University of Colorado - Boulder

ASEN 5067: Microavionics for Aerospace

September - 4 - 2023

Lab 1: Programming and Testing the Microavionics Development Board

Author:
Thomas Dunnington

Professor:
Trudy Schwartz

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

I. Questions

1) Yes, everything worked as it should.
2) There are three pins next to the LM35 sensor. The middle pin is connected to the Vout of the LM35 and the top pin is connected to the RA3 pin. The bottom pin is connected to the RF6 pin. In the current configuration, the middle pin is connected to the RA3 pin via a jumper. This means the Vout of the LM35 is connected to the RA3 pin. Both the RA3 and RF6 pins are connected to the data bus which the MPU can read data from. So in this case, the RA3 pin will contain the voltage data of the temperature sensor.
3) Pin RB6 is connected to a part of the ethernet connection on the EXT ICD connection. This pin is also what the buzzer is connected to which is evident from the voltage readings on pin RB6 when the hex program is uploaded. So the noises that occur while the program is being uploaded is a result of the change in voltage on RB6 which is connected to the buzzer and the ethernet connection.
4) The SW13 switches are used to determine which logic level will be applied to port pins when the buttons are pressed. When the port is set to the VCC position, the button presses will apply a logic one to the corresponding pin and when set to GND it will be a logic zero. For the tri-state pull-up and pull-down switches, the middle position disables both the pull-up and pull-down features. The up position connects the resistor in a pull-up state and the down position connects the resistor in the pull-down state. With the PG switch in the button press level at the middle state: The LED for RG7 is off for the pull and down position and is on for the up position. Pressing the button does not affect any of these positions. This is because the button press level is in the middle so when a button is pressed, nothing is changed and the LED does not turn on or off, it just remains as it was before the button was pressed. This is because the button press circuit is not complete with the setting in the middle and thus pressing the button has no effect. When the PG switch is on the Vcc setting: The LED is off for the pull and down positions and is on for the up position. However, in the pull and down positions, pressing the button turns the LED on while the button is pressed. Vcc means voltage common collector and is the higher voltage with respect to ground. In this case, when the SW13 switch is set to Vcc, it allows a button press to complete a circuit with the higher voltage as the PORTG LEVEL is set to the higher Vcc voltage. This is why by pressing the button in the pull and down positions, the LED turns on. With the PG switch in the GND position, the LED is off for the pull and down positions and is on for the up position. Pressing the button when the LED is in the up position turns the LED off. This is because the button switch level is connected to GND and thus by pressing the button the circuit for the LED is grounded and the LED turns off. The resistors on SW13 are used as voltage dividers which step down the voltage. They are pull-up and pull-down resistors which ensure the correct voltage supplied to the PORT. The resistors after the LEDs are used to limit the current through the LEDs by creating a voltage divider and reducing the voltage.
5) Computers execute machine code which is unreadable by humans. As such, there needs to be a level of abstraction between the executed machine code and the instructions humans want the computer to execute. This is where programming languages come into play. A low-level programming language such as assembly allows for more control over memory management and specific instructions. However, this also requires significantly more code and time to write the code. A higher-level programming language such as C allows for much more concise code and is easier for humans to read and understand. However, you also must rely on the compiler to compile your code and generate assembly code. This means you do not have as much control over the specific instructions that are executed. Higher-level languages such as Python and MATLAB have even less control over memory management which can cause longer run times when you do not have as much control over the specific instructions.
6) The address bus is used to identify memory registers and I/O devices. It is a unidirectional bus with information flowing from the processor to memory and I/O devices. Each register and I/O device is specified with an address. The number of possible registers depends on the size of the memory addresses. The data bus is used to transfer binary data or instructions. When the processor reads an instruction, data flows from the memory to the processor and when the processor writes an instruction, data flows from the processor to the memory. This bidirectional data transfer also occurs with I/O devices such as a sensor where the processor may need to send a command and read the output of a sensor all along the data bus. The control lines are comprised of individual lines called Read and Write. The Read and Write lines are used for timing signals to enable memory and I/O devices. They are essential in timing which direction data is flowing in the data bus.
7) The address bus is unidirectional because it is only necessary that information flows from the processor to memory and I/Os. The data bus is bidirectional because that is where the actual data between the processor and the memory and I/Os is transferred. The address bus is only used to identify the location of the memory and I/Os

with its address and is thus only unidirectional. When the processor reads an instruction, data flows from the memory to the processor and when the processor writes an instruction, data flows from the processor to the memory. Thus, it is required that the data bus is bidirectional.

8) A KB of memory is 1024 bytes. A single byte of memory is comprised of 8 bits. So 1 KB of memory would have $8 \cdot 1024 = 8192$ bits.
9) Using the convention that 1 MB of data is equal to $1024 \cdot 1024 = 1048567$ bytes, we have a total of 4194304 bytes. Assuming each register has 8 bits of data, we have a single byte per register meaning we have 4194304 registers. Since the first address is 00000, then the last address will be $4194304 - 1$ which is 4194303. This in hex is 0x3FFFFF.
10) The number 0x1FFFF in decimal is 131071. Since we start at 0x00000 for the register address, we have 0x1FFFF + 0x1 registers. This is 131072 in decimal. Assuming there are 8 bits per register, there would be 131072 bytes of data. We can then divide by 1024 to solve for the number of KB which gives us $131072/1024 = 128$. Thus, there are 128 KB of memory.
11) A program counter is used to keep track of where a program is in its execution. It holds the memory address of the next instruction to be executed and is incremented after each instruction is executed. It can be thought of as a pointer to the next instruction. The size is 21 bits in the PIC18 MCU.
12)

- 42: 0010 1010, 0x2A

$$\begin{aligned} 42/2 &= 21 \text{ R:}0 && 0 \\ 21/2 &= 10 \text{ R:}1 && 10 \\ 10/2 &= 5 \text{ R:}0 && 010 \\ 5/2 &= 2 \text{ R:}1 && 1010 \\ 2/2 &= 1 \text{ R:}0 && 01010 \\ 1/2 &= 0 \text{ R:}1 && 101010 \\ & && 00101010 \end{aligned}$$

  We can then convert to hex by using the binary representation by converting each nibble in the byte to a hexadecimal digit.

$$\begin{aligned} 0010 &= 2_{10} = 2_H \\ 1010 &= 10_{10} = A_H \\ 00101010 &= 0x2A \end{aligned}$$

- 76: 0100 1100, 0x4C

$$\begin{aligned} 76/2 &= 38 \text{ R:}0 && 0 \\ 38/2 &= 19 \text{ R:}0 && 00 \\ 19/2 &= 9 \text{ R:}1 && 100 \\ 9/2 &= 4 \text{ R:}1 && 1100 \\ 4/2 &= 2 \text{ R:}0 && 01100 \\ 2/2 &= 1 \text{ R:}0 && 001100 \\ 1/2 &= 0 \text{ R:}1 && 1001100 \\ & && 01001100 \end{aligned}$$

$$\begin{aligned} 0100 &= 4_{10} = 4_H \\ 1100 &= 12_{10} = C_H \\ 00101010 &= 0x4C \end{aligned}$$

- 130: 1000 0010, 0x82

$$\begin{aligned} 130/2 &= 65 \text{ R:0} & 0 \\ 65/2 &= 32 \text{ R:1} & 10 \\ 32/2 &= 16 \text{ R:0} & 010 \\ 16/2 &= 8 \text{ R:0} & 0010 \\ 8/2 &= 4 \text{ R:0} & 00010 \\ 4/2 &= 2 \text{ R:0} & 000010 \\ 2/2 &= 1 \text{ R:0} & 0000010 \\ 1/2 &= 0 \text{ R:1} & 10000010 \end{aligned}$$

$$1000 = 8_{10} = 8_H$$
$$0010 = 2_{10} = 2_H$$
$$10000010 = 0x82$$

- 142: 1000 1110, 0x8E

$$\begin{aligned} 142/2 &= 71 \text{ R:0} & 0 \\ 71/2 &= 35 \text{ R:1} & 10 \\ 35/2 &= 17 \text{ R:1} & 110 \\ 17/2 &= 8 \text{ R:1} & 1110 \\ 8/2 &= 4 \text{ R:0} & 01110 \\ 4/2 &= 2 \text{ R:0} & 001110 \\ 2/2 &= 1 \text{ R:0} & 0001110 \\ 1/2 &= 0 \text{ R:1} & 10001110 \end{aligned}$$

$$1000 = 8_{10} = 8_H$$
$$1110 = 14_{10} = E_H$$
$$10001110 = 0x8E$$

- 245: 1111 0101, 0xF5

$$\begin{aligned} 245/2 &= 122 \text{ R:1} & 1 \\ 122/2 &= 61 \text{ R:0} & 01 \\ 61/2 &= 30 \text{ R:1} & 101 \\ 30/2 &= 15 \text{ R:0} & 0101 \\ 15/2 &= 7 \text{ R:1} & 10101 \\ 7/2 &= 3 \text{ R:1} & 110101 \\ 3/2 &= 1 \text{ R:1} & 1110101 \\ 1/2 &= 0 \text{ R:1} & 11110101 \end{aligned}$$

$$1111 = 15_{10} = F_H$$
$$0101 = 5_{10} = 5_H$$
$$11110101 = 0xF5$$

- -42: 1101 0110, 0xD6
  From above we know the unsigned representation of 42 in binary is 0010 1010. We will use an 8-bit signed notation. Now flip the bits and add 1 to convert to an 8-bit signed two's complement representation:

$$!(00101010) = 11010101$$
$$11010101 + 1 = 11010110$$

$$1101 = 13_{10} = D_H$$
$$0110 = 6_{10} = 6_H$$
$$11010110 = 0xD6$$

- -76: 1011 0100, 0xB4
  From above we know the unsigned representation of 76 in binary is 0100 1100. Now flip the bits and add 1 to convert to an 8-bit signed two's complement representation:

$$!(01001100) = 10110011$$
$$10110011 + 1 = 10110100$$

$$1011 = 11_{10} = B_H$$
$$0100 = 4_{10} = 4_H$$
$$10110100 = 0xB4$$

- -255: 1111 1111 0000 0001, 0xFF01

$$\begin{array}{rl}
255/2 = 127 \text{ R:}1 & 1 \\
127/2 = 63 \text{ R:}1 & 11 \\
63/2 = 31 \text{ R:}1 & 111 \\
31/2 = 15 \text{ R:}1 & 1111 \\
15/2 = 7 \text{ R:}1 & 11111 \\
7/2 = 3 \text{ R:}1 & 111111 \\
3/2 = 1 \text{ R:}1 & 1111111 \\
1/2 = 0 \text{ R:}1 & 11111111
\end{array}$$

  From above we know the unsigned representation of 255 in binary is 1111 1111. The most negative number we can represent with two's complement 8-bit representation is -128. Thus, we must use a 16-bit signed representation. This gives us a positive representation of 255 as 0000 0000 1111 1111. Now flip the bits and add 1 to convert to a 16-bit signed two's complement representation:

$$!(0000000011111111) = 1111111100000000$$
$$1111111100000000 + 1 = 1111111100000001$$

$$\begin{aligned}1111 &= 15_{10} = F_H\\1111 &= 15_{10} = F_H\\0000 &= 0_{10} = 0_H\\0001 &= 1_{10} = 1_H\\1111111100000001 &= 0xFF01\end{aligned}$$

13) • 0x30: 0011 0000, 48

We start by converting each digit to a nibble in binary.

$$\begin{aligned}0x3 &= 0011\\0x0 &= 0000\\0x30 &= 00110000\end{aligned}$$

We now convert the binary representation to decimal using powers of 2.

$$\begin{aligned}00110000 &= 1 \cdot 2^5 + 1 \cdot 2^4\\&= 48\end{aligned}$$

• 0x22: 0010 0010, 34

$$\begin{aligned}0x2 &= 0010\\0x2 &= 0010\\0x22 &= 00100010\end{aligned}$$

$$\begin{aligned}00100010 &= 1 \cdot 2^5 + 1 \cdot 2^1\\&= 34\end{aligned}$$

• 0x72: 0111 0010, 114

$$\begin{aligned}0x7 &= 0111\\0x2 &= 0010\\0x72 &= 01110010\end{aligned}$$

$$\begin{aligned}01110010 &= 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^1\\&= 114\end{aligned}$$

• 0x4D: 0100 1101, 77

$$\begin{aligned}0x4 &= 0100\\0xD &= 1101\\0x4D &= 01001101\end{aligned}$$

$$\begin{aligned}01001101 &= 1 \cdot 2^6 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0\\&= 77\end{aligned}$$

- 0xAB: 1010 1011, Unsigned: 171, Signed: -85

$$\begin{aligned} 0xA &= 1010 \\ 0xB &= 1011 \\ 0xAB &= 10101011 \end{aligned}$$

$$\begin{aligned} 10101011 &= 1 \cdot 2^7 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 171 \end{aligned}$$

For the signed representation we have:

$$\begin{aligned} 10101011 &= -1 \cdot 2^7 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= -85 \end{aligned}$$

II. Additional Questions

14) The on-chip voltage regulator enable pin is called the ENVREG pin. All of the PIC18F87K22 devices are usually powered by 3.3V, however, there are two on-chip regulators that allow the device to run its logic from VDD with 5 or 3.3V. If the ENVREG pin is tied to the VDD pin, then the regulator is in enable mode and can be powered by 5V. In this configuration, power is provided to the core by the other VDD pins. The regulator is in disable mode if the ENVREG pin is connected to VSS. This causes the power to be supplied directly by VDD and the voltage cannot exceed the specified VDDCORE levels. The following figure shows these two configurations.

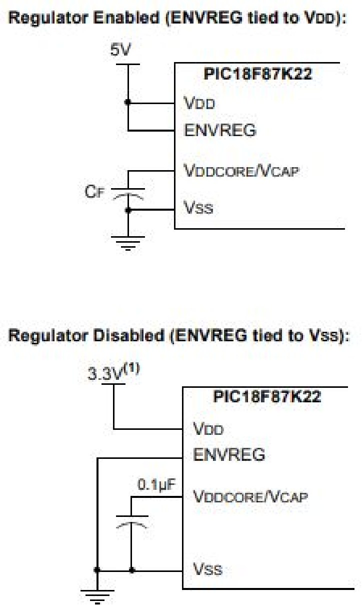
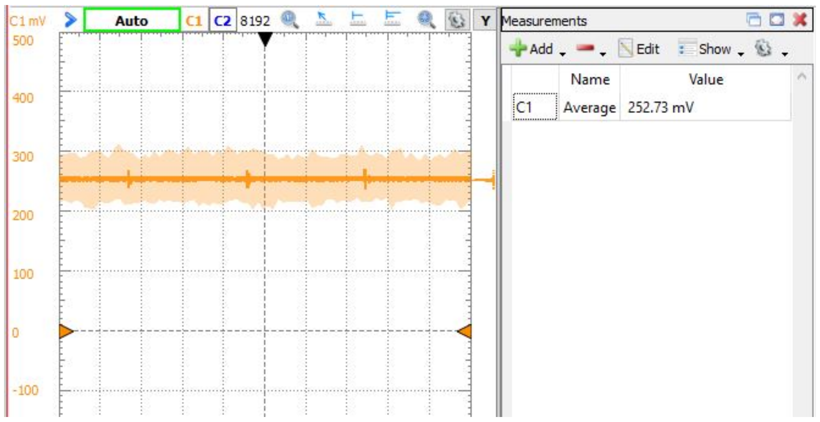


Fig. 1 Regulator Schematic

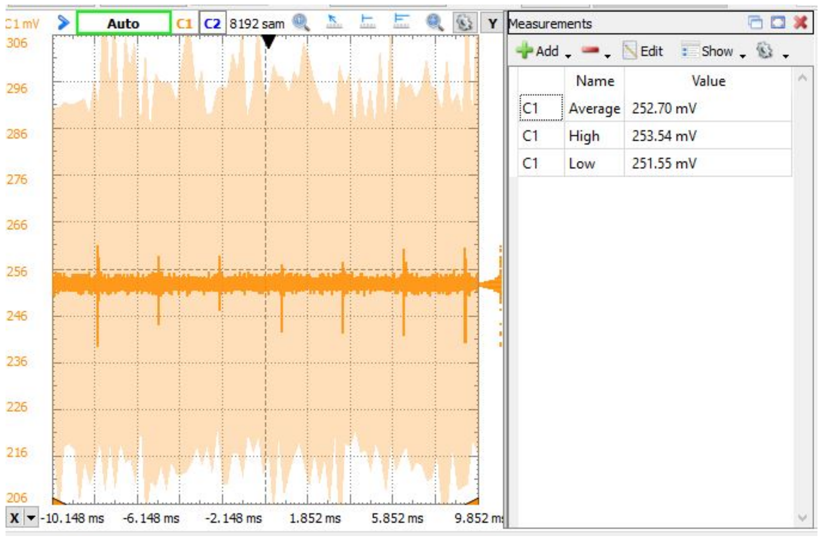
15) The average voltage output at ambient room temperature is 252 mV. This can be seen in the following figure. From the LM35 data sheet, the output voltage is 10 mV/°C · T. We can use this transfer function to calculate the temperature in celsius which is $252/10 = 25.2$. We can then convert this to Fahrenheit to get 77.36 degrees F. This result makes sense because the temperature sensor is measuring my ambient room temperature which is expected to be around that range.



	Name	Value
C1	Average	252.73 mV

Fig. 2 Output voltage of LM35

16) Figure 3 shows a zoomed version of the output voltage as measured by the AD2. From this figure, there is visible noise and a high and low measurement. The range between the high and low voltage measurements comes out to be around 2 mV as shown by the high and low measurements. This corresponds to about 0.2°C which is consistent with the data sheet's typical variation at room temperature. However, there is also considerable noise from the scope beyond this range shown in the light orange in Figure 3. Using the lighter orange color for the noise range there is a range of about 80 mV which is ±4°C which is considerably more error. Some of this error can come from the AD2 as it is sampling the voltage. To mitigate some of the errors due to noise, we could introduce a filter before making the voltage measurements. A lot of the noise comes from high frequency signals so a low pass filter before measuring with the AD2 could reduce some of the noise in the measurement. More signal conditioning such as using active filters and amplification could further reduce the noise present. You could also use a better sensor that has less noise in its output voltage or a better oscilloscope.



	Name	Value
C1	Average	252.70 mV
C1	High	253.54 mV
C1	Low	251.55 mV

Fig. 3 Output voltage of LM35

17) The output current limit for PORTB is 25 mA. For PORTG it is 2 mA. Thus, if we want to light a typical LED, we would have to use PORTB rather than PORTG since the current required to light the LED is within the limits of PORTB but is beyond the limits of PORTG. To use PORTG to power an LED, there would need to be a separate power source that can handle enough current.