Module 1

Lab 1: Running code on the LaunchPad using CCS

Lab: Running code on the LaunchPad using CCS

1.0 Objectives

The purpose of this lab is to prepare your workstation to write software that will be loaded on the LaunchPad.

1. You will learn how to install the CCS IDE
2. You will load starter code on the MSP432 LaunchPad.
3. You will learn and practice the debug capability inside the CCS IDE

Good to Know: Using an IDE is an important tool in embedded systems design. This is the crucial first step before interacting with the hardware.

1.1 Getting Started

1.1.1 Software Starter Projects

Look at these four projects:
SineFunction (a simple implementation of sine),
Input_Output (switch input LED output example)
TExaS (example use of logic analyzer and oscilloscope)
UART (serial output to Terminal program, implementing printf)

1.1.2 Student Resources

MSP432P401R SimpleLink™ Microcontroller LaunchPad™ Development Kit (MSP‑EXP432P401R) User Guide (SLAU597)
MSP−EXP432P401R Quick Start Guide (SLAU596)
MSP432P4xx Technical Reference Manual (SLAU356)
MSP432P401R Datasheet (SLAS826)
TI Resource Explorer (MSP432 SimpleLink SDK)
TI SimpleLink Academy (http://dev.ti.com/MSP432-Simplelink-Academy)
SimpleLink is a Texas Instruments' umbrella term that includes much of its embedded system produces, such as microcontrollers, wireless, TI RTOS, and IoT.

1.1.3 Reading Materials

TI Resource Explorer, http://dev.ti.com/tirex/
Development Tools-> Integrated Dev. Environ. -> Code Composer Studio

Volume 1 Chapter 1, Sections 2.1, 2.2, and 2.3
Embedded Systems: Introduction to the MSP432 Microcontroller",
or
Volume 2 Sections 1.1, 1.2, and 1.3
Embedded Systems: Real-Time Interfacing to the MSP432 Microcontroller"

1.1.4 Components needed for this lab

Quantity	Description	Manufacturer	Mfg P/N
1	MSP-EXP432P401R LaunchPad	TI	MSP-EXP432P401R

1.1.5 Lab equipment needed (none)

1.2 System Design Requirements

Throughout the course you will acquire knowledge that will allow you to build a system that includes mechanical and electrical subsystems. The goal of this first lab is to set up our ability to write firmware for the robot and learn what debugging options are available to troubleshoot the system. In this lab, you will

- Install Code Composer Studio 7.0 or above
- Download and unpack associated files for this course and import the example projects into Code Composer Studio
  1. Data sheet
  2. Software documentation
  3. TExaSdisplay application (scope, logic analyzer)
  4. Example CCS and lab starter projects
- Install the Windows Drivers needed to debug the MSP432 LaunchPad
- Learn the basic steps for software development with CCS
  1. Build (compile)
  2. Debug (download and start debugger)
  3. Run, step, step in, step over, step out
  4. Breakpoint
  5. Observe variables, ports, memory

Note: CCS provides a rich set of debugging tools. Because the robot is an embedded system, we are not concerned with just the software, but rather, we will debug the hardware and software together. As you progress in the course you will continue to discover new features with CCS to help in development and debugging. In general, we can group the techniques into two classifications: control (making the software/hardware system do what you want), and observability (visualizing what the software/hardware system did.)

Lab: Running code on the LaunchPad using CCS

1.3 Experiment set-up

This lab uses the LaunchPad without any external input or output hardware. All that is needed is your computer that you will use in the course, the MSP432 LaunchPad, and the included USB cable.

1.4 System Development Plan

1.4.1 Installing CCS

You will first need to download the latest CCS version from TI. It is recommended to get at least CCS 7.0 or above to do the work in this course.

http://www.ti.com/tool/ccstudio

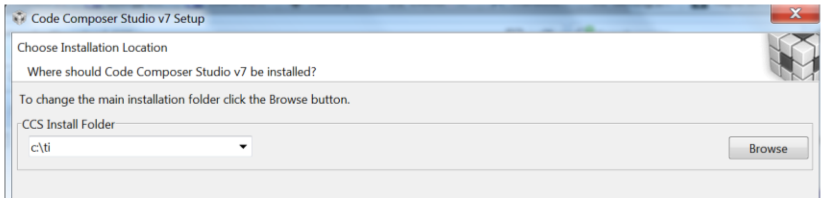
What is the difference between web installer and offline installer? The web installer is a small installation program that you download and execute. You then make your installation selections (device families and features desired) and the installer then downloads and installs only those selected packages. The offline installer is a large package that includes all packages (except for those only available via the CCS App Center). The offline installer is generally only recommended if you have issues with your firewall or anti-virus software blocking the web installer. The offline installer is also useful if you need to install CCS on a machine that does not have internet access. **For this curriculum, you can use either web or offline installer; we suggest using the web installer.**

General tips for installing CCSv7

- It is necessary for you to select MSP432 support during installation. MSP432 support includes the device drivers that allow CCS to program and debug LaunchPad software.
- Clean out all prior failed or incomplete installations (by deleting the install directory) before attempting a new one to the same directory. (On the install directory in Windows, use Shift+Del and in Linux and MacOS use rm -Rf <install directory>)
- If you plan to install two versions side-by-side, **always** use different workspaces. Sharing a workspace between two versions may cause severe impact in project building and debugging.
- Disable anti-virus (certain anti-virus software is known to cause problems). If it cannot be disabled, try the offline installer instead of web installer: Download CCS
- Ensure that your **Username** does not have any non-ASCII characters, and that you are installing CCS to a directory that does not have any non-ASCII characters. A temporary directory using the **Username** is created during installation. Eclipse is unable to handle non-ASCII characters. If your **Username** does have non-ASCII characters, please create a temporary admin user for installing CCS.

1.4.2 Running the CCS installer

Begin the installation process after downloading the latest version of CCS. By default it will ask you to install under a ti folder, which is recommended.


Code Composer Studio v7 Setup

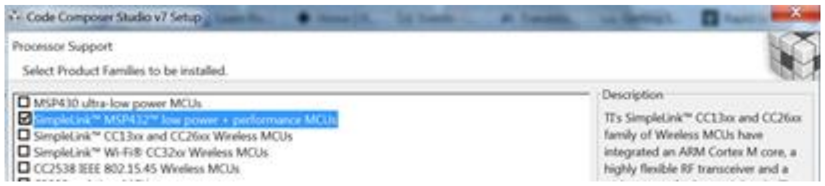
Choose Installation Location

Where should Code Composer Studio v7 be installed?

To change the main installation folder click the Browse button.

CCS Install Folder: c:\ti

During the initial setup please make sure that you select processor support for SimpleLink MSP432 MCUs. The processor support matches our MSP432P401R LaunchPad development kit. Installing other processor support is optional but this course will only use the MSP432.


Code Composer Studio v7 Setup

Processor Support

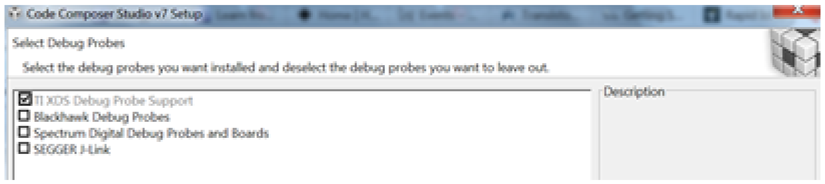
Select Product Families to be installed.

- [ ] MSP430 ultra-low power MCUs
- [✓] SimpleLink™ MSP432™ low power + performance MCUs
- [ ] SimpleLink™ CC13xx and CC26xx Wireless MCUs
- [ ] SimpleLink™ Wi-Fi® CC32xx Wireless MCUs
- [ ] CC2538 IEEE 802.15.4 Wireless MCUs

Description

TI's SimpleLink™ CC13xx and CC26xx family of Wireless MCUs have integrated an ARM Cortex M core, a highly flexible RF transceiver and a

Under Debug Probe support selection, make sure that the default "TI XDS Debug Probe Support" is selected. This is the debugger used on the LaunchPad development kit. The other options are for external debuggers, but these debuggers will not be used in this course.


Code Composer Studio v7 Setup

Select Debug Probes

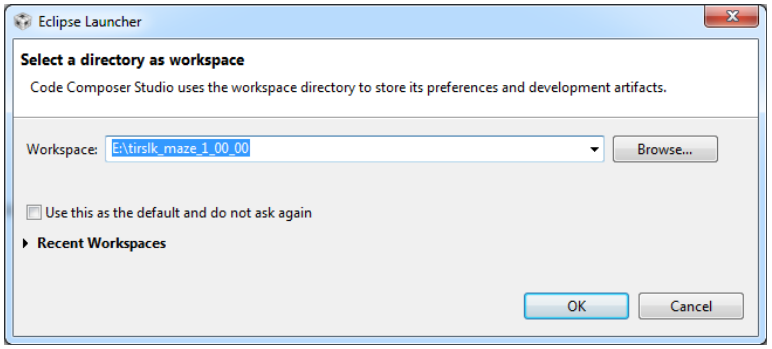
Select the debug probes you want installed and deselect the debug probes you want to leave out.

- [✓] TI XDS Debug Probe Support
- [ ] Blackhawk Debug Probes
- [ ] Spectrum Digital Debug Probes and Boards
- [ ] SEGGER J-Link

Description

Lab: Running code on the LaunchPad using CCS

Click finish and your installation should proceed to completion. When completed you can open CCS and select your workspace. The default workspace is recommended other projects but for this course you will create a custom workspace called **tirslk_maze,** as described in the next section.


Eclipse Launcher
Select a directory as workspace
Code Composer Studio uses the workspace directory to store its preferences and development artifacts.
Workspace: E:\tirslk_maze_1_00_00
Browse...
Use this as the default and do not ask again
Recent Workspaces
OK
Cancel

1.4.3 Import tirslk_maze

We are going to import all the curriculum project folders into CCS for our next step, creating one workspace for the entire course.

tirslk_maze is a set of software components that includes many (40) CCS example projects, html documentation, data sheets, and a Windows application called **TExaSdisplay**. Some of the example projects run out of the box and are intended to illustrate various functionalities of the MSP432. However, some of the projects have names beginning with "Lab", and these are starter projects for your labs. You will be developing code in these projects as part of the lab assignments in this curriculum. Furthermore, the folder **inc** has files you develop in one lab that will be used in subsequent labs. The steps to install **tirslk_maze** are

Step 1: Download the zip file www.ti.com/lit/zip/SLAC768.

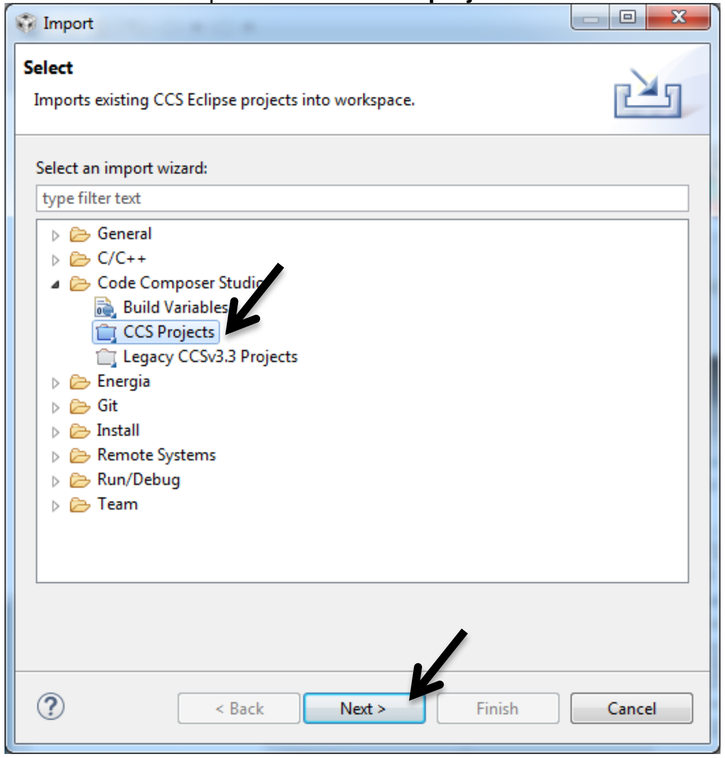
Step 2: Extract the zip to a file location you want the projects to reside. Preferably, an easy to find location on your computer. Once unzipped and compiled, the **tirslk_maze_1_00_00** folder will expand to about 200 MB. In the subsequent figures you can see I extracted it to E:\

Step 3: Open the software documentation by double-clicking on the file
tirslk_maze_1_00_00_Software_Documentation.html
This documentation includes software provided to you as examples and software you will write as part of the lab sequence.

Step 4: Open the datasheets folder. In this directory you will find descriptions of the hardware components used in this curriculum. We suggest you begin with these two reference manuals
Meet the LaunchPad, slau596.pdf
MSP432P4xx Technical Reference Manual, slau356f.pdf

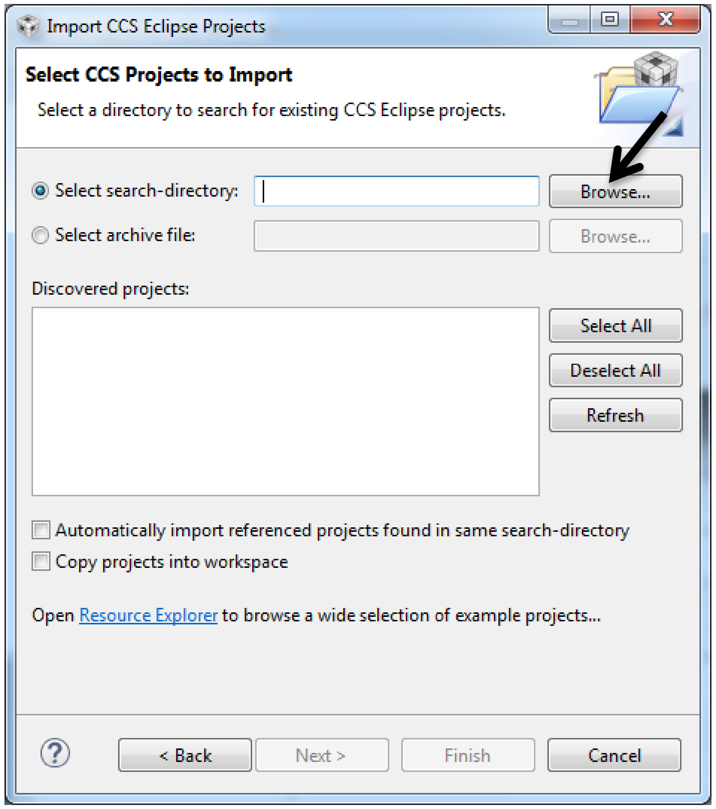
Step 5: Start CCS. The simplest approach to setting up the software for this course is to use the unzipped folder from step 2 as your workspace. You need to switch to this workspace using **File > Switch Workspace**. Browse to the unzipped folder from step 2 and select OK.

Step 6: Import all the projects into CCS. From the menu bar, click
File > Import…
Choose Code Composer Studio > **CCS projects** and click **Next>**



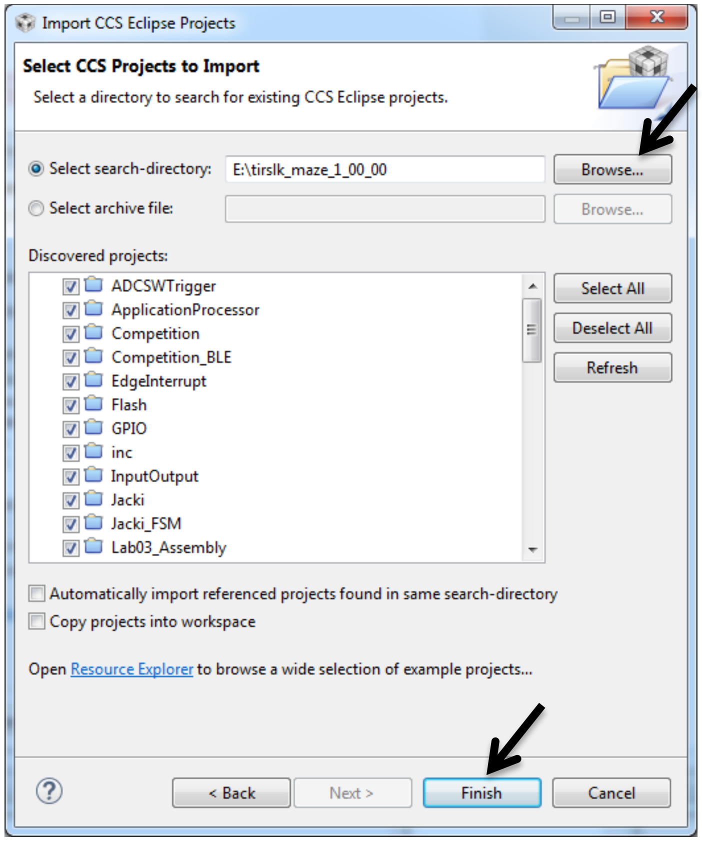
Lab: Running code on the LaunchPad using CCS

Select search-directory and click the **Browse…** option, and find the unzipped **tirslk_maze_1_00_00** folder that you created in step 2.


Import CCS Eclipse Projects
Select CCS Projects to Import
Select a directory to search for existing CCS Eclipse projects.
Select search-directory:
Browse...
Select archive file:
Browse...
Discovered projects:
Select All
Deselect All
Refresh
Automatically import referenced projects found in same search-directory
Copy projects into workspace
Open Resource Explorer to browse a wide selection of example projects...
< Back
Next >
Finish
Cancel

CCS should discover many projects inside the **tirslk_maze_1_00_00** folder.

Click **Select All** (do not check **Automatic import** or **Copy projects** options). This will have CCS reference the project from the original location and preserve the original directory structure required to build. Click Finish


Import CCS Eclipse Projects
Select CCS Projects to Import
Select a directory to search for existing CCS Eclipse projects.
Browse...
Select archive file:
Browse...
Discovered projects:
Select All
Deselect All
Refresh
Automatically import referenced projects found in same search-directory
Copy projects into workspace
Open Resource Explorer to browse a wide selection of example projects...
< Back
Next >
Finish
Cancel

Select search-directory: E:\tirslk_maze_1_00_00

Discovered projects:
- [✓] ADCSWTrigger
- [✓] ApplicationProcessor
- [✓] Competition
- [✓] Competition_BLE
- [✓] EdgeInterrupt
- [✓] Flash
- [✓] GPIO
- [✓] inc
- [✓] InputOutput
- [✓] Jacki
- [✓] Jacki_FSM
- [✓] Lab03_Assembly

Now your projects for the course are imported and visible in the project explorer. We are now set up with CCS!

Step 7: Notice over 40 projects in the project explorer. All but one of the projects will be used in this curriculum. Click on the **inc** folder within Project explorer and notice the files within the folder. We will not be using the **inc** project for any code development. The **inc** folder contains software that will be shared between projects throughout the curriculum. The **inc** project was created for the sole purpose of making it easy for you to open the files from the project explorer. The **inc** files, listed in Table 1, are completely written and available as example code for the MSP432. On the other hand, you are required to complete the **inc** files that are listed in Table 2 as part of the lab assignments. For both sets of files the software documentation explains what the functions are and how to use them.

Header	Code	Purpose
AP.h	AP.c	Application Processor, BLE
Clock.h	Clock.c	Sets bus clock to 48 MHz
CortexM.h	Cortex.c	Enable and disable interrupts
FlashProgram.h	FlashProgram.c	Erase and program flash
GPIO.h	GPIO.c	Digital I/O, CC2650 BLE
LaunchPad.h	LaunchPad.c	LaunchPad LEDs / switches
LPF.h	LPF.c	Low pass filters
SysTick.h	SysTick.c	24-bit system timer
SysTickInt.h	SysTickInt.c	Periodic interrupt
TA0InputCapture.h	TA0InputCapture.c	Period measurement
TA2InputCapture.h	TA2InputCapture.c	Period measurement
Tachometer.h	Tachometer.c	Tachometer interface
TExaS.h	TExaS.c	Oscilloscope, logic analyzer
Timer32.h	Timer32.c	32-bit periodic interrupt
TimerA0.h	TimerA0.c	16-bit periodic interrupt
TimerA2.h	TimerA2.c	16-bit periodic interrupt
UART.h	UART.c	Serial port
Ultrasound.h	Ultrasound.c	Ultrasonic sensor interface

Table 1. Shared files you can use. I.e., these files are complete and functional.

Header	Code	Purpose (Lab)
ADC14.h	ADC14.c	Analog to digital conv. (15)
Bump.h	Bump.c	Bump sensors (10)
BumpInt.h	BumpInt.c	Interrupting sensors (14)
	convert.asm	Assembly functions (3)
IRDistance.h	IRDistance.c	Distance conversions (15)
Motor.h	Motor.c	Motor interface (13)
MotorSimple.h	MotorSimple.c	Simple motor interface (12)
Nokia5110.h	Nokia5110.c	LCD interface (11)
PWM.h	PWM.c	Pulse width modulation (13)
Reflectance.h	Reflectance.c	Line sensor (6 and 10)
TA3InputCapture.h	TA3InputCapture.c	Input capture, tachometer (16)
TimerA1.h	TimerA1.c	Periodic interrupt (13)
UART1.h	UART1.c	Interrupting serial port (18)

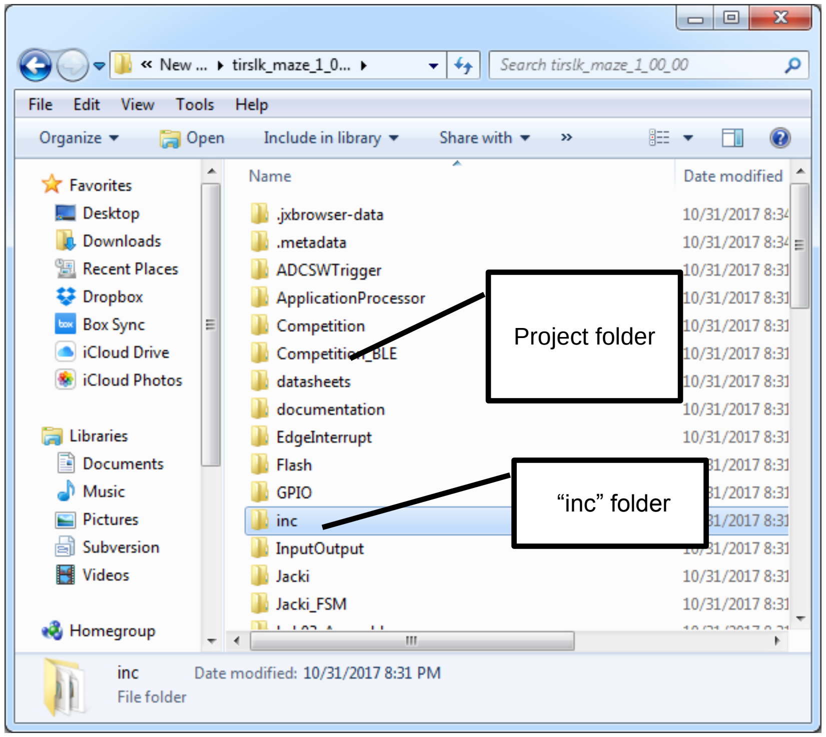
Table 2. Shared files you will need to complete. I.e., you need to complete the functions in these files in order for them to operate properly.

1.4.4 Project structure

To better understand this course we need to explain the project structure inside the **tirslk_maze_1_00_00** folder. **tirslk_maze_1_00_00** is an archive of the course projects that a student can unpack, compile and build in CCS. There are more than 40 projects with the same structure.

- CCS 7.x, C99 language, **doxygen** documentation
- Configured for the MSP432P401R LaunchPad
- No use of TI libraries or any external libraries
- Just C code (there is one **Solution.obj** in Lab4)

There is one folder with shared C code called "**inc**" that contains files used in multiple projects. For example, **bump.c** and **bump.h** are in the **inc** folder. Whenever a project wishes to use one of these shared files, the code file (e.g., **bump.c**) is added to the project (linked) and the header file is included using #include (e.g., **#include "..\inc\bump.h"**)



Note that projects with "Lab" in the name are intended as starter projects for each lab. Other projects are examples. The projects that begin with "Competition" can be used to develop high-level code without developing all the low-level I/O driver code.

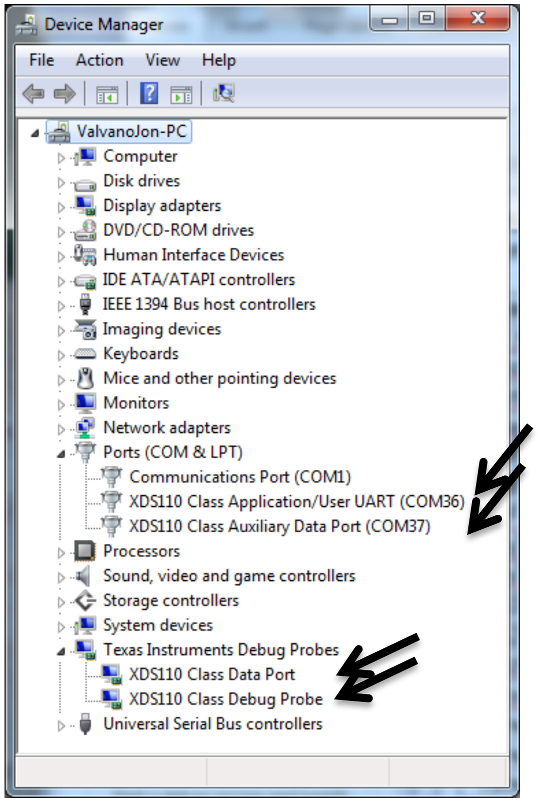
Lab: Running code on the LaunchPad using CCS

1.4.5 Installing OS drivers for the LaunchPad

Drivers are OS software that allow CCS to communicate with the XDS110 debugger on the LaunchPad.

The first step to installing drivers is to plug the MSP432 LaunchPad into the PC using the USB cable. Some LEDs on the LaunchPad should light up. When you plug your LaunchPad into a USB port on your computer, the operating system will attempt to load drivers. If you selected MSP432 support during CCS installation, then the operating system should automatically find the drivers. On Windows there will be four drivers in the device manager associated with the LaunchPad.

Notice there are two COM ports. We will exclusively be using the first one (the one with the lower number).



```
ValvanoJon-PC
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Human Interface Devices
  IDE ATA/ATAPI controllers
  IEEE 1394 Bus host controllers
  Imaging devices
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
  Ports (COM & LPT)
    Communications Port (COM1)
    XDS110 Class Application/User UART (COM36)
    XDS110 Class Auxiliary Data Port (COM37)
  Processors
  Sound, video and game controllers
  Storage controllers
  System devices
  Texas Instruments Debug Probes
    XDS110 Class Data Port
    XDS110 Class Debug Probe
  Universal Serial Bus controllers
```

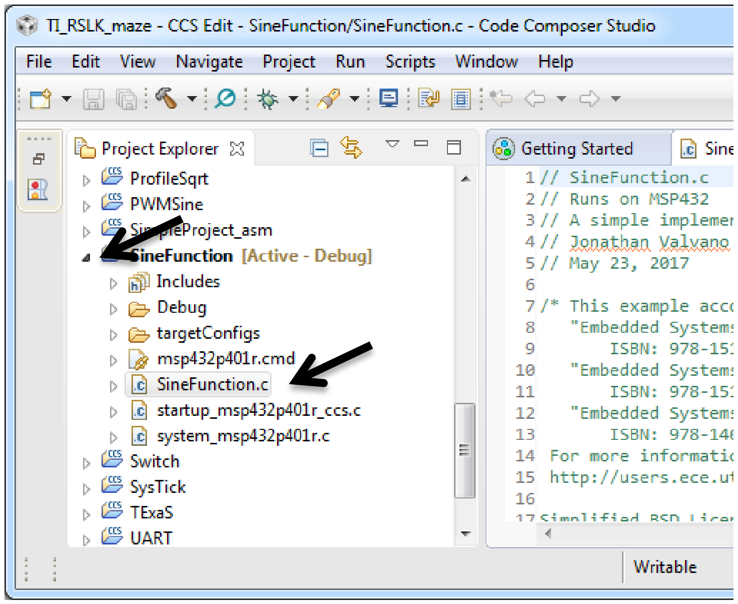
1.4.6 Run a simple example on the LaunchPad

Key Objectives

- Observe source code
- Edit-compile-link-download-debug cycle
- Step in, step over, step out
- Observing local and global variables

SineFunction is a very simple software project that performs no input/output. It calculates a **y=sin(x)** using a cubic approximation and fixed-point math. It fills an array with results. You can use the project to learn how to build (compile), debug (download and start the debugger), run, halt, and observe the array. You should also reset the processor, set a breakpoint, run until the breakpoint, and then single step (step in, step out, and step over).

1) Click on the **SineFunction** project, and open the view of the files in that project. Double click on **SineFunction.c** to see the source code.


TI_RSLK_maze - CCS Edit - SineFunction/SineFunction.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
Getting Started
Sine
Writable

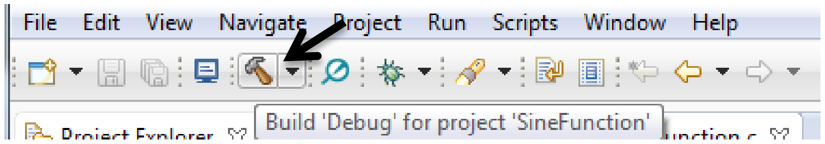
```
ProfileSqrt
PWMSine
SimpleProject_asm
SineFunction [Active - Debug]
  Includes
  Debug
  targetConfigs
  msp432p401r.cmd
  SineFunction.c
  startup_msp432p401r_ccs.c
  system_msp432p401r.c
Switch
SysTick
TExaS
UART
```

```
// SineFunction.c
// Runs on MSP432
// A simple implemen
// Jonathan Valvano
// May 23, 2017

/* This example acc
   "Embedded System
      ISBN: 978-15
   "Embedded System
      ISBN: 978-15
   "Embedded System
      ISBN: 978-14
 For more informatio
 http://users.ece.ut

 Simplified BSD Licen
```

Note: Make sure the desired project is in context (in bold) before building or debugging. Notice in the above figure, the Project Explorer bolds the project and specifies "**[Active-Debug]**"

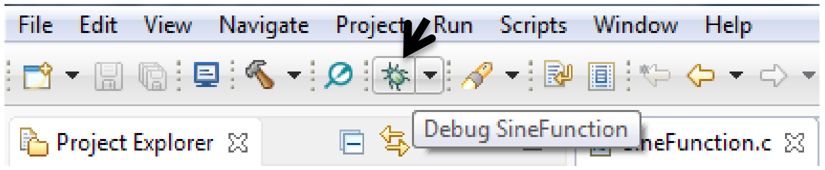
Lab: Running code on the LaunchPad using CCS

2) With the **SineFunction** project selected [Active-Debug], click Build


File Edit View Navigate Project Run Scripts Window Help
Build 'Debug' for project 'SineFunction'
Project Explorer

There should be no errors.

3) With the **SineFunction** project selected [Active-Debug], click **Debug**

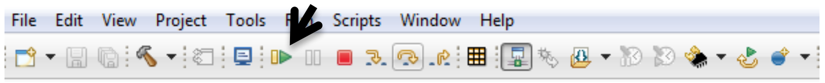


Note: When you debug on your LaunchPad for the first time it may prompt you to update the firmware. This step is recommended.

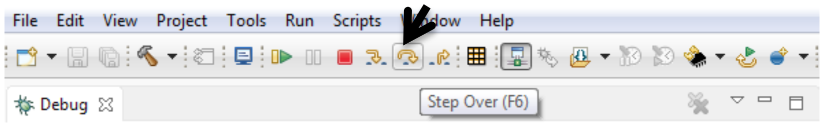
The debug operation causes several actions to be done automatically

- Prompt to save source files
- Build the project (incrementally)
- Start the debugger (CCS will switch to the CCS Debug perspective)
- Connect CCS to the target
- Load (flash) the program on the target
- Run to main

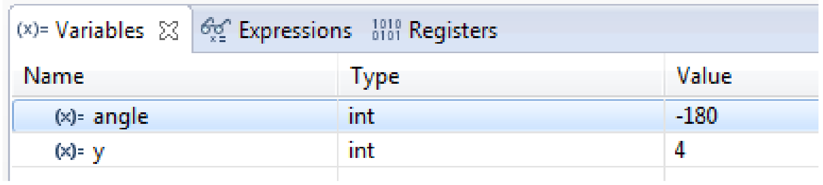
4) Once the flash is erased, and the image of this project is loaded, you will see the green triangle appear. That is the run icon, don't click run yet, but seeing this icon means the system is ready to debug



5) Single step the program by executing **Step Over** icon multiple times

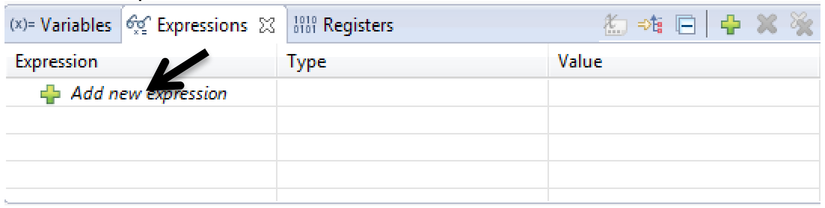


6) Observe the local variables in the **Variables** window

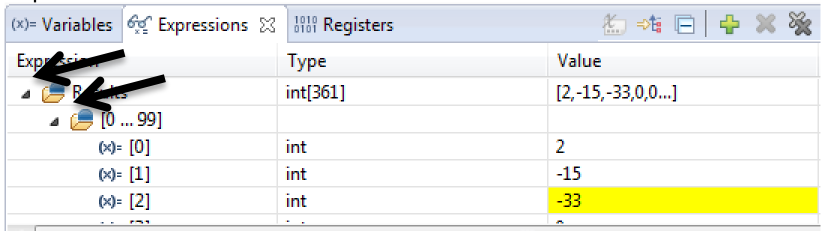


Name	Type	Value
angle	int	-180
y	int	4

7) Observe global variables in the Expressions window. Type **Results** in the "Add new expression" field and hit <enter>



Expand the Results field to see its data.

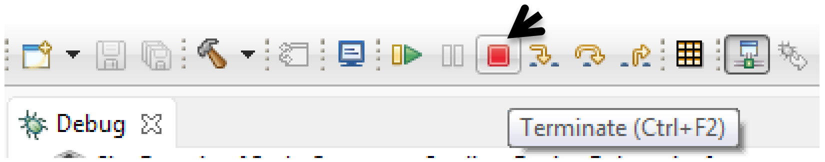


Expression	Type	Value
Results	int[361]	[2,-15,-33,0,0...]
[0 ... 99]		
[0]	int	2
[1]	int	-15
[2]	int	-33

Step over executes one line of C. If that line has a function, step over will execute the entire function. You can also experiment with **Step in** (which executes one line of C, and if that line has a function, it will step into that function. If you have stepped into a function, **Step return** will complete that function and stop at the spot the function was called.

You should also experiment the **Resume**, **Suspend**, and **Reset** commands.

8) To **halt** the debugger and terminate execution, click the Terminate icon



Lab: Running code on the LaunchPad using CCS

1.4.7 Run the Input_Output example on the LaunchPad

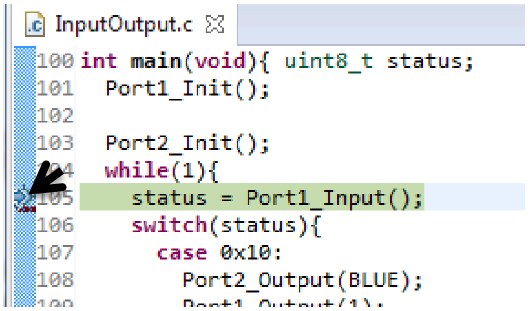
Key Objectives
- Observe I/O ports on the MSP432
- Interact with hardware on the LaunchPad
- Setting and clearing breakpoints

Input_Output is a simple project that showcases some the features of the LaunchPad. For example, it will input from the two switches on the LaunchPad and output to the LED. Follow the same steps 1, 2, 3, and 4 as you did to compile and load this project onto MSP432 LaunchPad.

1) Run the project and interact with the two switches on the LaunchPad. You should observe this simple behavior

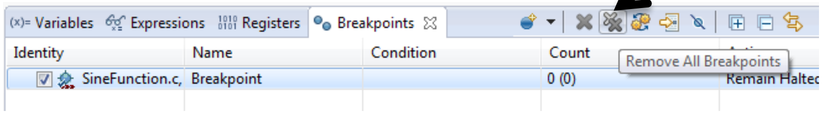
No switches	No LEDs on
Just SW1	Red LED is on, color LED is blue
Just SW2	Red LED is on, color LED is red
Both SW1,SW2	Red LED is on, color LED is blue+red=purple

2) Set a breakpoint at the line **status = Port1_Input();** To place a **breakpoint**, click on a line at which you want it to stop, right click and add hardware breakpoint. When you start the program it will run to the breakpoint and stop. The following figure shows the debugger halted at the breakpoint.


InputOutput.c

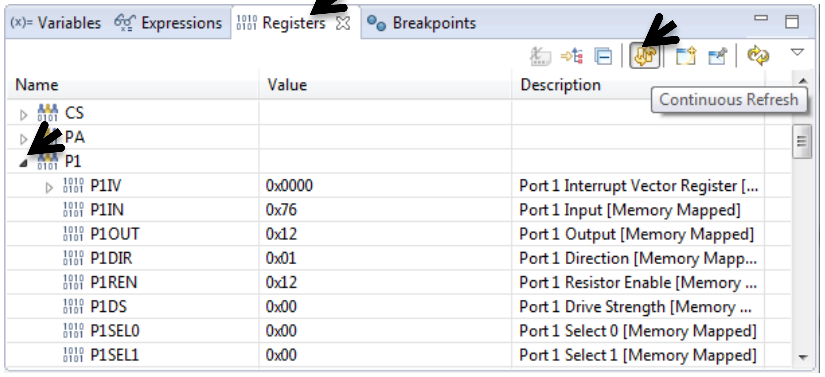
```
int main(void){ uint8_t status;
  Port1_Init();

  Port2_Init();
  while(1){
    status = Port1_Input();
    switch(status){
      case 0x10:
        Port2_Output(BLUE);
```

Remove all breakpoints by clicking the icon in the breakpoint window



Identity	Name	Condition	Count	Action
SineFunction.c,	Breakpoint		0 (0)	Remain Halted

3) Observe the I/O Port registers. First select the **Registers** tab, then select **P1** (I/O Port 1). Activate the Continuously Refresh mode. Run the program and touch the two switches on the LaunchPad. You will see the port input data in the P1IN field.



Name	Value	Description
CS		
PA		
P1		
P1IV	0x0000	Port 1 Interrupt Vector Register [...
P1IN	0x76	Port 1 Input [Memory Mapped]
P1OUT	0x12	Port 1 Output [Memory Mapped]
P1DIR	0x01	Port 1 Direction [Memory Mapp...
P1REN	0x12	Port 1 Resistor Enable [Memory ...
P1DS	0x00	Port 1 Drive Strength [Memory ...
P1SEL0	0x00	Port 1 Select 0 [Memory Mapped]
P1SEL1	0x00	Port 1 Select 1 [Memory Mapped]

1.4.8 Run the TExaSdisplay logic analyzer

Key Objectives
- Introduce TExaSdisplay in logic analyzer mode
- Observe digital signals on the LaunchPad

TExaSdisplay is a Windows application that does not require a separate installation. You should see the **TExaSdisplay.exe** executable within the **tirslk_maze_1_00_00** folder. To start the application, you simply double-click **TExaSdisplay.exe** executable file.

If you have access to a real logic analyzer, you should use it, and therefore can skip this section. If you do not have access to a real logic analyzer, then TExaS provides a no-cost, simple option. TExaS has these specifications:
- Up to 7 digital channels
- 10 kHz sampling (you can adjust the display but sampling is fixed)
- Runs in background alongside your software
- Data streamed through USB cable from MSP432 to PC

The first step is to activate the TExaS project, and open the **TExaSmain.c** file. There are three main programs in this project. Edit the **LogicAnalyzerMain** function so it is called main, and edit the other main to be Lab2main. Notice the MSP432 will be running at 48 MHz and the logic analyzer is configured to display Port 1. When it is running the seven bits of P1.6 – P1.0 will be streamed to the PC at 10 kHz. The logic analyzer works whether the pin is an input or output. In this example, P1.0 is an output (to the red LED) and P1.1/P1.4 are inputs from the two LaunchPad switches. We will talk about I/O in great detail in subsequent chapters, but for now let's focus on how the logic analyzer measures P1.4, P1.1, P1.0 by sending the digital information from the MSP432 to the PC via the USB cable.

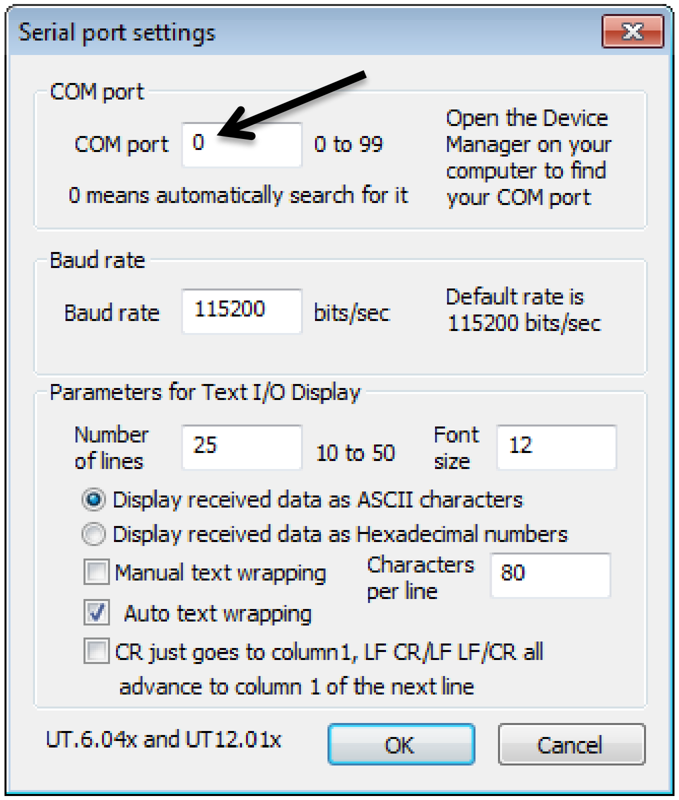
```
int LogicAnalyzerMain(void){
uint32_t status,delay,data;
  Clock_Init48MHz(); // makes bus clock 48 MHz
  LaunchPad_Init();  // use buttons to step through frequencies
  TExaS_Init(LOGICANALYZER_P1);
  data = 0;
  while(1){
    status = LaunchPad_Input();
    switch(status){    // negative logic on P1.1 and P1.4
      case 0x00: delay=1000; break;  // neither switch pressed
      case 0x01: delay=2000; break;  // SW2 pressed
      case 0x02: delay=3000; break;  // SW1 pressed
      case 0x03: delay=4000; break;  // both switches pressed
    }
    Clock_Delay1us(delay);
    data = data ^0x01;
    LaunchPad_LED(data); // toggle red LED
  }
}
```

You can see the various options for the logic analyzer by looking in the **TExaS.h** header file. These are the choices you have when configuring the TExaS.

```
enum TExaSmode{
  SCOPE,            //8-bit oscilloscope on J3.26/P4.4/A9
  LOGICANALYZER,    //7-bit logic analyzer
  LOGICANALYZER_P1, //  7-bit logic analyzer on P1.6-P1.0
  LOGICANALYZER_P2, //  7-bit logic analyzer on P2.6-P2.0
  LOGICANALYZER_P3, //  7-bit logic analyzer on P3.6-P3.0
  LOGICANALYZER_P4, //  7-bit logic analyzer on P4.6-P4.0
  LOGICANALYZER_P5, //  7-bit logic analyzer on P5.6-P5.0
  LOGICANALYZER_P6, //  7-bit logic analyzer on P6.6-P6.0
  LOGICANALYZER_P7, //  7-bit logic analyzer on P7.6-P7.0
  LOGICANALYZER_P8, //  7-bit logic analyzer on P8.6-P8.0
  LOGICANALYZER_P9, //  7-bit logic analyzer on P9.6-P9.0
  LOGICANALYZER_P10, //  7-bit logic analyzer on P10.6-P10.0
  LOGICANALYZER_P4_765432, //  6-bit logic analyzer on P4.7-P4.2
  LOGICANALYZER_P4_765320, //  6-bit logic analyzer on P4.7-5,3-2,0
  LOGICANALYZER_P2_7654  //  4-bit logic analyzer on P2.7-P2.4
};
```

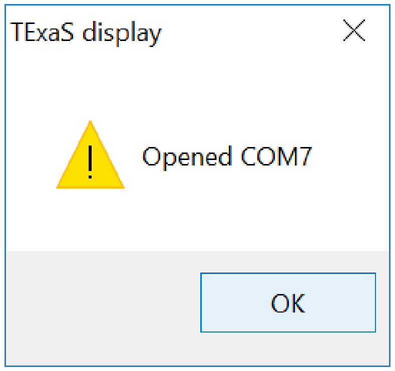
Build (compile), debug (erase flash, program flash with object code), and run the project. The red LED flashes, and you can change the rate of flashing by pushing the two switches.

Start **TExaSdisplay** and execute COM->Settings. You can enter the COM port number (which you can find from your device manager), or you can leave the field at "0", which means start at 1 and search for a COM port that will open. The baud rate is always 115200 bits/sec in this class, but for other situations you might need to set the baud rate. The other parameters in this dialog configure the look and feel of the text window, when using **TExaSdisplay** as a terminal application.


Serial port settings
COM port
COM port: 0
0 to 99
Open the Device Manager on your computer to find your COM port
0 means automatically search for it
Baud rate
Baud rate: 115200 bits/sec
Default rate is 115200 bits/sec
Parameters for Text I/O Display
Number of lines: 25
10 to 50
Font size: 12
[✓ Display received data as ASCII characters]
Display received data as Hexadecimal numbers
Manual text wrapping
Characters per line: 80
[✓ Auto text wrapping]
CR just goes to column1, LF CR/LF LF/CR all advance to column 1 of the next line
UT.6.04x and UT12.01x
OK
Cancel

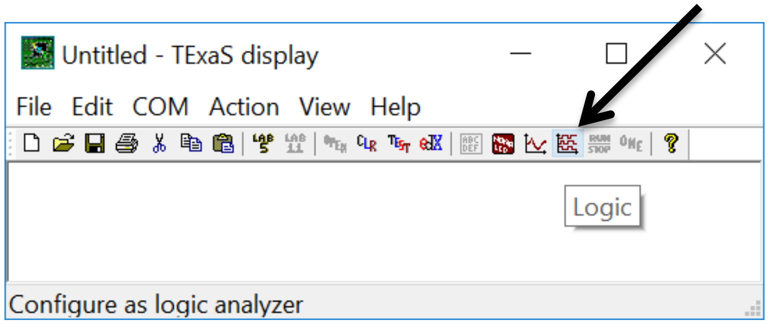
To connect **TExaSdisplay** with the MSP432 serial port, you click the **Open** tool button or execute the command **COM -> Open Port** (F4). On this computer, the MSP432 LaunchPad was found as COM7.

Lab: Running code on the LaunchPad using CCS



Opened COM7

To run **TExaSdisplay** in logic analyzer mode, you click the Logic Analyzer tool button, or execute **View -> Logic Analyzer**



Configure as logic analyzer

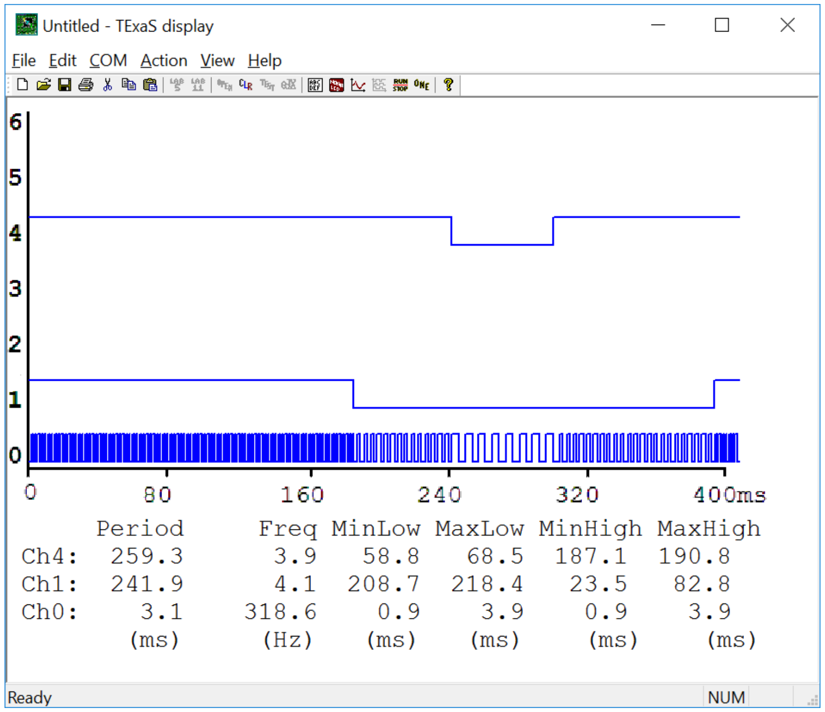
The logic analyzer always sends 7 bits at 10 kHz, but you can choose which ones to plot on the display. Execute **View -> Logic Analyzer Configuration** and disable pins 6,5,3,2 (because they have no value in this example). Specify a rising edge trigger in bit 0. The analyzer takes multiple readings and lines up the traces using the trigger. Using a trigger means one of the traces will not jitter around if the data are rapidly changing.

An **edge trigger** means the analyzer will search the incoming data stream for an edge on that pin, plotting the edge at a fixed place on the screen. The analyzer takes multiple readings. With a rising edge trigger active the rising edge is placed at the same location of the display. Using a trigger means one of the traces will not jitter around if the data are rapidly changing.

When your MSP432 program is running, you will be able to see digital data versus time. There are four commands to control the display

View->Slower F6 will increase the range of times displayed
View->Faster F7 will decrease the range of times displayed
View->Pause/Run F8 will stop/start the display
View->Single F9 will display one sweep and stop

You will see the three pins (SW1, SW2, and LED) plotted versus time. Push the two switches to observe the behavior that the switches affect the frequency of the oscillations on P1.0.



```
        Period     Freq MinLow MaxLow MinHigh MaxHigh
Ch4:    259.3      3.9   58.8   68.5  187.1   190.8
Ch1:    241.9      4.1  208.7  218.4   23.5    82.8
Ch0:      3.1    318.6    0.9    3.9    0.9     3.9
         (ms)     (Hz)   (ms)   (ms)   (ms)    (ms)
```

Lab: Running code on the LaunchPad using CCS

1.5 Troubleshooting

A project doesn't compile:

- Try a different project. All the projects in **tirslk_maze** should compile. If none of the **tirslk_maze** compile, then try reinstalling CCS and **tirslk_maze**.
- If other projects in **tirslk_maze** compile, but a project you have edited does not compile, it is possible you have introduced errors. Follow the error codes in the **problems** window. Remember to start with a project that compiles, make only a few changes, and then compile it. This way when it doesn't compile, there are only a few places to look for the error.

The debug command can't erase/download/run:

- Make sure the build step occurred without error.
- Check the device manager to make sure the proper drivers are installed for the LaunchPad board.
- Make sure the LaunchPad power is connected to the PC.
- Try another USB port.
- Try another micro USB cable

1.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab.

- What are components of a project on CCS?
- What are the steps involved in software design/test?
- What are breakpoints? How do I set them up? How do I use breakpoints to debug?
- What does it mean to step in, step over, and step out?
- What are some of the ways to observe intermediate results during software debugging?
- What is a logic analyzer? What is an oscilloscope?

1.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. Additional challenges are not required to complete the course.

- Run the UART example (Appendix A2)
- Run the TExaS oscilloscope example (Appendix A3)
- Load a project with resource explorer (Appendix A4)
- Run an energy trace on a system (Appendix A6)

1.8 Which modules are next?

Now that we have started, there are two paths forward. The hardware path involves learning about electronics and building the robot:

Module 2 - Study voltage current power and the batteries
Module 5 - Robot construction, including battery and voltage regulation
Module 12 - Interfacing the motors and wheels

The software path involves developing programming and debugging skills:

Module 3 - Introduce the Cortex M processor
Module 4 - Introduce the process of software design
Module 6 - Learn how to input and output on the pins of the microcontroller
Module 7 - Study finite state machines as a method to control the robot

1.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module:

- Install CCS
- Import projects for the **tirslk_maze** curriculum
- Compile and run a program on the MSP432 LaunchPad
- Use of Debug mode in CCS