CS194-126, Software Defined PCBs

Lecture 14: Embedded Debugging: System Bring-up

Richard "Ducky" Lin

11 October 2016

Introduction

Goals

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

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*?

Goals

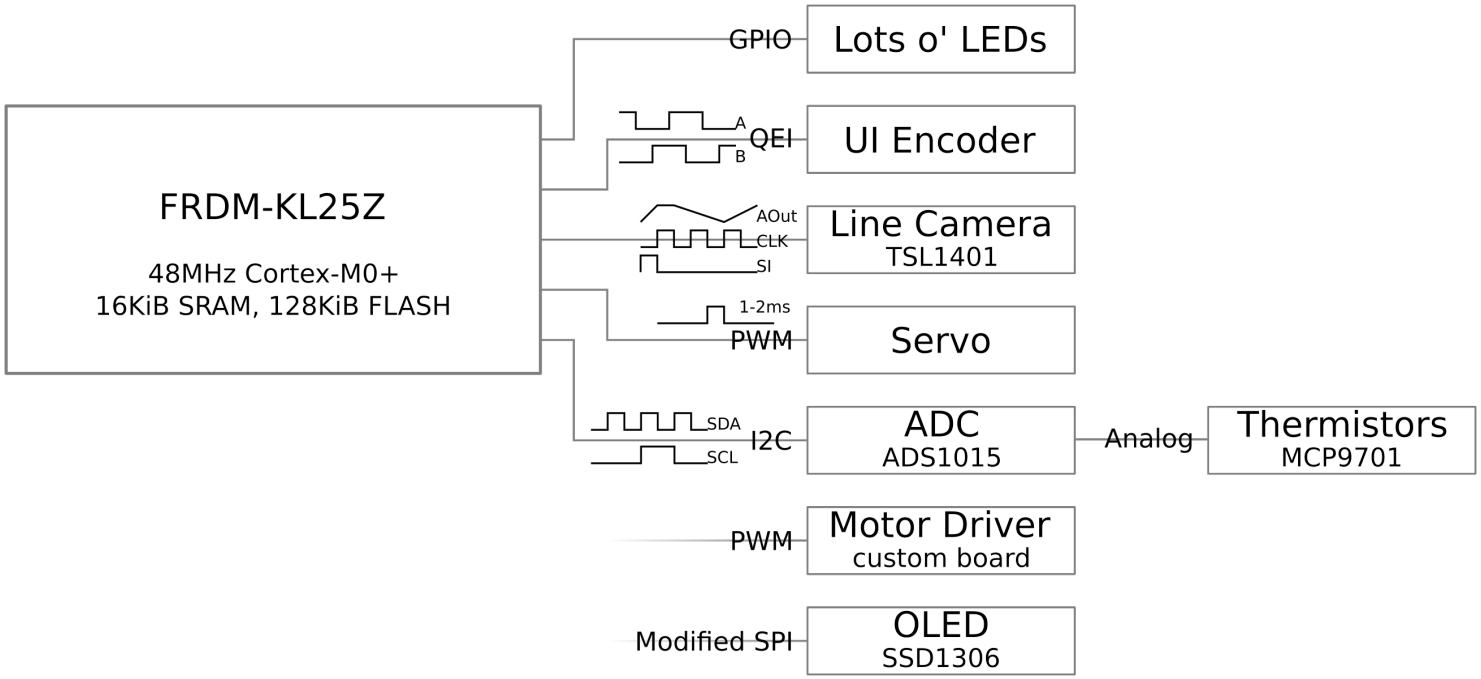
So, you have a great idea, and you just finished assembling your hardware. What now?

- Obviously, write firmware

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

- *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!

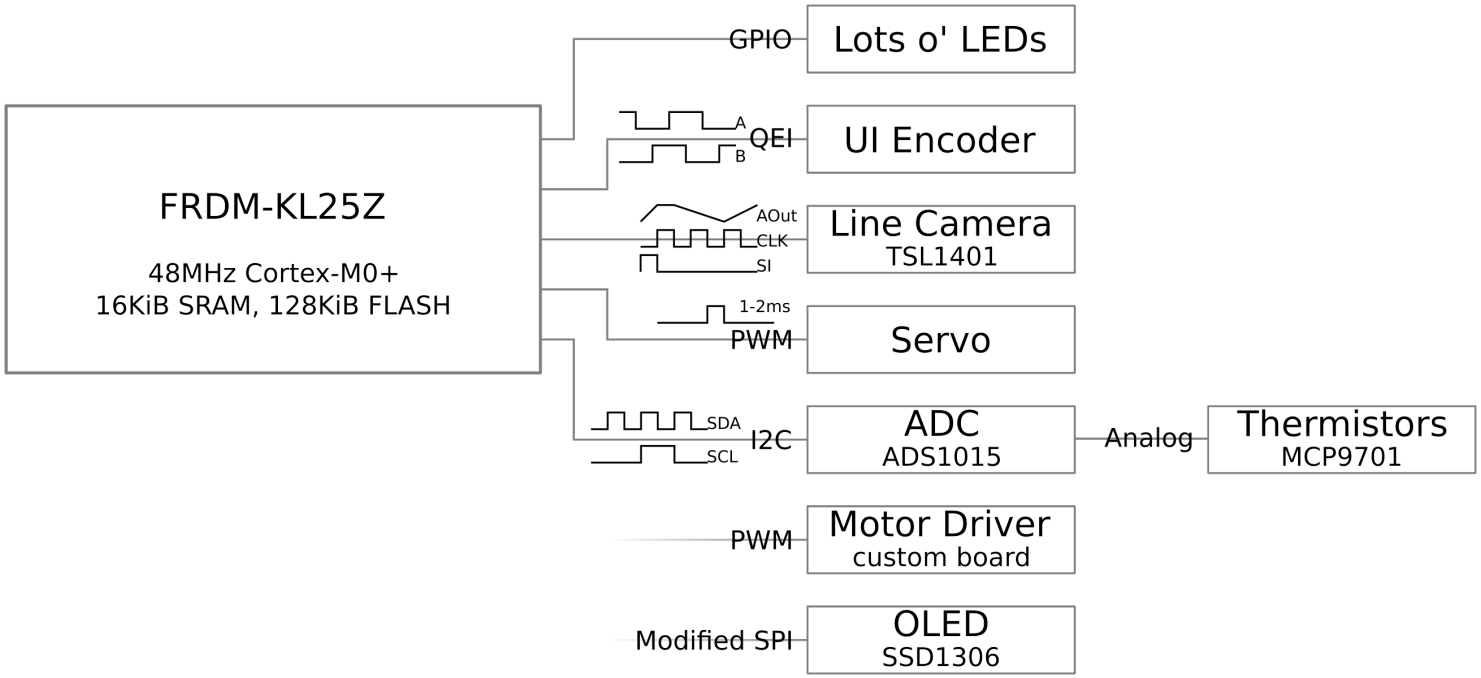
Platform Introduction no demo like a live demo



EECS192 course project: line-following robot car

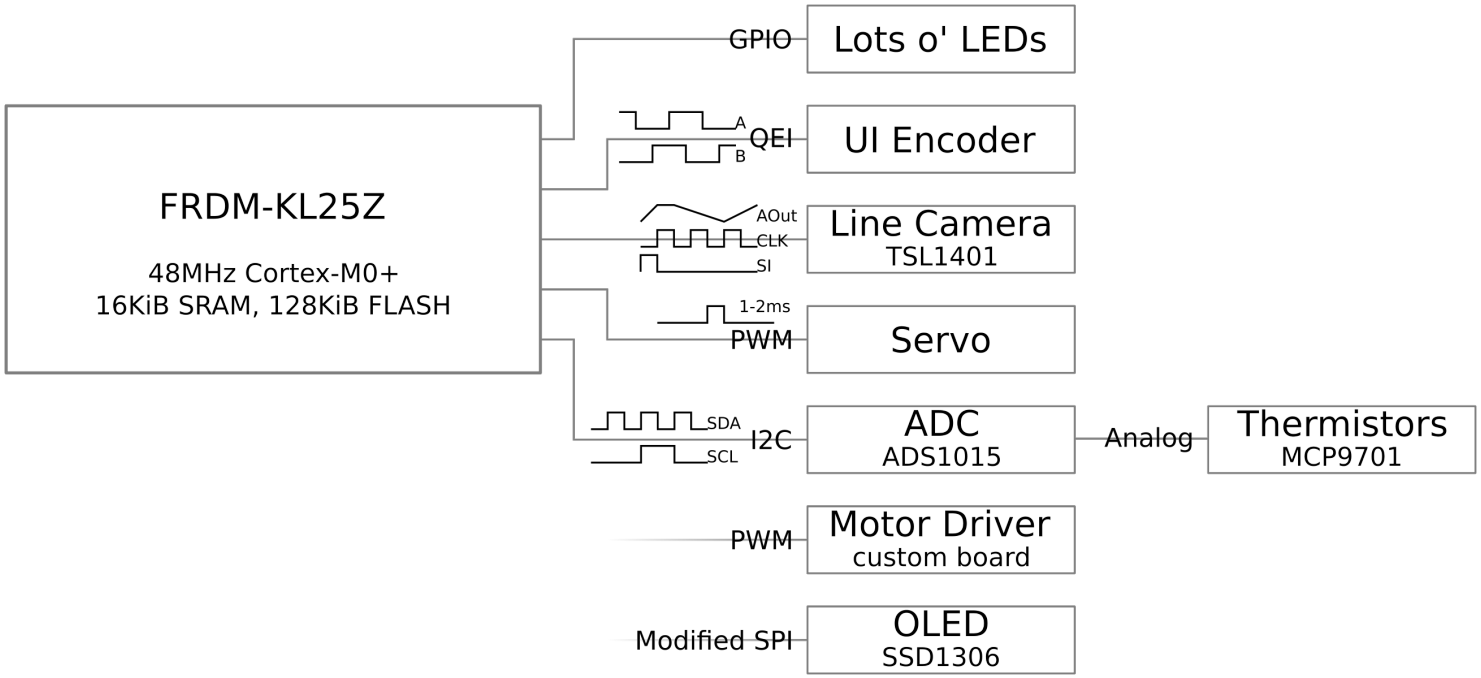
- Optical linescan camera, servo actuated steering
- Compute and control handled by microcontroller
- Lots of non-standard peripherals to bring up

Platform Introduction no demo like a live demo



What would be a logical way to bring up this system?

Platform Introduction no demo like a live demo



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, ...)

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

```
#include "mbed.h"

RawSerial serial(USBTX
     , USBRX);

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

int main() {

}
```

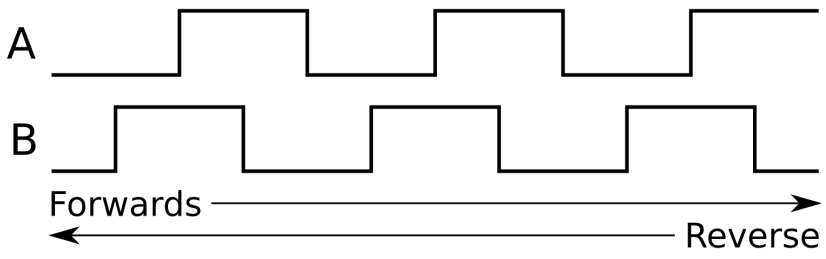
you start here

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

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

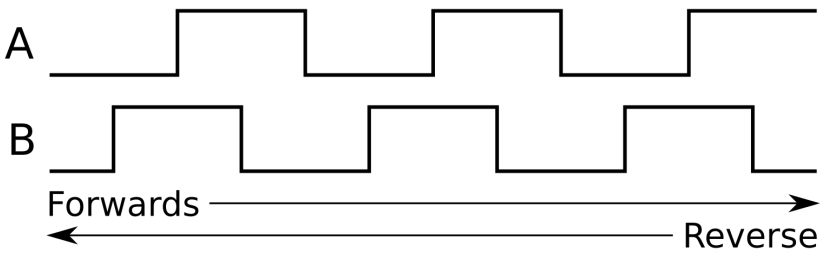
QEI - QEI Class Reference

Import library

Public Member Functions

	QEI (PinName channelA, PinName channelB, PinName index, int pulsesPerRev, Encoding encoding=X2_ENCODING) Constructor.
void	**reset** (void) Reset the encoder.
int	**getCurrentState** (void) Read the state of the encoder.
int	**getPulses** (void) Read the number of pulses recorded by the encoder.
int	**getRevolutions** (void) Read the number of revolutions recorded by the encoder on the index channel.

mbed QEI API



Quad encoder waveforms

Make the clicky knob work!

Now bring up the servo!

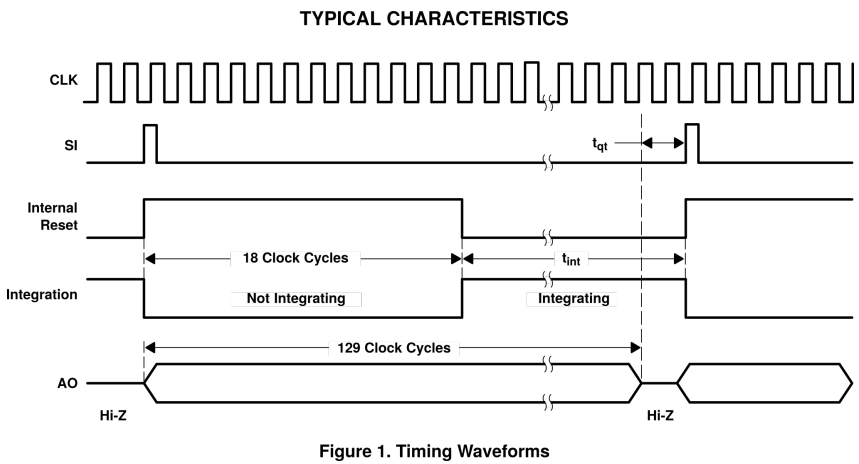
you've already learned the servo PWM protocol

Line Camera

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!

External ADC

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

ADS1013
ADS1014
ADS1015

TEXAS INSTRUMENTS

SBAS473C – MAY 2009 – REVISED OCTOBER 2009 www.ti.com

QUICKSTART GUIDE

This section provides a brief example of ADS1013/4/5 communications. Refer to subsequent sections of this data sheet for more detailed explanations. Hardware for this design includes: one ADS1013/4/5 configured with an I²C address of 1001000; a microcontroller with an I²C interface (TI recommends the MSP430 product line); discrete components such as resistors, capacitors, and serial connectors; and a 2V to 5V power supply. Figure 9 shows the basic hardware configuration.

The ADS1013/4/5 communicate with the master (microcontroller) through an I²C interface. The master provides a clock signal on the SCL pin and data are transferred via the SDA pin. The ADS1013/4/5 never drive the SCL pin. For information on programming and debugging the microcontroller being used, refer to the device-specific product data sheet.

The first byte sent by the master should be the ADS1013/4/5 address followed by a bit that instructs the ADS1013/4/5 to listen for a subsequent byte. The second byte is the register pointer. Refer to Table 6 for a register map. The third and fourth bytes sent from the master are written to the register indicated in the second byte. Refer to Figure 16 and Figure 17 for read and write operation timing diagrams, respectively. All read and write transactions with the ADS1013/4/5 must be preceded by a start condition and followed by a stop condition.

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: 0b00000100 (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

ADS1015 Datasheet, page 8

I2C Fun Time!

Thermistors

MCP9701 is a linear active thermistor

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

Sensor Output						
Output Voltage, T_A = 0°C	$V_{0°C}$	—	500	—	mV	**MCP9700/9700A**
Output Voltage, T_A = 0°C	$V_{0°C}$	—	400	—	mV	**MCP9701/9701A**
Temperature Coefficient	T_C	—	10.0	—	mV/°C	**MCP9700/9700A**
	T_C	—	19.5	—	mV/°C	**MCP9701/9701A**
Output Non-linearity	V_{ONL}	—	±0.5	—	°C	T_A = 0°C to +70°C **(Note 2)**
Output Current	I_{OUT}	—	—	100	µA	
Output Impedance	Z_{OUT}	—	20	—	Ω	I_{OUT} = 100 µA, f = 500 Hz
Output Load Regulation	$\Delta V_{OUT}/\Delta I_{OUT}$	—	1	—	Ω	T_A = 0°C to +70°C, I_{OUT} = 100 µA
Turn-on Time	t_{ON}	—	800	—	µs	

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.

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.

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

MCP9700/9700A and MCP9701/9701A

Note: Unless otherwise indicated, **MCP9700/9700A**: V_{DD} = 2.3V to 5.5V; **MCP9701/9701A**: V_{DD} = 3.1V to 5.5V; GND = Ground, C_{bypass} = 0.1 µF.

FIGURE 2-13: Output Voltage vs. Power Supply.

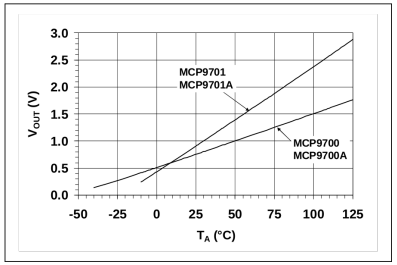


FIGURE 2-16: Output Voltage vs. Ambient Temperature.

MCP9701 Datasheet, pages 3 and 7

Thermistors

MCP9701 is a linear active thermistor

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

Sensor Output						
Output Voltage, T_A = 0°C	$V_{0°C}$	—	500	—	mV	**MCP9700/9700A**
Output Voltage, T_A = 0°C	$V_{0°C}$	—	400	—	mV	**MCP9701/9701A**
Temperature Coefficient	T_C	—	10.0	—	mV/°C	**MCP9700/9700A**
	T_C	—	19.5	—	mV/°C	**MCP9701/9701A**
Output Non-linearity	V_{ONL}	—	±0.5	—	°C	T_A = 0°C to +70°C **(Note 2)**
Output Current	I_{OUT}	—	—	100	µA	
Output Impedance	Z_{OUT}	—	20	—	Ω	I_{OUT} = 100 µA, f = 500 Hz
Output Load Regulation	$\Delta V_{OUT}/\Delta I_{OUT}$	—	1	—	Ω	T_A = 0°C to +70°C, I_{OUT} = 100 µA
Turn-on Time	t_{ON}	—	800	—	µs	

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.

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.

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

MCP9700/9700A and MCP9701/9701A

Note: Unless otherwise indicated, **MCP9700/9700A**: V_{DD} = 2.3V to 5.5V; **MCP9701/9701A**: V_{DD} = 3.1V to 5.5V; GND = Ground, C_{bypass} = 0.1 µF.

FIGURE 2-13: Output Voltage vs. Power Supply.

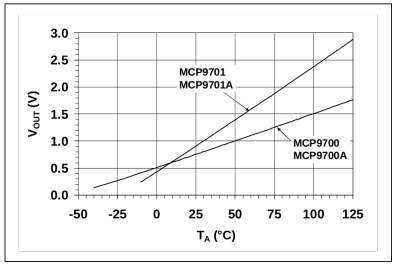


FIGURE 2-16: Output Voltage vs. Ambient Temperature.

MCP9701 Datasheet, pages 3 and 7

Thermistors

MCP9701 is a linear active thermistor

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

Sensor Output						
Output Voltage, T_A = 0°C	$V_{0°C}$	—	500	—	mV	**MCP9700/9700A**
Output Voltage, T_A = 0°C	$V_{0°C}$	—	400	—	mV	**MCP9701/9701A**
Temperature Coefficient	T_C	—	10.0	—	mV/°C	**MCP9700/9700A**
	T_C	—	19.5	—	mV/°C	**MCP9701/9701A**
Output Non-linearity	V_{ONL}	—	±0.5	—	°C	T_A = 0°C to +70°C **(Note 2)**
Output Current	I_{OUT}	—	—	100	µA	
Output Impedance	Z_{OUT}	—	20	—	Ω	I_{OUT} = 100 µA, f = 500 Hz
Output Load Regulation	$\Delta V_{OUT}/\Delta I_{OUT}$	—	1	—	Ω	T_A = 0°C to +70°C, I_{OUT} = 100 µA
Turn-on Time	t_{ON}	—	800	—	µs	

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.

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.

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

MCP9700/9700A and MCP9701/9701A

Note: Unless otherwise indicated, **MCP9700/9700A**: V_{DD} = 2.3V to 5.5V; **MCP9701/9701A**: V_{DD} = 3.1V to 5.5V; GND = Ground, C_{bypass} = 0.1 µF.

FIGURE 2-13: Output Voltage vs. Power Supply.

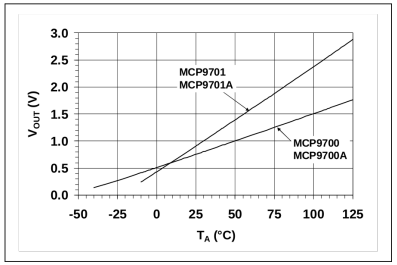


FIGURE 2-16: Output Voltage vs. Ambient Temperature.

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