Lab #1
Introduction to Embedded Systems

Mohammed Nafiz ALMadhoun

Introduction

In this lab, we are going to talk about the embedded systems course and lab, and talk about why embedded systems programming is not the same as programming a regular program as you used to in the previous years, then we will talk about our selected microcontroller that we will work within this semester, finally, we will talk about the research methodology that you should follow to help you in solving the labs assignments, and to work on your final project.

#1 Course vs Lab:

Starting with the course, you should understand that in this course you should grasp concepts from the book, just small ideas you need to pick from the examples in your assigned textbook, the course should give you a general idea on how to work with embedded systems, talk about some protocols that you should know, and explain electrical stuff that you will need to create a complete project, but in the lab, we will work on a different microcontroller, the ideas will stay the same, but you should be aware that we are working on a different microcontroller, the microcontroller we will work with is very close to the explained one in your textbook but still different, however, if you understand the concepts in your course, you should find it easy to reflect those ideas on any microcontroller you have its datasheet.

#2 Embedded Systems Programming:

Creating embedded systems means that you will have to work with very low and hardware-specific stuff, this task is not easy for a regular programmer, and in this kind of programming you should be very aware of the hardware itself, and the components connected to it to make it works well.

Then you should know that working with these systems means that you should work with a very time depended systems, these systems need you to be very precise as each millisecond will make your system behave differently.

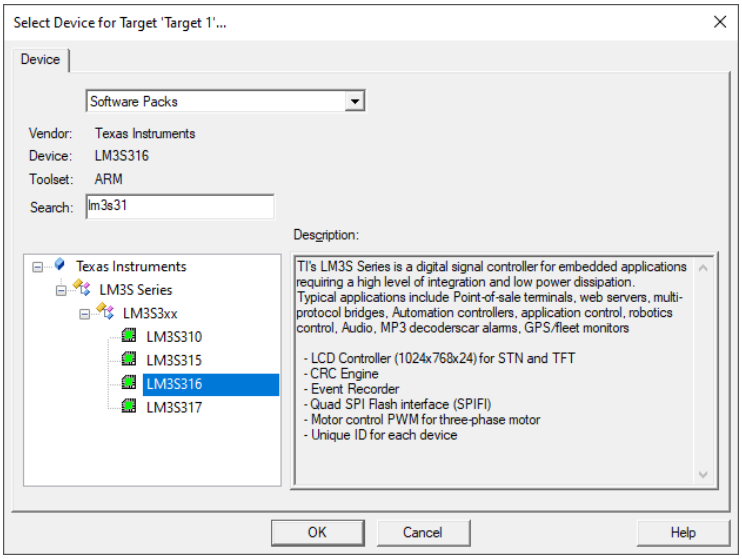
A final point a would like to talk about, is that these systems will mostly control stuff in real life, so a bug in your code might cost you your life (Ahh not for our small projects :V), so your code is a very important safety-critical code.

#3 LM3S316 Microcontroller:

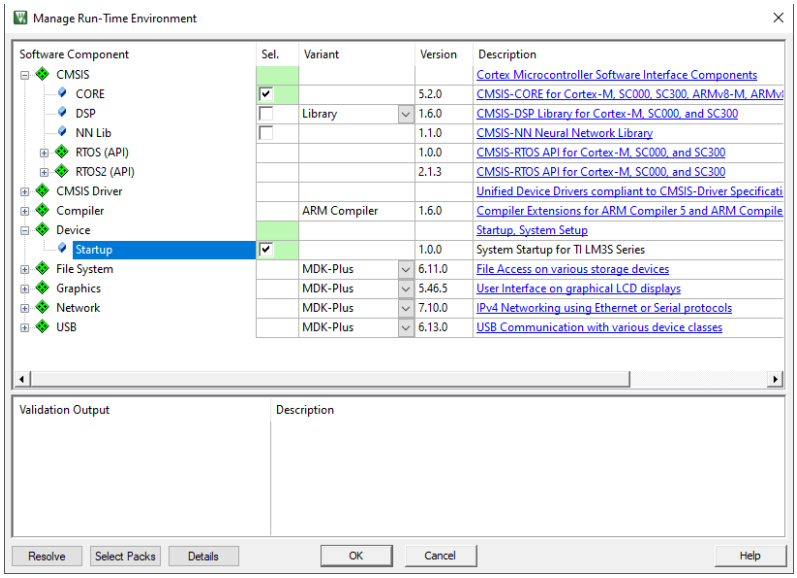
First, don't assume that this microcontroller is the best microcontroller on the earth, I've chosen this microcontroller as it has a simulator, and it's a Cortext-M3 microcontroller, and it has some cool peripherals like Times, Analog-To-Digital, PWM, UART, and I2C, so I find it very convent to work with and very close to the microcontroller that will be explained in your course, any time you think about a microcontroller to choose, think about what you need in it, and the cost of it, and if you can find in easily in the market.

So don't forget to install the LM3S316 in your Keil from the pack installer, and here are some screenshots on how to configure your project in Keil.

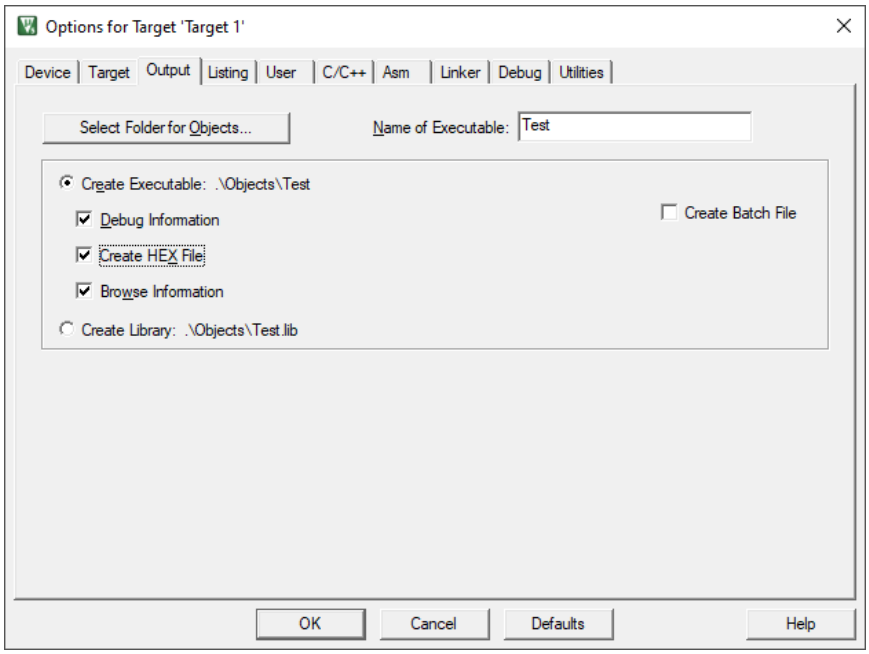
1- From Project menu click "new uVision Project" then select the microcontroller.



2- The enable the CORE and Startup components.

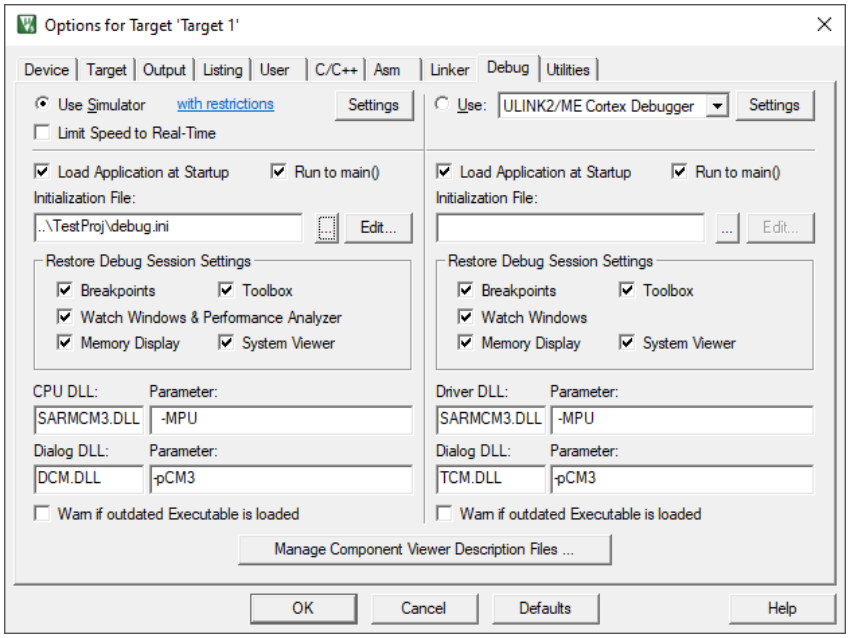


3- Now from Project view right-click on “Target 1” and click “Options for Target …”, then click on the Output tab, and enable create HEX file (This file will describe the ROM of the microcontroller for programmers and simulators).



4- From the Debug tab, click on Use Simulator, then in the initialization file, you should choose a configuration file that contains this line:

```
MAP  0x40000000, 0x40F00000  READ WRITE
```



This is just a dump solution to solve a problem with Keil and this microcontroller peripherals.

Now, to learn everything about this microcontroller, you should read the datasheet, (Not like read it all, just what do you need from it).

Link: http://www.ti.com/lit/ds/symlink/lm3s316.pdf

#4 The first example:

In this example, we will write a small code that reads the GPIO (General Purpose Input-Output) ports, we will read PORTA data, and we will write back on it, so the purpose of this example is to flip the status of some LEDs in case of you pushed a button.

```
#define SYSCTL_RCGC2_R      (*((volatile unsigned long *)0x400FE108))
#define GPIO_PORTA_DIR_R    (*((volatile unsigned long *)0x40004400))
#define GPIO_PORTA_DATA_R   (*((volatile unsigned long *)0x400043FC))

int main () {
        //setup phase
        SYSCTL_RCGC2_R |= 0b1;
        GPIO_PORTA_DIR_R |= 0b111100;
        GPIO_PORTA_DATA_R = 0b101000;

        while (1) {
                //looping phase
                if (GPIO_PORTA_DATA_R & 0x1) {
                        GPIO_PORTA_DATA_R ^= 0b111100;
                        while ((GPIO_PORTA_DATA_R & 0x1));
                }
        }
}
```

Starting from the main function, this function contains two blocks, the first block is the setup phase, in this phase we will configure and enable the microcontroller peripherals, to enable PORTA GPIO, we should enable its clock by writing 1 to the first bit in RCGC2 register.

Run Mode Clock Gating Control Register 2 (RCGC2)

Base 0x400F.E000
Offset 0x108
Type R/W, reset 0x00000000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved											GPIOE	GPIOD	GPIOC	GPIOB	GPIOA
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1: RCGC2 Register [1]

So this register will enable the clock for PORTA, so we can use it later.

Then we will configure the direction of PORTA, direction means if it's input or output, one means output, and zero means input, return to page 237 in the datasheet to understand more.

Then we wrote some data to the port, this will be our starting point, two LEDs will be shining and two LEDs will be off.

Now in the looping phase, this phase will run the whole time, so our main code will be here, in the first line in the loop, we checked if the first bit of PORTA is 1, if the condition is true, we will flip the values of bit2-5 by XORing the data with 0b111100, then we will wait until the button goes up (the value of it return to zero), this will work in the simulator but not in the real-life (As you know about bouncing issue).

***Note**: we will learn about a C keyword that we will use it a lot in embedded systems (You should understand the concept of pointers from your OS course), the keyword is volatile, the volatile keyword will prevent the code optimizer from removing the variables that you didn't use (Or what the compiler think that you didn't use), so if you want to address something that changes, and it's not a part of the memory (An address for a peripheral), you should describe the address that it is volatile, so the compiler won't get the value from a previously saved register, and won't optimize your code as you didn't use that memory area (volatile keyword is not that simple actually, you can read about it more).*

You should implement this in Keil then compile it, the last thing you should do is simulating this in Proteus, you will add the microcontroller itself at first, then double click on it to choose the hex file, then add your LEDs and Button.

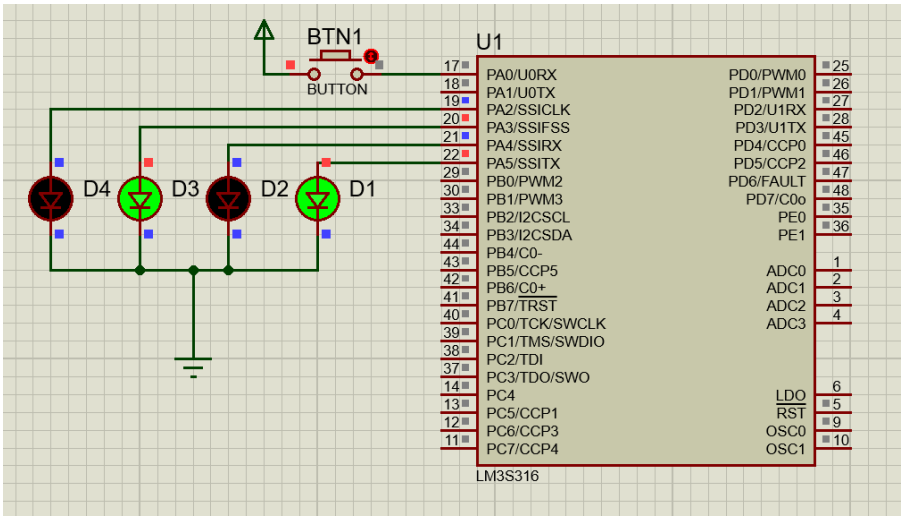


Figure 2: Proteus Simulation

Tada, You can click on the button now to flip the LEDs :D

Lab Tasks

Task #1: Just a simple task (20 min, 5 min)

In this task, you should edit the code above, to make a design that takes two buttons as input, if this first one clicked you should shine 4 LEDs (the four LEDs should be connected to PORTB), then if you clicked the second one you should turn all the LEDs off.

Task #2: Shift the LEDs (10 min, 5 min)

Using the your code and proteus project in Task #1, edit the code so that just one LED from the four LEDs will shine on the begging, and if the first button is clicked you should shift the shining LED to left, and if the second button is clicked you should shift the shining LED to the right, (Turn off the LED and turn on the sibling LED), implement this as rotation.