Carnegie Mellon University Qatar

15-348: Embedded Systems Lecture 9

Fall 2022

Ryan Riley
(based on original slides by Saquib Razak)

1

1

Where are we?

- Recently
  - SysTick timer
  - High-level interfacing to hardware
- Today
  - Interrupts

2

2

Reading

- Intro to Interrupts:
  267 – 275
- Interrupts with the timer:
  279 - 283

3

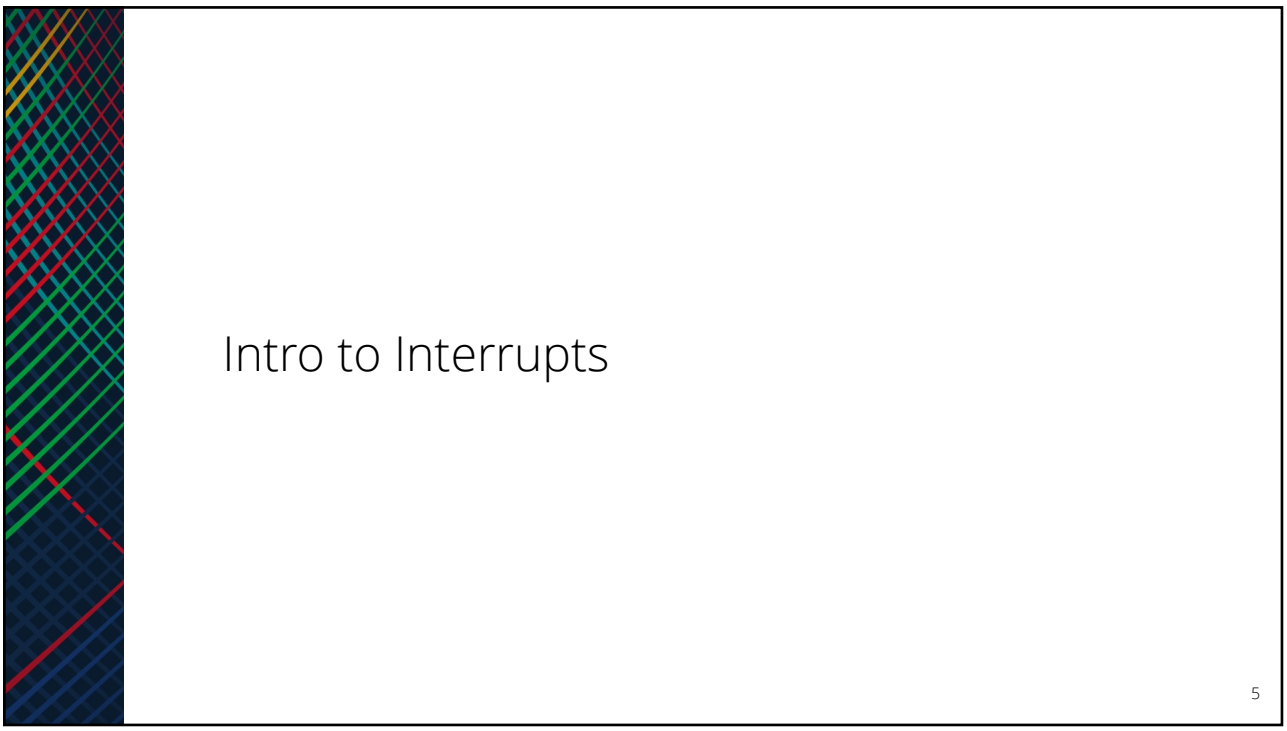
3

Quick CCS vs Keil Note

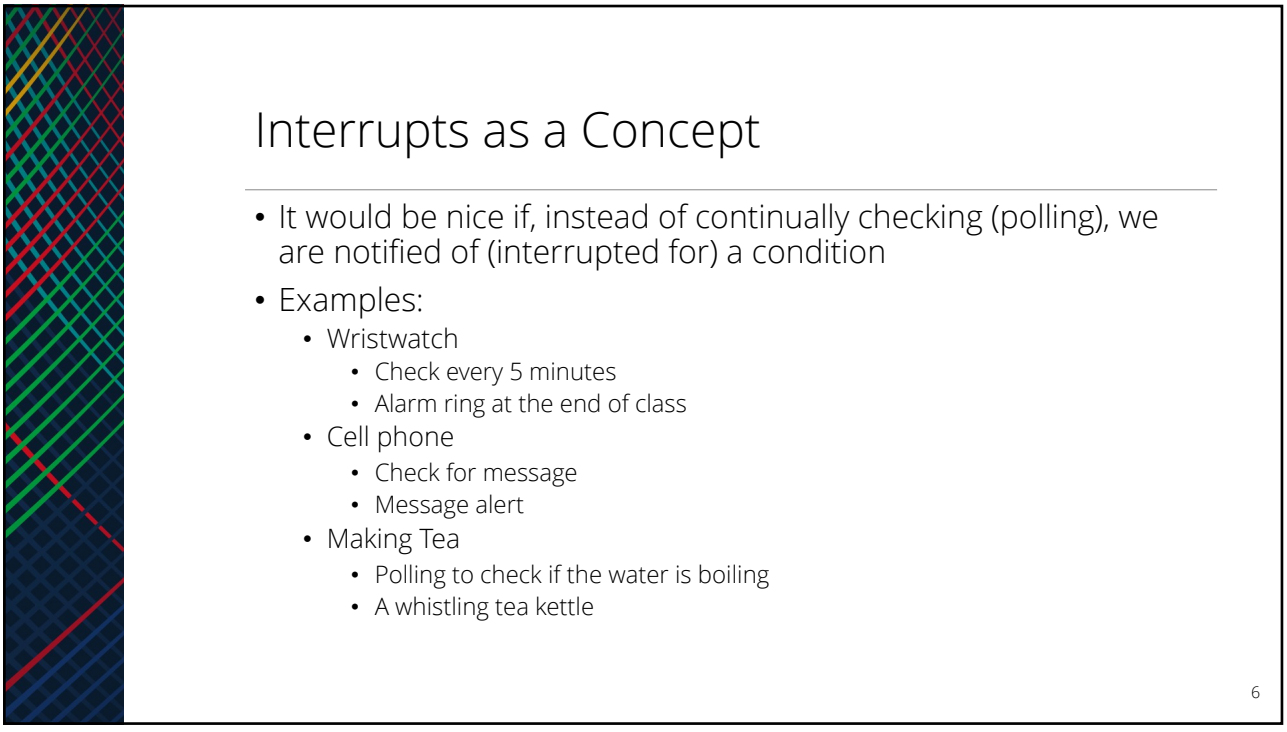
- The book code examples call a function called `EnableInterrupts()`
- We don't use it or need it
  - For us, interrupts are always enabled

4

4


Intro to Interrupts
5

5


Interrupts as a Concept
It would be nice if, instead of continually checking (polling), we are notified of (interrupted for) a condition
Examples:
Wristwatch
Check every 5 minutes
Alarm ring at the end of class
Cell phone
Check for message
Message alert
Making Tea
Polling to check if the water is boiling
A whistling tea kettle
6

6

Two Kinds of Interrupts

- Hardware Interrupt
  - CPU triggers interrupt based on some hardware condition
  - Finish the current instruction
  - Save current state of the CPU on the stack
  - Execute Interrupt Service Routine
  - Acknowledge the interrupts
  - Restore state from the stack
  - Resume execution of the main program
- Software Interrupt
  - Instruction within code causes an interrupt

7

7

HW Interrupt Sources

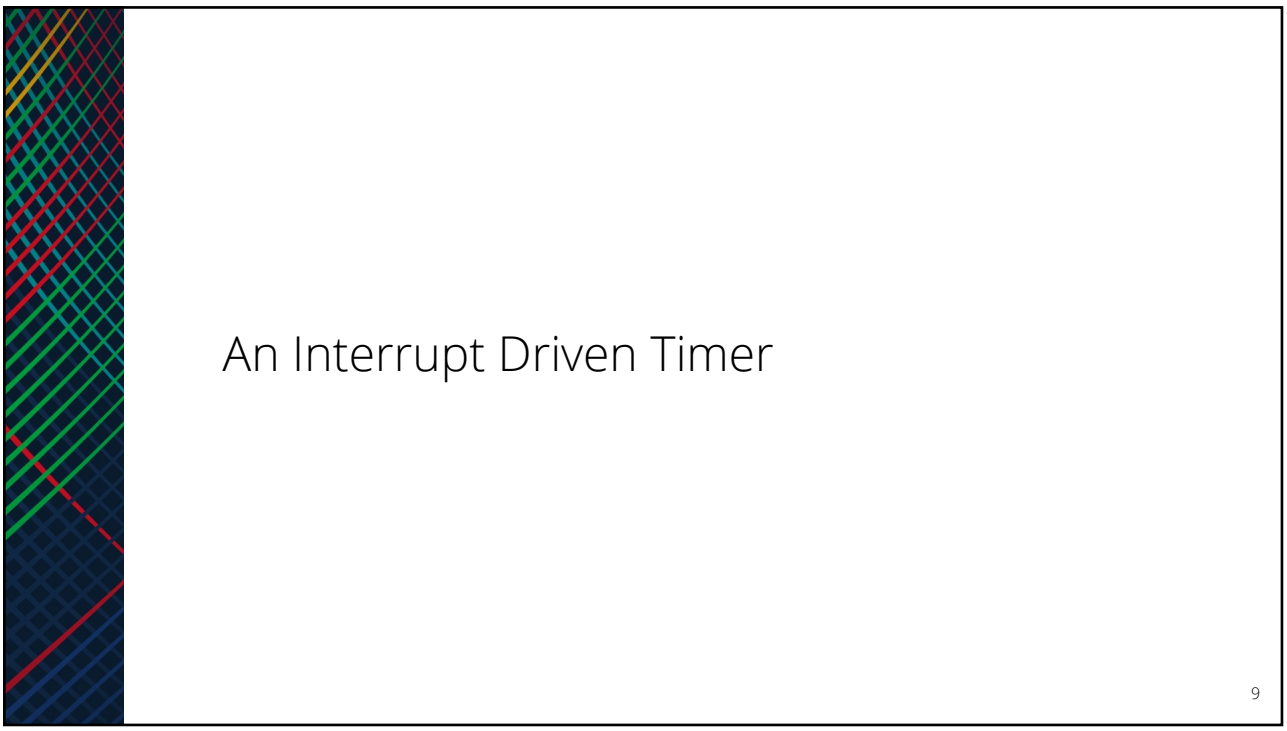
Something similar is on pages 104 and 152 in the June 2, 2014 datasheet for the TM4C123GH6PM

Source: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C12_Interrupts.htm

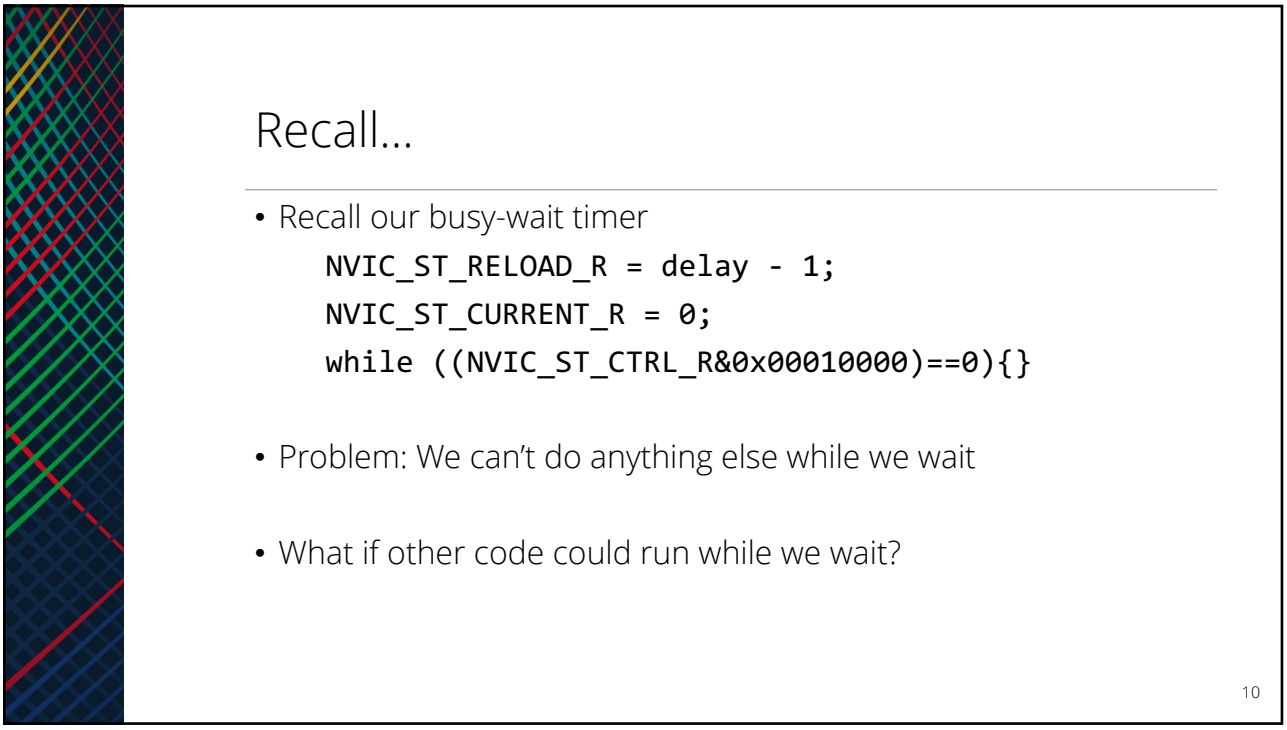
Vector address	Number	IRQ	ISR name in **Startup.s**	NVIC	Priority bits
0x00000038	14	-2	PendSV_Handler	NVIC_SYS_PRI3_R	23 – 21
0x0000003C	15	-1	SysTick_Handler	NVIC_SYS_PRI3_R	31 – 29
0x00000040	16	0	GPIOPortA_Handler	NVIC_PRI0_R	7 – 5
0x00000044	17	1	GPIOPortB_Handler	NVIC_PRI0_R	15 – 13
0x00000048	18	2	GPIOPortC_Handler	NVIC_PRI0_R	23 – 21
0x0000004C	19	3	GPIOPortD_Handler	NVIC_PRI0_R	31 – 29
0x00000050	20	4	GPIOPortE_Handler	NVIC_PRI1_R	7 – 5
0x00000054	21	5	UART0_Handler	NVIC_PRI1_R	15 – 13
0x00000058	22	6	UART1_Handler	NVIC_PRI1_R	23 – 21
0x0000005C	23	7	SSI0_Handler	NVIC_PRI1_R	31 – 29
0x00000060	24	8	I2C0_Handler	NVIC_PRI2_R	7 – 5
0x00000064	25	9	PWM0Fault_Handler	NVIC_PRI2_R	15 – 13
0x00000068	26	10	PWM0_Handler	NVIC_PRI2_R	23 – 21
0x0000006C	27	11	PWM1_Handler	NVIC_PRI2_R	31 – 29
0x00000070	28	12	PWM2_Handler	NVIC_PRI3_R	7 – 5
0x00000074	29	13	Quadrature0_Handler	NVIC_PRI3_R	15 – 13
0x00000078	30	14	ADC0_Handler	NVIC_PRI3_R	23 – 21
0x0000007C	31	15	ADC1_Handler	NVIC_PRI3_R	31 – 29
0x00000080	32	16	ADC2_Handler	NVIC_PRI4_R	7 – 5
0x00000084	33	17	ADC3_Handler	NVIC_PRI4_R	15 – 13
0x00000088	34	18	WDT_Handler	NVIC_PRI4_R	23 – 21
0x0000008C	35	19	Timer0A_Handler	NVIC_PRI4_R	31 – 29
0x00000090	36	20	Timer0B_Handler	NVIC_PRI5_R	7 – 5
0x00000094	37	21	Timer1A_Handler	NVIC_PRI5_R	15 – 13
0x00000098	38	22	Timer1B_Handler	NVIC_PRI5_R	23 – 21
0x0000009C	39	23	Timer2A_Handler	NVIC_PRI5_R	31 – 29
0x000000A0	40	24	Timer2B_Handler	NVIC_PRI6_R	7 – 5
0x000000A4	41	25	Comp0_Handler	NVIC_PRI6_R	15 – 13
0x000000A8	42	26	Comp1_Handler	NVIC_PRI6_R	23 – 21
0x000000AC	43	27	Comp2_Handler	NVIC_PRI6_R	31 – 29
0x000000B0	44	28	SysCtl_Handler	NVIC_PRI7_R	7 – 5
0x000000B4	45	29	FlashCtl_Handler	NVIC_PRI7_R	15 – 13
0x000000B8	46	30	GPIOPortF_Handler	NVIC_PRI7_R	23 – 21
0x000000BC	47	31	GPIOPortG_Handler	NVIC_PRI7_R	31 – 29
0x000000C0	48	32	GPIOPortH_Handler	NVIC_PRI8_R	7 – 5
0x000000C4	49	33	UART2_Handler	NVIC_PRI8_R	15 – 13
0x000000C8	50	34	SSI1_Handler	NVIC_PRI8_R	23 – 21
0x000000CC	51	35	Timer3A_Handler	NVIC_PRI8_R	31 – 29
0x000000D0	52	36	Timer3B_Handler	NVIC_PRI9_R	7 – 5
0x000000D4	53	37	I2C1_Handler	NVIC_PRI9_R	15 – 13
0x000000D8	54	38	Quadrature1_Handler	NVIC_PRI9_R	23 – 21
0x000000DC	55	39	CAN0_Handler	NVIC_PRI9_R	31 – 29
0x000000E0	56	40	CAN1_Handler	NVIC_PRI10_R	7 – 5
0x000000E4	57	41	CAN2_Handler	NVIC_PRI10_R	15 – 13
0x000000E8	58	42	Ethernet_Handler	NVIC_PRI10_R	23 – 21
0x000000EC	59	43	Hibernate_Handler	NVIC_PRI10_R	31 – 29
0x000000F0	60	44	USB0_Handler	NVIC_PRI11_R	7 – 5
0x000000F4	61	45	PWM3_Handler	NVIC_PRI11_R	15 – 13
0x000000F8	62	46	uDMA_Handler	NVIC_PRI11_R	23 – 21
0x000000FC	63	47	uDMA_Error	NVIC_PRI11_R	31 – 29

8

8


An Interrupt Driven Timer
9

9


Recall…
• Recall our busy-wait timer
NVIC_ST_RELOAD_R = delay - 1;
NVIC_ST_CURRENT_R = 0;
while ((NVIC_ST_CTRL_R&0x00010000)==0){}
• Problem: We can't do anything else while we wait
• What if other code could run while we wait?
10

10

Interrupt Driven Approach (Pseudo-code)

```
void timer_handler(void) {
    cnt++;
}

int main(void) {
    init_timer_interrupt_for_1ms();

    while(1) {
        // Do some work, whatever your loop needs
        if (cnt % 5 == 0) {
            // This runs every 5 ms and I don't busy-wait
        }
    }
}
```

11

11

Configuring SysTick Timer with Interrupts (NVIC_ST_CTRL_R)

SysTick Control and Status Register (STCTRL)

Base 0xE000.E000
Offset 0x010
Type RW, reset 0x0000.0004

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															COUNT
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													CLK_SRC	INTEN	ENABLE
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

- Count – 1 means SysTick timer has counted to 0
- CLK_SRC
  - 0 PIOSC divided by 4 (16MHz/4)
  - 1 System clock (80Mhz if `PLLInit()` was called)
- INTEN – Interrupts enabled
- ENABLE – Enable this timer

12

12

How Do I Specify My ISR?

- ISR is just a function you write
- Takes no arguments and returns nothing
- You need to add it to the Interrupt Vector Table
  - This is a big table of function pointers stored in the ROM
  - ISR must be added at compile time, not run-time

13

13

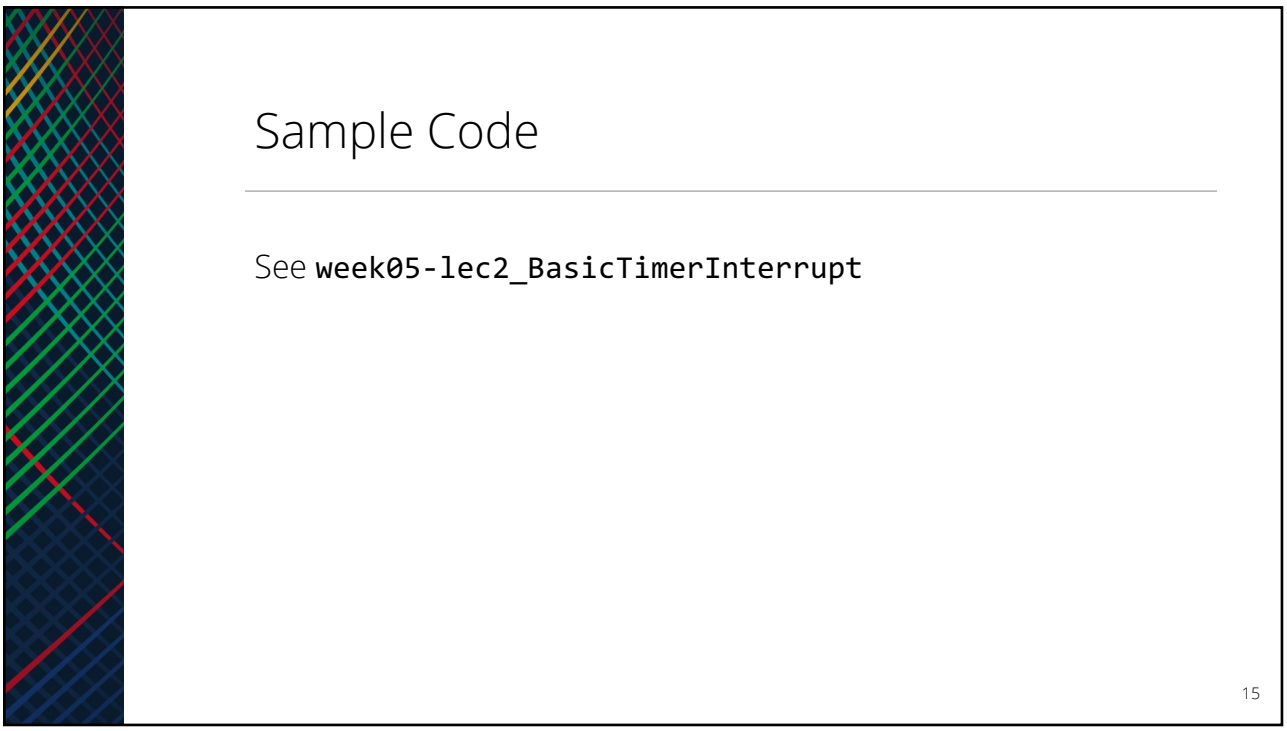
Adding ISR to the Vector Table

`tm4c123gh6pm_startup_ccs.c`

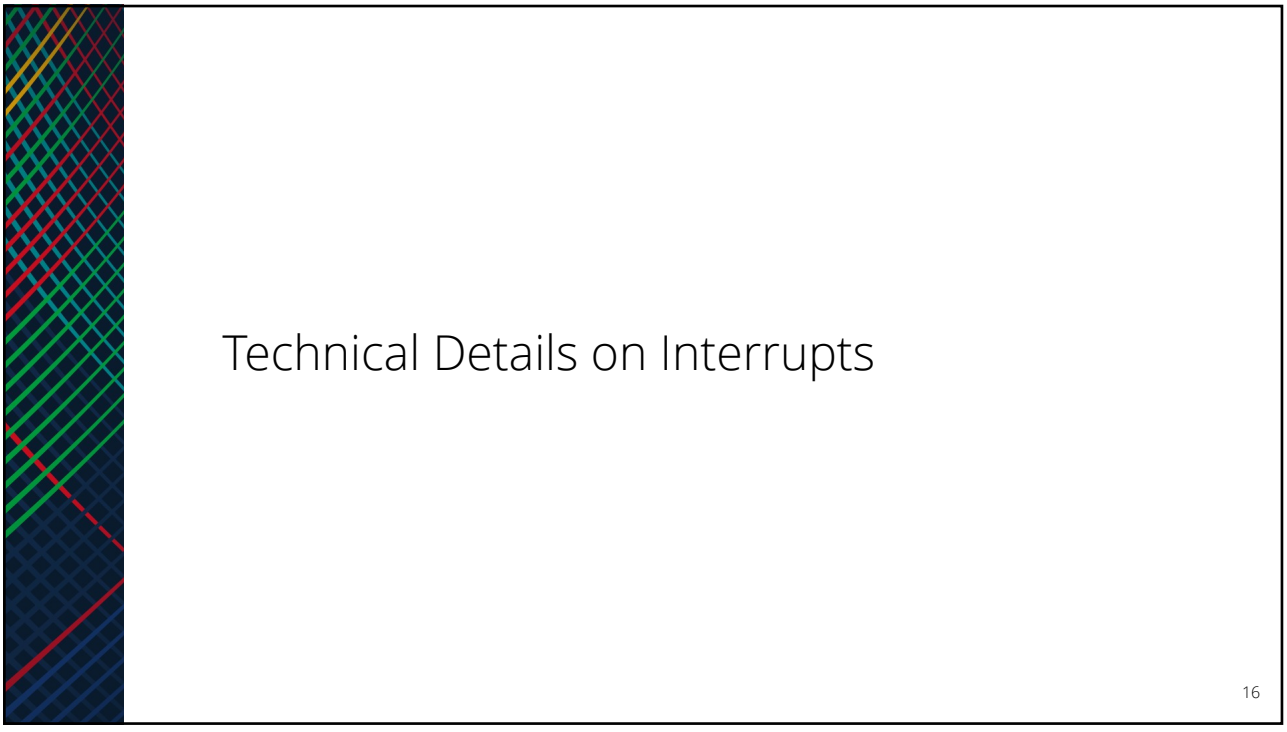
```
// To be added by user
extern void SysTickHandler(void);

...
#pragma DATA_SECTION( g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_TOP), // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                      // Reserved
    0,                      // Reserved
    0,                      // Reserved
    0,                      // Reserved
    IntDefaultHandler,                      // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
    0,                                      // Reserved
    IntDefaultHandler,                      // The PendSV handler
    SysTickHandler,                         // The SysTick handler
    IntDefaultHandler,                      // GPIO Port A
    IntDefaultHandler,                      // GPIO Port B
    IntDefaultHandler,                      // GPIO Port C
    IntDefaultHandler,                      // GPIO Port D
    IntDefaultHandler,                      // GPIO Port E

...
```

14

14


Sample Code
See week05-lec2_BasicTimerInterrupt
15

15


Technical Details on Interrupts
16

16

Simultaneous Interrupts

- Multiple interrupt generating things can be enabled at the same time
  - There are over 150 possible interrupts on our processor
  - Most are disabled by default
- A new interrupt can occur while servicing an existing interrupt
  - What if my timer interrupt goes off while I'm servicing an interrupt from a sensor?
- You can even have the same interrupt go off again while you are servicing it

17

17

Interrupt Priority

- You can set the priority of an interrupt
- If an interrupt is currently being serviced, and a higher priority interrupt happens, the the processor will interrupt the interrupt
- Priority registers
  - Each register holds 3-bit priorities for four interrupts
  - Priority is between 0 and 7
  - 0 highest, 7 lowest
- The TMC4C has 35 priority registers
  - See Table 5.2, page 270 for the most useful ones
  - See pages 104 and 152 in the datasheet to figure out all of them

18

18

Table 5.2 From the Book

Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	SysTick	PendSV	--	Debug	NVIC_SYS_PRI3_R

19

19

What Happens When an Interrupt Occurs?

- The current instruction is completed
- Eight registers (PSR, PC, LR, R12, R3, R2, R1, R0) are pushed on the stack
- The Address for the ISR is loaded in the PC
- IPSR register holds the Interrupt number
- LR is set to a special value
  - Top 24 bits of LR are set to 0xFFFFFF indicating ISR mode
  - Next 7 bits specify how to return from interrupt
    - 0xF1 Return to another interrupt (Handler mode)
    - 0xF9 Return to normal execution (Thread mode)

20

20

What Happens When an Interrupt Returns?

- ISR calls **BX LR**
  - Just like returning from a standard function call
- Processor notices top 24-bits are 0xFFFFFF and so doesn't do a normal function return
- Pops 8 registers from the stack
- Checks next 7-bits to determine whether we are returning to the main execution thread, or a different ISR

21

21

Warning: Registers and ISRs in Assembly

- Only 8 registers are saved and restored (PSR, PC, LR, R12, R3, R2, R1, R0)
- This means that if your ISR uses any other registers, it needs to save them at the beginning and restore them before the return
- The compiler does this if you are writing in C
- The assembler does not do this for you when writing in assembly

22

22

Interrupts from GPIO

Demo: Interrupts from GPIO Pins

`week05_lec2_InterruptFromPin`

Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	SysTick	PendSV	--	Debug	NVIC_SYS_PRI3_R

Source: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C12_Interrupts.htm