Introduction

Demonstration

Summary
Introduction
So, you have a great idea, and you just finished assembling your hardware. What now?
Introduction

Goals

So, you have a great idea, and you just finished assembling your hardware. What now?

▶ Obviously, write firmware

But how to write it efficiently and well?
So, you have a great idea, and you just finished assembling your hardware. What now?

- Obviously, write firmware

But how to write it *efficiently* and *well*?

- *Efficiently* writing code: write and test in small pieces
  - If something breaks, narrow down culprits easily
  - Make sure fundamentals are sane before building on top

- Writing *good* code: Thursday!
EECS192 course project: line-following robot car

- Optical linescan camera, servo actuated steering
- Compute and control handled by microcontroller
- Lots of non-standard peripherals to bring up
What would be a logical way to bring up this system?
What would be a logical way to bring up this system?

- Write hardware drivers (encoder, camera, ...)
- Debug hardware drivers, in combination with hardware
- Write application logic (line detect, control loops, ...)

Platform Introduction

no demo like a live demo
Sanity Check

What’s the first thing we should bring up?

```
#include "mbed.h"

RawSerial serial(USBTX, USBRX);

DigitalOut Led1(PTC8);
DigitalOut Led2(PTD0);
...

int main() {
}

you start here
```
Sanity Check

What’s the first thing we should bring up?

Basic system test and debugging tools

```
#include "mbed.h"

RawSerial serial(USBTX, USBRX);

DigitalOut Led1(PTC8);
DigitalOut Led2(PTD0);
...

int main() {
}
```

you start here
Sanity Check

What’s the first thing we should bring up?

Basic system test and debugging tools

- Classic embedded "hello, world": blinking LEDs
- `printf` serial console

```c
#include "mbed.h"

RawSerial serial(USBTX, USBRX);

DigitalOut Led1(PTC8);
DigitalOut Led2(PTD0);
...

int main() {
}
```

you start here
Demonstration
Quad Encoder

A quad encoder uses two digital lines to count pulses with directionality (very common device)

How should we code this up?

Quad encoder waveforms

A
B

Forwards
Reverse
Quad Encoder

A quad encoder uses two digital lines to count pulses with directionality (very common device)

How should we code this up?

- Find a library online, don’t reinvent the wheel without a good reason
- Someone else has written it, and many more people have tested it

API

Quad encoder waveforms

Ducky (UCB EECS)
Make the clicky knob work!
Now bring up the servo!

you’ve already learned the servo PWM protocol
Many peripherals use standard interfaces, like I2C and SPI

Some don’t, like this line camera
  ▶ Generally simple protocols
  ▶ Datasheet gives all the details
  ▶ Control integration time
  ▶ Pixel intensity on analog line
  ▶ Clock signal shifts out next pixel

TSL1401 Datasheet, page 5
Let’s write some code!
A thermistor is connected to an external ADC, connected by I2C

So let’s bring up the ADC

- Datasheet describes protocol in terms of I2C transactions
- Basically: write configuration registers, read conversion result

For example, to write to the configuration register to set the ADS1013/4/5 to continuous conversion mode and then read the conversion result, send the following bytes in this order:

Write to Config register:
- First byte: 0b10010000 (first 7-bit I2C address followed by a low read/write bit)
- Second byte: 0b00000001 (points to Config register)
- Third byte: 0b00001000 (MSB of the Config register to be written)
- Fourth byte: 0b10000011 (LSB of the Config register to be written)

Write to Pointer register:
- First byte: 0b10010000 (first 7-bit I2C address followed by a low read/write bit)
- Second byte: 0b00000000 (points to Conversion register)

Read Conversion register:
- First byte: 0b10010001 (first 7-bit I2C address followed by a high read/write bit)
- Second byte: the ADS1013/4/5 response with the MSB of the Conversion register
- Third byte: the ADS1013/4/5 response with the LSB of the Conversion register
I2C Fun Time!
Thermistors

MCP9701 is a linear active thermistor

- Voltage is proportional to temperature
- Details (offset and scale constants) in datasheet
- Around 20°C, what is the expected voltage?

<table>
<thead>
<tr>
<th>Sensor Output</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Voltage, $T_A = 0°C$</td>
<td>$V_{OC}$</td>
<td>500</td>
<td>mV</td>
</tr>
<tr>
<td>Output Voltage, $T_A = 0°C$</td>
<td>$V_{OC}$</td>
<td>400</td>
<td>mV</td>
</tr>
<tr>
<td>Temperature Coefficient</td>
<td>$T_C$</td>
<td>10.0</td>
<td>mV/°C</td>
</tr>
<tr>
<td></td>
<td>$T_C$</td>
<td>19.5</td>
<td>mV/°C</td>
</tr>
<tr>
<td>Output Non-linearity</td>
<td>$V_{OHE}$</td>
<td>±0.5</td>
<td>°C</td>
</tr>
<tr>
<td>Output Current</td>
<td>$I_{OUT}$</td>
<td>100</td>
<td>µA</td>
</tr>
<tr>
<td>Output Impedance</td>
<td>$Z_{OUT}$</td>
<td>20</td>
<td>Ω</td>
</tr>
<tr>
<td>Output Load Regulation</td>
<td>$\Delta V_{OUT}$</td>
<td>1</td>
<td>Ω</td>
</tr>
<tr>
<td>Turn-on Time</td>
<td>$t_{ON}$</td>
<td>800</td>
<td>µs</td>
</tr>
</tbody>
</table>

Note 1: The MCP9700/9700A family accuracy is tested with $V_{DD} = 3.3V$, while the MCP9701/9701A accuracy is tested with $V_{DD} = 5.0V$.

Note 2: The MCP9700/9700A and MCP9701/9701A family is characterized using the first-order or linear equation, as shown in Equation 4.2. Also refer to Figure 2-16.

Note 3: SC70-5 package thermal response with 1x1 inch, dual-sided copper clad, TO-92-3 package thermal response without PCB (leaded).
Thermistors

MCP9701 is a linear active thermistor

- Voltage is proportional to temperature
- Details (offset and scale constants) in datasheet
- Around 20°C, what is the expected voltage?
  - 0.8 V

### Sensor Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Voltage, $T_A = 0°C$</td>
<td>$V_{OC}$ -- 500 mV MCP9700/9700A</td>
</tr>
<tr>
<td>Output Voltage, $T_A = 0°C$</td>
<td>$V_{OC}$ -- 400 mV MCP9701/9701A</td>
</tr>
<tr>
<td>Temperature Coefficient</td>
<td>$T_C$ -- 10.0 mV/°C MCP9700/9700A</td>
</tr>
<tr>
<td></td>
<td>$T_C$ -- 19.5 mV/°C MCP9701/9701A</td>
</tr>
<tr>
<td>Output Non-linearity</td>
<td>$V_{OUL}$ ±0.5°C $T_A = 0°C$ to $+70°C$ (Note 2)</td>
</tr>
<tr>
<td>Output Current</td>
<td>$I_{OUT}$ -- 100 µA</td>
</tr>
<tr>
<td>Output Impedance</td>
<td>$Z_{OUT}$ -- 20 Ω $I_{OUT} = 100$ µA, $f = 500$ Hz</td>
</tr>
<tr>
<td>Output Load Regulation</td>
<td>$\Delta V_{OUT}$ -- 1 Ω $T_A = 0°C$ to $+70°C$, $I_{OUT} = 100$ µA</td>
</tr>
<tr>
<td>Turn-on Time</td>
<td>$I_{ON}$ -- 800 µs</td>
</tr>
</tbody>
</table>

**Note 1:** The MCP9700/9700A family accuracy is tested with $V_{DD} = 3.3$V, while the MCP9701/9701A accuracy is tested with $V_{DD} = 5.0$V.

**Note 2:** The MCP9700/9700A and MCP9701/9701A family is characterized using the first-order or linear equation, as shown in Equation 4-2. Also refer to Figure 2-16.

**Note 3:** SC70-5 package thermal response with 1x1 inch, dual-sided copper clad, TO-92-3 package thermal response without PCB (lead).
MCP9701 is a linear active thermistor

- Voltage is proportional to temperature
- Details (offset and scale constants) in datasheet
- Around 20°C, what is the expected voltage? 
  - 0.8 V
- How would you code up the conversion from voltage to temperature?

MCP9701 Datasheet, pages 3 and 7
Closed-loop tracking fun demo

Bringing it all together
**Summary**

- Whirlwind tour of embedded system bring-up with custom peripherals
  - A lot of artificial failures as examples of common failure modes
  - ... and a lot of details glossed over
- Write code incrementally
  - Less things added, less things to question if something goes wrong
  - Verify each part before building on top of it
- When stuff goes wrong, get visibility into the system
  - `printf` for software visibility
  - Multimeter for non-time-varying quantities, like power rails
  - Oscilloscope for analog signals
  - Logic analyzer for digital signals, with protocol analyzers
- Design for Test
  - Make system visibility *easy*, like with probe points