Interrupts

- Fundamental concept in computation
- Interrupt execution of a program to "handle" an event
  - Don’t have to rely on program relinquishing control
  - Can code program without worrying about others
- Issues
  - What can interrupt and when?
  - Where is the code that knows what to do?
  - How long does it take to handle interruption?
  - Can an interruption be, in turn, interrupted?
  - How does the interrupt handling code communicate its results?
  - How is data shared between interrupt handlers and programs?

What is an Interrupt?

- Reaction to something in I/O (human, comm link)
- Usually asynchronous to processor activities
- "interrupt handler" or "interrupt service routine" (ISR) invoked to take care of condition causing interrupt
  - Change value of internal variable (count)
  - Read a data value (sensor, receive)
  - Write a data value (actuator, send)

Interrupts

- Code sample that does not interrupt
  ```c
  char SPI_SlaveReceive(void) {
    /* Wait for reception complete */
    while(!(SPSR & (1<<SPIF)));
    /* Return data register */
    return SPDR;
  }
  ```
- Instead of busy waiting until a byte is received the processor can generate an interrupt when it sets SPIF
  ```c
  SIGNAL(SIG_SPI) {
    RX_Byte = SPDR
  }
  ```

Saving and Restoring Context

- Processor and compiler dependent
- Where to find ISR code?
  - Different interrupts have separate ISRs
- Who does dispatching?
  - Direct
    - Different address for each interrupt type
  - Indirect
    - One top-level ISR
    - Switch statement on interrupt type
- A mix of these two extremes?

Saving and Restoring Context

- How much context to save?
  - Registers, flags, program counter, etc.
  - Save all or part?
  - Agreement needed between ISR and program
- Where should it be saved?
  - Stack, special memory locations, shadow registers, etc.
  - How much room will be needed on the stack?
  - Nested interrupts may make stack reach its limit – what then?
- Restore context when ISR completes

Ignoring Interrupts

- Can interrupts be ignored?
  - It depends on the cause of the interrupt
  - No, for nuclear power plant temperature warning
  - Yes, for keypad on cell phone (human timescale is long)
- When servicing another interrupt
  - Ignore others until done
  - Can’t take too long – keep ISRs as short as possible
    - Just do a quick count, or read, or write – not a long computation
- Interrupt disabling
  - Will ignored interrupt "stick"?
    - Rising edge sets a flip-flop
  - Or will it be gone when you get to it?
    - Level changes again and its as if it never happened
  - Don’t forget to re-enable
Prioritizing Interrupts

- When multiple interrupts happen simultaneously
  - Which is serviced first?
  - Fixed or flexible priority?
- Priority interrupts
  - Higher priority can interrupt
  - Lower priority can’t
- Maskable interrupts
  - “don’t bother me with that right now”
  - Not all interrupts are maskable, some are non-maskable

Interrupts in the ATmega16

- External interrupts
  - From I/O pins of microcontroller
- Internal interrupts
  - Timers
    - Output compare
    - Input capture
    - Overflow
  - Communication units
    - Receiving something
    - Done sending
  - ADC
    - Completed conversion

Interrupt Jump Vector Table

- Fixed location in memory to find first instruction for each type of interrupt
- Only room for one instruction
  - JMP to location of complete ISR

Chain of Events on Interrupt

- Finish executing current instruction
- Disable all interrupts
- Push program counter on to stack
- Jump to interrupt vector table
- Jump to start of complete ISR
- Save any context that ISR may otherwise change
  - Registers and flags must be saved within ISR and restored before it returns – this is very important!
  - Re-enable interrupts if nested interrupts are ok
    - Complete ISR’s code
    - Re-enable interrupts upon return
    - Jump back to next instruction before interruption

Shared Data Problem

- When you use interrupts you create the opportunity for multiple sections of code to update a variable.
- This might cause a problem in your logic if an interrupt updates a variable between two lines of code that are directly dependent on each other (e.g., if statement)
- One solution is to create critical sections where you disable the interrupts for a short period of time while you complete your logic on the shared variable

    cli();
    …..critical section code goes here…..
    sei();

External Interrupts

- Special pins: INT0, INT1, INT2
  - Can interrupt on edge or level
  - Can interrupt even if set to be output pins
  - Implements “software interrupts” by setting output

    MCS Control Register – MCSR
    The MCS Control Register contains control bits for interrupt access control and general MCS functions.

    | RS1 | RS0 | Description |
    |-----|-----|-------------|
    | 0   | 0   | The low level of INT1 generates an interrupt request. |
    | 0   | 1   | Any logical change on INT1 generates an interrupt request. |
    | 1   | 0   | The rising edge of INT1 generates an interrupt request. |
    | 1   | 1   | The falling edge of INT1 generates an interrupt request. |
Closer Look at a Timer/Counter

- Timer0/Counter0
  - Clear timer on compare match (auto reload)
  - Prescaler (divide clock by up to 1024)
  - Overflow and compare match interrupts
  - Registers
    - Configuration
    - Count value
    - Output compare value

Timer/Counter Registers

- Timer/Counter Control Register TCCR0
  - Bit 6-5 = COM0<1:0>: Compare Match Output Mode
    - Defined names for each register and bit
    - Set timer to clear on match
    - Set prescaler to 1024
    - Set count value to compare against
    - Set timer to interrupt when it reaches count

Defined names for each register and bit

- TCCR0 = (1<<WGM01) | (1<<CS02) | (1<<CS00);
- OCR0 = 150;
- TIMSK = (1<<OCIE0);
Writing an Interrupt Handler in C

- Set and clear interrupt enable
  - sei();
  - cli();
- Interrupt handler
  - SIGNAL(SIG_OUTPUT_COMPARE0)
    ```c
    i++;
    ```
- Setting I/O registers
  - TCCR0 = (1<<WGM01) | (1<<CS02) | (1<<CS00);
- Enabling specific interrupts
  - TIMSK = (1<<OCIE0);

Analog to digital conversion

- Use charge-redistribution technique
  - no sample and hold circuitry needed
  - even with perfect circuits quantization error occurs
- Basic capacitors
  - sum parallel capacitance

A-to-D – sample

- During the sample time the top plate of all capacitors is switched to reference low \( V_L \)
- Bottom plate is set to unknown analog input \( V_X \)
- \( Q = CV \)
- \( Q_S = 16 \times (V_X - V_L) \)

A-to-D – successive approximation

- Each capacitor successively switched from \( V_L \) to \( V_H \)
  - Largest capacitor corresponds to MSB
- Output of comparator determines bottom plate voltage of cap
  - \( > 0 \): remain connected to \( V_H \)
  - \( < 0 \): return to \( V_L \)
Suppose $V_n = 21/32 \ (V_H - V_L)$ and already sampled

- Compare after shifting half of capacitance to $V_H$
  - $V_I$ goes up by $+8/16 \ (V_H - V_I) - 8/16 \ (V_L - V_I)$
  - Original $V_L - V_I$ goes down and becomes $V_L - (V_I + 0.5 \ (V_H - V_L)) = V_L - V_I - 0.5 \ (V_H - V_L)$
- Output > 0

Input sample of 21/32
- Gives result of 1010 or 10/16 = 20/32
- 3% error
Closer Look at A-to-D Conversion

- Needs a comparator and a D-to-A converter
- Takes time to do successive approximation
- Interrupt generated when conversion is completed

A-to-D Conversion on the ATmega16

- 10-bit resolution (adjusted to 8 bits as needed)
- 65-260 usec conversion time
- 8 multiplexed input channels
- Capability to do differential conversion
  - Difference of two pins
  - Optional gain on differential signal (amplifies difference)
- Interrupt on completion of A-to-D conversion
- 0-VCC input range
- 2*LSB accuracy (2 * 1/1024 = ~0.2%)
- Susceptible to noise – special analog supply pin (AVCC) and capacitor connection for reference voltage (AREF)

A-to-D Conversion (cont’d)

- 10-bit resolution (adjusted to 8 bits as needed)
- 65-260 usec conversion time
- 8 multiplexed input channels
- Capability to do differential conversion
  - Difference of two pins
  - Optional gain on differential signal (amplifies difference)
- Interrupt on completion of A-to-D conversion
- 0-VCC input range
- 2*LSB accuracy (2 * 1/1024 = ~0.2%)
- Susceptible to noise – special analog supply pin (AVCC) and capacitor connection for reference voltage (AREF)

A-to-D Conversion (cont’d)

- Single-ended or differential
- 1 of 8 single-ended
- ADCx – ADC1 at 1x gain
- ADCx – ADC0 at 10x
- ADCx – ADC0 at 200x
- ADCx – ADC2 at 10x
- ADCx – ADC3 at 200x
- ADCx{0,1,2,3,4,5} – ADC2 at 1x
Writing an Interrupt Handler in C (again)

- Ensure main program sets up all registers
- Enable interrupts as needed
- Enable global interrupts (SEI)
- Write handler routine for each enabled interrupt
  - What if an interrupt occurs and a handler isn’t defined?
  - Make sure routine does not disrupt others
  - Data sharing problem
  - Save any state that might be changed (done by compiler)
  - Re-enable interrupts upon return
    - done by compiler with RETI

Power modes

- Processor can go to “sleep” and save power
- Different modes put different sets of modules to sleep
  - Which one to use depends on which modules are needed to
  - wake up processor
  - Timers, external interrupts, ADC, serial communication lines, etc.
  - set_sleep_mode (mode);
  - sleep_mode();

Power modes (cont’d)

- Wake up sources and active clocks

<table>
<thead>
<tr>
<th>Module</th>
<th>Clock</th>
<th>Power</th>
<th>Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td></td>
<td></td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>External</td>
<td></td>
<td></td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>Ext. I/O</td>
<td></td>
<td></td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>Serial</td>
<td></td>
<td></td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>ADC</td>
<td></td>
<td></td>
<td>0, 1, 2</td>
</tr>
</tbody>
</table>

Note:
- Modes 0, 1, and 2 mean the module is active.
- Mode 3 puts the module into low-power mode.
- Mode 4 puts the module into sleep mode.
- Mode 5 allows the module to be powered down.
- Mode 6 allows the module to be powered up.
- Mode 7 allows the module to be powered on.
- Mode 8 allows the module to be powered off.
- Mode 9 allows the module to be powered back on.
- Mode 10 allows the module to be powered back off.
- Mode 11 allows the module to be powered back on.
- Mode 12 allows the module to be powered back off.
- Mode 13 allows the module to be powered back on.
- Mode 14 allows the module to be powered back off.
- Mode 15 allows the module to be powered back on.
- Mode 16 allows the module to be powered back off.
- Mode 17 allows the module to be powered back on.
- Mode 18 allows the module to be powered back off.
- Mode 19 allows the module to be powered back on.
- Mode 20 allows the module to be powered back off.
- Mode 21 allows the module to be powered back on.
- Mode 22 allows the module to be powered back off.
- Mode 23 allows the module to be powered back on.
- Mode 24 allows the module to be powered back off.
- Mode 25 allows the module to be powered back on.
- Mode 26 allows the module to be powered back off.
- Mode 27 allows the module to be powered back on.
- Mode 28 allows the module to be powered back off.
- Mode 29 allows the module to be powered back on.
- Mode 30 allows the module to be powered back off.
- Mode 31 allows the module to be powered back on.
- Mode 32 allows the module to be powered back off.
- Mode 33 allows the module to be powered back on.
- Mode 34 allows the module to be powered back off.
- Mode 35 allows the module to be powered back on.
- Mode 36 allows the module to be powered back off.
- Mode 37 allows the module to be powered back on.
- Mode 38 allows the module to be powered back off.
- Mode 39 allows the module to be powered back on.
- Mode 40 allows the module to be powered back off.
- Mode 41 allows the module to be powered back on.
- Mode 42 allows the module to be powered back off.
- Mode 43 allows the module to be powered back on.
- Mode 44 allows the module to be powered back off.
- Mode 45 allows the module to be powered back on.
- Mode 46 allows the module to be powered back off.
- Mode 47 allows the module to be powered back on.
- Mode 48 allows the module to be powered back off.
- Mode 49 allows the module to be powered back on.
- Mode 50 allows the module to be powered back off.
- Mode 51 allows the module to be powered back on.
- Mode 52 allows the module to be powered back off.
- Mode 53 allows the module to be powered back on.
- Mode 54 allows the module to be powered back off.
- Mode 55 allows the module to be powered back on.
- Mode 56 allows the module to be powered back off.
- Mode 57 allows the module to be powered back on.
- Mode 58 allows the module to be powered back off.
- Mode 59 allows the module to be powered back on.
- Mode 60 allows the module to be powered back off.
- Mode 61 allows the module to be powered back on.
- Mode 62 allows the module to be powered back off.
- Mode 63 allows the module to be powered back on.