EE 458 Lab 1 Presentation

Rahul Mallik
(rmallik@uw.edu)

ELECTRICAL ENGINEERING
UNIVERSITY of WASHINGTON

Contents

- Intro to Real-time simulation
- Intro to CCS
- Intro to PLECS RT Box

Background of real time simulation

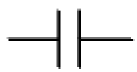
- In this section, we will learn about real time simulation
- We will use a basic buck converter as an example
- We expect you to know the following

  - circuit theory

  - fundamentals of PWM converters

Real time simulation

In real life, we have physical components like capacitors, inductors, devices relating voltage through them and current across them by some mathematical equations.

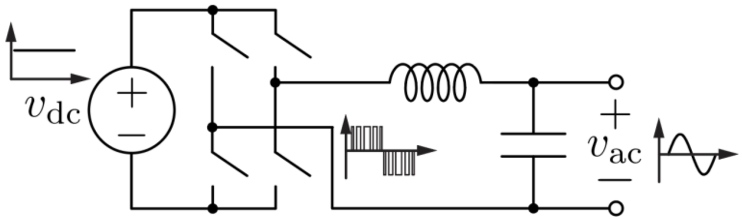


$$i = C\frac{dv}{dt}$$

$$v = L\frac{di}{dt}$$

$$v = iR$$

In simulation environment, the algebraic and differential equations are solved numerically.



A power converter is made up of many such equations

$$\dot{x} = f(x, t)$$
$$g(x, t) = 0$$

The idea of time step

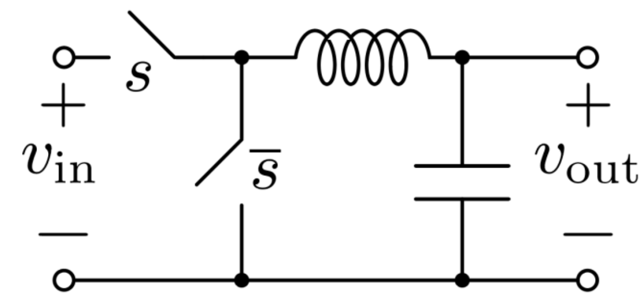
Let us say a buck converter is switched

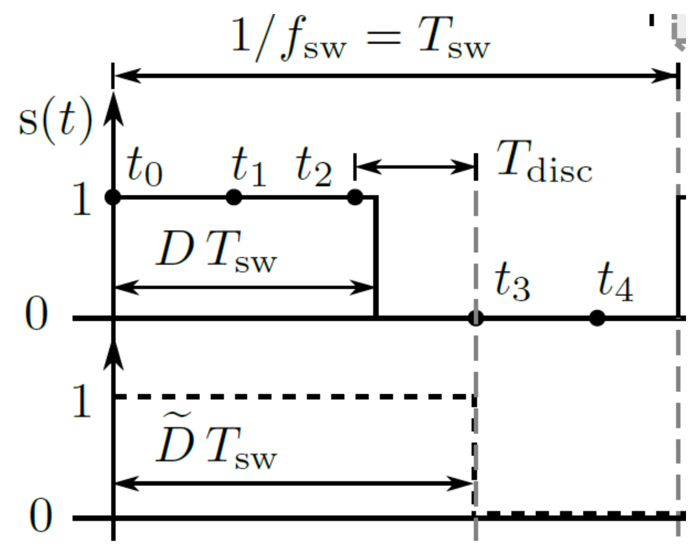
The simulation calculates the value of variables like voltage and currents of different active and passive components at **certain fixed intervals.** This is called the discrete time step.

From the figure, we can see the finer the resolution, the lesser the error between the actual switching in real world, vs what the simulation predicts.

Error in resolution : $\left|D - \widetilde{D}\right|$

From fig, we can say : $\boxed{\left|D - \widetilde{D}\right| < \frac{T_{disc}}{T_{sw}}}$



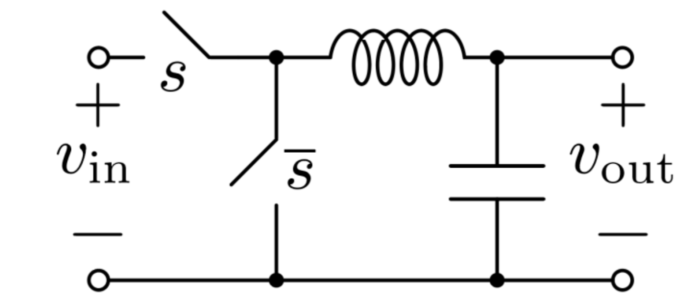


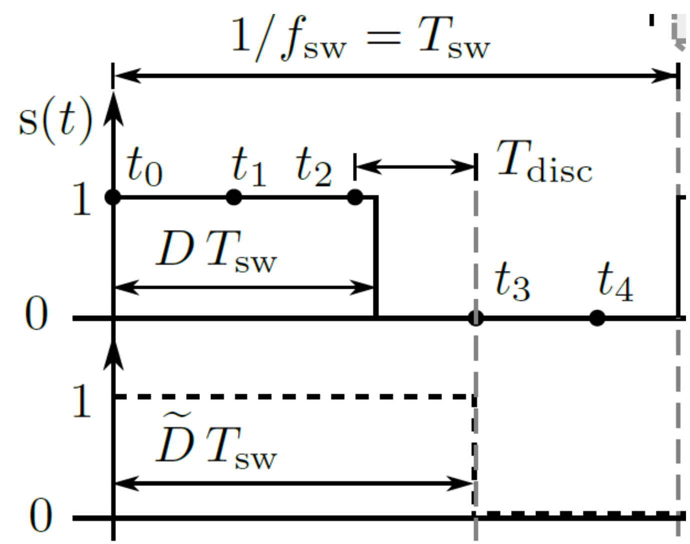
How small can the time step be?

In simulation?
Make time step as small as possible, say 1ns, it still works.

What does it do?
Lets say, converter switching frequency is 10 kHz. So, $T_{sw} = 100\mu s$.
Now, In each switching cycle, we have to solve, all the algebraic and differential equations $\frac{100\mu s}{1ns} = 100000$ times, with each step solving from where the last step left off.

In PLECS simulation, this might take about an hour, to get a whole 1s simulation to complete





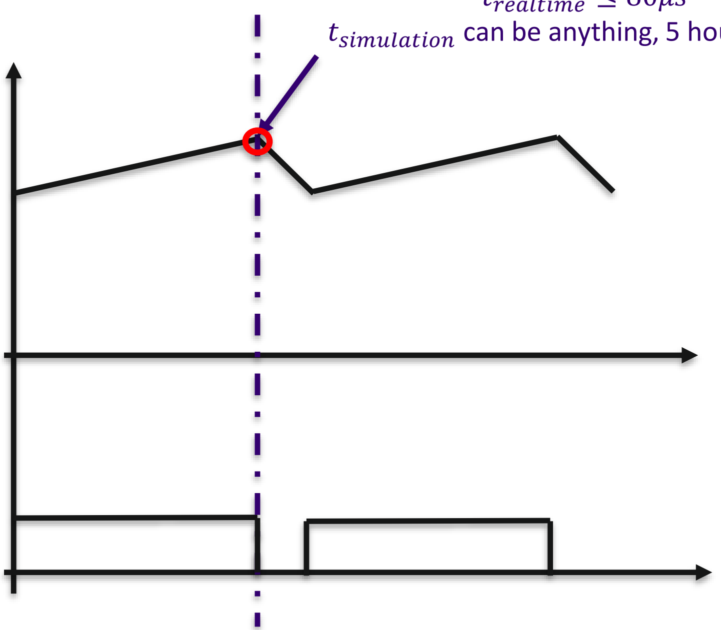
Idea of real time

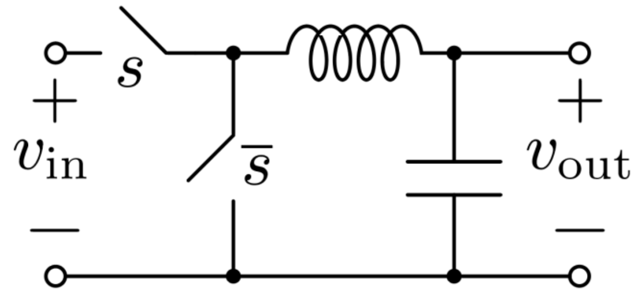
In this same simulation, what we want is, the following: Same waveforms at same time!

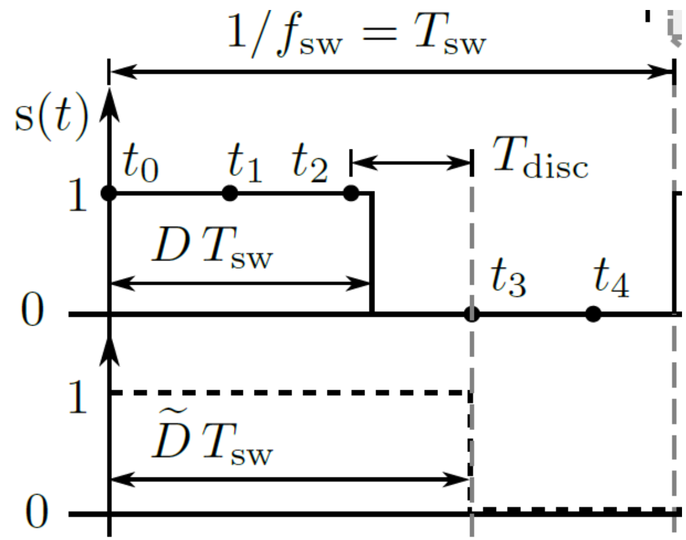
$t_{hardware} = 80\mu s\ (say\ D = 0.8)$

$t_{realtime} \leq 80\mu s$

$t_{simulation}$ can be anything, 5 hours, 1 day!

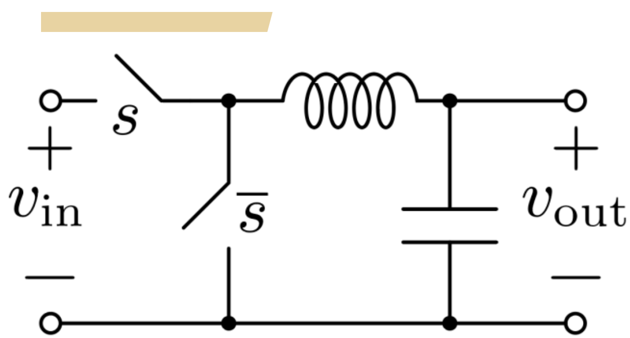


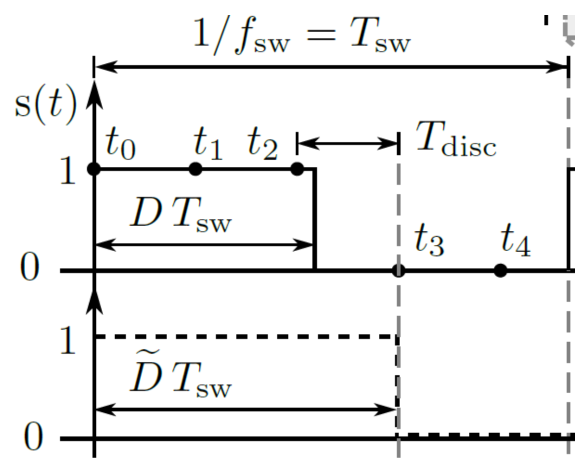


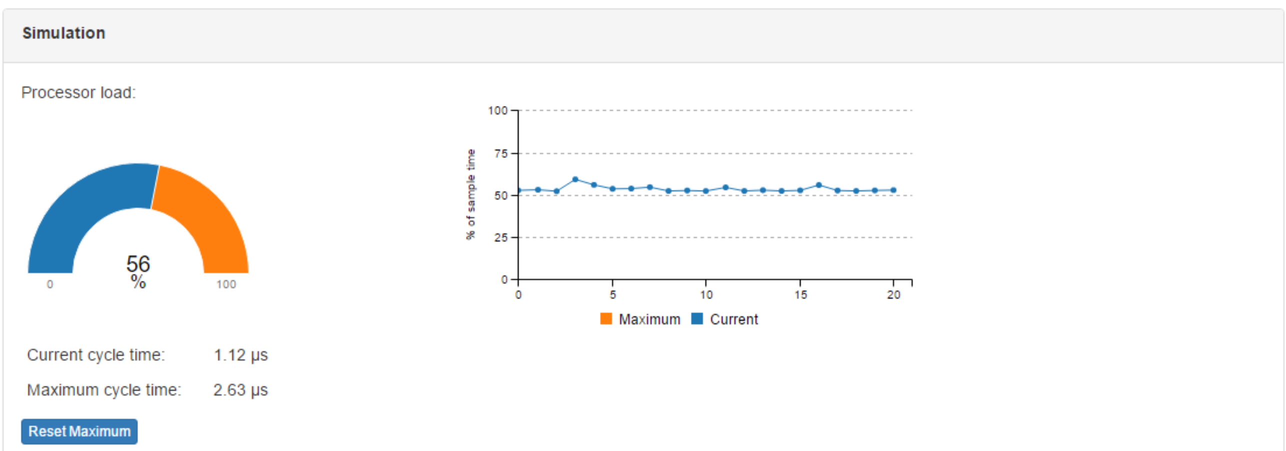


The value of the inductor current should be say, 3.4 A in the hardware at the end of the S on duration. The simulation should exactly give that value, if not, then it would give an error message to reduce the time step. In the real time, it should take lesser time than real time and approximate the solution as close as possible

PLECS Realtime Simulation



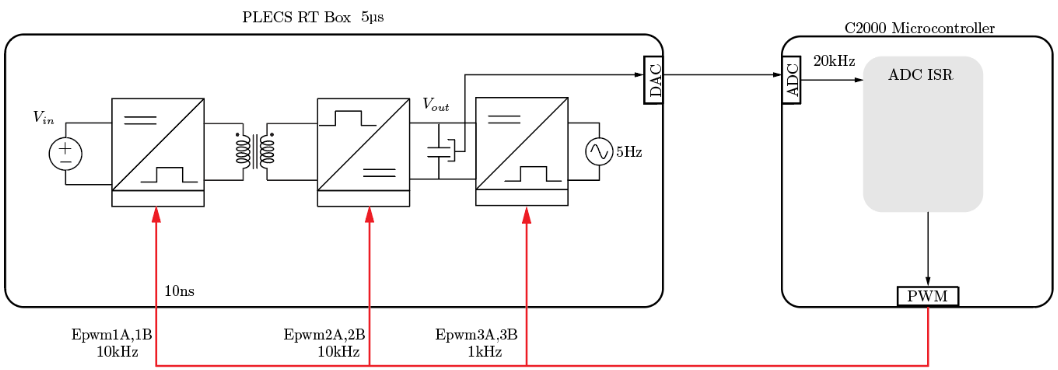




$$T_{sw} = 100\mu s$$
$$T_{disc} = 5\mu s$$

In this implementation, it shows that, all calculations in each step takes less than the step duration of 5us, so, all the predicted values of current and voltages are correct, and we *are doing* real time simulation

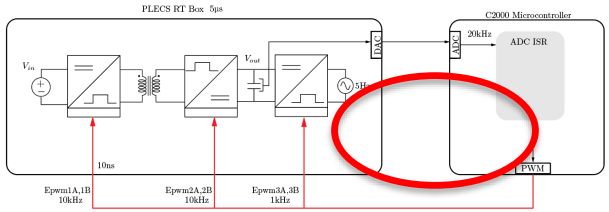
Why do we need real time simulation: Hardware-in-loop



We want to study the controller performance. So imagine from the real world, physical hardware, we keep the controller intact and just replace the actual hardware with the box in picture.
The RTBox now ensures that it is an exact time-synced replica of the hardware. So new control algorithms can now be written on the microcontroller and we will be able to predict, from the behavior of real time, how the actual hardware will behave.

Hardware-in-loop

Here the HIL is the controller. It is like a closed loop.



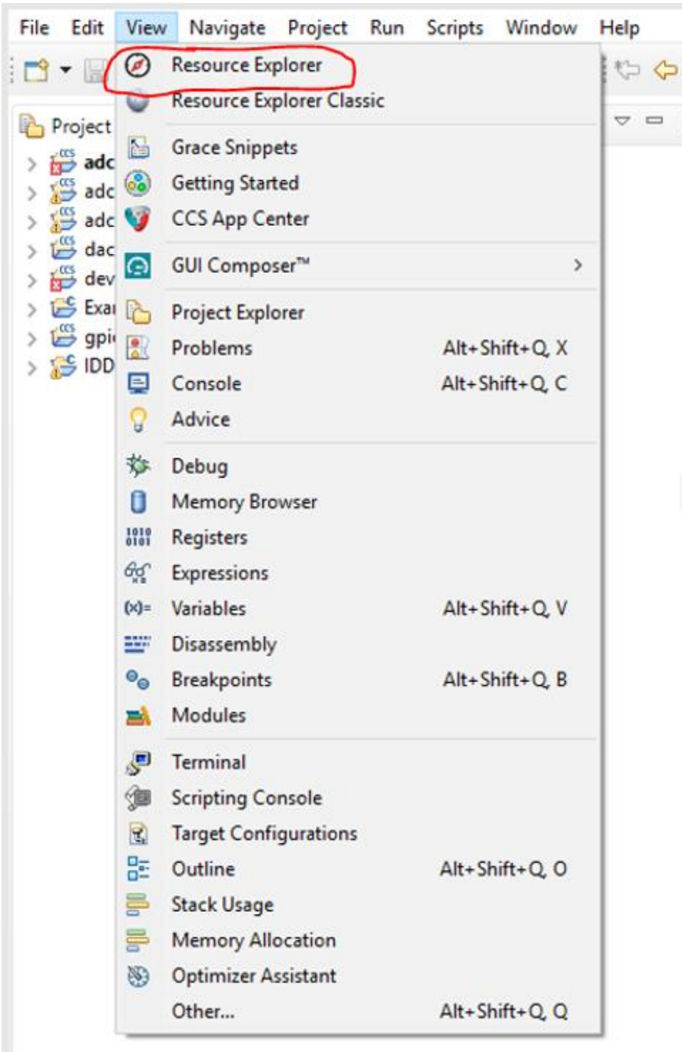
The advantages of HIL are :

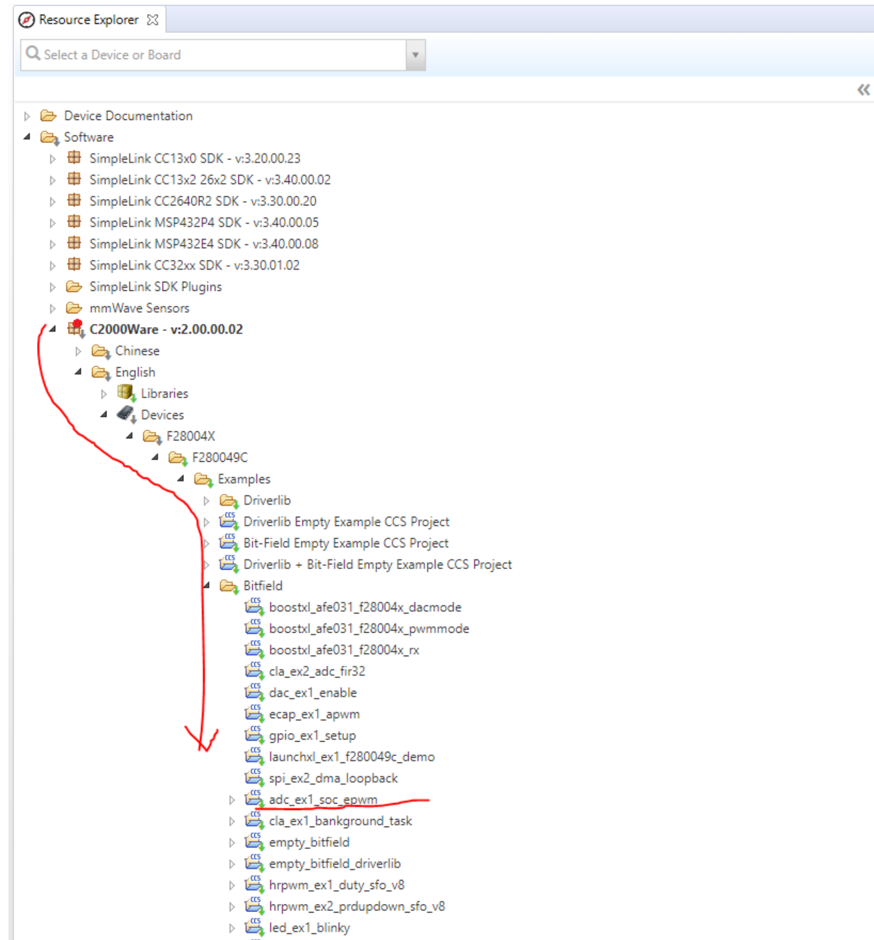
- No associated hardware risks, or cost involved
- Large number of uncertainties (signal corruption through power supplies, faulted gate drivers etc) that contribute to erroneous results cannot be isolated from control performance. RT Box (HIL) ensures we decouple control performance from all such uncertainties.
- Extremely quick execution from design to verification

Getting started with CCS

- In this section, we will learn how to start with CCS and embedded C coding.
- Microcontroller used : TI 280049C (Launchpad version)
- We expect you to read the following **before starting Experiment 1** (see Canvas Files/Lab/Reference Materials directory).
  - ADC and DAC submodules (F280049C_ADC.pdf)
  - EPWM submodule (F280049C_EPWM.pdf, understand Figs 2.7 through 2.10)
  - Background reading of this specific microcontroller (F280049C_Overview.pdf)

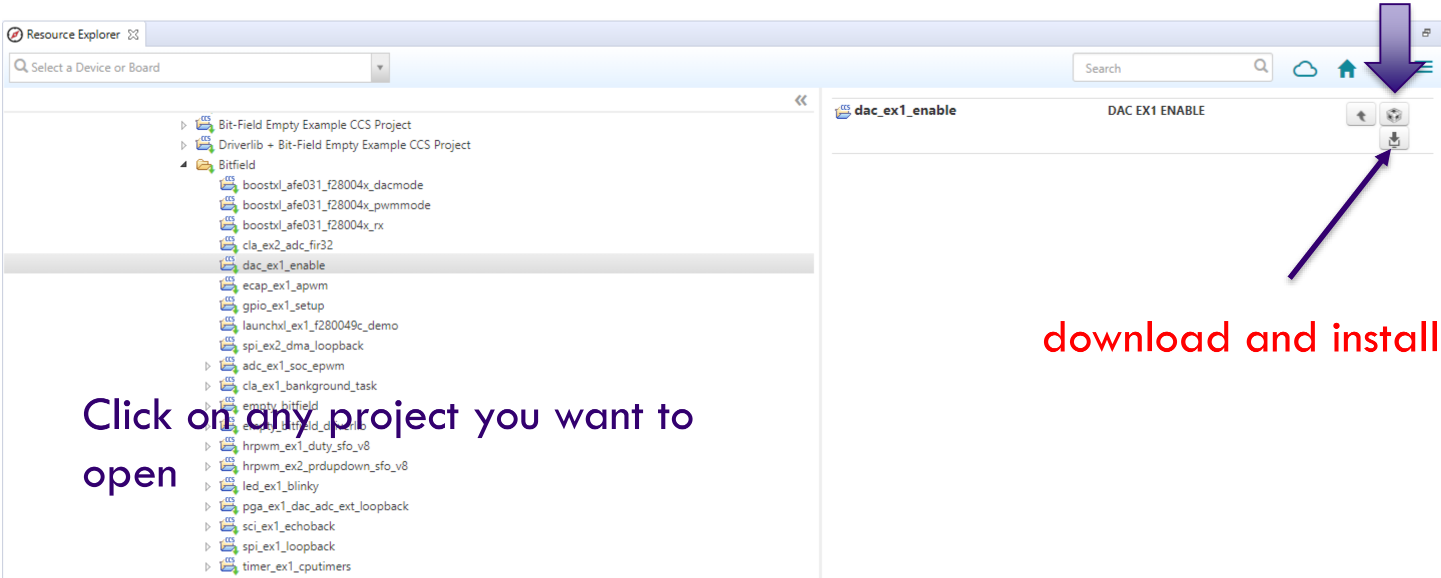
Creating a new project : Open an existing project from the Resource Explorer





Open the project titled : adc_ex1_soc_epwm

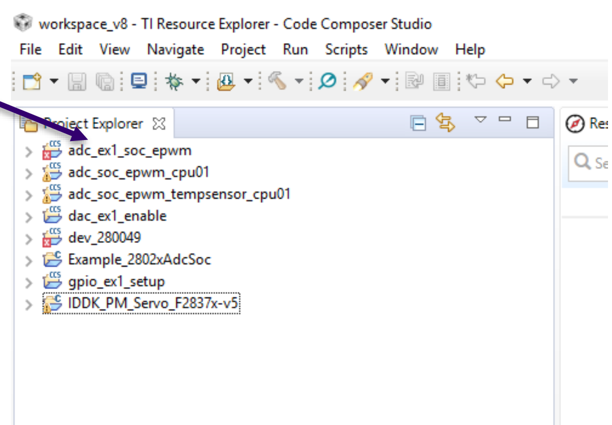
Click on this to move project into the CCS workbench (you need to download and install it when you are using CCS for the first time)



download and install

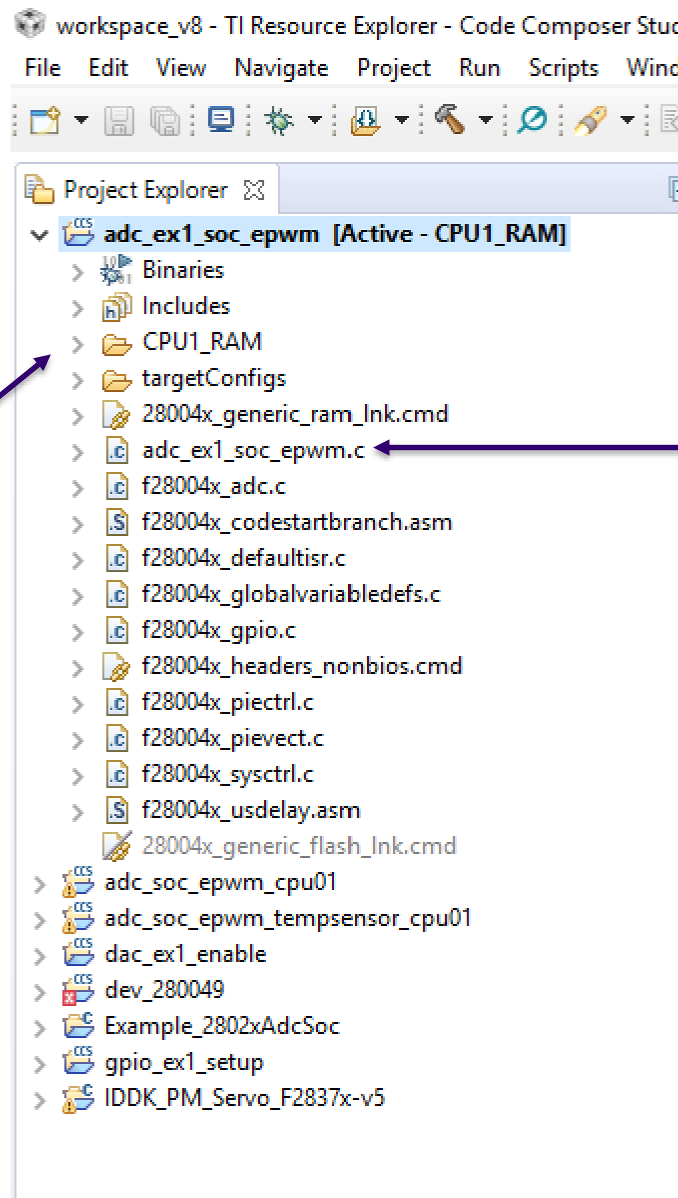
Click on any project you want to open

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



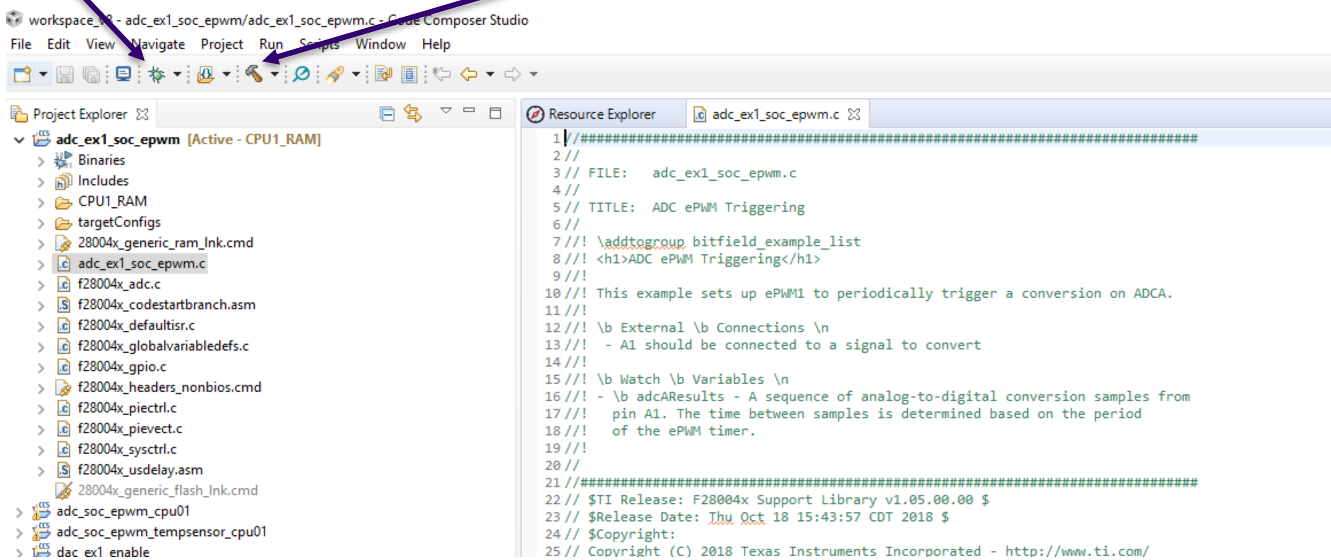
Double-click this file to open the file in an adjacent window

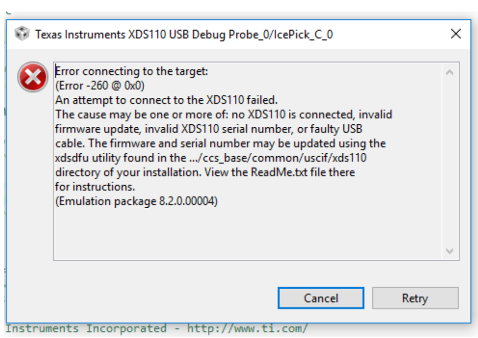
Open the project titled : adc_ex1_soc_epwm

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

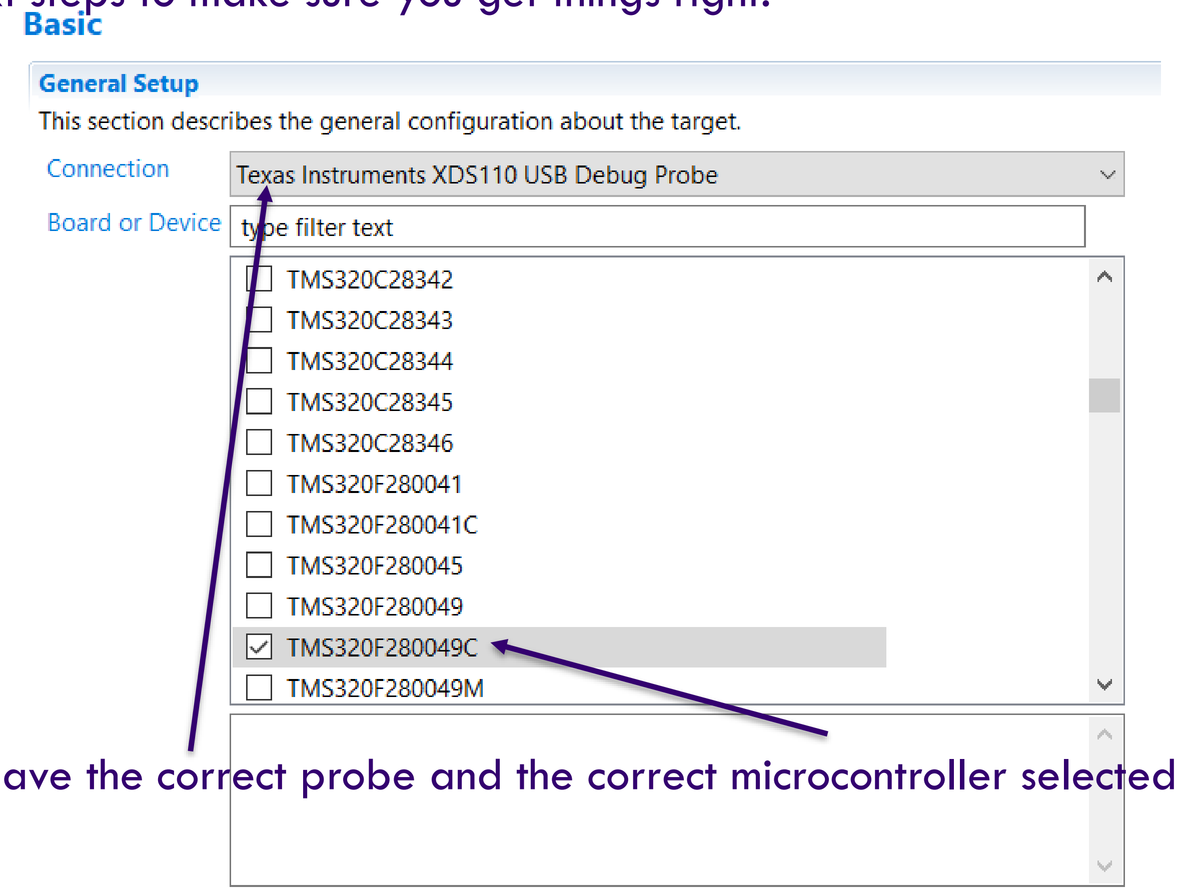




Getting this?
Check step #2

Errors!!

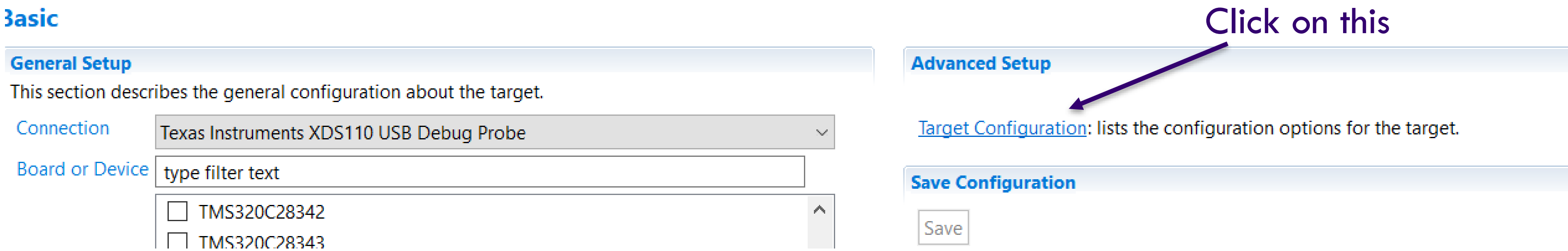
Like any other software, even CCS takes regular updates. And, it messes up, stuff a bit. Follow the next steps to make sure you get things right.



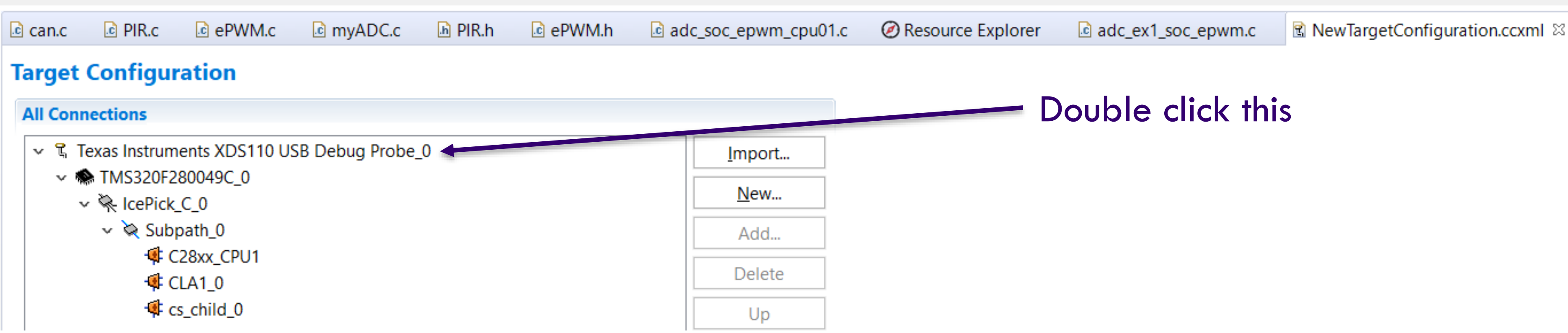
Make sure you have the correct probe and the correct microcontroller selected

Errors!!

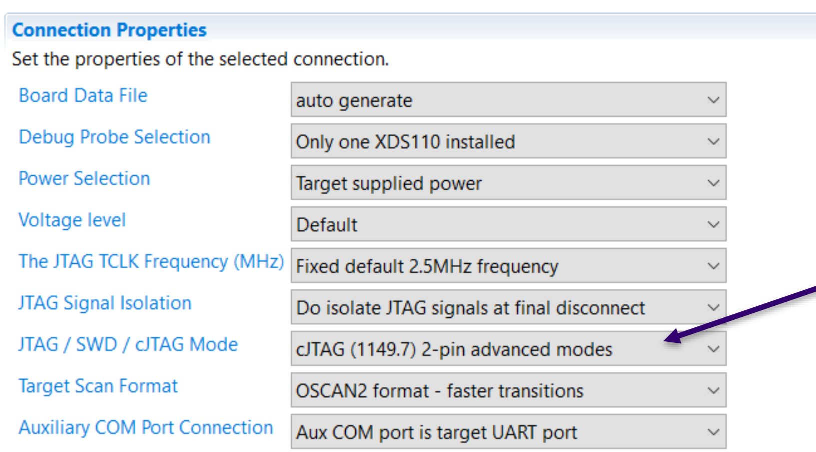
Click on this



Double click this



Make sure this is correct. Once it is, automatically the bottom two tabs would open.



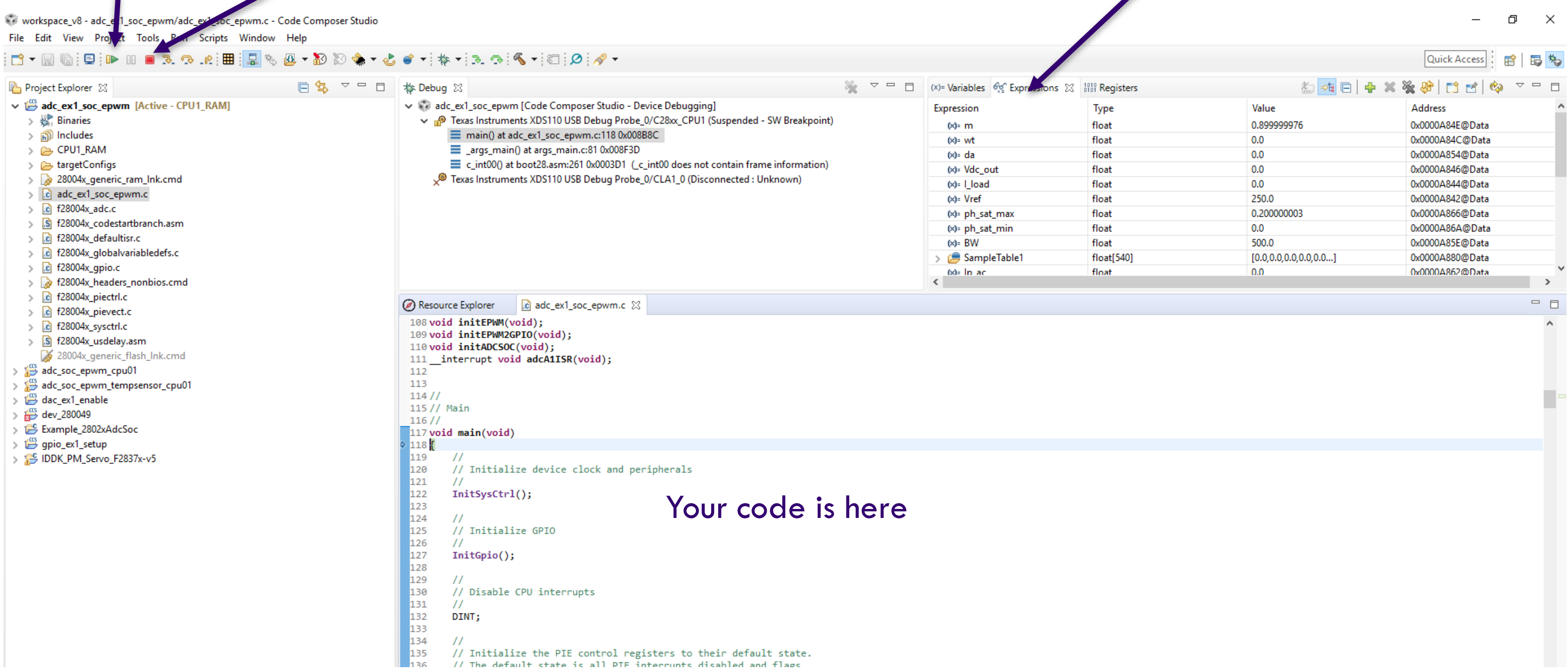
If it asks for Update, consent

Open the project titled : adc_ex1_soc_epwm

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

Click on this to start the code

Click this to stop code execution by CCS



Open the project titled : adc_ex1_soc_epwm

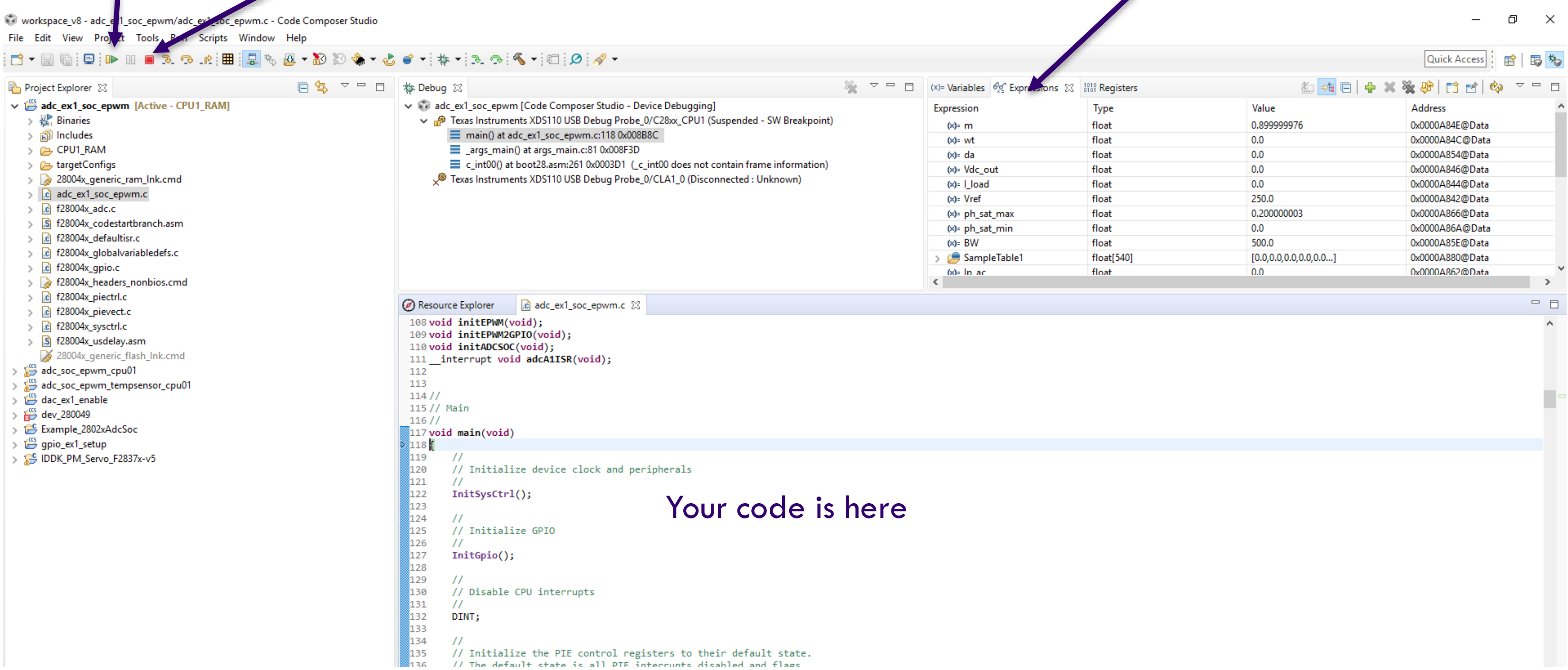
Click on this to start the code

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

```c
//
#include "F28x_Project.h"

//
// Defines
//
#define RESULTS_BUFFER_SIZE     256

//
// Globals
//
uint16_t adcAResults[RESULTS_BUFFER_SIZE];   // Buffer for results
uint16_t index;                              // Index into result buffer
volatile uint16_t bufferFull;                // Flag to indicate buffer is full

//
// Function Prototypes
//
void initADC(void);
void initEPWM(void);
void initADCSOC(void);
__interrupt void adcA1ISR(void);

//
// Main
//
void main(void)
{
    //
    // Initialize device clock and peripherals
    //
    InitSysCtrl();

    //
    // Initialize GPIO
    //
    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

```c
void initADC(void)
{
    //
    // Setup VREF as internal
    //
    SetVREF(ADC_ADCA, ADC_INTERNAL, ADC_VREF3P3);

    EALLOW;

    //
    // Set ADCCLK divider to /4
    //
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;

    //
    // Set pulse positions to late
    //
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //
    // Power up the ADC and then delay for 1 ms
    //
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    EDIS;

    DELAY_US(1000);
}
```

Sets up ADC timing and reference voltage

Sampling time

Code Description

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

```
void initEPWM(void)
{
    EALLOW;

    EPwm1Regs.ETSEL.bit.SOCAEN = 0;     // Disable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = 4;    // Select SOC on up-count
    EPwm1Regs.ETPS.bit.SOCAPRD = 1;     // Generate pulse on 1st event

    EPwm1Regs.CMPA.bit.CMPA = 0x0800;   // Set compare A value to 2048 counts
    EPwm1Regs.TBPRD = 0x1000;           // Set period to 4096 counts

    EPwm1Regs.TBCTL.bit.CTRMODE = 3;    // Freeze counter

    EDIS;
}
//
// initADCSOC - Function to configure ADCA's SOC0 to be triggered by ePWM1.
//
void initADCSOC(void)
{
    //
    // Select the channels to convert and the end of conversion flag
    //
    EALLOW;

    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 1;      // SOC0 will convert pin A1
                                            // 0:A0  1:A1  2:A2  3:A3
                                            // 4:A4   5:A5   6:A6   7:A7
                                            // 8:A8   9:A9   A:A10  B:A11
                                            // C:A12  D:A13  E:A14  F:A15
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 9;      // Sample window is 10 SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5;    // Trigger on ePWM1 SOCA

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0; // End of SOC0 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   // Enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Make sure INT1 flag is cleared

    EDIS;
```

Sets up a specific channel to do analog to digital conversion

Code Description

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

```
void initEPWM(void)
{
    EALLOW;

    EPwm1Regs.ETSEL.bit.SOCAEN = 0;     // Disable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = 4;    // Select SOC on up-count
    EPwm1Regs.ETPS.bit.SOCAPRD = 1;     // Generate pulse on 1st event

    EPwm1Regs.CMPA.bit.CMPA = 0x0800;   // Set compare A value to 2048 counts
    EPwm1Regs.TBPRD = 0x1000;           // Set period to 4096 counts

    EPwm1Regs.TBCTL.bit.CTRMODE = 3;    // Freeze counter

    EDIS;
}
//
// initADCSOC - Function to configure ADCA's SOC0 to be triggered by ePWM1.
//
```

Sets up a specific channel to do analog to digital conversion

```
void initADCSOC(void)
{
    //
    // Select the channels to convert and the end of conversion flag
    //
    EALLOW;

    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 1;     // SOC0 will convert pin A1
                                           // 0:A0  1:A1  2:A2  3:A3
                                           // 4:A4   5:A5   6:A6   7:A7
                                           // 8:A8   9:A9   A:A10  B:A11
                                           // C:A12  D:A13  E:A14  F:A15
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 9;     // Sample window is 10 SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5;   // Trigger on ePWM1 SOCA

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0; // End of SOC0 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   // Enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Make sure INT1 flag is cleared

    EDIS;
```

Code Description

```c
__interrupt void adcA1ISR(void)
{
    //
    // Add the latest result to the buffer
    // ADCRESULT0 is the result register of SOC0
    adcAResults[index++] = AdcaResultRegs.ADCRESULT0;

    //
    // Set the bufferFull flag if the buffer is full
    //
    if(RESULTS_BUFFER_SIZE <= index)
    {
        index = 0;
        bufferFull = 1;
    }

    //
    // Clear the interrupt flag
    //
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;

    //
    // Check if overflow has occurred
    //
    if(1 == AdcaRegs.ADCINTOVF.bit.ADCINT1)
    {
        AdcaRegs.ADCINTOVFCLR.bit.ADCINT1 = 1; //clear INT1 overflow flag
        AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    }

    //
    // Acknowledge the interrupt
    //
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

We will implement the controller here

This has the sampled value of feedback variable

Delete these

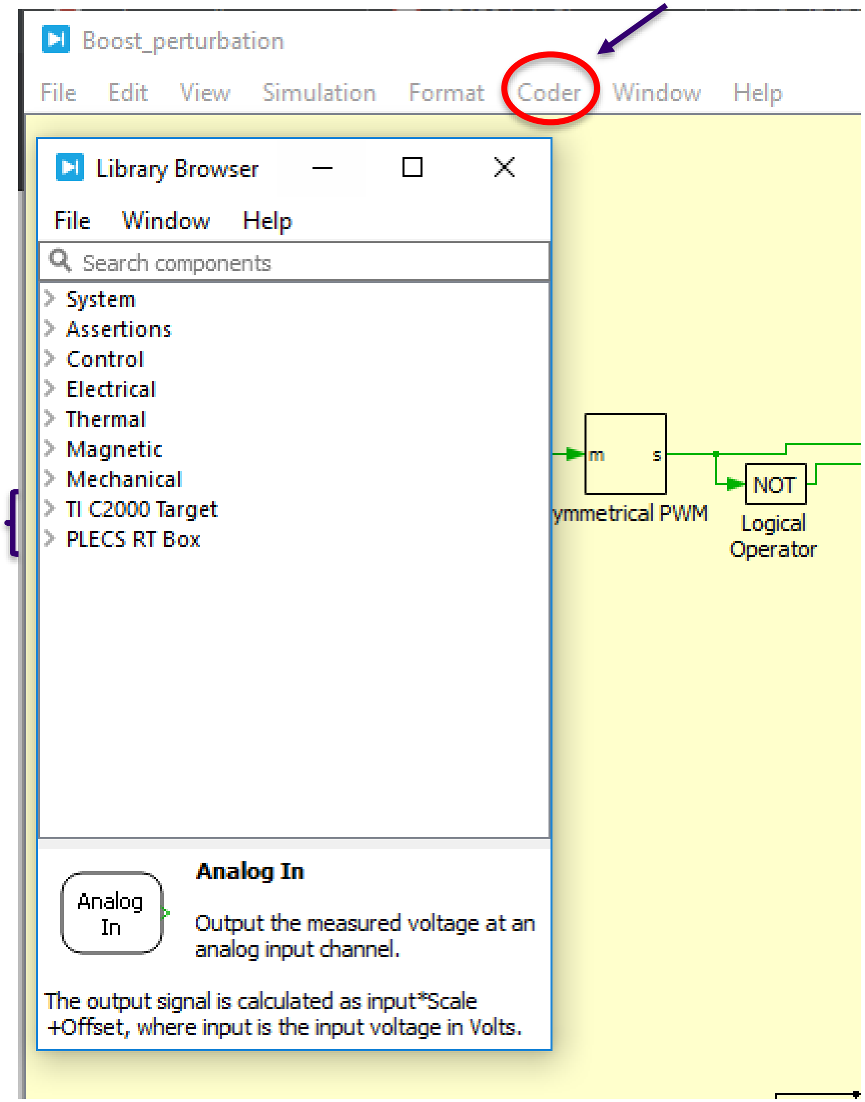
Getting started with PLECS RT-Box

- In this section, we will learn how to use PLECS RT-Box feature.
- PLECS RT-Box used with an interconnection board which will fit into it the launchpad
- We expect you to read the following
  - Read **LAUNCHXL-F280049C Pin Map** document before Experiment 2.
  - Understand all components of PLECS RT Box library

Getting started with PLECS RT-Box

New tab for RT Box

New libraries for RT Box

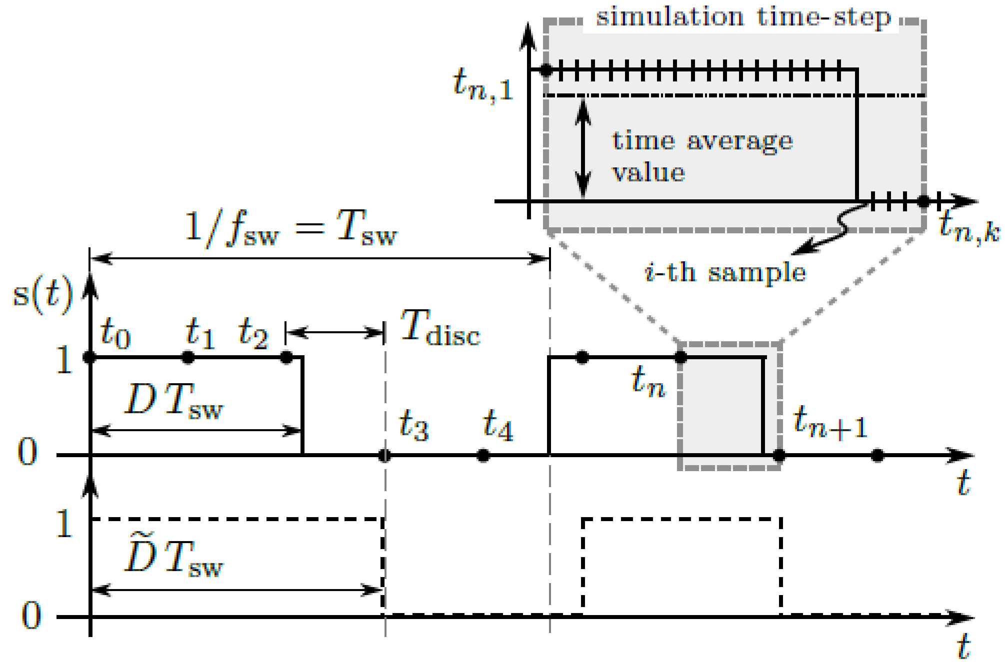


The power model needs the following changes

- Offline simulations run on variable time solvers and that is highly efficient.
- Error from fixed time solvers:

$$\left|D - \widetilde{D}\right| < \frac{T_{disc}}{T_{sw}}$$

- Real time simulation is a fixed step discretization operation and there is a chance we might run into this error.
- There are two approaches of solving this.



1. Method correcting state variables due to missed switching events once the event is detected by interpolation-extrapolation [Dinavahi,Iravani, Bonert '01]
2. Using time averaged models [Lian, Lehn '05]

Summary: We do not use switching models, but we do not also use averaged models

- We use "sub-cycle averaged models"
- Switching signal : $s'\left((n+1)T_{disc}\right) = \frac{1}{k}\sum s(\left(n+\frac{i}{k}\right)T_{disc})$
- Broadens b/w from $2/T_{disc}$ to $2k/T_{disc}$. So I get to see switching freq harmonics which are otherwise not visible in the avg models
- k (*sub-sampling rate*) is the number of samples in the T_{disc}. Maximum value of k would depend on the FPGA clock.
- Typ commercial RTDS runs in wit T_{disc} of 7.5 ns
- Thereafter we can use any popular average model of converters. [Middlebrook '73, Maksimovic,Stankovic, Thottuvelil, Verghese '01,]
- These models show almost same performance as an offline variable time switched models solution.