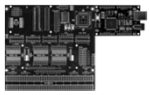
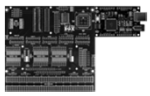


EEL 3744

Menu

- Interrupt motivation (polling example)
- Interrupt description
- Interrupt-driven I/O
  - > Interrupt Protocol
  - > Preemptive-Resume Priority Interrupts
  - > Vectored Interrupt
- Resets and Interrupts in XMEGA, 68HC11-12
  - > 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

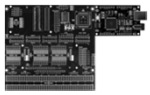
See readings & examples on web-site:
`Textbook (ch 10), doc8331 (sec 12-14), doc8385 (sec 14), External_Interrupt.asm`



EEL 3744

Polling Example

- You used polling in lab 2 when you were 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, …

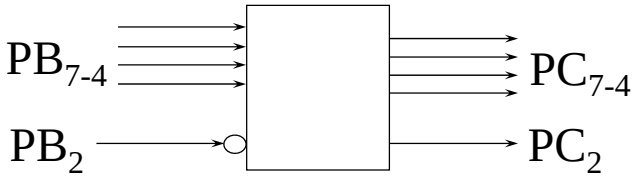


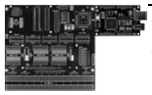
EEL 3744

Polling Example

Example:
Read in a 4-bit value from PB7-PB4, when PB2=0; CPU indicates ready status by setting PC2. CPU writes value of PB7-PB4 to PC7-PC4.

- Processor sets PC_2 to indicate ready for new data
- Wait for PB_2 to go true, then read the 4-bits at PB_7-PB_4
- Copy the read values at PB_7-PB_4 to at PC_7-PC_4
- Processor sets PC_2 to indicate ready for new data





EEL 3744

Interrupt Description

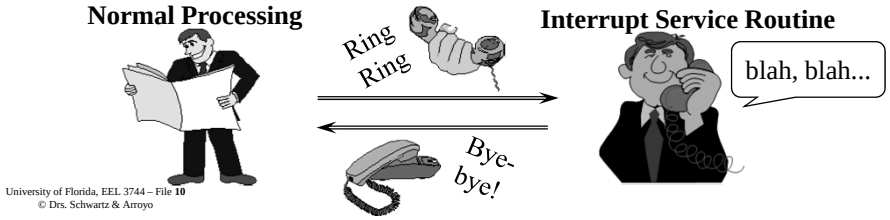
- In the interrupt-driven alternative to polling, a peripheral or an I/O device that is ready for service indicates so by "**interrupting**" the μP
- If the device is allowed to interrupt, then the μP will complete the execution of the current instruction, save the processor status (**the CPU registers, 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 68HC11/12: PCL, PCH, YL, YH, XL, XH, A, B, and CCR

EEL 3744 Interrupt Description

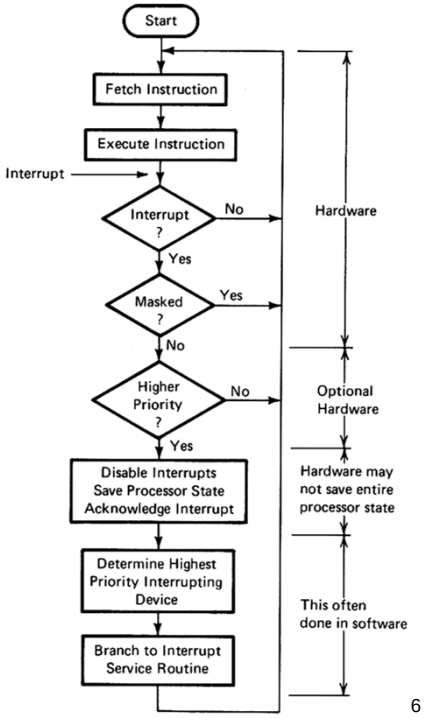
- At the completion of the interrupt-service routine, the processor status is restored from the stack by executing a return from interrupt (e.g., RETI [or RTI or IRET]) instruction
  - >This returns control to the program instruction that was originally scheduled to be executed before the interrupt was acknowledged
- The μP then resumes its normal processing

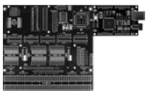


EEL 3744 Interrupt-driven I/O

- Processing interrupts

Doty: Fig 6.7-6

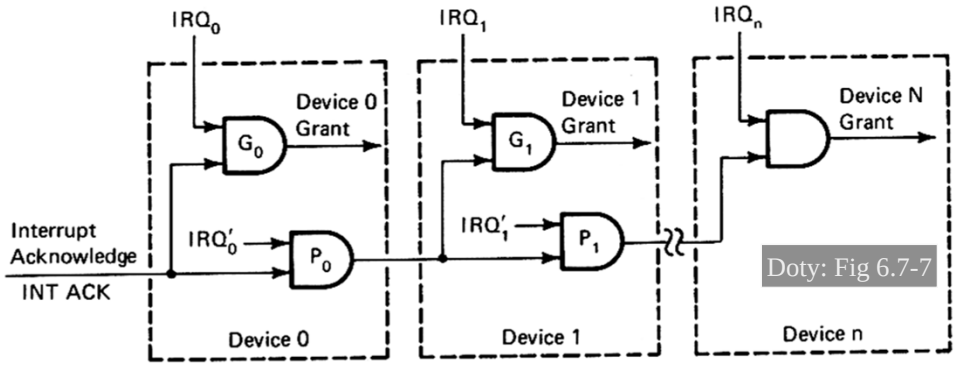




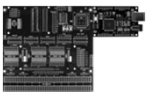
EEL 3744

Interrupt-driven I/O

- Daisy-chain priority and polling logic



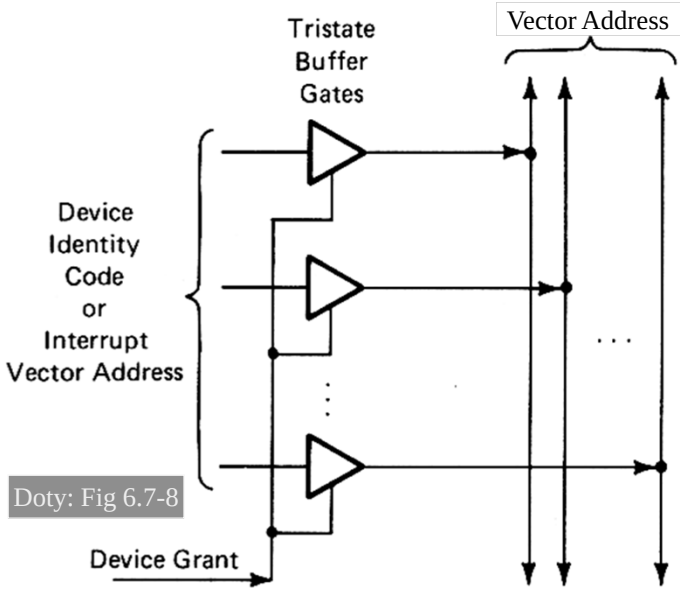
Doty: Fig 6.7-7



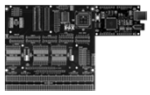
EEL 3744

Interrupt-driven I/O

- Generation of a device interrupt vector address



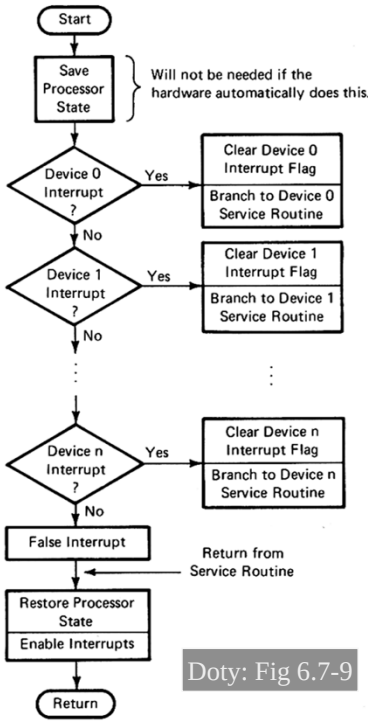
Doty: Fig 6.7-8



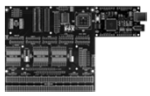
EEL 3744

Interrupt-driven I/O

- Determining interrupting device through software polling



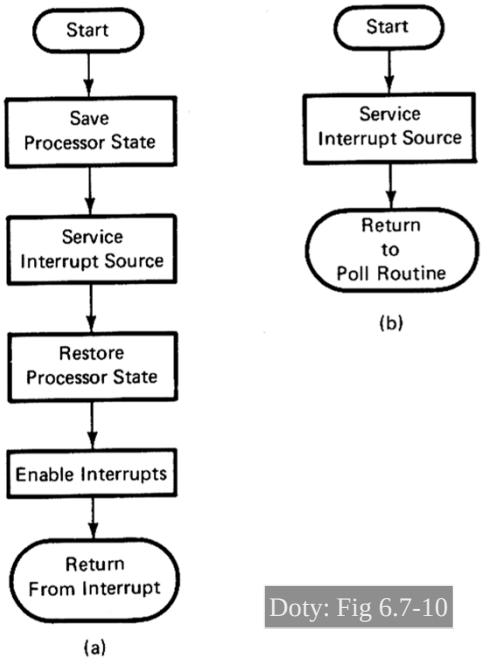
Doty: Fig 6.7-9



EEL 3744

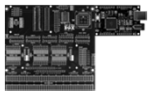
Interrupt-driven I/O

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



Doty: Fig 6.7-10

University of Florida, EEL 3744 – File **10**
© Drs. Schwartz & Arroyo

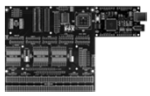


EEL 3744

68HC11
Interrupt
Promotion Table

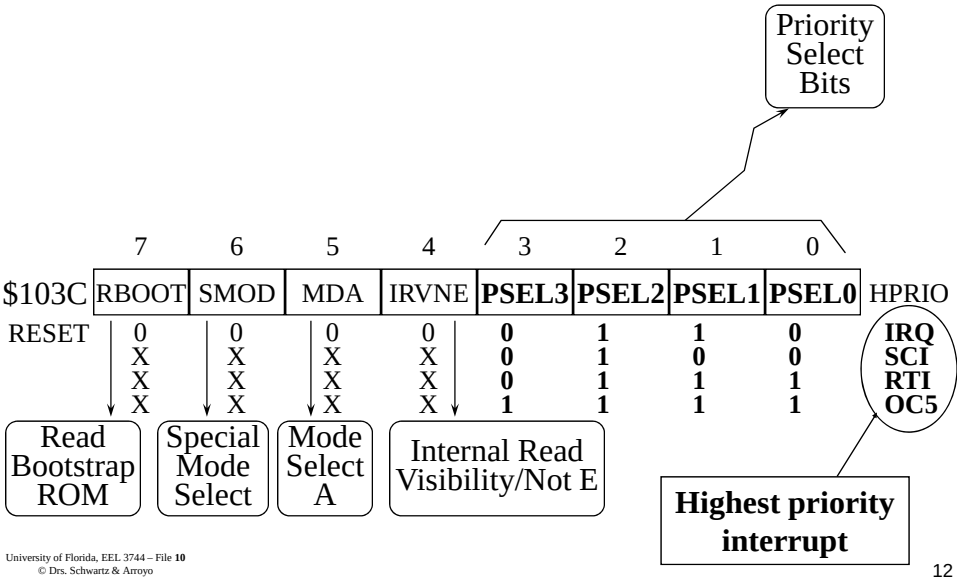
- Reset & Interrupt Priority
  - > HPRIO - Highest priority I-Bit interrupt using (PSEL**i**)

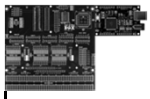
PSEL[3:0]	Interrupt Source Promoted
0 0 0 0	Timer Overflow
0 0 0 1	Pulse Accumulator Overflow
0 0 1 0	Pulse Accumulator Input Edge
0 0 1 1	SPI Serial Transfer Complete
0 1 0 0	SCI Serial System
0 1 0 1	Reserved (Default to $\overline{\text{IRQ}}$)
0 1 1 0	$\overline{\text{IRQ}}$ (External Pin or Parallel I/O)
0 1 1 1	Real-Time Interrupt
1 0 0 0	Timer Input Capture 1
1 0 0 1	Timer Input Capture 2
1 0 1 0	Timer Input Capture 3
1 0 1 1	Timer Output Compare 1
1 1 0 0	Timer Output Compare 2
1 1 0 1	Timer Output Compare 3
1 1 1 0	Timer Output Compare 4
1 1 1 1	Timer Input Capture 4/Output Compare 5



EEL 3744

Resets and Interrupts in **68HC11**





EEL 3744

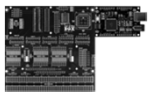
68HC11 Interrupt and Reset Vectors

Bible: Page 3

TD: Table 5-4

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

13



EEL 3744

68HC11 EVBU Interrupt Pseudo-Vectors (with BUFFALO)

BUFFALO memory dump:

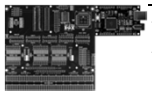
```
FFD6: 00 C4
FFD8: 00 C7 00 CA 00 CD 00 D0
FFE0: 00 D3 00 D6 00 D9 00 DC
FFE8: 00 DF 00 E2 00 E5 00 E8
FFF0: 00 EB 00 EE 00 F1 00 F4
FFF8: 00 F7 00 FA 00 FD B6 00
```

Reset Pseudo Vector

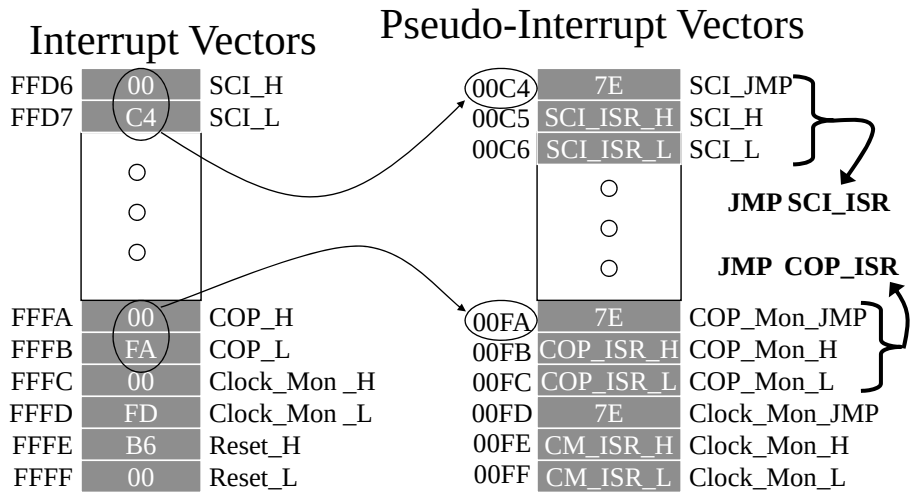
Pseudo Vector	Interrupt Source
$00C4-$00C6	Serial Comm. Interface (SCI)
$00C7-$00C9	Serial Peripheral Interface (SPI)
$00CA-$00CC	Pulse Accumulator Input Edge
$00CD-$00CF	Pulse Accumulator Overflow
$00DO-$00D2	Timer Overflow
$00D3-$00D5	Timer Output Compare 5
$00D6-$00D8	Timer Output Compare 4
$00D9-$00DB	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

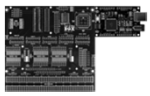
Vector Addr	Interrupt Source
FFD6, D7	Serial Comm. Interface (SCI)
FFE0, E1	Timer Output Compare 5
FFF0, F1	Real Time Interrupt
FFF2, F3	IRQ
FFFE, FF	RESET

14



EEL 3744 Interrupt Vectors & Interrupt Pseudo-vectors for **68HC11** with BUFFALO





EEL 3744 **68HC11** SCI Interrupts

- Make a pseudo-vector (for BUFFALO) for the SCI interrupt service routine:

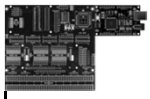
```
ORG     $????
JMP     SCI_ISR
```

Pseudo-vector (with BUFFALO): $00C4

- Make vector (for no BUFFALO) for the SCI interrupt service routine:

```
ORG   $????
DC.W  IRQ_ISR
```

Vector without BUFFALO: $FFD6

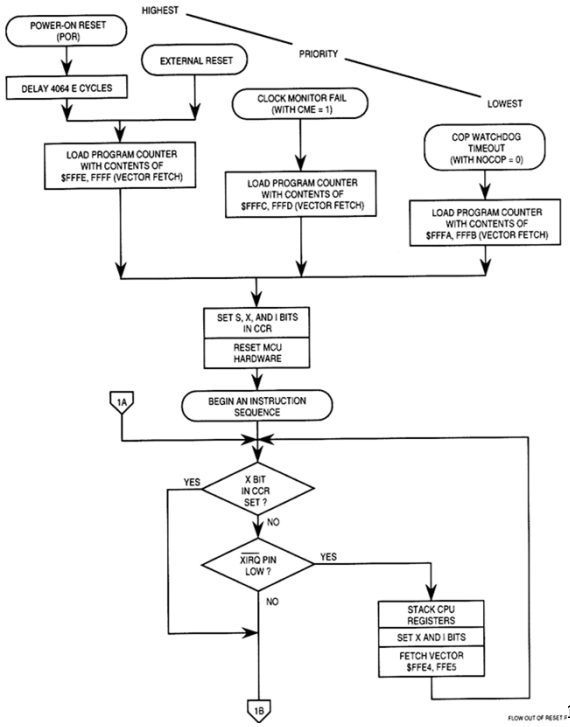


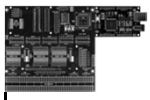
EEL 3744

68HC11 Resets and Interrupts Flow Chart (1/2)

- Processing flow out of reset (part a)

TD: Fig 5-1a
(Ref: Fig 5-2)



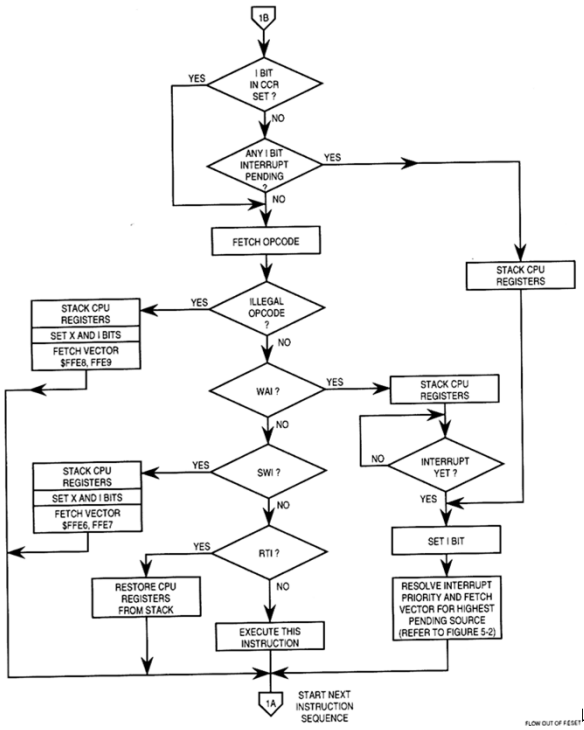


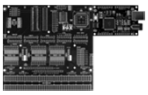
EEL 3744

68HC11 Resets and Interrupts Flow Chart (2/2)

- Processing flow out of reset (part b)

TD: Fig 5-1b



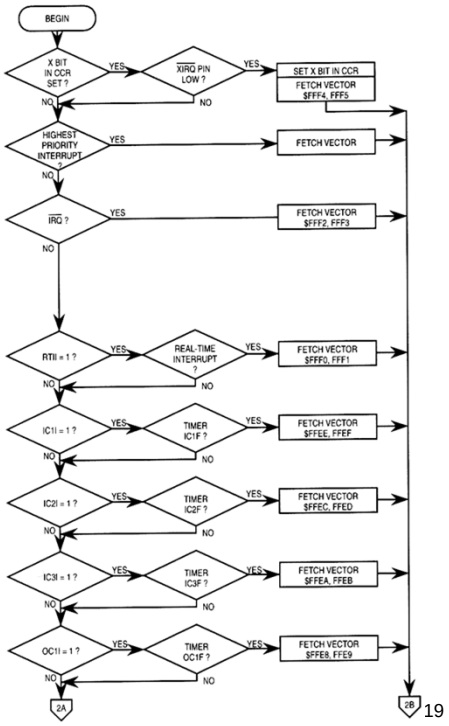


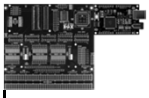
EEL 3744

68HC11 Interrupt Priority Resolution (1/2)

- Interrupt priority resolution

TD: Fig 5-2a
(Ref: Fig 5-3)

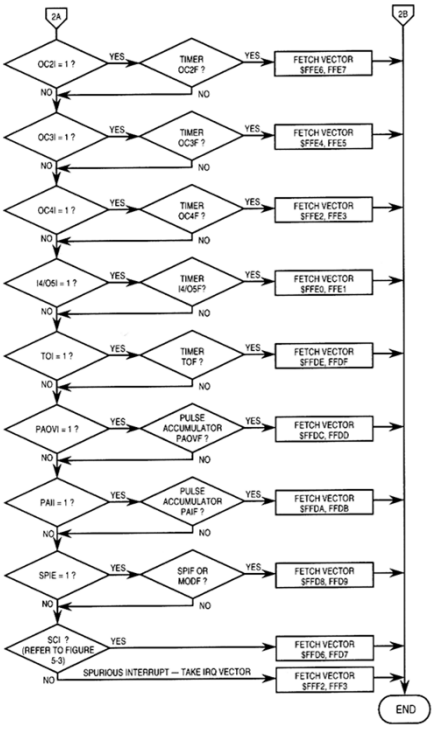




EEL 3744

68HC11 Interrupt Priority Resolution (2/2)

TD: Fig 5-2b



EEL 3744

68HC11
Interrupts
Source
Resolution in
SCI

TD: Fig 5-3

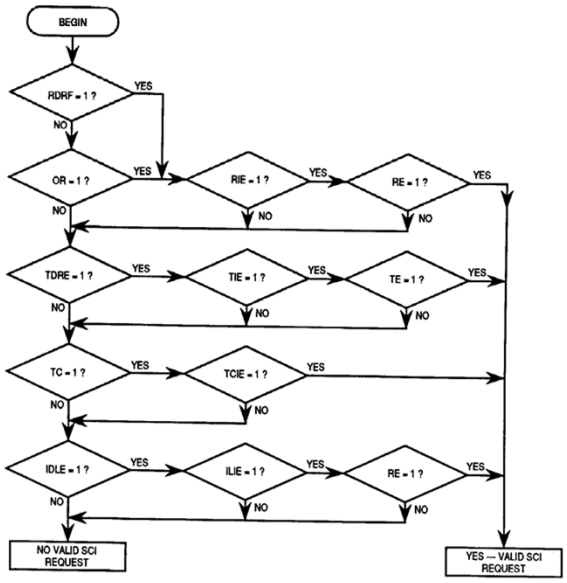


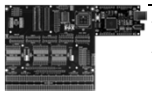
Figure 5–3. Interrupt Source Resolution Within SCI

EEL 3744

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

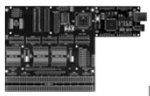


EEL 3744

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)

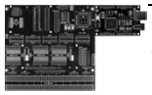


EEL 3744

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 most 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**

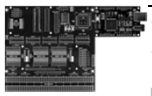


EEL 3744

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

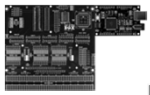


EEL 3744

doc8331: Section 3.14.9

I-Bit and SEI/CLI

- 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
    - – 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

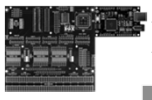


EEL 3744

doc8331: Section 12

XMEGA: Non-Maskable Interrupt (NMI)

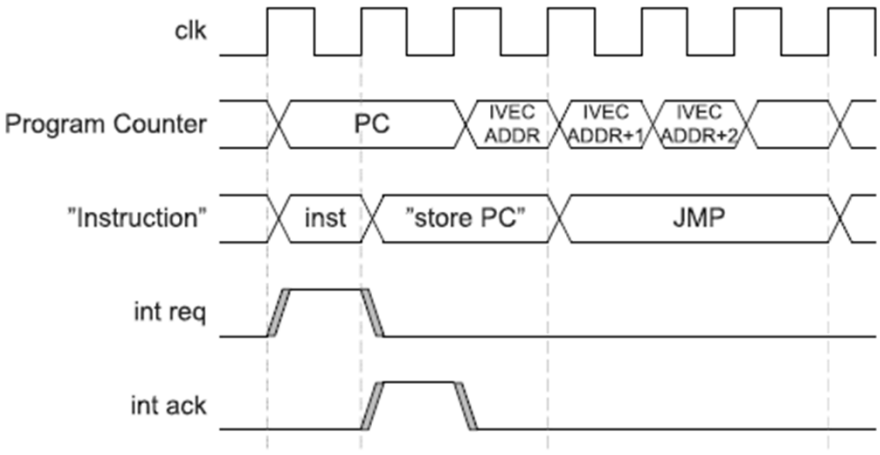
- Non-maskable interrupts must be enabled before they can be used
- An NMI will be executed regardless of the setting of the I bit, and it will never change the I bit
- No other interrupts can interrupt a NMI handler
- If more than one NMI is requested at the same time, priority is set according to the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

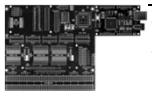


EEL 3744

doc8331: Figure 12-2

XMEGA: Interrupt execution of Instruction



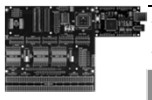


EEL 3744

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

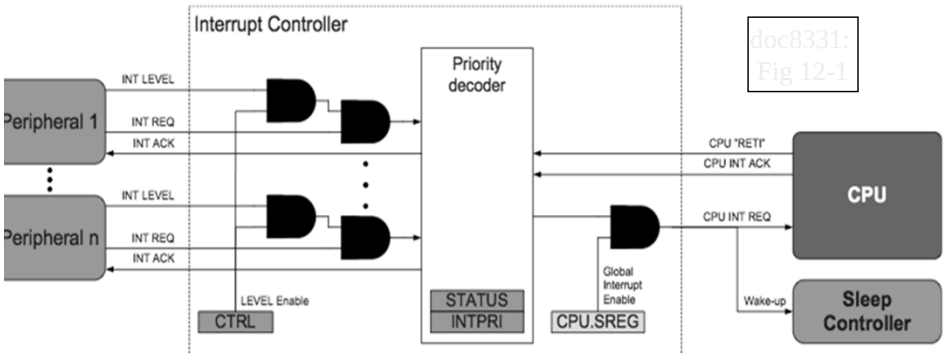


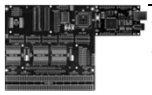
EEL 3744

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

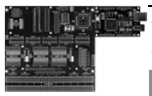




EEL 3744 **XMEGA** Reset & Interrupt Vector Locations

doc8385: Section 14

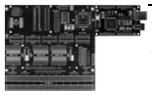
- The interrupt vector is the sum of the peripheral's **base interrupt address** (see next page) and the offset address for specific interrupts in each peripheral
- Vector interrupt (or vector base) addresses are shown in doc8385, Table 14-1 (on next pages)
  - > The program address is the word address, 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 3744 **XMEGA** Reset & Interrupt Vector Locations – Part 1/4

doc8385: Table 14-1

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 3744

XMEGA: Example Interrupt Vectors

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

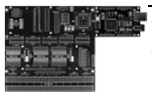
atxmega128a1udef.inc

```
.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 3744

XMEGA: More Example Interrupt Vectors

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

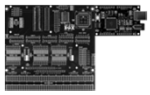
atxmega128a1udef.inc

```
.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
```

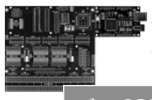


EEL 3744

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 3744

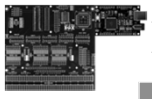
doc8331: Section 12.5

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

doc8331: Table 12-1

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

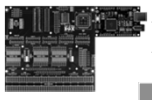


EEL 3744

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

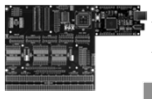


EEL 3744

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

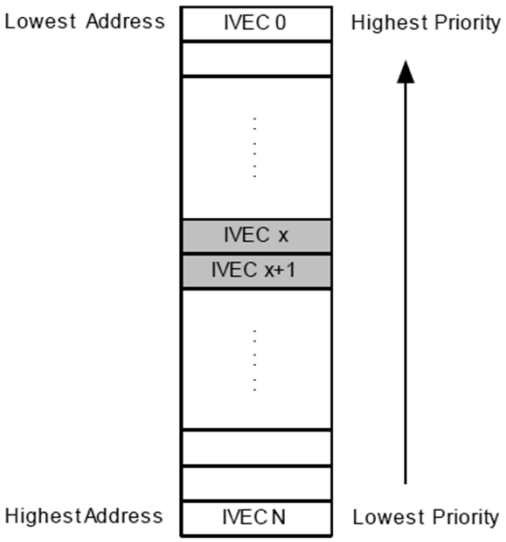


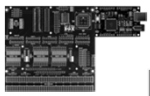
EEL 3744

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





EEL 3744

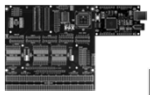
See doc8331, Section 12.8

XMEGA: Interrupt Status Register

- **NMIEX: Non-Maskable Interrupt Executing (not normally used)**
  - > This flag is set if a non-maskable interrupt is executing
  - > The flag will be cleared with RETI
- **HILVLEX: High-level Interrupt Executing (not normally used)**
  - > This flag is set when a high-level interrupt is executing or when the interrupt handler has been interrupted by an NMI
  - > The flag will be cleared with RETI

PMIC_STATUS

Bit	7	6	5	4	3	2	1	0	
+0x00	**NMIEX**	–	–	–	–	**HILVLEX**	**MEDLVLEX**	**LOLVLEX**	**STATUS**
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	



EEL 3744

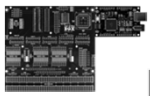
See doc8331, Section 12.8

XMEGA: Interrupt Status Register

- **MEDLVLEX: Medium-level Interrupt Executing (not normally used)**
  - > This flag is set when a medium-level interrupt is executing or when the interrupt handler has been interrupted by an interrupt from higher level or an NMI
  - > The flag will be cleared with an RETI
- **LOLVLEX: Low-level Interrupt Executing (not normally used)**
  - > This flag is set when a low-level interrupt is executing or when the interrupt handler has been interrupted by an interrupt from higher level or an NMI
  - > The flag will be cleared with an RETI

PMIC_STATUS

Bit	7	6	5	4	3	2	1	0	
+0x00	NMIEX	–	–	–	–	HILVLEX	MEDLVLEX	LOLVLEX	STATUS
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	



EEL 3744

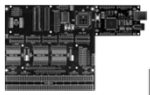
See doc8331, Section 12.8

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 3744

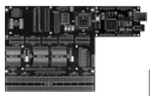
See doc8331, Section 12.8

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 3744

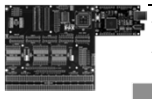
See doc8331, Section 12.8

XMEGA: Interrupt Priority Register

- **Bit 7:0 – INTPRI: Interrupt Priority (Not normally needed)**
  - > When round-robin scheduling is enabled, this register stores the interrupt vector of the last acknowledged low-level interrupt
  - > The stored interrupt vector will have the lowest priority the next time one or more low-level interrupts are pending
  - > This register is not reinitialized to its initial value if round-robing scheduling is disable
  - > If default static priority is needed, the register must be written to zero

PMIC_INTPRI

Bit	7	6	5	4	3	2	1	0
+0x01	INTPRI[7:0]							
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 3744

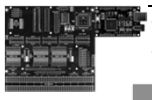
doc8331: Section 13.13.15

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

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 3744

doc8331: Section 13.13.15

PINnCTRL – Pin n Configuration Register

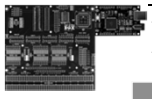
- **Bit 5:3 – OPC: Output and Pull Configuration**
  See doc8331, Table 13-5
  – Totem, Buskeeper, **pull-down, pull-up, wired-OR, wired-AND**, …

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 3744

PINnCTRL – Pin n Configuration Register

doc8331: Section 13.13.15

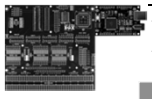
- **Bit 2:0 – ISC[2:0]: Input/Sense Configuration**
  - > The sense configuration decides how the pin can trigger port interrupts and events
  - > If the input buffer is disabled, the input cannot be read in the IN register

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

PORT**x**_PIN0CTRL

doc8331: Table 13-6

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 3744

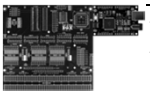
MPCMask – Multi-pin Configuration Mask Register

doc8331: Section 13.14.1

- **Bit 7:0 – MPCMASK[7:0]: Multi-pin Configuration Mask**
  - > The MPCMASK register enables configuration of several pins of a port **<u>at the same time</u>**
    - – Writing a one to bit **n** makes pin **n** part of the multi-pin configuration.
    - – When one or more bits in the MPCMASK register is set, writing any of the PIN**n**CTRL registers will update only the PIN**n**CTRL registers matching the mask in the MPCMASK register for that port
    - – The MPCMASK register is automatically cleared after any PIN**n**CTRL register is written

PORTCFG_MPCMASK

Bit	7	6	5	4	3	2	1	0
+0x00	MPCMASK[7:0]							
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



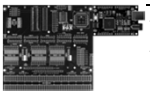
EEL 3744 **INT0MASK** – Interrupt 0 Mask register

doc8331: Section 13.13.11

- **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	



EEL 3744 **INTCTRL** – Interrupt Control register

doc8331: Section 13.13.11

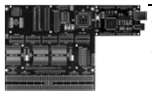
- **Bit 3:2/1:0 – INTnLVL[1:0]: Interrupt n Level**
  - >These bits enable port interrupt n (n = 0 or 1) and select the interrupt level as described in “Interrupts and Programmable Multilevel Interrupt Controller”

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



EEL 3744

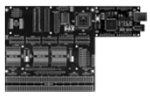
doc8331: Section 13.13.13

INTFLAGS: Interrupt Flags Register

- **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	

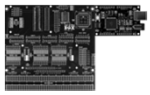


EEL 3744

External Interrupt Example

- Simulate this example
  - >This program will generate an external interrupt on low level 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, 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 3744

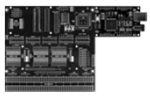
Must Save the Status Register in an ISR

- 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:

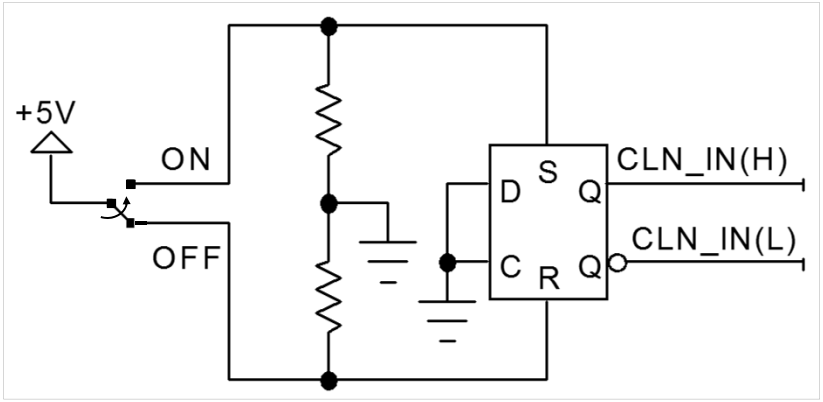
```
; Always (almost) do next 4 lines at the end of ISRs
   pop  R16
   sts  CPU_SREG, R16
   pop  R16
   reti
```

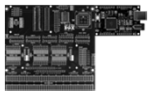
EEL 3744

debounce.cct

DEBOUNCING



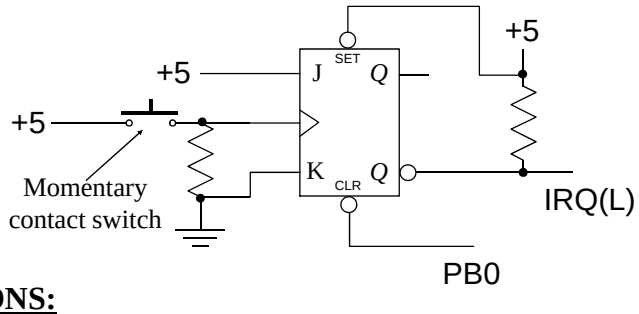
- Clean inputs can be made with various flip-flops or latches
- Above is a Delay (D) flip-flop used as a Set-Reset (S-R) latch



EEL 3744

IRQ Processing Example

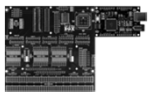
This circuit has a bouncing problem! The switch must stop bouncing before PB0 is cleared.



- **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)
3. Solutions online are for the 68HC12

`IRQ_Inta-d.asm`



EEL 3744

The End!