CprE 288 – Homework Question Set 1 - SOLUTIONS

Question 1: C Variables and Number Representation

Complete the table below. Assume ASCII encoding of characters, and a 2's complement encoding of negative numbers. When representing a number in hexadecimal or binary, **use the proper number of digits** for the TM4C123 architecture (depends on type of variable). The first row is already completed.

	Value of x		
	x (decimal)	**x (hex)**	**x (binary)**
signed char x = 60;	60	0x3c	0b00111100
unsigned char x = 'K';	75	0x4B	0b0100_1011
unsigned char x = '7';	55	0x37	0b0011_0111
short x = 0x0381;	897	0x0381	0000_0011_1000_0001
unsigned short x = 0x4060 + 0x2051;	24753	0x60B1	0b0110 0000 1011 0001
signed char x = 0x7 + 6;	13	0x0D	0b0000 1101
unsigned int x = 0xE9;	233	0x000000E9	0b0000 0000 0000 0000 0000 0000 1110 1001
signed short x = -1;	-1	0xFFFF	0b1111 1111 1111 1111
signed char x = -128;	-128	0x80	0b1000 0000
signed char x = -1;	-1	0xFF	0b1111 1111
int x = -1;	-1	0xFFFF FFFF	0b1111 1111 1111 1111 1111 1111 1111 1111

Quick table to help with conversions:

Hex Digit	Decimal Value	Binary Value
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A or a	10	1010
B or b	11	1011
C or c	12	1100
D or d	13	1101
E or e	14	1110
F or f	15	1111

Question 2: TM4C123G Microcontroller & Documentation

Throughout this class you will be using the textbook and TM4C123G datasheet as a reference to learn about and design systems using the TM4C123G microcontroller. The TM4C123G datasheet can be found with the lab reference files and on Canvas with this homework assignment.

Bai Book Reading: Read Chapter 1, Section 1.2; Chapter 2, Sections 2.1 and 2.7. Refer to Chapter 4, Sections 4.5.7.1-4.5.7.2.4.

Datasheet Reading: Read Chapter 1 (pages 45-48), and scan Chapter 22 and Chapter 23.

V&Y ES Book Reading: Browse Valvano & Yerraballi, Chapter 2: Fundamental Concepts, Sections 2.1 – 2.4, http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C2_FundamentalConcepts.htm

Answer the following questions using the readings.

a. What is the maximum clock speed of the CPU of TM4C123G?
   80 MHZ

b. How many package pins does the TM4C123G microcontroller have?
   64 pins

c. What are the types and sizes of memory available, and the use for each type of memory?
   i) Flash memory: 256 KB; used for program code (non-volatile)
   ii) SRAM: 32 KB; used for program data
   iii) EEPROM: 2 KB; available for data storage (non-volatile)
   iv) Internal ROM: loaded with TivaWare software (e.g., Peripheral Driver Library, Boot Loader), AES cryptography tables, CRC error detection functionality

FYI, here's a short blog about using EEPROM:
http://humboldtmcu.blogspot.com/2014/08/energia-using-eeprom-on-ti-stellaris_4.html

d. What does it mean for a General Purpose I/O (GPIO) port pin to have an *alternative function*? Give one example of a GPIO port pin and its alternative function.

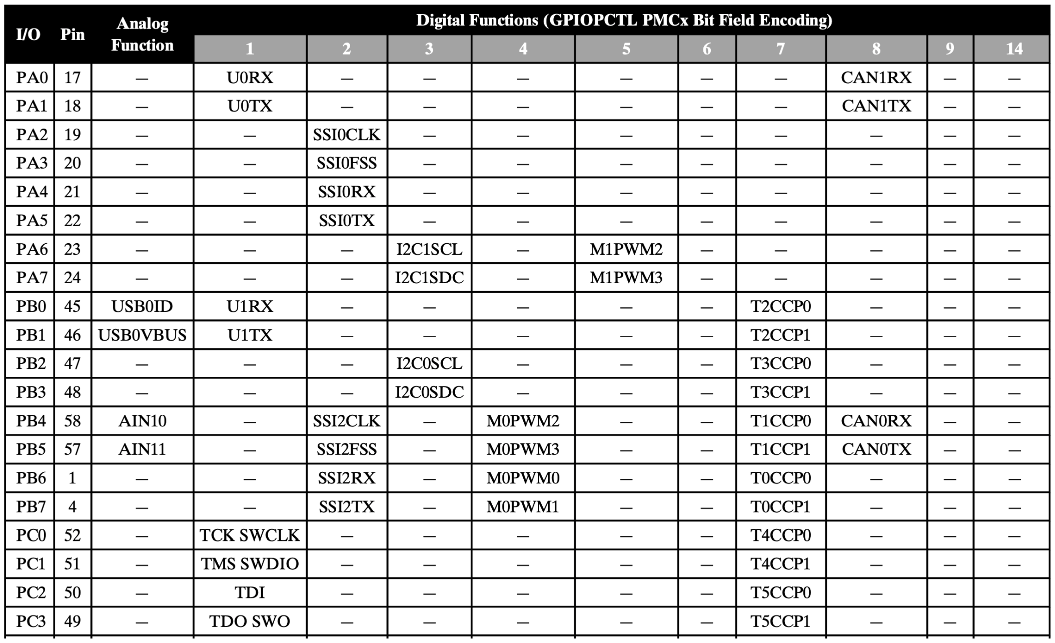
Textbook pg. 434 etc.
The regular function of a pin is to perform parallel I/O. Each pin can be configured as either an input or output pin. Most pins, however, have an alternative function. Using the GPIO Alternate Function Selection register (GPIOAFSEL), each pin can be configured to perform multiple or different functions, which are called alternative functions. The pins connect to special peripherals to provide special functions. For example, port pin PA0 can be either regular parallel port pin or an asynchronous serial port called universal asynchronous receiver/transmitter (UART) (U0RX).

Any port and pin under the I/O column provides the associated alternate functions in the row (columns 1 - 14).

Ex: Alternate function U0RX on Port A pin 0 (PA0)

The advantage of this feature in the GPIO module is that multiple functions can be configured and fulfilled by using a limited number of GPIO Ports and pins. The shortcoming is that this will make the GPIO Port configurations and programming more complicated and difficult.

Table 2.6. GPIO pins and alternate functions.



I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)									
			1	2	3	4	5	6	7	8	9	14
PA0	17	–	U0RX	–	–	–	–	–	–	CAN1RX	–	–
PA1	18	–	U0TX	–	–	–	–	–	–	CAN1TX	–	–
PA2	19	–	–	SSI0CLK	–	–	–	–	–	–	–	–
PA3	20	–	–	SSI0FSS	–	–	–	–	–	–	–	–
PA4	21	–	–	SSI0RX	–	–	–	–	–	–	–	–
PA5	22	–	–	SSI0TX	–	–	–	–	–	–	–	–
PA6	23	–	–	–	I2C1SCL	–	M1PWM2	–	–	–	–	–
PA7	24	–	–	–	I2C1SDC	–	M1PWM3	–	–	–	–	–
PB0	45	USB0ID	U1RX	–	–	–	–	–	T2CCP0	–	–	–
PB1	46	USB0VBUS	U1TX	–	–	–	–	–	T2CCP1	–	–	–
PB2	47	–	–	–	I2C0SCL	–	–	–	T3CCP0	–	–	–
PB3	48	–	–	–	I2C0SDC	–	–	–	T3CCP1	–	–	–
PB4	58	AIN10	–	SSI2CLK	–	M0PWM2	–	–	T1CCP0	CAN0RX	–	–
PB5	57	AIN11	–	SSI2FSS	–	M0PWM3	–	–	T1CCP1	CAN0TX	–	–
PB6	1	–	–	SSI2RX	–	M0PWM0	–	–	T0CCP0	–	–	–
PB7	4	–	–	SSI2TX	–	M0PWM1	–	–	T0CCP1	–	–	–
PC0	52	–	TCK SWCLK	–	–	–	–	–	T4CCP0	–	–	–
PC1	51	–	TMS SWDIO	–	–	–	–	–	T4CCP1	–	–	–
PC2	50	–	TDI	–	–	–	–	–	T5CCP0	–	–	–
PC3	49	–	TDO SWO	–	–	–	–	–	T5CCP1	–	–	–

PC4	16	C1−	U4RX	U1RX	–	M0PWM6	–	IDX1	WT0CCP0	U1RTS	–	–
PC5	15	C1+	U4TX	U1TX	–	M0PWM7	–	PHA1	WT0CCP1	U1CTS	–	–
PC6	14	C0+	U3RX	–	–	–	–	PHB1	WT1CCP0	USB0EPEN	–	–
PC7	13	C0−	U3TX	–	–	–	–	–	WT1CCP1	USB0PFLT	–	–
PD0	61	AIN7	SSI3CLK	SSI1CLK	I2C3SCL	M0PWM6	M1PWM0	–	WT2CCP0	–	–	–
PD1	62	AIN6	SSI3FSS	SSI1FSS	I2C3SDC	M0PWM7	M1PWM1	–	WT2CCP1	–	–	–
PD2	63	AIN5	SSI3RX	SSI1RX	–	M0FAULT0	–	–	WT3CCP0	USB0EPEN	–	–
PD3	64	AIN4	SSI3TX	SSI1TX	–	–	–	IDX0	WT3CCP1	USB0PFLT	–	–
PD4	43	USB0DM	U6RX	–	–	–	–	–	WT4CCP0	–	–	–
PD5	44	USB0DP	U6TX	–	–	–	–	–	WT4CCP1	–	–	–
PD6	53	–	U2RX	–	–	M0FAULT0	–	PHA0	WT5CCP0	–	–	–
PD7	10	–	U2TX	–	–	–	–	PHB0	WT5CCP1	NMI	–	–
PE0	9	AIN3	U7RX	–	–	–	–	–	–	–	–	–
PE1	8	AIN2	U7TX	–	–	–	–	–	–	–	–	–
PE2	7	AIN1	–	–	–	–	–	–	–	–	–	–
PE3	6	AIN0	–	–	–	–	–	–	–	–	–	–
PE4	59	AIN9	U5RX	–	I2C2SCL	M0PWM4	M1PWM2	–	–	CAN0RX	–	–
PE5	60	AIN8	U5TX	–	I2C2SDC	M0PWM5	M1PWM3	–	–	CAN0TX	–	–
PF0	28	–	U1RTS	SSI1RX	CAN0RX	–	M1PWM4	PHA0	T0CCP0	NMI	C0O	–
PF1	29	–	U1CTS	SSI1TX	–	–	M1PWM5	PHB0	T0CCP1	–	C1O	TRD1
PF2	30	–	–	SSI1CLK	–	M0FAULT0	M1PWM6	–	T1CCP0	–	–	TRD0
PF3	31	–	–	SSI1FSS	CAN0TX	–	M1PWM7	–	T1CCP1	–	–	TRCLK
PF4	5	–	–	–	–	–	M1FAULT0	IDX0	T2CCP0	USB0EPEN	–	–

e. The textbook describes two programing models presented in the textbook: Direct Register Access (DRA) and Software Driver (SD). This course **will only use** the Direct Register Access programming model. Briefly describe each of these models. Note: these are not covered in the assigned readings, so you will need to search a little farther.

Direct Register Access (DRA): In this model, code is written that writes values directly to a register and reads values from a register by specifying/dereferencing the address of a register.

Software Driver (SD): In this model, code is written that indirectly accesses registers via high-level function calls.

Question 3: Endianness

Find an online article on Endianness (i.e., little-endian and big-endian). Briefly summarize the concept of Endianness and the difference between little-endian and big-endian. Properly cite the article you use to obtain your information to receive credit. For proper formatting, see the IEEE citation reference manual: https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf . Note: while Wikipedia is discouraged for citations used within published academic papers, it can be useful for obtaining an overview on a topic. If you use a Wikipedia article, then use this guide for properly citing your “source”: http://www.wikihow.com/Cite-a-Wikipedia-Article-in-MLA-Format

Citation: Wikipedia Contributors, “Endianness,” *Wikipedia*. Wikipedia, The Free Encyclopedia, 27-Jan-2019.

Endianness is the order that data is stored in memory by a system.

When addressing memory in big-endian format, the most significant byte (contains most significant bit) is stored first and has the lowest address. The least significant byte stored last (having the highest address). In little endian format, the order is reversed. The byte with the least significant bit (called the least significant byte) is given the lowest address or is stored first, whereas the most significant byte is stored last (has the highest address)

Question 4: Basic Review

a. Fill in the units

__ms__	10^{-3} seconds (answer given)		
__µs__	10^{-6} seconds	__kB__	2^{10} Bytes
__ns__	10^{-9} seconds	__MB__	2^{20} Bytes
__kHz__	10^{3} Hz	__GB__	2^{30} Bytes
__MHz__	10^{6} Hz		

b. Period or frequency

i) What is the period of a 2 MHz clock?

$$T(sec) = \frac{1}{f(Hz)} \Rightarrow T = 1/(2 \times 10^6 Hz) \; = 0.5 \times 10^{-6}\, sec \;\Leftrightarrow .5\mu s \;\Leftrightarrow 500ns$$

ii) What is the frequency of a clock with a 1 ns period?

$$f(Hz) \;=\; \frac{1}{T(sec)} \Rightarrow f = 1/(1 \times 10^{-9} sec) \; = 1 \times 10^{9}\, Hz \;\Leftrightarrow 1GHz$$

iii) How many positive edges of a 4 MHz clock will occur in 4 ms?

In 1 clock period, # of positive edges = 1. Thus need to calculate the number of clock periods or cycles in 4 ms for the given clock frequency.

The clock period is 1/f = 1/4MHz = 0.25 μs

The number of periods is 4 ms / 0.25 μs = 16×10^3 = 16,000, so there are <u>16,000 positive edges</u>

Alternatively, the rate of edges per second = $(\#edges)/T = \#edges * f$

$$In\ 4\ ms \Rightarrow (1\ edges) * (4 \times 10^6 Hz) * (4 \times 10^{-3} sec) = 1.6 \times 10^4\ edges$$

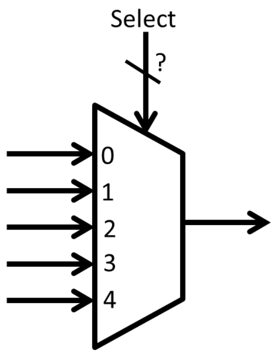
c. Digital logic

i) Briefly describe how a multiplexer works?

A device that allows several input channels to share one output channel. Input channels are assigned a code/number which is used to tell the MUX which input to feed through the output.

A n-channel multiplexer has n input channels, 1 output channel, and $\log_2(n)$ input select signals

ii) What is the **minimum** number of select bits (wires) need for the multiplexer below? Why?



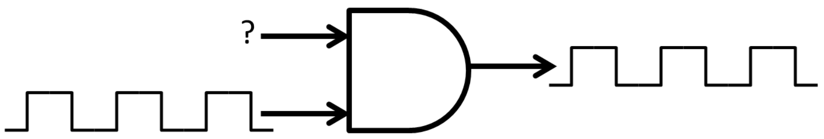
If *m* is the number of select bits, and *n* is the number of input channels, then the minimum value of *m* must satisfy $2^m \geq n$ *(where m and n are both integers)*

$$\Rightarrow 2^m \geq 5 \Rightarrow m \geq log_2(5) \Rightarrow m \geq roundup(2.32) \Rightarrow m = 3$$

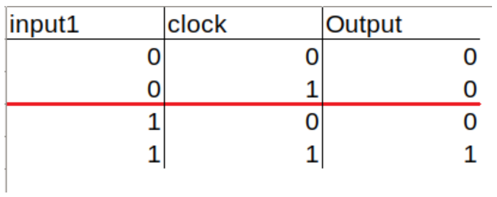
In short, 2 select bits can designate up to 2^2 = 4 input channels (0-3, or 0b00, 0b01, 0b10, 0b11). Here there are 5 input channels. Thus at least 3 select bits are needed.

Note: You will see multiplexers in TM4C microcontroller hardware interfaces. The muxes connect one of several possible inputs to an output. The select bits are associated with bit fields in control registers in the interface and are assigned values in software. More on this later.

iii) A clock signal is given as an input to an AND gate, as shown below. What value needs to be placed on the other input to allow the clock signal to propagate through the AND gate? Why?



The other input must equal logic 1 (or high). The AND function is shown in the truth table, in which a 0 input forces the output to 0, and a 1 input lets the output follow the other input.



input1	clock	Output
0	0	0
0	1	0
1	0	0
1	1	1

Note: While the above is a simplification, it represents the principle of “clock gating”. Clock gating is a method of reducing power consumption in systems-on-chip (“turning off” the clock to some circuits when it’s not needed). You will see “clock gating control registers” in the TM4C microcontroller. Before using an I/O component, the clock to the component needs to be “turned on” during initialization in software.

Question 5: C Variables and Memory Use

Reminder: Assume a TM4C123 is being used unless specified otherwise.

a. For each declaration, indicate how many bytes will be allocated in memory

i) ____**4**____ `long input_num2;`

ii) ____**40**____ `int sensor_readings[10];`

iii) ____**1600**____ `unsigned short scan_data[20][40];`

b. For each, indicate the value of `my_length` after executing each fragment of C code. Function strlen is a standard C library function. Specify N/A if the value of `my_length` cannot be determined.

i) `my_length` is ______7______.

```
char msg[] = “CPRE288”;
int my_length = 0xFFFFFFFF;

my_length = strlen(msg);
```

ii) `my_length` N/A__________. May be compiler-dependent. No room for NULL byte in msg initialization. Value depends on how the C compiler handles this.

```
char msg[7] = “CPRE288”;
int my_length = 0xFFFFFFFF;

my_length = strlen(msg);
```

iii) `my_length` is ____4____.

```
char msg[100] = {'C','P','R','E','\0','2','8','8','\0'};
int my_length = 0xFFFFFFFF;

my_length = strlen(msg);
```

iv) `my_length` is ____2____.

```
char msg[] = {5, 7, 10, 0, 9, 2, 0, 3, 5, 77, 23};
int my_length = 0xFFFFFFFF;

my_length = strlen(msg + 4);
```

Question 6: C-string Formatting

Given:

```
char message[100];
char str1[] = "CprE";
char str2[] = "iRobot";
int  num  = 100;
char ch1 = 50;  // 50 → 2; Use an ASCII table to convert these
char ch2 = 56;  // 56 → 8
```

Predict the C-string contained in `message` after each sprint.

Note 1: Treat each part independently.

Note 2: You may need to look up on your own more details on printf and sprintf.

a) `sprintf(message, "Read %d datasheet pages every week", num);`

Read 100 datasheet pages every week

b) `sprintf(message, "CprE%c%c%c is %s!", ch1, ch2, ch2, "fun");`

CprE288 is fun!

c) `sprintf(message, "Move the %s forward for %d cm", str2+1, num/2);`

Move the Robot forward for 50 cm

str2 is a pointer to the first character in the string, so str2+1 would be the second character onwards.

d)
```
sprintf(message, “The ASCII value for %c is decimal %d and hex %X”,
        ch1, ch1, ch1);
```

The ASCII value for 2 is decimal 50 and hex 32