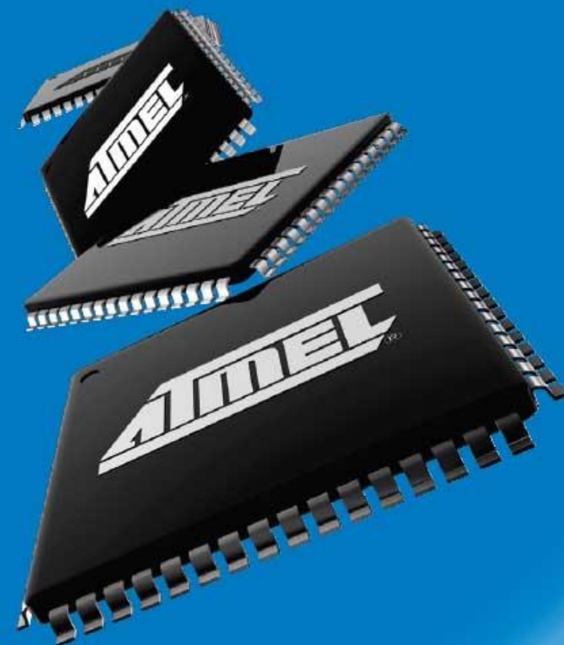
AVR®

8-bit Microcontrollers

AVR®32

32-bit Microcontrollers and Application Processors



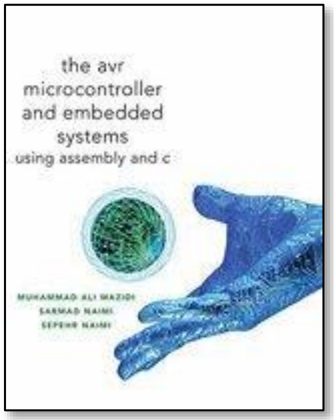
The Real World of External Interrupts

February 2009

Everywhere You Are®

Atmel AVR External Interrupts

Reading



The AVR Microcontroller and Embedded Systems using Assembly and C)
by Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi

Chapter 10 AVR Interrupt Programming in Assembly and C

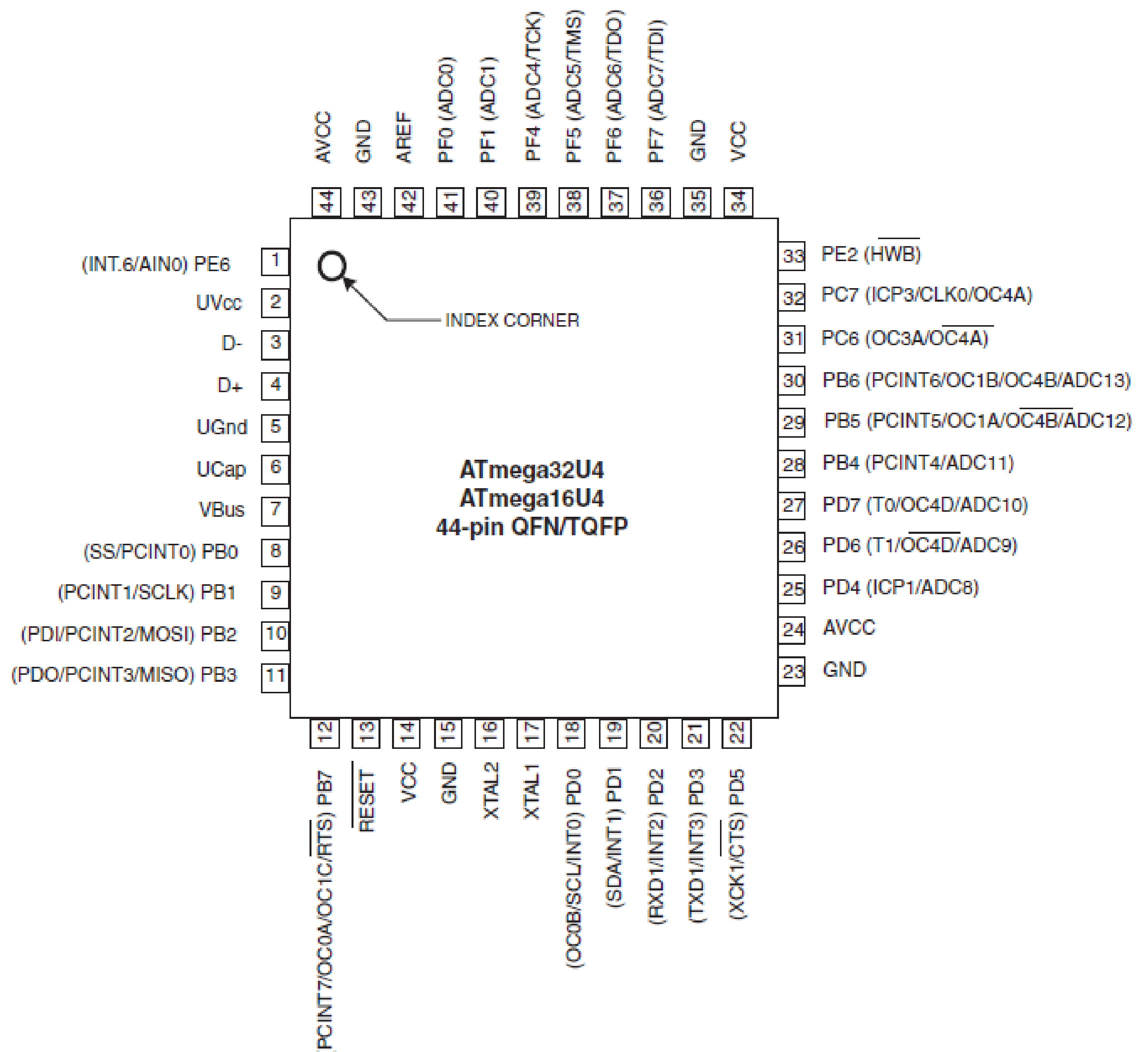
10.3 Programming External Interrupts

10.5 Interrupt Programming in C

Table of Contents

External Interrupts

- Review ATmega32U4 Interrupts Lecture Notes page 7 "**ATmega32U4 Interrupt Vector Table**"
- External Interrupts are triggered by the INT0, INT1, INT2, INT3, and INT6 pins
- Pin Change Interrupt mapped to 8 General Purpose I/O Port Pins: `PCINT7 (PB7)` ⇨ `PCINT0 (PB0)`



ATmega32U4 Interrupt Vector Table

Vector No	Program Address	Source	Interrupt Definition	Arduino/C++ ISR() Macro Vector Name
1	0x0000	RESET	Reset	
2	0x0002	INT0	External Interrupt Request 0 (pin D0)	(INT0_vect)
3	0x0004	INT1	External Interrupt Request 1 (pin D1)	(INT1_vect)
4	0x0006	INT2	External Interrupt Request 2 (pin D2)	(INT2_vect)
5	0x0008	INT3	External Interrupt Request 3 (pin D3)	(INT3_vect)
6	0x000A	Reserved	Reserved	
7	0x000C	Reserved	Reserved	
8	0x000E	INT6	External Interrupt Request 6 (pin E6)	(INT6_vect)
9	0x0010	Reserved		
10	0x0012	PCINT0	Pin Change Interrupt Request 0 (pins PB7 to PB0)	(PCINT0_vect)
11	0x0014	USB General	USB General Interrupt request	(USB_GENERAL_vect)
12	0x0016	USB Endpoint	USB Endpoint Interrupt request	(USB_ENDPOINT_vect)
13	0x0018	WDT	Watchdog Time-out Interrupt	(WDT_vect)
14	0x001A	Reserved	Reserved	
↓				
37	0x0048	TWI	2-wire Serial Interface (I2C)	(TWI_vect)
38	0x004A	SPM READY	Store Program Memory Ready	(SPM_READY_vect)
39	0x004C	TIMER4 COMPA	Timer/Counter4 Compare Match A	(TIMER4_COMPA_vect)
40	0x004E	TIMER4 COMPB	Timer/Counter4 Compare Match B	(TIMER4_COMPB_vect)
41	0x0050	TIMER4 COMPD	Timer/Counter4 Compare Match D	(TIMER4_COMPD_vect)
42	0x0052	TIMER4 OVF	Timer/Counter4 Overflow	(TIMER4_OVF_vect)
43	0x0054	TIMER4 FPF	Timer/Counter4 Fault Protection Interrupt	(TIMER4_FPF_vect)

ATMEGA32U4 EXTERNAL INTERRUPT SENSE CONTROL

- The INT0 and INT1 interrupts can be triggered by a low logic level, logic change, and a falling or rising edge.

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

Bit	7	6	5	4	3	2	1	0	
	-	-	ISC61	ISC60	-	-	-	-	EICRB
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	

- This is set up as indicated in the specification for the External Interrupt Control Register as defined in Section 11.0.1 EICRA and 11.0.2 EICRB of the Datasheet. The number “n” can be 0, 1, 2, 3, or 6.

ISCn1	ISCn0	Arduino mode	Description
0	0	LOW	The low level of INTn generates an interrupt request
0	1	CHANGE	Any logical change on INTn generates and interrupt request
1	0	FALLING	The falling edge of INTn generates an interrupt request
1	1	RISING	The rising edge of INTn generates an interrupt request

ATmega32U4 External Interrupt Enable

- All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register (SREG) in order to enable the interrupt.

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
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	

- The ATmega 32U4 supports five external interrupts which are individually enabled by setting bits INT6, INT3, INT2, INT1 and INT0 in the External Interrupt Mask Register (Section 11.0.3 EIMSK).

Bit	7	6	5	4	3	2	1	0	
	-	**INT6**	**-**	**-**	**INT3**	**INT2**	**INT1**	**IINT0**	**EIMSK**
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	

- Let's look at an example. When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed.

Bit	7	6	5	4	3	2	1	0	
	-	**INTF6**	**-**	**-**	**INTF3**	**INTF2**	**INTF1**	**IINTF0**	**EIFR**
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	

- If the *interrupt is disabled*, the flag bit can be cleared by writing a logical one to it. The EIFR register at I/O address 0x1C is within the I/O address range (0x00 to 0x1F) of the Set Bit in I/O Register (SBI) Instruction.

When Will External Interrupts Be Triggered?

- When the external interrupts are enabled and are configured as low level triggered (Type 2), the interrupts will trigger as long as…

  1. The pin is held low.
  2. The low level is held until the completion of the currently executing instruction.
  3. The level is held long enough for the MCU to completely wake-up (assuming it was asleep).
     - Low level interrupts are detected asynchronously (no clock required). The I/O clock is halted in all sleep modes except idle mode. Therefore low level interrupts can be used for waking the part from all sleep modes.
  4. Among other applications, low level interrupts may be used to implement a handshake protocol.

- When external interrupts are enabled and are configured as edge or logic change (toggle) triggered, (Type 1[1]) the interrupts will trigger as long as…

  1. The I/O clock is present.
     - This implies that these interrupts cannot be used for waking up the part from sleep modes other than idle mode.
  2. The pulse lasts longer than one I/O clock period. Shorter pulses are not guaranteed to generate an interrupt.

Pin Change Interrupts

- In addition to our five (5) external interrupts, the eight (8) pins assigned to GPIO Port B can be programmed to trigger an interrupt if the pin changes state.
- These 8 pins are assigned to interrupt group PCI 0 and correspond to GPIO Port B.
- This group is assigned to one pin change interrupt flag PCIF0 within the PCIFR register.

Bit	7	6	5	4	3	2	1	0	
			–	–	–	–	–	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

How To Enable a Pin Change Interrupt

- The pin change interrupt flag PCIF0 will be set, if the following conditions are met.

1. The SREG global interrupt enable bit I is set.

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
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	

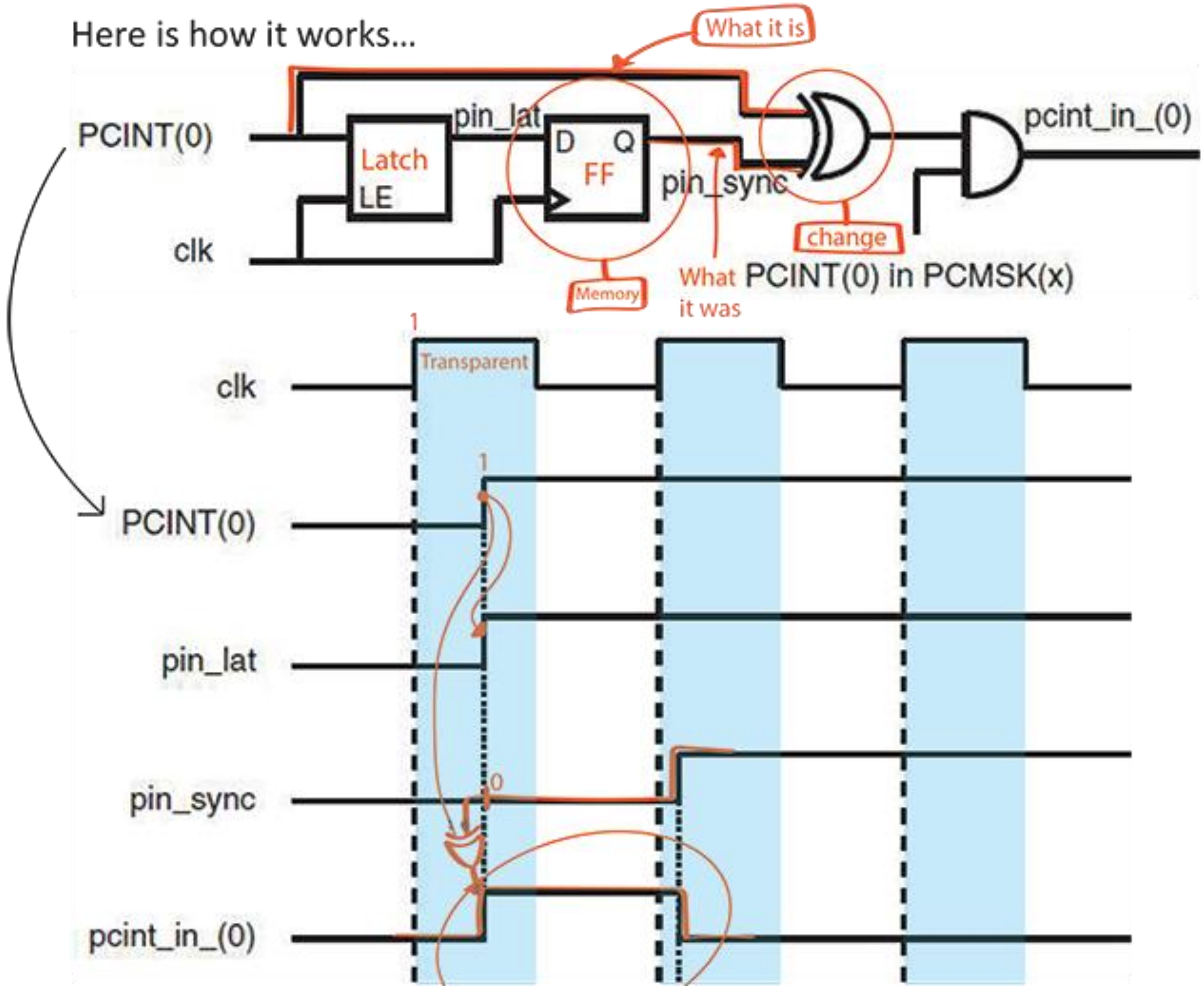
2. The pin change interrupt enable bit 0 (PCIE0) within the PCICR register at extended I/O SRAM address 0x68 is set.

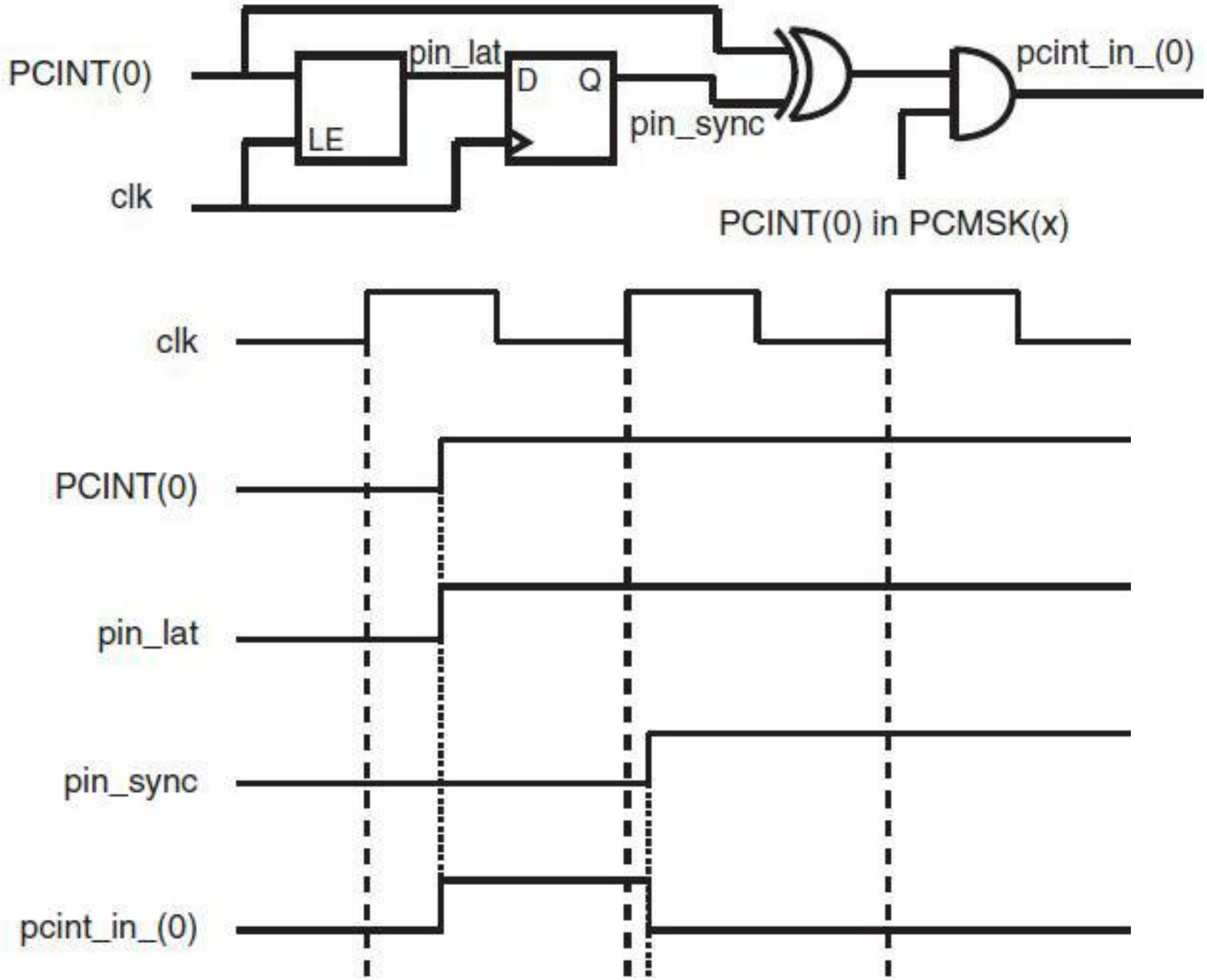
Bit	7	6	5	4	3	2	1	0	
			–	–	–	–	–	PCIE0	PCICR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3. The individual pin change interrupt enable mask bit assigned to the pin (PCINT 7:0) within the PCMSK0 register at extended I/O SRAM address 0x6B is set.

Bit	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

How a Pin Change Interrupt Works





ATMEGA32U4 INTERRUPT PROCESSING (*REVIEW*)

- ①When an interrupt occurs, ②the microcontroller completes the current instruction and ③stores the address of the next instruction on the stack
- It also turns off the interrupt system to prevent further interrupts while one is in progress. This is done by ④clearing the SREG Global Interrupt Enable I-bit.

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
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	

- The ⑤Interrupt flag bit is cleared for Type 1 Interrupts only (see the next page for Type definitions).
- The execution of the ISR is performed by ⑥loading the beginning address of the ISR specific for that interrupt into the program counter. The AVR processor starts running the ISR.
- ⑦Execution of the ISR continues until the return from interrupt instruction (`reti`) is encountered. The ⑧SREG I-bit is automatically set when the `reti` instruction is executed (i.e., Interrupts enabled).
- When the AVR exits from an interrupt, it will always ⑨return to the interrupted program and ⑩execute one more instruction before any pending interrupt is served.
- The Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

```
push  reg_F
in    reg_F,SREG
  :
out   SREG,reg_F
pop   reg_F
```

Programming the Arduino to Handle External Interrupts[1]

- Stop compiler optimization of variables within an ISR by adding the `volatile` qualifier. This keeps the current value in SRAM until needed.

```
const byte pin = 8;                       // green LED 0
volatile int state = LOW;
```

- Add jumps in the IVT to ISR routine, configure External Interrupt Control Register A (EICRA), and enable local and global Interrupt Flag Bits.

External Interrupt 0 or 1

Interrupt Sense Control (ISC)

```
 void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);  // protoshield button
}
```

ISR

- Write Interrupt Service Routine (ISR)

```
void blink()
{
  state = !state;
}
```

To disable interrupts globally (clear the I bit in SREG) call the `noInterrupts()` function. To once again enable interrupts (set the I bit in SREG) call the `interrupts()` function.

[1] Source: [Arduino attachInterrupt]

PROGRAMMING THE ARDUINO TO HANDLE INTERRUPTS[2]

- In the AVR-GCC environment upon which the Arduino language is built, the interrupt vector table (IVT) is predefined to point to interrupt routines with predetermined names (see “ATmega32U4 Interrupt Vector Table” on page 5).

- You create an ISR by using the Macro `ISR()` and these names.

```
#include <avr/interrupt.h>

ISR(ADC_vect)
{
    // user code here
}
```

- Now that you have defined the ISR you need to locally and globally enable it. Here are the relevant links for learning how to complete your ISR definition.
    - Global manipulation of the interrupt flag
    - Gammon Software Solutions forum - Interrupts
    - ISR() macro

[2] Source: Arduino attachInterrupt

Practice Problems

1. Initialize Interrupt 1 (INT1) pin to generate an Interrupt on a rising edge. Set all unused bits to default values.

```
________      R16, 0x_____
________      EICRA, R16
```

2. Initialize Interrupt 0 (INT0) pin to generate an Interrupt on a falling edge. Do not change the other bits (ISC11, ISC10) in the External Interrupt Control Register A (EICRA). You should assume the previous code (problem 1) has been written (i.e., bits 1 and 0 cleared)

```
________      R16, EICRA
________      R16, 0x_____
________      EICRA, R16
```

3. Configure Pin 15 PB1 (OC1A/PCINT1) to generate an Interrupt whenever the pin changes state. Do not change any other bits. To make things more interesting, do not use the SBR instruction.

```
________      R16, PCMSK0             ; load
________      R17, PCICR
________      R18, SREG
________      R16, 0x______           ; do something
________      R17, 0x______
________      R18, 0x______
________      PCMSK0, R16             ; store
________      PCICR, R17
________      SREG, R18
```