Last Time

- Finite state machine implementation in code
- Connecting code to the computer architecture
  - Program counter
  - Status register
  - General registers
  - Assembly language
Today

• Another FSM example
• Analog input/output
Administrivia

• Homework 5 out by tonight
• Project 2 due Thursday
  – Your FSMs should be designed (and implemented)

• AME Faculty candidate talk today:
  Dr. Brian Argrow
  Small UAVs for Ad-Hoc Networking
  3:00, FH 214
FSM Toy Example

• What is the FSM?
A Comment About FSM Implementation in Code

- switch() statements are convenient for selecting between different pieces of code based on state
- But: may not be easy to use when processing events
Event Handling for the Vending Machine

case STATE_10cents:
    // $.10 has already been deposited
    switch(event) {
        case EVENT_NICKEL:    // Nickel
            state = STATE_15cents; // Transition to $.15
            break;
        case EVENT_DIME:      // Dime
            state = STATE_20cents; // Transition to $.2
            break;
        case EVENT_JOLT:
        case EVENT_BUZZ:      // Select Buzzwater
            display_NOT_ENOUGH();
            break;

        case EVENT_NONE:      // No event
            break;            // Do nothing
    };
break;

Use of switch is simple in this case: a small number of events that can all occur in each state
Event Handling in More Complicated Domains

case STATE_FORWARD_SEARCH_LEFT:
    // Moving forward while searching left
    if (sensor[TURRET_LEFT] > 0 || sensor[TURRET_RIGHT] > 0)) {
        // Event: we have found a beacon on the left
        turn_left();
        state = STATE_FORWARD_SEARCH_RIGHT;
    } else{
        // No event: handle actions taken while in this state
        drive_toward_beacon_front();
    }

    break; // DON’T FORGET THE “BREAK”

case FOO:
    :
    :
    :

The “if” statement allows us to capture many different situations (or events) in one
Digital to Analog and Back

• Analog: encoding information using voltage
  – Many sensors use voltage as an output
  – Motors torque is determined by current passing through the motor

• Digital: encoding information with bits

How to move between these?
Digital to Analog Conversion: Pulse Width Modulation

What does this circuit do?
Digital to Analog Conversion: Pulse Width Modulation

• Processor digital pin: generate PWM signal
• RC circuit “smooths” this PWM signal out
• Pulse width determines smoothed voltage
D2A: Pulse Width Modulation

- Easy to implement
- But:
  - Smoothed signal may not be smoothed enough
  - Filter induces a delay
Digital to Analog Conversion: Resistive Network

• Sometimes need faster response

• Solution: use multiple digital pins
Analog to Digital Conversion

• For a given voltage, what is the digital representation of the voltage?

• Common approach: successive approximation
  – Use a D2A converter to produce a voltage V
  – Compare this with the input voltage Vi
  – If different, then increase/decrease V
  – Repeat (stopping when V is close to Vi)
Last Time

- Relationship between analog and digital encoding of information

- **D2A:**
  - With pulse-width modulation
  - With resistive network

- **A2D:**
  - Successive approximation
Today

- Analog to digital with the Mega8
- Project 3: following air currents
- Interrupts
Administrivia

• Project 2 due today @5:00
  – Demonstration
  – Group report (pdf or postscript)
  – Personal report (raw text only!)

• Homework 5 is available on the web site
  – Due Tuesday
Robot Administrivia

- 3 functional robots right now
- A 4th will be ready early today
- The 5th won’t be ready until sometime Friday
A2D in the Mega8

• The mega8 contains hardware that implements successive approximation
• 5 mega8 pins can be configured as analog input pins
A2D in the Mega8

AVCC: connect to +5V

AREF: (optional) connect to +5V

• Measuring voltages between 0 and +5V

Connect input analog signal to the appropriate ADC pin
A Code Example

// Initialize adc
adc_set_reference(ADC_REF_AREF);       // Use the AREF reference pin
adc_set_adlar(0);                      // For our purposes, always use 0
adc_set_prescalar(ADC_PRESCALAR_128);  // Necessary with 16MHz clock
                                          // and 10 bit resolution

// Turn on ADC Converter
adc_set_enable(ADC_ENABLE);

: :
:
long val;

// Can do the following an arbitrary number of times

adc_set_channel(ADC_CHANNEL_0);        // ADC0
// Actually start a conversion
adc_start_conversion();

<Could go off and do something else for a while>

val = adc_read();       // Read the analog value
Analog Conversion Notes

- All functions are provided in oulib.c
- See oulib.h for the definition of constants

- Can get to the example code from the Atmel HowTo
  www.cs.ou.edu/~fagg/classes/general/atmel
Analog Conversion Notes

• Setting the maximum voltage:

```c
adc_set_reference(ADC_REF_AREF);        // Use the AREF reference pin
```

• Can also used a fixed voltage (+2.56V):

```c
adc_set_reference(ADC_REF_2p56V);
```
Analog Conversion Notes

• Determining how fast the conversion requires:

```
adc_set_prescalar(ADC_PRESCALAR_128);  // Necessary with 16MHz clock
                                       // and 10 bit resolution
```

• Conversion requires:
  128 * 15 / 16000000  seconds
  – Can convert faster, but may not get the full 10-bit resolution
Analog Conversion Notes

- Reading out the value:

```c
val = adc_read();  // Read the analog value
```

- Will receive a value between 0 and 0x3FF (1023)
Project 3

• Problem: follow the air currents
• Task:
  – Orient robot to face wind
  – Drive toward it
  – Stop when a beacon is observed on the left
Project 3

Sensing relative wind direction:

- Weather vane mounted on an encoder
- Encoder essentially tells us how orientation is changing – so we must integrate this signal to estimate position
Using the Lego Encoder

“Two-wire” interface

• One wire connected to ground, the other to an I/O pin

• Must first charge the sensor’s power source (a capacitor):
  – Drive the pin high for ~3 ms

• Then read the sensor’s state
  – “float” the pin
  – Read out the voltage
Using the Lego Encoder

The sensed voltage will be one of four values:

• 2.6V, 1.7V, 3.9V, and 4.5V
• We will refer to these as integer values 0, 1, 2, and 3, respectively

• As the shaft is rotated, the sensed voltage will change from one level to the next (in the given sequence)
• Rotation in the opposite direction yields the opposite order
Using the Lego Encoder

• ... this give us a 2-bit counter (of sorts)
• But – for one complete rotation of the shaft, we will pass through this sequence a total of 4 times

How do we turn this information into absolute position?
Lego Encoder Caveats

• It is easy to miss the 4.5V level (due to sensor design)

• For the intermediate voltages (2.6V and 3.9V), a single analog sample cannot tell the difference between a stable voltage or a transient one
Project 3 Group Time

What is the FSM for the sensor processing of project 3?

• What are the states?
• For each state, what are the relevant events?
• What actions are taken for the events?
• Think carefully about how to handle the caveats

Once you have absolute position of the shaft, what is the code that turns this into steering commands? (give the pseudo-code)
Next Time

Project 3:
• Interrupt processing