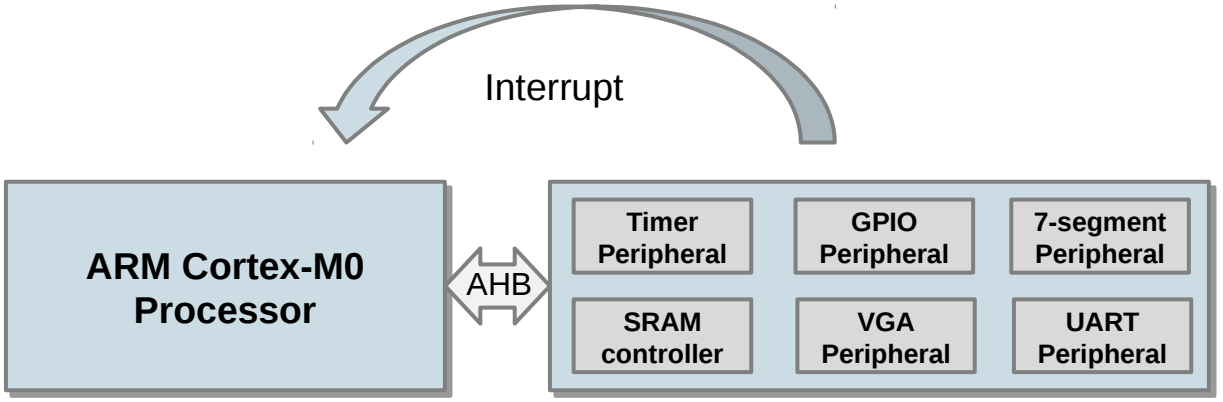
Design and Implementation Interrupt Mechanism

Module Overview

- Study processor interruption;
- Design and implement of an interrupt mechanism which responds to interrupts from timer and UART;
- Program interrupt handler using assembly (low-level);
- Lab practice.

Hardware design



Module Syllabus

- Polling Vs. Interrupt
- Exception and Interruption
- Interrupt Preemption
- ARMv6-M Exception Model
- Cortex-M0 Interrupt Controller
- NVIC Configuration
- Hardware Implementation of Interrupt
- Interrupts Handling in Software
- Lab Practice

Interruption in General

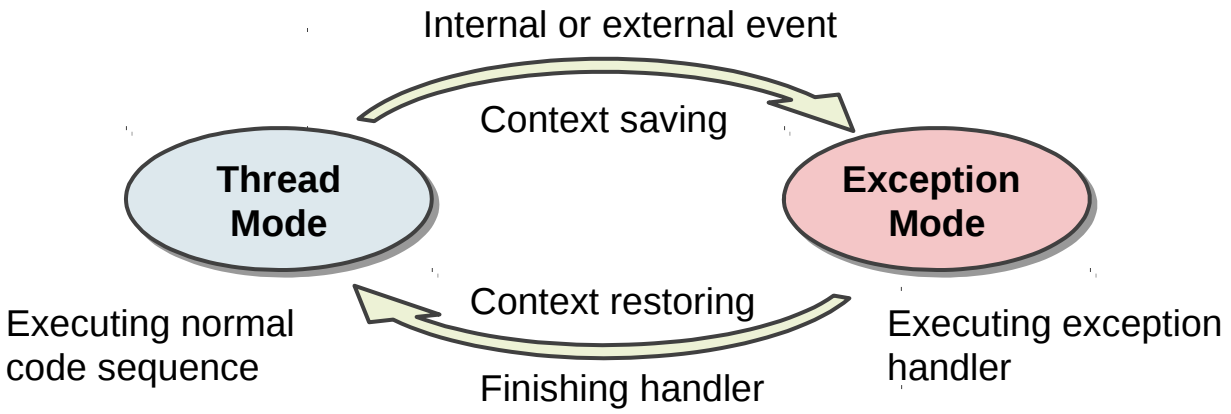
Polling Vs. Interrupt

How to display a number counting-up every second:

- Polling - use software to check if the hardware timer reaches a value;
  - Slow - need to explicitly check to see if the timer has reached a value
  - Wasteful of CPU time - the faster a response we need, the more often we need to check
  - Scales badly - difficult to build system with many activities which can respond quickly. Response time depends on all other processing.
- Interrupt - the timer generates an interrupt every second, and the processor runs specific code (interrupt service routine - ISR) in response;
  - Efficient - code runs only when necessary
  - Fast - hardware mechanism
  - Scales well
  - ISR response time doesn't depend on most other processing.
  - Code modules can be developed independently

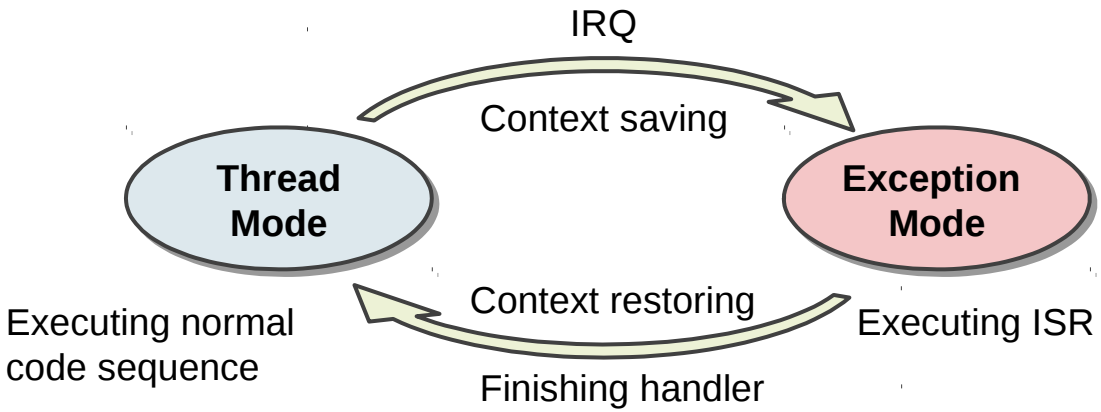
Exception and Interruption

- Exception
  - Events (internal or external) that cause the program flow to exit the current program thread, and execute a piece of code associated with the event;
  - Events can be either internal or external.
  - Exception Handler is a piece of software code that is executed in the exception mode;



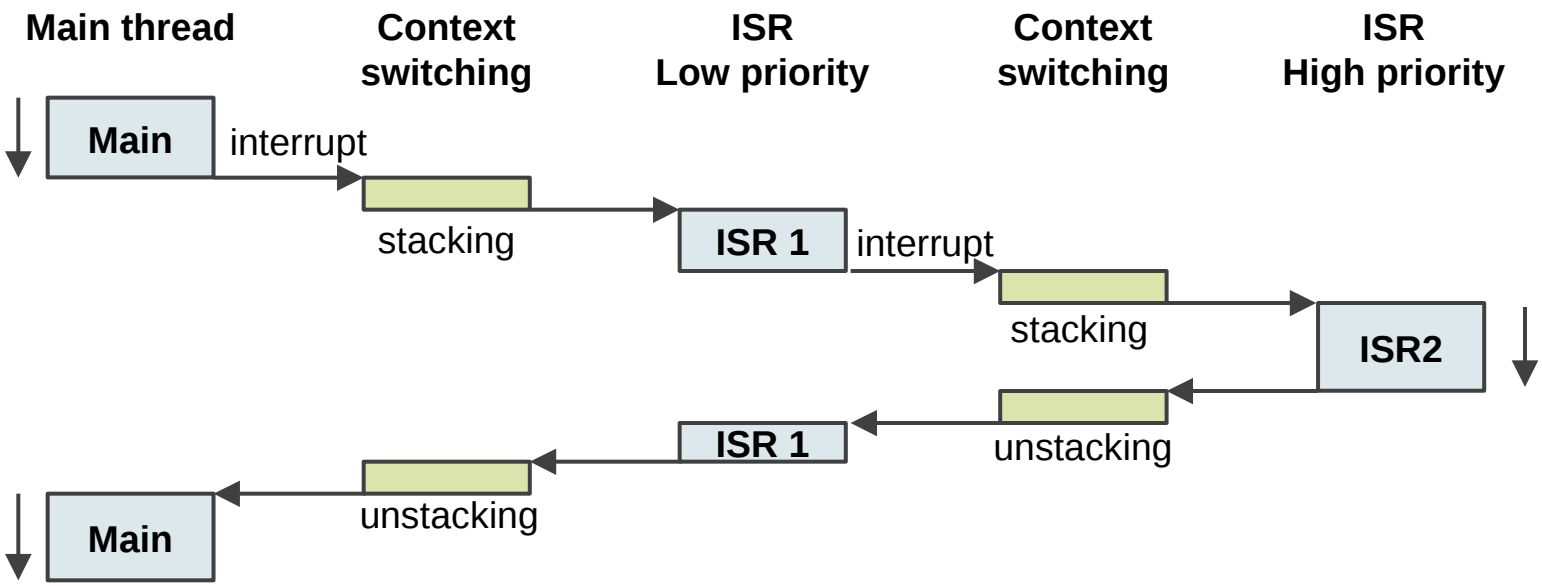
Exception and Interruption

- Interrupt
  - Refers to the exception caused by external events;
  - External event is also called interrupt request (IRQ).
  - The exception handler here is also called interrupt service routine (ISR);



Interrupt Preemption

- The exceptions (or interrupts) are commonly divided into multiple levels of priorities;
- A higher priority exception can be triggered and serviced during a lower priority exception;
- Commonly known as a nested exception, or interrupt preemption.



ARMv6-M Exception Model

ARMv6-M Exception Model

Exception number	Exception type	Priority
1	Reset	-3 (highest)
2	NMI	-2
3	HardFault	-1
4-10	Reserved	
11	SVCall	Programmable
12-13	Reserved	
14	PendSV	Programmable
15	SysTick, optional	Programmable
16 + N	External interrupt 0-31	Programmable

ARMv6-M Exception Model

- Reset
  - The ARMv6-M profile supports two levels of reset;
    - Power-on reset resets the processor, SCS and debug logic;
    - Local reset resets the processor and SCS except for debug-related resources;
  - The Reset exception is permanently enabled with a fixed priority of -3;
- Non Maskable Interrupt (NMI)
  - The second highest priority exception, cannot be disabled;
  - Useful for safety critical systems like industrial control or automotive;
  - Can be used for power failure handling or a watchdog;

ARMv6-M Exception Model

- HardFault
  - HardFault is the generic fault that exists for all classes of fault that cannot be handled by any of the other exception mechanisms.
  - For example, trying to execute an unknown optcode, or a fault on a bus interface or memory system;
- SVCall (SuperVisor Call)
  - SVCall exception takes place when SVC instruction is executed;
  - Is usually used in an operating system control;
- PendSV (Pendable Service Call)
  - Similar to SVCall, but does not immediately take place;
  - Starts only when high-priority tasks are completed;

ARMv6-M Exception Model

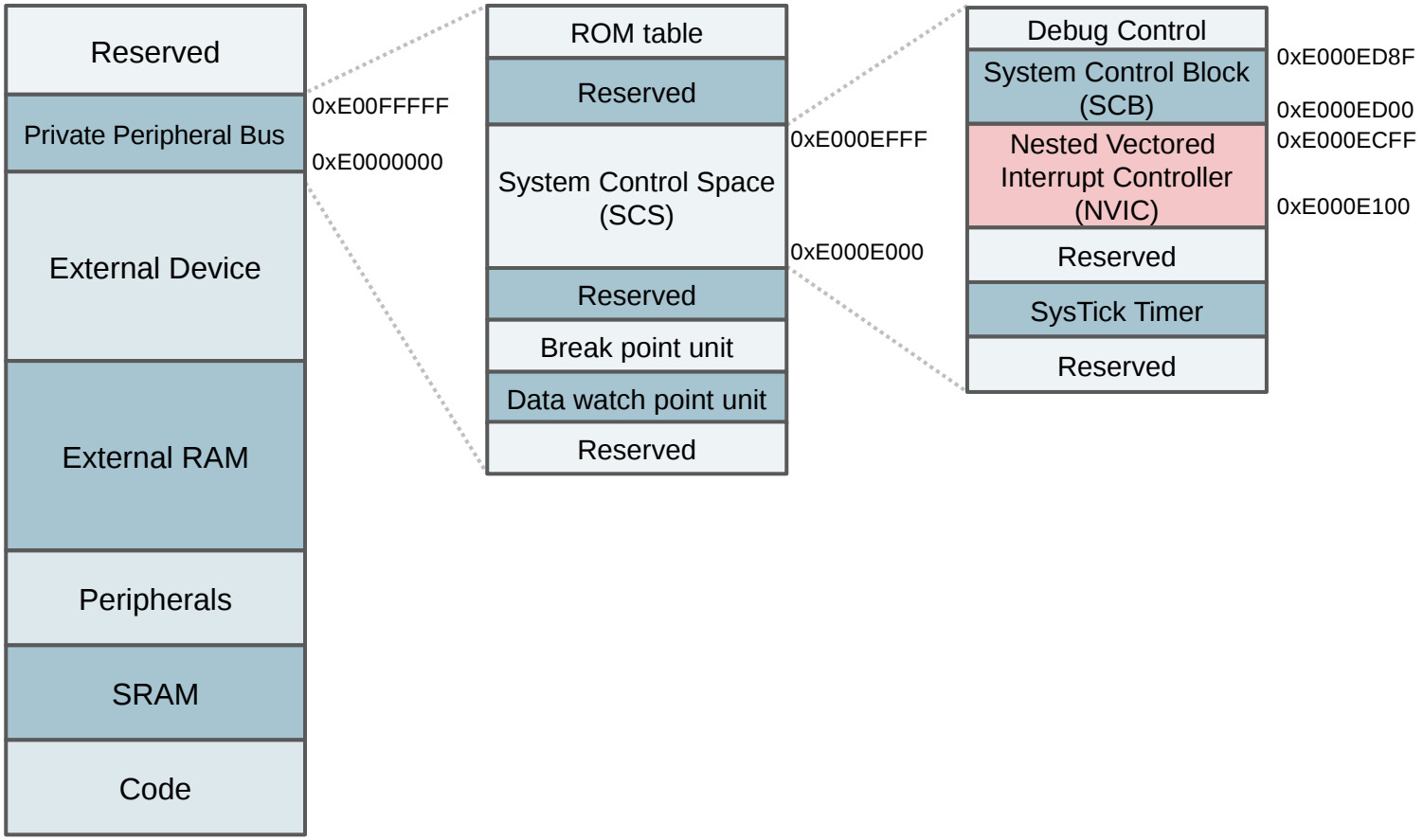
- SysTick
  - The System Tick timer is another feature for an OS application;
  - Exception starts when an regular interrupt is generated from a timer;
- External interrupts
  - Supports up to 32 external interrupts, which can be connected from on-chip peripherals;
  - Needs to be enabled before being used;
  - Programmable priorities.

Cortex-M0 Interrupt Controller

- Nested Vectored Interrupt Controller (NVIC)
  - An integrated part of the Cortex-M0 processor;
  - Supports up to 32 IRQ inputs and a non-maskable interrupt (NMI) inputs;
  - Flexible interrupt management
    - Enable/ disable interrupt;
    - Pending control;
    - Priority configuration;
  - Hardware nested interrupt support;
  - Vectored exception entry;
  - Interrupt masking;
  - Can be easily accessed using C or assembly language.
  - Location: Private Peripheral Bus ␣ System Control Space ␣NVIC

Cortex-M0 Interrupt Controller

- Memory map of Nested Vectored Interrupt Controller (NVIC)

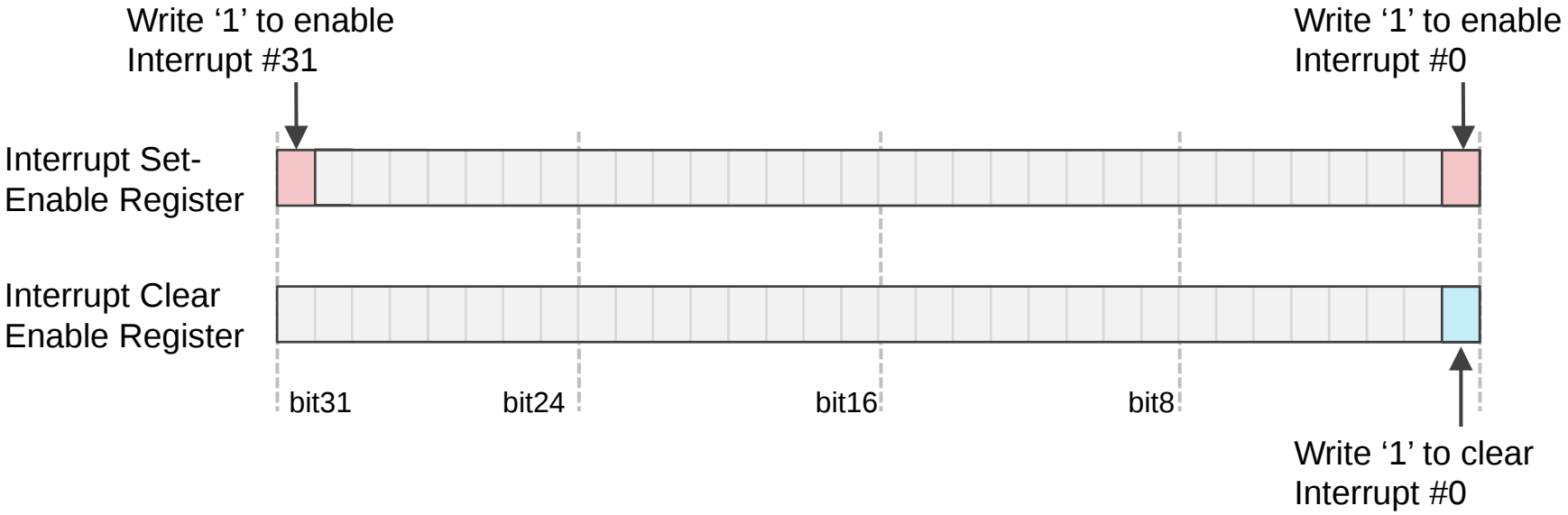


NVIC Registers

Address	Register
0xE000E100	**Interrupt Set-Enable Register**
0xE000E104 — 0xE000E17F	Reserved
0xE000E180	**Interrupt Clear Enable Register**
0xE000E184 — 0xE000E1FF	Reserved
0xE000E200	**Interrupt Set-Pending Register**
0xE000E204 — 0xE000E27F	Reserved
0xE000E280	**Interrupt Clear-Pending Register**
0xE000E300 — 0xE000E3FC	Reserved
0xE000E400 — 0xE000E41C	**Interrupt Priority Registers**
0xE000E420 — 0xE000E43C	Reserved

NVIC Registers

- Interrupt Set-Enable Register
  - Write ‘1’ to enable one or more interrupts;
  - Write ‘0’ has no effect;
- Interrupt Clear Enable Register
  - Write ‘1’ to one or more interrupts.
  - Write ‘0’ has no effect;

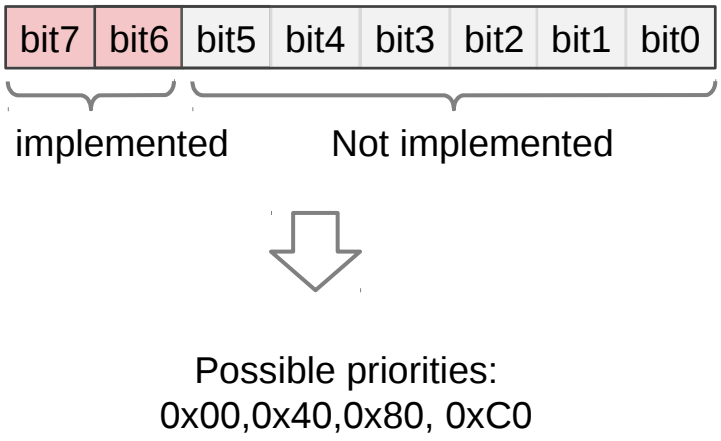


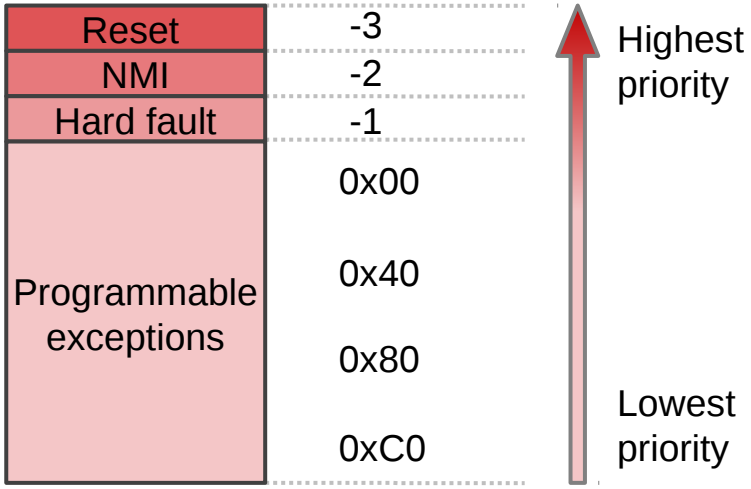
NVIC Registers

- Why use separated register address
  - Compared with the “read-modify-write” process the benefit of using separated address includes:
    - Reduces the steps needed for enabling/ disabling an interrupt, resulting in smaller code and less execution time;
    - Prevents the race condition, e.g. the main thread is accessing a register by “read-modify-write” process, and it is interrupted between its “read” and “write” operation. If the ISR again modifies the same register that is currently being accessed by the main thread, a conflict will occur.
- Interrupt pending and clear pending
  - An interrupt goes into pending status if it cannot be processed immediately, e.g. a lower priority interrupt will be pended if a higher interrupt is being processed.

NVIC Registers

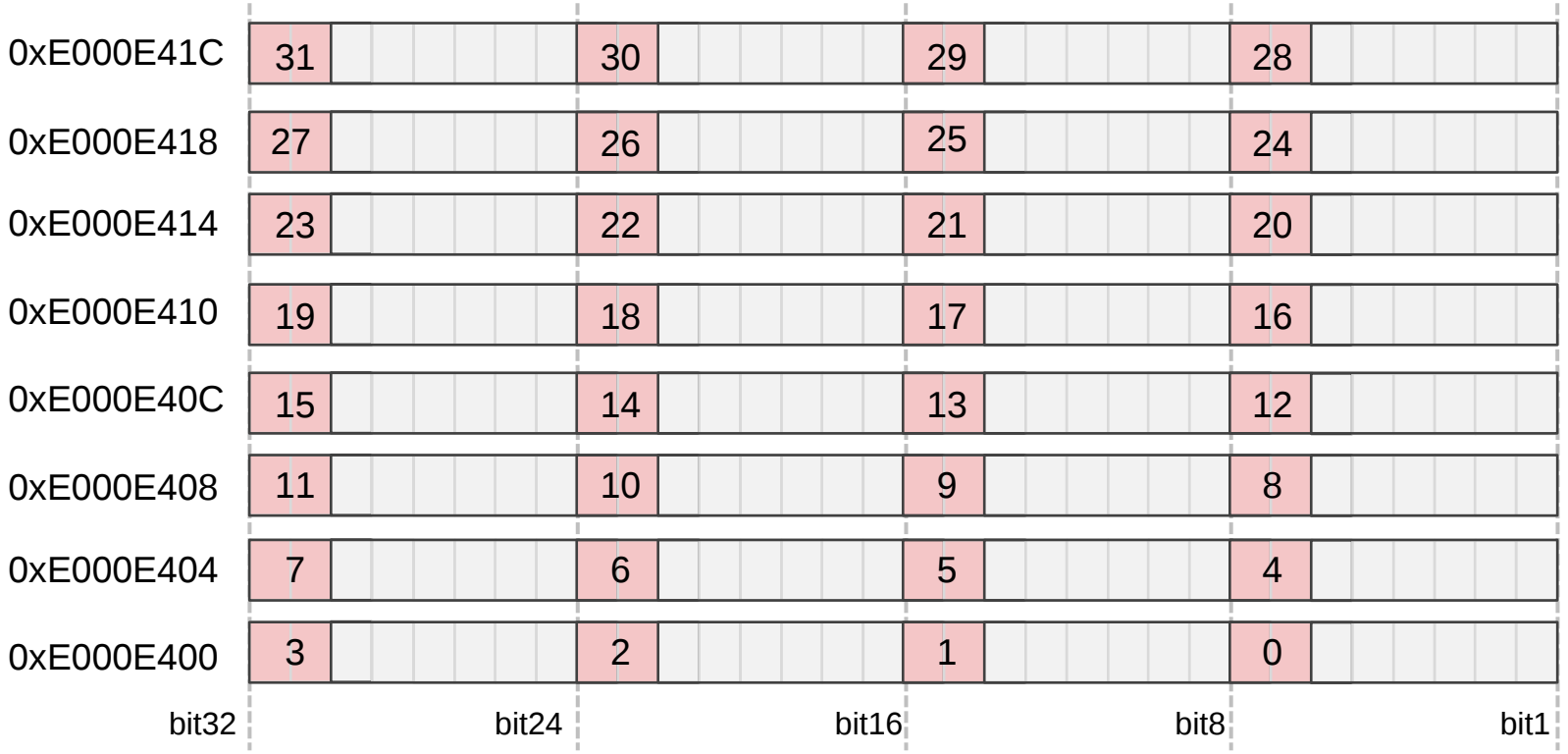
- Cortex-M0 Priority
  - The priority level configuration registers are 8 bits wide, but only two bits are implemented in Cortex-M0;
  - Since only the two MSBs are used, four levels of priority can be represented: 0x00,0x40,0x80 and 0xC0.





NVIC Registers

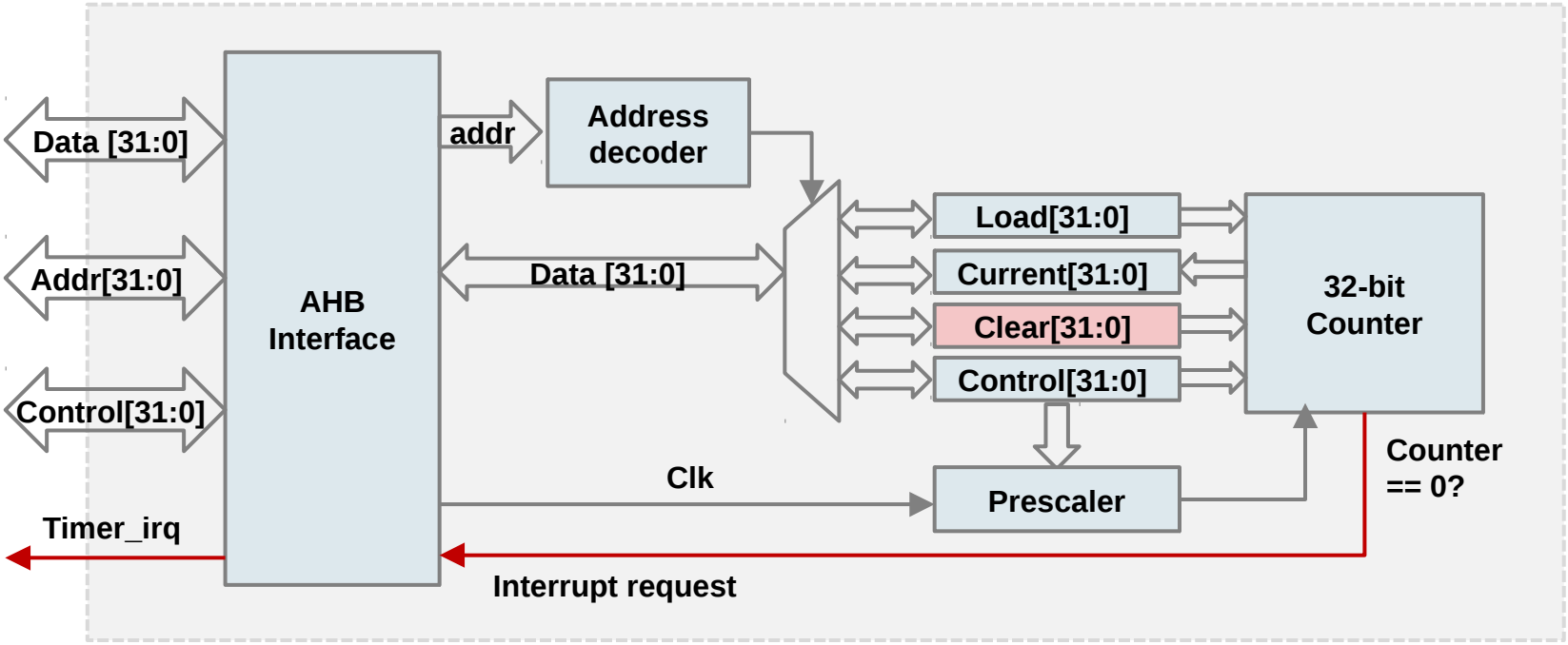
- Interrupt Priority Registers
  - Use eight 32-bit registers to sets interrupt priorities for the 32 interrupts;
  - Each register contains the priority for 4 interrupts; and each interrupt priority is implemented using 2 bits.



Implementation of Interrupt Mechanism

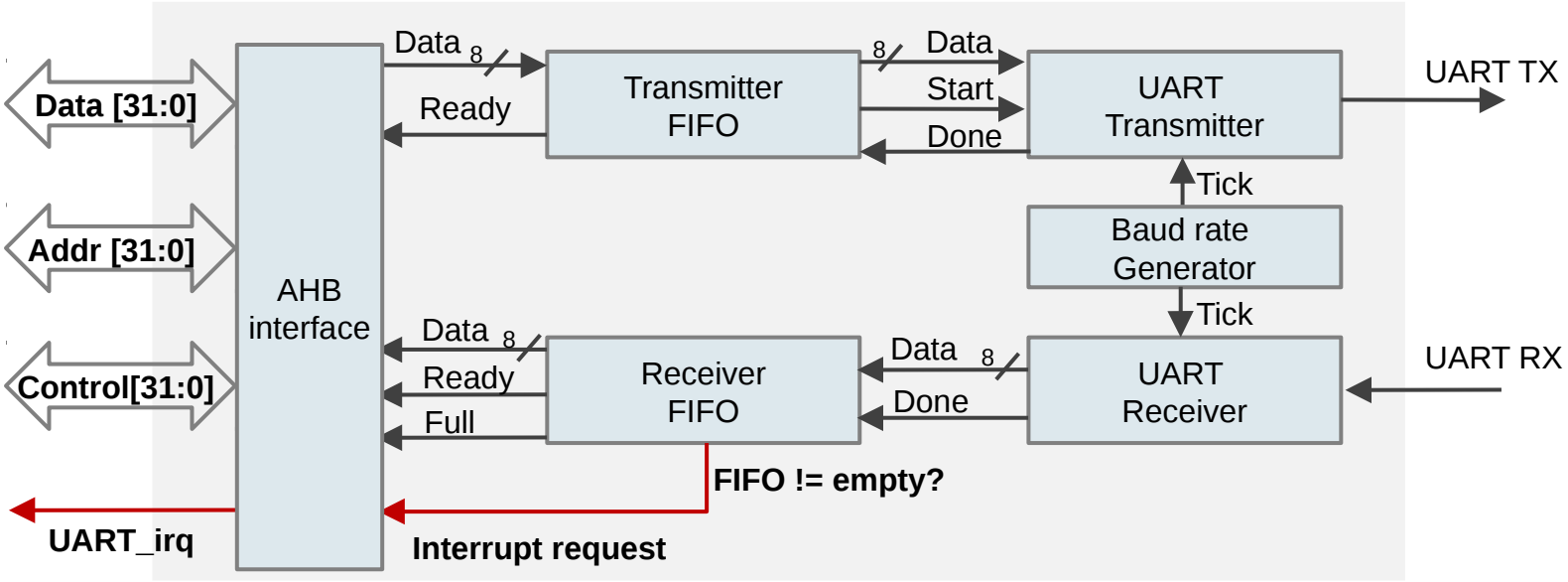
Interrupt Implementation for Timer

- Implement the interrupt mechanism for the AHB timer peripheral, for example:
  - An interrupt is generated every time the counter reaches zero.
  - A clear register needs to be added, which is used to clear the interrupt request once the processor finishes its ISR.



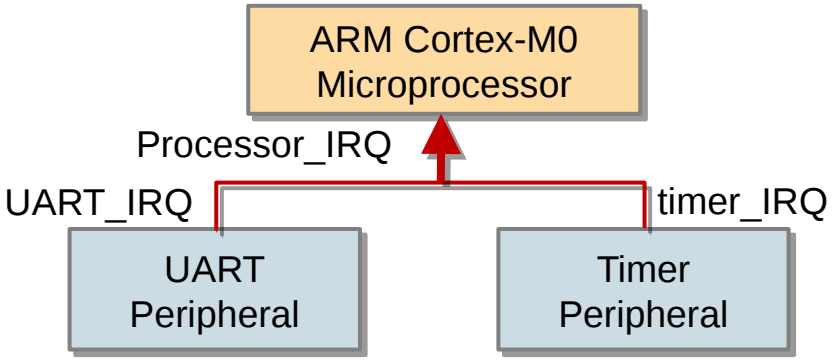
Interrupt Implementation for UART

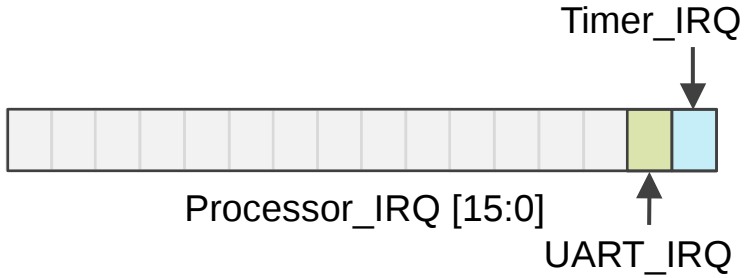
- Implement the interrupt mechanism for the AHB UART peripheral;
  - For example, the interrupt can be generated if the receiver FIFO is not empty.



Connect Interrupts to Processor

- Connect the interrupts from peripherals to the Cortex-M0 microprocessor;
- Note that in this set of teaching materials, we use a simplified version of Cortex-M0 (Cortex-M0 DesignStart), which only supports 16 external interrupts.





Enable Interrupts in Software

1. Configure NVIC:
   - Set interrupt priority registers, for example:

```
LDR  R0, =0xE000E400      ; Address of priority0 register
LDR  R1, [R0]             ; Get priority0 register
MOVS      R2, #0xFF       ; Byte mask
BICS R1, R1, R2           ; R1= R1 AND (Not (0x000000FF))
MOVS      R2, #0x40       ; Priority level
ORRS      R1, R1, R2      ; Update the value of priority register
STR  R1, [R0]             ; Write back the priority register
```

   - Set interrupt enable register, for example:

```
LDR  R0, =0xE000E100      ; NVIC Enable register
MOVS      R1, #0x1        ; Interrupt #0
STR  R1, [R0]             ; Enable interrupt #0
```

2. Make sure PRIMASK register is zero:
   - Set to one will block all the interrupts apart from nonmaskable interrupt (NMI) and the hard fault exception, e.g.

```
MOVS      R0, #0x0        ;
MSR PRIMASK,  R0   ; Clear PRIMASK register
```

Entering an Exception Handler

1. Finish current instruction (except for lengthy instructions);
2. Look up the interrupt vector, and branch to the entry address of the exception handler;
3. Push context onto current stack (MSP or PSP)
4. Load PC with address of exception handler;
5. Load LR with EXC_RETURN code;
6. Load IPSR with exception number;
7. Start executing code of exception handler;

- Usually 16 cycles from exception request to execution of first instruction in the handler;
- The interrupt latency is the time delayed before entering an interrupt, which is an overhead that should be minimised.

Exiting an Exception Handler

1. Clear the interrupt request of the peripheral, for example:
   - The timer can have a clear register, which is used to clear its interrupt request;
   - Alternatively, the interrupt request can be automatically cleared by the peripheral itself, under a certain condition, e.g. UART can clear its interrupt request after all data has been read out from it receiver FIFO.
2. Context restoring, pop up the registers from the stack;
3. Update IPSR;
4. Load PC with the return address;
5. Continue executing code of the previous program;

Lab Practice

Lab Practice

- Step1- Hardware implementation
  - Design and implement the interrupt mechanism (e.g. for the timer peripheral and the UART peripheral) in hardware using Verilog;
  - Connect the interrupts to the Cortex-M0 processor through AHB bus;
- Step2- Software programming
  - Test the interrupt mechanism using Cortex-M0 processor programed in assembler language;
- Step3- System Demonstration
  - Receive characters from the UART, and then print them to the UART terminal using UART interrupt handler;
  - Implement a counting-up counter, which is incremented every second, and the number will be displayed on the monitor.

Useful Resources

- Reference1
  - Nexys3 Reference Manual:

    http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf
- Reference2
  - Book: “The Definitive Guide to the ARM Cortex-M0” by Joseph Yiu, ISBN-10: 0123854776, ISBN-13: 978-0123854773, March 11, 2011