Interrupts

Interrupts

- An Interrupt is an *asynchronous* event that is outside of the normal flow of control.
- Without interrupts, we have to Poll
  - Polling takes time best spent elsewhere. With polling we could miss important events while processing elsewhere in the code.
- Interrupt Types
  - Hardware Interrupts
    - Occur in response to external events, such as a pin changing state.
    - Can occur when some internal hardware element changes state. (When the ADC is ready for another conversion, for example.)

Interrupts

- Software Interrupts
  - In response to a software signal/instruction in an event-driven program
- Timer Interrupts
  - Special case of hardware interrupt

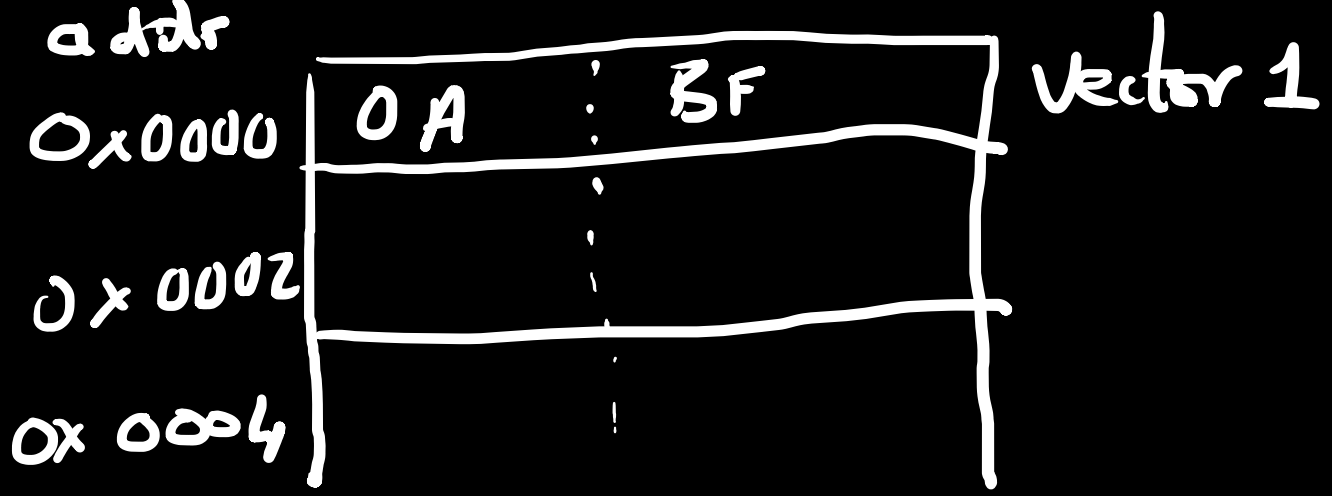
External Hardware Interrupts

- From the datasheet for our microcontroller, we see a list of sources and event types
- When interrupts are enabled, and one of these events occur, the function associated with the interrupt is called.

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
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
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

Memory


addr
0x0000
0x0002
0x0004
0A
BF
Vector 1


0x0ABF
Code associated with Vector 1

External Hardware Interrupts

- We need to do 3 things to set up an interrupt
  1. Set the **global interrupt bit** in the status register
  2. Set the **enable bit for our specific interrupt type** (each interrupt has its own on/off bit)
  3. Write an **Interrupt Service Routine** (ISR) and **attach it** to the interrupt vector.

External Hardware Interrupts

- In our code, we need to include the IO and Interrupt libraries: `avr/io.h`, `avr/interrupt.h`

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

Void setup(){
EIMSK |= (1<<INT0); //set the interrupt bit for INT0 in the
                        //External Interrupt Mask Register
sei() ; //enable global interrupts
}
```

7.3.1 SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

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	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

13.2.2 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	

- **Bit 7:2 – Reserved**

These bits are unused bits in the ATmega48A/PA/88A/PA/168A/PA/328/P, and will always read as zero.

- **Bit 1 – INT1: External Interrupt Request 1 Enable**

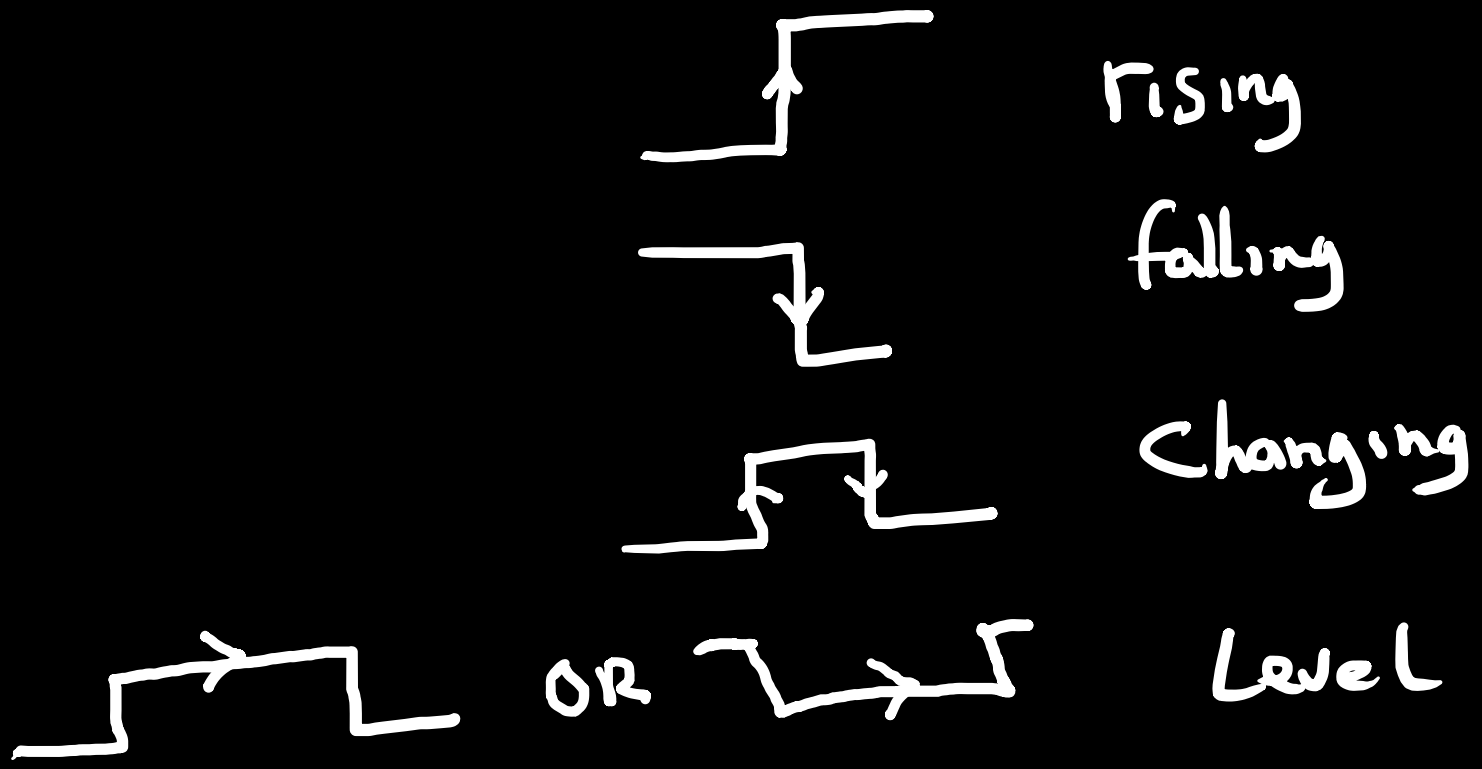
When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

- **Bit 0 – INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 Interrupt Vector.

External Hardware Interrupt

- Setting the sensing mode: rising, falling, changing, level



13.2.1 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	

- **Bit 7:4 – Reserved**

These bits are unused bits in the ATmega48A/PA/88A/PA/168A/PA/328/P, and will always read as zero.

- **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

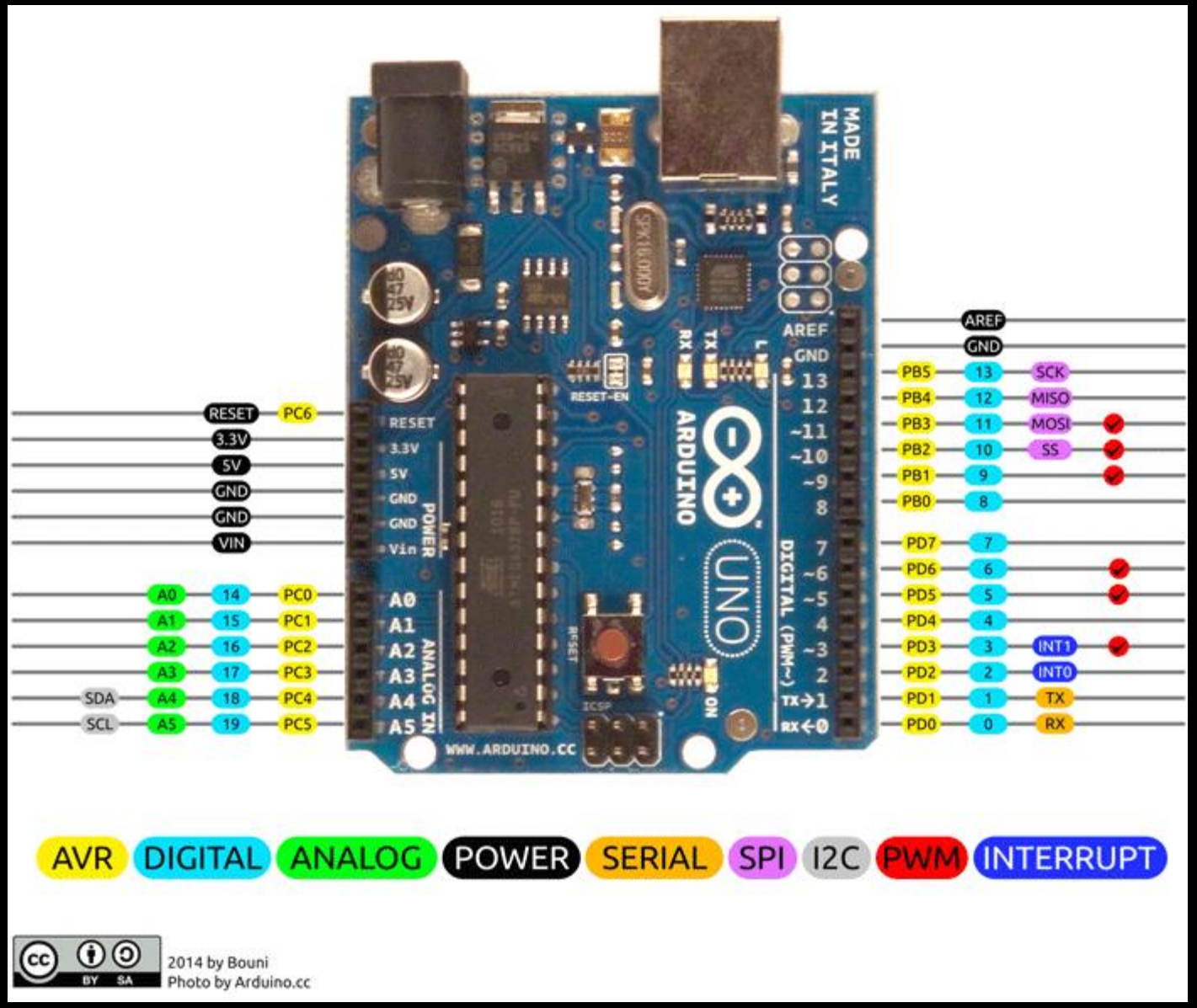
The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 13-1. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 13-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

To Sense the falling Edge on INT0:

```
EICRA |= (1 << ISC01);
```



Writing the ISR

In general we have:

```
ISR ({VECTOR}_vect){
...
}
```

```
ISR (INT0_vect){
  PORTB ^= (1 << PB5);
}
```

Writing the ISR

- `ISR` is a predefined macro in interrupt.h

```
#ifdef __cplusplus
#  define ISR(vector, ...)            \
    extern "C" void vector (void) __attribute__ ((signal,__INTR_ATTRS)) __VA_ARGS__; \
    void vector (void)
#else
#  define ISR(vector, ...)            \
    void vector (void) __attribute__ ((signal,__INTR_ATTRS)) __VA_ARGS__; \
    void vector (void)
#endif
```

Arduino Interrupt Functionality

- Arduino (as does mbed) provides a simplified way of using interrupts:

```
void setup(){
    attachInterrupt(2, foo, FALLING);
}
Void foo(){…}
```

External Interrupt 0 (pin 2 on arduino)

Callback function

Sensing mode.

Arduino Interrupt Functionality (New Version)

- More portable approach.

```
void setup(){
    attachInterrupt(digitalPintToInterrupt(2),
                    foo, FALLING);
}
Void foo(){…}
```

External Interrupt 0 (pin 2 on arduino)

Callback function

Sensing mode.

Volatile Variables

- NB: Any variable used in an ISR (`foo()` in this case) MUST be declared as `volatile`.
- E.G.: `volatile int my_int;`
- This is because these variables can change in ways that are unknown to the compiler.
- The volatile keyword tells the compiler that the variable can be modified outside of the normal flow of control.
- This usually applies to variables that are mapped to a fixed memory address such as a device register.
- Statements containing these variables should not be reordered, for optimization reasons, for example, by the compiler.

Volatile Variables

- For Example
  - Suppose `KEYBOARD` is a device register that accepts characters from the keyboard

```
void get_two_kbd_chars(){
 extern char KEYBOARD;
 char c0, c1;
 c0 = KEYBOARD;
 c1 = KEYBOARD;
}
```

this is the desired behaviour

```
void get_two_kbd_chars(){
 extern char KEYBOARD;
 char c0, c1;
 register char tmp;
 tmp = KEYBOARD;
 c0 = tmp;
 c1 = tmp
}
```

The Compiler could produce this code

`Extern volatile char KEYBOARD;` will prevent the compiler making this optimization