CSSE2010/CSSE7201
Learning Lab 15

AVR Interrupts

School of Information Technology and Electrical Engineering
The University of Queensland

Today

- Lab Preparation tasks
- Timer control revisited
- Interrupts
- Creating a stopwatch

- Take notes as you go – you’ll need the answers
- You’ll need to refer to the ATmega324A datasheet also

Review of Preparation Tasks

- Discuss the preparation task 1 – the stopwatch controller
- Check the state diagram
  - Will it do what was asked?
    - There are many right answers
  - Does it deal with buttons being pushed AND then released?
    - Start/stop action should take effect on button push
    - Reset action should take effect on button release
    - What if a button is held down?
  - What if both buttons are pressed at once
    - And released in either order?
- Check the timer prescale and output compare values

Timer/Counter 1 Control Values

- We will setup timer / counter 1 to reach a compare match (A) every 10ms (i.e. 100 times per second)
  - We will divide 8MHz AVR clock by 8
  - We will reset count when we reach compare match (i.e. CTC mode)
  - No output pins to change
- What value (decimal) needs to go in OCR1A?
  - (Will be split into two bytes OCR1AH, OCR1AL. Just work out 16 bit value.)
- What values need to go in TCCR1A & TCCR1B?
  - See pages 136 to 139 of datasheet – or previous lab slides

Reminder about Interrupts (Interrupts were covered in lecture 17)

- Interrupt
  - **Hardware event** happens, e.g.
    - Rising edge on pin
    - Timer reaches output compare value
    - Timer overflows
  - Software flow is interrupted
  - **Interrupt service routine** is executed
    - General purpose & status register values must be preserved (saved & restored)
      - C interrupt handlers take care of this automatically
  - Control returned to original software flow

I bit in Status Register

- I bit – indicates whether interrupts are enabled globally or not (bit 7 of SREG)
  - 1 = interrupts enabled
  - 0 = interrupts disabled
- To enable
  - Assembly: **`sei`** C: **`sei();`**
- To disable
  - Assembly: **`cli`** C: **`cli();`**
- Individual interrupts must be enabled also

Recall from lecture 17: Setting up interrupts

Besides the handler, also need to:

- Set up conditions for interrupt
- Enable that particular interrupt
  - An I/O register bit for each interrupt controls whether that interrupt is enabled or not – usually there is an interrupt mask register
- Clear the specific interrupt flag (to ensure interrupt doesn’t trigger immediately) – usually there is an interrupt flag register
  - Usually done by writing **1** to some I/O register bit
- Turn on interrupts globally
  - **`sei()`** macro (same as **`sei`** assembly language instruction)

Recall from lecture 17: AVR Interrupts in C

- `#include <avr/interrupt.h>`
- Interrupt handler can be written as a C function, using a special ISR macro, e.g.

```
ISR(INT0_vect) {
        … /* code here */
}
```

- This is the handler for the INT0 external interrupt
    - Interrupt vector table setup and register saving/restoring happen automatically
    - Use source name from datasheet (page 69-70) followed by _vect
- See AVR LibC manual – avr/interrupt.h documentation

Recall from lecture 17: Volatile variables in C

- Example:

```
volatile uint8_t count;
```

- Where a variable can be changed outside the normal flow of a program (e.g. in an interrupt handler) it should be declared as volatile
  - Prevents code optimiser from assuming variable value doesn’t change and optimising code away

Setting up interrupts - Example 1
Interrupt on Timer 1 Output Compare Match A

- Handler is `ISR(TIMER1_COMPA_vect) { … }`
- Set up conditions for interrupt
  - Set output compare value: OCR1A = …;
  - Ensure timer is started: TCCR1A = …; TCCR1B = …;
- Enable that particular interrupt – set bit in I/O register
  - **OCIE1A** bit (bit 1) in **TIMSK1** register (see pp 142)
- Clear the specific interrupt flag
  - Write 1 to **OCF1A** bit in **TIFR1** register (see pp 143)
- Turn on interrupts globally
  - `sei();`

Setting up interrupts - Example 2 Interrupt on Rising Edge on INT0 pin (same as pin D2)

- Handler is `ISR(INT0_vect) { … }`
- Set up conditions for interrupt
  - **ISC01** and **ISC00** bits in **EICRA** register should both be 1 for a rising edge interrupt (see pp 75-76)
- Enable that particular interrupt – set bit in I/O register
  - Write 1 to **INT0** bit in **EIMSK** register (see pp 76)
    - Datasheet labels this bit IINT0 – should be INT0
- Clear the specific interrupt flag
  - Write 1 to **INTF0** bit in **EIFR** register (see pp 76)
    - Datasheet labels this bit IINTF0 – should be INTF0
- Turn on interrupts globally
  - `sei();`

Exercises

1. Create project, use base lab15-1.c code from Blackboard
   - Modify the code where indicated - use earlier register values & examples
   - Add controller code based on your state diagram
   - Compile, simulate (set breakpoint in handler – make sure it is reached)
     - Check the cycle counter is as expected
   - Check that it works on the board as expected
2. Start from lab15-2.c code from Blackboard
   - Add a stop/start push button based on an external interrupt – rising edge on pin INT0 (D2) – see slide 11
   - Set up interrupt (rising edge, enable interrupt, clear flag)
   - Compile, simulate, download, test
3. Add a reset button – interrupt on falling edge on pin INT1(D3)
4. Extension: Modify stopwatch to display seconds (10, 11, 12…) after reaching 9.9s instead of wrapping around.