Microprocessors, Lecture 11

AVR Interrupts

Hamid Fadishei
Assistant Professor, University of Bojnord
Spring 2015

Some WinAVR GCC hints

If you need a variable with exactly known precision, then #include <stdint.h>

Example usage:

```
uint8_t my_var = 0x55;
PORTB = my_var;
```

stdint.h

```
70
71  /** \ingroup avr_stdint
72      8-bit signed type. */
73
74  typedef signed char int8_t;
75
76  /** \ingroup avr_stdint
77      8-bit unsigned type. */
78
79  typedef unsigned char uint8_t;
80
81  /** \ingroup avr_stdint
82      16-bit signed type. */
83
84  typedef signed int int16_t;
85
86  /** \ingroup avr_stdint
87      16-bit unsigned type. */
88
89  typedef unsigned int uint16_t;
```

If you need to access registers inside an AVR, or their individual bits then #include <avr/io.h>

This will effectively include another .h file depending on the platform you are compiling for (for example iom16.h if you are compiling for ATmega16

Example usage:

```
// Set PB3 to 1
// (Note: bit names are not
// always as straightforward
// as PB3)
PORTB |= (1<<PB3);

// Set PB3 to 1
// Another method
PORTB |= _BV(PB3);

// Clear PB3
PORTB &= ~(1<<PB3);
```

iom16.h

```
229  #define DDB4     4
230  #define DDB5     5
231  #define DDB6     6
232  #define DDB7     7
233
234  #define PORTB    _SFR_IO8(0x18)
235  #define PB0      0
236  #define PB1      1
237  #define PB2      2
238  #define PB3      3
239  #define PB4      4
240  #define PB5      5
241  #define PB6      6
242  #define PB7      7
```

AVR interrupts

- Interrupt sources
  - External interrupts
    - Triggered by external I/O devices
    - Different number of external interrupts are provided by different AVR family members
      - ATmega16: INT0, INT1, INT2
      - ATmega64: INT0…INT7
  - Internal interrupts
    - Triggered by on-chip I/O devices
      - Timers
      - Analog to digital converters
      - Communications subsystem
      - …

Enabling/Disabling interrupts

Remember SREG (Status Register)?

I	T	H	S	V	N	Z	C

- I is the global interrupt enable bit
- Setting I=1 will enable all interrupts (except ones with individual enable/disable bit)
- Setting I=0 will disable all interrupts
- I is cleared when an interrupt occurs and set when reti instruction (return from interrupt) is executed
- Programmer can use sei and cli instructions to set and clear the I
- If I is set in ISR, nested interrupts are allowed
- SREG is not saved automatically by hardware when entering an ISR
  - Should be pushed into stack by software at the start of the ISR and retrieved at the end

Interrupt Service Routine

When an (enabled) interrupt occurs

- Microcontroller completes the current instruction
- Stores the address of the next instruction in the stack and clears the global interrupt enable bit
- Starts executing the instructions from the ISR corresponding to the interrupt source
- ISR continues until the return from interrupt instruction (reti) is seen
- Program control then returns back to the main program and the I bit is set

Where are the ISRs located?

- A vector table at the start of the program memory is referred by the microcontroller
- Usually contains a jump instruction for each interrupt

Interrupt vectors

Part of the interrupt vector of ATmega16

The lower the address the higher is the priority level

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow

•
•
•

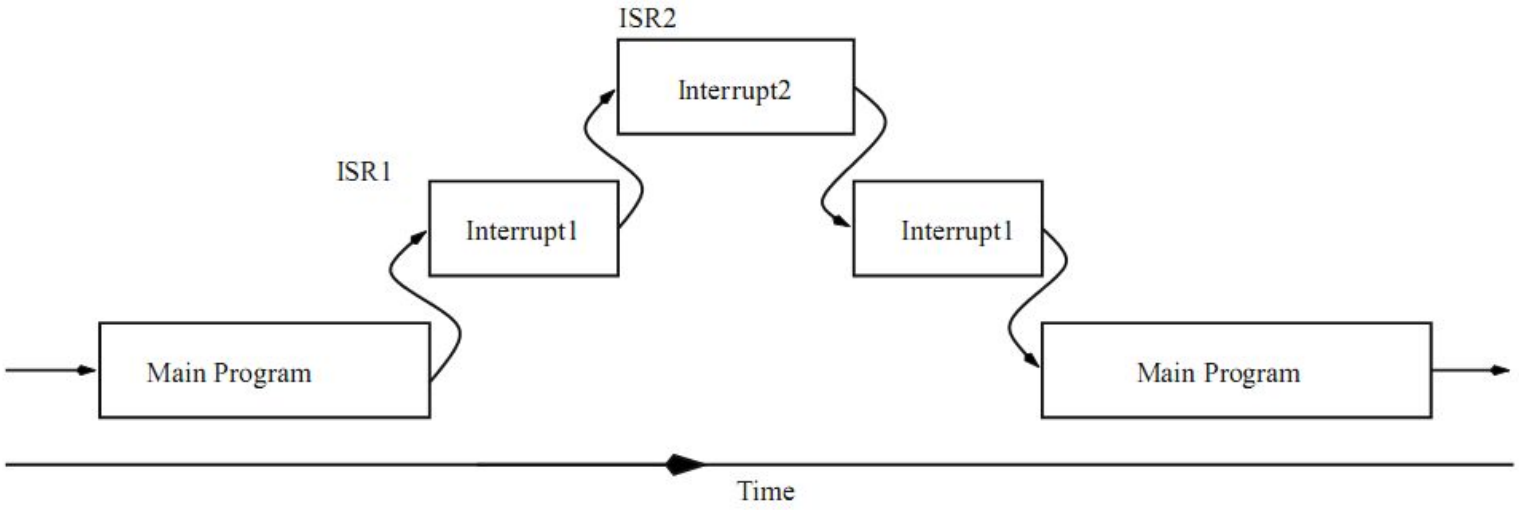
Interrupt vectors

```
org 0
jmp    RESET          ; Reset handler
jmp    EXT_INT0       ; External interrupt 0 handler
jmp    EXT_INT1       ; External interrupt 0 handler
jmp    TIM2_COMP      ; Timer2 compare handler
.
.
.
```

Nested interrupts

The user software ISR can write logic one to the I-bit to enable nested interrupts

The ISR of a lower priority interrupts may be interrupted by an interrupt of a higher priority



External interrupts

Can be triggered in different ways

- rising edge-triggered
- falling edge-triggered
- change-triggered
- level-triggered

ATmega16 as an example

MCU Control Register - MCUCR

				ISC11	ISC10	ISC01	ISC00
7							0

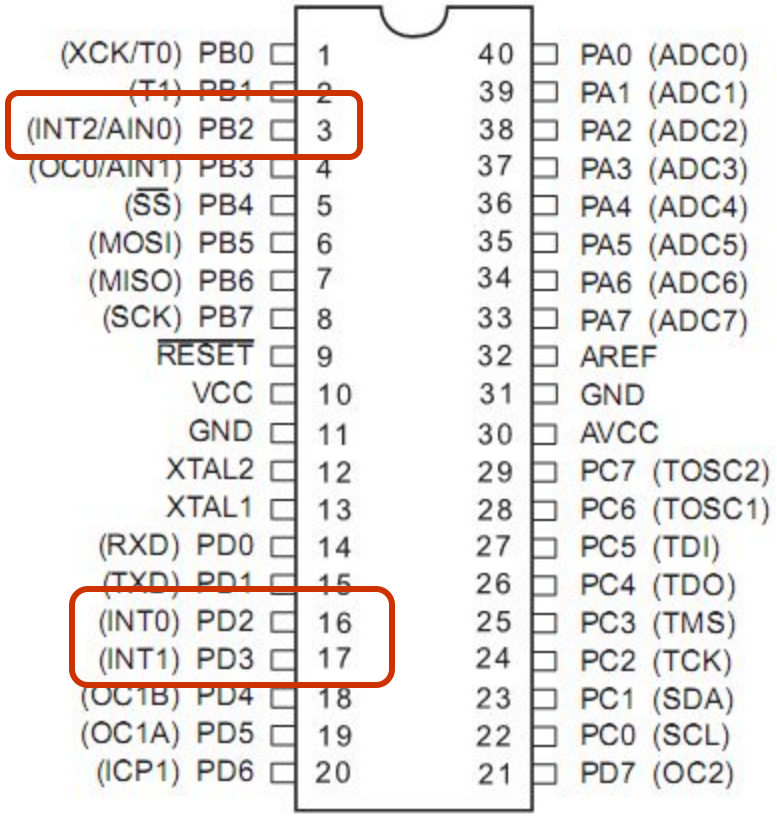
INT1
00: low level
01: logic change
10: falling edge
11: rising edge

INT0
00: low level
01: logic change
10: falling edge
11: rising edge

MCU Control and Status Register - MCUCSR

	ISC2						
7							0

INT2
0: falling edge
1: rising edge



External interrupts

External interrupt 1 is enabled when
- (INT1 is 1 in GICR) and (I is 1 in SREG)
- Activity on interrupt pin will cause an interrupt even if the pin is set as an output

In response to interrupt request on INT1 pin
- INTF1 will be set to 1 in GIFR (Even if the interrupt is disabled)
- The flag is cleared when the interrupt routine is executed
- Alternatively, the flag can be cleared by writing a logical one to it
- This flag is always cleared when INT1 is configured as a level interrupt

General Interrupt Control Register - GICR

INT1	INT0	INT2					
7							0

0: interrupt disabled
1: interrupt enabled

General Interrupt Flag Register - GIFR

INTF1	INTF0	INTF2					
7							0

Notes:
- INTFx flag sets when corresponding interrupt occurs.
- INTFx flag reset by executing ISR or writing logic one to flag.

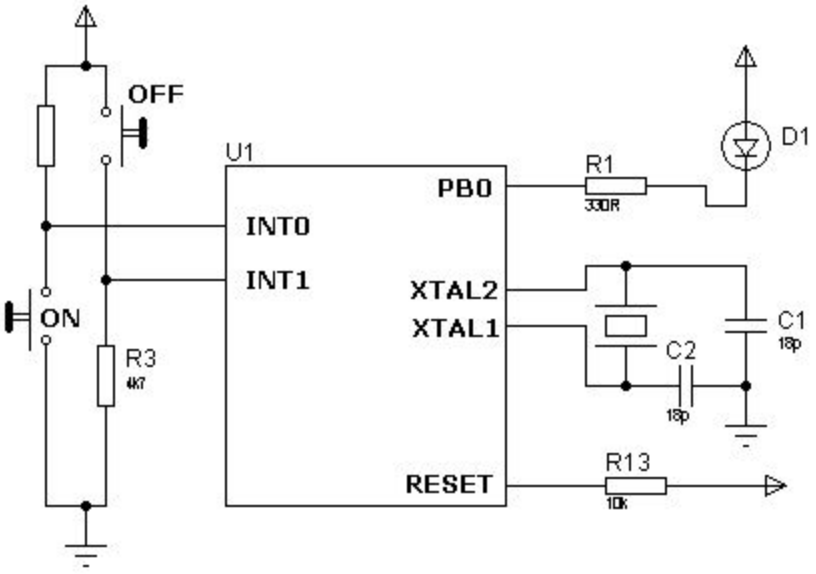
External interrupt example

An example circuit to test external interrupt functionality

- One of the switches turns the LED on and the other turns it off

What should the programmer do?

- Initialize PB0 as output
- Write proper ISRs for INT0, INT1
- Set INT0 to be falling edge-triggered
- Set INT1 to be rising edge-triggered
- Enable INT0, INT1
- Enable interrupts globally



External interrupt example

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

volatile uint8_t led;    // Variable to hold LED status

ISR(INT0_vect)
{
    led = 1; // On state
}

ISR(INT1_vect)
{
    led = 0; // Off state
}

int main()
{
    // ISC10=0
    MCUCR &= ~(1<<ISC00);

    // ISC00=1, ISC01=1, ISC11=1
    MCUCR |= (1<<ISC01) | (1<<ISC10) | (1<<ISC11);

    // INT1=1, INT0=1
    GICR |= (1<<INT0) | (1<<INT1);

    led = 0;

    sei();
    while (1)
    {
        // Update led state
        if (led)
            PORTB &= ~(1<<PB0);
        else
            PORTB |= (1<<PB0);
    }
}
```

Internal interrupts

Will be discussed as we study each internal I/O peripheral