Lecture 1
Introduction

Joe Steinmeyer, Joel Voldman, Stefanie Mueller

iesc-s2.mit.edu/608/spring18
Sign up for Piazza if you haven’t already

February 12, 2018
Administrative Things

• Sign up for Piazza (only 68% are currently signed up)
• All work and grades are done through the course site (there is no Stellar) iesc-s2.mit.edu/608/spring18
• Exercise 01 was due last night (see how late submissions are handled on site)
• Exercise 02 is out now and will be due this upcoming MONDAY 19\textsuperscript{th} @11:59pm because of holiday

One day later than normal
6.08 Class Architecture

• One lecture/wk: Mon 1-2
• Labs: Tue,Thu 150 min* 38-530
• Weekly homeworks, which will involve the kit and/or server
• 1 Midterm Exam (Tue 4/10/2018, 7pm EST)
• 1 Project!!
• Grading and other policies are on the class website

iesc-s2.mit.edu/608/spring18/syllabus
Labs (Tue/Thu)

- Two of them per week
- Work in partners
- Unfinished checkoffs can be made up in office hours only (we will not do checkoffs remotely. You cannot ask LAs to do checkoffs for you in dorms)
- Work *with* your partner
Two Types of Exercises

• **Regular Exercises:**
  - You should do them all
  - Often made of review work, practice work, and work to prepare for future labs and other exercises

• **Design Exercises:**
  - You need to do *at least four* of them throughout the semester to get full credit
  - Several released per week
  - Due at end of the week, by sharing link to working system (in video if necessary) and code
  - Manually graded in a very discretized manner (zero credit, half-credit, full credit)
Grading

• Everything in this class has fixed due dates, but partial credit can be attained for quite a while after the date (linear rolloff)

• Two one-week extensions will be applied at the end of the semester to your maximum benefit. You do not choose where they go, a script is your best friend chooses for you

• We have no slack-days as a result

• Any other extensions must receive written support from S^3
Final Project

• Last ~five/six weeks of the course (following midterm)
• Done in teams (size to be determined, prob ~4)
• Design and build a complex IOT-like system using the core components of 6.08 hardware
• We’ll go over more details as the project approaches
• 35% of final grade, and much of this comes from the work in the lead-up to the final project due-date
6.08 Final Project Examples

• Trivia game
• Smart-toaster integrated with social media
• Smart Bike (direction-finding)
• Laser Tag
• IOT Archery
• Theremin Hero
• Venmo-Integrated Laundry Coin System
• Med-Watch (smart device for med identification)
• Lit-Bit (festival friend/health monitor)
• ...quite a few more
Introductory EECS classes (6.01, 6.02, 6.03, 6.08)

• Show how a few different areas in EECS connect to each other
• Provide an introduction to engineering design
• Help you understand
  • Is EECS right for me?
  • What topics (and classes) in EECS might I want to pursue?
• Provide you with some relevant and applicable skills that you can go apply to projects and other work
6.08: interconnected embedded systems

- An ideal platform for EECS topics
- And for engineering design

Environment

- Sense
- Compute
- Communicate
- Store in db

Sensors
Signal processing
Microcontrollers
Communications
Algorithms
Energy management

Computer systems
Distributed computing
Databases
What is the Internet of Things?

- Embedded systems + communications + computation
The Hype Cycle and IOT

Hype

- Along with hype comes poor choices in the deployment of the technology...

Analogy to Cells

• Paramecia are single-celled organisms
• Cells with the ability to communicate and synchronize are capable of amazing things

• A lot of embedded systems are independent and isolated
• What could happen and be possible if we can start to connect them?
  • Improvements in efficiency
  • Unexpected behaviors/functionality

https://en.wikipedia.org/wiki/Paramecium
Embedded Systems are Everywhere

- Embedded system: “computer” within a device
- ~70-100 embedded systems in a modern car
- These are often microcontrollers
Microcontrollers

- A computer on a chip
  - Microprocessor: to perform computation
  - Memory: to store programs and data
  - Peripherals: to connect to the world

TMS1000
The first microcontroller
1974
4-bits, 400 kHz
1kB ROM, 256 bits RAM
$3 in volume ($15 in 2017 dollars)

http://www.datamath.org/Story/Intel.htm
http://smithsonianchips.si.edu/augarten/p38.htm

SR-16 calculator
Microcontrollers (used in 6.08)

- ESP32
- Made by Espressif (China)
- Released ~Sept 2016
- Dual Core, 280 MHz
- Built-in WiFi, Bluetooth
- Analog, digital pins

- We’ll be using it with the Arduino Core
What exactly is “Arduino”?  

• Originally a complete programming development environment with Arduino-brand compatible parts

• Other developers have built hooks/cores to integrate with the Arduino Integrated Development Environment, and that’s how we’ll be using it

• Is it perfect? No, but it’ll do for now

• As you advance in your careers, you may very well move beyond it
Our embedded system

- We will build this up during Feb/Mar
ESP32 Development Module

- Reset/Restart
- MicroUSB (Power and Data)
- Boot Mode
- 3.3V Regulator
- USB-to-Serial Adapter Chip
- ESP32 Microcontroller lives in there
- WiFi Antenna
Deeper Down the Rabbit Hole...

- ESP32 microcontroller
- 4MB Flash
- 40 MHz Crystal Oscillator (electric heartbeat)
Even Deeper...

If you crack the chip open:
Questions?

• For today:
• State Machine Review/Reflection
• Timing and `millis()`
• Upcoming Week
State Machines
State Machines

• Systems that respond to \textbf{inputs} based on their \textbf{state}

• Inputs can be user-based (buttons), from other systems (output of HTTP request), or even just time

• Moving from one state to another is called a \textbf{transition}
• The reason(s) for a transition is/are called \textbf{transition conditions}

• More formally called Finite State Machines (FSMs) when the number of states they can be in is finite...often we’ll just call them SMs
FSM Diagrams

- Diagram/Tables can sometimes help in laying out FSMs.

**State Transition Diagram**

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button Pushed</td>
<td>State 1</td>
</tr>
<tr>
<td>Button !Pushed</td>
<td>State 2</td>
</tr>
<tr>
<td></td>
<td>State 0</td>
</tr>
</tbody>
</table>

**Output Diagram**

<table>
<thead>
<tr>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0</td>
</tr>
<tr>
<td>State 1</td>
</tr>
<tr>
<td>State 2</td>
</tr>
<tr>
<td>State 3</td>
</tr>
</tbody>
</table>

Next states

Current outputs
Scrolling State Machine

System always draws String at (scroll_x, 15)

State Transition Diagram

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Far Right</td>
<td>Scroll Right</td>
</tr>
<tr>
<td>Too Far Left</td>
<td>Scroll Left</td>
</tr>
</tbody>
</table>

Output Diagram

<table>
<thead>
<tr>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scroll Right</td>
</tr>
<tr>
<td>Scroll Left</td>
</tr>
</tbody>
</table>

Output:

- scroll_x++
- scroll_x--
One Solution

void loop()
{
  scroller_update();
}

void scroller_update()
{
  while (scroll_x + message_len+1<127){
    scroll_x++;
    oled.clearBuffer();
    oled.setCursor(scroll_x,15);
    oled.print(message);
    oled.sendBuffer();
    delay(loop_speed);
  }
  while (scroll_x>0){
    scroll_x--;
    oled.clearBuffer();
    oled.setCursor(scroll_x,15);
    oled.print(message);
    oled.sendBuffer();
    delay(loop_speed);
  }
}

• Does actually work

*See code in lecture zip for details
Merging Code

• The **loop** function is the primary way in which events get carried out.

• As we add more functionality, one way to ensure reliable operation is to minimize excessively blocking code (while loops)

• The implementation on the previous page is blocking
System always draws String1 at (scroll1_x, 15) and String2 at (scroll2_x, 30)

Output Diagram 1

<table>
<thead>
<tr>
<th>Current State</th>
<th>Scroll Right</th>
<th>Scroll Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>scroll_x++</td>
<td>scroll_x--</td>
</tr>
</tbody>
</table>

State Transition Diagram (1 and 2)

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Far Right</td>
<td>Scroll Left</td>
</tr>
<tr>
<td>Too Far Left</td>
<td>Scroll Right</td>
</tr>
</tbody>
</table>
2 Scrollers Together v1:

*Blocking implementation causes problems*

```c
void loop()
{
  oled.clearBuffer();
  scroller1_update();
  oled.setCursor(scroll1_x,15);
  oled.print(message1);
  scroller2_update();
  oled.setCursor(scroll2_x,30);
  oled.print(message2);
  oled.sendBuffer();
  while(millis()-timer < loop_speed);
  timer = millis();
}
```

```c
void scroller1_update()
{
  while (scroll1_x + message1_len+1<127)
  {
    scroll1_x++;
    oled.clearBuffer();
    oled.setCursor(scroll1_x,15);
    oled.print(message1);
    oled.sendBuffer();
    delay(loop_speed);
  }
  while (scroll1_x<0)
  { 
    scroll1_x--;
    oled.clearBuffer();
    oled.setCursor(scroll1_x,15);
    oled.print(message1);
    oled.sendBuffer();
    delay(loop_speed);
  }
}
```

```c
void scroller2_update()
{
  while (scroll2_x + message2_len+1<127)
  {
    scroll2_x++;
    oled.clearBuffer();
    oled.setCursor(scroll2_x,30);
    oled.print(message2);
    oled.sendBuffer();
    delay(loop_speed);
  }
  while (scroll2_x<0)
  { 
    scroll2_x--;
    oled.clearBuffer();
    oled.setCursor(scroll2_x,30);
    oled.print(message2);
    oled.sendBuffer();
    delay(loop_speed);
  }
}
```

*See code in lecture zip for details*
2 Scrolling State Machine

System always draws String1 at (scroll1_x, 15) and String2 at (scroll2_x, 30)

State Transition Diagram (1 and 2)

Current State

<table>
<thead>
<tr>
<th>Input</th>
<th>Scroll Right</th>
<th>Scroll Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Far Right</td>
<td>Scroll Left</td>
<td></td>
</tr>
<tr>
<td>Too Far Left</td>
<td></td>
<td>Scroll Right</td>
</tr>
</tbody>
</table>

Output Diagram 1

<table>
<thead>
<tr>
<th>Current State</th>
<th>Scroll Right</th>
<th>Scroll Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>scroll_x++</td>
<td>scroll_x--</td>
</tr>
</tbody>
</table>

Output Diagram 2

<table>
<thead>
<tr>
<th>Current State</th>
<th>Scroll Right</th>
<th>Scroll Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>scroll_x+=2</td>
<td>scroll_x-=2</td>
</tr>
</tbody>
</table>
Better Implementation

- Non-blocking function calls to update state

*See code in lecture zip for details*
3 Scrolling State Machines (all independent)

- We can continue to expand this approach to get more and more “simultaneous” behavior

- Four code files are included on the site (next to lecture slides) building up to this so you can see how it was done

*See code in lecture zip for details*
From Lab 01B Last Week

void loop(){
    int input = digitalRead(input_pin);
    switch(state){
        case IDLE:
            num_count = 0;
            if(!input) state = DOWN;
            break;
        case DOWN:
            oled_print("pushed");
            if(input){
                state = UP;
                num_count+=1;
                counter = millis(); //start timeout
            }
            break;
        case UP:
            oled_print(String(num_count));
            if (millis()-counter>button_timeout){
                state = LOOKUP;
            } else if(!input){
                state = DOWN;
            }
            break;
        case LOOKUP:
            getNumberInfo();
            break;
    }
}

• Working SM from Lab01B last week
• `getNumberInfo()` performs full GET request by using `num_count`
• Talk with a neighbor/friend
• What will this code do?
State Machine Sketched Out

Start State

Button Pressed

IDLE

Button Pressed

DOWN

Button UnPressed

Button Pressed And No Timeout

UP

Button UnPressed And No Timeout

LOOKUP

Timeout

Only allow one GET request
Avoid Getting Stuck

```c
void loop(){
  int input = digitalRead(input_pin);
  switch(state){
  case IDLE:
    num_count = 0;
    if(!input) state = DOWN;
    break;
  case DOWN:
    oled_print("pushed");
    if(input){
      state = UP;
      num_count+=1;
      counter = millis(); //start timeout
    }
    break;
  case UP:
    oled_print(String(num_count));
    if (millis()-counter>button_timeout){
      state = LOOKUP;
    }else if(!input){
      state = DOWN;
    }
    break;
  case LOOKUP:
    getNumberInfo();
    state = IDLE;
    break;
  }
}
```

- Always make sure that all transition events are covered
- Fix the earlier code by transitioning from LOOKUP to IDLE automatically
Timing
Things take different times to run

- **digitalRead**: 15 microseconds
- **Serial.print()** (of an integer): 434 microseconds
- `+` (add) maybe just a clock cycle or so (~8 nanoseconds)

- We need to be aware of this and learn how to deal with it when we need predictable periodic behavior
millis()

• millis() links to an automatically-incrementing 4-byte unsigned long

Running this code...

```c
void setup() {
  Serial.begin(115200);  // startup
}

void loop() {
  Serial.println(millis());
  delay(1000);
}
```

...Results in this Serial Output

```
0
20
40
60
80
100
120
140
160
```

• The counter increments at 1 kHz (1000 Hz)
**millis()**

- An **unsigned long** on the ESP32 corresponds to a 4 byte value that represents only positive values (same as an **unsigned int**)
- A byte is made of 8 bits (8 binary values):
  - 0b00000000 to 0b11111111
  - Corresponding to 256 different values \((2^{\text{num bits}} = 2^8)\)
- Four bytes allows for \(2^{32}\) possible values which corresponds to 4,294,967,296 different values
- If the **millis()** timer starts at 0, for how long can it count?

*0b is notation for “binary”...0x is notation for hex....0d is notation for decimal!
millis()

- 4,294,967,296 milliseconds is...
- 4,294,967.296 seconds, which is...
- 1,193.0 hours, which is...
- 49.71 days

- What happens when this happens?

millis() “rolls over”:
  - Goes from 0b11111111 11111111 11111111 11111111
  - To: 0b00000000 00000000 00000000 00000000
micros()

- There’s also a micros() timer that works exactly the same, but which increments at 1 MHz.

- How long after start until micros() rolls over?

- If millis() rolls over every 49.71 days, micros() will roll over every 49.71 millidays (1000 times faster)

- So about ~1/20 of a day...every 71.5 minutes
delay(int amt)

- **delay** stalls any program until the specified amount of time has passed (in milliseconds)
- The problem with this is our system can’t do anything else during that delay

*Approximate definition of delay:*

```c
void delay(int amt){
    unsigned long timer = millis();
    while (millis() - timer < amt){
        //do nothing
    }
}
```
**delay() vs. using millis() directly**

- Use of delay also impedes accurate timing of events in our loop

```java
void loop()
{
    oled.clearBuffer();
    scroller_update();
    oled.sendBuffer();
    while (millis() - timer < loop_speed);
    timer = millis();
}
```

**More predictable**

Will execute every `loop_speed` milliseconds

```java
void loop()
{
    oled.clearBuffer();
    scroller_update();
    oled.sendBuffer();
    delay(loop_speed);
}
```

**Less Predictable**

Will execute every `loop_speed` milliseconds *plus* some unknown amount (however long it takes to run the first three lines)
Fixed Timing

• After carrying out tasks, we almost always want to wait “whatever amount is left” rather than a fixed amount.
• This becomes very useful in particular with functionality that:
  • Isn’t deterministic (parsing data of various lengths)
  • Isn’t under our direct control:
    • Anything using WiFi (A GET->response cycle varies)
    • Communicating with external components
    • Taking measurements (analog readings)

These request-responses are not coming in perfectly spaced
Lab this Week (Lab02A/B)

- Add an Inertial Measurement Unit
- Count Your Steps
- Integrate with class-wide step-tracker on a server

If you have not grabbed a fire-resistant lunchbox, please do so in Lab tomorrow

- Sign up for Piazza