Introduction to RTX Project and Keil MCB1700

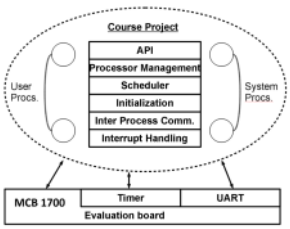
Irene Huang

RTX PROJECT OVERVIEW

RTX Project Introduction

- Keil MCB1700 Cortex-M3 Board
- Design and Implement a small RTX
  - Basic multiprogramming environment
  - 5 Priority queues, preemption
  - Simple memory management
  - Message-based IPC(Interprocess Communication)
  - Basic timing Services
  - System console I/O
  - Debugging support

Functional Overview



Deliverables

Project Parts	Requirements	Submissions	Deadlines
Group Signup	Four members per group	N/A	Jan 16th 4:30pm
RTX P1	Memory management (data structure + APIs) Specified processes in the SPECs as well as few testing processes	Source code + Documentation	Jan. 29th 4:30pm
RTX P2	Simplified version of the RTX	Source code + Documentation	Mar. 04th 4:30pm
RTX P3	Final version of the RTX	Source code + Documentation	Mar. 21st 8:30am
RTX P4	Final project documents + timing code	Source code + Report	Apr. 04th 4:30pm

Recommended
- write documentation to help with report
- Documentation not graded

RTX 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 *memory_block)
```

- Processor Management

```
int release_processor()
```

- Process Priority Management

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

Start with memory

No memory → nullptr

→ process gets locked

Release p
run H
highest
process

return 0 if
Success
not 0 => Error

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.

When no process ready to run
- lowest priority

rocessor:

e next

priority

Set priority:

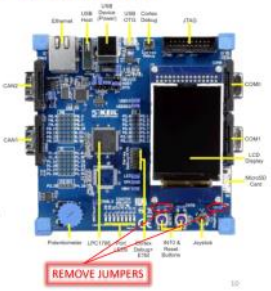
Increase priority to allow a process to interrupt current process or lower so it is delayed

Scheduler then determines correct process to run

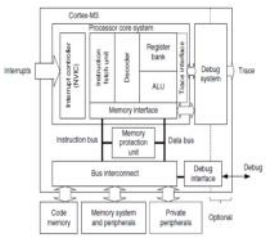
MCB1700 HARDWARE

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

32 bit

Harvard Architecture
- Separate data and memory bus
- instruction and data buses share the same memory space (a unified memory system)

Cortex-M3 Registers

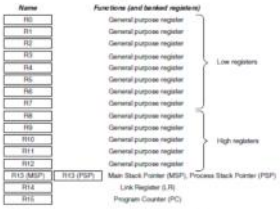
- General Purpose Registers (R0-R15)
  - Low registers (R0-R7)
    - 16-bit Thumb instructions and 32-bit Thumb-2 instructions
  - High registers (R8-R12)
    - All Thumb-2 instructions
  - Stack Pointer (R13)
    - **MSP:** Privileged, default after reset, OS kernel, exception handler
    - **PSP:** Uesr-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	☆

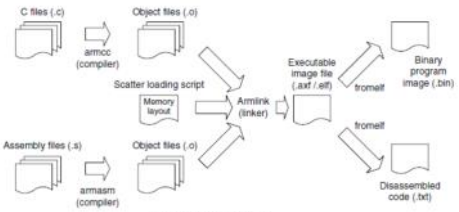
(Table Courtesy of [1])

LPC1768 Memory Map

☆ IMPORTANT
for some reason

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

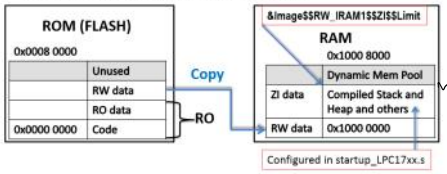
Example Flow Using ARM Development Tools



(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)



Compiler knows symbol at end

End Address of the Image

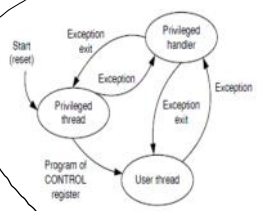
- Linker defined symbol Image$$RW_IRAM1$$ZI$$Limit

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

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

Operation Modes

Not in interrupt handler

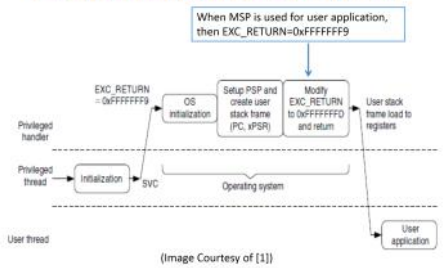
- Two modes
  - Thread mode
  - Handler mode
- Two privilege levels
  - Privileged level
  - User level



(Image Courtesy of [1])

Can only be in Handler mode or Privileged level

OS Initialization Mode Switch



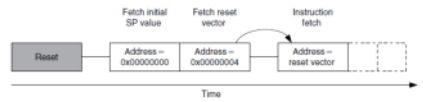
(Image Courtesy of [1])

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)

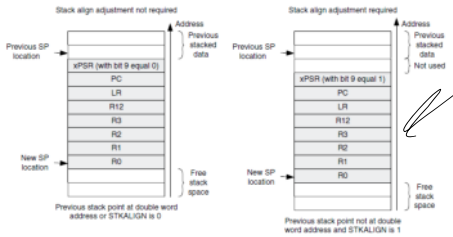
Exceptions (2)

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])

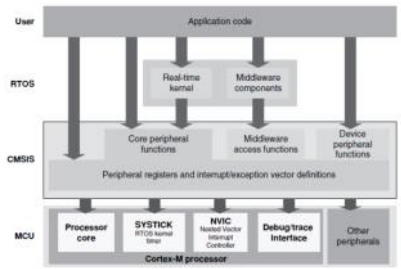
ARM stores these registers automatically

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);
```

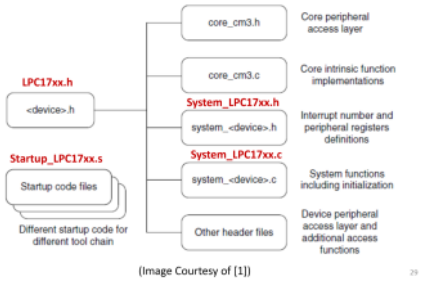
- *Common access functions for communication*
- Standardized way for embedded software to determine system clock frequency
  - `SystemFrequency` variable is defined in device driver code

27

CMSIS Example

```
__asm UART0_IRQHandler(void)
{
    IMPORT c_UART0_IRQHandler

    ; push registers

    BL __cpp(c_UART0_IRQHandler)

    ; pop registers
}

void c_UART0_IRQHandler(void)
{
    /* C function that does the actual Interrupt Handling
}
```

CMSIS Files

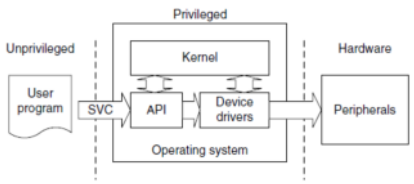


(Image Courtesy of [1])

29

SVC as a Gateway for OS Functions

mechanism to write system call



(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
```

Generated by the compiler

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

SVC _Handler: BLX R12 — HAL.c

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

31

SVC Handler in HAL.c

```
__asm void SVC_Handler(void)
{
    MRS R0, MSP
    ; push registers
    ; extract SVC number from stacked PC

    LDM R0, {R0-R3, R12}
    BLX R12

    ; code to handle context switching
    ; pop registers
    MVN LR, , #:NOT:0xFFFFFFF9
          ; set EXC_RETURN value, Thread mode, MSP
    BX LR
}
```

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*