

# EE 458 Lab 1 Presentation



Rahul Mallik  
([rmallik@uw.edu](mailto:rmallik@uw.edu))



# 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.



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

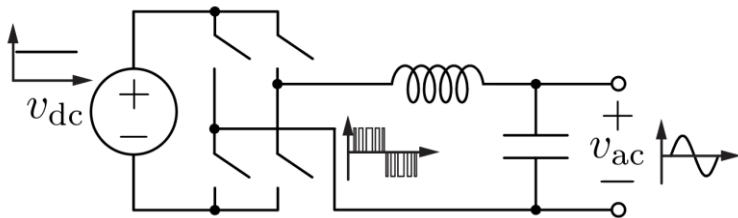


$$v = L \frac{di}{dt}$$



$$v = iR$$

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



A power converter is made up of many such equations

$$\begin{aligned}\dot{x} &= f(x, t) \\ g(x, t) &= 0\end{aligned}$$



# The idea of time step

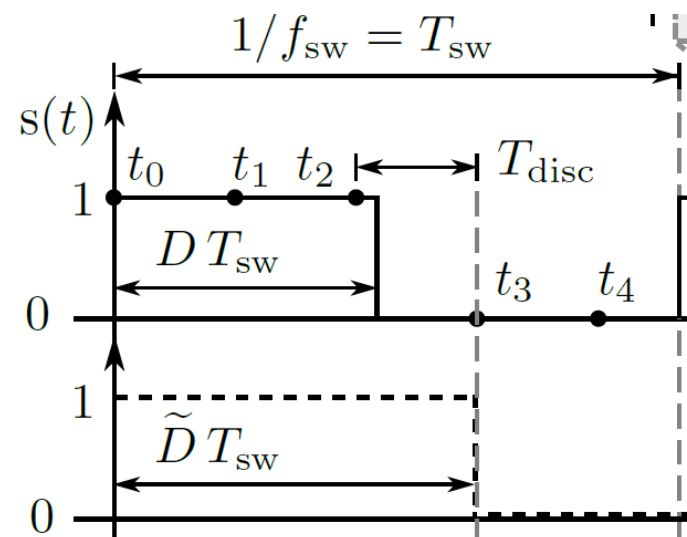
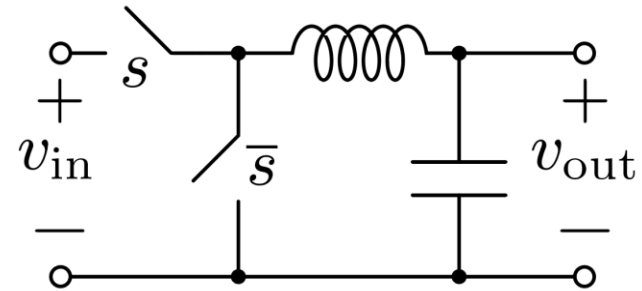
Let us say a buck converter is switched

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 :  $|D - \tilde{D}|$

From fig, we can say :  $|D - \tilde{D}| < \frac{T_{disc}}{T_{sw}}$



# How small can the time step be?

In simulation?

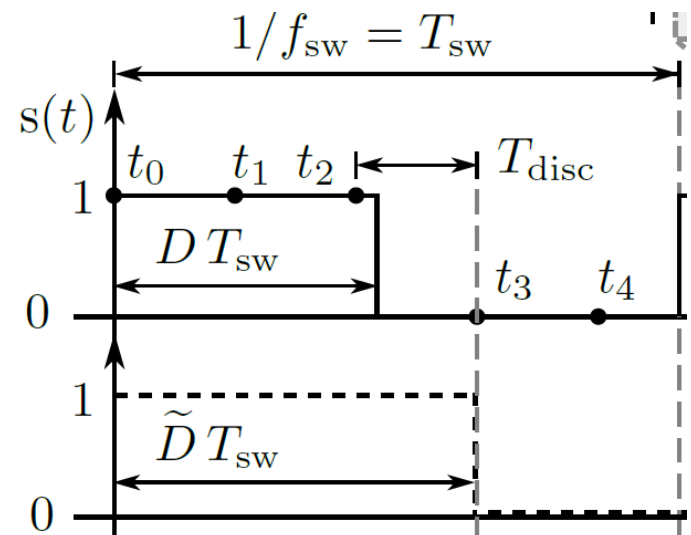
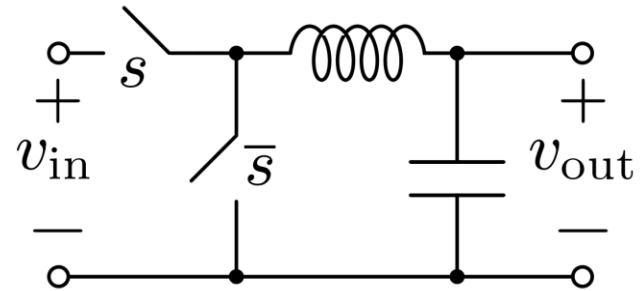
Make time step as small as possible, say 1ns, it still works.

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.

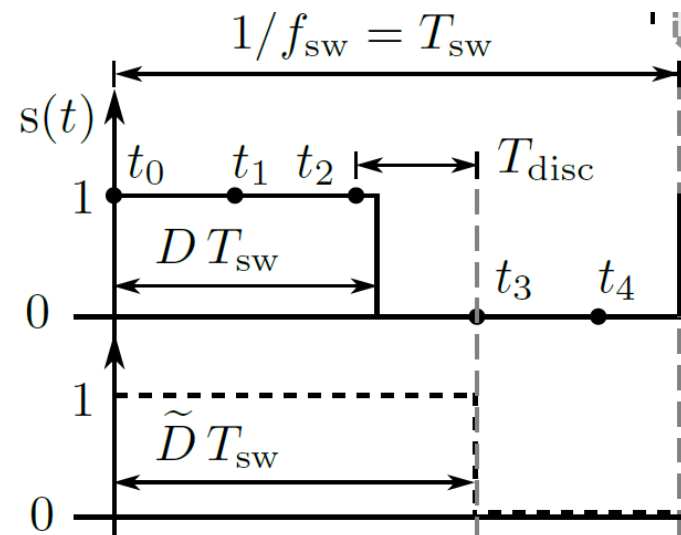
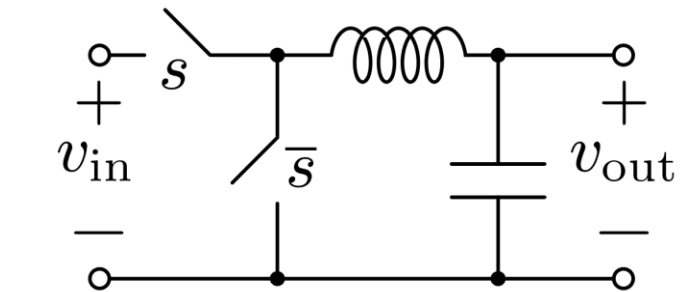
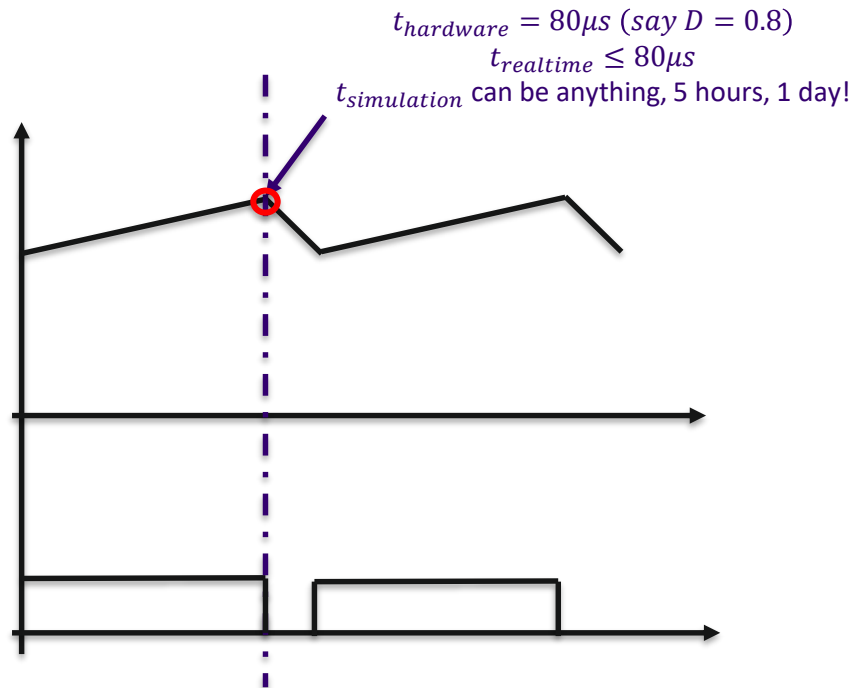
In PLECS simulation, this might take about an hour, to get a whole 1s simulation to complete



W

# Idea of real time

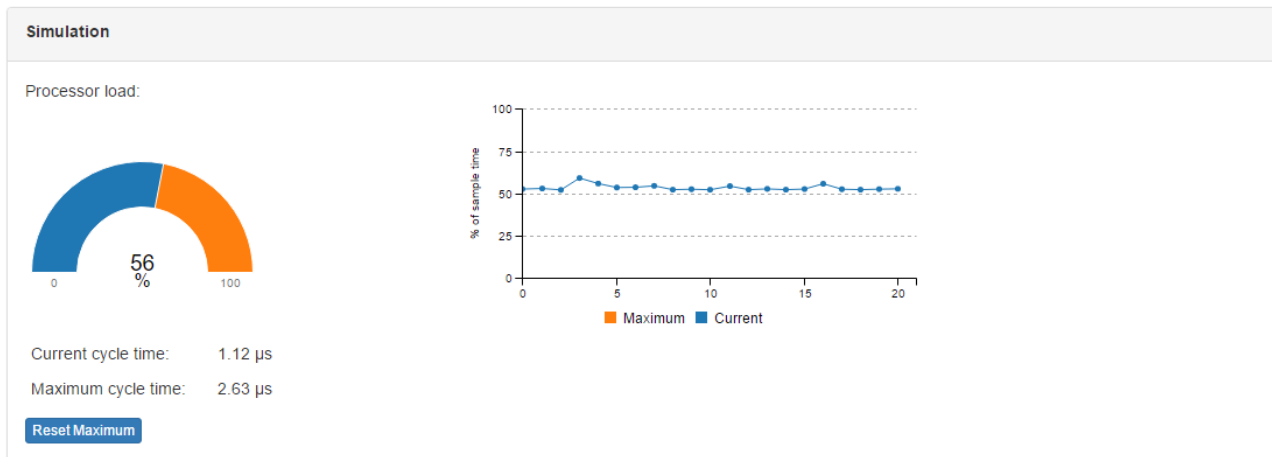
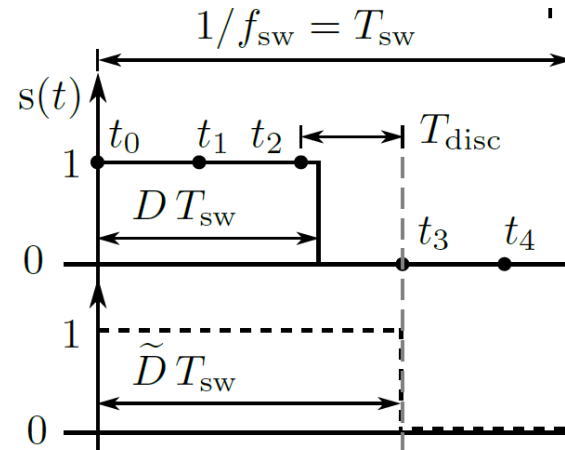
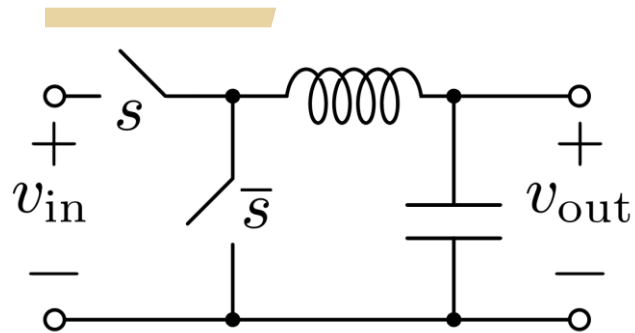
In this same simulation, what we want is, the following: Same waveforms at same time!



- 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



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

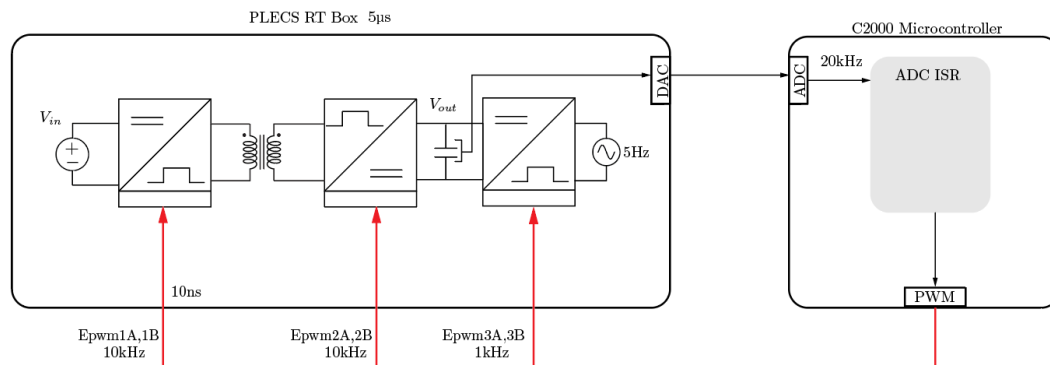
$$T_{disc} = 5\mu s$$

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.

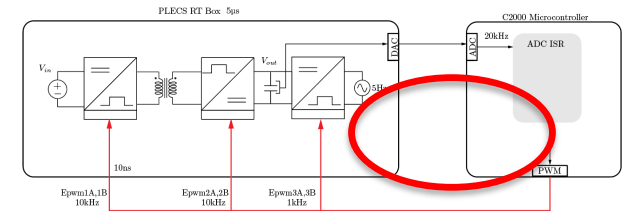


# Hardware-in-loop

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



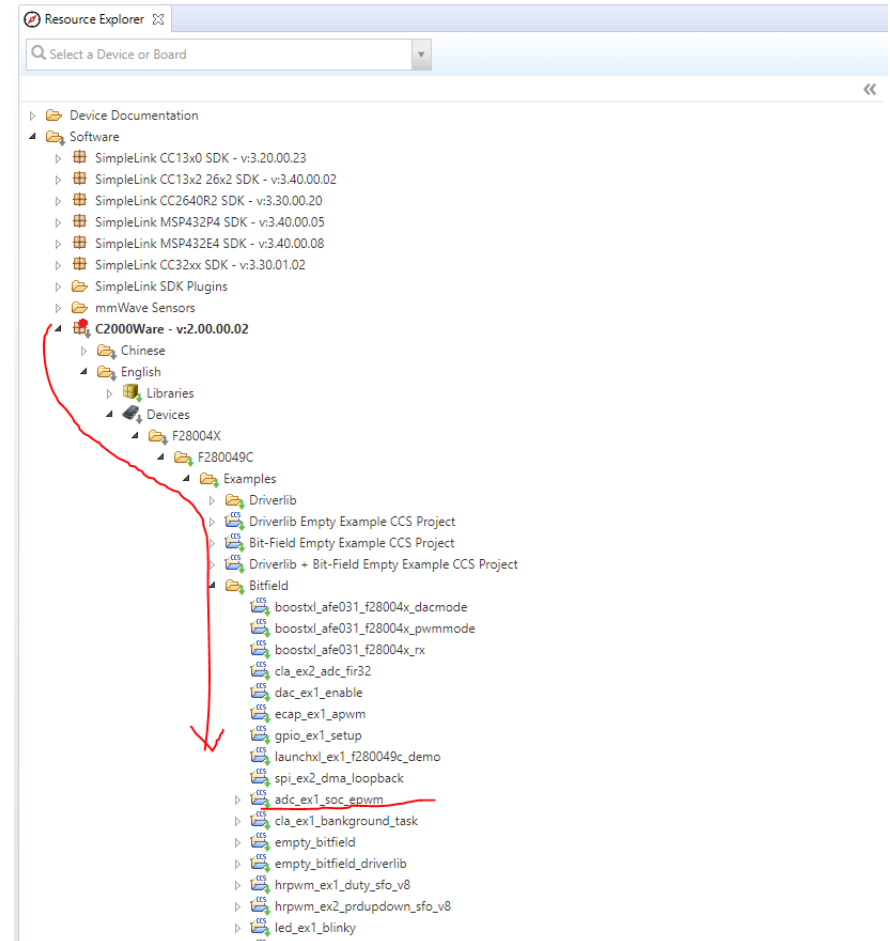
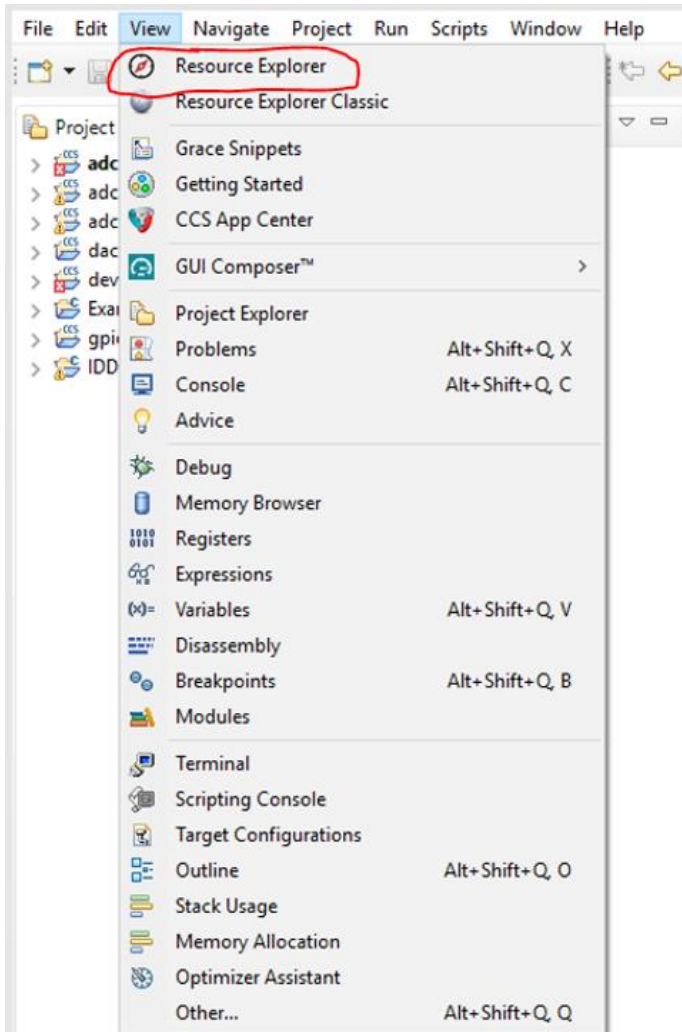
# Getting started with CCS



- 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)

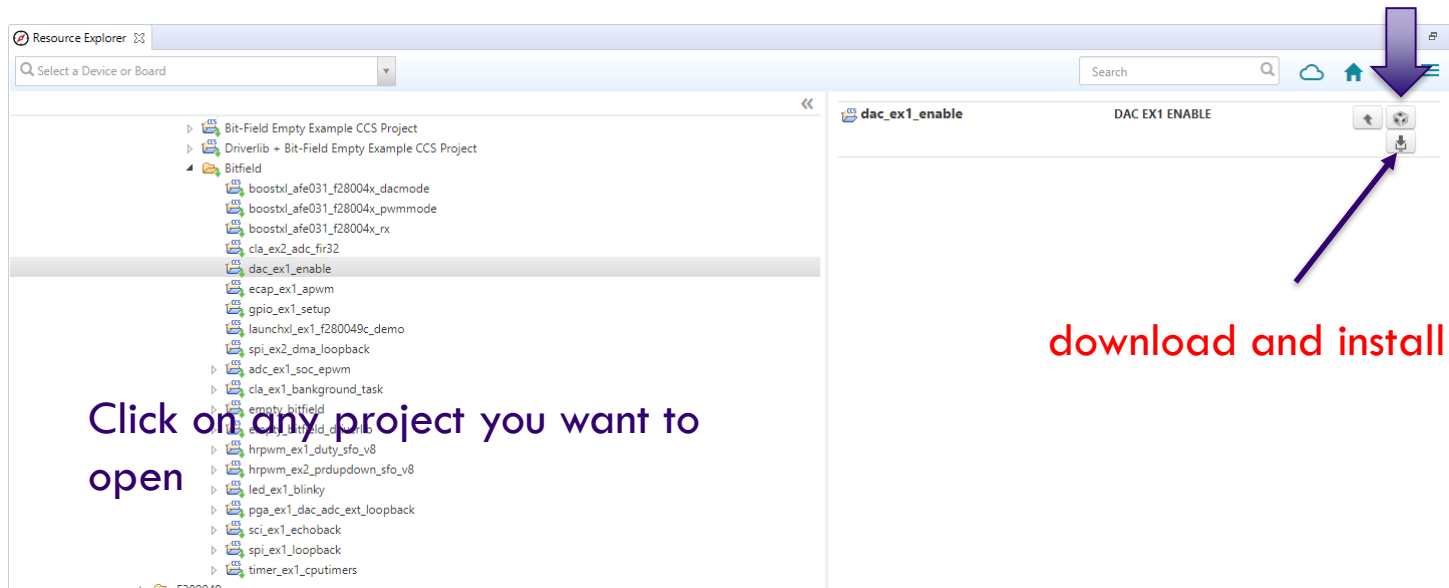


# Creating a new project : Open an existing project from the Resource Explorer

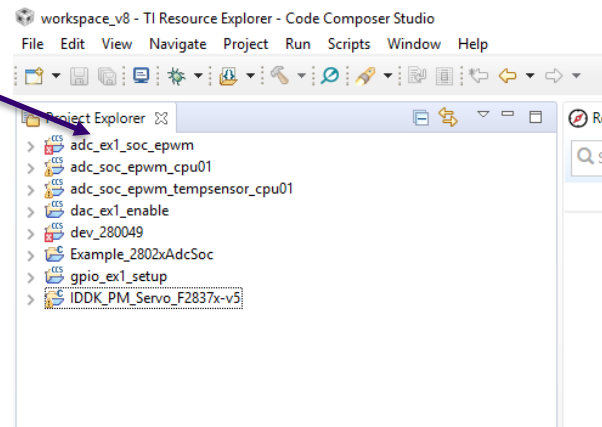


# Open the project titled : adc\_ex1\_soc\_epwm

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 )

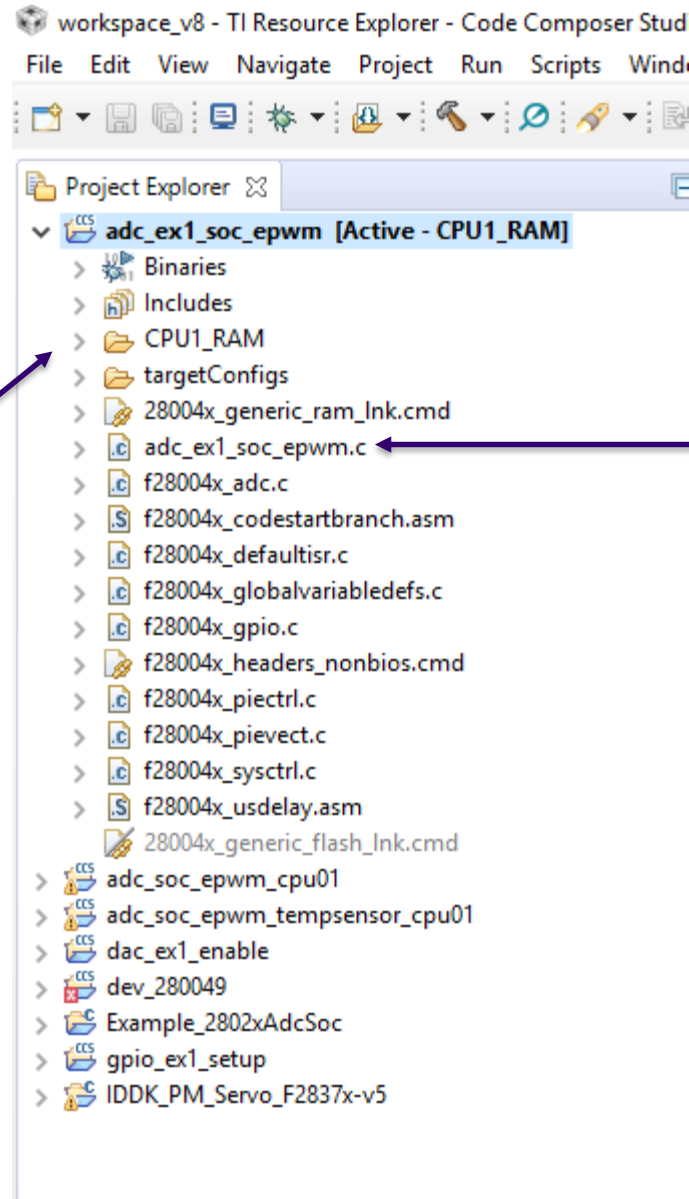


Now find the project in this list. A cross means that project has compilation or debug errors



# W

# 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

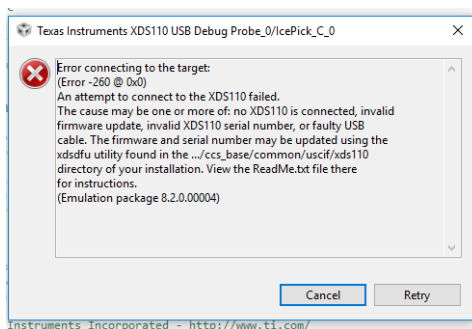
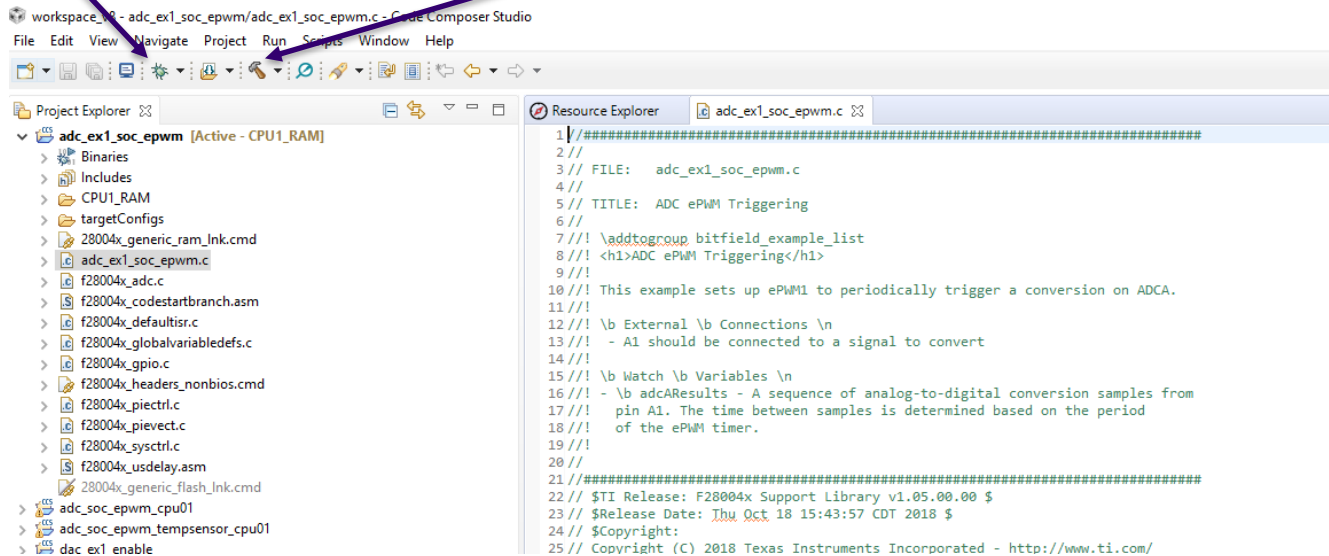


# Open the project titled : adc\_ex1\_soc\_epwm

Step 3: Click this (bug) to debug

Step 1: Click this (hammer) to compile

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



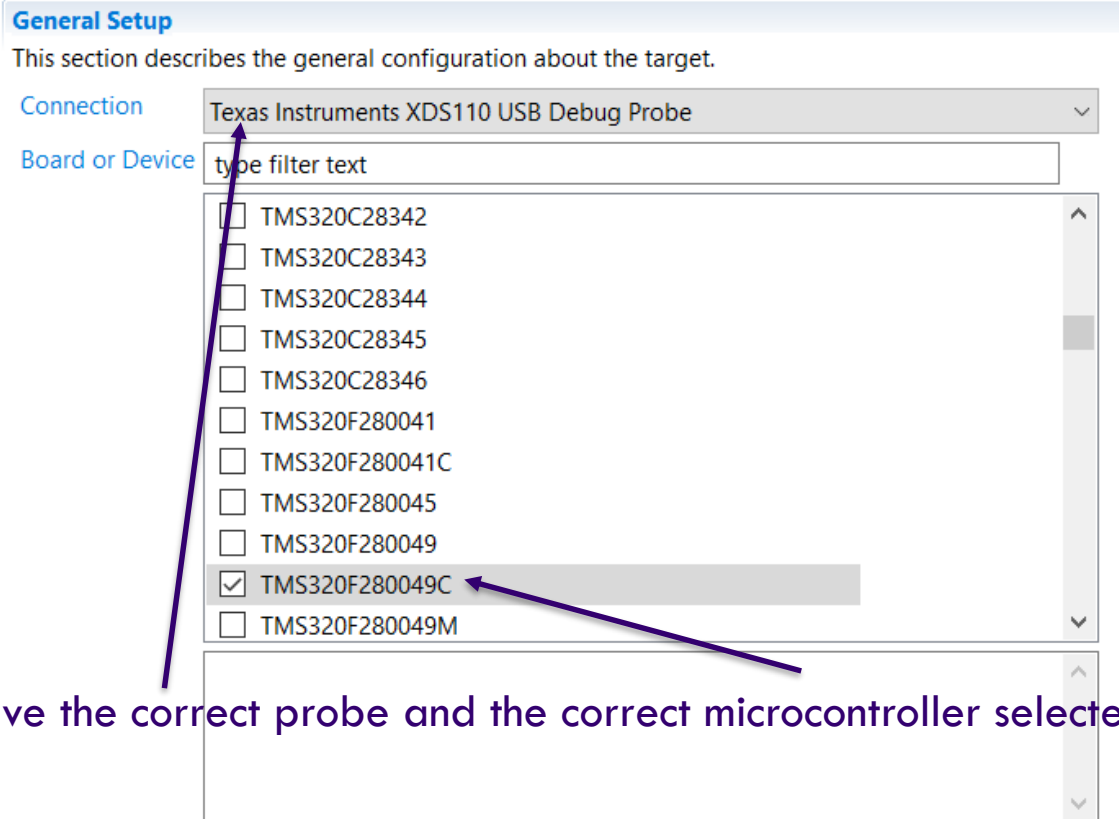
Getting this?  
Check step #2

W

# 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.

## 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
- ☐ TMS320C28344
- ☐ TMS320C28345
- ☐ TMS320C28346
- ☐ TMS320F280041
- ☐ TMS320F280041C
- ☐ TMS320F280045
- ☐ TMS320F280049
- ☒ TMS320F280049C
- ☐ TMS320F280049M

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

# W



# 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

# W

# 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.

The screenshot displays the Code Composer Studio (CCS) interface. The Project Explorer on the left shows the project structure for 'adc\_ex1\_soc\_epwm'. The Debug Console in the center shows the execution status. The Watch Window on the right displays a list of variables and their values.

**Project Explorer:**

- adc\_ex1\_soc\_epwm [Active - CPU1\_RAM]
  - Binaries
  - Includes
  - CPU1\_RAM
  - targetConfigs
  - 28004x\_generic\_ram\_lnk.cmd
  - adc\_ex1\_soc\_epwm.c
  - f28004x\_adc.c
  - f28004x\_codeloadbranch.asm
  - f28004x\_defaultisr.c
  - f28004x\_globalvariabledefs.c
  - f28004x\_gpio.c
  - f28004x\_headers\_nonbios.cmd
  - f28004x\_piectrl.c
  - f28004x\_pievect.c
  - f28004x\_sysctrl.c
  - f28004x\_usdelay.asm
  - 28004x\_generic\_flash\_lnk.cmd
  - adc\_soc\_epwm\_cpu01
  - adc\_soc\_epwm\_tempsensor\_cpu01
  - dac\_ex1\_enable
  - dev\_280049
  - Example\_2802xAdcSoc
  - gpio\_ex1\_setup
  - IDDK\_PM\_Servo\_F2837x-v5

**Debug Console:**

```
adc_ex1_soc_epwm [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe_0/C28x_CPU1 (Suspended - SW Breakpoint)
    main() at adc_ex1_soc_epwm.c:118 0x008B8C
    _args_main() at args_main.c:81 0x008F3D
    _c_int00 at boot28.asm:261 0x0003D1 _c_int00 does not contain frame information
  Texas Instruments XDS110 USB Debug Probe_0/CLA1_0 (Disconnected : Unknown)
```

**Watch Window:**

Expression	Type	Value	Address
(*) m	float	0.899999976	0x0000A84E@Data
(*) wt	float	0.0	0x0000A84C@Data
(*) da	float	0.0	0x0000A844@Data
(*) Vdc_out	float	0.0	0x0000A846@Data
(*) I_load	float	0.0	0x0000A844@Data
(*) Vref	float	250.0	0x0000A842@Data
(*) ph_sat_max	float	0.200000003	0x0000A866@Data
(*) ph_sat_min	float	0.0	0x0000A86A@Data
(*) BW	float	500.0	0x0000A85E@Data
> SampleTable1	float[540]	[0.0,0.0,0.0,0.0,0.0,...]	0x0000A880@Data
(*) In_Ar	float	0.0	0x0000A862@Data

**Resource Explorer:**

```
108 void initEPWM(void);
109 void initEPWM2GPIO(void);
110 void initADCSOC(void);
111 __interrupt void adcA1ISR(void);
112
113
114 //
115 // Main
116 //
117 void main(void)
118 {
119     //
120     // Initialize device clock and peripherals
121     //
122     InitSysCtrl();
123
124     //
125     // Initialize GPIO
126     //
127     InitGpio();
128
129     //
130     // Disable CPU interrupts
131     //
132     DINT;
133
134     //
135     // Initialize the PIE control registers to their default state.
136     // The default state is all PIE interrupts disabled and flags
```

Your code is here

# W

# 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.

The screenshot displays the Code Composer Studio (CCS) interface. The Project Explorer on the left shows the project structure for 'adc\_ex1\_soc\_epwm'. The Debug Console in the center shows the execution status. The Watch Window on the right displays a list of variables and their values.

**Project Explorer:**

- adc\_ex1\_soc\_epwm [Active - CPU1\_RAM]
  - Binaries
  - Includes
  - CPU1\_RAM
  - targetConfigs
  - 28004x\_generic\_ram\_lnk.cmd
  - adc\_ex1\_soc\_epwm.c
  - f28004x\_adc.c
  - f28004x\_codeloadbranch.asm
  - f28004x\_defaultisr.c
  - f28004x\_globalvariabledefs.c
  - f28004x\_gpio.c
  - f28004x\_headers\_nonbios.cmd
  - f28004x\_piectrl.c
  - f28004x\_pievect.c
  - f28004x\_sysctrl.c
  - f28004x\_usdelay.asm
  - 28004x\_generic\_flash\_lnk.cmd
  - adc\_soc\_epwm\_cpu01
  - adc\_soc\_epwm\_tempsensor\_cpu01
  - dac\_ex1\_enable
  - dev\_280049
  - Example\_2802xAdcSoc
  - gpio\_ex1\_setup
  - IDDK\_PM\_Servo\_F2837x-v5

**Debug Console:**

```
adc_ex1_soc_epwm [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe_0/C28x_CPU1 (Suspended - SW Breakpoint)
    main() at adc_ex1_soc_epwm.c:118 0x008B8C
    _args_main() at args_main.c:81 0x008F3D
    _c_int00 at boot28.asm:261 0x0003D1 _c_int00 does not contain frame information
  Texas Instruments XDS110 USB Debug Probe_0/CLA1_0 (Disconnected : Unknown)
```

**Watch Window:**

Expression	Type	Value	Address
(*) m	float	0.899999976	0x0000A84E@Data
(*) wt	float	0.0	0x0000A84C@Data
(*) da	float	0.0	0x0000A844@Data
(*) Vdc_out	float	0.0	0x0000A846@Data
(*) I_load	float	0.0	0x0000A844@Data
(*) Vref	float	250.0	0x0000A842@Data
(*) ph_sat_max	float	0.200000003	0x0000A866@Data
(*) ph_sat_min	float	0.0	0x0000A86A@Data
(*) BW	float	500.0	0x0000A85E@Data
> SampleTable1	float[540]	[0.0,0.0,0.0,0.0,0.0...]	0x0000A880@Data
(*) In_Ar	float	0.0	0x0000A862@Data

**Resource Explorer:**

```
108 void initEPWM(void);
109 void initEPWM2GPIO(void);
110 void initADCSOC(void);
111 __interrupt void adcA1ISR(void);
112
113
114 //
115 // Main
116 //
117 void main(void)
118 {
119     //
120     // Initialize device clock and peripherals
121     //
122     InitSysCtrl();
123
124     //
125     // Initialize GPIO
126     //
127     InitGpio();
128
129     //
130     // Disable CPU interrupts
131     //
132     DINT;
133
134     //
135     // Initialize the PIE control registers to their default state.
136     // The default state is all PIE interrupts disabled and flags
```

Your code is here

# W

# Code Description

```
55 //
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.ADCPDNZ = 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 }
```

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

```
259 //
260 // initADC SOC - Function to configure ADCA's SOC0 to be triggered by ePWM1.
261 //
262 void initADC SOC(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.INT1IE = 1;  // Enable INT1 flag
279     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Make sure INT1 flag is cleared
280
281     EDIS;
```

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 }
```

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

```
259 //
260 // initADC SOC - Function to configure ADCA's SOC0 to be triggered by ePWM1.
261 //
262 void initADC SOC(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.INT1IE = 1;  // Enable INT1 flag
279     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Make sure INT1 flag is cleared
280
281     EDIS;
```

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     // Set the bufferFull flag if the buffer is full
295     //
296     if (RESULTS_BUFFER_SIZE <= index)
297     {
298         index = 0;
299         bufferFull = 1;
300     }
301     //
302     // Clear the interrupt flag
303     //
304     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
305     //
306     // Check if overflow has occurred
307     //
308     if (1 == AdcaRegs.ADCINTOVF.bit.ADCINT1)
309     {
310         AdcaRegs.ADCINTOVFCLR.bit.ADCINT1 = 1; //clear INT1 overflow flag
311         AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
312     }
313     //
314     // Acknowledge the interrupt
315     //
316     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
317 }
318
```

← 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 screenshot displays the PLECS software interface. The main window has a menu bar with 'File', 'Edit', 'View', 'Simulation', 'Format', 'Coder', 'Window', and 'Help'. The 'Coder' tab is highlighted with a red circle and a purple arrow pointing to it from the text 'New tab for RT Box'. Below the menu bar is a yellow workspace area containing a block diagram. The diagram shows a block labeled 'm' (symmetrical PWM) connected to a block labeled 's' (Logical Operator), which is then connected to a 'NOT' block. To the left of the workspace is a 'Library Browser' window. It has a search bar and a list of component categories: System, Assertions, Control, Electrical, Thermal, Magnetic, Mechanical, TI C2000 Target, and PLECS RT Box. A purple arrow points from the text 'New libraries for RT Box' to the 'PLECS RT Box' entry in the list. At the bottom of the 'Library Browser' window, there is a section titled 'Analog In' with a block icon and a description: 'Output the measured voltage at an analog input channel. The output signal is calculated as input\*Scale +Offset, where input is the input voltage in Volts.'

W

# The power model needs the following changes

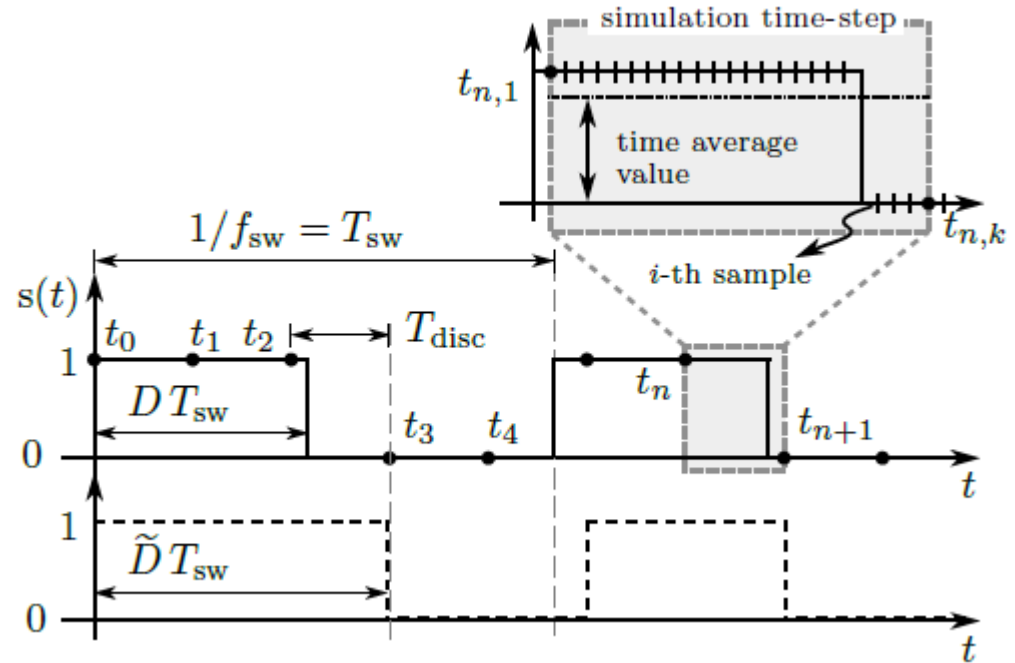
- Offline simulations run on variable time solvers and that is highly efficient.

- Error from fixed time solvers:

$$|D - \tilde{D}| < \frac{T_{disc}}{T_{sw}}$$

- 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 :  $s'((n+1)T_{disc}) = \frac{1}{k} \sum s((n + \frac{i}{k})T_{disc})$
- > 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.

