ECE3411 – Fall 2016

Lecture 3c.

# External Interrupts
# Pin Change Interrupts

**Marten van Dijk**
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lecture 3c, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

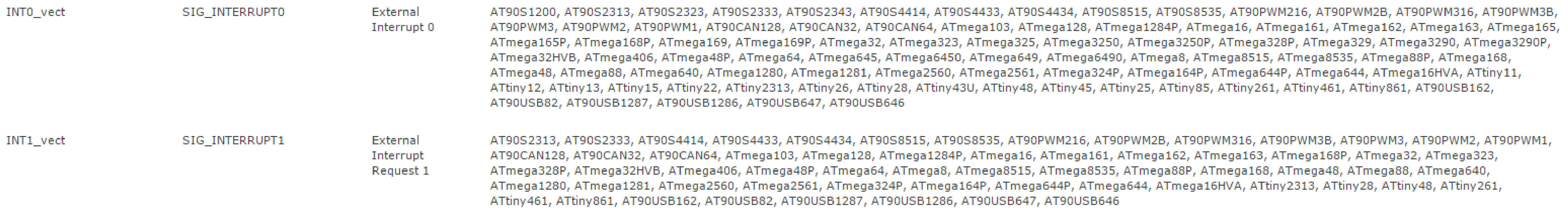
Based on the Atmega328P datasheet

# External Interrupts

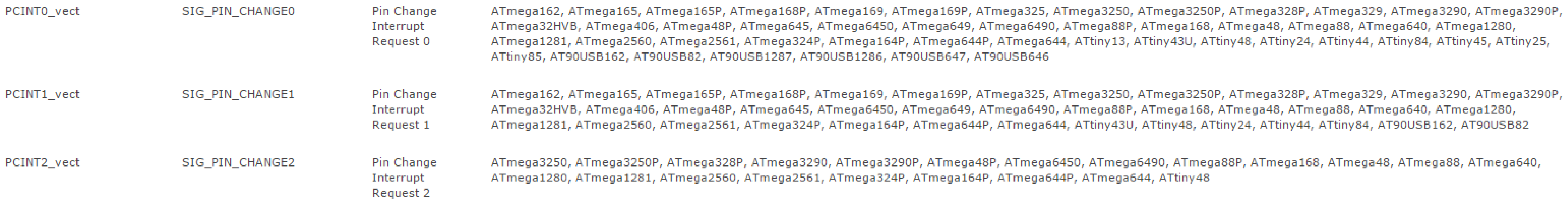
- Chapter 12 datasheet
- INT0 & INT1
  - Can be triggered by a falling or rising edge or a low level → EICRA (External Interrupt Control Register A)
  - Low level interrupt is detected asynchronously → can be used to wake from idle mode as well as sleep modes (will see one such example in a forthcoming lecture)
    - If used for wake-up from power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. (Start-up time defined by SUT and CKSEL fuses, chapter 8)
- PCINT23..0
  - The pin change interrupt PCI0 will trigger if any enabled PCINT7..0 pin toggles
  - The pin change interrupt PCI1 will trigger if any enabled PCINT14..8 pin toggles
  - The pin change interrupt PCI2 will trigger if any enabled PCINT23..16 pin toggles

# Interrupt Vectors

- http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__interrupts.html



| | | | |
|---|---|---|---|
| INT0_vect | SIG_INTERRUPT0 | External Interrupt 0 | AT90S1200, AT90S2313, AT90S2323, AT90S2333, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega406, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, ATmega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny11, ATtiny12, ATtiny13, ATtiny15, ATtiny22, ATtiny2313, ATtiny26, ATtiny28, ATtiny43U, ATtiny48, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646 |
| INT1_vect | SIG_INTERRUPT1 | External Interrupt Request 1 | AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega168P, ATmega32, ATmega323, ATmega328P, ATmega32HVB, ATmega406, ATmega48P, ATmega64, ATmega8, ATmega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny2313, ATtiny28, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646 |



| | | | |
|---|---|---|---|
| PCINT0_vect | SIG_PIN_CHANGE0 | Pin Change Interrupt Request 0 | ATmega162, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega406, ATmega48P, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny13, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646 |
| PCINT1_vect | SIG_PIN_CHANGE1 | Pin Change Interrupt Request 1 | ATmega162, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega406, ATmega48P, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, AT90USB162, AT90USB82 |
| PCINT2_vect | SIG_PIN_CHANGE2 | Pin Change Interrupt Request 2 | ATmega3250, ATmega3250P, ATmega328P, ATmega3290, ATmega3290P, ATmega48P, ATmega6450, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny48 |

# Interrupt Vector Table

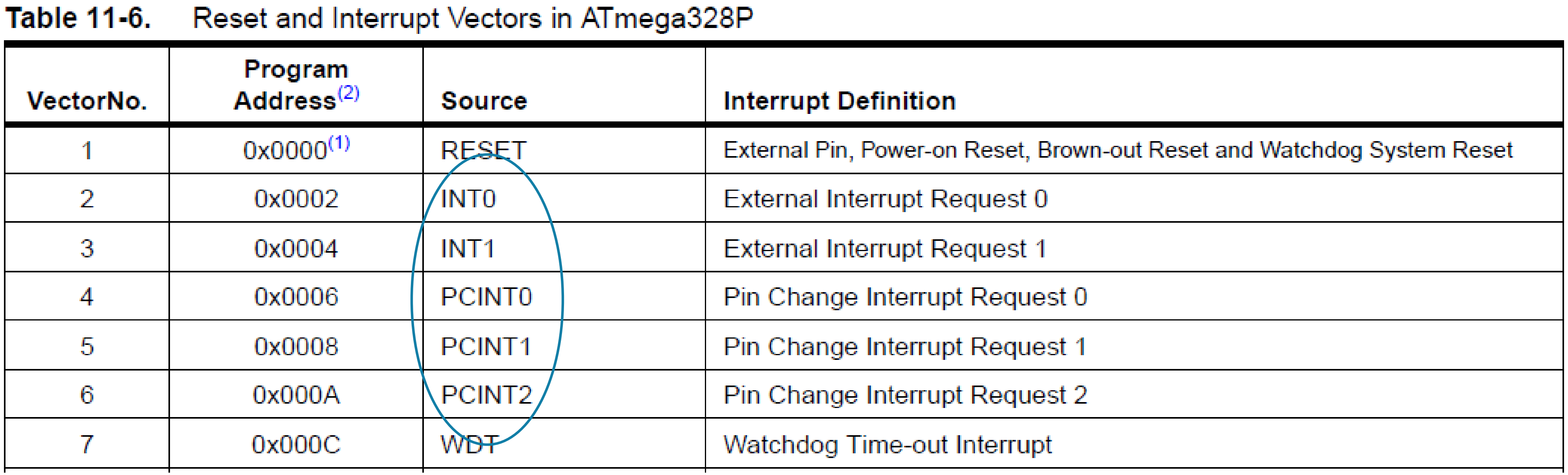


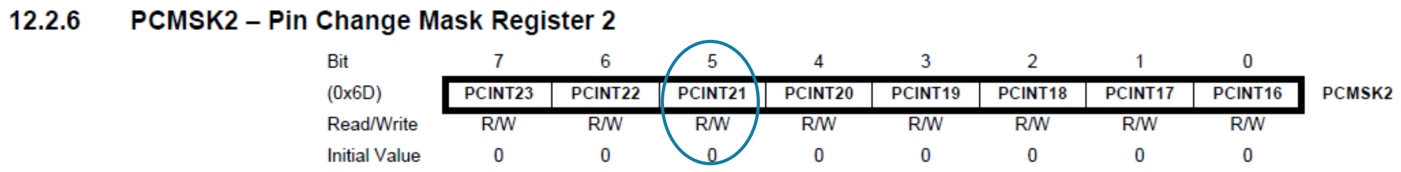
Table 11-6. Reset and Interrupt Vectors in ATmega328P

| VectorNo. | Program Address[2] | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x0000[1] | RESET | External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | External Interrupt Request 1 |
| 4 | 0x0006 | PCINT0 | Pin Change Interrupt Request 0 |
| 5 | 0x0008 | PCINT1 | Pin Change Interrupt Request 1 |
| 6 | 0x000A | PCINT2 | Pin Change Interrupt Request 2 |
| 7 | 0x000C | WDT | Watchdog Time-out Interrupt |

- Notice that the external interrupts and pin interrupt are at the top of the table
- They will be the first to be checked after an ISR finishes → They have priority
- Usage: Program a SW interrupt for executing an *atomic* piece of code
  - A pin is set as an output
  - Main code toggles the pin
  - This creates a PCINT HW event and sets a corresponding flag
  - Interrupt unit will scan this flag first and prioritizes the corresponding PCINT ISR (i.e., if during toggling another ISR is called due to some other HW event, then once this ISR is finished the PCINT ISR will be called next)
  - The PCINT ISR will be fully executed without interruption → an atomic execution

# Example PCINT21 = PD5

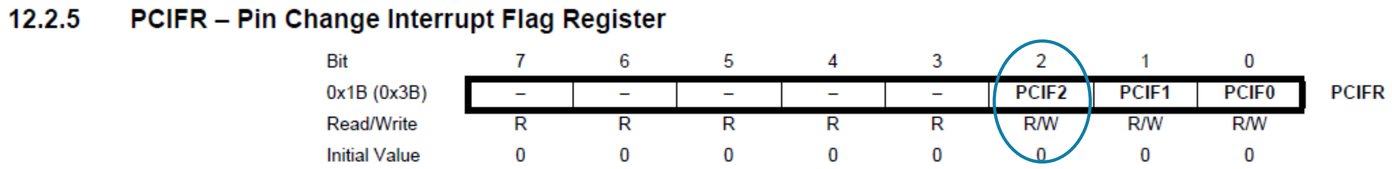
DDRD |= (1<<DDD5); //**PD5=PCINT21 is output**



**12.2.6 PCMSK2 – Pin Change Mask Register 2**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x6D) | PCINT23 | PCINT22 | PCINT21 | PCINT20 | PCINT19 | PCINT18 | PCINT17 | PCINT16 | PCMSK2 |
| 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 | |

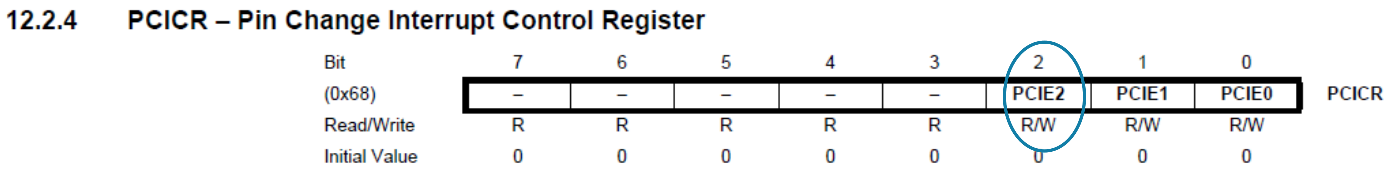
PCMSK2 = (1<<PCINT21); //**toggling PD5 sets flag**



**12.2.5 PCIFR – Pin Change Interrupt Flag Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x1B (0x3B) | – | – | – | – | – | PCIF2 | PCIF1 | PCIF0 | PCIFR |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

When PD5 toggles, flag PCIFR & (1<<PCIF2) changes from 0 (0 as an integer represents 0x00) to (1<<PCIF2) (which, represented as an integer, equals 4)



**12.2.4 PCICR – Pin Change Interrupt Control Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x68) | – | – | – | – | – | PCIE2 | PCIE1 | PCIE0 | PCICR |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

PCICR |= (1<<PCIE2); //**Enable interrupt for**
//**PCIFR.PCIF2**

Write

- ISR(PCINT2_vect){ Atomic code;}
- If the atomic code needs to be executed in the main program, just toggle PORTD ^= (1<<PORTD5);

# Sequence of Events

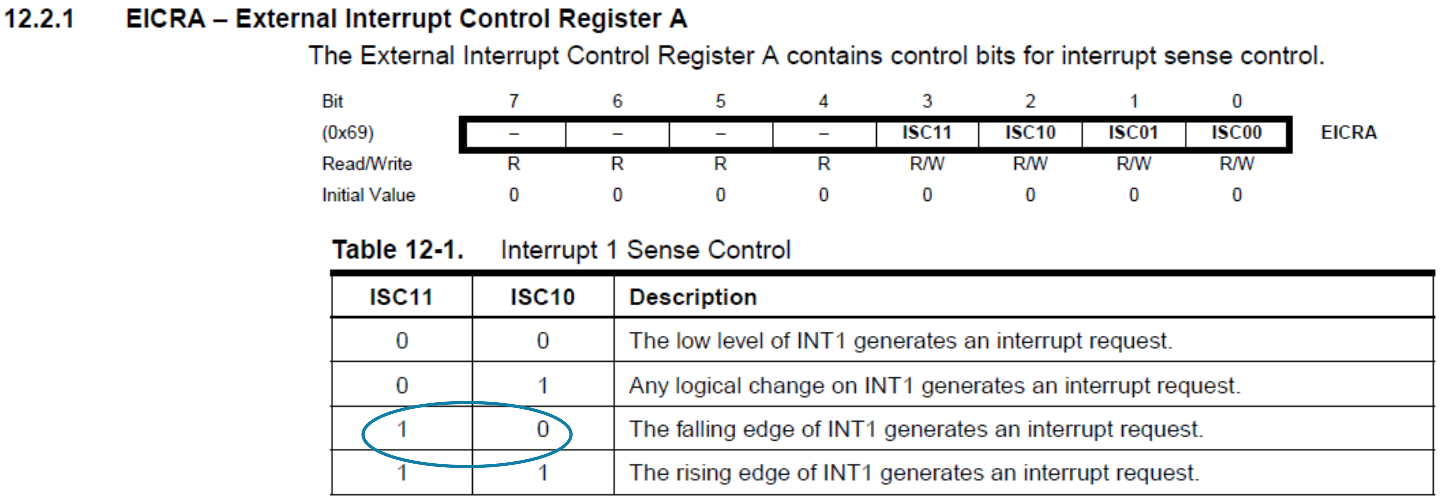
1. In main program toggle PORTD ^= (1<<PORTD5);
2. PCIFR.PCIF2 is set to 1
3. PCICR |= (1<<PCIE2); → Interrupt unit checks PCIFR.PCIF2
4. If currently an ISR is executing, finish its execution and start the next instruction in the main program
5. As soon as the current instruction in the main program is finished, the interrupt unit checks for flags with enabled interrupts
6. The interrupt unit does this in round robin fashion but starts at the top of the interrupt vector table after an ISR is finished → prioritizes RESET over external interrupts over pin interrupts over the rest
7. Looks up address corresponding to ISR(PCINT2_vect), saves register state, puts PC on stack, etc.
8. Execute without any interruption ISR(PCINT2_vect){ Atomic code;}
9. During RETI state is restored, flag PCIFR.PCIF2 is cleared, and PC points to the next instruction in the main program

NOTE: Instead of PORTD ^= (1<<PORTD5); the main code can also directly set PCIFR |= (1<<PCIF2);

# INT1

- Programming external interrupt INT1 = PD3 on falling edge
  - Switch connected to PD3 (set to PD3 to input): DDRD &= ~(1<<DDD3);
  - #define SW_PRESSED !(PIND & (1<<PIND3))
  - If SW_PRESSED {…} checks whether PIND & (1<<PIND3) == 0
  - PD3 low means pressed and PD3 high means not pressed: Want to detect falling edge

- EICRA |= (1<<ISC11);



**12.2.1 EICRA – External Interrupt Control Register A**

The External Interrupt Control Register A contains control bits for interrupt sense control.

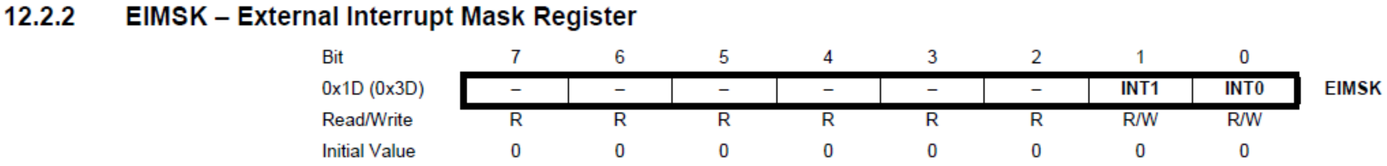
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| (0x69) | – | – | – | – | ISC11 | ISC10 | ISC01 | ISC00 | EICRA |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Table 12-1.** Interrupt 1 Sense Control

| ISC11 | ISC10 | Description |
|---|---|---|
| 0 | 0 | The low level of INT1 generates an interrupt request. |
| 0 | 1 | Any logical change on INT1 generates an interrupt request. |
| 1 | 0 | The falling edge of INT1 generates an interrupt request. |
| 1 | 1 | The rising edge of INT1 generates an interrupt request. |

- EIMSK |= (1<<INT1);

Need to write ISR and implement a debounce state machine …



**12.2.2 EIMSK – External Interrupt Mask Register**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x1D (0x3D) | – | – | – | – | – | – | INT1 | INT0 | EIMSK |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# INT1

```
#define SW_PRESSED !(PIND & (1<<PIND3))

void Initialize(void)
{
   DDRD &= ~(1<<DDD3);
   EICRA |= (1<<ISC11);
   EIMSK |= (1<<INT1);
   …. Timer 0 ….
   poll_time = POLLING_DELAY;
   DebounceFlag = 0;
}

void ISR(TIMER0_COMPA_vect)
{
   if ((poll_time>0) && (DebounceFlag==1)) --poll_time;
   …
}

ISR(INT1_vect)
{
   EIMSK &= ~(1<<INT1); // Disable interrupt
   … record this event …
   DebounceFlag = 1;
}
```

```
void PollButton(void)
{
  if SW_PRESSED { … latest recorded event is for a button push …}
  DebounceFlag = 0;
  poll_time = POLLING_DELAY;
  EIMSK |= (1<<INT1);
}

int main(void)
{
   Initialize();
   sei();

   while(1)
   {
      if (poll_time == 0) {PollButton();}
      …
    }
}
```

# Debouncing with a Pin Interrupt

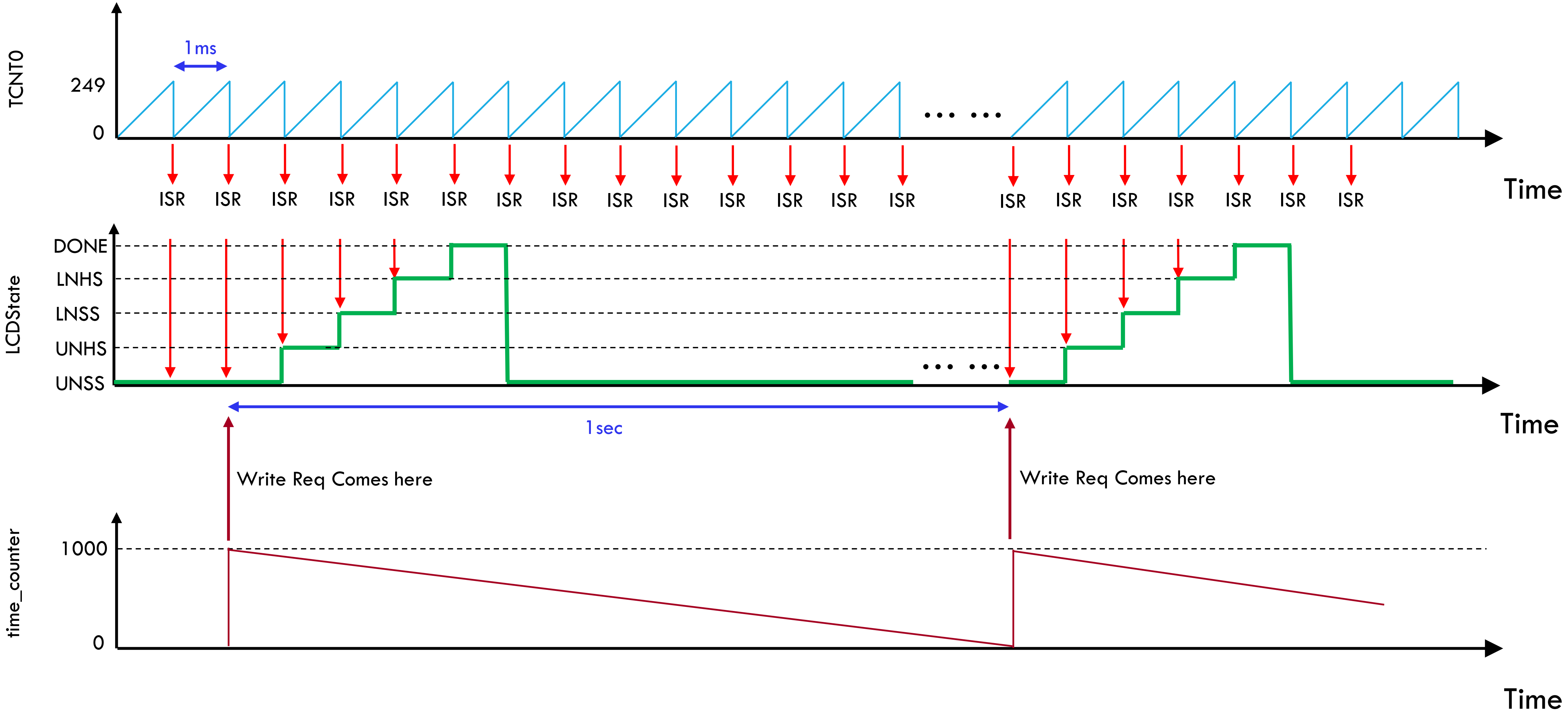
- Instead of using INT1 we can use a pin interrupt
- The pin toggles:
  - Wrap all ISR code in an extra if statement
  - If SW_PRESSED { .. Code .. }
  - Now we will only execute Code if the button transitions from not-pressed to pressed.

# Stop Watch

- The ISR records the moment of the falling edge
- Represented by a SW counter maintained in ISR(TIMER0_COMPA_vect)
- Only if the button is really pressed, PollButton() will set a flag telling the main program that the recorded event is valid.
- The main while loop polls the flag and as soon as it is set it e.g. prints the recorded time after which the flag is set back to invalid.

- All kinds of variations possible

# Overall System Timing (Lab 3b & 3c)


TCNT0
1ms
249
0
ISR
Time
LCDState
DONE
LNHS
LNSS
UNHS
UNSS
Time
1sec
Write Req Comes here
Write Req Comes here
time_counter
1000
0
Time