Priority Management

```
int set_process_priority(int process_ID, int priority)
int get_process_priority(int process_ID )
```

P1 Requirements: Processes

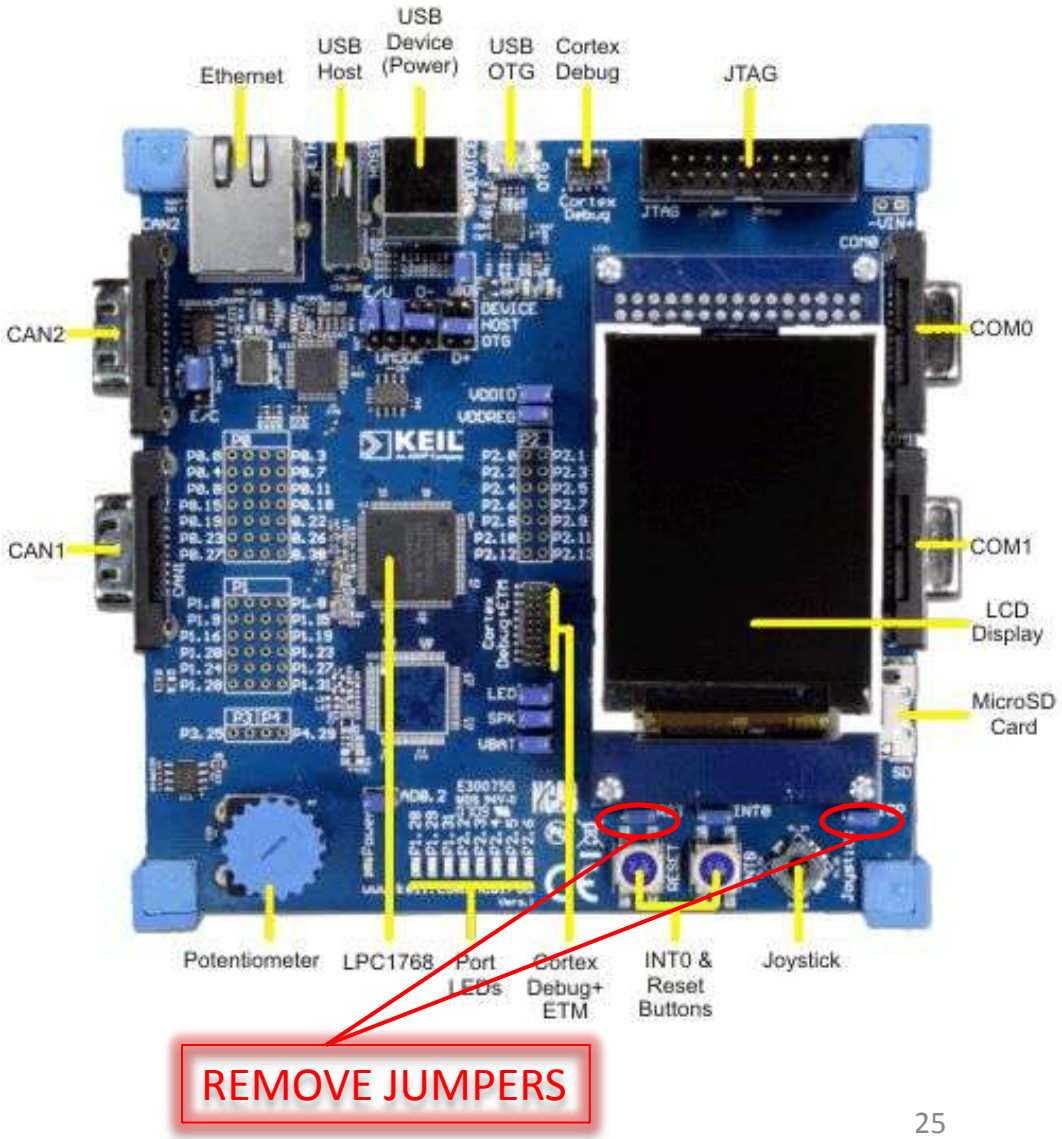
- Null Process
  - A system process which does nothing in an infinite loop. PID=0.
- Test Processes
  - Up to six test processes with PIDs = 1,2, …, 6
  - User level processes, only calls the user APIs
- Initialization
  - Memory, system processes and user processes

All processes never terminate!
No new process created on the fly.

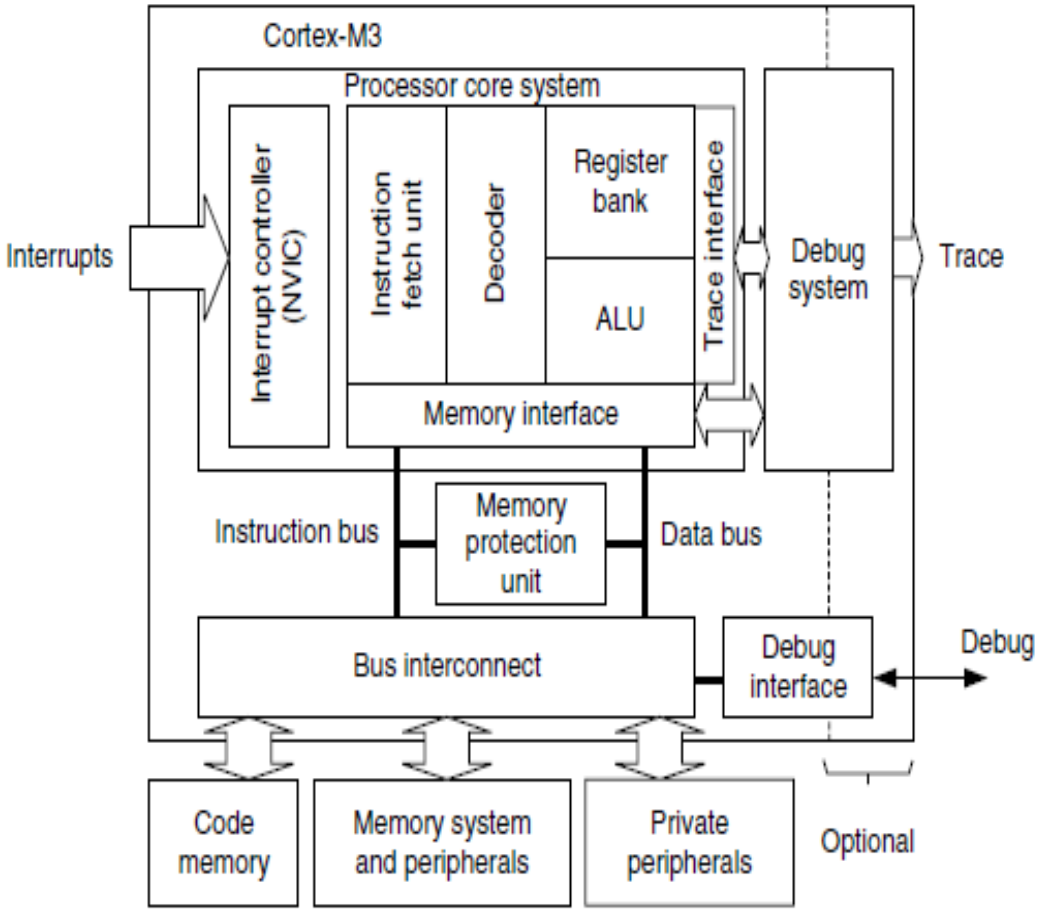
MCB1700 HARDWARE

Keil MCB 1700 Board

- Cortex-M3 Processor
- NXP LPC1768 MCU
- Up to 100 MHZ cpu
- One SystemTick Timer
- Four Timers
- Four UARTs
- Many other cool stuff..



Cortex-M3 Overview



(Image Courtesy of [1])

- 32-bit microprocessor
  - 32-bit data path
  - 32-bit register bank
  - 32-bit memory interface
- Harvard Architecture
  - Separate data and memory bus
  - instruction and data buses share the same memory space (a unified memory system)

Cortex-M3 Registers

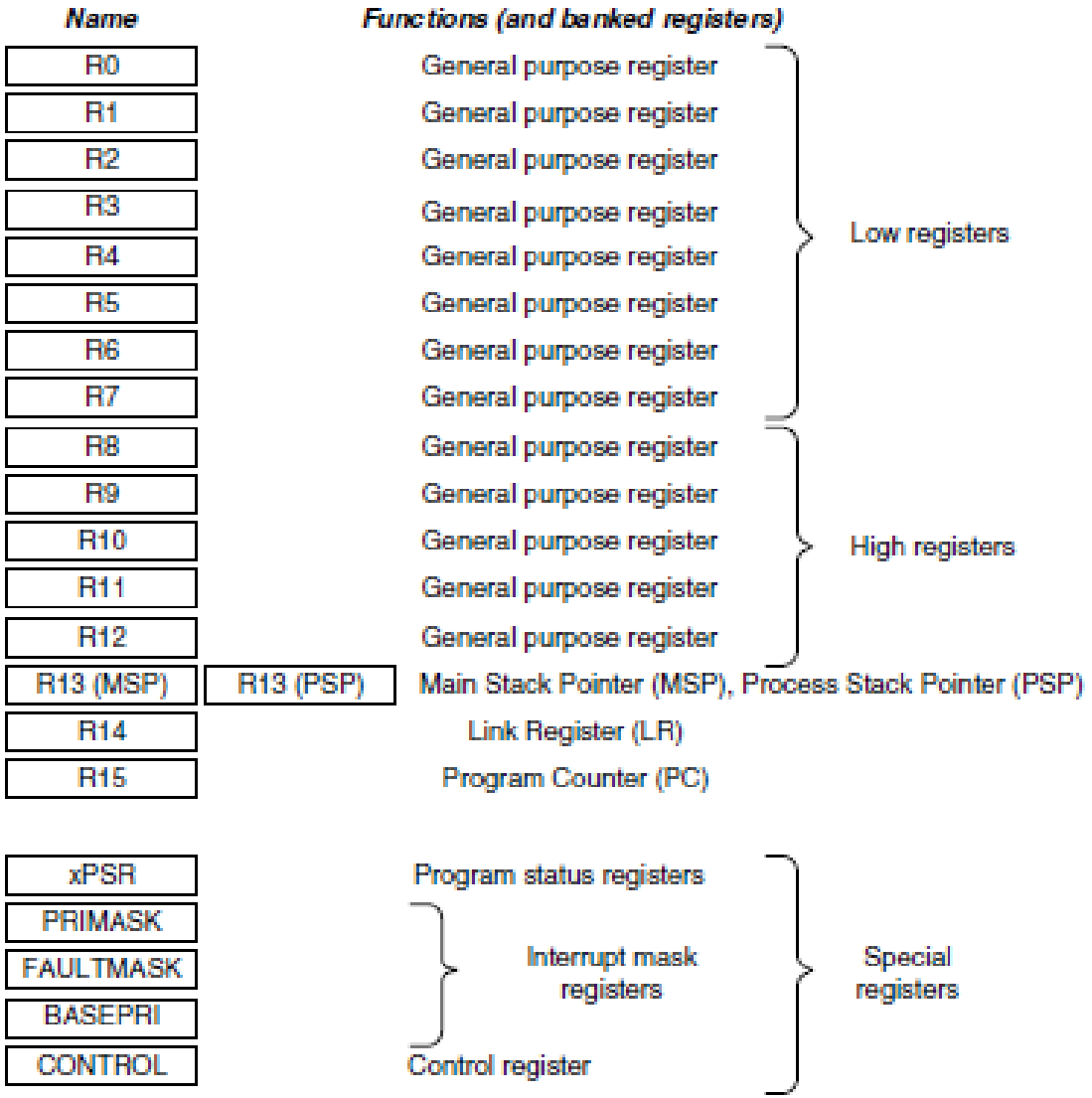
- General Purpose Registers (R0-R15)
  - Low registers (R0-R7)
    - 16-bit Thumb instructions and 32-bit Thumb-2 instructions
  - High registers (R8-R12)
    - All Thumb-2 instructions
  - Stack Pointer (R13)
    - **MSP:** Privileged, default after reset, os kernel, exception handler
    - **PSP:** Uer-level (i.e. unprivileged) base-level application
  - Link Register (R14)
  - Program Counter (R15)
- Special Registers
  - Program Status registers (PSRs)
  - Interrupt Mask registers (PRIMASK, FAULTMASK, and BASEPRI)
  - Control register (CONTROL)

Cortex-M3 Registers

32-bit microprocessor
32-bit data path
32-bit register bank
32-bit memory interface
Harvard Architecture
Separate data and memory bus

Low registers: R0-R7
16-bit Thumb instructions
32-bit Thumb-2 instructions
High registers: R9-R12
All Thumb-2 instructions

MSP: default after reset
os kernel, exception handler
Privileged
PSP: base-level application
unprivileged, user-level



(Image Courtesy of [1])

Cortex-M3 Memory Map

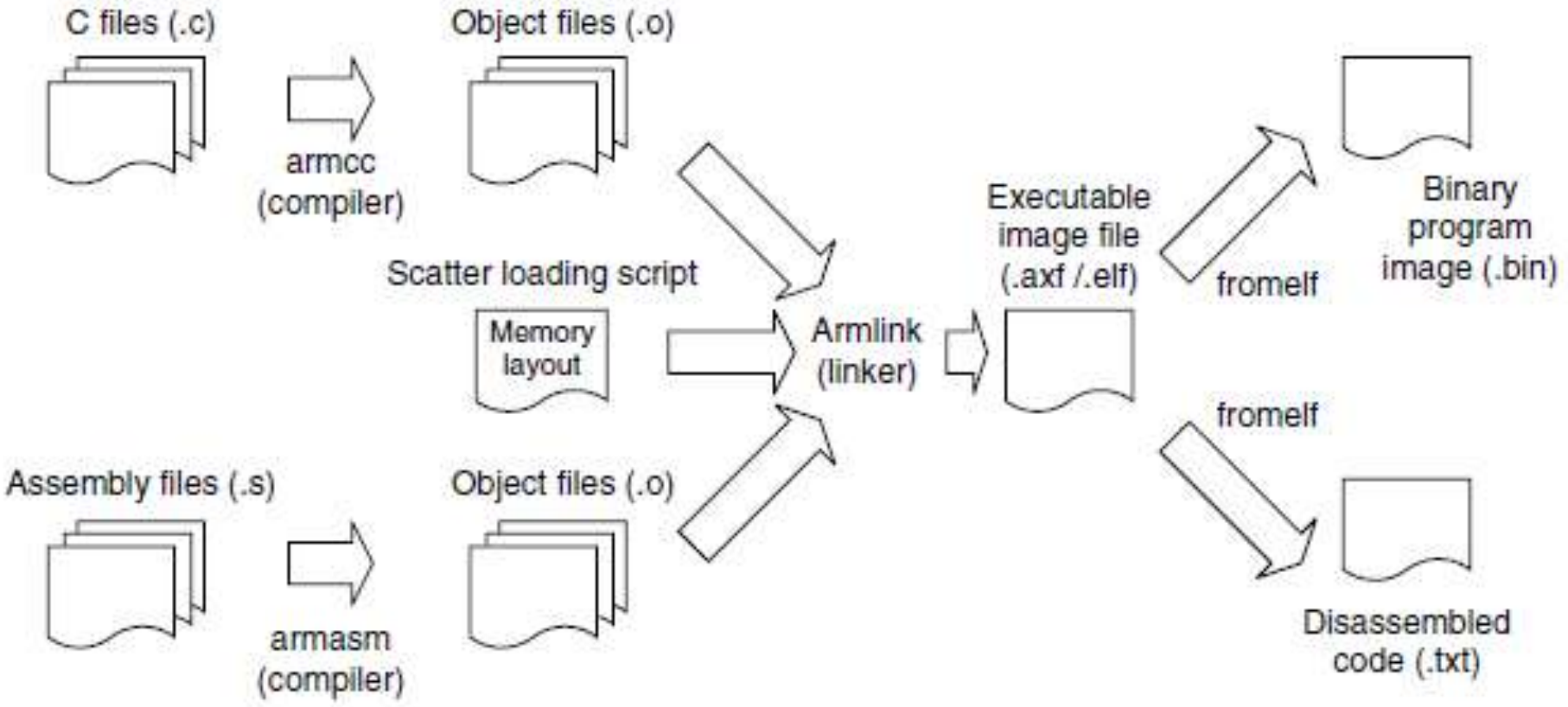
0xFFFF FFFF 0xE000 0000	0.5G	System level	Private peripherals including NVIC, MPU and debug components	★
0xA000 0000	1.0G	External device	Mainly used as external peripherals	
0x6000 0000	1.0G	External RAM	Mainly used as external memory	
0x4000 0000	0.5G	Peripherals	Mainly used as peripherals	
0x2000 0000	0.5G	SRAM	Mainly used as static RAM	
0x0000 0000	0.5G	Code	Mainly used fro program code. Exception vector table after reset	★

(Table Courtesy of [1])

LPC1768 Memory Map

0x2008 4000		
0x2007 C000	32 KB	AHB SRAM (2 blocks of 16 KB)
0x1FFF 2000		Reserved
0x1FFF 0000	8 KB	Boot ROM
0x1000 8000		Reserved
0x1000 0000	32 KB	Local SRAM
0x0008 0000		Reserved
0x0000 0000	512 KB	On-chip flash

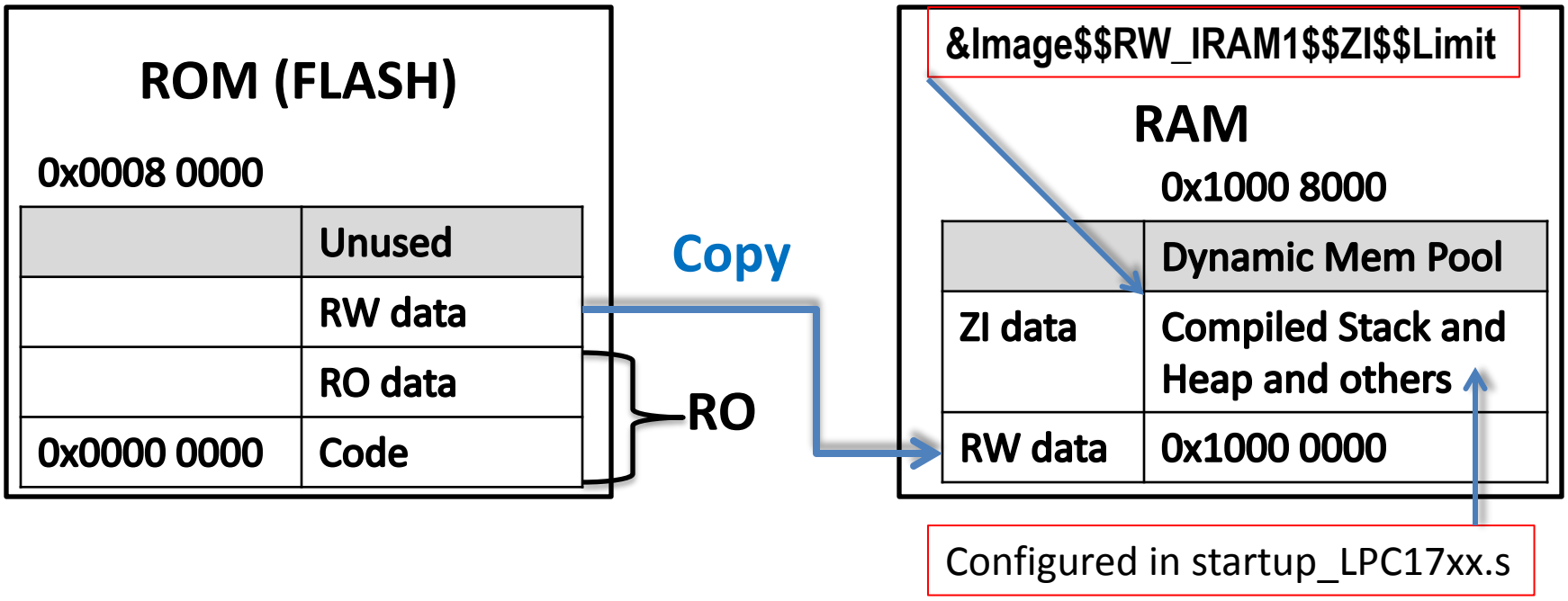
Example Flow Using ARM Development Tools



(Image Courtesy of [1])

Image memory layout

- A simple image consists of:
  - read-only (RO) section (Code + RO-data)
  - a read-write (RW) section (RW-data)
  - a zero-initialized (ZI) section (ZI-data)



End Address of the Image

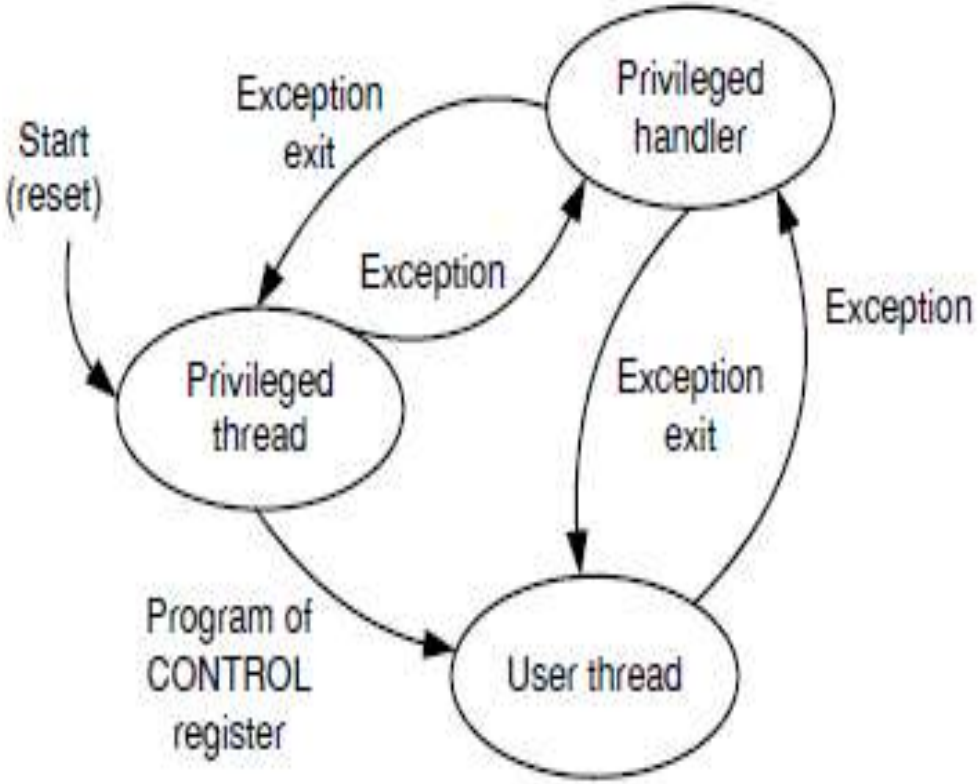
- Linker defined symbol Image$$RW_IRAM1$$ZI$$Limit

```
extern unsigned int Image$$RW_IRAM1$$ZI$$Limit;

unsigned int free_mem =
    (unsigned int) &Image$$RW_IRAM1$$ZI$$Limit;
```

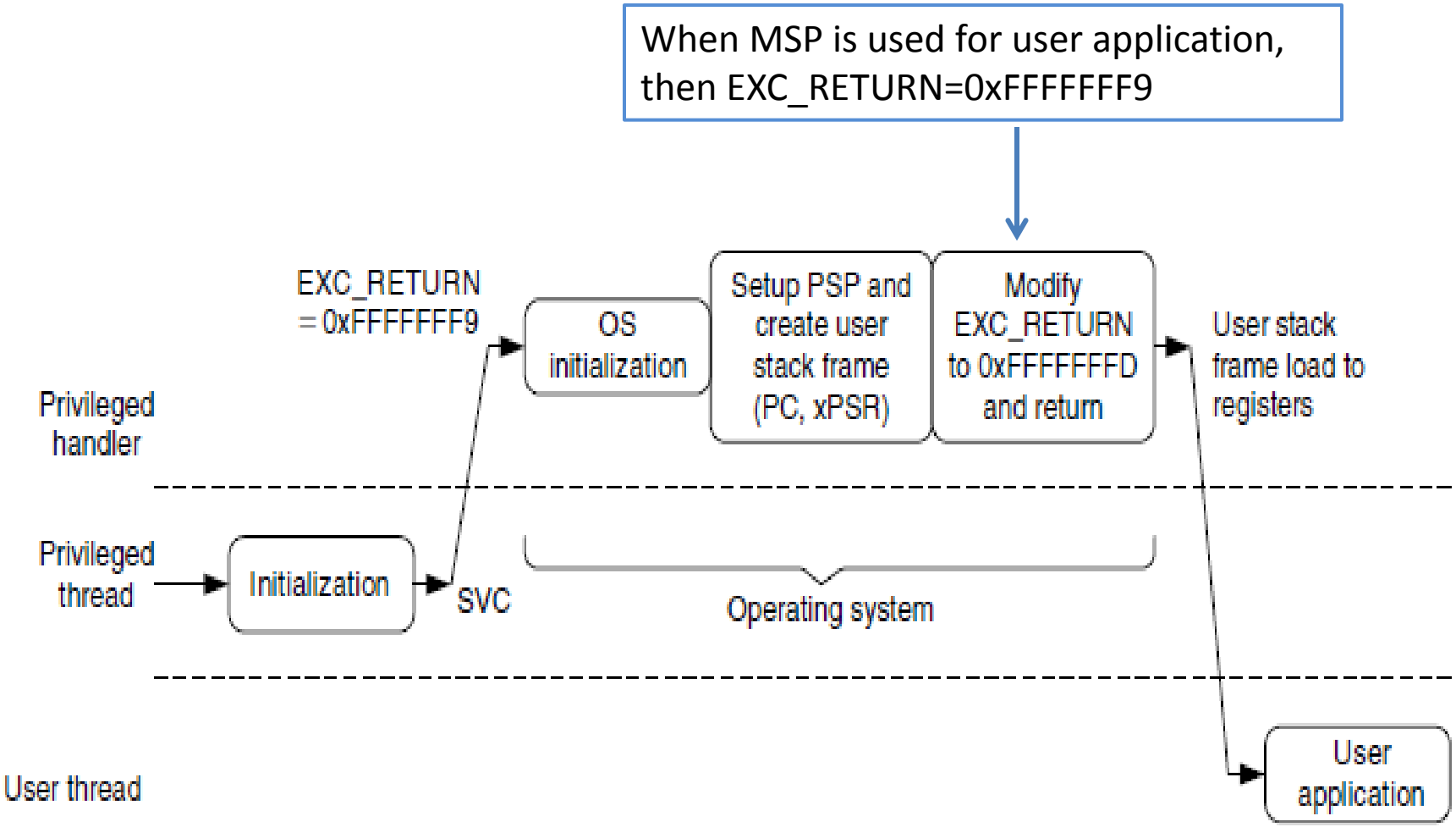
Operation Modes

- Two modes
  - Thread mode
  - Handler mode
- Two privilege levels
  - Privileged level
  - User level



(Image Courtesy of [1])

OS Initialization Mode Switch



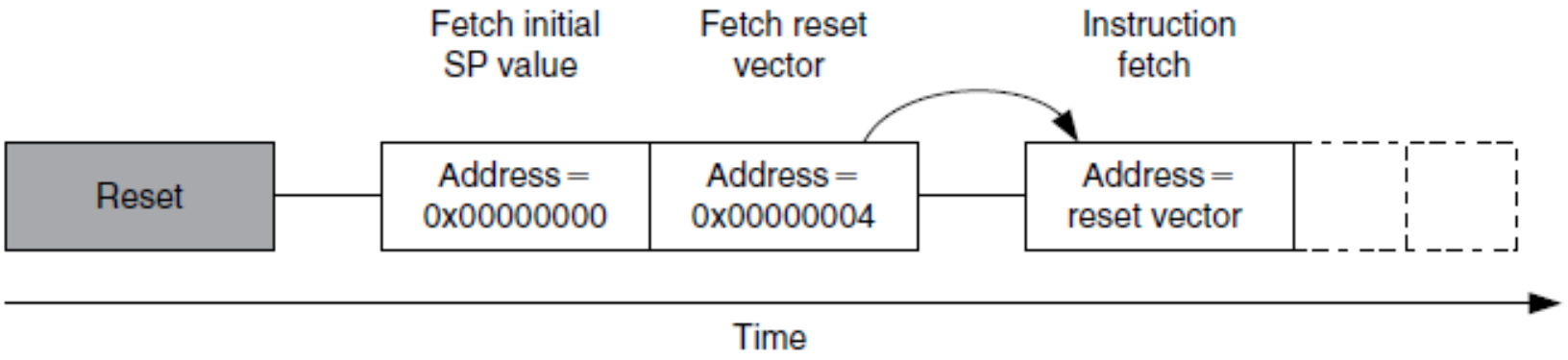
(Image Courtesy of [1])

Exceptions (1)

- NVIC (Nested Vectored Interrupt Controller)
  - System Exceptions
    - Exception Numbers 1 -15
    - SVC call exception number is 11
  - External Interrupts
    - Exception Numbers 16-50
    - Timer0-3 IRQ numbers are 17-20
    - UART0-3 IRQ numbers are 21-24
- Vector table is at 0x0 after reset.
- 32 programmable priorities
- Each vector table entry contains the exception handler's address (i.e. entry point)

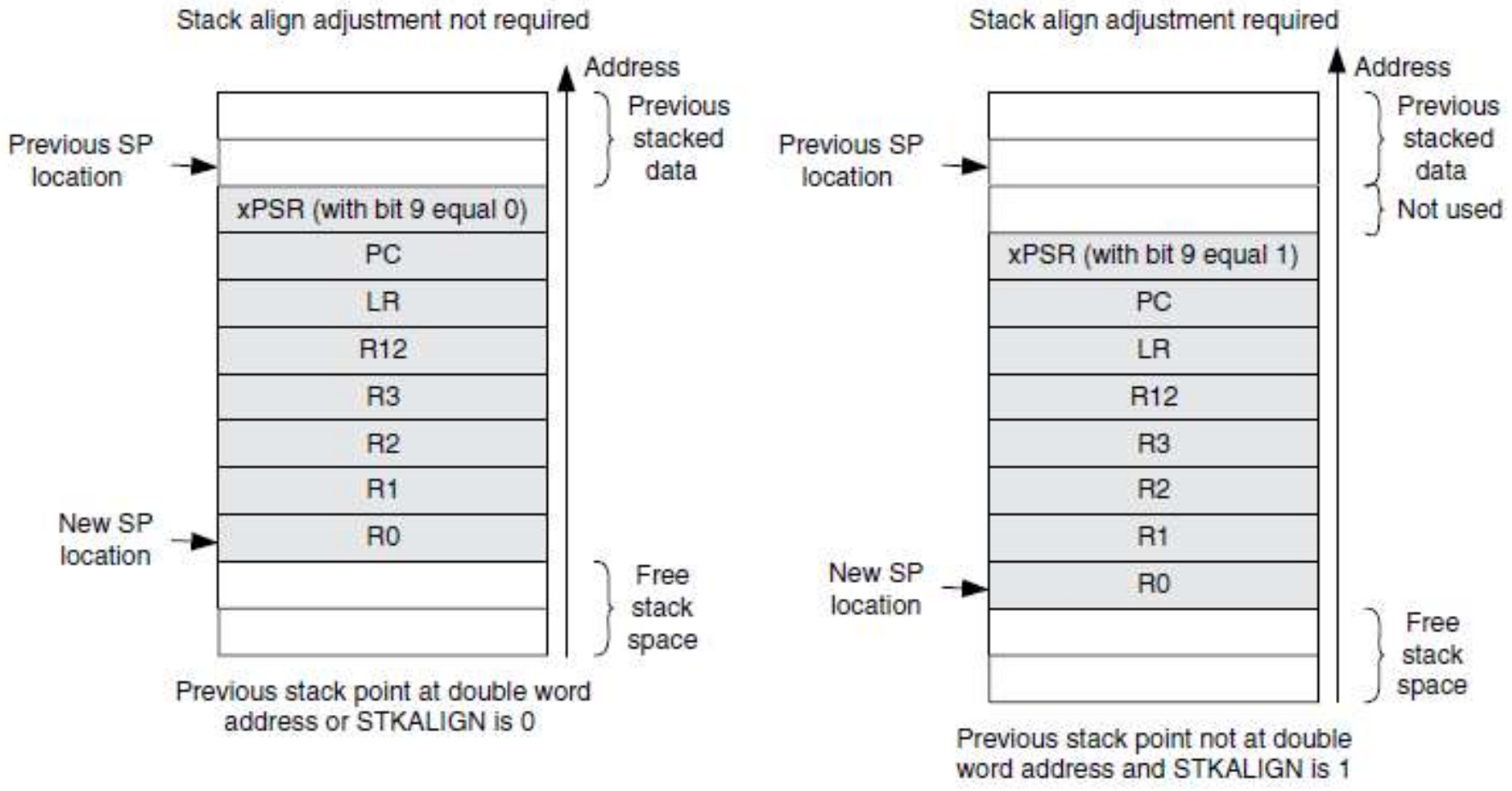
Exceptions (2)

Address	Exception Number	Value (Word Size)
0x0000 0000	-	MSP initial value
0x0000 0004	1	Reset vector (program counter initial value)
0x0000 0008	2	NMI handler starting address
0x0000 000C	3	Hard fault handler starting address
…	…	Other handler starting address



(Image Courtesy of [1])

Exception Stack Frame



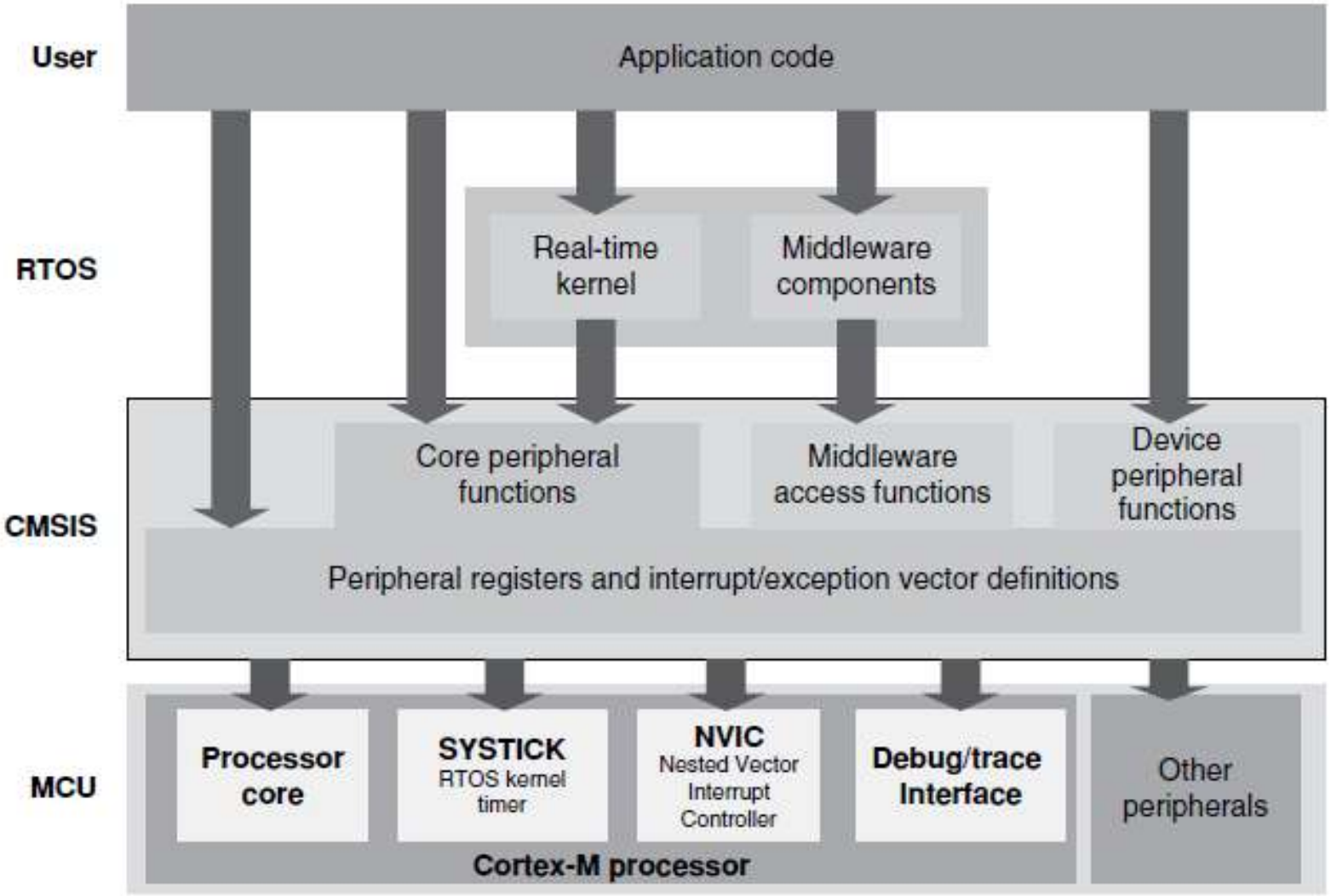
(Image Courtesy of [1])

CORTEX-M3 SOFTWARE DEVELOPMENT

AAPCS (ARM Architecture Procedure Call Standard)

- R0-R3
  - Input parameters Px of a function. R0=P1, R1=P2, R2=P3 and R3=P4
  - **R0** is used for **return value** of a function
- R12, SP, LR and PC
  - R12 is the Intra-Procedure-call scratch register.
- R4-R11
  - Must be preserved by the called function. C compiler generates push and pop assembly instructions to save and restore them automatically.

CMSIS Structure



(Image Courtesy of [1])

CMSIS Structure

- Hardware Abstraction Layer (HAL) for Cortex-M processor registers
  - NVIC, MPU
- Standardized system exception names. For example:

```
void SVC_Handler();
void UART0_IRQHandler();
```

- Standardized method of header file organization
- Common method for system initialization

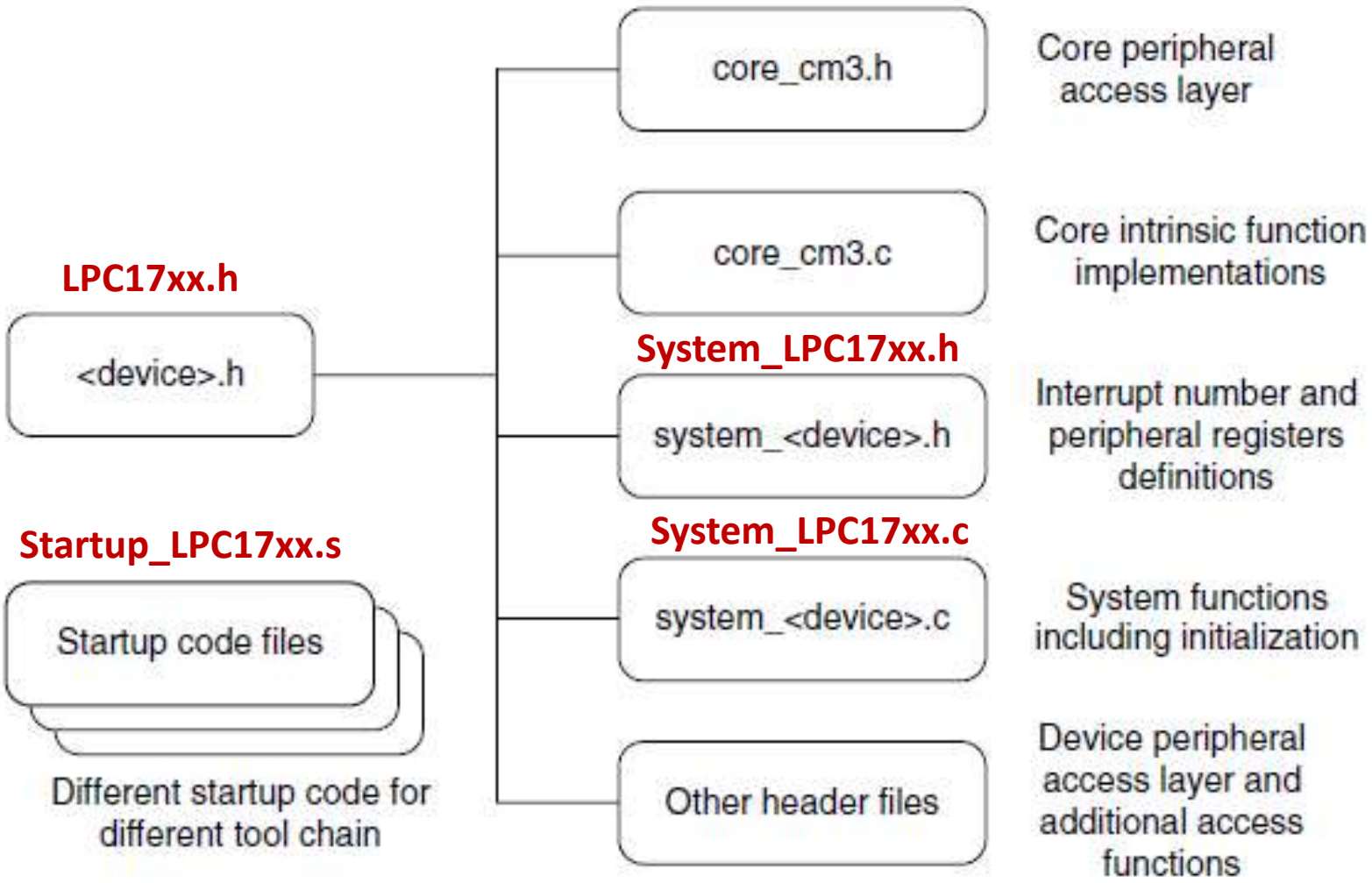
```
SystemInit()
```

- Standardized intrinsic functions. For example:

```
void __disable_irq(void);
void enable_irq(void);
```

- *Common access functions for communication*
- Standardized way for embedded software to determine system clock frequency
  - **`SystemFrequency`** variable is defined in device driver code

CMSIS Files



(Image Courtesy of [1])

External Interrupt Programming

- `IRQn` is defined in <device.h> file (i.e. LPC17xx.h)
- A set of functions
  - `void NVIC_EnableIRQ(IRQn_Type IRQn)`
  - `void NVIC_DisableIRQ(IRQn_Type IRQn)`
  - `void NVIC_SetPriority(IRQn_Type IRQn, int32_t priority)`
  - `uint32_t NVIC_GetPriority(IRQn_Type IRQn)`
  - `void NVIC_SetPendingIRQ(IRQn_Type IRQn)`
  - `void NVIC_ClearPendingIRQ(IRQn_Type IRQn)`
  - `IRQn_Type NVIC_GetPendingIRQ(IRQn_Type IRQn)`

CMSIS Example

```
#include "vendor_device.h"  // For example,
  // lm3s_cmsis.h for LuminaryMicro devices
  // LPC17xx.h for NXP devices
  // stm32f10x.h for ST devices

void main(void) {
  SystemInit();
  …
  NVIC_SetPriority(UART1_IRQn, 0x0);
  NVIC_EnableIRQ(UART1_IRQn);
  …
}
void UART1_IRQHandler {
  ...
}

void SysTick_Handler(void) {
…
}
```

Common name for system initialization code (from CMSIS v1.30, this function is called from startup code)

NVIC setup by core access functions

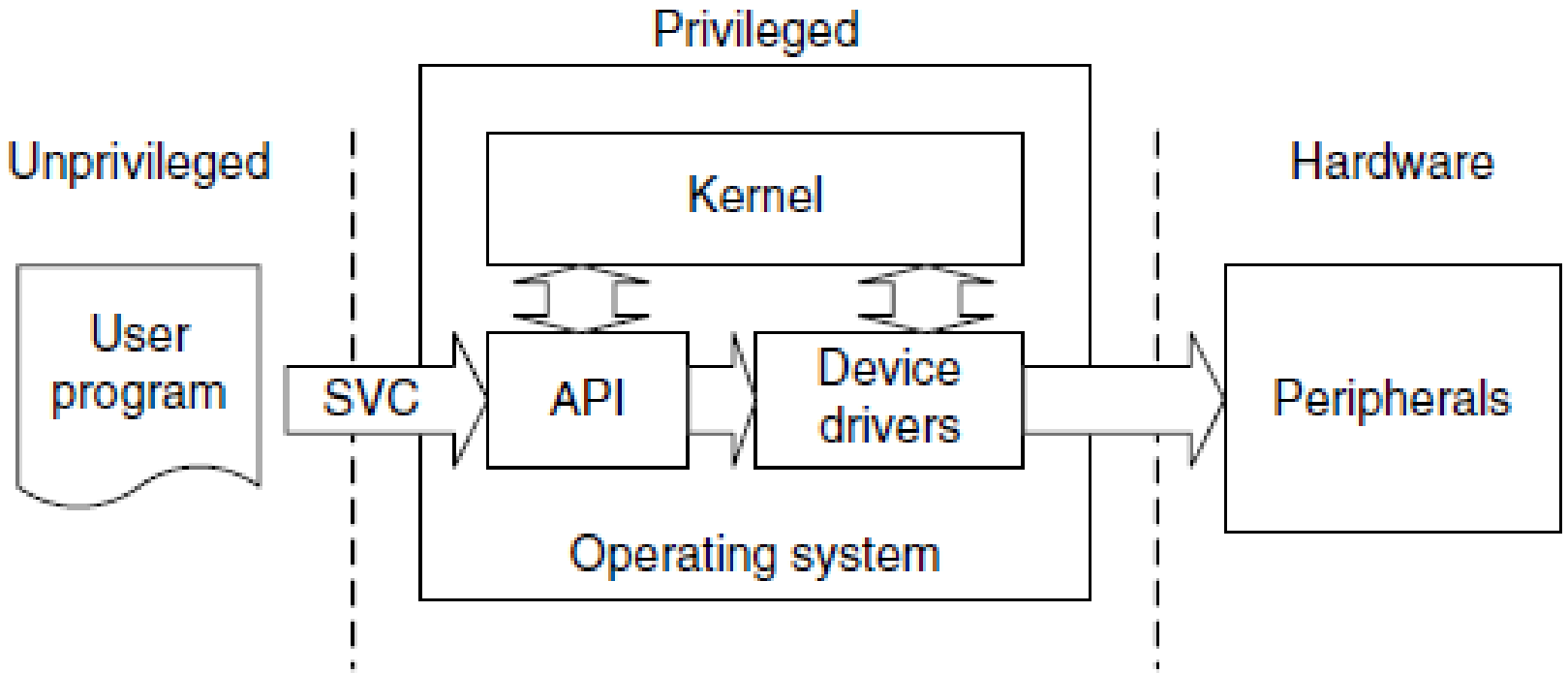
Interrupt numbers defined in system_<device>.h

Peripheral interrupt names are device specific, define in device specific startup code

System exception handler names are common to all Cortex microcontrollers

(Image Courtesy of [1])

SVC as a Gateway for OS Functions



(Image Courtesy of [1])

System calls through SVC in C

User Space **int rls_mem_blk(void *)** rtx.h

```
extern int k_rls_mem_blk(void *);
#define __SVC_0  __svc_indirect(0)
#define rls_mem_blk(blk)
        _rls_mem_blk((U32)k_rls_mem_blk, blk)
extern int _rls_mem_blk (U32 p, void* blk)  __SVC_0
```

```
LDR.W r12, [pc, #offset]
        ;Load k_rls_mem_blk  in r12
SVC 0x00,
```

Generated by the compiler

```
SVC _Handler:  BLX R12
```

HAL.c

Kernel Space **int k_rls_mem_blk(void*)** rtx.c

Hints

- Start with compile time memory pool and then change it later to dynamic memory pool.
- Start with just one ready queue and two simple user processes in your system to implement the context switching between the two processes.
- Once you get two processes working properly under context switching, add more ready queues to different priority levels and add user test process one by one to re-fine your context switching logic.

References

1. Yiu, Joseph, *The Definite Guide to the ARM Cortex-M3*, 2009
2. *RealView® Compilation Tools Version 4.0 Developer Guide*
3. *ARM Software Development Toolkit Version 2.50 Reference Guide*
4. *LPC17xx User's Manual*