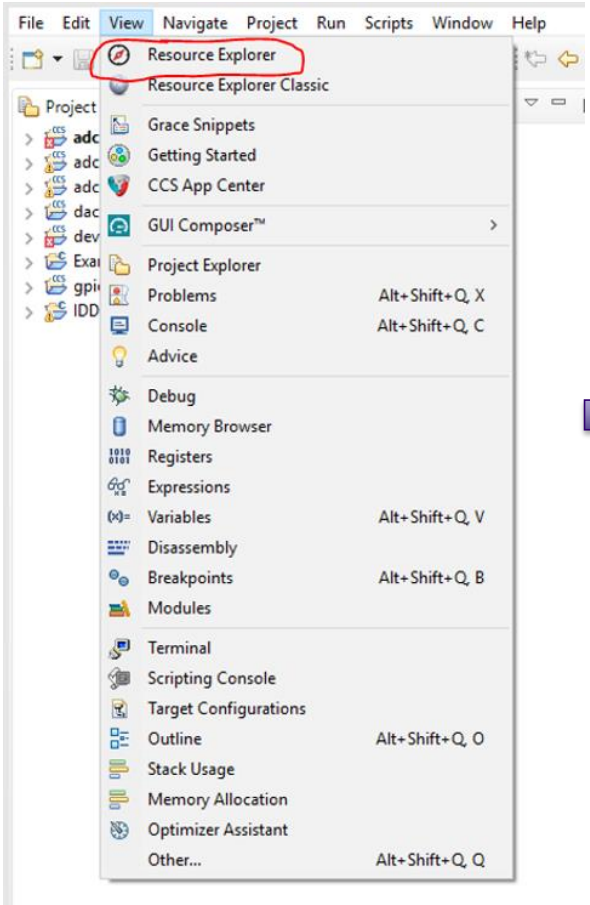
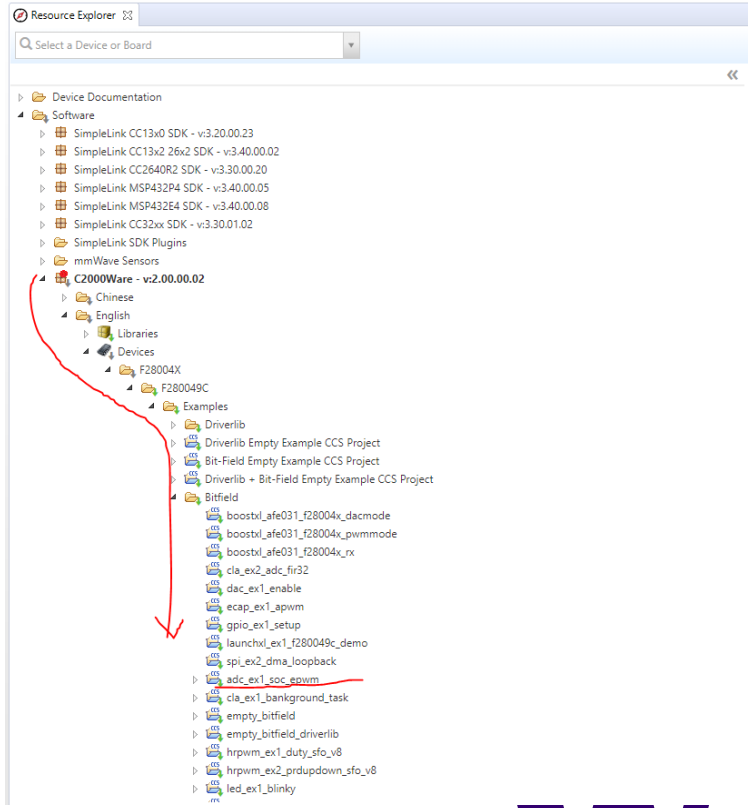
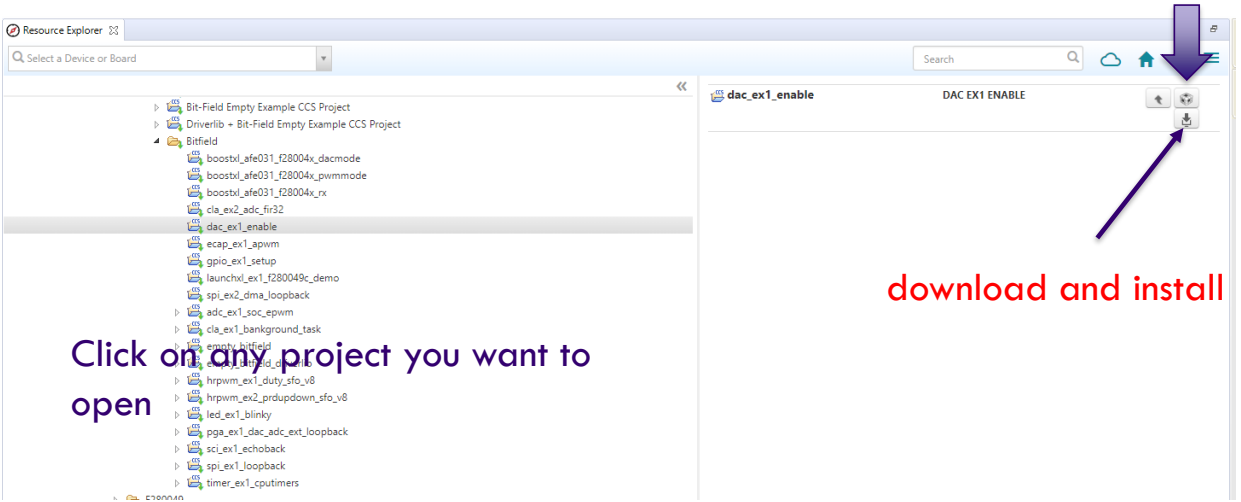
Creating a new project : Open an existing project from the Resource Explorer





Open the project titled : adc_ex1_soc_epwm

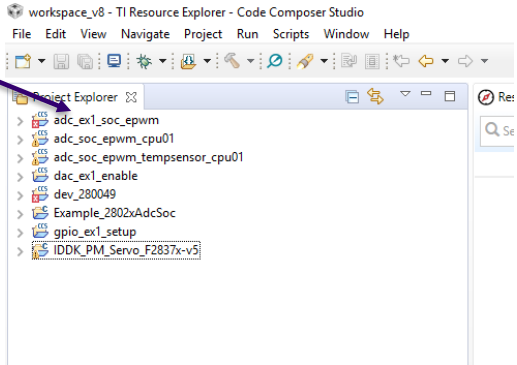
Click on this to move project into the CCS workbench (you need to download and install it when you are using CCS for the first time)



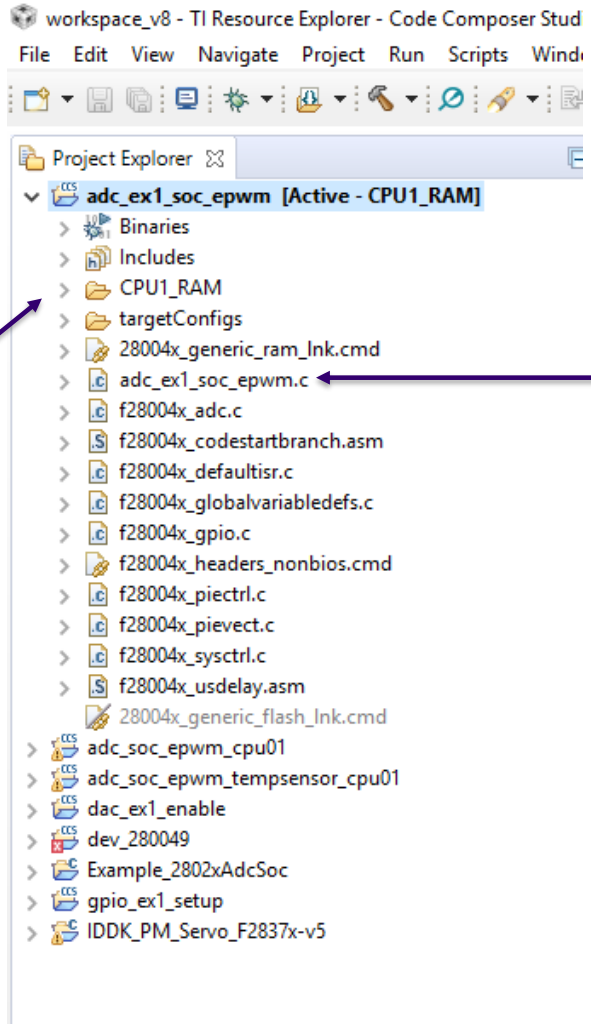
download and install

Click on any project you want to open

Now find the project in this list. A cross means that project has compilation or debug errors



Open the project titled : adc_ex1_soc_epwm



Expand the project you want to work on

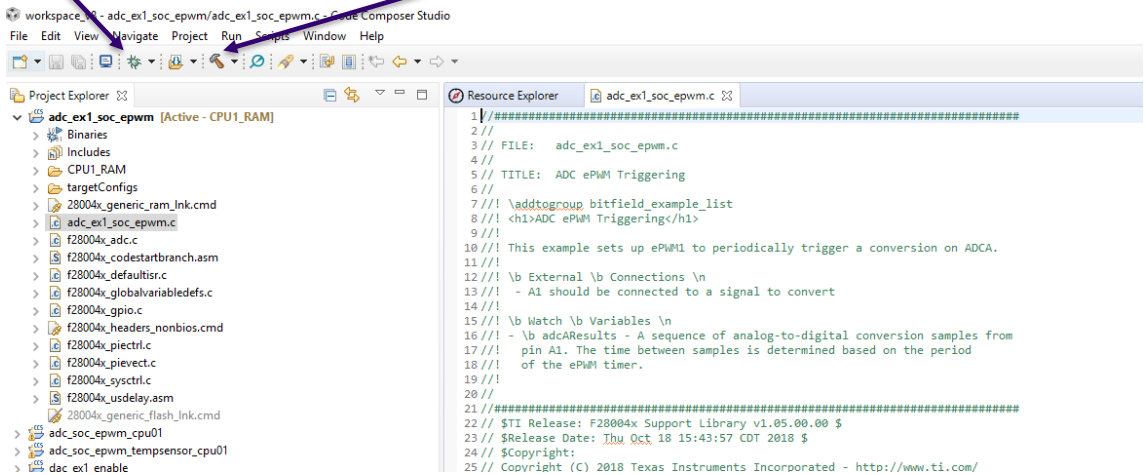
Double-click this file to open the file in an adjacent window

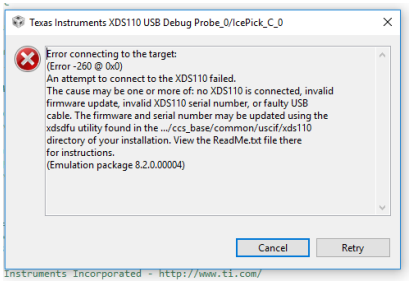
Open the project titled : adc_ex1_soc_epwm

Step 2: Connect the board through an USB to the computer

Step 3: Click this (bug) to debug

Step 1: Click this (hammer) to compile

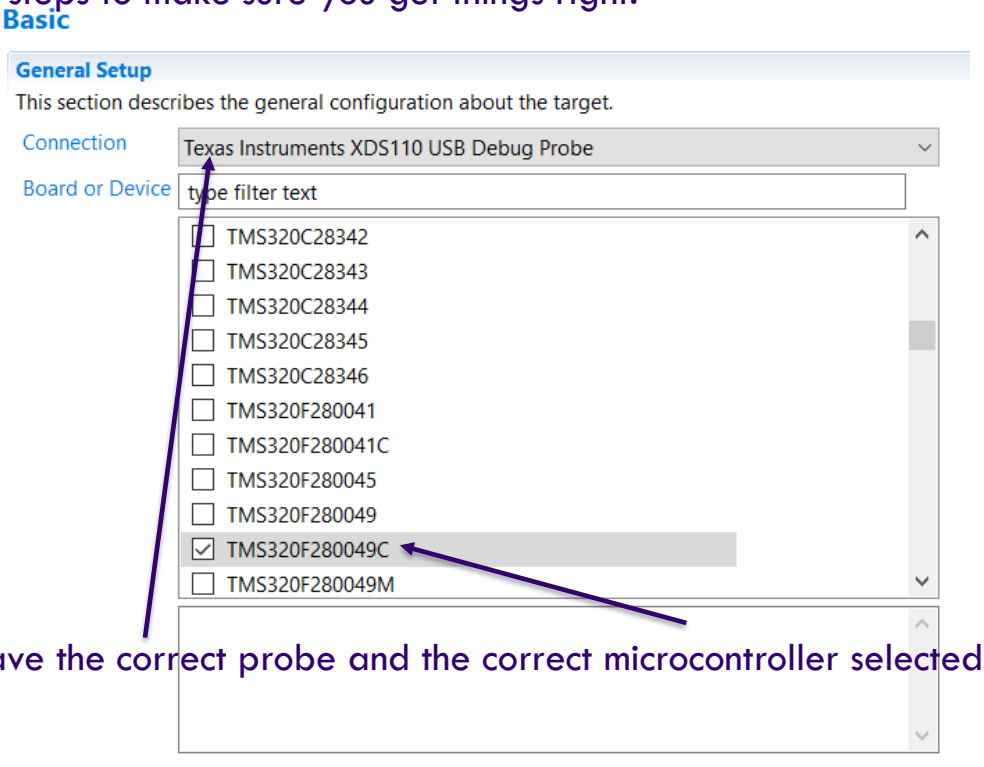




Getting this?
Check step #2

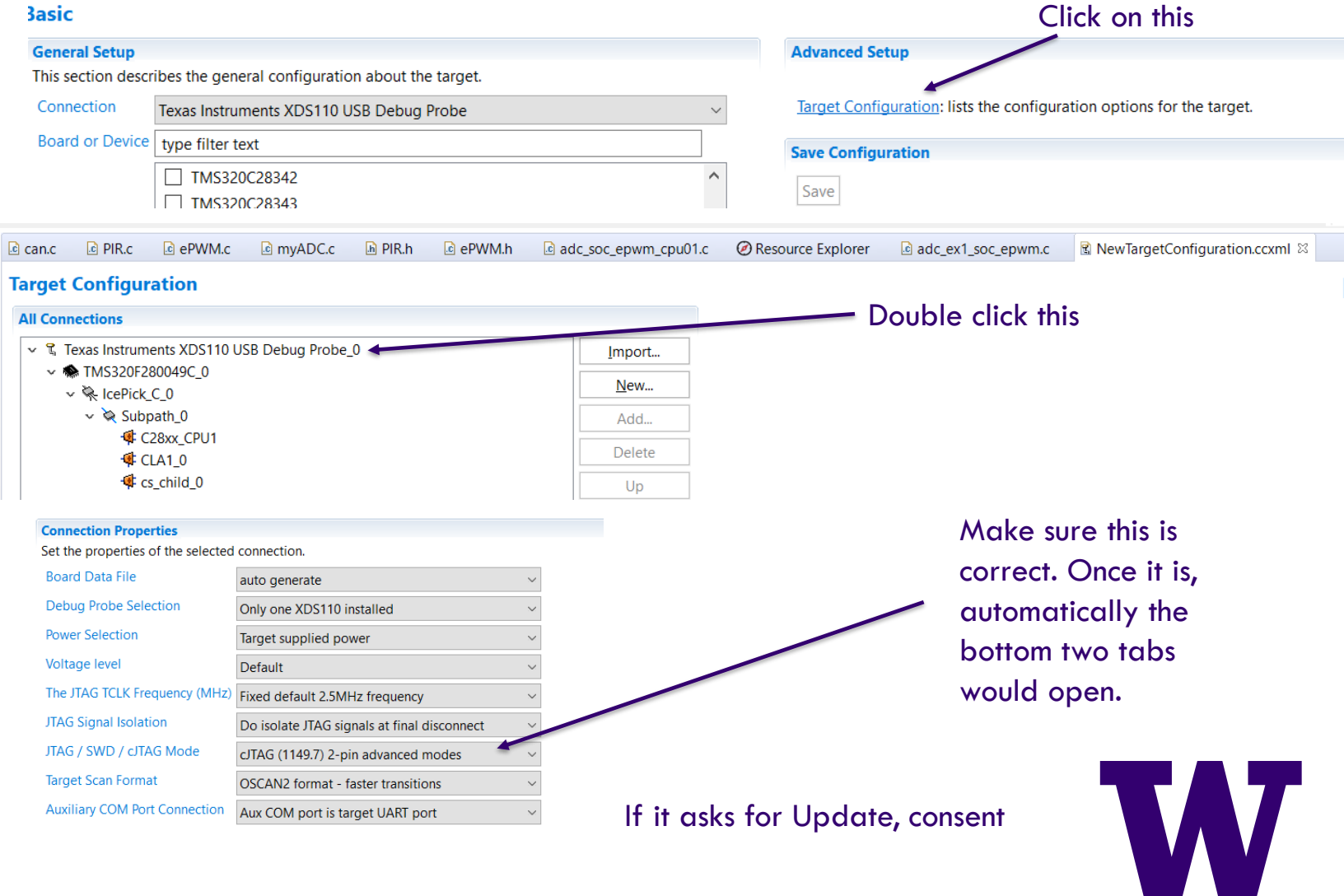
Errors!!

Like any other software, even CCS takes regular updates. And, it messes up, stuff a bit. Follow the next steps to make sure you get things right.



Make sure you have the correct probe and the correct microcontroller selected

Errors!!


Basic
General Setup
This section describes the general configuration about the target.
Connection
Texas Instruments XDS110 USB Debug Probe
Board or Device
type filter text
TMS320C28342
TMS320C28343
Advanced Setup
Target Configuration: lists the configuration options for the target.
Save Configuration
Save
Click on this
can.c
PIR.c
ePWM.c
myADC.c
PIR.h
ePWM.h
adc_soc_epwm_cpu01.c
Resource Explorer
adc_ex1_soc_epwm.c
NewTargetConfiguration.ccxml
Target Configuration
All Connections
Texas Instruments XDS110 USB Debug Probe_0
TMS320F280049C_0
IcePick_C_0
Subpath_0
C28xx_CPU1
CLA1_0
cs_child_0
Import...
New...
Add...
Delete
Up
Double click this
Connection Properties
Set the properties of the selected connection.
Board Data File
auto generate
Debug Probe Selection
Only one XDS110 installed
Power Selection
Target supplied power
Voltage level
Default
The JTAG TCLK Frequency (MHz)
Fixed default 2.5MHz frequency
JTAG Signal Isolation
Do isolate JTAG signals at final disconnect
JTAG / SWD / cJTAG Mode
cJTAG (1149.7) 2-pin advanced modes
Target Scan Format
OSCAN2 format - faster transitions
Auxiliary COM Port Connection
Aux COM port is target UART port
Make sure this is correct. Once it is, automatically the bottom two tabs would open.
If it asks for Update, consent

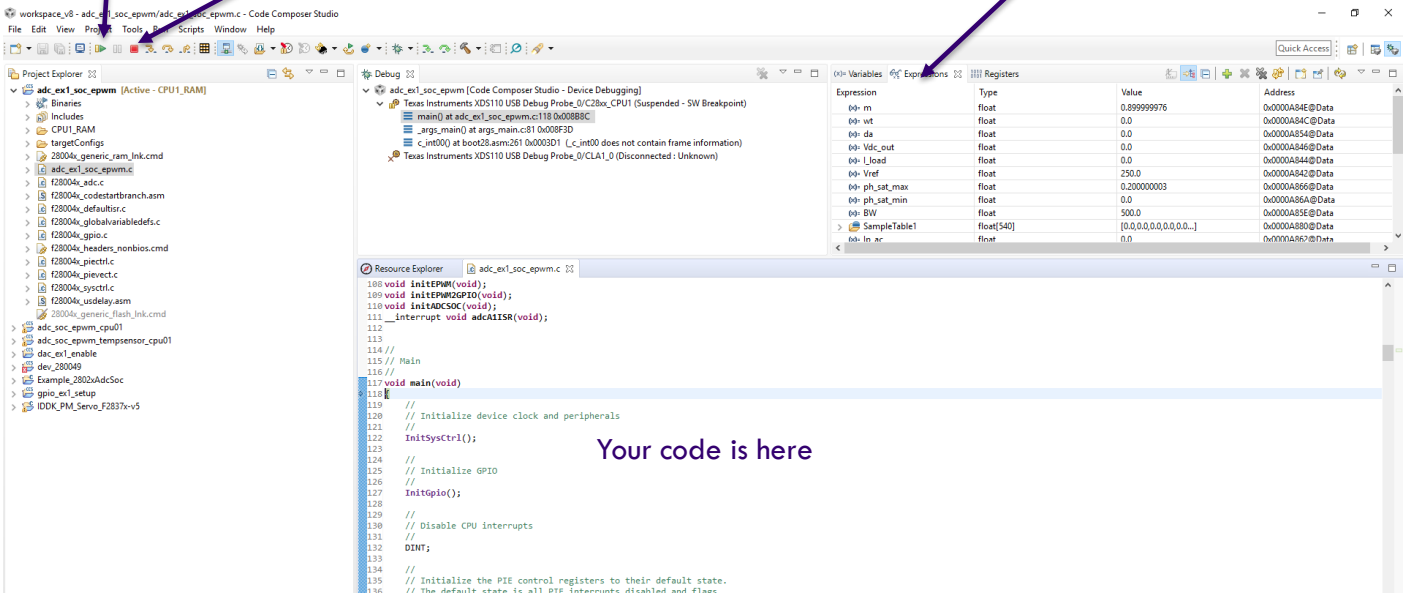
Open the project titled : adc_ex1_soc_epwm

Click on this to start the code

Click this to stop code execution by CCS

Watch Window

This is where you will add variables by scrolling down to : '*add new expression*' These variables change in real time.



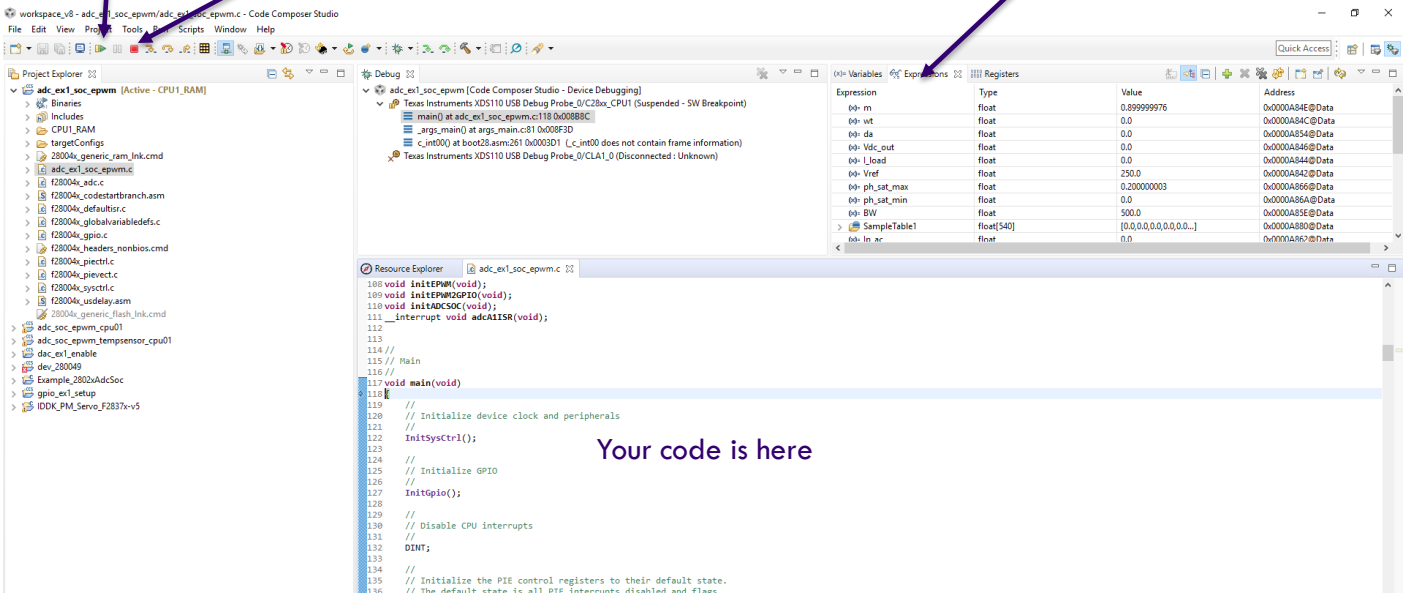
Open the project titled : adc_ex1_soc_epwm

Click on this to start the code

Click this to stop code execution by CCS

Watch Window

This is where you will add variables by scrolling down to : '*add new expression*' These variables change in real time.



Code Description

```
60 #include "F28x_Project.h"
61
62 //
63 // Defines
64 //
65 #define RESULTS_BUFFER_SIZE     256
66
67 //
68 // Globals
69 //
70 uint16_t adcAResults[RESULTS_BUFFER_SIZE];   // Buffer for results
71 uint16_t index;                              // Index into result buffer
72 volatile uint16_t bufferFull;                // Flag to indicate buffer is full
73
74 //
75 // Function Prototypes
76 //
77 void initADC(void);
78 void initEPWM(void);
79 void initADCSOC(void);
80 __interrupt void adcA1ISR(void);
81
82 //
83 // Main
84 //
85 void main(void)
86 {
87     //
88     // Initialize device clock and peripherals
89     //
90     InitSysCtrl();
91
92     //
93     // Initialize GPIO
94     //
95     InitGpio();
```

#include “math.h” goes here

ignore

Global variables. Use only int/float. You can initialize variables here.
e.g. float a=5; is permitted

User defined functions, feel free to change the names to whatever you like. These are definitions

Main function starts here. The function name and format is fixed.

This sets up your microcontroller. Hover around to the name, to see the whole function

This calls the function

Code Description

```
while(1)
{
    //
    // Start ePWM
    //
    EPwm1Regs.ETSEL.bit.SOCAEN = 1;    // Enable SOCA
    EPwm1Regs.TBCTL.bit.CTRMODE = 0;   // Unfreeze, and enter up cc

    //
    // Wait while ePWM causes ADC conversions, which then cause int
    // which fill the results buffer, eventually setting the buffer
    // flag
    //
    while(!bufferFull)
    {
    }
    bufferFull = 0; //clear the buffer full flag

    //
    // Stop ePWM
    //
    EPwm1Regs.ETSEL.bit.SOCAEN = 0;    // Disable SOCA
    EPwm1Regs.TBCTL.bit.CTRMODE = 3;   // Freeze counter

    //
    // Software breakpoint. At this point, conversion results are s
    // adcAResults.
    //
    // Hit run again to get updated conversions.
    //
    ESTOP0;
}
```

Always true condition

Ignore/ Delete

Delete

Code Description

```
213 void initADC(void)
214 {
215     //
216     // Setup VREF as internal
217     //
218     SetVREF(ADC_ADCA, ADC_INTERNAL, ADC_VREF3P3);
219
220     EALLOW;
221
222     //
223     // Set ADCCLK divider to /4
224     //
225     AdcaRegs.ADCCTL2.bit.PRESCALE = 6;
226
227     //
228     // Set pulse positions to late
229     //
230     AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
231
232     //
233     // Power up the ADC and then delay for 1 ms
234     //
235     AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
236     EDIS;
237
238     DELAY_US(1000);
239 }
240
```

Sets up ADC timing and reference voltage

Sampling time

Code Description

```
244 void initEPWM(void)
245 {
246     EALLOW;
247
248     EPwm1Regs.ETSEL.bit.SOCAEN = 0;      // Disable SOC on A group
249     EPwm1Regs.ETSEL.bit.SOCASEL = 4;     // Select SOC on up-count
250     EPwm1Regs.ETPS.bit.SOCAPRD = 1;      // Generate pulse on 1st event
251
252     EPwm1Regs.CMPA.bit.CMPA = 0x0800;    // Set compare A value to 2048 counts
253     EPwm1Regs.TBPRD = 0x1000;            // Set period to 4096 counts
254
255     EPwm1Regs.TBCTL.bit.CTRMODE = 3;     // Freeze counter
256
257     EDIS;
258 }
259 //
260 // initADCSOC - Function to configure ADCA's SOC0 to be triggered by ePWM1.
261 //
262 void initADCSOC(void)
263 {
264     //
265     // Select the channels to convert and the end of conversion flag
266     //
267     EALLOW;
268
269     AdcaRegs.ADCSOC0CTL.bit.CHSEL = 1;      // SOC0 will convert pin A1
270                                             // 0:A0  1:A1  2:A2  3:A3
271                                             // 4:A4   5:A5   6:A6   7:A7
272                                             // 8:A8   9:A9   A:A10  B:A11
273                                             // C:A12  D:A13  E:A14  F:A15
274     AdcaRegs.ADCSOC0CTL.bit.ACQPS = 9;      // Sample window is 10 SYSCLK cycles
275     AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5;    // Trigger on ePWM1 SOCA
276
277     AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0; // End of SOC0 will set INT1 flag
278     AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   // Enable INT1 flag
279     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Make sure INT1 flag is cleared
280
281     EDIS;
```

Sets up the PWM channels to trigger the ADC, as well as perform the PWM operation

Sets up a specific channel to do analog to digital conversion

Code Description

```
244 void initEPWM(void)
245 {
246     EALLOW;
247
248     EPwm1Regs.ETSEL.bit.SOCAEN = 0;      // Disable SOC on A group
249     EPwm1Regs.ETSEL.bit.SOCASEL = 4;     // Select SOC on up-count
250     EPwm1Regs.ETPS.bit.SOCAPRD = 1;      // Generate pulse on 1st event
251
252     EPwm1Regs.CMPA.bit.CMPA = 0x0800;    // Set compare A value to 2048 counts
253     EPwm1Regs.TBPRD = 0x1000;            // Set period to 4096 counts
254
255     EPwm1Regs.TBCTL.bit.CTRMODE = 3;     // Freeze counter
256
257     EDIS;
258 }
259 //
260 // initADCSOC - Function to configure ADCA's SOC0 to be triggered by ePWM1.
261 //
262 void initADCSOC(void)
263 {
264     //
265     // Select the channels to convert and the end of conversion flag
266     //
267     EALLOW;
268
269     AdcaRegs.ADCSOC0CTL.bit.CHSEL = 1;      // SOC0 will convert pin A1
270                                             // 0:A0  1:A1  2:A2  3:A3
271                                             // 4:A4   5:A5   6:A6   7:A7
272                                             // 8:A8   9:A9   A:A10  B:A11
273                                             // C:A12  D:A13  E:A14  F:A15
274     AdcaRegs.ADCSOC0CTL.bit.ACQPS = 9;      // Sample window is 10 SYSCLK cycles
275     AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5;    // Trigger on ePWM1 SOCA
276
277     AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0; // End of SOC0 will set INT1 flag
278     AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   // Enable INT1 flag
279     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Make sure INT1 flag is cleared
280
281     EDIS;
```

Sets up the PWM channels to trigger the ADC, as well as perform the PWM operation

Sets up a specific channel to do analog to digital conversion

Code Description

```
287 __interrupt void adcA1ISR(void)
288 {
289     //
290     // Add the latest result to the buffer
291     // ADCRESULT0 is the result register of SOC0
292     adcAResults[index++] = AdcaResultRegs.ADCRESULT0;
293
294     //
295     // Set the bufferFull flag if the buffer is full
296     //
297     if(RESULTS_BUFFER_SIZE <= index)
298     {
299         index = 0;
300         bufferFull = 1;
301     }
302
303     //
304     // Clear the interrupt flag
305     //
306     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
307
308     //
309     // Check if overflow has occurred
310     //
311     if(1 == AdcaRegs.ADCINTOVF.bit.ADCINT1)
312     {
313         AdcaRegs.ADCINTOVFCLR.bit.ADCINT1 = 1; //clear INT1 overflow flag
314         AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
315     }
316
317     //
318     // Acknowledge the interrupt
319     //
320     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
321 }
```

We will implement the controller here

This has the sampled value of feedback variable

Delete these