ECE 354 – Introduction to Lab 2

February 23rd, 2003
Fun Fact

• Press release from Microchip:
  “Microchip Technology Inc. announced it provides PICmicro® field-programmable microcontrollers and system supervisors for the Segway Human Transporter (HT) […] The PIC16F87x Flash microcontrollers process sensor data from the inertial monitoring unit and communicate information to the control module. Other PIC16F87x devices located in the battery packs provide monitoring functions. […]”
Lab 1

• Most groups completed Lab 1 – good job!
• Understand how UART works, not just how to use it
• Lab reports due Wednesday
  – Don’t go overboard – be concise
• Questions?
• Were lab times sufficient?
Overview

• Lab 2
• Interrupts
  – Why we need them
  – How to use them
• Timer
• Efficient printing of strings
• Interfacing PICs and PLDs
  – External bus design
  – READ and WRITE transactions
Lab 2

- Interconnect PIC with PLD
- PLD acts as coprocessor
- PIC and PLD communicate via bus
- You have to design bus interface
- Timer on PIC is used to generate periodic interrupts
  - Will make sure your interface is robust
- PLD programmed in VHDL
  - Great example of hardware/software co-design!
- Lab 2 instructions will be available on course web site
Interrupts

• Lab 1 used “polling”
  – What is bad about polling?

• Interrupts
  – Triggered by internal or external events (examples?)
  – Cause program to “interrupt” and treat interrupt
  – After interrupt processing, processing returns to previous code

• What is better about interrupts?
Interrupt Example
What Happens During an Interrupt?

1. Interrupts have to be enabled
   - Bits set in INTCON (internal interrupts) or PIE1 (peripheral interrupts) registers
2. Interrupt stimulus
   - Timer/counter overflows, change on external pin
3. Interrupts automatically disabled
   - Bit 7 of INTCON, why?
4. Jump to interrupt vector
   - Address 0x4, typically calls interrupt subroutine
5. Jump to and execute interrupt service routine
   - PIR1 register identifies which interrupt has triggered
6. Return to previous code
   - RETFIE instruction, also enables interrupts
# INTCON Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>GIE: Global Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Enables all unmasked interrupts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Disables all interrupts</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PEIE: Peripheral Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Enables all unmasked peripheral interrupts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Disables all peripheral interrupts</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>T0IE: TMR0 Overflow Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Enables the TMR0 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Disables the TMR0 interrupt</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INTE: RB0/INT External Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Enables the RB0/INT external interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Disables the RB0/INT external interrupt</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RBIE: RB Port Change Interrupt Enable bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Enables the RB port change interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Disables the RB port change interrupt</td>
<td></td>
</tr>
</tbody>
</table>
PIE1 Register

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PSPIE</strong>&lt;sup&gt;(1)&lt;/sup&gt;</td>
<td><strong>ADIE</strong></td>
<td><strong>RCIE</strong></td>
<td><strong>TXIE</strong></td>
</tr>
<tr>
<td><strong>SSPIE</strong></td>
<td><strong>CCP1IE</strong></td>
<td><strong>TMR2IE</strong></td>
<td><strong>TMR1IE</strong></td>
</tr>
</tbody>
</table>

**bit 7**

**PSPIE**: Parallel Slave Port Read/Write Interrupt Enable bit
- 1 = Enables the PSP read/write interrupt
- 0 = Disables the PSP read/write interrupt

**bit 6**

**ADIE**: A/D Converter Interrupt Enable bit
- 1 = Enables the A/D converter interrupt
- 0 = Disables the A/D converter interrupt

**bit 5**

**RCIE**: USART Receive Interrupt Enable bit
- 1 = Enables the USART receive interrupt
- 0 = Disables the USART receive interrupt

**bit 4**

**TXIE**: USART Transmit Interrupt Enable bit
- 1 = Enables the USART transmit interrupt
- 0 = Disables the USART transmit interrupt
## PIR1 Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><strong>PSPIF</strong>&lt;sup&gt;(1)&lt;/sup&gt;</td>
<td>Parallel Slave Port Read/Write Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = A read or a write operation has taken place (must be cleared in software)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = No read or write has occurred</td>
</tr>
<tr>
<td>6</td>
<td><strong>ADIF</strong></td>
<td>A/D Converter Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = An A/D conversion completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = The A/D conversion is not complete</td>
</tr>
<tr>
<td>5</td>
<td><strong>RCIF</strong></td>
<td>USART Receive Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = The USART receive buffer is full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = The USART receive buffer is empty</td>
</tr>
<tr>
<td>4</td>
<td><strong>TXIF</strong></td>
<td>USART Transmit Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = The USART transmit buffer is empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = The USART transmit buffer is full</td>
</tr>
</tbody>
</table>

<sup>(1)</sup> Note: PSPIF flag should be cleared in software after a read or write operation.
# RETFIE Instruction

<table>
<thead>
<tr>
<th>RETFIE</th>
<th>Return from Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>`[label] RETFIE</td>
</tr>
<tr>
<td>Operands:</td>
<td>None</td>
</tr>
<tr>
<td>Operation:</td>
<td>TOS → PC, 1 → GIE</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>None</td>
</tr>
</tbody>
</table>
Interrupt Code Sample

org H’000’ ; Reset vector
goto Mainline; Location of start of program

org H’004’ ; Interrupt vector
goto IntServ ; Start of int service routine

Mainline ....
....

org H’100’ ; put service routing at 0x100

IntServ .... ; first inst. of service routine
....
retfie ; return from interrupt instr.
Saving State

- Some “state” of PIC is not preserved during interrupt
  - What is “state”?  
  - What is preserved?  
  - What can get lost?
- How to avoid problems:
  - Preserve state (w, STATUS) before interrupt processing or
  - Do not change state during interrupt processing (difficult)
- See section 12.11 of data sheet and Peatman section 4.5
  - Note: use of swapf instruction instead of movf
Interrupt Limitations

• What happens if interrupt is too long?
  – Other critical interrupts cannot be handled or
  – Livelock (not on PIC due to lack of recursion)

• Beware of function calls in interrupt service routine
  – Stack overflow could happen
  – max nesting of program + max nesting of ISR + 1 = 8
Timer Interrupts

- The PIC has several built-in timers
- timer0 is a simple 8-bit counter
  - External or internal clock
  - Prescaler possible

See Peatman pp. 100–103, data sheet pp. 47–49
String Printing

• Printing strings on terminal can be awkward
  – Example: print “Hello”
    
    ```
    call wait ; subroutine which checks PIR bit 4.
    movlw 'H' ; send ASCII w char. to W
    movwf TXREG ; send w char. to UART trans. buffer
    call wait ; subroutine which checks PIR bit 4.
    movlw 'e' ; send ASCII o char. to W
    movwf TXREG ; send o char. to UART trans. buffer
    call wait ; subroutine which checks PIR bit 4.
    movlw 'l' ; send ASCII w char. to W
    movwf TXREG ; send w char. to UART trans. Buffer
    ...
    ```

• Any ideas?
Advanced String Handling (1)

- Store text into memory
- Process one character a time
- Consider:

```
ORG 0x0
    goto Start ; jump to the start of the program
ORG 0x5
sub1  nop ; start of a subroutine
    ...
    return ; return from subroutine
ORG 0x30
Start
    ...
    call sub1 ; call the routine
    nop ; first instr. after return
```
Advanced String Handling (2)

• **call instruction**:

<table>
<thead>
<tr>
<th>CALL</th>
<th>Call Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] CALL k</td>
</tr>
<tr>
<td>Operands:</td>
<td>0 ≤ k ≤ 2047</td>
</tr>
<tr>
<td>Operation:</td>
<td>(PC)+ 1 → TOS,</td>
</tr>
<tr>
<td></td>
<td>k → PC&lt;10:0&gt;,</td>
</tr>
<tr>
<td></td>
<td>(PCLATH&lt;4:3&gt;) → PC&lt;12:11&gt;</td>
</tr>
</tbody>
</table>

• **return instruction**

<table>
<thead>
<tr>
<th>RETURN</th>
<th>Return from Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] RETURN</td>
</tr>
<tr>
<td>Operands:</td>
<td>None</td>
</tr>
<tr>
<td>Operation:</td>
<td>TOS → PC</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>None</td>
</tr>
</tbody>
</table>
Advanced String Handling (3)

- `retlw` instruction returns and puts value into w

<table>
<thead>
<tr>
<th>RETLW</th>
<th>Return with Literal in W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td><code>[label] RETLW k</code></td>
</tr>
<tr>
<td>Operands:</td>
<td><code>0 ≤ k ≤ 255</code></td>
</tr>
<tr>
<td>Operation:</td>
<td><code>k → (W);</code></td>
</tr>
<tr>
<td></td>
<td><code>TOS → PC</code></td>
</tr>
</tbody>
</table>

- Call subroutine for each character
- Use `retlw` to return each character and place into w
- Send w to UART
String Handling Code (1)

```
BANKORAM EQU H’20’ ; equate a constant to hex 20.
ORG BANKORAM ; reserve space in DATA MEMORY
cblock ; create a pointer in bank 0 at 0x20
POINTER ; name of value
endc

ORG 0x0
goto Start ; jump to the start of the program

ORG 0x5
sub1 movf POINTER, W ; move value in POINTER to W
addwf PCL, F ; add value to PC
retlw A’H’
retlw A’e’
retlw A’l’
retlw A’l’
retlw A’o’
retlw 0
RETURN ; shouldn’t get here
```
String Handling Code (2)

ORG 0x30

Start
  clrf POINTER
Loop
  call sub1

  ... ; check if return value is 0
  btfs status, z ; branch if not 0
  goto Done ; else done

  ... ; check bit 4 in PIR
  movwf TXREG

  ... ; increment POINTER
  goto Loop ; print another character

  ORG 0x60

Done
  nop
  goto Done
String Handling

- A bit complex
  - Simplified by ‘dt’ assembler directive
    dt “Hello” translates into
    retlw A’H’
    retlw A’e’
    retlw A’l’
    retlw A’l’
    retlw A’l’
    retlw A’o’

- Note: does not terminate string (with A’0’)!
- Saves memory space
- More easily modifiable
- You need to understand how it works, but you don’t need to use it.
OK, Who’s Confused?

• See Peatman Section 8.6, pp.154–157
Data Exchange Via Bus

- Multiple devices can be connected through a bus
  - We connect PIC to PLD
- PIC should be able to read from and write to PLD
  - PLD acts as coprocessor
  - For example: write value to 0x1 and 0x2 and read f(0x1, 0x2) from 0x3
Bus Interface

- A bus needs:
  - Address, data, control signals
- For Lab 2:
  - Port A: four bit address value
  - Port B: four bit data input
  - Port D: four bit data output
  - Port C: up to six control signals
- Build “read” and “write” transactions
  - Need to be robust and reliable (interrupts!)
Bus Issues

- Control signals indicate valid address/data

  - address 0
  - address 1
  - address 2
  - address 3
  - address valid

- PIC and PLD use same clock (synchronous)
  - Interrupts on PIC can cause delays!

- Transactions needs to be acknowledged (why?)
  - PLD acks when WRITE result was received
  - PIC acks when READ result was received

- The logic analyzer is your best friend 😊
Example WRITE

address 0

address 1

address 2

address 3

write data 0

write data 1

write data 2

write data 3

address/data valid

PLD ack
Example READ

address 0

address 1

address 2

address 3

read data 0

read data 1

read data 2

read data 3

address valid

data ack

PIC ack
Bus Implementation

- Need signal to distinguish READ and WRITE
- There are better, simpler ways
  - Consider using control bits for multiple purposes
- Use PLD state machines
  - One for address
  - One for read
  - One for write
- Implement on PLD using VHDL
  - Basically what you have done in ECE 353
Lab 2

• BEFORE YOU START: READ!
  – Lab Assignment
  – PIC data sheet section 12.10 – 12.11 (interrupts)
  – Peatman, chapter 4 (timer and interrupts)

• Think about how you want to split work
  – If you separate PIC and PLD design, make sure you have a good bus interface!

• Think about steps to take to get it working

• Start working early!
  – You will need more time than for Lab 1
  – Lab due March 12th