

ECE 3567 Microcontroller Lab

Lecture #3 – The Microcontroller & Code Composer Studio v 9.1.0



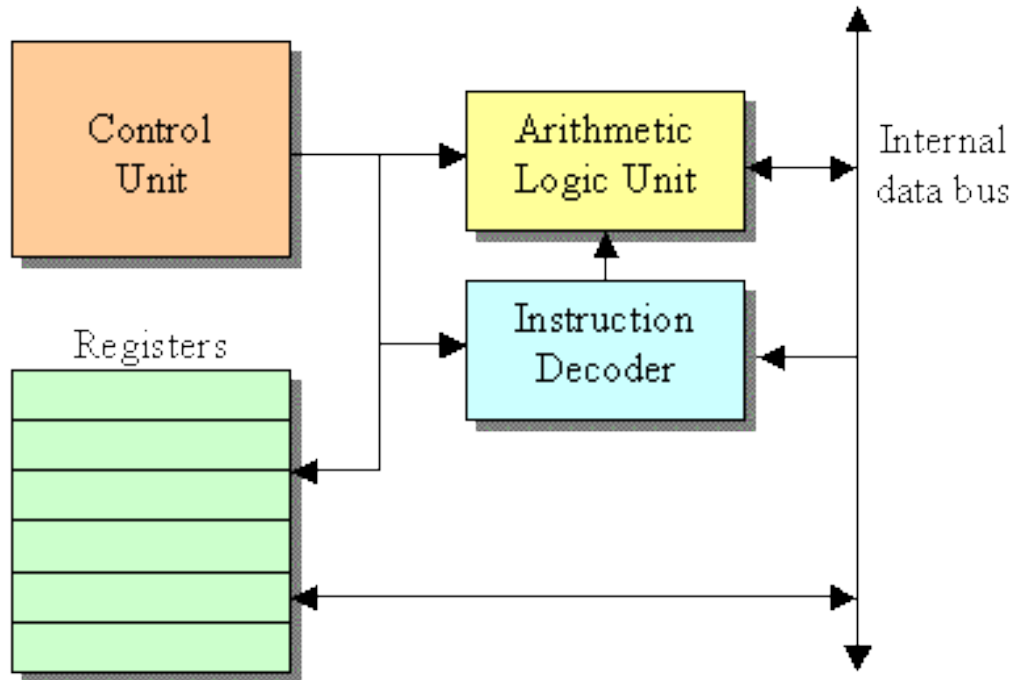
Spring 2020
Dr. Gregg Chapman

- *Microcontroller*
- *Registers Types*
- *I/O Ports*
- *Code Composer Studio v 9.1.0*

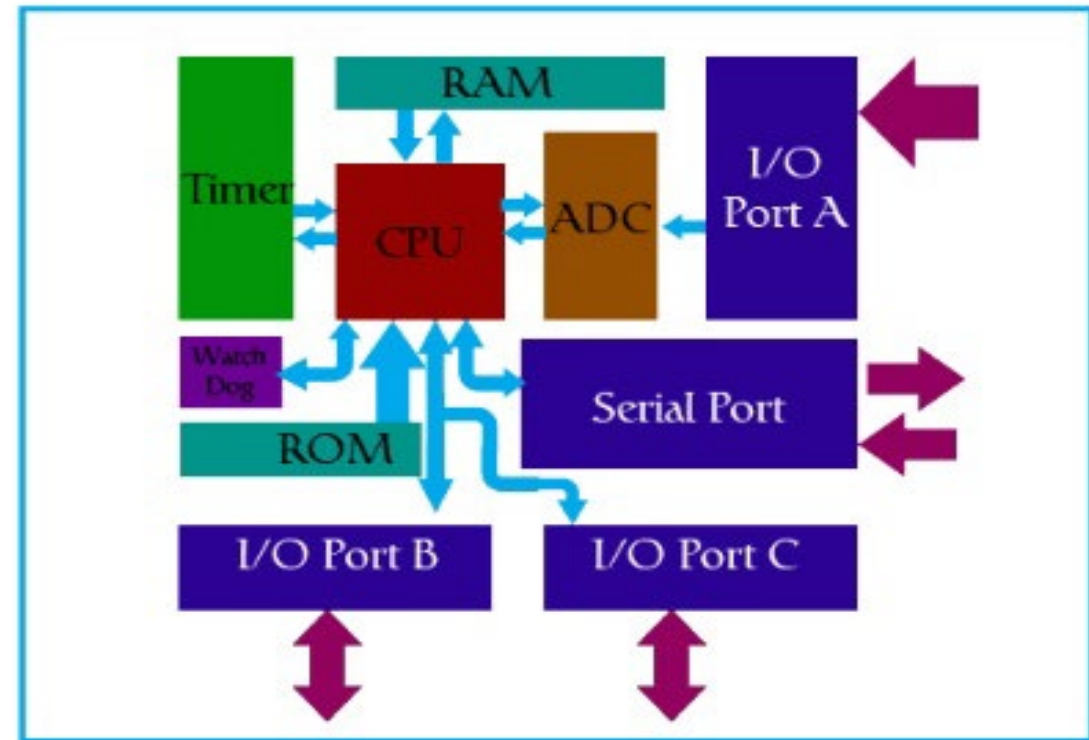
The Microcontroller

ECE 3567 Microcontrollers

Embedded Hardware



Microprocessor (μP)

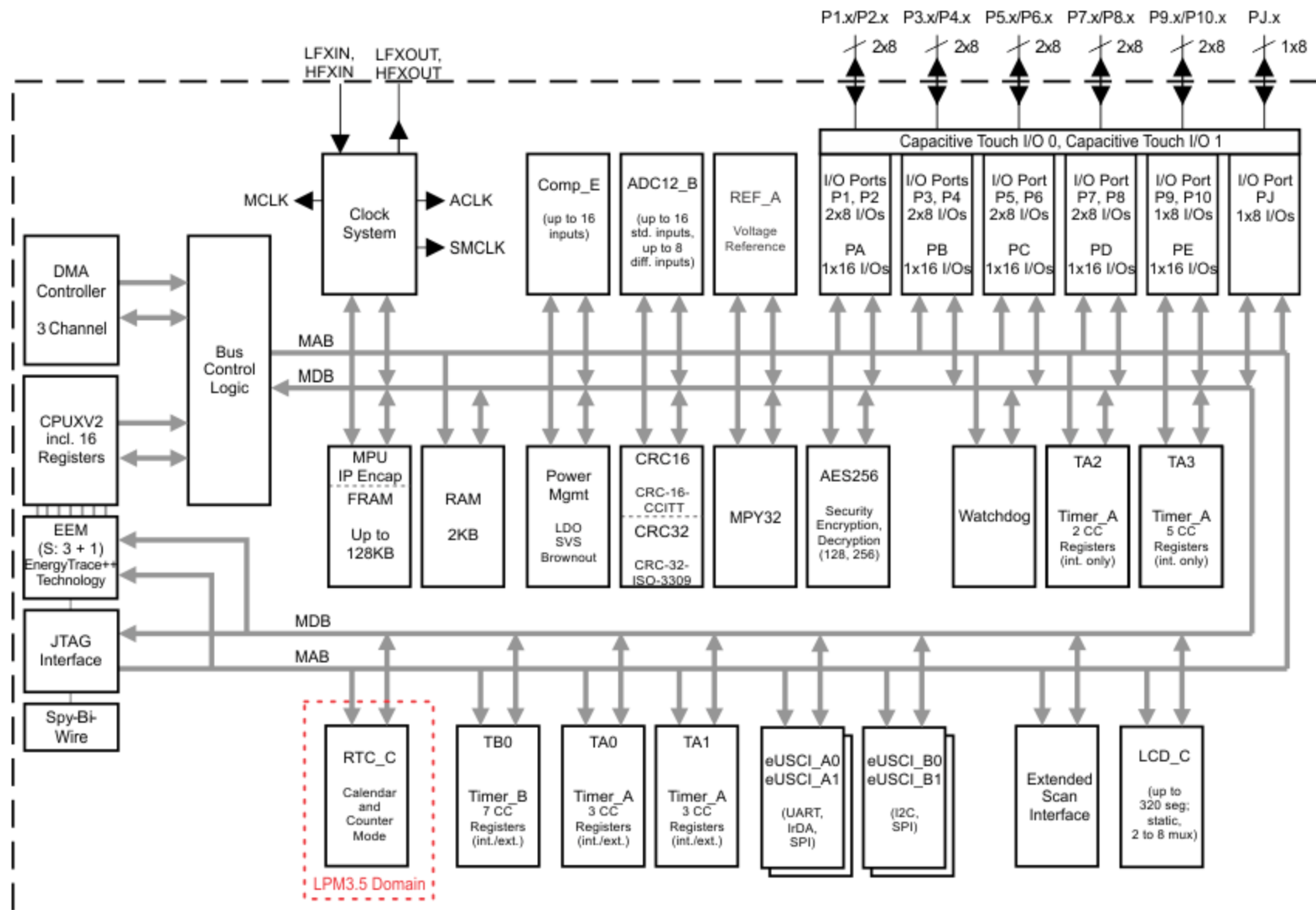


Micro-Controller (MCU)

The MSP430FR6989 Microcontroller - Overview

- Embedded Microcontroller
 - 16-Bit RISC Architecture up to 16-MHz Clock
 - Wide Supply Voltage Range From 3.6 V Down to 1.8 V (Minimum Supply Voltage is Restricted by SVS Levels, See the [SVS Specifications](#))
- Optimized Ultra-Low-Power Modes
 - Active Mode: Approximately 100 μ A/MHz
 - Standby (LPM3 With VLO): 0.4 μ A (Typical)
 - Real-Time Clock (RTC) (LPM3.5): 0.35 μ A (Typical) ⁽¹⁾
 - Shutdown (LPM4.5): 0.02 μ A (Typical)
- Ultra-Low-Power Ferroelectric RAM (FRAM)
 - Up to 128KB of Nonvolatile Memory
 - Ultra-Low-Power Writes
 - Fast Write at 125 ns per Word (64KB in 4 ms)
 - Unified Memory = Program + Data + Storage in One Single Space
 - 10^{15} Write Cycle Endurance
 - Radiation Resistant and Nonmagnetic
- Intelligent Digital Peripherals
 - 32-Bit Hardware Multiplier (MPY)
 - Three-Channel Internal Direct Memory Access (DMA)
 - RTC With Calendar and Alarm Functions
 - Five 16-Bit Timers With up to 7 Capture/Compare Registers Each
 - 16-Bit and 32-Bit Cyclic Redundancy Checker (CRC16, CRC32)
- High-Performance Analog
 - Extended Scan Interface (ESI) for Background Water, Heat, and Gas Volume Measurement
 - 16-Channel Analog Comparator
 - 12-Bit Analog-to-Digital Converter (ADC) With Internal Reference and Sample-and-Hold and up to 16 External Input Channels
 - Integrated LCD Driver With Contrast Control for up to 320 Segments
- Multifunction Input/Output Ports
 - All P1 to P10 and PJ Pins Support Capacitive Touch Capability Without Need for External Components
 - Accessible Bit-, Byte- and Word-Wise (in Pairs)
 - Edge-Selectable Wakeup From LPM on Ports P1, P2, P3, and P4
 - Programmable Pullup and Pulldown on All Ports
- Code Security and Encryption
 - 128-Bit or 256-Bit AES Security Encryption and Decryption Coprocessor
 - True Random Number Seed for Random Number Generation Algorithm
- Enhanced Serial Communication
 - eUSCI_A0 and eUSCI_A1 Support:
 - UART With Automatic Baud-Rate Detection
 - IrDA Encode and Decode
 - SPI
 - eUSCI_B0 and eUSCI_B1 Support:
 - I²C With Multiple-Slave Addressing
 - SPI
 - Hardware UART and I²C Bootloader (BSL)
- Flexible Clock System
 - Fixed-Frequency DCO With 10 Selectable Factory-Trimmed Frequencies
 - Low-Power Low-Frequency Internal Clock Source (VLO)
 - 32-kHz Crystals (LFXT)
 - High-Frequency Crystals (HFXT)
- Development Tools and Software
 - Free Professional Development Environments With EnergyTrace++™ Technology
 - Experimenter and Development Kits

Functional Diagram



MCU Statistics

- Number of Bits
- Processor Speed
 - Clock Speed
 - MIPS
- Instruction Set
 - RISC
 - ARM
- Intended Application
- Peripheral Included
 - ADC
 - DAC
 - UART/I2C/SCI/SPI/CAN
 - Analog Comparators
 - Incorporated Sensors
 - RAM
 - DMA
 - Watchdog Timer
 - Low Power Module
 - ALU/DSP
 - Timers
 - Clock Module
 - I/O Ports
 - CRC
 - Encryption

Instruction Sets

What is RISC?

- **RISC?**

RISC, or *Reduced Instruction Set Computer*, is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures.

- **History**

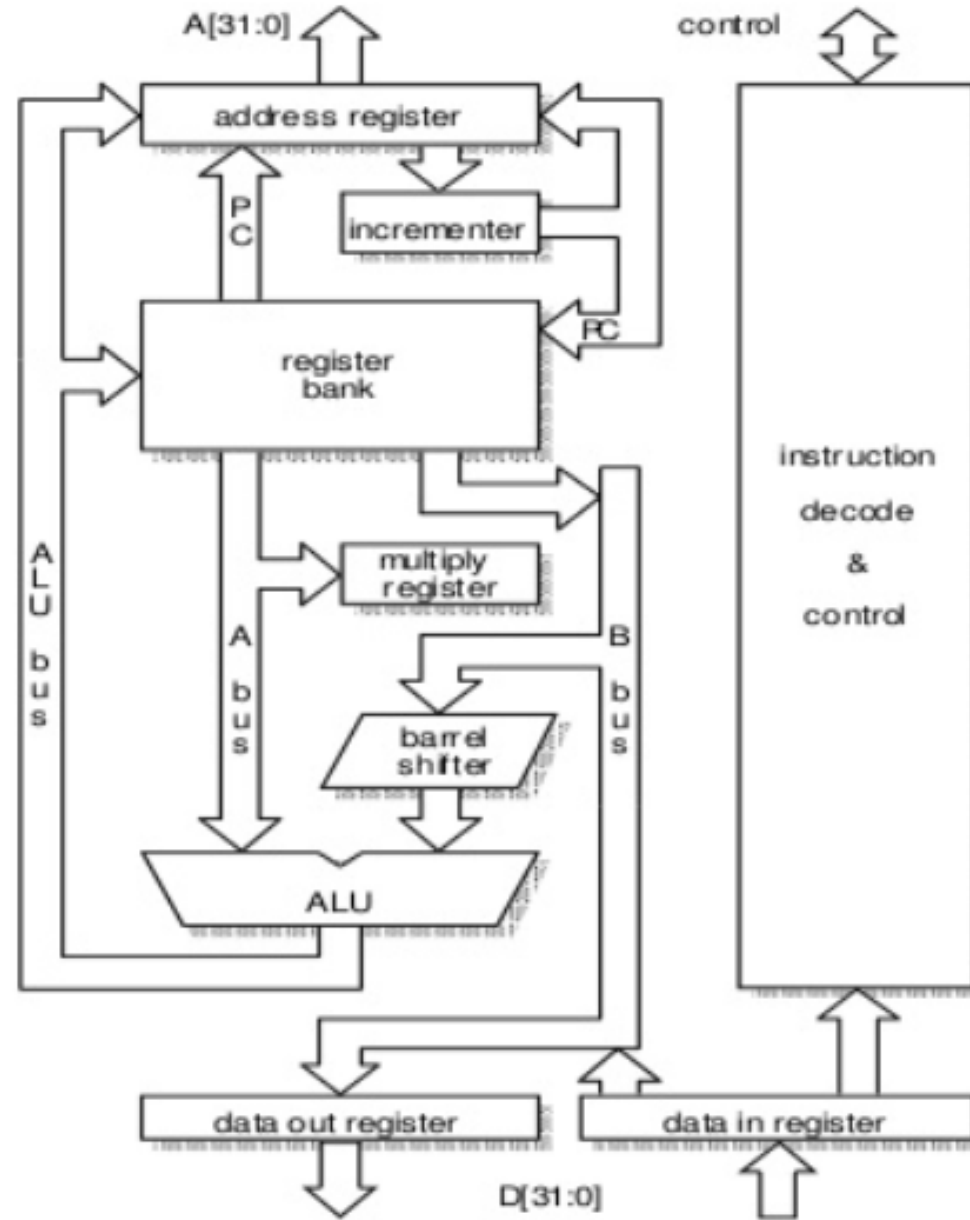
The first RISC projects came from IBM, Stanford, and UC-Berkeley in the late 70s and early 80s. The IBM 801, Stanford MIPS, and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC. Certain design features have been characteristic of most RISC processors:

- *one cycle execution time*: RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU and a technique called PIPELINING
- *pipelining*: a technique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process instructions;
- *large number of registers*: the RISC design philosophy generally incorporates a larger number of registers to prevent in large amounts of interactions with memory

ARM Processor (Advanced RISC Machine)

- A 32-bit Processor Core that incorporates
 - Reduced Instruction Set Computing
 - Instruction Set Architecture

Instruction Set Architecture (ISA)



Register Types

Registers

- Control Registers (xxx**CTLx**)– Module level. Use to configure functions.
- Count Registers (xxx**R**) – Up or down counter
- Capture/Compare Registers (xxx**CCR**) – Work in conjunction with Counters to take action at a certain count.
- Capture/Compare Control Registers (xxx**CCTLx**)– Used to configure what happens when the CCR matches R (counter).
- I/O Registers (Ports) – Input/Output ports can have 1 of 4 functions.

Control Registers

- Every MODULE in a Microcontroller has one or more CONTROL REGISTERS to configure the functions of the module.
- Each CONTROL REGISTER is divided into FIELDS
- Each FIELD sets one PARAMETER of the MODULE to a specific function
- The number of options for the PARAMETER determines the number of BITS in the FIELD
- Each BIT has a Power-up DEFAULT value, usually 0.

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID	MC		Reserved		TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Control Registers

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit clears TAR, the clock divider logic (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Microcontroller Registers

Count Registers

25.3.2 TAxR Register

Timer_Ax Counter Register

Figure 25-17. TAxR Register

15	14	13	12	11	10	9	8
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 25-5. TAxR Register Description

Bit	Field	Type	Reset	Description
15-0	TAxR	RW	0h	Timer_A register. The TAxR register is the count of Timer_A.

Microcontroller Registers

Capture / Compare Registers

Timer_A Capture/Compare n Register

Figure 25-19. TAxCCRn Register

15	14	13	12	11	10	9	8
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 25-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	<p>Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR.</p> <p>Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed.</p>

Capture / Compare Control Registers

25.3.3 TAxCTLn Register

Timer_Ax Capture/Compare Control n Register

Figure 25-18. TAxCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections. 00b = CC1xA 01b = CC1xB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read from this bit.
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode

7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Timer A Example:

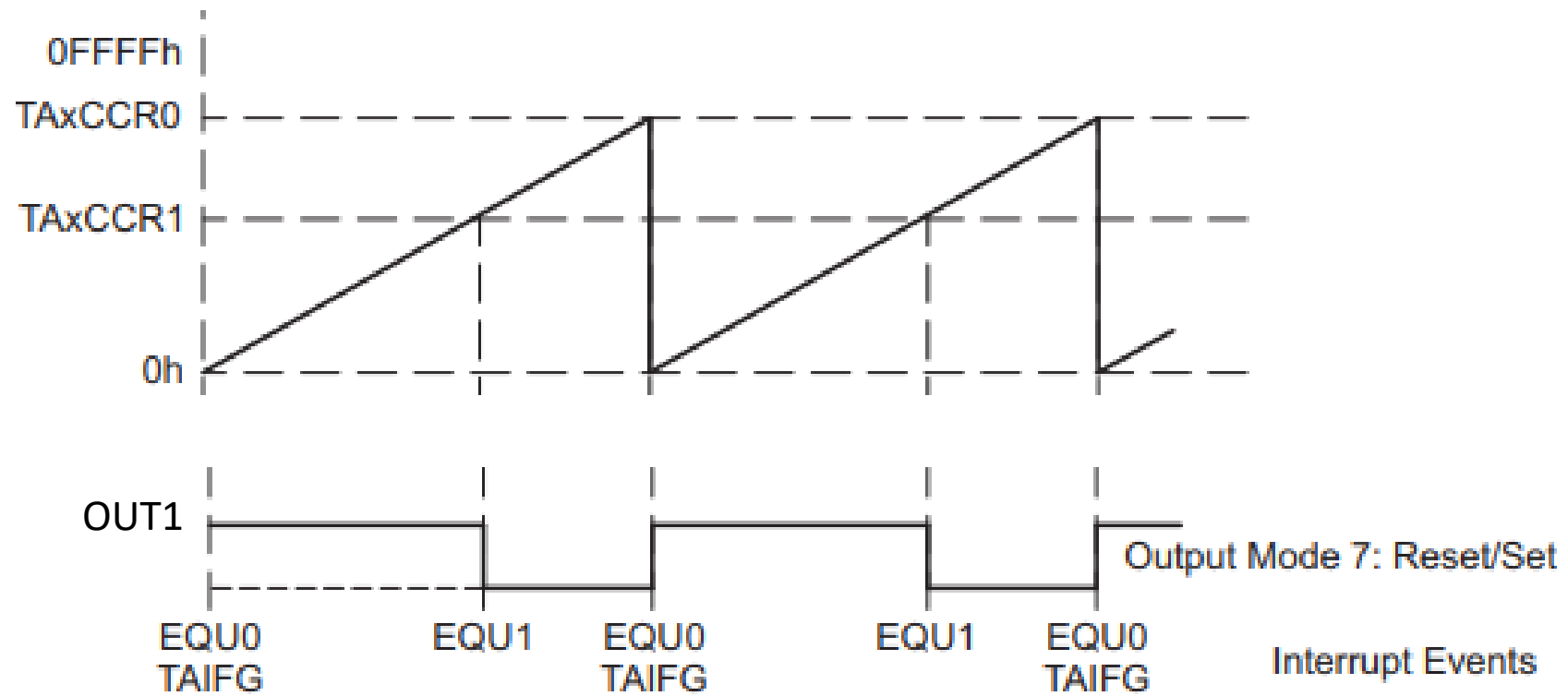


Figure 25-12. Output Example – Timer in Up Mode

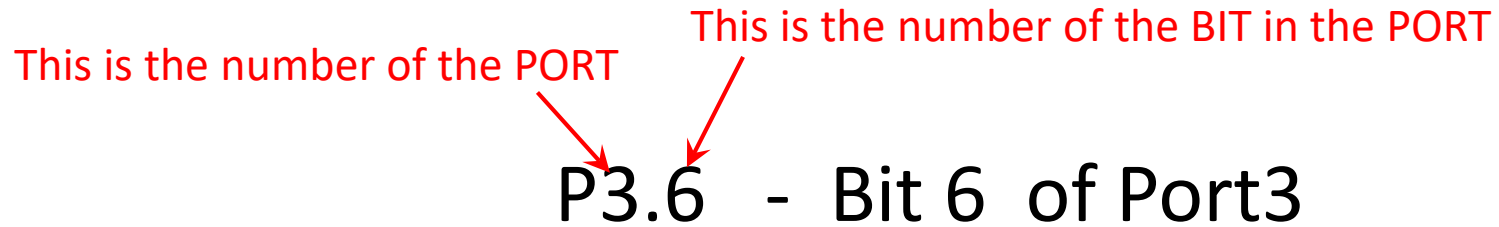
I/O Ports

Section 12.4 in the Users Manual

I/O Ports Port Number and Bit Number TI Conventions

This is the number of the PORT This is the number of the BIT in the PORT

P3.6 - Bit 6 of Port3



Registers Associated with each I/O Port

- Input Registers (PxIN) - Read from this one
- Output Registers (PxOUT) – Write to this one
- Direction Registers (PxDIR) – Control direction of individual bits.
- Pullup or Pulldown Resistor Enable Registers (PxREN) – Also bit by bit
- **Function Select Registers (PxSEL0, PxSEL1)**
- Interrupt Settings (PxIFG, PxIES, PxIE)

NOTE: x – substitute the number of the PORT

- 1-10 and J, (8 bit)
- A,B,C,D, and E (16 bit)

I/O Ports

12.2.1 *Input Registers (PxIN)*

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function. These registers are read only.

- Bit = 0: Input is low
- Bit = 1: Input is high

NOTE: Writing to read-only registers PxIN

Writing to these read-only registers results in increased current consumption while the write attempt is active.

I/O Ports

12.2.2 *Output Registers (PxOUT)*

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction.

- Bit = 0: Output is low
- Bit = 1: Output is high

If the pin is configured as I/O function, input direction and the pullup or pulldown resistor are enabled; the corresponding bit in the PxOUT register selects pullup or pulldown.

- Bit = 0: Pin is pulled down
- Bit = 1: Pin is pulled up

I/O Ports

12.2.3 *Direction Registers (PxDIR)*

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other functions must be set as required by the other function.

- Bit = 0: Port pin is switched to input direction
- Bit = 1: Port pin is switched to output direction

I/O Ports

12.2.4 Pullup or Pulldown Resistor Enable Registers (PxREN)

Each bit in each PxREN register enables or disables the pullup or pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin contains a pullup or pulldown.

- Bit = 0: Pullup or pulldown resistor disabled
- Bit = 1: Pullup or pulldown resistor enabled

Table 12-1 summarizes the use of PxDIR, PxREN, and PxOUT for proper I/O configuration.

Table 12-1. I/O Configuration

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

I/O Ports

12.2.5 Function Select Registers (*PxSEL0*, *PxSEL1*)

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each port pin uses two bits to select the pin function – I/O port or one of the three possible peripheral module function. [Table 12-2](#) shows how to select the various module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function – I/O port or peripheral module function.

Table 12-2. I/O Function Selection

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

I/O Ports

12.2.6 *Port Interrupts*

At least each pin in ports P1 and P2 have interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. Some devices may contain additional port interrupts besides P1 and P2. See the device-specific data sheet to determine which port interrupts are available.

All Px interrupt flags are prioritized, with PxIFG.0 being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the PxIV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Px interrupts do not affect the PxIV value. The PxIV registers are word or byte access.

Each PxIFG bit is the interrupt flag for its corresponding I/O pin, and the flag is set when the selected input signal edge occurs at the pin. All PxIFG interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Software can also set each PxIFG flag, providing a way to generate a software-initiated interrupt.

- Bit = 0: No interrupt is pending
 - Bit = 1: An interrupt is pending
-
- Individually configurable P1 and P2 interrupts.
(and possibly P3 and P4)

I/O Ports – Putting It All Together

Suppose that you wanted to configure BIT 2 of PORT 2 as a Timer B0.4 output for a PWM application

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2OUT	X	X	X	X	X	OUT	X	X
P2DIR	X	X	X	X	X	1	X	X
P2SEL1	X	X	X	X	X	1	X	X
P2SEL0	X	X	X	X	X	0	X	X
P2REN	X	X	X	X	X	X	X	X

Table 12-1. I/O Configuration

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

Table 12-2. I/O Function Selection

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

The Microcontroller

Code Composer Studio v 9.1.0



Project Set-up

Before doing ANYTHING, make a folder on the **U: Drive** that will be your Workspace.

Never place more than one project in the same workspace.

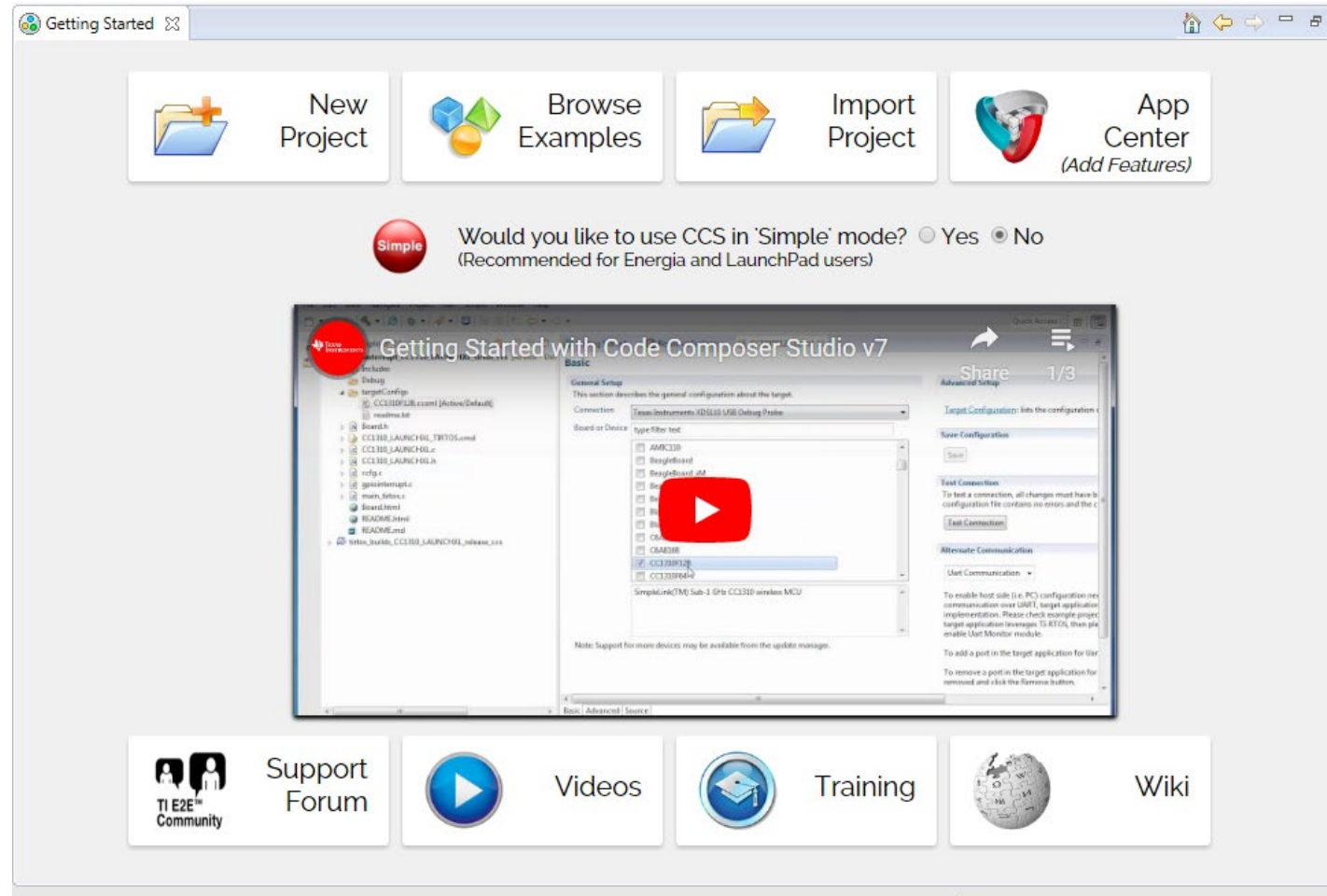
Project Set-up

Invoke the Code Composer Studio version 9.1.0 software by double clicking on the CCS ICON



Project Set-up

If you are beginning with the Getting Started screen, select **New Project**:



If only Project Explorer is open, select **View → Getting Started**

Project Set-up

New CCS Project

CCS Project

Create a new CCS Project.

Target: MSP430FRxxx Family MSP430FR6989

Connection: TI MSP430 USB1 [Default] Identify...

MSP430

Project name: Lab2

☐ Use default location

Location: U:\ECE3567\Lab2 Browse...

Compiler version: TI v16.9.3.LTS More...

Advanced settings

Project templates and examples

type filter text

- Empty Assembly-only Project
- Empty RTSC Project
- Basic Examples
 - Blink The LED
 - Hello World
- FRAM Utilities
- MSP430 DriverLib
 - Empty Project with DriverLib Source

Initial starting point for using MSP430 DriverLib.

Copies DriverLib sources into your project and adds the appropriate include path. Everything you need to get started using DriverLib in a new project.

Open [Resource Explorer](#) to browse a wide selection of example projects...

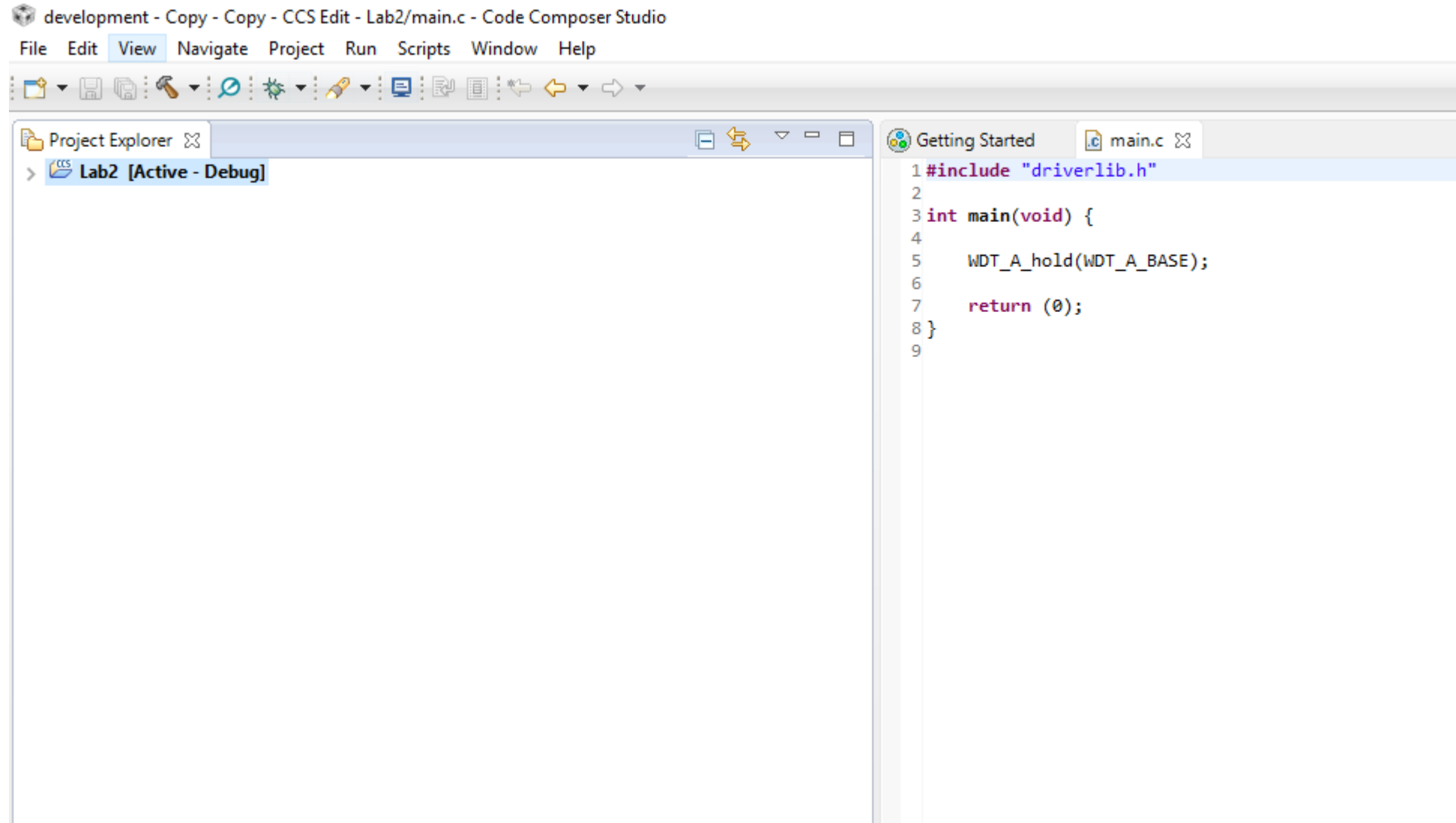
Finish

In the **CCS Project** window, configure the following fields:

1. Target: **MSP430FRxxx** Family
2. **MSP430FR6989**
3. Connection: Leave at Default
4. Project Name (e.g. Lab 2)
5. Uncheck use default location and browse to your workspace (e.g. U:\ECE3567\Lab2)
6. Leave Compiler at Default
7. In Project templates and examples, you must **SCROLL DOWN** to **MSP430 Driverlib**, select the down arrow v and **SCROLL DOWN AGAIN**. Highlight:
Empty Project with DriverLib Source
8. Select Finish

Project Set-up

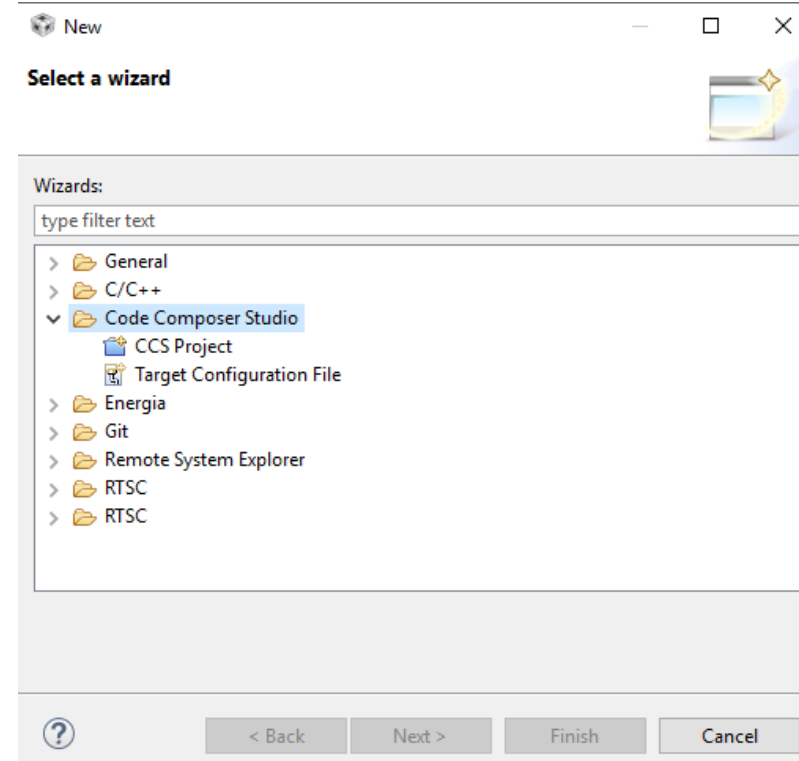
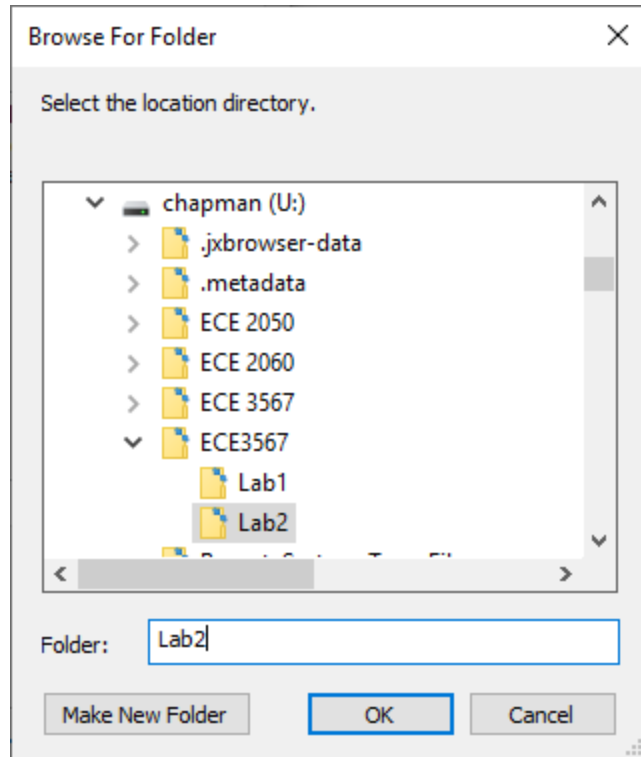
"Voilà"



Project Set-up

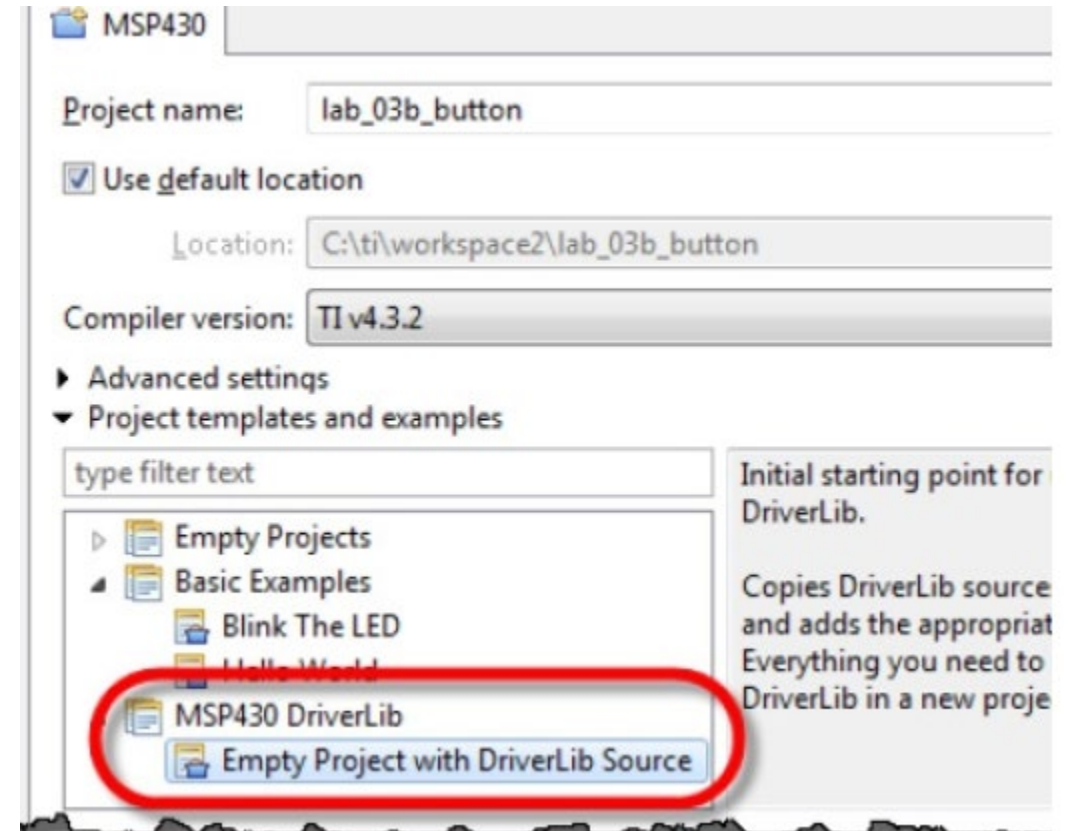
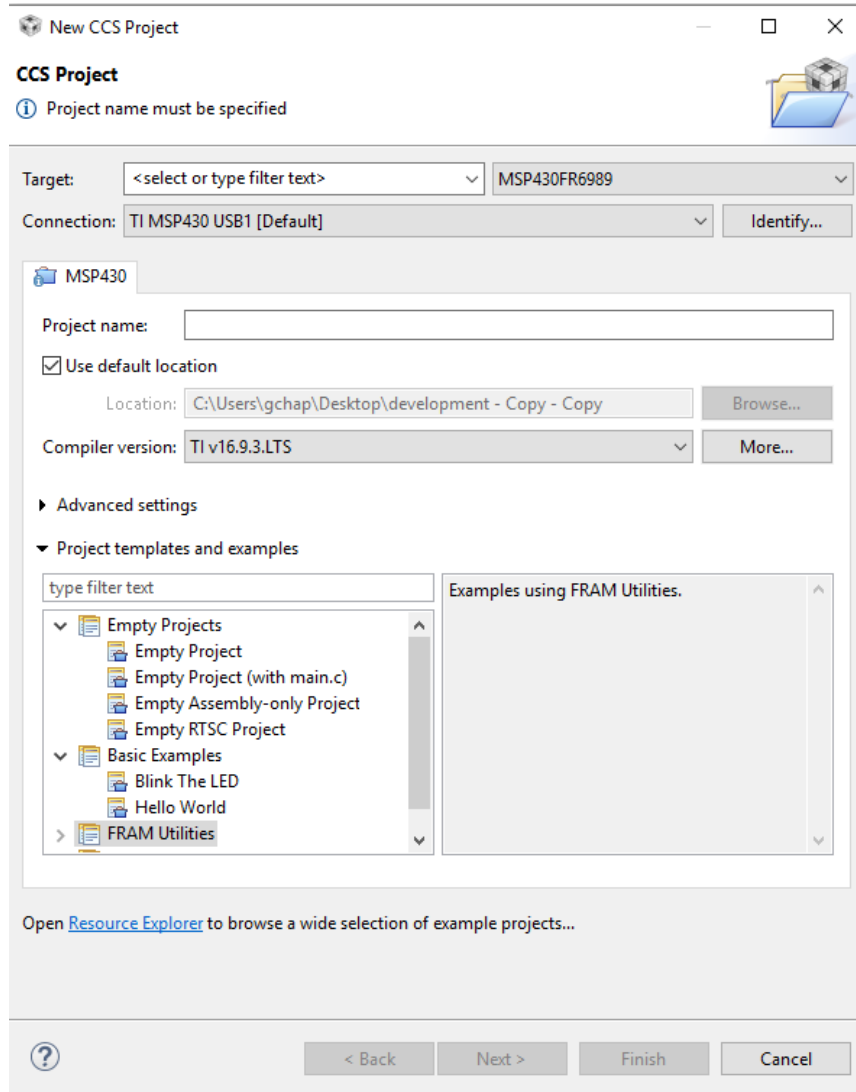
ALTERNATIVELY: If you are beginning with another project already open in Project Explorer:

1. Select **File** → **Switch Workspace** and navigate to your new workspace
2. Select **File** → **New** → **Project** → **Code Composer Studio** → **CCS Project**



Project Set-up

3. This should open the **New CSS Project** window. Proceed as previously described.

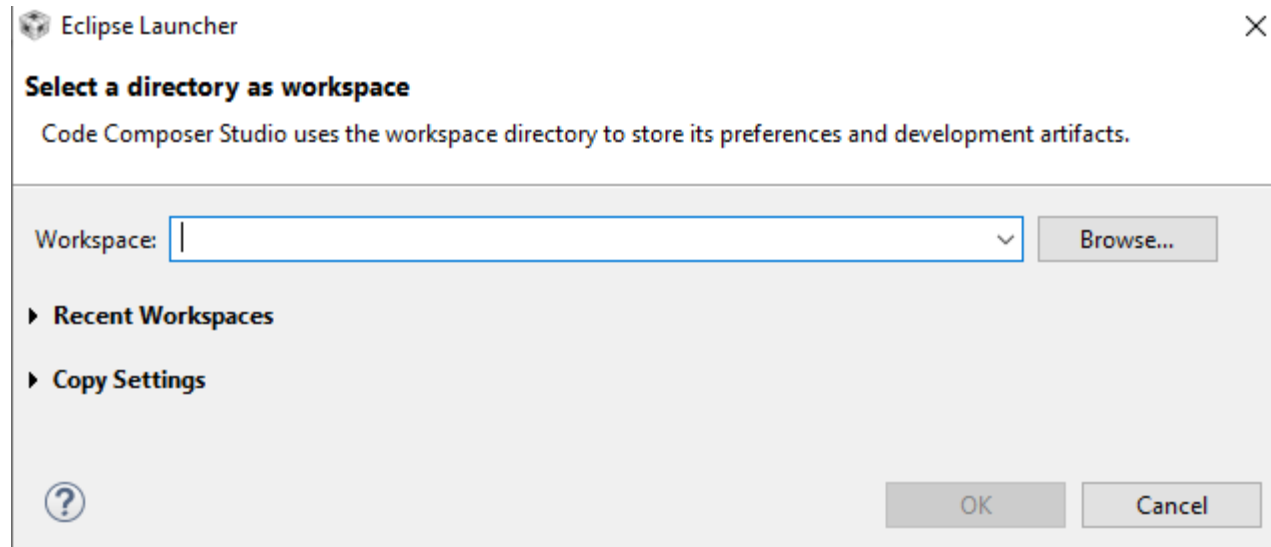


Don't Forget to select this template.

Project Set-up

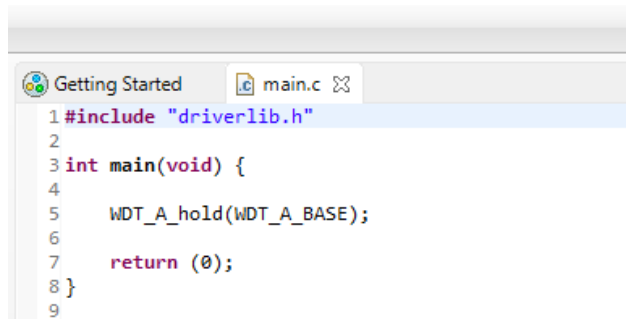
The FIRST time you use the workspace, CCS MAY open the Eclipse Launcher.

Browse to the workspace that you have created, then select OK



Code Composer Studio – Adding Code

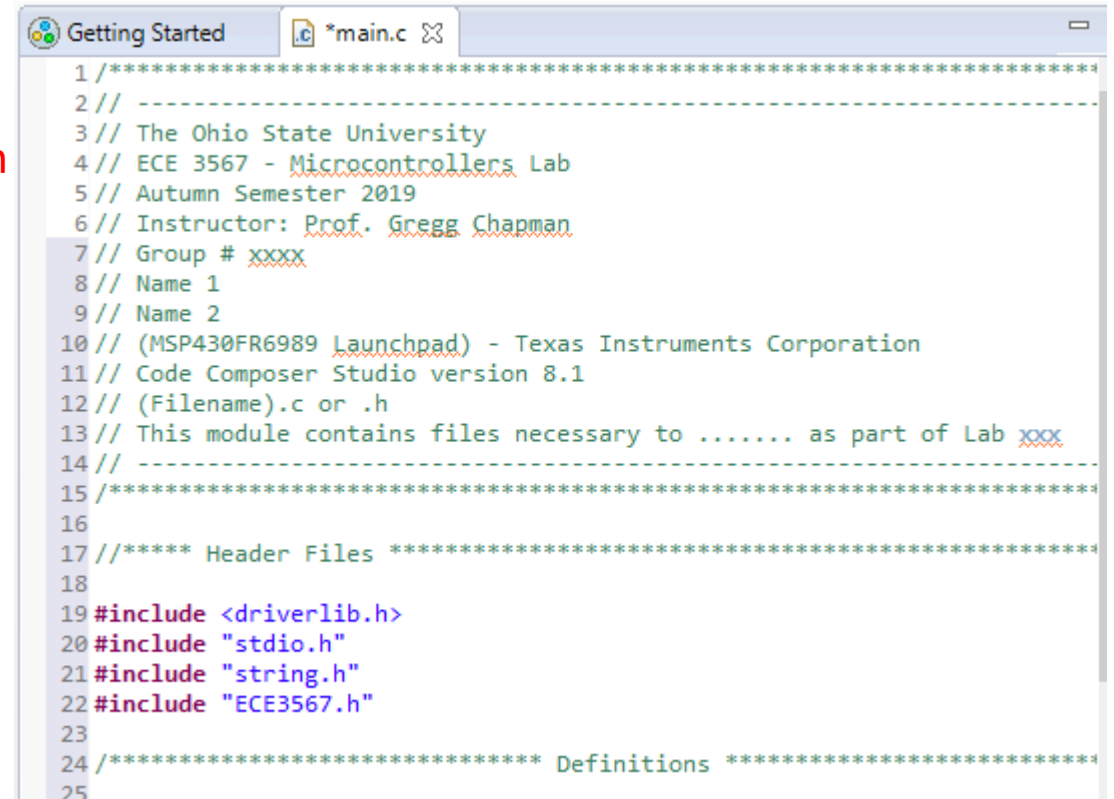
1. Select File → New → Source File
2. Enter Code
3. Save the File
4. Select Project → Rebuild Project
5. You may also COPY files into your project folder then add them to the project with a Right-Click on your **Project Name [Active-Debug]** . Then select **Add Files . . .**



The screenshot shows the Code Composer Studio interface with a single tab titled 'main.c'. The code in the editor is as follows:

```
1 #include "driverlib.h"
2
3 int main(void) {
4     WDT_A_hold(WDT_A_BASE);
5
6     return (0);
7 }
8
9
```





NOTE: It's better to copy in the standard file header and edit it than starting with the default main.c

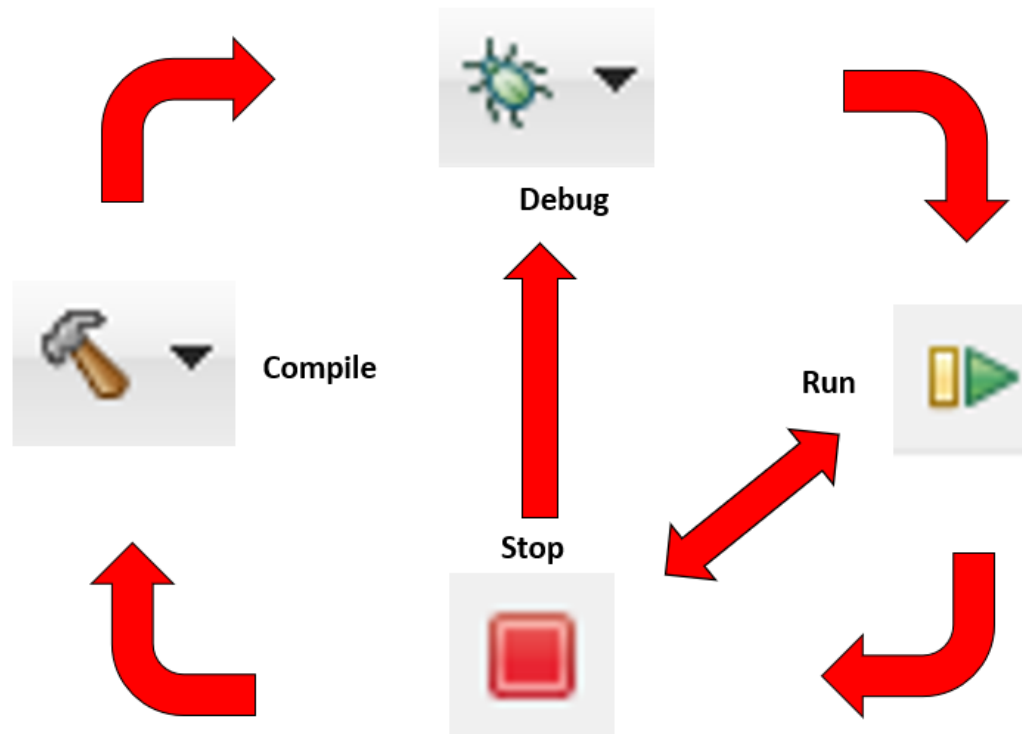


The screenshot shows the Code Composer Studio interface with a tab titled '*main.c'. The code in the editor is as follows:

```
1 /*****
2 // -----
3 // The Ohio State University
4 // ECE 3567 - Microcontrollers Lab
5 // Autumn Semester 2019
6 // Instructor: Prof. Gregg Chapman
7 // Group # xxxx
8 // Name 1
9 // Name 2
10 // (MSP430FR6989 Launchpad) - Texas Instruments Corporation
11 // Code Composer Studio version 8.1
12 // (Filename).c or .h
13 // This module contains files necessary to ..... as part of Lab xxx
14 // -----
15 /*****
16
17 //**** Header Files ****
18
19 #include <driverlib.h>
20 #include "stdio.h"
21 #include "string.h"
22 #include "ECE3567.h"
23
24 /***** Definitions ****
25
```

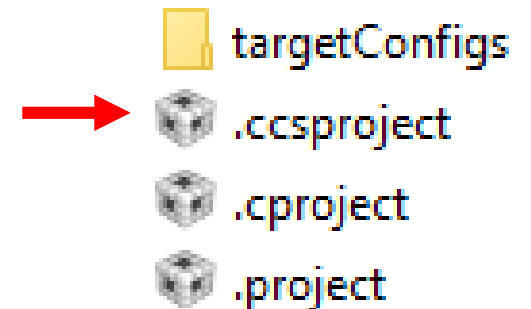
Code Composer Studio – Running the Project Code

1. At this point it is essential to connect the hardware
2. Make sure that the Project is selected as [Active – Debug]
3. You can check to see if the code compiles by selecting the hammer ICON 
4. Select the Debug ICON  (NOTE: This will also recompile the project)
5. Once the GREEN ARROW comes up you can run the code 
6. Halt execution with the RED SQUARE 



Code Composer Studio – To Open an Existing Project (Automatic Method)

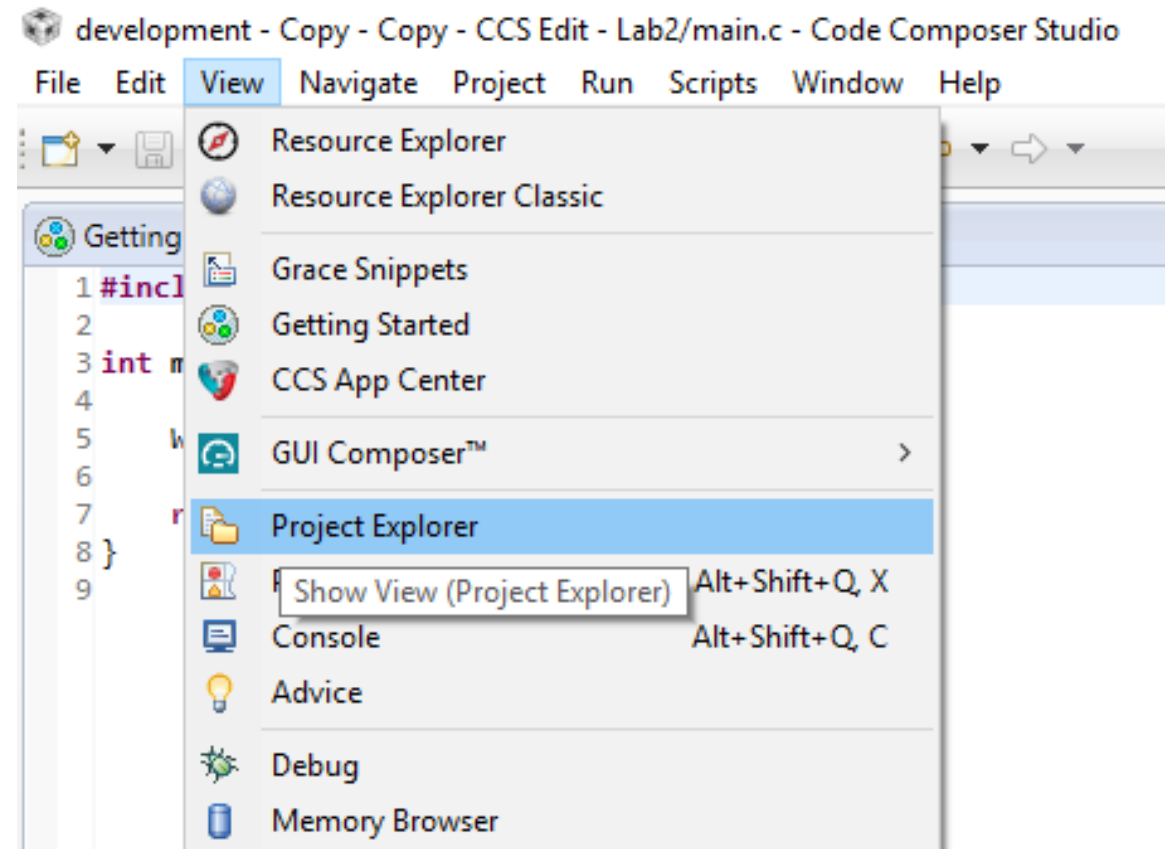
- This doesn't always work
- Make sure that the Workspace is set to your project location. To change it, select **File → Switch Workspace**, and navigate to the project location
- Double-click the **.ccsproject** ICON in the project folder
- If it doesn't work, try the Manual Method



Code Composer Studio – To Open an Existing Project (Manual Method)

If your project does not appear

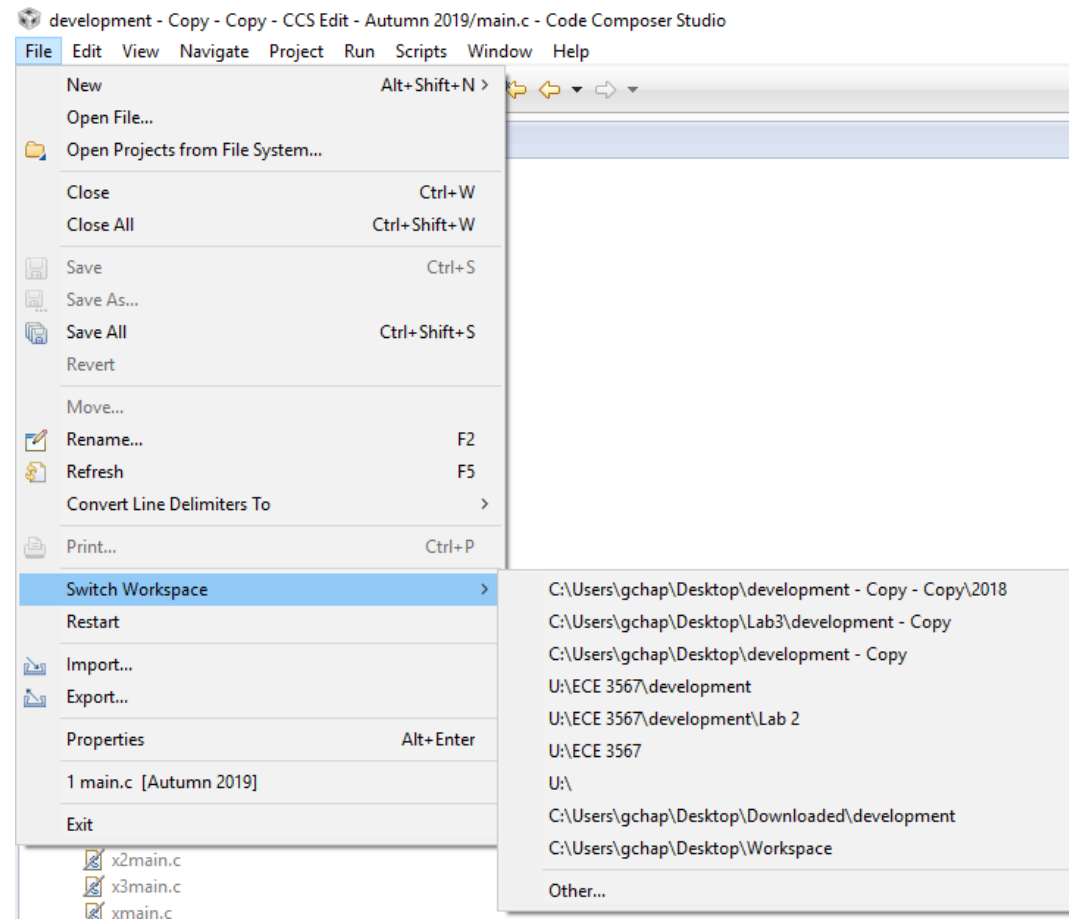
- Select **View** → **Project Explorer**



Code Composer Studio – To Open an Existing Project

If you still don't see anything in Project Explorer

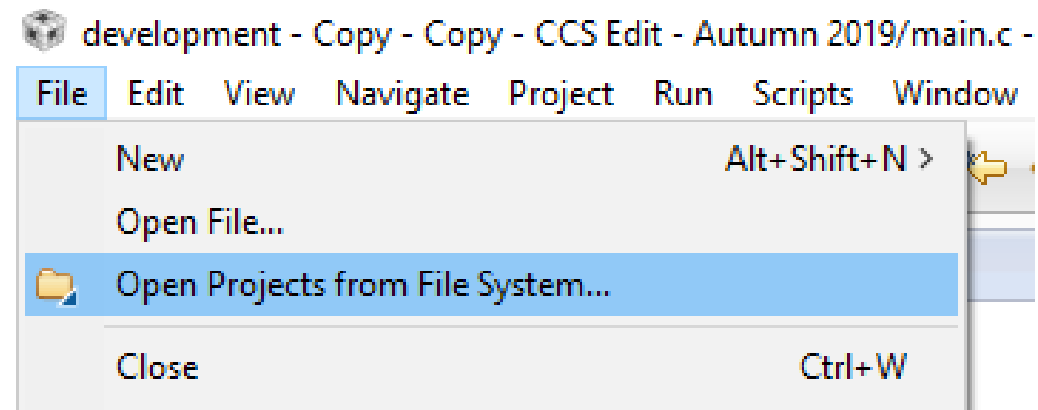
- Select **File → Switch Workspace**, and navigate to the project location



Code Composer Studio – To Open an Existing Project

If you **STILL** don't see your project in Project Explorer, re-open the file:

- **File → Open Project Files from File System . . .**



Change Workspace

1. Select **File → Switch Workspace → Other**
2. Enter the new workspace. **Always select the file ABOVE your Project file that CCS created.**

When you select OK, Code Composer will restart

Code Composer Studio

Common Error Messages

ERROR: Unable to Launch. The selection cannot be launched, and there are no recent launches

Solution: Switch Workspace : File => Switch Workspace => Other
Always select the folder that is ONE LEVEL ABOVE your project folder as the workspace.

ERROR: Cannot open source file “driverlib.h”

Solution: Re-create the project with the correct Template

IF ALL ELSE FAILS:

1. Right click the project set as **[Active-Debug]** and open Properties
2. Under MSP430 Compiler options, select Include Options
3. Delete old paths to driverlib

Add **ALL THREE** new paths:

- .../ driverlib
- .../ driverlib/MSP430FR5xx_6xx
- .../ driverlib/MSP430FR5xx_6xx/inc

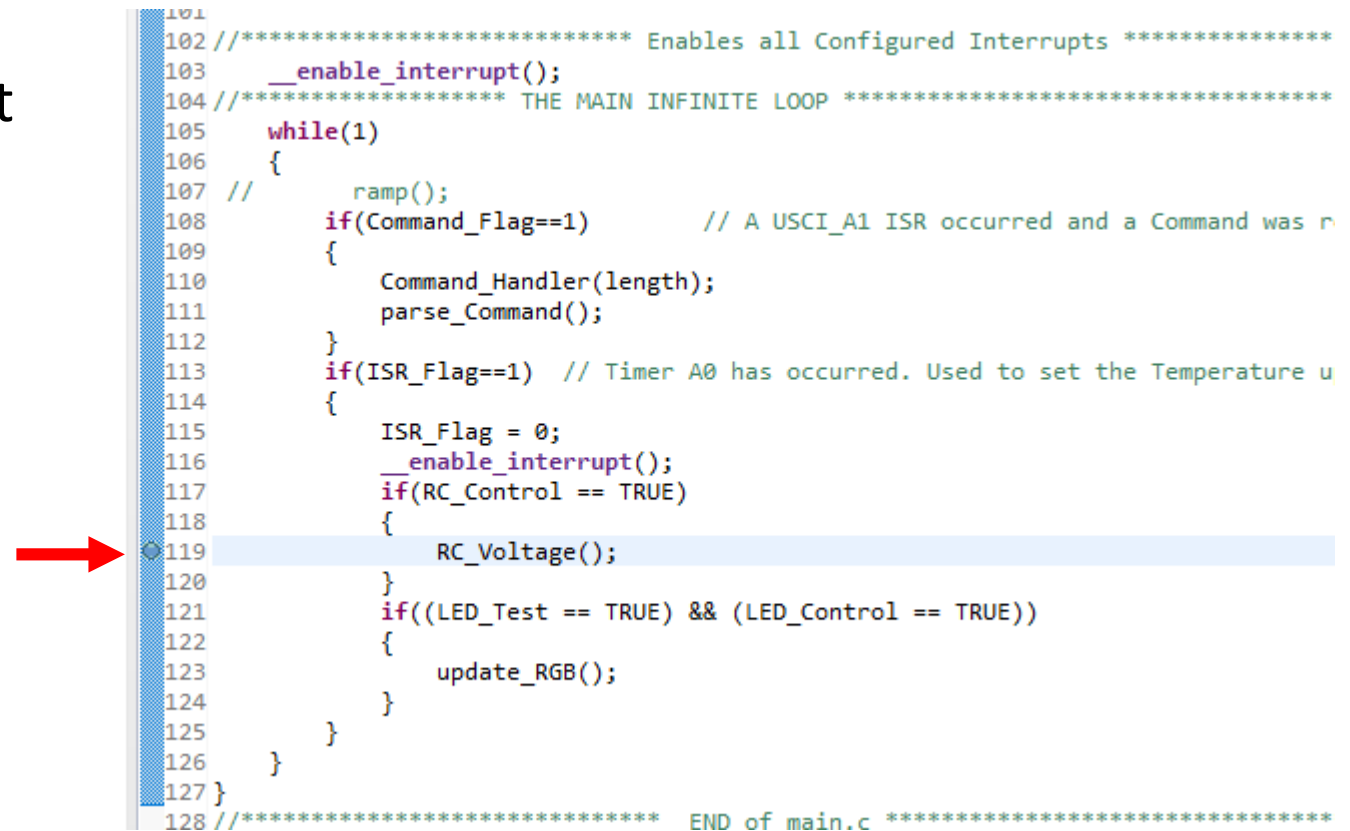
Code Composer Studio - Watch Window

Note that, unlike the TI C2000 series there is not a dynamic debugger for the TI MPS430 LaunchPads. Expressions and Variables are only updated after a Breakpoint is reached, and NOT available of the Debugger is halted.

1. Select View → Expressions
2. Right click to add registers or variable names.
3. Set breakpoints
4. Run the Debugger

Troubleshooting - Breakpoints

1. Double click the line number to set a breakpoint
2. You must re-compile to incorporate the breakpoint
3. Click twice more to clear a breakpoint



```
101
102 //***** Enables all Configured Interrupts *****
103 __enable_interrupt();
104 //***** THE MAIN INFINITE LOOP *****
105 while(1)
106 {
107     // ramp();
108     if(Command_Flag==1) // A USCI_A1 ISR occurred and a Command was r
109     {
110         Command_Handler(length);
111         parse_Command();
112     }
113     if(ISR_Flag==1) // Timer A0 has occurred. Used to set the Temperature u
114     {
115         ISR_Flag = 0;
116         __enable_interrupt();
117         if(RC_Control == TRUE)
118         {
119             RC_Voltage();
120         }
121         if((LED_Test == TRUE) && (LED_Control == TRUE))
122         {
123             update_RGB();
124         }
125     }
126 }
127 }
128 //***** END of main.c *****
```