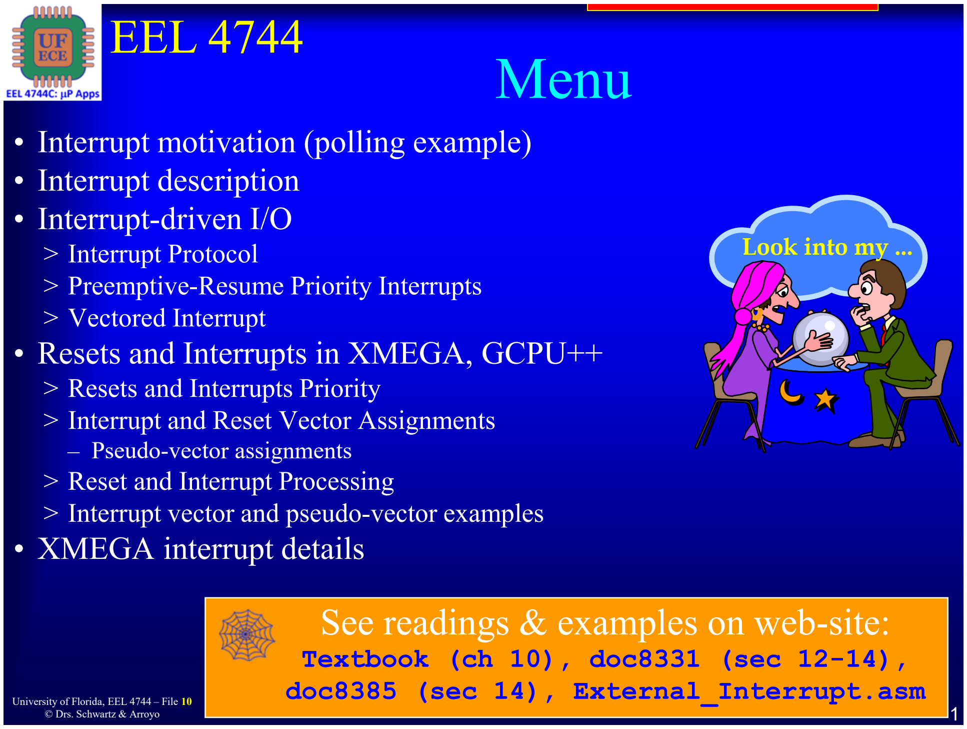
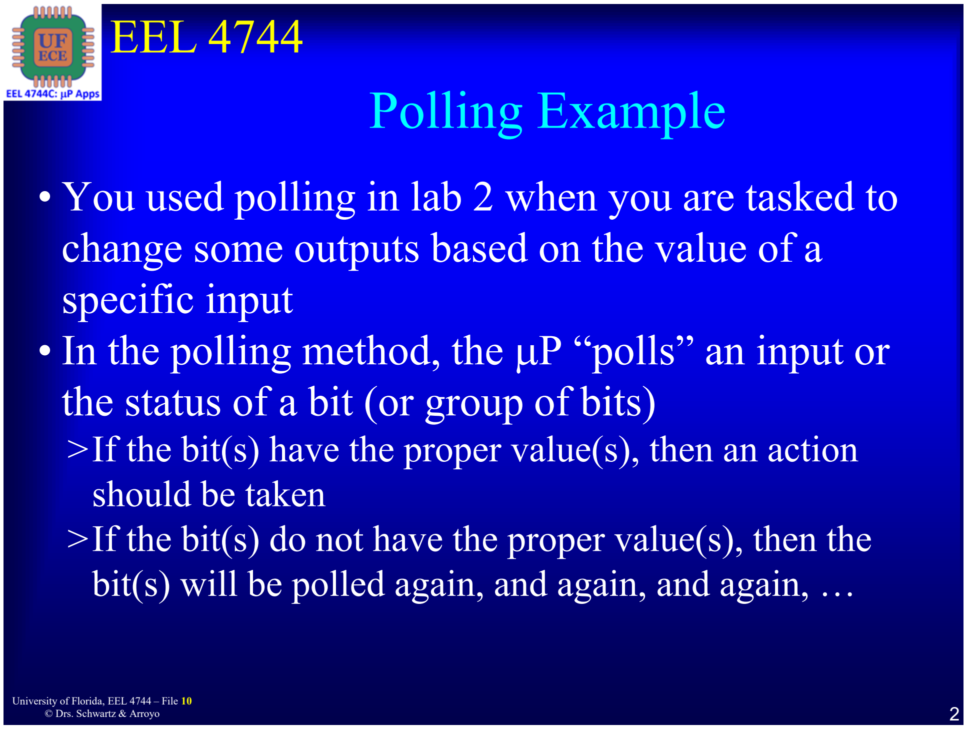


EEL 4744

Menu

- Interrupt motivation (polling example)
- Interrupt description
- Interrupt-driven I/O
  - > Interrupt Protocol
  - > Preemptive-Resume Priority Interrupts
  - > Vectored Interrupt
- Resets and Interrupts in XMEGA, GCPU++
  - > Resets and Interrupts Priority
  - > Interrupt and Reset Vector Assignments
    - – Pseudo-vector assignments
  - > Reset and Interrupt Processing
  - > Interrupt vector and pseudo-vector examples
- XMEGA interrupt details

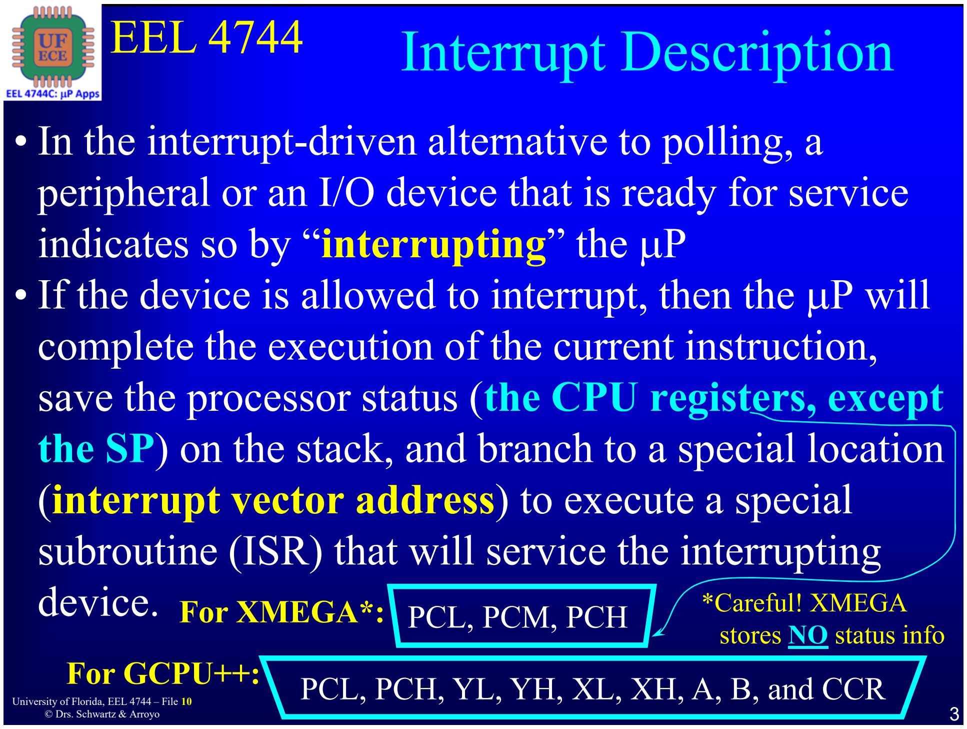
See readings & examples on web-site: `Textbook (ch 10), doc8331 (sec 12-14), doc8385 (sec 14), External_Interrupt.asm`



EEL 4744

Polling Example

- You used polling in lab 2 when you are tasked to change some outputs based on the value of a specific input
- In the polling method, the µP "polls" an input or the status of a bit (or group of bits)
  - >If the bit(s) have the proper value(s), then an action should be taken
  - >If the bit(s) do not have the proper value(s), then the bit(s) will be polled again, and again, and again, …



EEL 4744

Interrupt Description

- In the interrupt-driven alternative to polling, a peripheral or an I/O device that is ready for service indicates so by "**interrupting**" the µP
- If the device is allowed to interrupt, then the µP will complete the execution of the current instruction, save the processor status (**the CPU registers, except the SP**) on the stack, and branch to a special location (**interrupt vector address**) to execute a special subroutine (ISR) that will service the interrupting device.

For XMEGA*: PCL, PCM, PCH

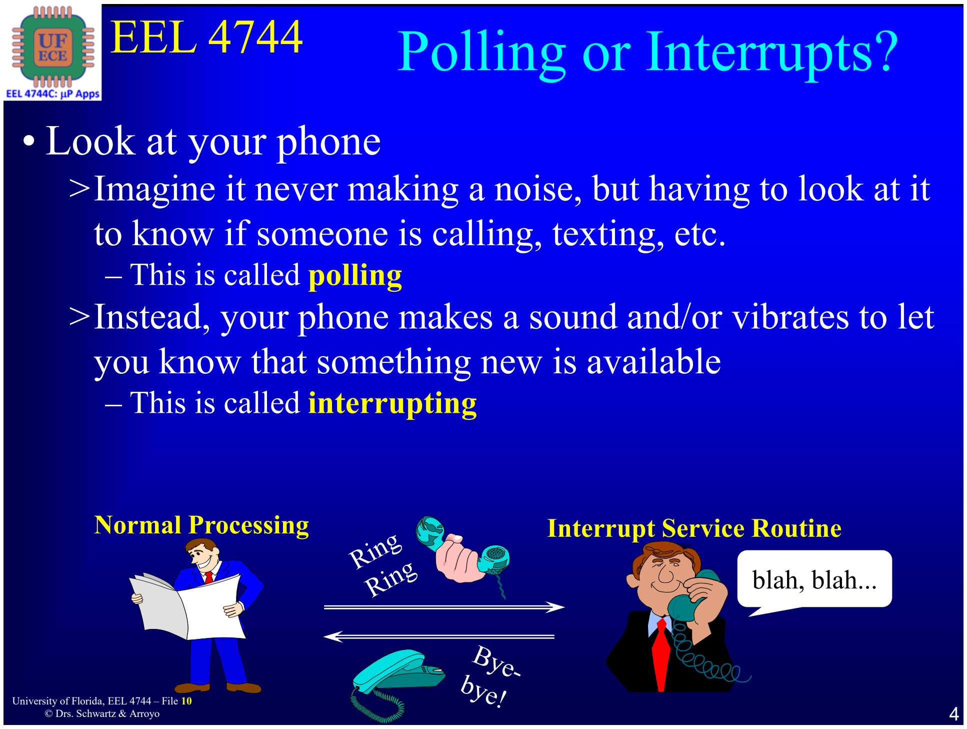
*Careful! XMEGA stores **NO** status info

For GCPU++: PCL, PCH, YL, YH, XL, XH, A, B, and CCR

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

3

3



EEL 4744

Polling or Interrupts?

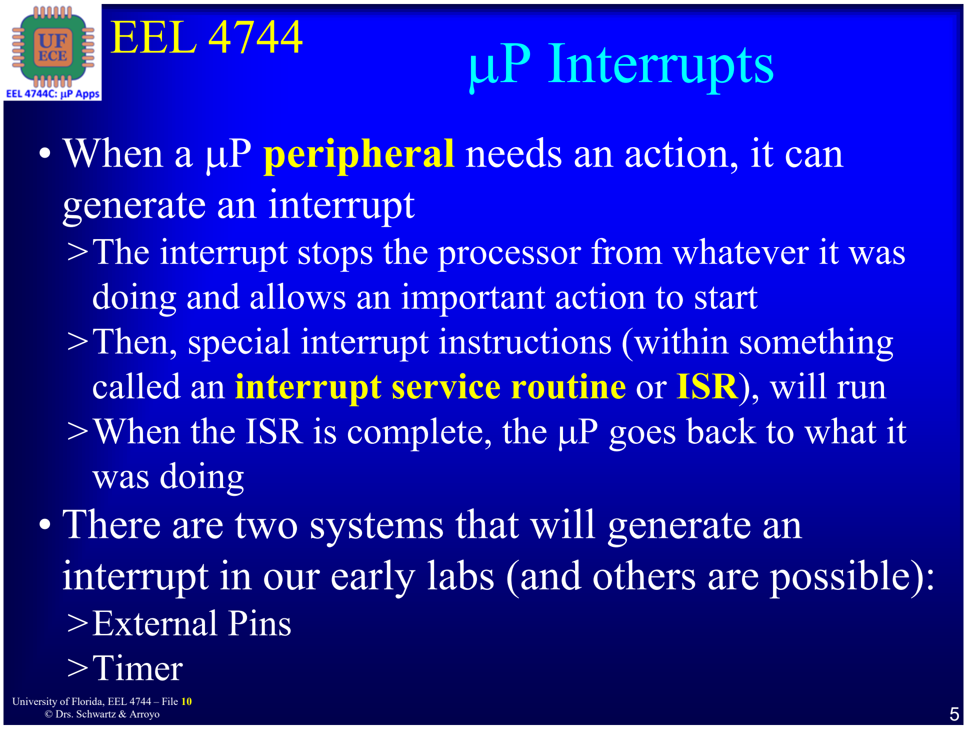
- Look at your phone
  > Imagine it never making a noise, but having to look at it to know if someone is calling, texting, etc.
  – This is called **polling**
  > Instead, your phone makes a sound and/or vibrates to let you know that something new is available
  – This is called **interrupting**

Normal Processing
Ring Ring
Interrupt Service Routine
blah, blah...
Bye-bye!

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

4

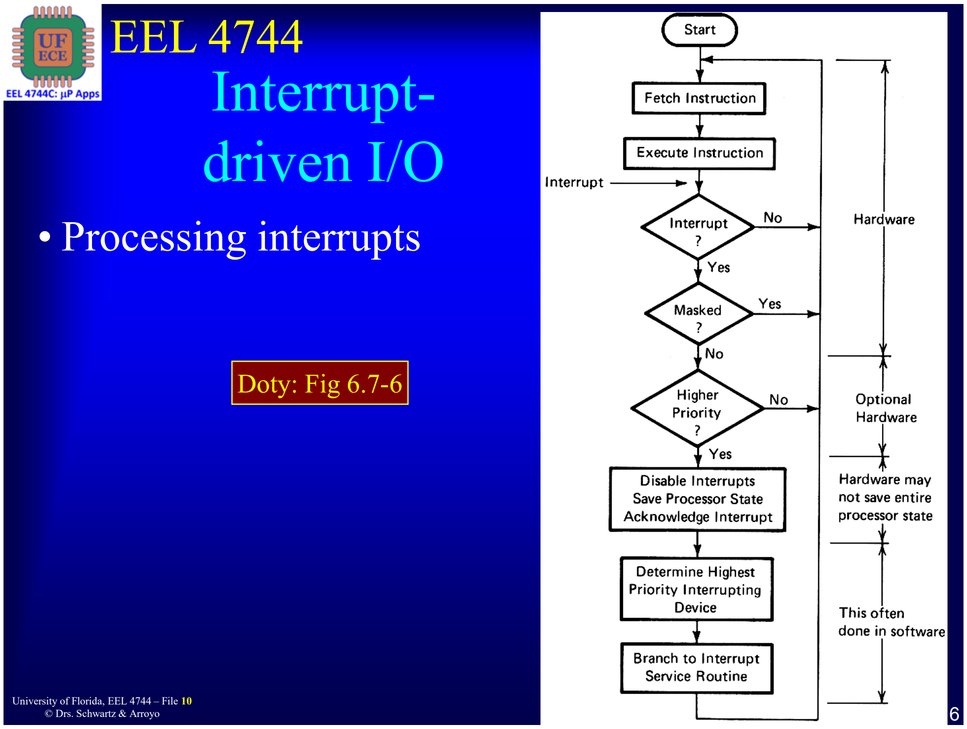
4



EEL 4744

µP Interrupts

- When a µP **peripheral** needs an action, it can generate an interrupt
  - >The interrupt stops the processor from whatever it was doing and allows an important action to start
  - >Then, special interrupt instructions (within something called an **interrupt service routine** or **ISR**), will run
  - >When the ISR is complete, the µP goes back to what it was doing
- There are two systems that will generate an interrupt in our early labs (and others are possible):
  - >External Pins
  - >Timer

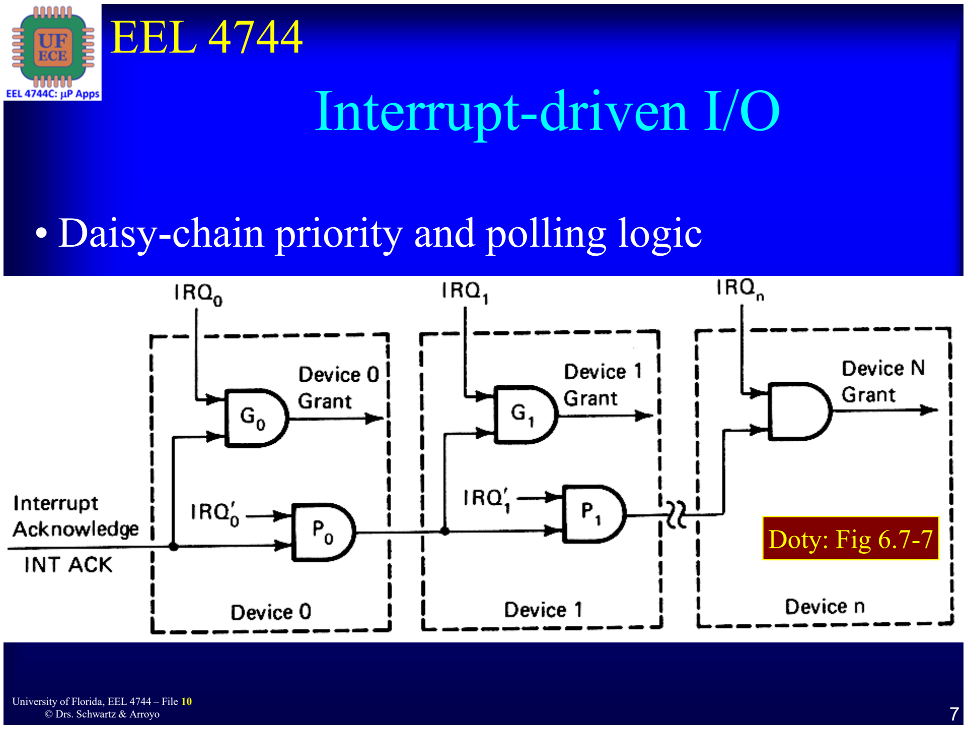


EEL 4744

Interrupt-driven I/O

- Processing interrupts

Doty: Fig 6.7-6



EEL 4744

Interrupt-driven I/O

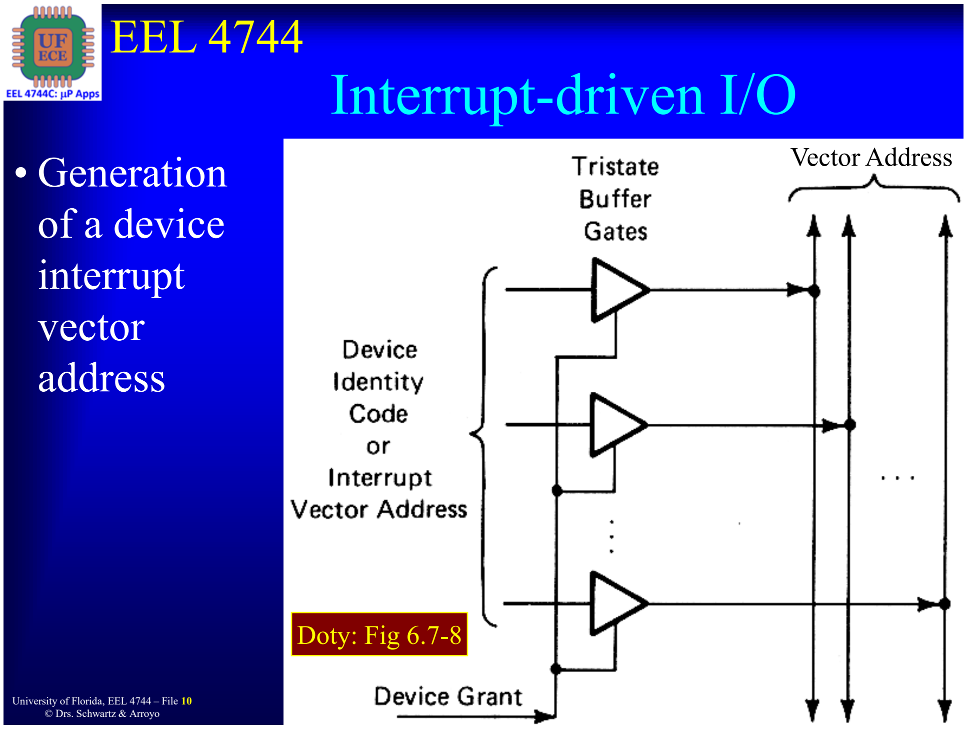
- Daisy-chain priority and polling logic

Doty: Fig 6.7-7

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

7

7



EEL 4744

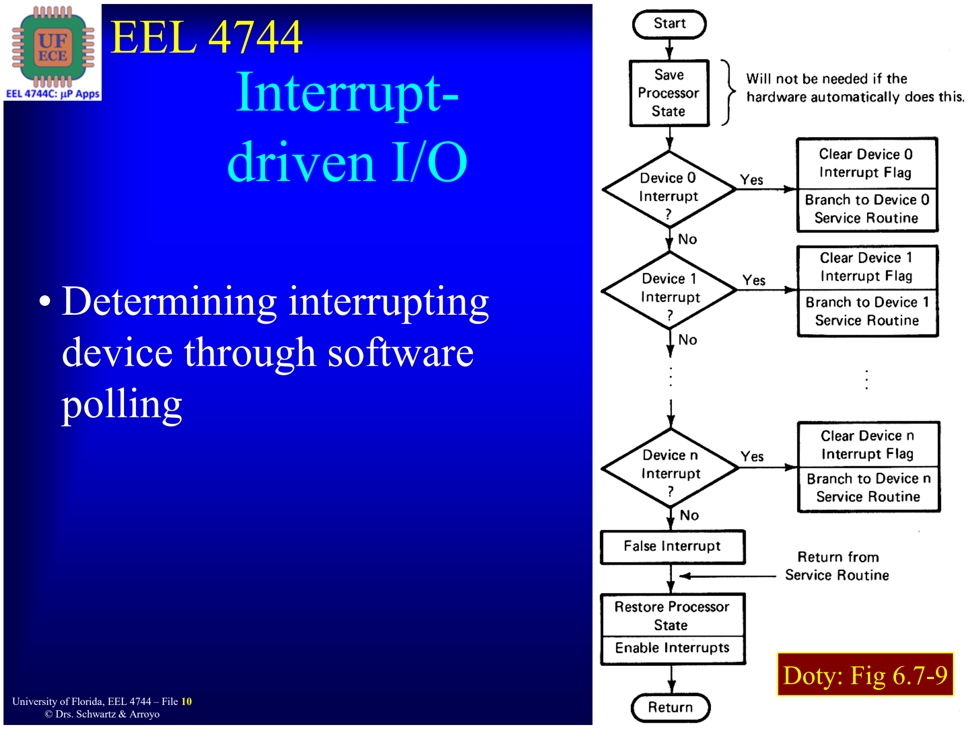
Interrupt-driven I/O

- Generation of a device interrupt vector address

Doty: Fig 6.7-8

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

8



EEL 4744C: µP Apps

EEL 4744

Interrupt-driven I/O

- Determining interrupting device through software polling

Start

Save Processor State

Will not be needed if the hardware automatically does this.

Device 0 Interrupt ? — Yes → Clear Device 0 Interrupt Flag / Branch to Device 0 Service Routine — No

Device 1 Interrupt ? — Yes → Clear Device 1 Interrupt Flag / Branch to Device 1 Service Routine — No

⋮

Device n Interrupt ? — Yes → Clear Device n Interrupt Flag / Branch to Device n Service Routine — No

False Interrupt

Return from Service Routine

Restore Processor State

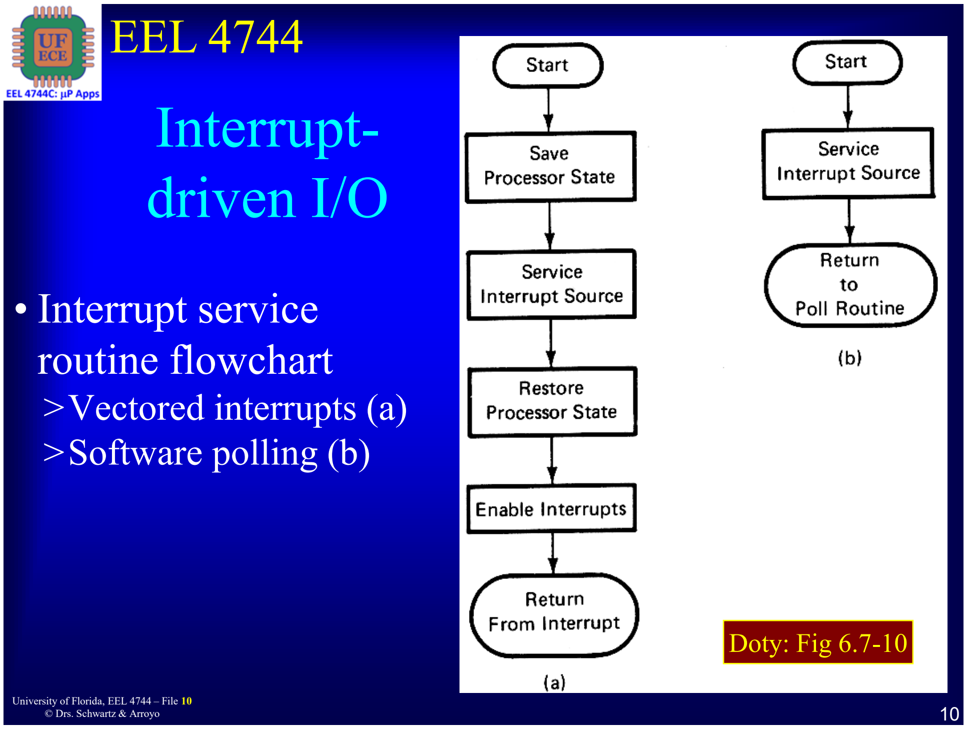
Enable Interrupts

Return

Doty: Fig 6.7-9

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

9



EEL 4744C: µP Apps

EEL 4744

Interrupt-driven I/O

- Interrupt service routine flowchart
  - >Vectored interrupts (a)
  - >Software polling (b)

Start

Save Processor State

Service Interrupt Source

Restore Processor State

Enable Interrupts

Return From Interrupt

(a)

Start

Service Interrupt Source

Return to Poll Routine

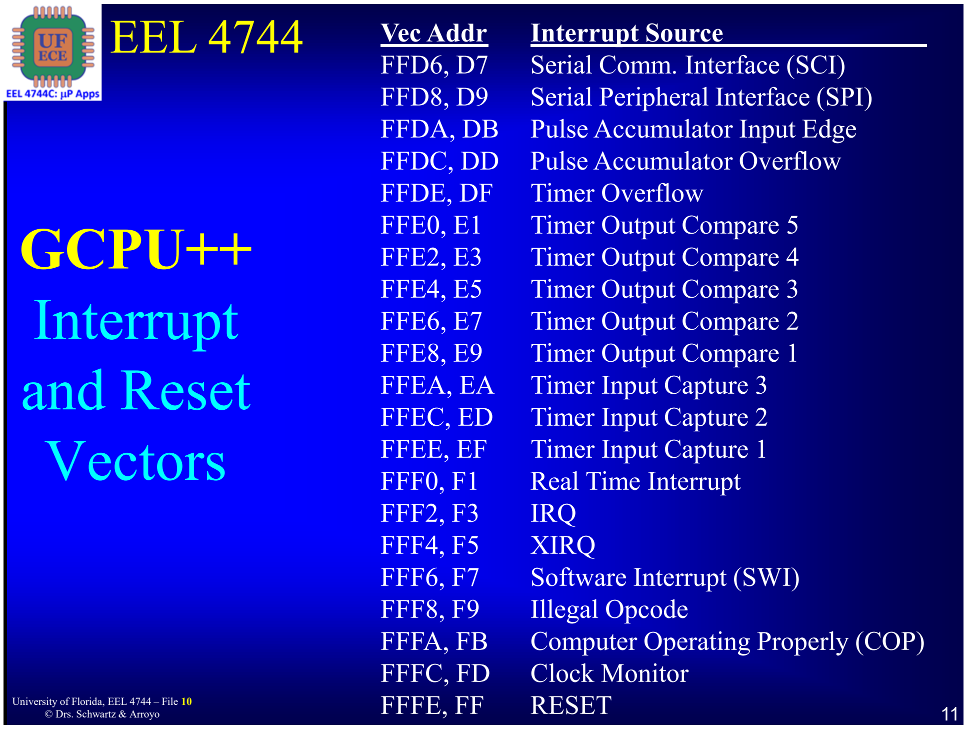
(b)

Doty: Fig 6.7-10

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

10

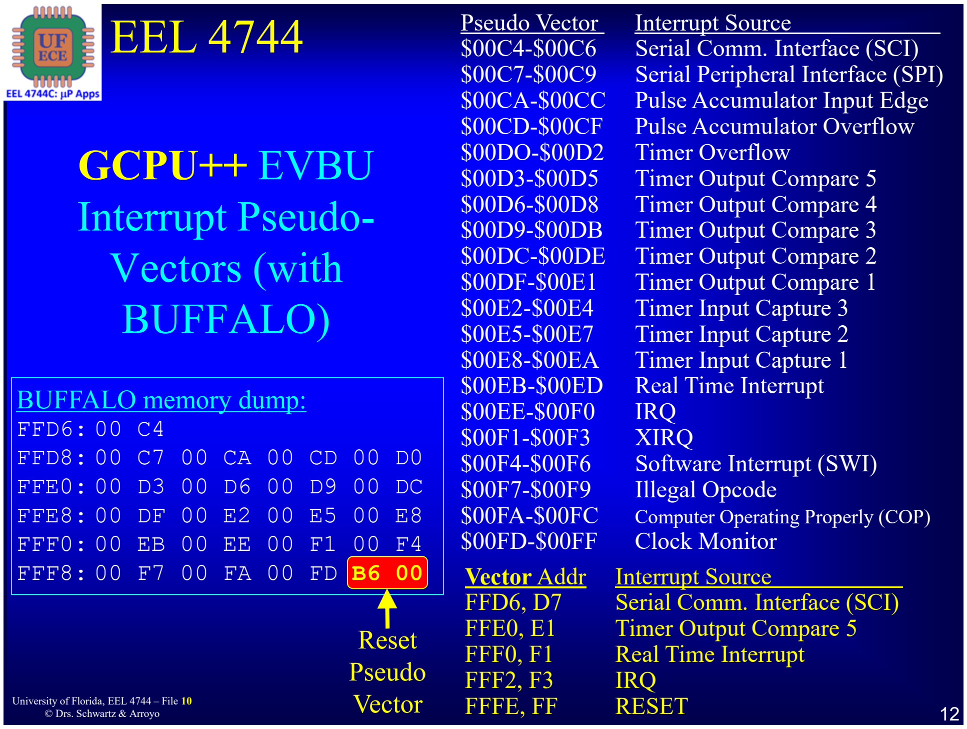
10



EEL 4744

GCPU++ Interrupt and Reset Vectors

Vec Addr	Interrupt Source
FFD6, D7	Serial Comm. Interface (SCI)
FFD8, D9	Serial Peripheral Interface (SPI)
FFDA, DB	Pulse Accumulator Input Edge
FFDC, DD	Pulse Accumulator Overflow
FFDE, DF	Timer Overflow
FFE0, E1	Timer Output Compare 5
FFE2, E3	Timer Output Compare 4
FFE4, E5	Timer Output Compare 3
FFE6, E7	Timer Output Compare 2
FFE8, E9	Timer Output Compare 1
FFEA, EA	Timer Input Capture 3
FFEC, ED	Timer Input Capture 2
FFEE, EF	Timer Input Capture 1
FFF0, F1	Real Time Interrupt
FFF2, F3	IRQ
FFF4, F5	XIRQ
FFF6, F7	Software Interrupt (SWI)
FFF8, F9	Illegal Opcode
FFFA, FB	Computer Operating Properly (COP)
FFFC, FD	Clock Monitor
FFFE, FF	RESET



EEL 4744

GCPU++ EVBU Interrupt Pseudo-Vectors (with BUFFALO)

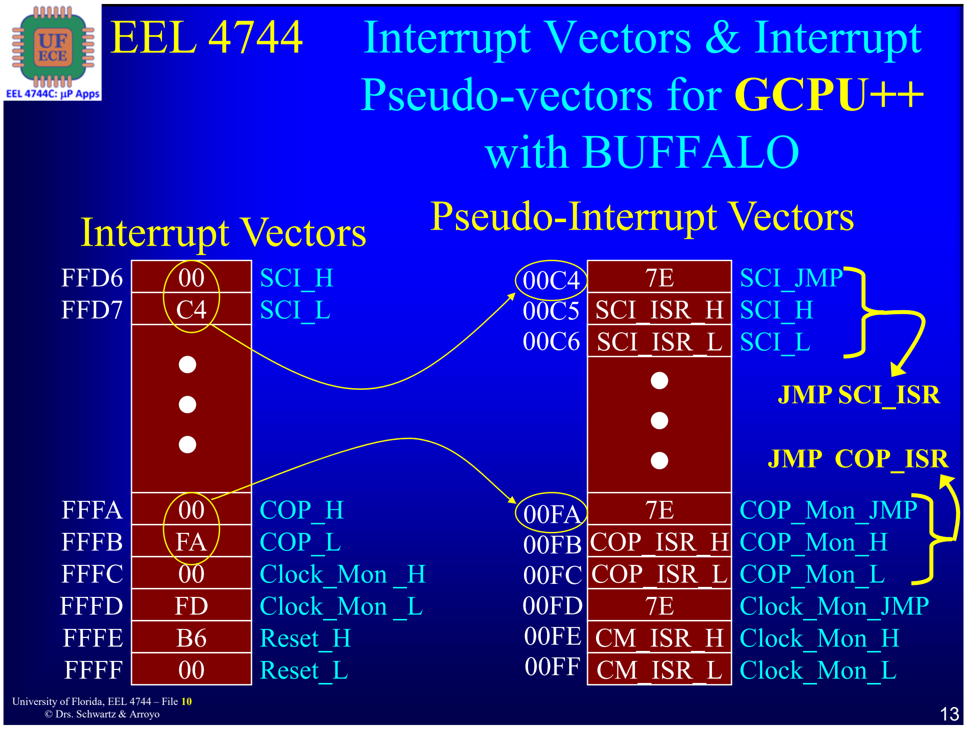
BUFFALO memory dump:

```
FFD6: 00 C4
FFD8: 00 C7 00 CA 00 CD 00 D0
FFE0: 00 D3 00 D6 00 D9 00 DC
FFE8: 00 DF 00 E2 00 E5 00 E8
FFF0: 00 EB 00 EE 00 F1 00 F4
FFF8: 00 F7 00 FA 00 FD B6 00
```

Reset Pseudo Vector

Pseudo Vector	Interrupt Source
$00C4-$00C6	Serial Comm. Interface (SCI)
$00C7-$00C9	Serial Peripheral Interface (SPI)
$00CA-$00CC	Pulse Accumulator Input Edge
$00CD-$00CF	Pulse Accumulator Overflow
$00DO-$00D2	Timer Overflow
$00D3-$00D5	Timer Output Compare 5
$00D6-$00D8	Timer Output Compare 4
$00D9-$00DB	Timer Output Compare 3
$00DC-$00DE	Timer Output Compare 2
$00DF-$00E1	Timer Output Compare 1
$00E2-$00E4	Timer Input Capture 3
$00E5-$00E7	Timer Input Capture 2
$00E8-$00EA	Timer Input Capture 1
$00EB-$00ED	Real Time Interrupt
$00EE-$00F0	IRQ
$00F1-$00F3	XIRQ
$00F4-$00F6	Software Interrupt (SWI)
$00F7-$00F9	Illegal Opcode
$00FA-$00FC	Computer Operating Properly (COP)
$00FD-$00FF	Clock Monitor

Vector Addr	Interrupt Source
FFD6, D7	Serial Comm. Interface (SCI)
FFE0, E1	Timer Output Compare 5
FFF0, F1	Real Time Interrupt
FFF2, F3	IRQ
FFFE, FF	RESET



EEL 4744C: µP Apps

EEL 4744 Interrupt Vectors & Interrupt Pseudo-vectors for GCPU++ with BUFFALO

Interrupt Vectors

FFD6	00	SCI_H
FFD7	C4	SCI_L
	● ● ●	
FFFA	00	COP_H
FFFB	FA	COP_L
FFFC	00	Clock_Mon _H
FFFD	FD	Clock_Mon _L
FFFE	B6	Reset_H
FFFF	00	Reset_L

Pseudo-Interrupt Vectors

00C4	7E	SCI_JMP
00C5	SCI_ISR_H	SCI_H
00C6	SCI_ISR_L	SCI_L
	● ● ●	
00FA	7E	COP_Mon_JMP
00FB	COP_ISR_H	COP_Mon_H
00FC	COP_ISR_L	COP_Mon_L
00FD	7E	Clock_Mon_JMP
00FE	CM_ISR_H	Clock_Mon_H
00FF	CM_ISR_L	Clock_Mon_L

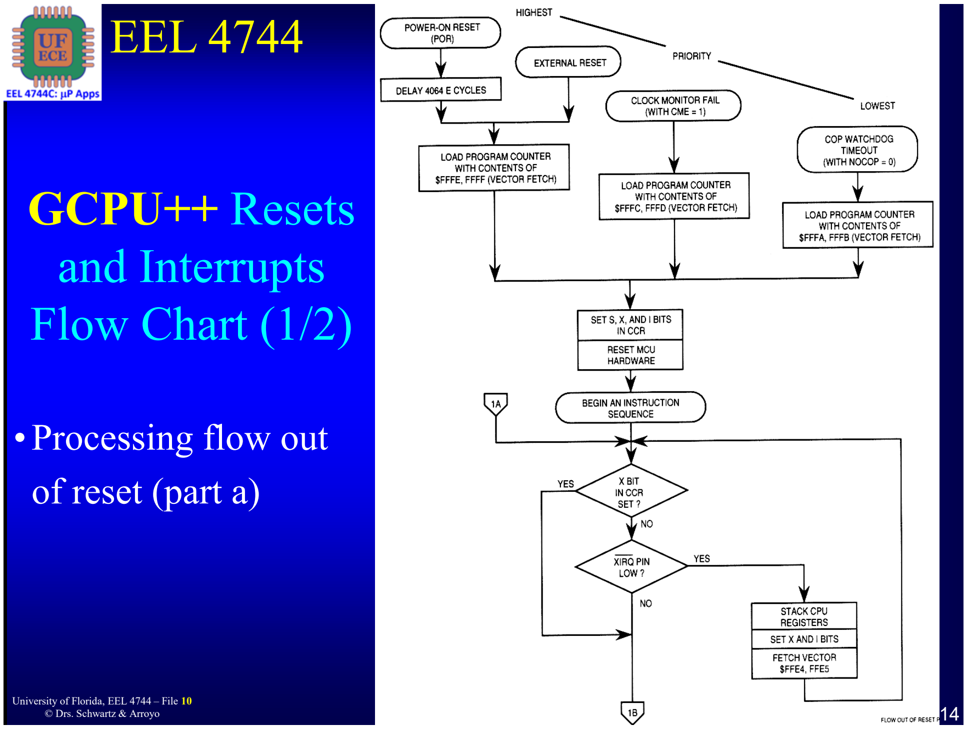
JMP SCI_ISR

JMP COP_ISR

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

13

13



EEL 4744C: µP Apps

EEL 4744

GCPU++ Resets and Interrupts Flow Chart (1/2)

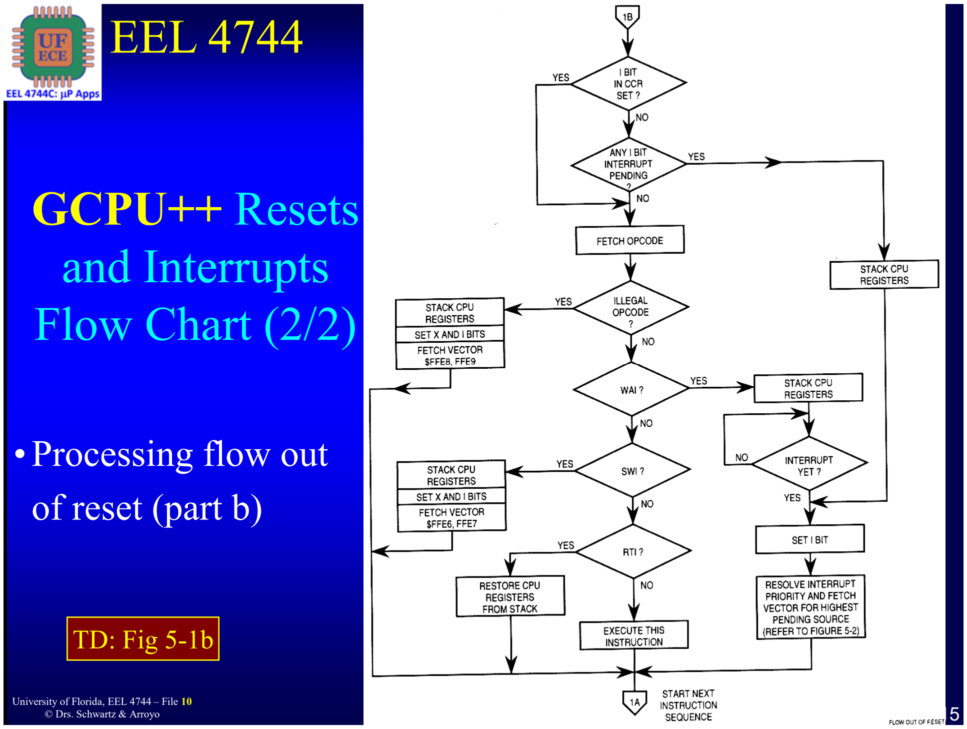
- Processing flow out of reset (part a)

HIGHEST
PRIORITY
LOWEST
POWER-ON RESET (POR)
DELAY 4064 E CYCLES
EXTERNAL RESET
CLOCK MONITOR FAIL (WITH CME = 1)
COP WATCHDOG TIMEOUT (WITH NOCOP = 0)
LOAD PROGRAM COUNTER WITH CONTENTS OF $FFFE, FFFF (VECTOR FETCH)
LOAD PROGRAM COUNTER WITH CONTENTS OF $FFFC, FFFD (VECTOR FETCH)
LOAD PROGRAM COUNTER WITH CONTENTS OF $FFFA, FFFB (VECTOR FETCH)
SET S, X, AND I BITS IN CCR
RESET MCU HARDWARE
BEGIN AN INSTRUCTION SEQUENCE
1A
X BIT IN CCR SET ?
YES
NO
$\overline{\text{XIRQ}}$ PIN LOW ?
YES
NO
STACK CPU REGISTERS
SET X AND I BITS
FETCH VECTOR $FFE4, FFE5
1B
FLOW OUT OF RESET P

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

14

14



EEL 4744

EEL 4744C: µP Apps

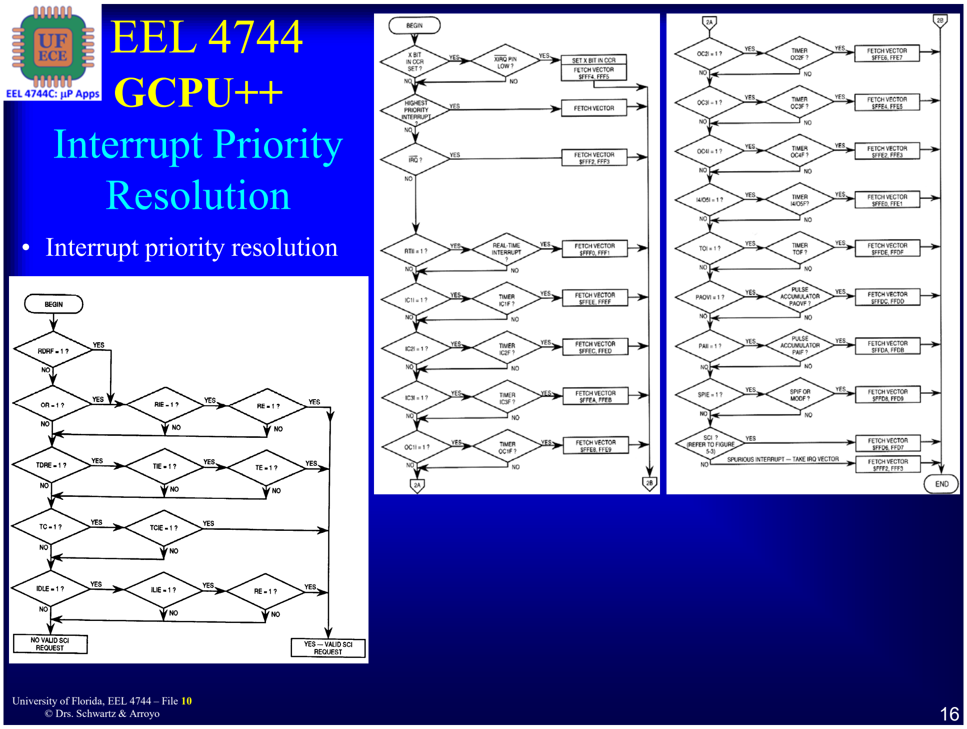
GCPU++ Resets and Interrupts Flow Chart (2/2)

- Processing flow out of reset (part b)

TD: Fig 5-1b

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

15



EEL 4744

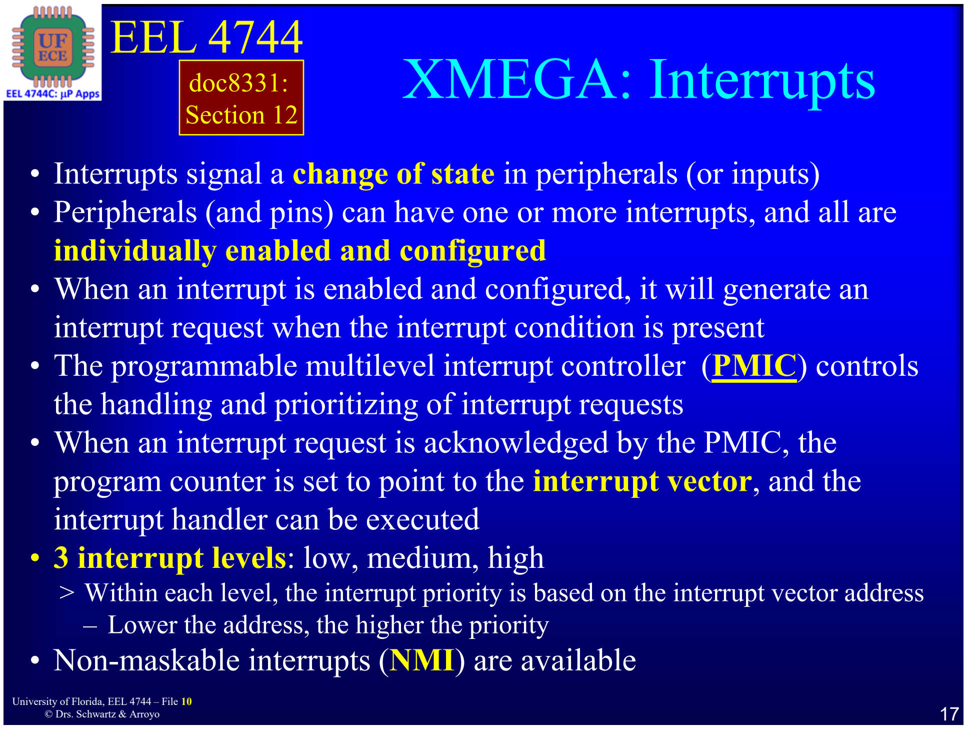
EEL 4744C: µP Apps

GCPU++ Interrupt Priority Resolution

- Interrupt priority resolution

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

16



EEL 4744C: µP Apps

EEL 4744

doc8331:
Section 12

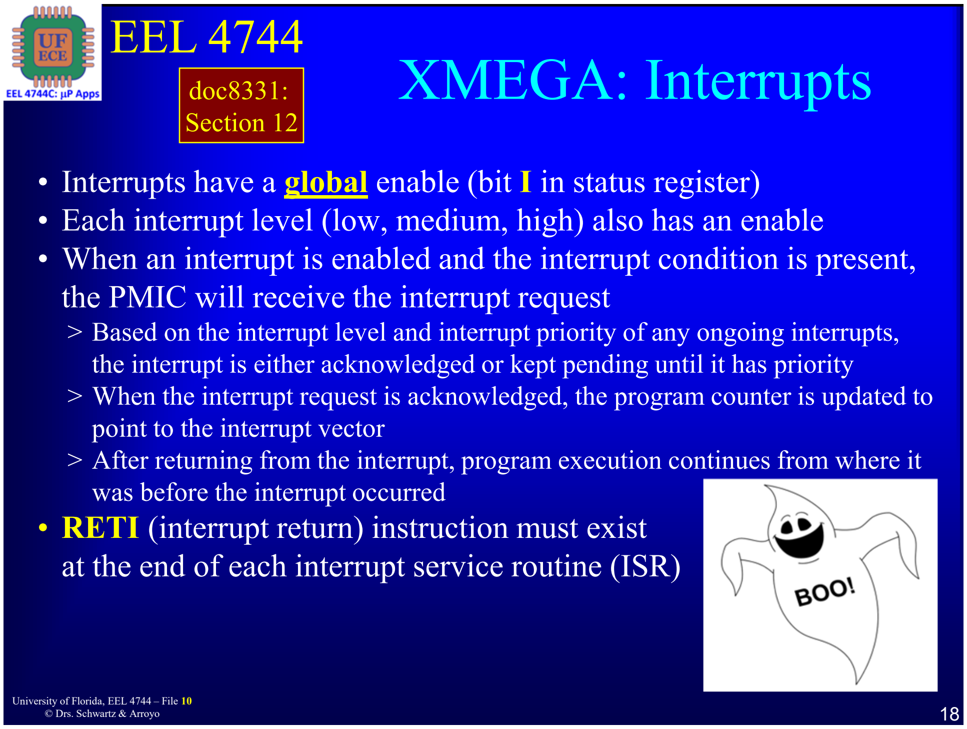
XMEGA: Interrupts

- Interrupts signal a **change of state** in peripherals (or inputs)
- Peripherals (and pins) can have one or more interrupts, and all are **individually enabled and configured**
- When an interrupt is enabled and configured, it will generate an interrupt request when the interrupt condition is present
- The programmable multilevel interrupt controller (**PMIC**) controls the handling and prioritizing of interrupt requests
- When an interrupt request is acknowledged by the PMIC, the program counter is set to point to the **interrupt vector**, and the interrupt handler can be executed
- **3 interrupt levels**: low, medium, high
  - > Within each level, the interrupt priority is based on the interrupt vector address
    - – Lower the address, the higher the priority
- Non-maskable interrupts (**NMI**) are available

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

17

17



EEL 4744C: µP Apps

EEL 4744

doc8331:
Section 12

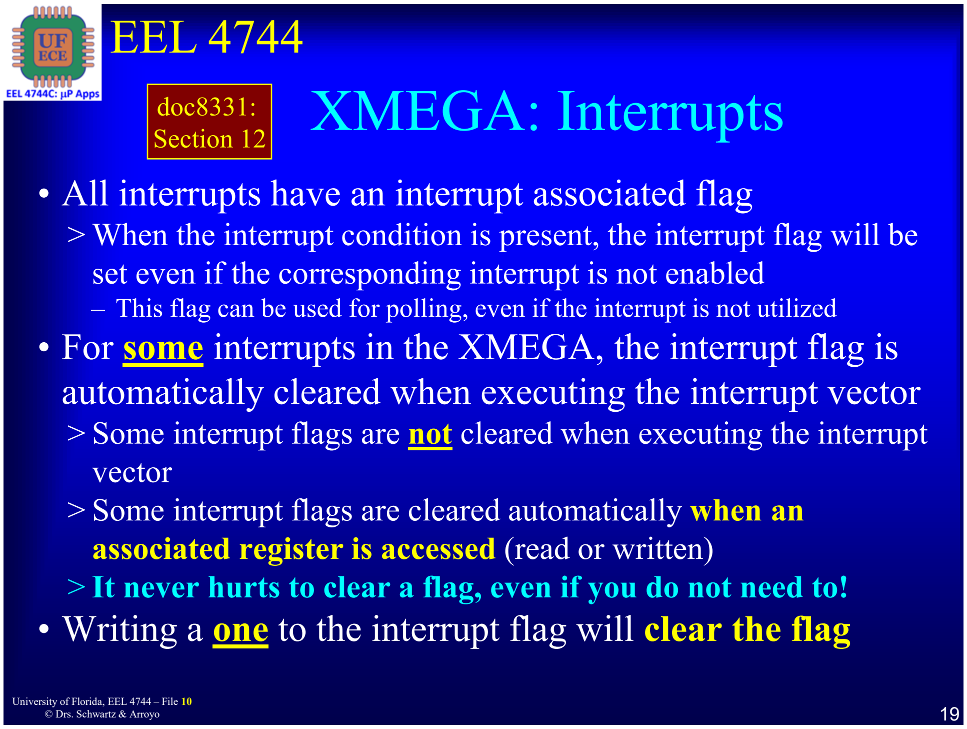
XMEGA: Interrupts

- Interrupts have a **global** enable (bit **I** in status register)
- Each interrupt level (low, medium, high) also has an enable
- When an interrupt is enabled and the interrupt condition is present, the PMIC will receive the interrupt request
  - > Based on the interrupt level and interrupt priority of any ongoing interrupts, the interrupt is either acknowledged or kept pending until it has priority
  - > When the interrupt request is acknowledged, the program counter is updated to point to the interrupt vector
  - > After returning from the interrupt, program execution continues from where it was before the interrupt occurred
- **RETI** (interrupt return) instruction must exist at the end of each interrupt service routine (ISR)

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

18

18



EEL 4744

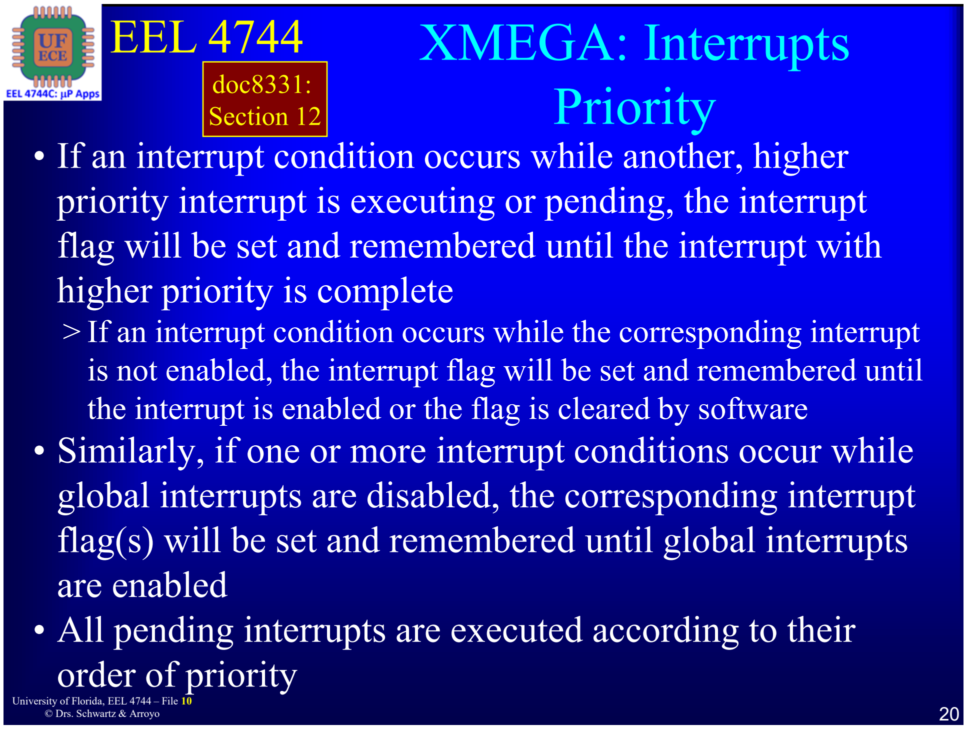
doc8331: Section 12

XMEGA: Interrupts

- All interrupts have an interrupt associated flag
  - > When the interrupt condition is present, the interrupt flag will be set even if the corresponding interrupt is not enabled
    - – This flag can be used for polling, even if the interrupt is not utilized
- For **some** interrupts in the XMEGA, the interrupt flag is automatically cleared when executing the interrupt vector
  - > Some interrupt flags are **not** cleared when executing the interrupt vector
  - > Some interrupt flags are cleared automatically **when an associated register is accessed** (read or written)
  - > **It never hurts to clear a flag, even if you do not need to!**
- Writing a **one** to the interrupt flag will **clear the flag**

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo
19

19



EEL 4744

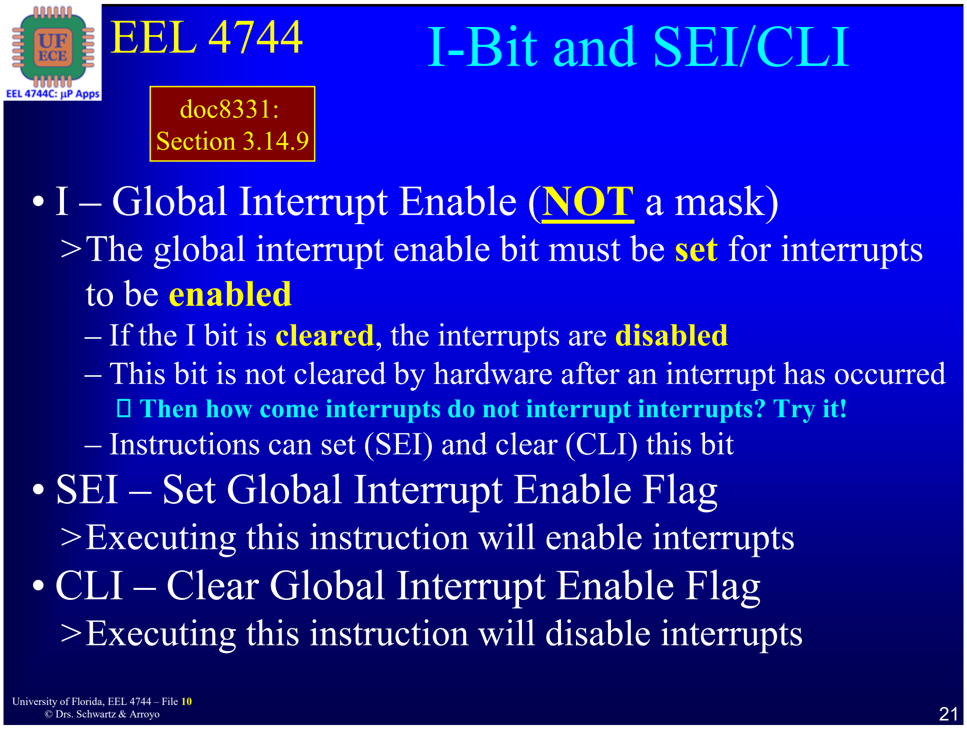
doc8331: Section 12

XMEGA: Interrupts Priority

- If an interrupt condition occurs while another, higher priority interrupt is executing or pending, the interrupt flag will be set and remembered until the interrupt with higher priority is complete
  - > If an interrupt condition occurs while the corresponding interrupt is not enabled, the interrupt flag will be set and remembered until the interrupt is enabled or the flag is cleared by software
- Similarly, if one or more interrupt conditions occur while global interrupts are disabled, the corresponding interrupt flag(s) will be set and remembered until global interrupts are enabled
- All pending interrupts are executed according to their order of priority

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo
20

20

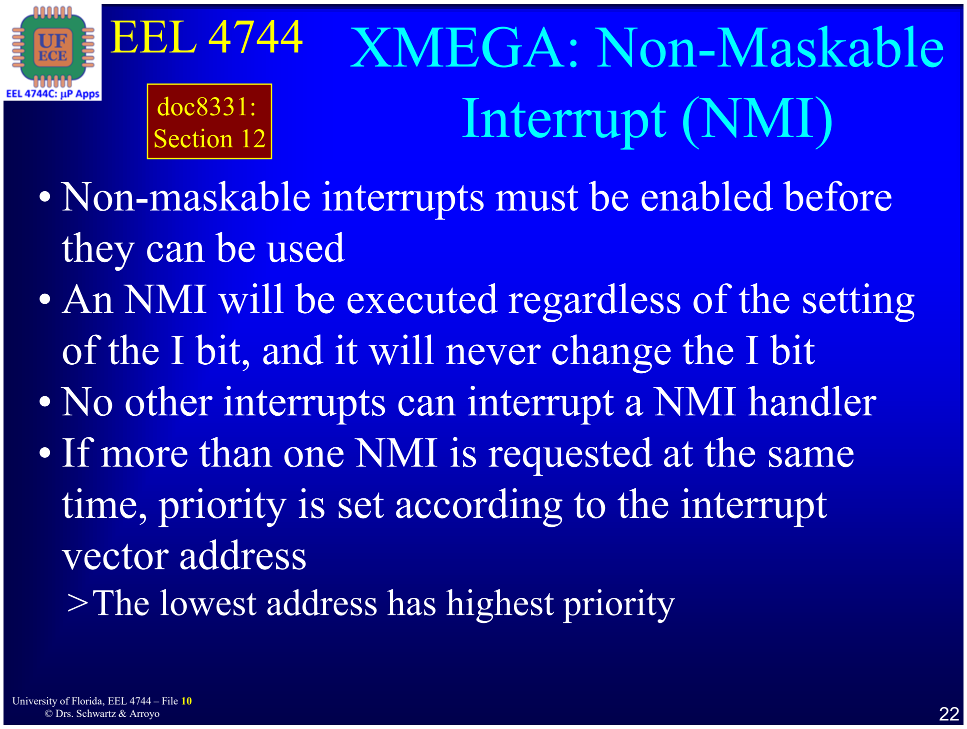


EEL 4744

I-Bit and SEI/CLI

doc8331: Section 3.14.9

- I – Global Interrupt Enable (**NOT** a mask)
  - > The global interrupt enable bit must be **set** for interrupts to be **enabled**
    - – If the I bit is **cleared**, the interrupts are **disabled**
    - – This bit is not cleared by hardware after an interrupt has occurred
      - **Then how come interrupts do not interrupt interrupts? Try it!**
    - – Instructions can set (SEI) and clear (CLI) this bit
- SEI – Set Global Interrupt Enable Flag
  - > Executing this instruction will enable interrupts
- CLI – Clear Global Interrupt Enable Flag
  - > Executing this instruction will disable interrupts

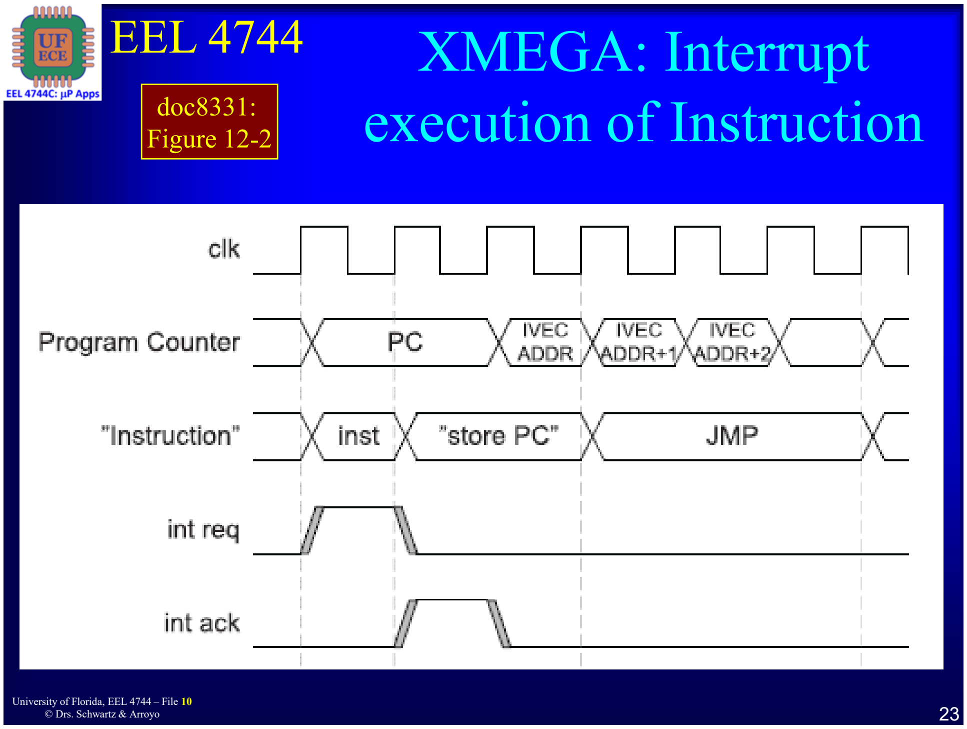


EEL 4744

XMEGA: Non-Maskable Interrupt (NMI)

doc8331: Section 12

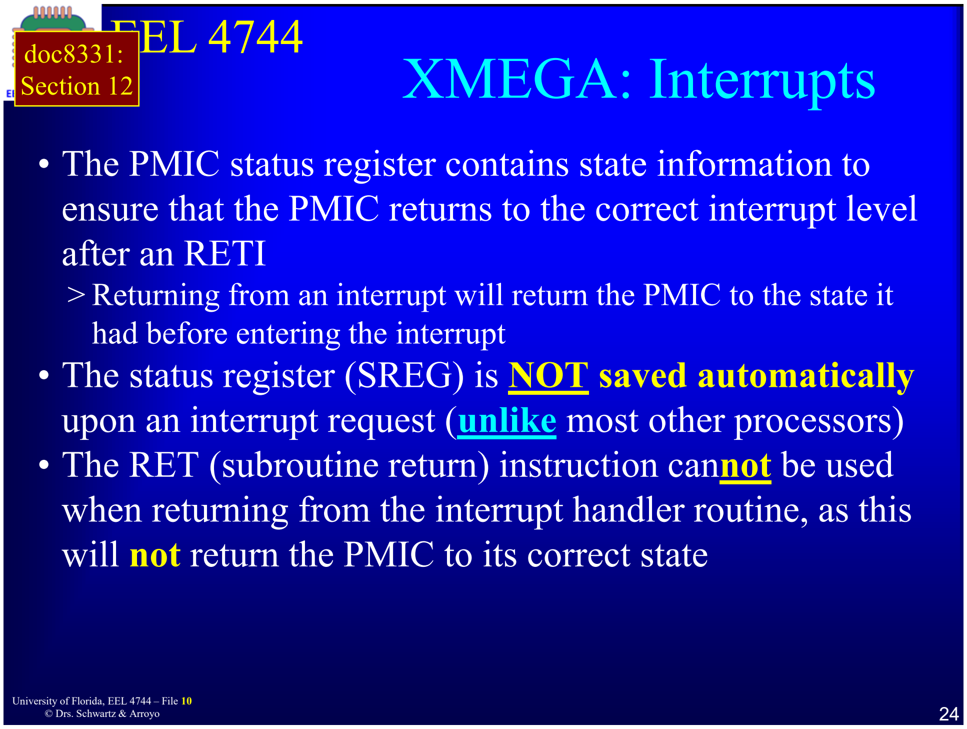
- Non-maskable interrupts must be enabled before they can be used
- An NMI will be executed regardless of the setting of the I bit, and it will never change the I bit
- No other interrupts can interrupt a NMI handler
- If more than one NMI is requested at the same time, priority is set according to the interrupt vector address
  - > The lowest address has highest priority



EEL 4744

doc8331: Figure 12-2

XMEGA: Interrupt execution of Instruction

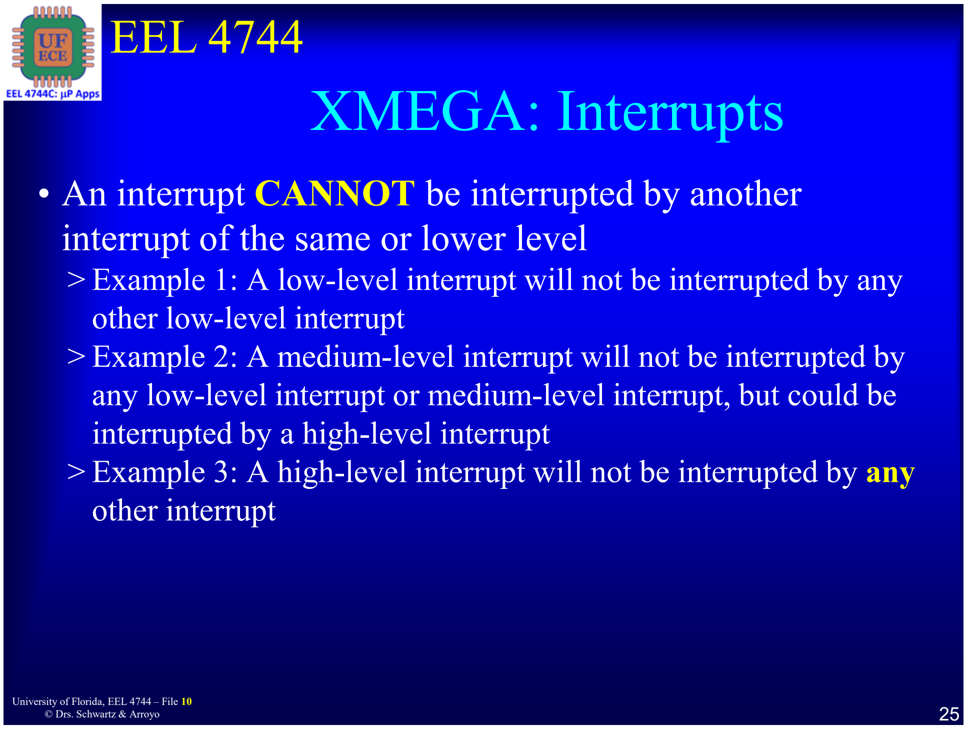


EEL 4744

doc8331: Section 12

XMEGA: Interrupts

- The PMIC status register contains state information to ensure that the PMIC returns to the correct interrupt level after an RETI
  > Returning from an interrupt will return the PMIC to the state it had before entering the interrupt
- The status register (SREG) is **NOT saved automatically** upon an interrupt request (**unlike** most other processors)
- The RET (subroutine return) instruction can**not** be used when returning from the interrupt handler routine, as this will **not** return the PMIC to its correct state



EEL 4744

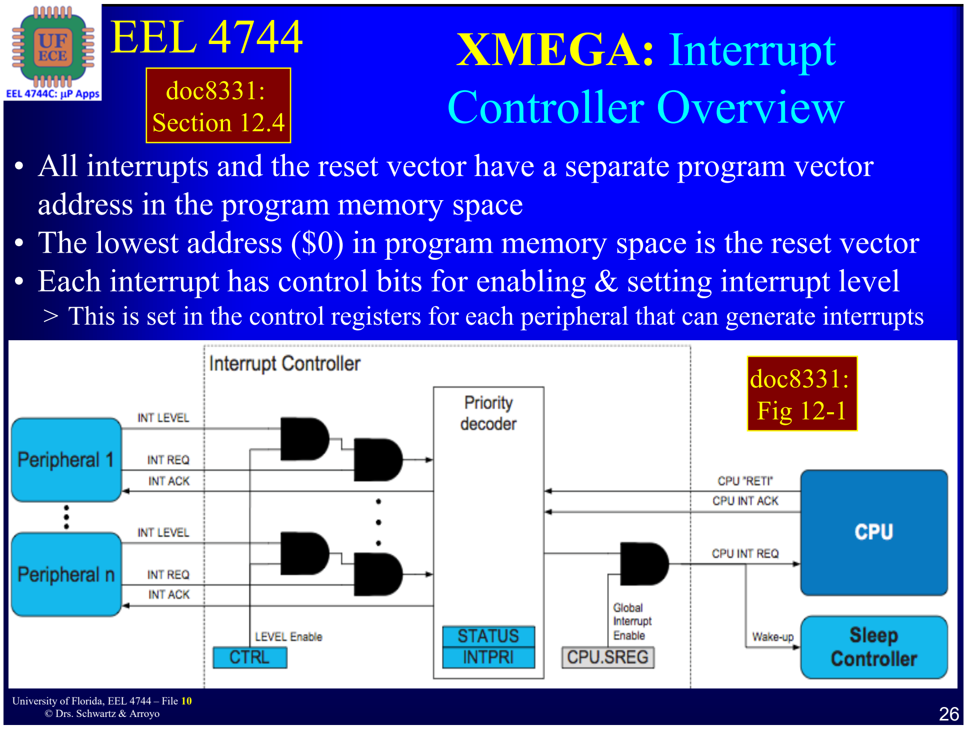
EEL 4744C: µP Apps

XMEGA: Interrupts

- An interrupt **CANNOT** be interrupted by another interrupt of the same or lower level
  - > Example 1: A low-level interrupt will not be interrupted by any other low-level interrupt
  - > Example 2: A medium-level interrupt will not be interrupted by any low-level interrupt or medium-level interrupt, but could be interrupted by a high-level interrupt
  - > Example 3: A high-level interrupt will not be interrupted by **any** other interrupt

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo 25

25



EEL 4744

EEL 4744C: µP Apps

doc8331: Section 12.4

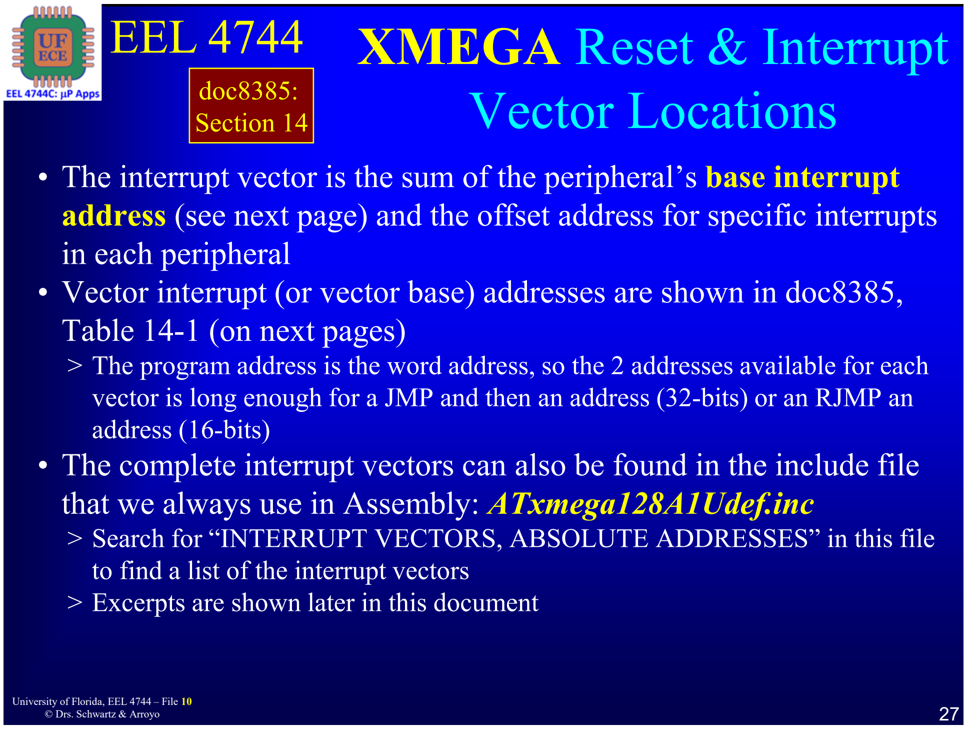
XMEGA: Interrupt Controller Overview

- All interrupts and the reset vector have a separate program vector address in the program memory space
- The lowest address ($0) in program memory space is the reset vector
- Each interrupt has control bits for enabling & setting interrupt level
  - > This is set in the control registers for each peripheral that can generate interrupts

doc8331: Fig 12-1

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo 26

26

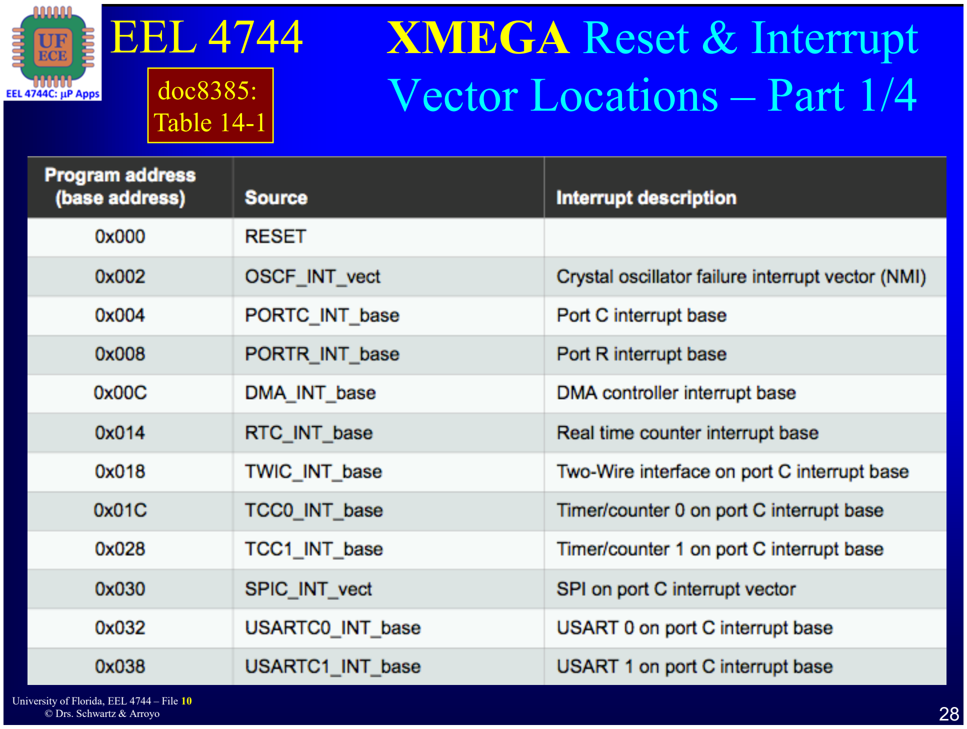


EEL 4744

doc8385: Section 14

XMEGA Reset & Interrupt Vector Locations

- The interrupt vector is the sum of the peripheral’s **base interrupt address** (see next page) and the offset address for specific interrupts in each peripheral
- Vector interrupt (or vector base) addresses are shown in doc8385, Table 14-1 (on next pages)
  > The program address is the word address, so the 2 addresses available for each vector is long enough for a JMP and then an address (32-bits) or an RJMP an address (16-bits)
- The complete interrupt vectors can also be found in the include file that we always use in Assembly: ***ATxmega128A1Udef.inc***
  > Search for “INTERRUPT VECTORS, ABSOLUTE ADDRESSES” in this file to find a list of the interrupt vectors
  > Excerpts are shown later in this document

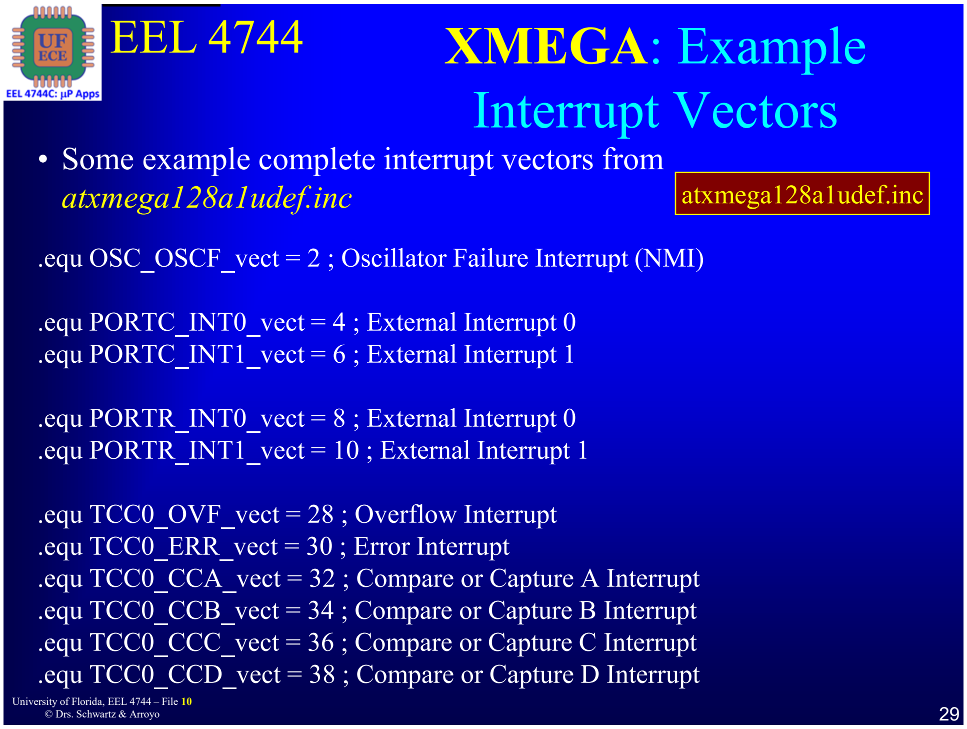


EEL 4744

doc8385: Table 14-1

XMEGA Reset & Interrupt Vector Locations – Part 1/4

Program address (base address)	Source	Interrupt description
0x000	RESET	
0x002	OSCF_INT_vect	Crystal oscillator failure interrupt vector (NMI)
0x004	PORTC_INT_base	Port C interrupt base
0x008	PORTR_INT_base	Port R interrupt base
0x00C	DMA_INT_base	DMA controller interrupt base
0x014	RTC_INT_base	Real time counter interrupt base
0x018	TWIC_INT_base	Two-Wire interface on port C interrupt base
0x01C	TCC0_INT_base	Timer/counter 0 on port C interrupt base
0x028	TCC1_INT_base	Timer/counter 1 on port C interrupt base
0x030	SPIC_INT_vect	SPI on port C interrupt vector
0x032	USARTC0_INT_base	USART 0 on port C interrupt base
0x038	USARTC1_INT_base	USART 1 on port C interrupt base



EEL 4744

XMEGA: Example Interrupt Vectors

- Some example complete interrupt vectors from *atxmega128a1udef.inc*

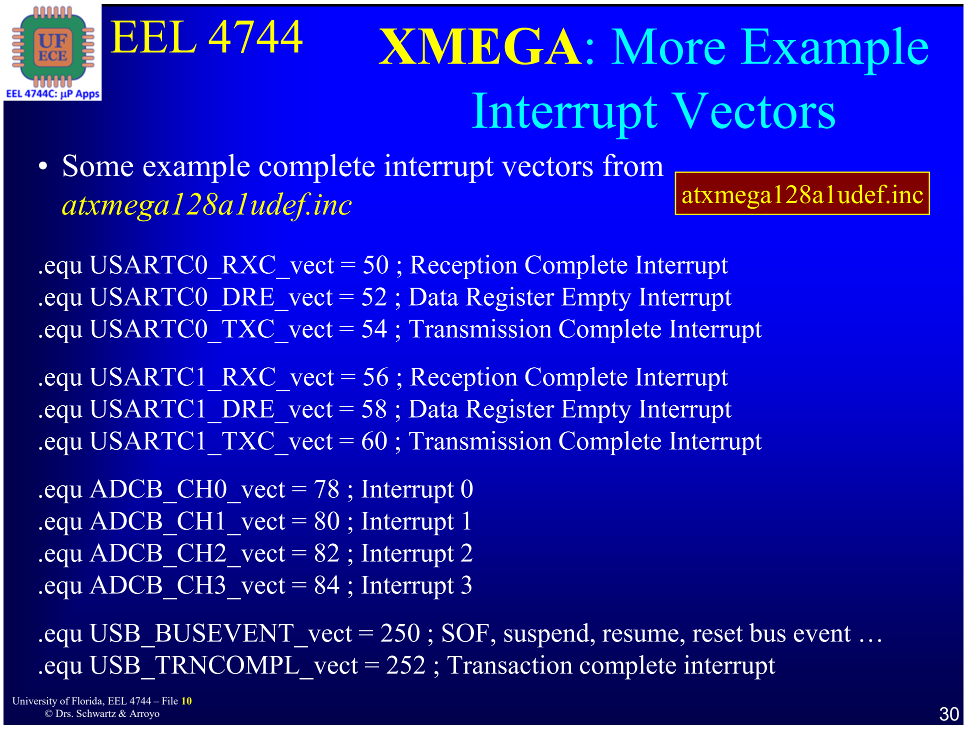
atxmega128a1udef.inc

```
.equ OSC_OSCF_vect = 2 ; Oscillator Failure Interrupt (NMI)

.equ PORTC_INT0_vect = 4 ; External Interrupt 0
.equ PORTC_INT1_vect = 6 ; External Interrupt 1

.equ PORTR_INT0_vect = 8 ; External Interrupt 0
.equ PORTR_INT1_vect = 10 ; External Interrupt 1

.equ TCC0_OVF_vect = 28 ; Overflow Interrupt
.equ TCC0_ERR_vect = 30 ; Error Interrupt
.equ TCC0_CCA_vect = 32 ; Compare or Capture A Interrupt
.equ TCC0_CCB_vect = 34 ; Compare or Capture B Interrupt
.equ TCC0_CCC_vect = 36 ; Compare or Capture C Interrupt
.equ TCC0_CCD_vect = 38 ; Compare or Capture D Interrupt
```



EEL 4744

XMEGA: More Example Interrupt Vectors

- Some example complete interrupt vectors from *atxmega128a1udef.inc*

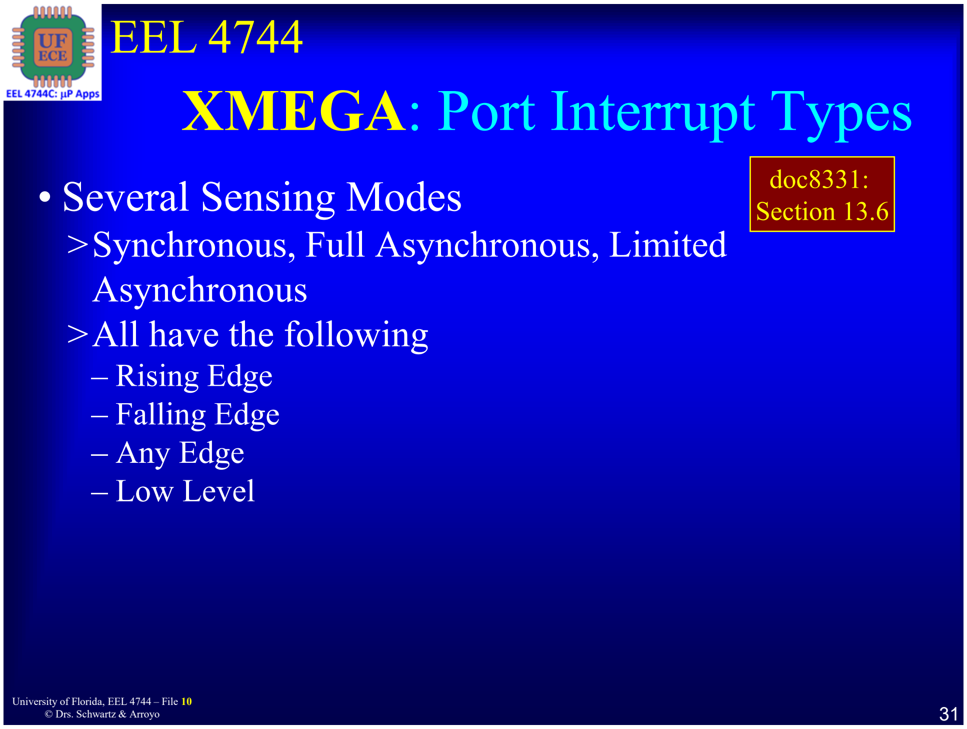
atxmega128a1udef.inc

```
.equ USARTC0_RXC_vect = 50 ; Reception Complete Interrupt
.equ USARTC0_DRE_vect = 52 ; Data Register Empty Interrupt
.equ USARTC0_TXC_vect = 54 ; Transmission Complete Interrupt

.equ USARTC1_RXC_vect = 56 ; Reception Complete Interrupt
.equ USARTC1_DRE_vect = 58 ; Data Register Empty Interrupt
.equ USARTC1_TXC_vect = 60 ; Transmission Complete Interrupt

.equ ADCB_CH0_vect = 78 ; Interrupt 0
.equ ADCB_CH1_vect = 80 ; Interrupt 1
.equ ADCB_CH2_vect = 82 ; Interrupt 2
.equ ADCB_CH3_vect = 84 ; Interrupt 3

.equ USB_BUSEVENT_vect = 250 ; SOF, suspend, resume, reset bus event …
.equ USB_TRNCOMPL_vect = 252 ; Transaction complete interrupt
```

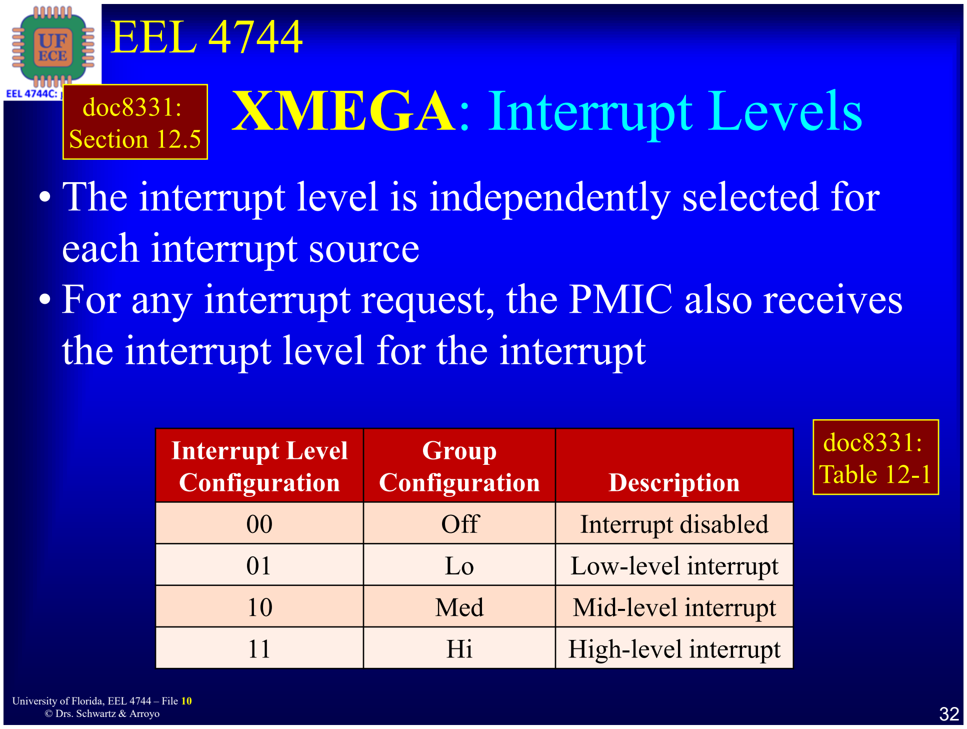


EEL 4744

XMEGA: Port Interrupt Types

doc8331:
Section 13.6

- Several Sensing Modes
  - >Synchronous, Full Asynchronous, Limited Asynchronous
  - >All have the following
    - – Rising Edge
    - – Falling Edge
    - – Any Edge
    - – Low Level



EEL 4744

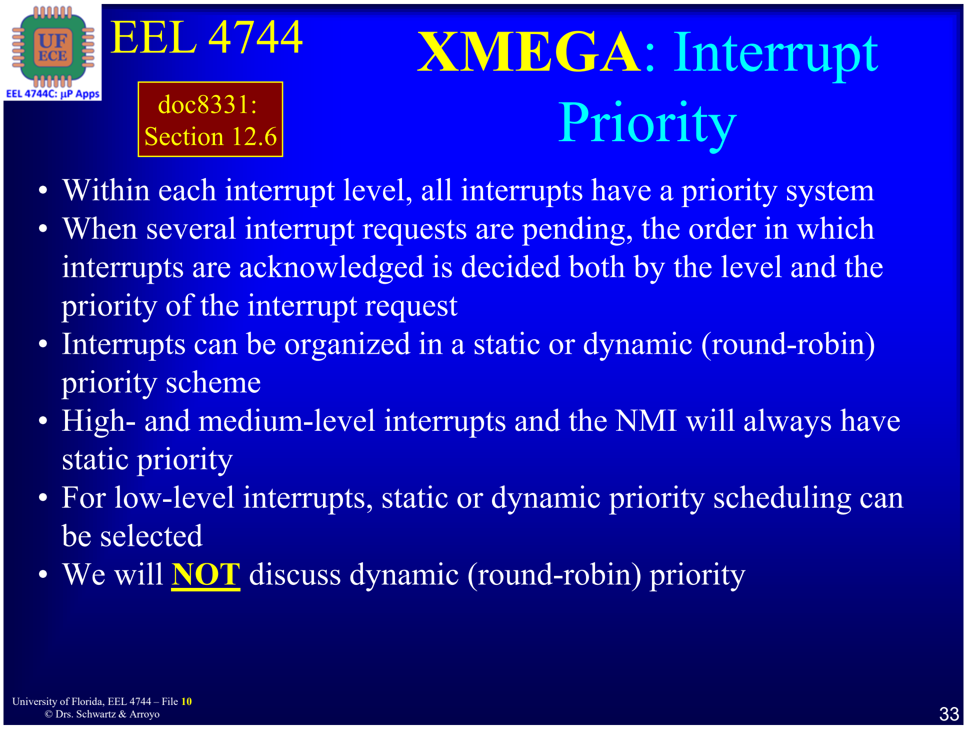
doc8331:
Section 12.5

XMEGA: Interrupt Levels

- The interrupt level is independently selected for each interrupt source
- For any interrupt request, the PMIC also receives the interrupt level for the interrupt

Interrupt Level Configuration	Group Configuration	Description
00	Off	Interrupt disabled
01	Lo	Low-level interrupt
10	Med	Mid-level interrupt
11	Hi	High-level interrupt

doc8331:
Table 12-1



EEL 4744C: µP Apps

EEL 4744

doc8331: Section 12.6

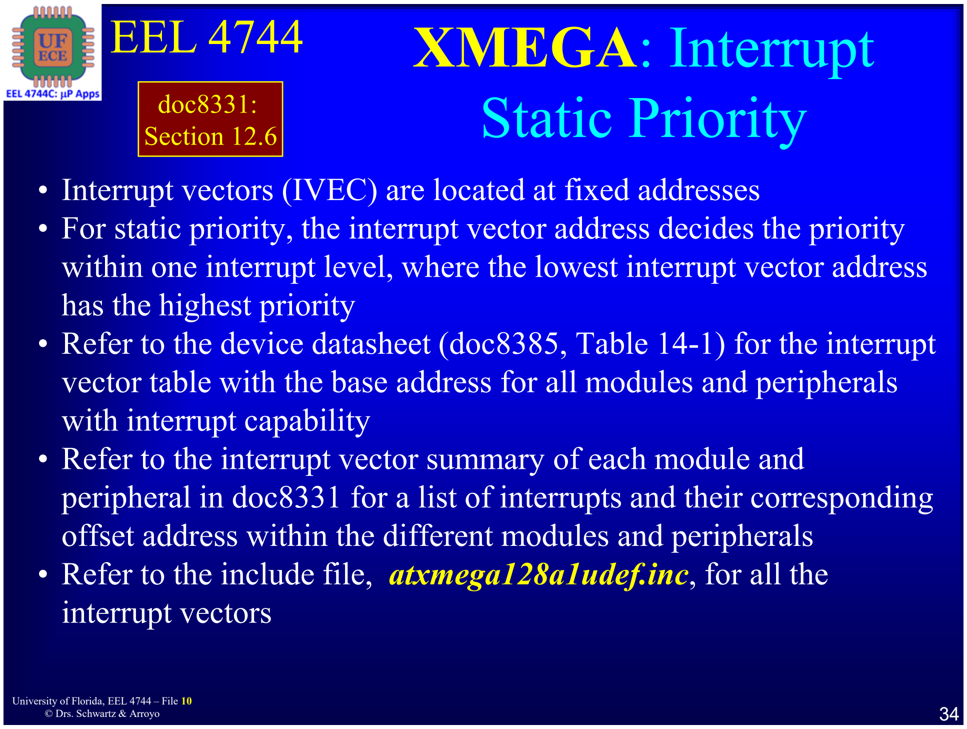
XMEGA: Interrupt Priority

- Within each interrupt level, all interrupts have a priority system
- When several interrupt requests are pending, the order in which interrupts are acknowledged is decided both by the level and the priority of the interrupt request
- Interrupts can be organized in a static or dynamic (round-robin) priority scheme
- High- and medium-level interrupts and the NMI will always have static priority
- For low-level interrupts, static or dynamic priority scheduling can be selected
- We will **NOT** discuss dynamic (round-robin) priority

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

33

33



EEL 4744C: µP Apps

EEL 4744

doc8331: Section 12.6

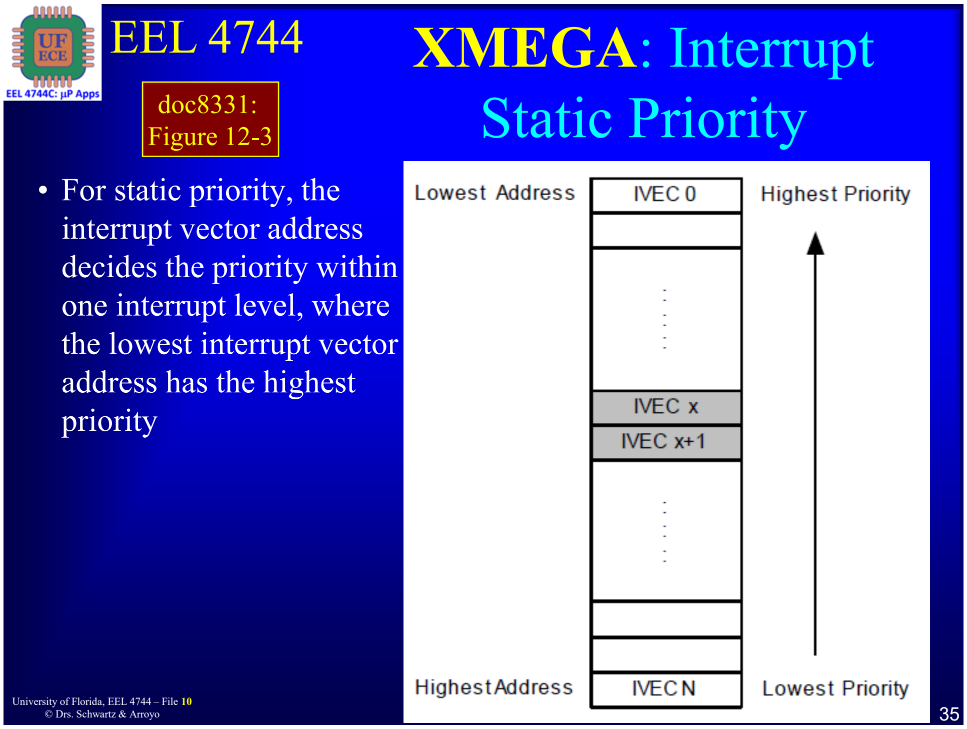
XMEGA: Interrupt Static Priority

- Interrupt vectors (IVEC) are located at fixed addresses
- For static priority, the interrupt vector address decides the priority within one interrupt level, where the lowest interrupt vector address has the highest priority
- Refer to the device datasheet (doc8385, Table 14-1) for the interrupt vector table with the base address for all modules and peripherals with interrupt capability
- Refer to the interrupt vector summary of each module and peripheral in doc8331 for a list of interrupts and their corresponding offset address within the different modules and peripherals
- Refer to the include file, ***atxmega128a1udef.inc***, for all the interrupt vectors

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

34

34



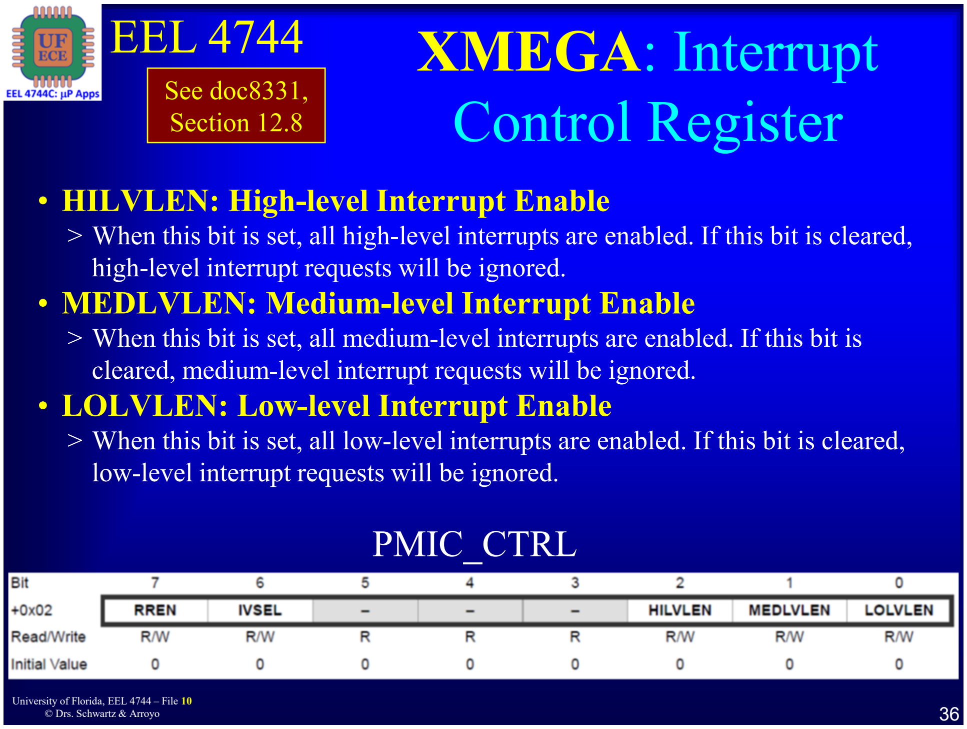
EEL 4744

doc8331: Figure 12-3

XMEGA: Interrupt Static Priority

- For static priority, the interrupt vector address decides the priority within one interrupt level, where the lowest interrupt vector address has the highest priority

Lowest Address
IVEC 0
Highest Priority
IVEC x
IVEC x+1
Highest Address
IVEC N
Lowest Priority



EEL 4744

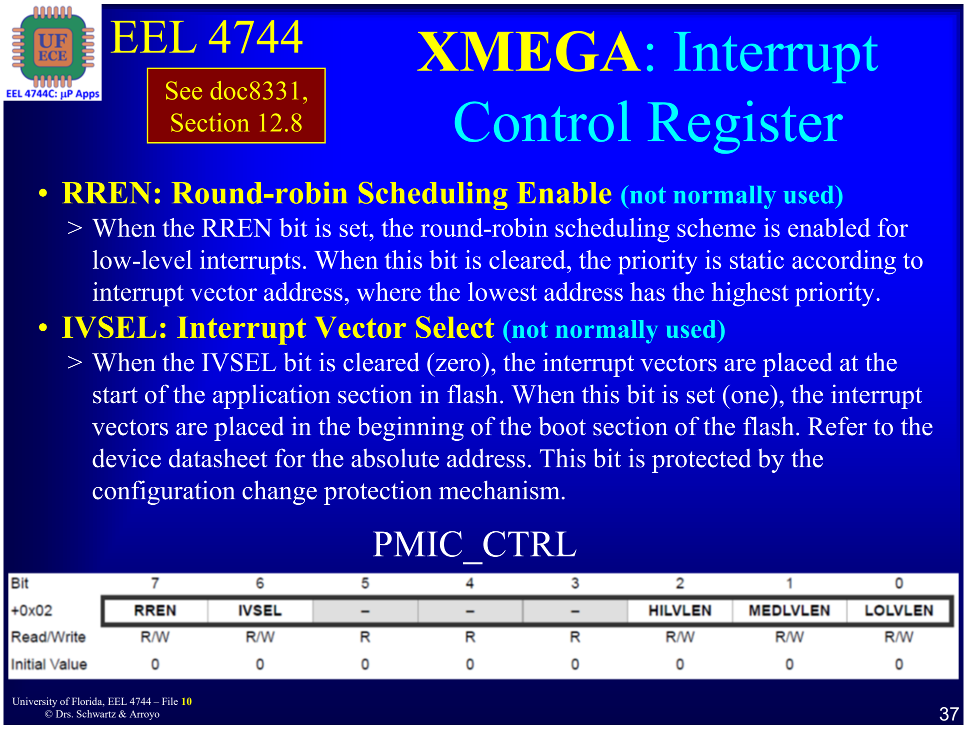
See doc8331, Section 12.8

XMEGA: Interrupt Control Register

- **HILVLEN: High-level Interrupt Enable**
  > When this bit is set, all high-level interrupts are enabled. If this bit is cleared, high-level interrupt requests will be ignored.
- **MEDLVLEN: Medium-level Interrupt Enable**
  > When this bit is set, all medium-level interrupts are enabled. If this bit is cleared, medium-level interrupt requests will be ignored.
- **LOLVLEN: Low-level Interrupt Enable**
  > When this bit is set, all low-level interrupts are enabled. If this bit is cleared, low-level interrupt requests will be ignored.

PMIC_CTRL

Bit	7	6	5	4	3	2	1	0
+0x02	RREN	IVSEL	–	–	–	HILVLEN	MEDLVLEN	LOLVLEN
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0



EEL 4744

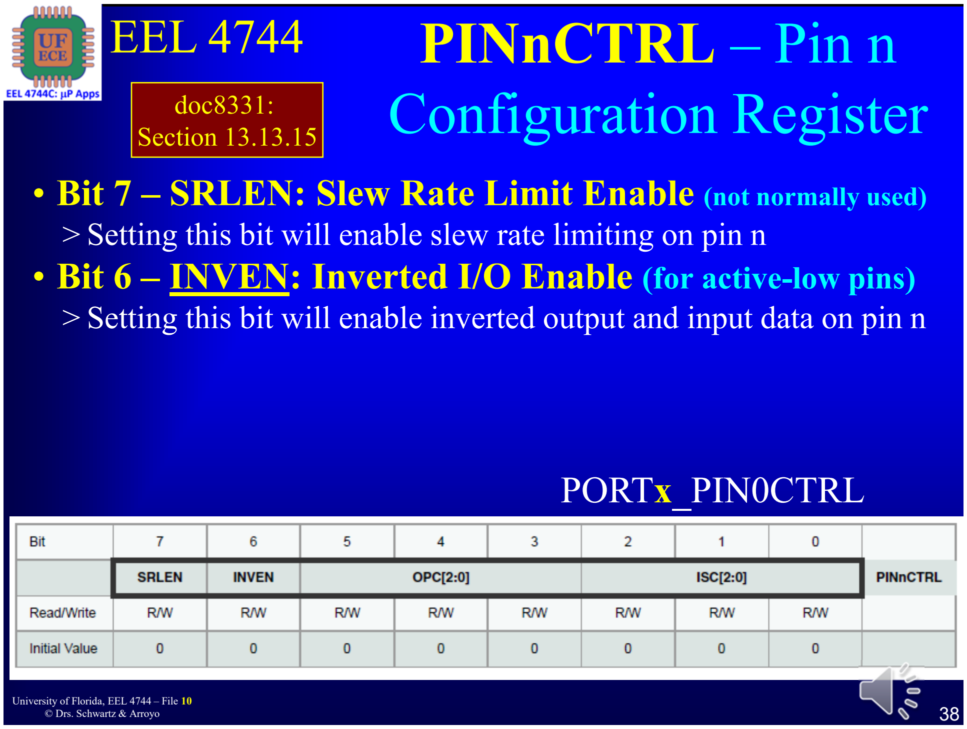
See doc8331, Section 12.8

XMEGA: Interrupt Control Register

- **RREN: Round-robin Scheduling Enable** **(not normally used)**
  > When the RREN bit is set, the round-robin scheduling scheme is enabled for low-level interrupts. When this bit is cleared, the priority is static according to interrupt vector address, where the lowest address has the highest priority.
- **IVSEL: Interrupt Vector Select** **(not normally used)**
  > When the IVSEL bit is cleared (zero), the interrupt vectors are placed at the start of the application section in flash. When this bit is set (one), the interrupt vectors are placed in the beginning of the boot section of the flash. Refer to the device datasheet for the absolute address. This bit is protected by the configuration change protection mechanism.

PMIC_CTRL

Bit	7	6	5	4	3	2	1	0
+0x02	RREN	IVSEL	–	–	–	HILVLEN	MEDLVLEN	LOLVLEN
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0



EEL 4744

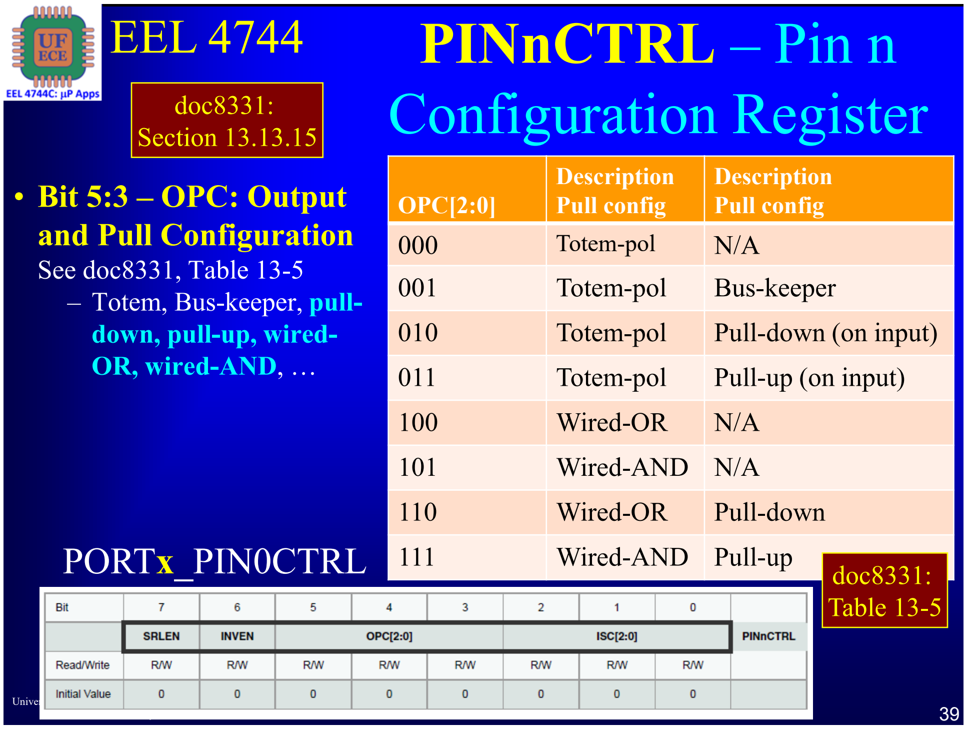
doc8331: Section 13.13.15

PINnCTRL – Pin n Configuration Register

- **Bit 7 – SRLEN: Slew Rate Limit Enable** **(not normally used)**
  > Setting this bit will enable slew rate limiting on pin n
- **Bit 6 – INVEN: Inverted I/O Enable** **(for active-low pins)**
  > Setting this bit will enable inverted output and input data on pin n

PORTx_PIN0CTRL

Bit	7	6	5	4	3	2	1	0	
	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			PINnCTRL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



EEL 4744

doc8331: Section 13.13.15

PINnCTRL – Pin n Configuration Register

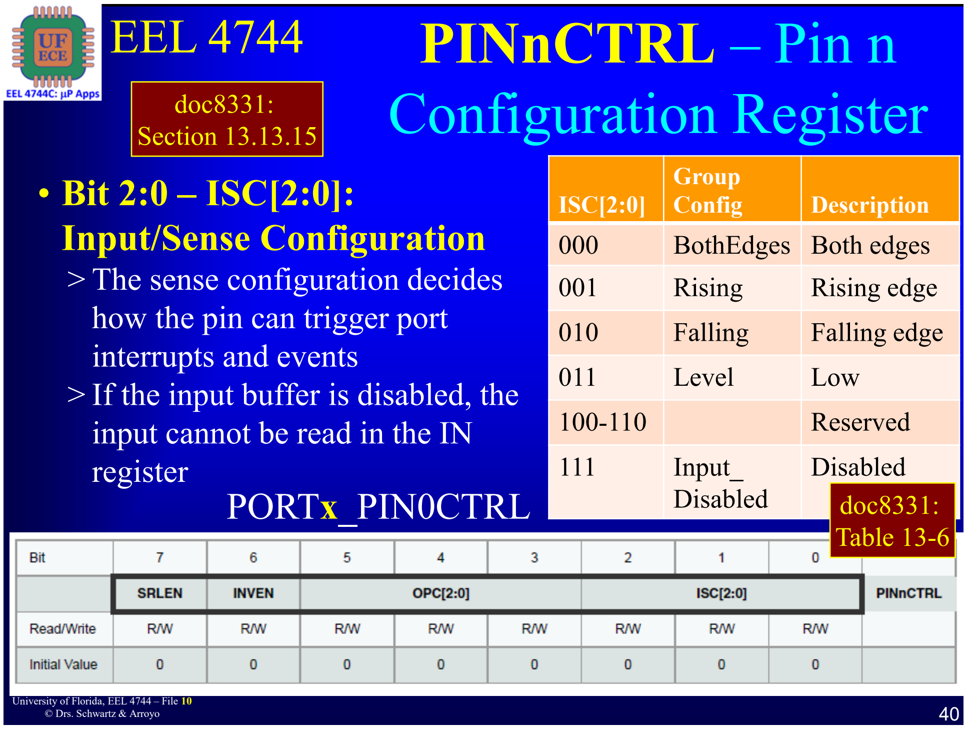
- **Bit 5:3 – OPC: Output and Pull Configuration**
  See doc8331, Table 13-5
  - Totem, Bus-keeper, **pull-down, pull-up, wired-OR, wired-AND**, …

OPC[2:0]	Description Pull config	Description Pull config
000	Totem-pol	N/A
001	Totem-pol	Bus-keeper
010	Totem-pol	Pull-down (on input)
011	Totem-pol	Pull-up (on input)
100	Wired-OR	N/A
101	Wired-AND	N/A
110	Wired-OR	Pull-down
111	Wired-AND	Pull-up

doc8331: Table 13-5

PORT**x**_PIN0CTRL

Bit	7	6	5	4	3	2	1	0	
	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			PINnCTRL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



EEL 4744

doc8331: Section 13.13.15

PINnCTRL – Pin n Configuration Register

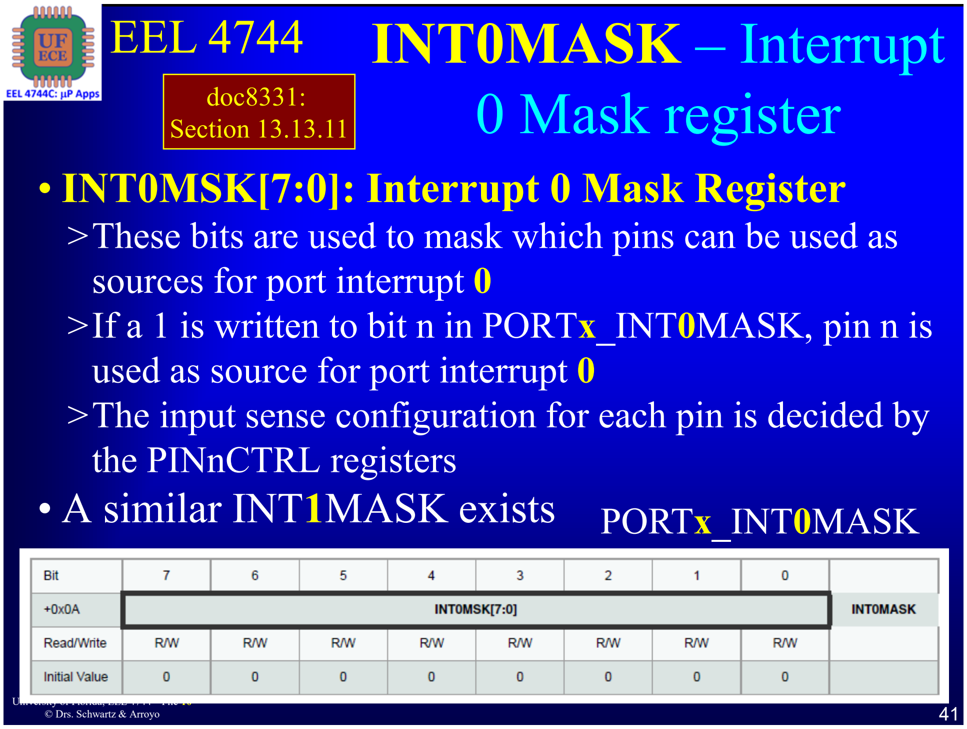
- **Bit 2:0 – ISC[2:0]: Input/Sense Configuration**
  > The sense configuration decides how the pin can trigger port interrupts and events
  > If the input buffer is disabled, the input cannot be read in the IN register

ISC[2:0]	Group Config	Description
000	BothEdges	Both edges
001	Rising	Rising edge
010	Falling	Falling edge
011	Level	Low
100-110		Reserved
111	Input_ Disabled	Disabled

doc8331: Table 13-6

PORT**x**_PIN0CTRL

Bit	7	6	5	4	3	2	1	0	
	SRLEN	INVEN	OPC[2:0]			ISC[2:0]			PINnCTRL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



EEL 4744

doc8331: Section 13.13.11

INT0MASK – Interrupt 0 Mask register

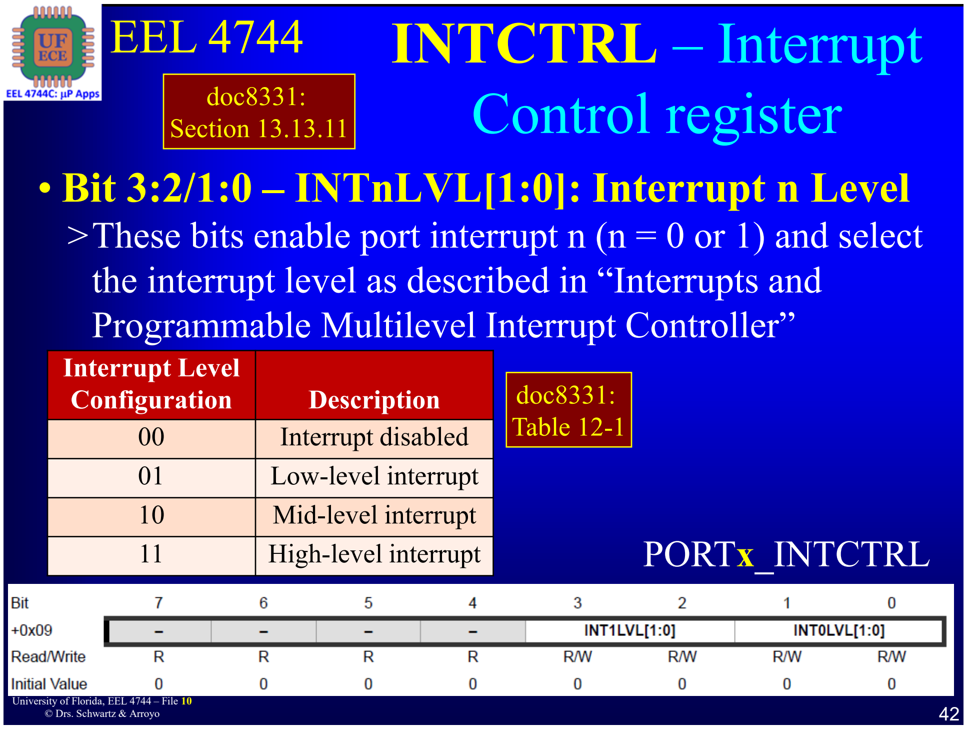
- **INT0MSK[7:0]: Interrupt 0 Mask Register**
  - >These bits are used to mask which pins can be used as sources for port interrupt **0**
  - >If a 1 is written to bit n in PORT**x**_INT**0**MASK, pin n is used as source for port interrupt **0**
  - >The input sense configuration for each pin is decided by the PINnCTRL registers
- A similar INT**1**MASK exists

PORT**x**_INT**0**MASK

Bit	7	6	5	4	3	2	1	0	
+0x0A	INT0MSK[7:0]								INT0MASK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

© Drs. Schwartz & Arroyo 41

41



EEL 4744

doc8331: Section 13.13.11

INTCTRL – Interrupt Control register

- **Bit 3:2/1:0 – INTnLVL[1:0]: Interrupt n Level**
  - >These bits enable port interrupt n (n = 0 or 1) and select the interrupt level as described in “Interrupts and Programmable Multilevel Interrupt Controller”

Interrupt Level Configuration	Description
00	Interrupt disabled
01	Low-level interrupt
10	Mid-level interrupt
11	High-level interrupt

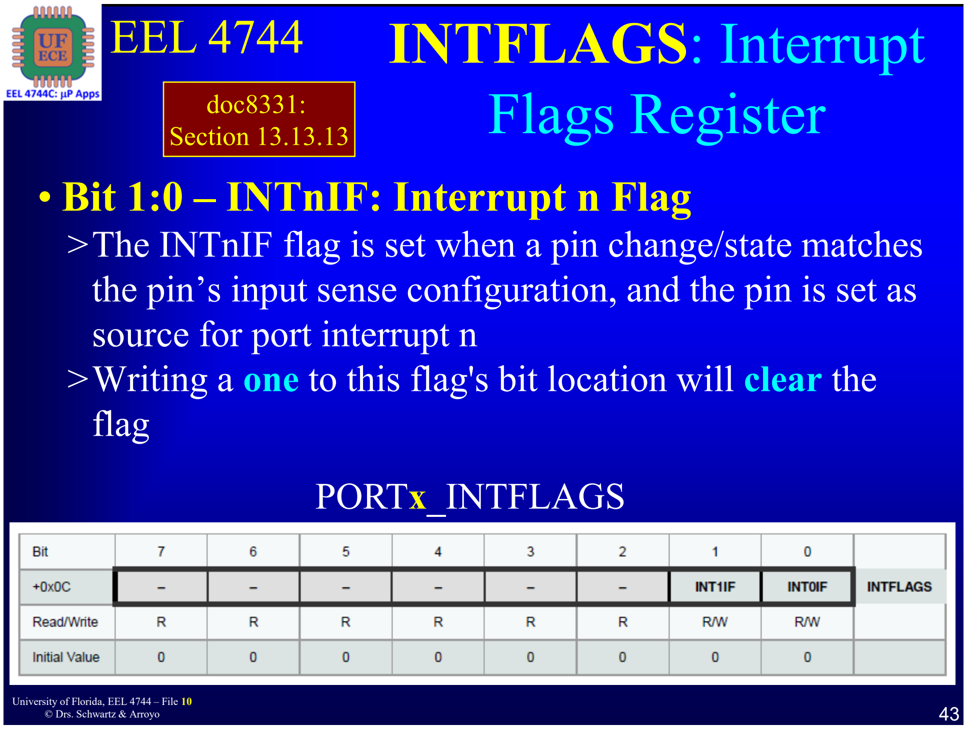
doc8331: Table 12-1

PORT**x**_INTCTRL

Bit	7	6	5	4	3	2	1	0
+0x09	–	–	–	–	INT1LVL[1:0]		INT0LVL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

University of Florida, EEL 4744 – File **10**
© Drs. Schwartz & Arroyo 42

42



EEL 4744

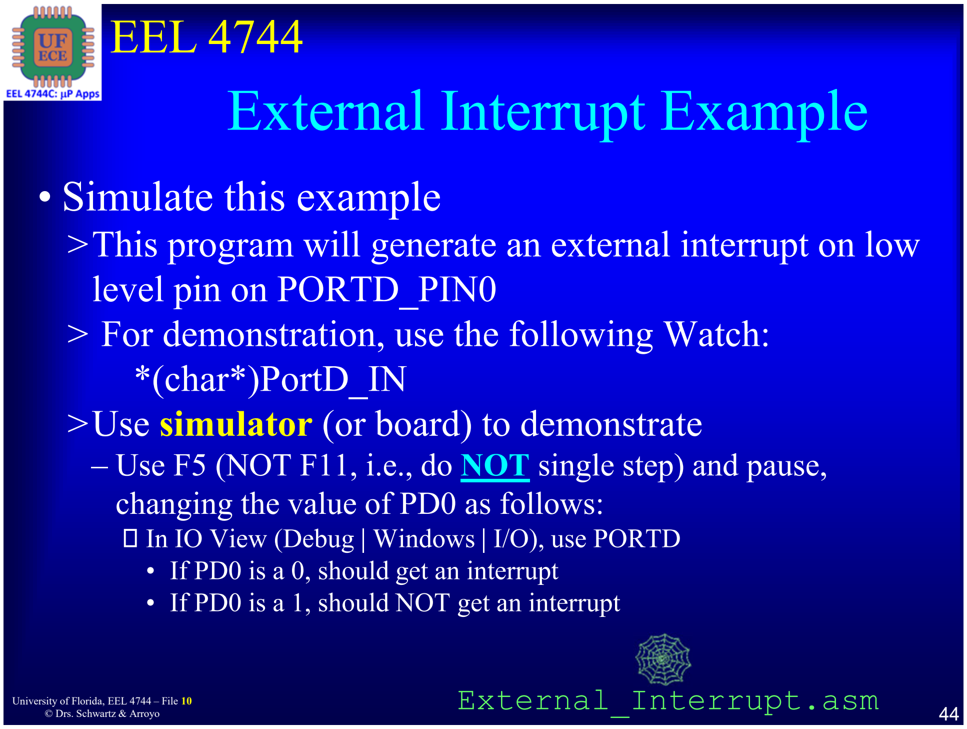
INTFLAGS: Interrupt Flags Register

doc8331: Section 13.13.13

- **Bit 1:0 – INTnIF: Interrupt n Flag**
  - >The INTnIF flag is set when a pin change/state matches the pin's input sense configuration, and the pin is set as source for port interrupt n
  - >Writing a **one** to this flag's bit location will **clear** the flag

PORT**x**_INTFLAGS

Bit	7	6	5	4	3	2	1	0	
+0x0C	–	–	–	–	–	–	INT1IF	INT0IF	INTFLAGS
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

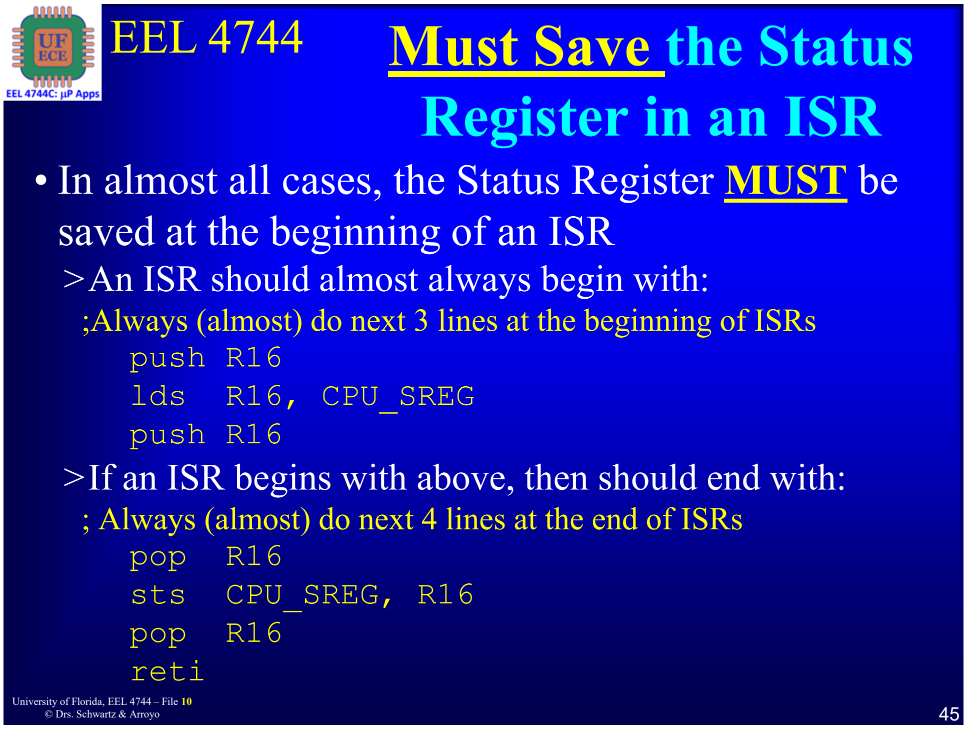


EEL 4744

External Interrupt Example

- Simulate this example
  - >This program will generate an external interrupt on low level pin on PORTD_PIN0
  - > For demonstration, use the following Watch:
    (char)PortD_IN
  - >Use **simulator** (or board) to demonstrate
    - – Use F5 (NOT F11, i.e., do **NOT** single step) and pause, changing the value of PD0 as follows:
      - In IO View (Debug | Windows | I/O), use PORTD
        - If PD0 is a 0, should get an interrupt
        - If PD0 is a 1, should NOT get an interrupt

External_Interrupt.asm



EEL 4744

Must Save the Status Register in an ISR

- In almost all cases, the Status Register **MUST** be saved at the beginning of an ISR
  - >An ISR should almost always begin with:

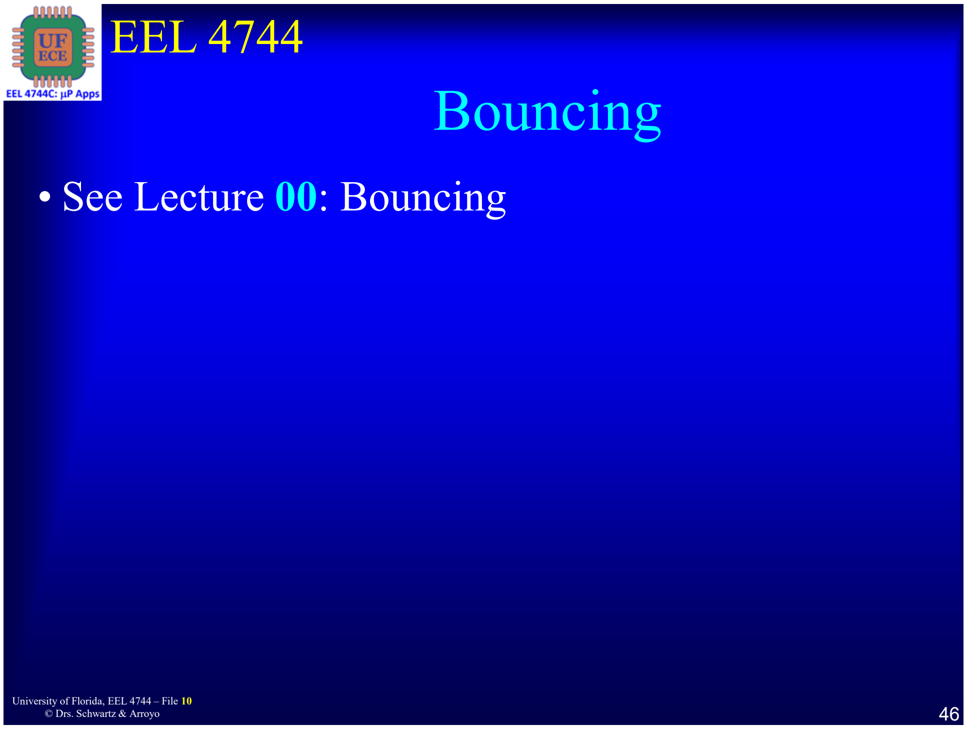
```
;Always (almost) do next 3 lines at the beginning of ISRs
    push R16
    lds  R16, CPU_SREG
    push R16
```

  - >If an ISR begins with above, then should end with:

```
; Always (almost) do next 4 lines at the end of ISRs
    pop  R16
    sts  CPU_SREG, R16
    pop  R16
    reti
```

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

45



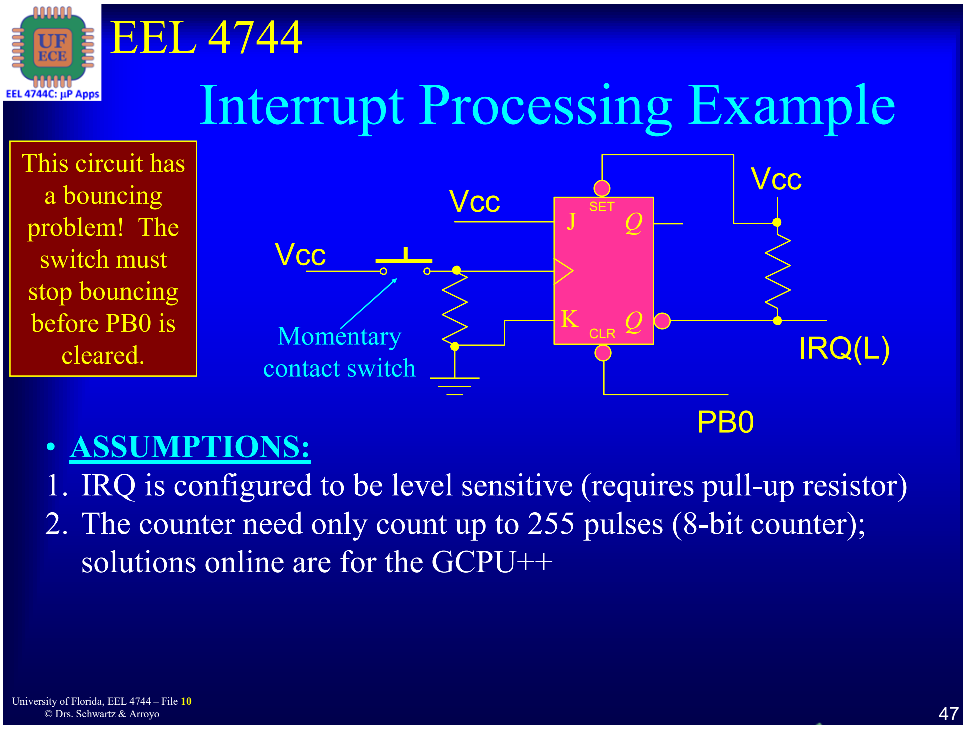
EEL 4744

Bouncing

- See Lecture **00**: Bouncing

University of Florida, EEL 4744 – File 10
© Drs. Schwartz & Arroyo

46



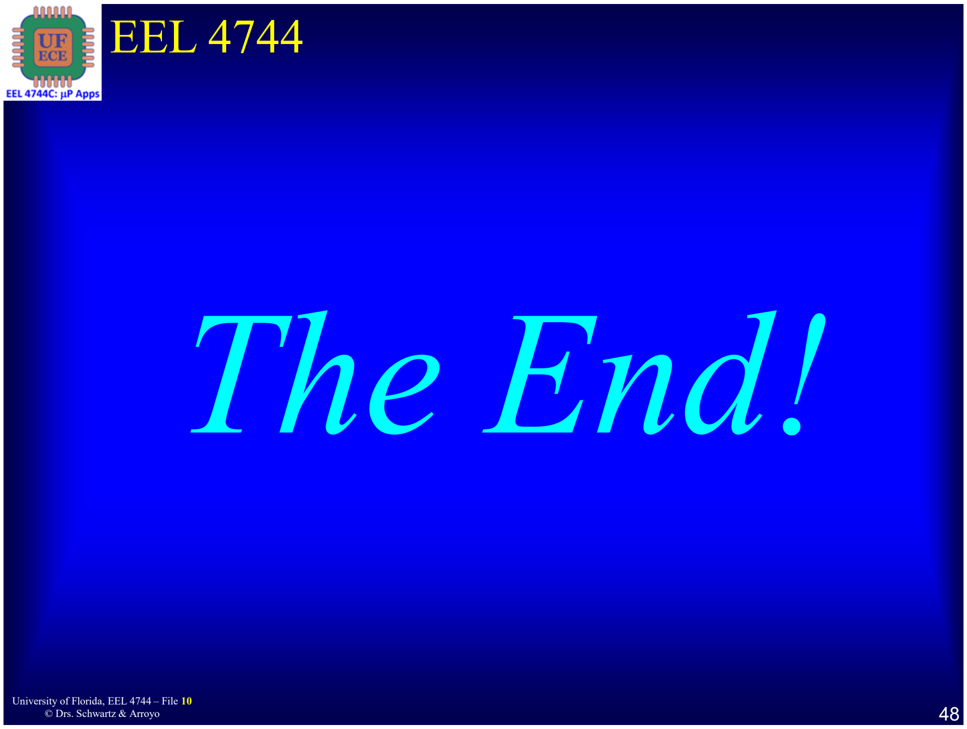
EEL 4744

Interrupt Processing Example

This circuit has a bouncing problem! The switch must stop bouncing before PB0 is cleared.

- **ASSUMPTIONS:**

1. IRQ is configured to be level sensitive (requires pull-up resistor)
2. The counter need only count up to 255 pulses (8-bit counter); solutions online are for the GCPU++



EEL 4744

The End!