edX (https://www.edx.org)

UTAustinX: UT.6.01x Embedded Systems - Shape the World

KarenWest (/dashboard)

Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) **Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)**

Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) **Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)**

Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)

Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

Help

The basic approach to Lab 6 will be to develop and debug your system using the simulated negative logic switch (PF4) and positive logic LED (PF2). In Lab 8, you will build and test an external switch and LED, which you will need to connect yourself. However, this lab will run on the LaunchPad using SW1 on PF4 and the blue LED on PF2, which come pre-built into the LaunchPad.

To run the Lab 6 grader in simulation mode verify the two settings in Figure 6.2. In particular, execute Project->Options and select the Debug tab. The "Use Simulator" option must be selected. Second, notice the parameter field includes **-dedXLab6**.

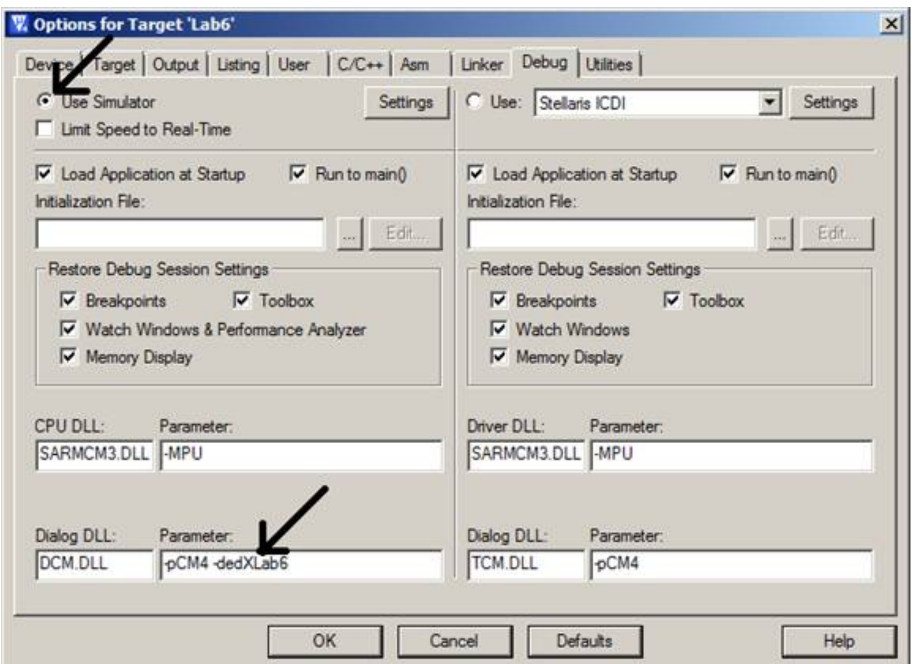

Figure 6.2. Configuration used to run Lab 6 in simulation.

When debugging your software you can press the switch, shown as A in Figure 6.3. While running you can see the output values on the LED, which is a three-color LED connected to pins PF3, PF2 and PF1. When you are ready to grade, 1) place the 4-digit number into the **NumberFromEdX** field, 2) click **Grade** and wait for the grading to finish, and then 3) copy the 8-digit ASCII string in **CopyThisToEdX** back to edX.

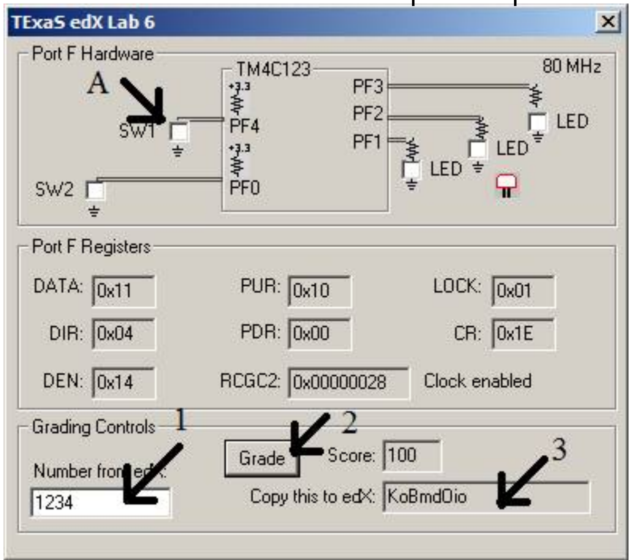

Figure 6.3. Using TExaS to debug your software in simulation mode. Press the switch at "A" to press and release the input SW1. When you are finished: 1) get the Number from edX, 2) hit the Grade button, and 3) Copy the code back into edX.

Time is very important in embedded systems. One of the simplest ways in which we manage time is determining how long it takes to run our software. There are two ways to determine how long each instruction takes to execute. The first method uses the ARM data sheet. For example, Figure 6.4 is a page from the Cortex-M4 Technical Reference Manual. E.g., see pages 34-38 of: CortexM4_TRM_r0p1.pdf (/c4x/UTAustinX/UT.6.01x/asset/CortexM4_TRM_r0p1.pdf)

Operation	Description	Assembler	Cycles
Count	Leading zeroes	`CLZ Rd, Rn`	1
Load	Word	`LDR Rd, [Rn, <op2>]`	2^b
	To PC	`LDR PC, [Rn, <op2>]`	$2^b + P$
	Halfword	`LDRH Rd, [Rn, <op2>]`	2^b
	Byte	`LDRB Rd, [Rn, <op2>]`	2^b
	Signed halfword	`LDRSH Rd, [Rn, <op2>]`	2^b
	Signed byte	`LDRSB Rd, [Rn, <op2>]`	2^b

Figure 6.4. From the Technical Reference Manual page 34.

On the TM4C123 the default bus clock is 16 MHz ±1%. When using the automatic grader, we will activate the phase lock loop (PLL), and the bus clock will be exactly 80 MHz. The following is a portion of a listing file with a simple delay loop. The SUBS and BNE are executed 800 times. The SUBS takes 1 cycle and the BNE takes 1 to 4 (a branch takes 0 to 3 cycles to refill the pipeline). The minimum time to execute this code is 800*(1+1)*12.5 ns = 20 us. The maximum time to execute this code is 800*(1+4)*12.5 ns = 50 us. Since it is impossible to get an accurate time value using the cycle counting method, we will need another way to estimate execution speed. An accurate method to measure time uses a logic analyzer or oscilloscope. In the simulator, we will use a simulated logic analyzer, and on the real board we will use a real oscilloscope. To measure execution time, we cause rising and falling edges on a digital output pin that occur at known places within the software execution. We can use the logic analyzer or oscilloscope to measure the elapsed time between the rising and falling edges. In this lab we will measure the time between edges on output PF2.

```
0x00000158 F44F7016       MOV R0,#800
0x0000015C 3801     wait  SUBS R0,R0,#0x01
0x0000015E D1FD           BNE  wait
```

*(note: the **BNE** instruction executes in 3 cycles on the simulator, but in 4 cycles on the real board)*

The following C function can be used to delay. The number 1333333 assumes 6 cycles per loop (100ms/12.5ns/6). The Keil optimization is set at Level 0 (-O0) and the “Optimize for Time” mode is unchecked.

```
void Delay100ms(unsigned long time){
 unsigned long i;
 while(time > 0){
  i = 1333333;  // this number means 100ms
  while(i > 0){
   i = i - 1;
  }
  time = time - 1; // decrements every 100 ms
 }
}
```

The corresponding assembly code for the above C function is:

```
0x000003A8 E005             B     test1
0x000003AA 4904 loop1   LDR   r1,=1333333
0x000003AC E000             B     test2
0x000003AE 1E49 loop2   SUBS  r1,r1,#1   ;1 cycle
0x000003B0 2900 test2   CMP   r1,#0x00   ;1 cycle
0x000003B2 D1FC             BNE   loop2     ;4 cycles
0x000003B4 1E40             SUBS  r0,r0,#1
0x000003B6 2800 test1   CMP   r0,#0x00
0x000003B8 D1F7             BNE   loop1
0x000003BA 4770             BX    lr
```

Part a) Write a main program in C that implements the input/output system. Pseudo code and flowchart are shown, illustrating the basic steps for the system. We recommend at this early stage of your design career you access the entire I/O port using GPIO_PORTF_DATA_R. After you fully understand how I/O works, then you can use bit-specific addressing to access I/O ports.

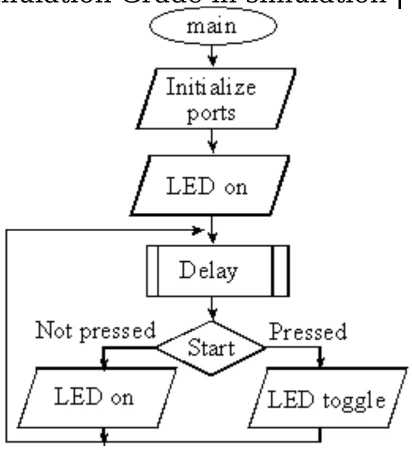

Figure 6.5. Flowchart for Lab6

main Turn on the clock for Port F
Clear the PF4 and PF2 bits in Port F AMSEL to disable analog
Clear the PF4 and PF2 bit fields in Port F PCTL to configure as GPIO
Set the Port F direction register so
PF4 is an input and
PF2 is an output
Clear the PF4 and PF2 bits in Port F AFSEL to disable alternate functions
Set the PF4 and PF2 bits in Port F DEN to enable digital
Set the PF4 bit in Port F PUR to activate an internal pullup resistor
Set the PF2 bit in Port F DATA so the LED is initially ON
loop Delay about 100 ms
Read the switch and test if the switch is pressed
If PF4=0 (the switch is pressed),
toggle PF2 (flip bit from 0 to 1, or from 1 to 0)
If PF4=1 (the switch is not pressed),
set PF2, so LED is ON
Go to loop

Part b) Test the program in simulation mode. Use the built-in logic analyzer to verify the LED is toggling at the rate at which it was designed. In particular, try to recreate a screenshot like Figure 6.1 showing when the switch is pressed, the LED is ON for 100 ms and OFF for 100 ms. The same code runs at a speed just a little bit slower on the real board as compared to running on the simulator. For this reason the graders will allow for some tolerance when measuring times of your lab solution. For example, a 100-ms delay running on the real board may look like a 86-ms delay running in simulation.

Simulators typically run slower than real hardware. Use the built in logic analyzer to measure how much Cortex-M time is simulated in 10 actual seconds. Hit the reset twice to clear the time axis on the plot. Run the simulator for 10 human seconds (real time with your watch), and then stop simulation. Observe the logic analyzer time to determine how much Cortex-M time was simulated. For example, my computer simulated 15 sec of Cortex-M time in 10 sec of human time, meaning the simulator was running 150% faster than a real Cortex-M. There are many factors that affect this ratio, so do expect to see this ratio vary a lot. The point of the exercise is to get a sense of how a simulator manages time.

GRADE IN SIMULATION

	0:00 / 2:12	1.0x				

PROFESSOR JONATHAN VALVANO: Let me show you the steps to get a grade for

the lab six in simulation mode.

We need two Windows open.

We need edX open, and we need Keil open.

First you write your program and debug it.

And when you're ready to grade, these are the steps

Help