ECE 354 – Computer Systems Lab II

D/A and A/D Conversion
Labs Etc.

- Lab 2 reports graded
- Lab 3
  - Demos Thursday and Friday
  - Have logic analyzer printout for SRAM read & write
  - Reports due Thursday next week
- Lab 4 quiz
  - Starts 4/22 (Thursday next week)
  - Ends 4/26 (Monday following week)
Midterm Exam

- Wednesday 4/21, 2:30 in class
- Closed Books
- Should require more “understanding” than “memorization”
  - There will be “programming questions”
  - You’ll get the sheet with the PIC instruction set
  - I will not ask what bit 4 of the STATUS register does 😊
  - But I might ask what an interrupt is, how it’s used, and what registers are involved with it.

- Might contain very simple concepts from Lab 4
- 90 minutes
Lab 4 Overview

- Analog communication between two PICs
  - Uses A/D and D/A
- Basic functionality
  - Enter character on terminal
  - 1st PIC converts character to analog voltage
  - Analog value transmitted via wire to 2nd PIC
  - 2nd PIC converts voltage back to digital character
  - 2nd PIC displays character on terminal
- Configurable:
  - PIC sends or receives, how many characters (voltage levels) are allowed
Why Analog?

- Not everything is digital!
  - Analog circuits are still necessary
- Physical phenomena are often analog
  - Many sensors are analog (potentiometer, phototransistor, thermo-sensor, microphone)
  - Many actuators are analog (solenoid, speakers)
  - Some signals need to be processed in analog domain before conversion to digital (amplification, filtering, linearization)
- Requires conversion between analog and digital domain
  - DAC: digital to analog converter
  - ADC: analog to digital converter
  - PIC has both
Digital and Analog Conversion

- ADC transfer function:
  - 10-bit ADC converter
  - 1024 voltage levels between 0V and $V_{\text{REF}}$
  - 10-bit digital value
- Usually $V_{\text{DD}}=V_{\text{REF}}$
- How does D/A and A/D conversion work?
D/A Conversion

• How can a digital “value” be converted into a corresponding analog voltage?
D/A Conversion

- Need to generate analog voltage that corresponds to 10-bit digital value
  - PIC uses Pulse Width Modulation (PWM)
- PWM: Use square wave generator
  - Period and duty cycle adjustable
  - Use low-pass filter to “smooth out” wave
  - DC value depends on length of duty cycle
- Shorter period (higher frequency) gives better results
- PIC
  - Period and duty cycle set through registers
Pulse Width Modulation

- Pulse width modulated signal
- PWM signal, low-pass filtered with short time constant
- PWM signal, low-pass filtered with long time constant
PWM on PIC

- Registers involved:
  - PR2 register: PWM period
  - CCPR1L and CCP1CON<5:4>: 10-bit duty cycle
- Basic operation:
  - Start of period
    - Timer TMR2 cleared
    - CCP1 pin set to high
    - 10-bit duty cycle latched to CCPR1H
  - When TMR2 = CCPR1H
    - Clear CCP1 (duty cycle over)
  - When TMR2 = PR2
    - New period starts

Note 1: The 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time-base.
# CCP1CON Register

**REGISTER 8-1: CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS: 17h/1Dh)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-6</td>
<td>Unimplemented: Read as '0'</td>
</tr>
<tr>
<td>5-4</td>
<td>CCPxX: CCPxY: PWM Least Significant bits</td>
</tr>
<tr>
<td></td>
<td>Capture mode: Unused</td>
</tr>
<tr>
<td></td>
<td>Compare mode: Unused</td>
</tr>
<tr>
<td></td>
<td>PWM mode: These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.</td>
</tr>
<tr>
<td>3-0</td>
<td>CCPxM3: CCPxM0: CCPx Mode Select bits</td>
</tr>
</tbody>
</table>

- 0000 = Capture/Compare/PWM disabled (resets CCPx module)
- 0100 = Capture mode, every falling edge
- 0101 = Capture mode, every rising edge
- 0110 = Capture mode, every 4th rising edge
- 0111 = Capture mode, every 16th rising edge
- 1000 = Compare mode, set output on match (CCPxIF bit is set)
- 1001 = Compare mode, clear output on match (CCPxIF bit is set)
- 1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
- 1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
- 11xx = PWM mode
Timer 2 Prescaler

- Prescaler determines effective clock rate for TMR2
- Postscaler irrelevant for us
  - Used if Timer 2 needs to drive additional component at different frequency
- PMW period formula:

\[
PMW\,\text{period} = [(PR2) + 1] \times 4 \times T_{OSC} \times (\text{TMR2 prescaler})
\]
PWM Setup

Setup steps:

- Set PWM period by writing to PR2 register
- Set PWM duty cycle by writing to CCPR1L register and CCP1CON<5:4> bits
- Make CCP1 pin output by clearing the TRISC<2> bit
- Set TMR2 prescale value enable Timer 2 by writing to TCON2
- Configure the CCP1 module for PWM operation

Example for 20 MHz clock:

<table>
<thead>
<tr>
<th>PWM Frequency</th>
<th>1.22 kHz</th>
<th>4.88 kHz</th>
<th>19.53 kHz</th>
<th>78.12kHz</th>
<th>156.3 kHz</th>
<th>208.3 kHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer Prescaler (1, 4, 16)</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PR2 Value</td>
<td>0xFFh</td>
<td>0xFFh</td>
<td>0xFFh</td>
<td>0x3Fh</td>
<td>0x1Fh</td>
<td>0x17Fh</td>
</tr>
<tr>
<td>Maximum Resolution (bits)</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>5.5</td>
</tr>
</tbody>
</table>
A/D Conversion

- How can we generate digital value of analog voltage?
A/D Conversion

- Use D/A converter to generate different analog values and compare
  - Control logic decides which values to try
  - When comparison complete, best match is put on output
- How can D/A be matched to input in fewest steps?
Successive Approximation

• Matching strategies:
  – Counting conversion (slow)
  – Successive approximation (faster)

• Successive Approximation:
  – Basically binary search:
  – 10 steps instead of 1024
PIC ADC Characteristics (1)

- **“Sample and Hold”**
  - ADC samples for a given time (charges hold capacitor)
  - Then sample value is disconnected from source (”hold”)
  - A/D conversion is performed
  - On completion, ADC can sample again

- **Sampling takes some time**
  - Depends on source impedance (max 10 kΩ)
  - Lower source impedance reduces sample time because hold capacitance charges faster
    (see Peatman Figure 10-5(b))
  - Also depends on temperature, etc.
 PIC ADC Characteristics (2)

- Analog input model and acquisition time formula:

  \[ T_{ACQ} = \text{Amplifier Settling Time} + \text{Hold Capacitor Charging Time} + \text{Temperature Coefficient} \]

  \[ = T_{AMP} + T_c + T_{COFF} \]

  \[ = 2\mu s + T_c + [(\text{Temperature} - 25^\circ C)(0.05\mu s/^\circ C)] \]

  \[ T_c = \text{CHOLD} (R_{IC} + R_{SS} + R_S) \ln(1/2047) \]

  \[ = -120\mu F (1k\Omega + 7k\Omega + 10k\Omega) \ln(0.0004885) \]

  \[ = 16.47\mu s \]

  \[ T_{ACQ} = 2\mu s + 16.47\mu s + [(50^\circ C - 25^\circ C)(0.05\mu s/^\circ C)] \]

  \[ = 19.72\mu s \]
ADCON0 Register

- **ADC enable (bit 0)**
- **Busy/idle (conversion takes some time):**
  - Bit 2 in ADCON0 register
  - Check by polling or enable interrupt
- **Channel selection (bits 5-3):**
  - Selects pins to be used
  - Selects external or internal reference voltage
- **A/D conversion clock setting (bits 7-6)**

<table>
<thead>
<tr>
<th>ADCON0 REGISTER (ADDRESS: 1Fh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W-0</td>
</tr>
<tr>
<td>AD0S1</td>
</tr>
<tr>
<td>bit 7</td>
</tr>
</tbody>
</table>

bit 0
Channel Selection
A/D Conversion Clock Setting

- Time to convert 1 bit must be \( \geq 1.6 \mu s \)
  - Clock setting must be adjusted to external clock

<table>
<thead>
<tr>
<th>AD Clock Source (TAD)</th>
<th>ADCS1:ADCS0</th>
<th>Maximum Device Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2TOSC</td>
<td>00</td>
<td>1.25 MHz</td>
</tr>
<tr>
<td>8TOSC</td>
<td>01</td>
<td>5 MHz</td>
</tr>
<tr>
<td>32TOSC</td>
<td>10</td>
<td>20 MHz</td>
</tr>
<tr>
<td>RC(^{(1,2,3)})</td>
<td>11</td>
<td>(Note 1)</td>
</tr>
</tbody>
</table>

Conversion starts

Holding capacitor is disconnected from analog input (typically 100 ns)

Set GO bit

ADRES is loaded
GO bit is cleared
ADIF bit is set
Holding capacitor is connected to analog input
ADCON1 Register

- Port configuration (bits 3-0)
  - Chooses pins to be digital I/O or analog input (see data sheet)
- Result Format Selection (bit 7)
  - Chooses justification of 10-bit conversion result:

![Diagram showing result format selection with ADFM = 1 and ADFM = 0]
ADC Setup

1. Configure A/D module:
   - Configure analog pins/voltage reference and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D conversion clock (ADCON0)
   - Turn on A/D module (ADCON0)

2. Configure A/D interrupt (if desired):
   - Clear ADIF bit
   - Set ADIE bit
   - Set PEIE bit
   - Set GIE bit

3. Wait required acquisition time

4. Start conversion
   - Set GO_/DONE bit (ADCON0)

5. Wait for A/D conversion to complete (polling or interrupt)

6. Read A/D result from (ADRESH:ADRESL) and clear ADIF bit

7. Goto 1. or 2. wait 2 A/D clock ticks
Reference

- PWM (D/A conversion)
  - PIC data sheet pp. 61-62
  - Peatman Section 6.9 (pp. 112-119)
- A/D conversion
  - PIC data sheet pp. 111-116
  - Peatman Chapter 10
Lab 4

• Lab setup:
  — 2 PICs (available in lab kit)
  — Need 2 terminals

• Send characters coded in analog between terminals
• Low pass filter needs to be adapted to PWM settings
We use adaptable signal coding

Choose between \(2^x\) (\(x=\{1,2,\ldots,6\}\)) symbols:

- Robustness depends on \(x\):
  - Most robust: only two characters and two voltages
    - 512 ADC results can represent one character
  - Most information: 64 characters and voltage ranges
    - 16 ADC results can represent one character
Lab 4 Demo

- Each PIC can be used for transmitting or receiving
  - User can specify function after reset
- User also specifies coding level (1..6)
  - Same on both PICs
  - Requires well-designed ASCII manipulation
- When character is entered on terminal
  - PIC 1 receives character
  - Converts it to analog signal
  - PIC 2 received analog signal
  - Converts it to digital value
  - Prints result on terminal
- Should be robust for low coding levels
Final Comments

- Questions?