Arduino Pin Change Interrupts

I recently needed to do some work with Pin Change Interrupts and it was a bit of a learning experience for me. As it turns out, they're actually pretty easy. I'm posting this so that when I need to look it up in the future, I can easily find it and maybe I can help somebody else out too.

I'm going to talk specifically about the ATMEGA328 chip here since it is by far the most common in Arduinos and in my lab, but the information here should transfer easily other ATMEGAs as well. Before we begin I want to make sure we're all using the same terms. There are two main categories of interrupts: Hardware and Software. A Hardware interrupt is triggered by something outside of the chip like a button while a Software interrupt is triggered from inside the chip like a timer.

Within the Hardware interrupt there are two categories: External interrupts and Pin Change Interrupts. The nomenclature here is confusing since all hardware interrupts are external to the chip. But the things we are now calling External Interrupts are limited to only a couple pins, while the Pin Change interrupts can occur on all input pins. For instance, on the ATMEGA328, there are two External Interrupts but 24 Pin Change Interrupts.

Each time an interrupt occurs, it triggers the associated ISR (Interrupt Service Routine) assuming you have turned that interrupt on. Each External Interrupt has its own ISR and they can be triggered independently by either a rising signal, falling signal, or by both. But the Pin Change Interrupts share an ISR between all the pins on a port (port B, C, and D). And anytime a pin changes on that port, it calls the port's ISR which must then decide which pin caused the interrupt. So Pin Change Interrupts are harder to use but you get the benefit of being about to use any pin.

12.4 Interrupt Vectors in ATmega328 and ATmega328P

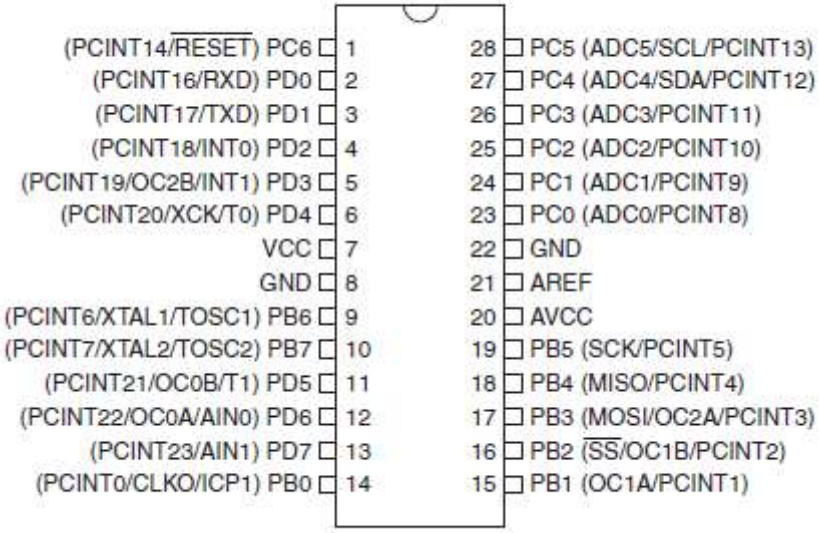
Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

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 D8
5	0x0008	PCINT1	Pin Change Interrupt Request 1 D9
6	0x000A	PCINT2	Pin Change Interrupt Request 2 D10
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Notes: 1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Boot Loader Support – Read-While-Write Self-Programming" on page 269.

(https://mwwalk.files.wordpress.com/2014/08/atmega328-interrupt-list.png)

Hardware interrupts are also easier to use in the Arduino environment. You just call the function attachInterrupt and input the interrupt number and the function to call when it triggers. But up until recently, there wasn't a good Pin Change Interrupt library and even now it isn't included so you have to download it separately. Plus, I'm a fan of not using libraries for simple tasks like this that could be accomplished in a few lines of code. I just feel like it's easier to debug and control what's going on. For these reasons, all of the below will use AVR C commands, which can be used regardless of whether you're using the Arduino boot-loader and IDE or another.



(https://mwwalk.files.wordpress.com/2014/08/arduino-atmega328-pinout.png)

There are three steps to using Pin Change Interrupts.
1) Turn on Pin Change Interrupts
2) Chose which pins to interrupt on
3) Write an ISR for those pins

1 – Turn on Pin Change Interrupts

The Pin Change Interrupts are turned on by setting certain bits in the PCICR register as seen below. Bit 0 turns on port B (PCINT0 – PCINT7), bit 1 turns on port C (PCINT8 – PCINT14), and bit 2 turns on port D (PCINT16 – PCINT23). This code shows how to turn them on or off. Note, I'm using |= instead of = because it is more versatile but either would work. You can also use decimal or hexadecimal instead of binary, but I think the binary is the easiest to understand.

```
PCICR |= 0b00000001;    // turn on port b
PCICR |= 0b00000010;    // turn on port c
PCICR |= 0b00000100;    // turn on port d
PCICR |= 0b00000111;    // turn on all ports
```

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	

(https://mwwalk.files.wordpress.com/2014/08/atmega328-pcicr.png)

2 – Choose Which Pins to Interrupt

I know I said earlier that a change on any of the pins in a port would trigger the port's ISR, but that is true only when you turn that particular pin on. This is down with what is called a mask. Since the ATMEGA328 has 3 ports, it also has three masks: PCMSK0, PCMSK1, and PCMSK2. These are set the same way the register for the PCICR was set. Again, you can use |= or just = but |= gives you the ability to separate them onto different lines if you'd like.

```
PCMSK0 |= 0b00000001;    // turn on pin PB0, which is PCINT0, physical pin 14
PCMSK1 |= 0b00010000;    // turn on pin PC4, which is PCINT12, physical pin 27
PCMSK2 |= 0b10000001;    // turn on pins PD0 & PD7, PCINT16 & PCINT23
```

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	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	

(https://mwwalk.files.wordpress.com/2014/08/atmega329-pcmsk0.png)

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

(https://mwwalk.files.wordpress.com/2014/08/atmega329-pcmsk1.png)

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	

(https://mwwalk.files.wordpress.com/2014/08/atmega329-pcmsk2.png)

3 – Write the ISR

The last step is to write the ISR that will be called for each of these interrupts. The general guidelines on ISRs are to make them as short as possible and not use delays in them. Also, you should spell and capitalize them correctly (see **here (index.php?p=1128)**). Finally, if using variables in these ISRs you want to make the variable volatile; this tells the compiler that it could change at any time and to reload it each time instead of optimizing it. To define these ISRs just type the function below your loop.

```
ISR(PCINT0_vect){}    // Port B, PCINT0 - PCINT7
ISR(PCINT1_vect){}    // Port C, PCINT8 - PCINT14
```

```
ISR(PCINT2_vect){}    // Port D, PCINT16 - PCINT23
```

Putting It All Together

Below is code putting all of these together. You'll notice I'm using cli() before enabling the interrupts and sei() afterward. cli() turns interrupts off while we're messing with them and then sei() turns them back on. Also, you should include avr/interrupt.h at the top of the sketch.

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

volatile int value = 0;

void setup()
{
cli();
PCICR |= 0b00000011; // Enables Ports B and C Pin Change Interrupts
PCMSK0 |= 0b00000001; // PCINT0
PCMSK1 |= 0b00001000; // PCINT11
sei();

Serial.begin(9600);
}

void loop()
{
Serial.println(value);
}
```

```
ISR(PCINT0_vect)
{
value++;
}

ISR(PCINT1_vect)
{
value–;
}
```

a

(http://www.amazon.com/s/ref=as_acph_ceb_drv_56_on?rh=i%3Acomputers%2Cn%3A1292116011&hidden-keywords=state&field-pct-off-mp-owner=30-&ie=UTF&tag=wanderingengineer-20&camp=0&creative=0&linkCode=ur1&adid=0MGTSR7H0M00HRTJJ2E6&)
(https://www.amazon.com/gp/goldbox?&tag=thewandengi-20&camp=0&creative=0&linkCode=ur1&adid=153Y41XPC3V4P5J1FHW3&)
Posted on **August 11, 2014May 16, 2015** by **mwwalk**Posted in **Electrical**, **Engineering**, **Software**, **Tutorials**Tagged **Arduino**, **AVR**, **Interrupt**, **Pin Change Interrupt**.

34 thoughts on “Arduino Pin Change Interrupts”

1. **VANCE SAYS:**
   September 2, 2014 at 4:04 am
   What’s up to every single one, it’s actually a fastidious for me to pay a visit this website, it consists of priceless Information.

   ↳ **Reply**
2. **LOUIS SAYS:**
   June 26, 2015 at 9:39 am
   It’s a pity, you write a magnificent article about pin change interrupts then end it with a terrible example.

   ↳ **Reply**
   MWWALK SAYS:
   June 26, 2015 at 12:11 pm