COEN-4720 Embedded Systems

Lecture 5
Interrupts

Cris Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

Outline

- Introduction
- NVIC and Interrupt Control
- Program Image and Start-up Sequence

How Does it Work?

- Something tells the processor core (which is running the main execution flow) there is an interrupt/exception
- Core transfers control to code that needs to be executed to address the interrupt
- Said code "returns" to the main (old) program

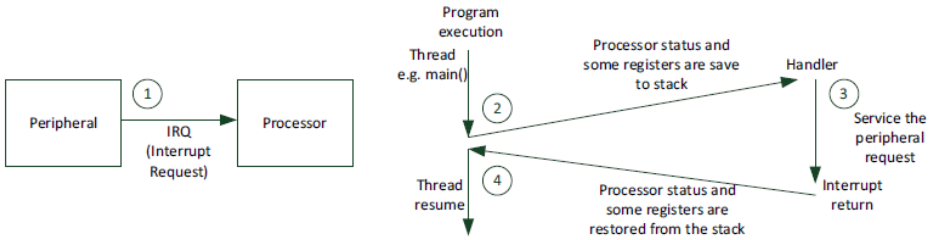


Figure 8.1
Interrupt handling concept.

3

Some Questions

- How do you figure out where to branch/jump to?
  - If you know number the possible interrupt cases, and an interrupt comes in, you can just branch to a location, using that number as an offset
- How to you ensure that you can get back to where you started?
  - Store return address to stack or dedicated register
- Don't we have a pipeline? What about partially executed instructions?
  - Complex architectures
- What if we get an interrupt while we are already "processing" an interrupt?
  - Nested interrupts: handle directly, ignore, prioritize
- What if we are in a "critical section?"
  - Prioritization

4

Interrupts

- An interrupt is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution
- This hardware event is called a trigger and it breaks the execution flow of the main thread of the program
- The event causes the CPU to stop executing the current program and begin executing a special piece of code called an interrupt handler or interrupt service routine (ISR)
- Typically, the ISR does some work and then resumes the interrupted program

5

Interrupts

- The hardware event can either be:

  1) A **busy-to-ready transition** in an external I/O device. Caused by the external world
  - Peripheral/device, e.g., UART input/output device
  - Reset button, Timer expires, Power failure, System error
  - Names: exception, interrupt, **external interrupt**

  2) An **internal event**
  - Bus fault, memory fault
  - A periodic timer
  - Div. by zero, illegal/unsupported instruction
  - Names: exception, trap, **system exception**
- When the hardware needs service, signified by a busy to ready state transition, it will request an interrupt by setting its trigger flag

6

Cortex-M Interrupts

- Exceptions:
  - System exceptions: numbered 1 to 15
  - External interrupt inputs: numbered from 16 up
- Different numbers of external interrupt inputs (from 1 to 32-240) and different numbers of priority levels
- Value of current running exception is indicated by:
  - Special register Interrupt Program Status Register (IPSR)

7

List of System Exceptions

Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fault, memory access fault, and other address/memory related.
6	Usage Fault[c]	Configurable	Usage fault, such as Undefined instruction executed or illegal state transition attempt.
7	SecureFault[d]	Configurable	SecureFault is available when the CPU runs in *Secure state*. It is triggered by the various security checks that are performed. For example, when jumping from Non-secure code to an address in Secure code that is not marked as a valid entry point.
8-10	-	-	RESERVED
11	SVCall	Configurable	System service call with SVC instruction.
12	Debug Monitor[c]	Configurable	Debug monitor – for software based debug.
13	-	-	RESERVED
14	PendSV	Configurable	Pending request for system service.
15	SysTick	Configurable	System tick timer has fired.

8

List of External Interrupts

Table 8.1: List of exceptions in the Cortex-M0 and Cortex-M0+ processors

Exception number	Exception type	Priority	Descriptions
1	Reset	-3 (Highest)	Reset
2	NMI	-2	Non-Maskable Interrupt
3	HardFault	-1	Fault handling exception
4–10	Reserved	NA	–
11	SVCall	Programmable	Supervisor call via SVC instruction
12–13	Reserved	NA	–
14	PendSV	Programmable	Pendable request for system service
15	SysTick	Programmable	System Tick Timer
16	Interrupt #0	Programmable	External Interrupt #0
17	Interrupt #1	Programmable	External Interrupt #1
…	…	…	…
47	Interrupt #31	Programmable	External Interrupt #31

9

Outline

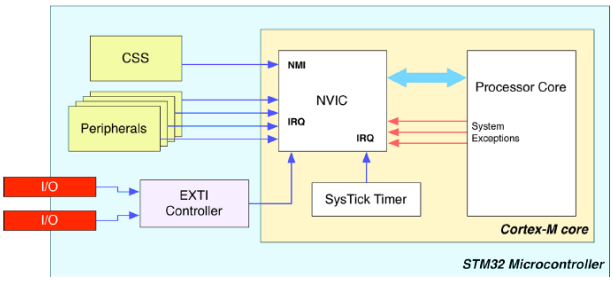
- Introduction
- NVIC and Interrupt Control
- Program Image and Start-up Sequence

10

10

NVIC

- Interrupts on Cortex-M processors are controlled by Nested Vectored Interrupt Controller (NVIC)
- A unit dedicated to exceptions management
- Each exception has an associated 32-bit vector that points to the memory location where the ISR that handles the exception is located
- Vectors are stored in ROM at the beginning of the memory
- Relationship between NVIC, Cortex-M processor, and peripherals:



11

NVIC

- A programmable unit that allows software to manage interrupts and exceptions
- It has a number of memory mapped registers for the following:
  - Defining the priority levels of each interrupts and some of the system exceptions
  - Enabling or disabling of each of the interrupts
  - Enabling the software to access the pending status of each interrupt, including the capability to trigger interrupts by setting pending status in software

12

12

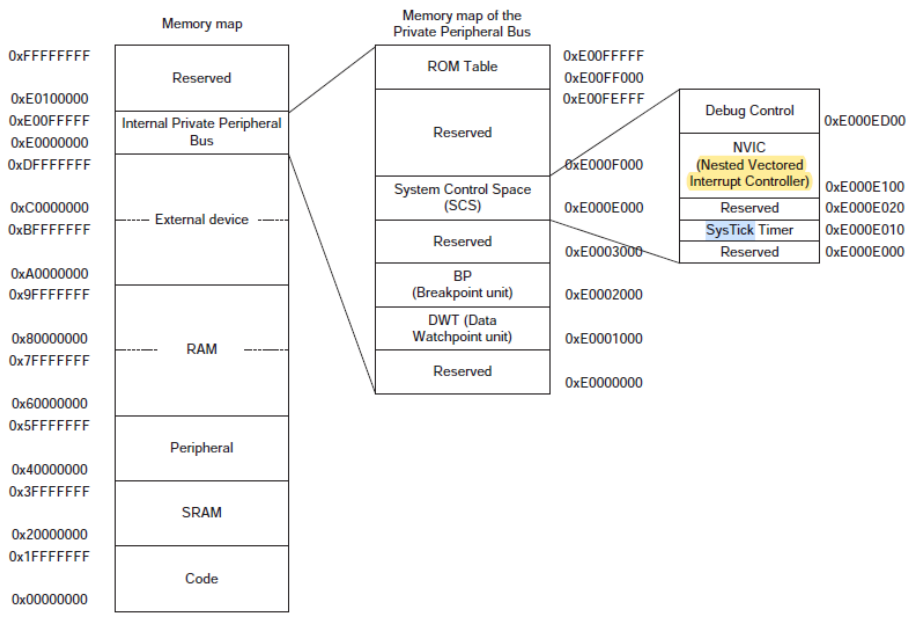


Figure 7.2
Architecturally defined memory map of the Cortex®-M0/M0+ processor.

13

13

Interrupt Programming

- PRIMASK - Interrupt Mask Special Register
- A 1-bit wide interrupt mask register
- When set, it disables (i.e., blocks) all interrupts apart from the Non-Maskable Interrupt (NMI) and the HardFault exception
- When reset, it enables interrupts; means to allow interrupts at this time

14

Interrupt Programming

- To activate an "interrupt source" we need to set its priority and enable that source in NVIC

Activate = Set priority + Enable source in NVIC

- This activation is in addition to the "enable" step discussed earlier

15

Priority Levels

- Cortex-M0 and Cortex-M0+ processors support three fixed highest priority levels for three of the system exceptions (Reset, NMI, and HardFault) and four programmable levels for all other exceptions including interrupts.
- Priority levels:
  - 0x00 (high priority), 0x40, 0x80, and 0xC0 (low priority)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented		Not implemented, read as zero					

Figure 8.3
A Priority Level Register with 2 bits implemented.

16

16

Priority Levels

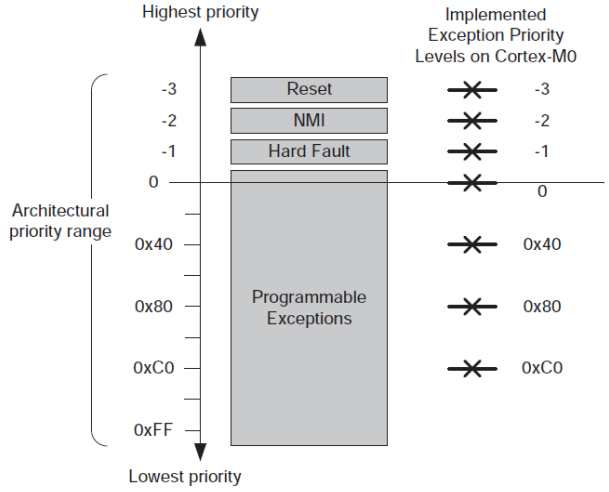


Figure 8.4
Available priority levels in the Cortex®-M0 and Cortex-M0+ Processors.

17

17

Basic Interrupt Configuration

- Each external interrupt has several registers associated with it:
  - Enable and clear enable registers
  - Set-pending and clear-pending registers
  - Active status
  - Priority level
- In addition, several other registers can also affect the interrupt processing:
  - Exception-masking registers (PRIMASK, FAULTMASK, and BASEPRI)
  - Vector Table Offset register
  - Software Trigger Interrupt register
  - Priority Group

18

18

Interrupt Enable and Clear Enable

- The Interrupt Enable register is programmed via two addresses
  - To set the enable bit, we write to the SETENA register address
  - To clear the enable bit, you need to write to the CLRENA register address

Table 8.4: Interrupt Enable Set and Clear Register

Address	Name	Type	Reset value	Descriptions
0xE000E100	SETENA	R/W	0x00000000	Set enable for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status
0xE000E180	CLRENA	R/W	0x00000000	Clear enable for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status

19

Interrupt Pending and Clear Pending

- If an interrupt takes place but cannot be executed immediately (e.g., if another higher-priority interrupt handler is running), it will be pended
- The interrupt pending status can be accessed through the Interrupt Set Pending (SETPEND) and Interrupt Clear Pending (CLRPEND) registers

Table 8.5: Interrupt Pending Set and Clear Register

Address	Name	Type	Reset value	Descriptions
0xE000E200	SETPEND	R/W	0x00000000	Set pending for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status
0xE000E280	CLRPEND	R/W	0x00000000	Clear pending for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status

20

Priority Levels

- Each external interrupt has an associated priority level register.
- Each of them is 2 bit wide, occupying the two MSBs of the Interrupt Priority Level Registers.
- Each Interrupt Priority Level Register occupies 1 byte (8 bits:

Bit	31 30	24	23 22	16	15 14	8	7 6	0
0xE000E41C	31		30		29		28	
0xE000E418	27		26		25		24	
0xE000E414	23		22		21		20	
0xE000E410	19		18		17		16	
0xE000E40C	15		14		13		12	
0xE000E408	11		10		9		8	
0xE000E404	7		6		5		4	
0xE000E400	IRQ 3		IRQ 2		IRQ 1		IRQ 0	

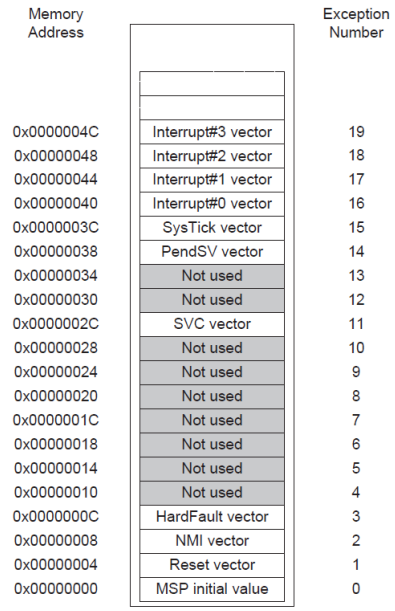
Figure 8.11
Interrupt Priority Level Registers for each interrupt.

21

Vector Table

- The interrupt handling in the Cortex-M Processor is vectored, which means the processor's hardware automatically determines which interrupt or exception to service
- When an exception takes place and is being handled by the Cortex-M processor, it will need to locate the starting address of the exception handler
- This information is stored in the vector table
  - 32-bit vectors that point to memory locations where ISRs are located
  - Stored in ROM at the beginning of the memory

22



Memory Address		Exception Number
0x0000004C	Interrupt#3 vector	19
0x00000048	Interrupt#2 vector	18
0x00000044	Interrupt#1 vector	17
0x00000040	Interrupt#0 vector	16
0x0000003C	SysTick vector	15
0x00000038	PendSV vector	14
0x00000034	Not used	13
0x00000030	Not used	12
0x0000002C	SVC vector	11
0x00000028	Not used	10
0x00000024	Not used	9
0x00000020	Not used	8
0x0000001C	Not used	7
0x00000018	Not used	6
0x00000014	Not used	5
0x00000010	Not used	4
0x0000000C	HardFault vector	3
0x00000008	NMI vector	2
0x00000004	Reset vector	1
0x00000000	MSP initial value	0

Figure 8.5
Vector table.

The SYSTICK Timer

- Often a hardware timer is used:
  - To generate interrupts so that the OS can carry out task management
  - As an alarm timer, for timing measurement, etc.
- Cortex-M processors include a simple timer: 24-bit down counter
- Interrupts each 10 milliseconds
- The SYSTICK Timer is integrated with the NVIC and can be used to generate a SYSTICK exception (exception type #15)
- SYSTICK Timer is controlled by four registers

The SYSTICK Timer

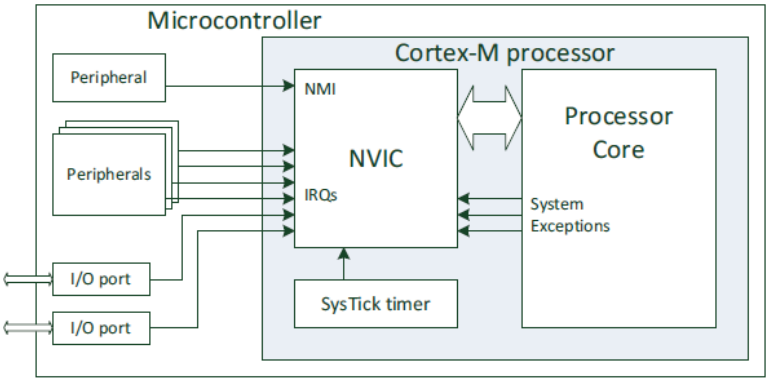


Figure 8.2
The NVIC in the Cortex®-M0 and Cortex-M0+ processors can deal with up to 32 IRQ inputs, an NMI, and a number of system exceptions.

5

25

SYSTICK Timer Control and Status Registers

Table 10.1: SysTick register names in CMSIS

Register	CMSIS Name	Details	Address
SysTick Control and Status Register	**SysTick->CTRL**	Table 10.2	0xE000E010
SysTick Reload Value Register	**SysTick->LOAD**	Table 10.3	0xE000E014
SysTick Current Value Register	**SysTick->VAL**	Table 10.4	0xE000E018
SysTick Calibration Value Register	**SysTick->CALIB**	Table 10.5	0xE000E01C

Table 10.2: SysTick control and status register (0xE000E010)

EBits	Field	Type	Reset value	Descriptions
31:17	Reserved	-	-	Reserved
16	COUNTFLAG	RO	0	Set to 1 when the SysTick timer reaches zero. Clear to 0 by reading of this register.
15:3	Reserved	-	-	Reserved
2	CLKSOURCE	R/W	0/1	Value of 1 indicates that the core clock is used for the SysTick timer. Otherwise a reference clock frequency (depending on MCU design) would be used.
1	TICKINT	R/W	0	SysTick interrupt enable. When this bit is set, the SysTick exception is generated when the SysTick timer count down to 0.
0	ENABLE	R/W	0	When set to 1 the SysTick timer is enabled. Otherwise the counting is disabled.

26

Simplified Procedure for Working with an Interrupt

1) Power up peripheral, if not powered by default
2) Configure clock for peripheral, if the case and necessary
3) Possibly configure other peripheral parameters
4) Enable the interrupt
5) Define or edit the ISR to include what is wanted to be done during servicing of the interrupt

27

27

Interrupt Service Routine (ISR)

- When an interrupt/exception takes place, a number of things happen:
  1. Stacking (automatic pushing of eight registers' contents to stack)
     - PC, PSR (processor status register), R0–R3, R12, and LR (link register)
  2. Vector fetch (reading the exception handler starting address from the vector table)
  3. Exception vector starts to execute. On the entry of the exception handler, a number of registers are updated:
     - Stack pointer (SP) to new location
     - IPSR (low part of PSR) with new exception number
     - Program counter (PC) to vector handler
     - Link register (LR) to special value EXC_RETURN
- Several other registers get updated
- Latency: as short as 12 cycles
- At the end of the exception handler, an exception exit (a.k.a. interrupt return in some processors) is required to restore the system status so that the interrupted program can resume normal execution

28

Example 1 - Toggle LD2 with User Button

- Use interrupts to toggle the LD2 LED every time we press the user-programmable button, which is connected to the PC13 pin.
- We enable the interrupt of the EXTI line associated with the Px13 pins, that is EXTI4_15_IRQn.
- We do that because EXTI lines 4 to 15 share the same IRQ inside the NVIC (and hence are serviced by the same ISR).
- This can be seen inside file:
  - Drivers/CMSIS/Device/ST/STM32XXxx/Include/stm32l053xx.h
- Please also see Table 55 and Figure 30 from the MCU Reference Manual to double check this fact!

29

Configure and Enable EXTI Interrupt

- Done inside **MX_GPIO_Init(void)**
  - Declared and defined inside main.c
- Called inside main()

```
static void MX_GPIO_Init(void)
{
   …
   // Configure GPIO pin : PC13
   GPIO_InitStruct.Pin = GPIO_PIN_13;
   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
   …
   // Enable interrupt
   HAL_NVIC_EnableIRQ(EXTI4_15_IRQn);
}
```

30

ISR associated with IRQ for EXTI4_15_IRQn line

- Declared inside:
  - Core/Startup/startup_ stm32l053r8tx.s
- Defined by us, inside main.c

```
void EXTI4_15_IRQHandler(void)
{
   if(__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_13) != RESET) {
   __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
   HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
   }
}
```

31

31

Example 2 - Toggle LD2 with User Button

- Same as Example 1, but, implemented differently
- Using the callback functions approach
- This mechanism is used by almost all IRQ handler routines inside ST HAL.

```
void EXTI4_15_IRQHandler(void)
{
   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
}


void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
   // this is where user code should go, inside the callback function
   // rather than in EXTI4_15_IRQHandler(); also, as a rule, we should
   // minimize the amount of code inside ISRs; put as little code inside
   // ISRs; ok, to put more code inside main();
   if (GPIO_Pin == GPIO_PIN_13) {
       HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
   }
}
```

32

32

Example 3 – TIM6 Timer to Blink LD2

- Use basic timer TIM6 in interrupt mode to blink the green LED once per second

Configure and Enable TIM6 Interrupt

- Done inside **MX_TIM6_Init(void)**
  - Declared and defined inside main.c
- Called inside main()

```
static void MX_TIM6_Init(void)
{
    __HAL_RCC_TIM6_CLK_ENABLE();
    // configuration of Prescaler and Period are v. important!
    htim6.Instance = TIM6;
    htim6.Init.Prescaler = 15999;
    htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim6.Init.Period = 499;
    htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
    {
        Error_Handler();

    // start the TIM Base generation in interrupt mode;
    HAL_TIM_Base_Start_IT(&htim6);

    // set priority 0 and enable this source of interrupts with the NVIC;
    HAL_NVIC_SetPriority(TIM6_DAC_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM6_DAC_IRQn);
}
```

ISR for TIM6 Interrupt: TIM6_DAC_IRQHandler

- Declared inside: Core/Startup/startup_stm32l053r8tx.s
  - `.word TIM6_DAC_IRQHandler /* TIM6 and DAC */`
- "Exported" inside: Core/Inc/stm32l0xx_it.h
  - `void TIM6_DAC_IRQHandler(void);`
- Defined inside: Core/Src/stm32l0xx_it.c
  - `void TIM6_DAC_IRQHandler(void)`

```
extern TIM_HandleTypeDef htim6;
…
void TIM6_DAC_IRQHandler(void)
{
   HAL_TIM_IRQHandler(&htim6);
}
```

35

HAL_TIM_IRQHandler()

- Declared inside: Drivers/STM32L0xx_HAL_Driver/Inc/stm32l0xx_hal_tim.h
  - `void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim);`
- Defined inside: Drivers/STM32L0xx_HAL_Driver/Src/stm32l0xx_hal_tim.c

```
void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim)
{
    …
    /* TIM Update event */
    if (__HAL_TIM_GET_FLAG(htim, TIM_FLAG_UPDATE) != RESET)
    {
        if (__HAL_TIM_GET_IT_SOURCE(htim, TIM_IT_UPDATE) != RESET)
        {
            __HAL_TIM_CLEAR_IT(htim, TIM_IT_UPDATE);
            #if (USE_HAL_TIM_REGISTER_CALLBACKS == 1)
                htim->PeriodElapsedCallback(htim);
            #else
                HAL_TIM_PeriodElapsedCallback(htim);
            #endif /* USE_HAL_TIM_REGISTER_CALLBACKS */
        }
    }
    …
}
```

36

HAL_TIM_PeriodElapsedCallback()

- Declared inside: Drivers/STM32L0xx_HAL_Driver/Inc/stm32l0xx_hal_tim.h
  - `void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);`
- Defined inside: main.c

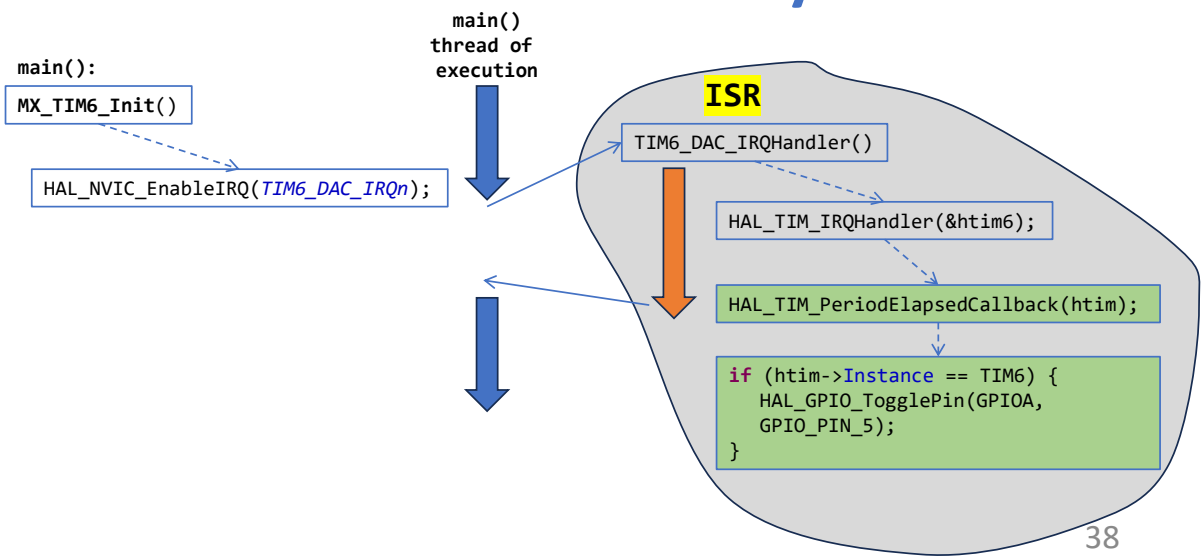
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // this routine should be called every 0.5 seconds based on how we configured TIM6 timer;
    // first toggle the pin driving the green LED:
    if (htim->Instance == TIM6) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
        // NOTE: the above could also be: HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    }

    // then, increment my global counter;
    if (tim6_counter >= 20) { // 20 x 0.5 sec = 10 sec
        message_was_sent = false; // set false so user-message not printed multiple times inside main()
        tim6_counter = 0; // reset global counter;
    }
    tim6_counter++;
}
```

37

37

TIM6 Summary



38

38

Outline

- Introduction
- NVIC and Interrupt Control
- Program Image and Start-up Sequence

39

Program Image

- The program image for the Cortex-M0/M0+ processor is located from address 0x00000000.

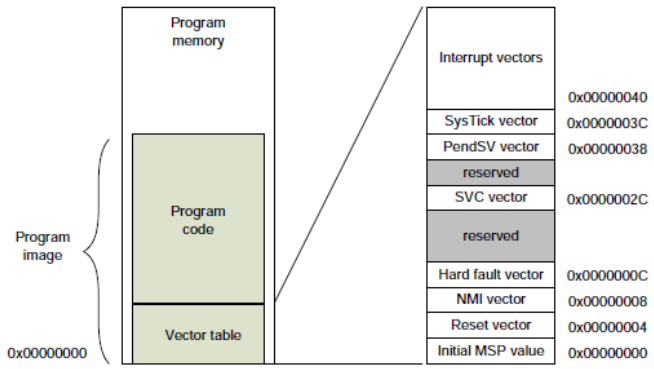


Figure 4.15
Vector table in a program image.

40

Reset Sequence

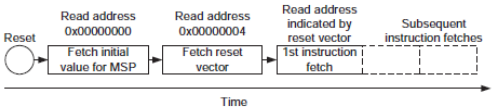


Figure 4.16
Reset sequence.

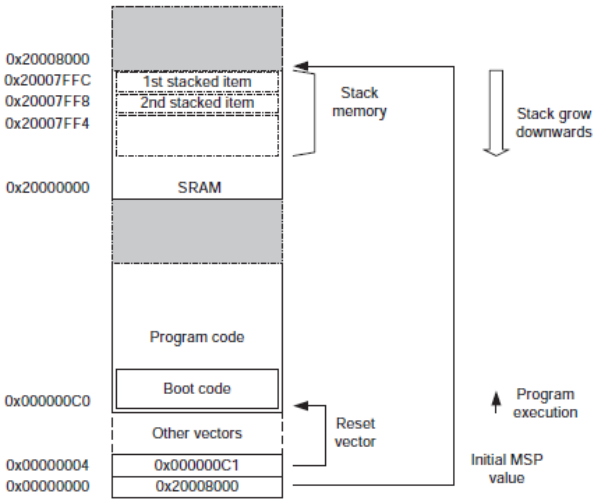


Figure 4.17
Example of MSP and PC initialization.

41

Credits and references

- [1] Carmine Noviello, Mastering STM32, Second Edition, 2022. (Required, Book 1). Available to purchase online.
- [2] Joseph Yiu, The Definitive Guide to ARM Cortex-M0 and Cortex-M0+ Processors, 2nd Ed., 2015. (Book 2). Can be found online.

42