Using DSP Designer Code Generators for the TI Piccolo DSP [5]

Background: DSP Designer builds upon a large technological base. It assumes that you are familiar with control system stability analysis using Bode Plots [1]. Moreover, you need some background in using z-transforms for the controller functions [2]. The models shown in the schematics are built upon large signal average models that are described in technical literature [3]. These models dispense with cycle-by-cycle switching in favor of an average approach that models the control system up to ½ the sampling frequency. Average models speed the simulation time and in many cases the simulation runs faster than real time. Using the ICAP/4 schematic, SpiceNet, requires studying the ICAP/4 getting started manual [4]. The result is very impressive, with one-button Bode Plots and transient simulations running considerably faster than real time and automatic code generation. This document will lead you through the details of operating the ICAP/4 user interface to accomplish these tasks; however, you need to be versed in the background technology in order for these tools to be used to their greatest potential.

Schematic: A DSP controller can be embedded in an ICAP/4 schematic. That's done by using the configuration capabilities of SpiceNet. First you need to add 2 layers for each controller in the schematic by selecting the "Options" menu, "Layer…". One layer will contain interface voltage sources and node names. The other layer consists of the z-transform controller portion of the schematic. Then you must make a configuration using the "Options", "Configurations…" menu for the "ExportDSP…" operation that uses only these 2 layers. The other configurations keep the controller parts but won't include the interface parts. Additional layers may be needed for more complex controllers. The name given to the configuration will be used by the code generator to make the various ".c" , ".h" and assembly files.

Files:

<Config name>.h	Definitions
<Config name>.c	Initialization
<Config name+a>.asm	Assembly Code
hardware.h	Hardware specific definitions

It is assumed the main portion of the DSP code is in the C programming language. You will need to add the appropriate initialization to that code along with any special code to detect faults and assist startup. Templates shipped with DSP Designer include this code along with code necessary for Real Time Communication. The advantage of a C Language based approach is in the ease of management of variable and data structures. Assembly code is only used in the high-speed interrupt service routines (ISR's).

Sample Drawings: The DSP control algorithm in MPID2.dwg is contained in the "MPID2" configuration. This configuration obeys the rules required for code generation. It is also used in the "main" configuration where SPICE simulation gives performance analysis using the AC, DOP and TRAN analyses. To operate the schematic, use the file explorer to go to the project folder "…\Piccolo controlSTICK\DualSyncBuckBoost" and double click on MDIP2.dwg. The toolbar should look like Figure 1, showing the MPID2 configuration and the DSP analysis. The configuration will be represented by <config_name> in the following discussion. You can change the name of <config_name> by pressing the button to the left of the text box and then the edit button.

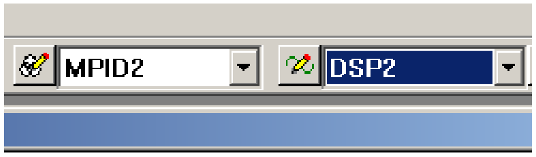
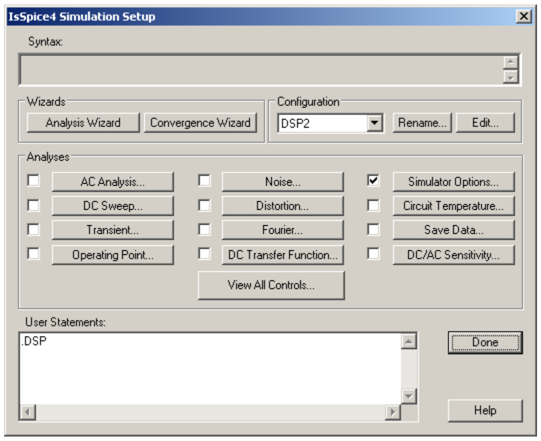


Figure 1, MPID2 configuration and DSP analysis

This is the proper configuration for code generation. Pressing the button to the left of the DSP2 analysis brings up the Simulation setup dialog shown in figure 2.



Simulator Options [✓]

User Statements:
```
.DSP
```

Figure 2, Simulation setup for the .dsp analysis.

Notice that the User Statement field contains the SPICE directive for code generation. The syntax is .DSP [arglist]. .DSP is an analysis type, just like AC or TRAN. The [arglist] is an optional list of white space delimited nodes, telling SPICE to order the matrix solution such that these nodes appear first in the matrix backward substitution or solution order (the matrix is ordered in the reverse of the arglist). IsSpice4 understands this syntax and produces a triangular matrix using gauss elimination that has the following properties:

- Numbered nodes are first in the state variable order
- Then non-trivial states (equations with more than one coefficient)
- Inputs, trivial states, Z-delay inputs

The critical factor in using this matrix solution approach is that the system of equations is linear so that the matrix right-hand-side, RHS is zero for everything except the trivial nodes. *Then the matrix coefficients can be used over and over for each solution time step.* Before proceeding with the solution, the ADC inputs and other inputs must be fetched. After the solution, the Zdelay inputs must propagate to their respective outputs. By placing the numbered nodes first, their solution can be bypassed, effectively eliminating them from the matrix. All of this is done by the code generator if you follow certain rules for the code generator configuration.

Rules:

- State Variables are identified as IsSice4 node names
- Parts: Use only V-source, B-element, and Zdelay components.
- Nodes
  - Numbered, these states will not be calculated
  - zi prefix for Zdelay inputs
  - zo prefix for Zdelay outputs
  - imbedded pwmx, identifies a pwm output for channel x (1,2,3,4)
  - indx postfix, identifies an ADC input where x=(0,1,2,…)
- Parameters
  - PWMPERIOD, integer size of PWM for unity duty ratio
  - radix, number of fractional bits in MAC computation
  - refx, reference for indx (refx-ADCx)
  - LIMIT_HI+node, Hi Limit
  - LIMIT_LOI+node, Lo Limit
  - <config_name>_Statei.zox, init value for node x

Parameters are algebraic calculations done in the order present in the Parameters block shown on the schematic. The .par files shows their evaluation and is used by the code generator to define state variable limits and the reference for the input error signal and the PWM scale factor. You can change the ".par" file accuracy placing:

```
.option paramdigits="# of digits for .par file"
```

in the User Statements field along with the .DSP directive shown in Figure 2..

An important part of the DSP code is the re_init_<config_name> function. This function calculates the proper integrator initialization after a successful ADC measurement. It's call can be found in main.c

Special cases: Connecting a Zdelay output to another Zdelay input needs to use a B-element with gain = 1 so that the prefix rules are obeyed. If an indx postfix is used, the result is (refx - <prefix>indx) and limiting is applied to that result. These rules are error checked prior to running the code generator. If errors exist, a dialog will explain problems and the code generation will be aborted. Detailed error description is in the ".err" file.

Code Generator Results: Selecting "Export DSP …" in the file menu brings up code generation options. There are two TI code generators. Both produce the same assembly code; however, the register locations for the A/D converter and PWM outputs are different. The 2812x part is much faster and more costly than the Piccolo part. The discussion here will be for the Piccolo part. Code generation begins by selecting TI 28207x Piccolo. When completed successfully, there will be 3 files added to your project folder. The base name is derived from the configuration name so for this case the files are called MPID2.c, MPID2.h and MPID2a.asm. It's important to make the .asm file name different from the .c file name because the TI compiler produces 2 object or .o files, one from the .c source and one from the .asm source. To use this code you must do the following:

- Add include "MPID2.h" to main.c
- Add include "MPID2.h" to ISR.c
- Add init_MPID2() to the end of the main.c initialization
- Add MPID2(); in the ISR.c file
- Add MPID2.c and MPID2a.asm to your project.

These additions have already been incorporated for this example, but it's a process you must follow for different code generator files. Remember, the code generator will over-write these files so that you may want to rename them. Using the wrong code generator could damage your hardware.

Testing the Code Generator: It's a good idea to step through the code in the debugger before running it in real time. To prevent damage to the power components, you should

1. remove power from the PWM inputs and

2. uncomment the #define DISABLE_UVLO in configure.h.

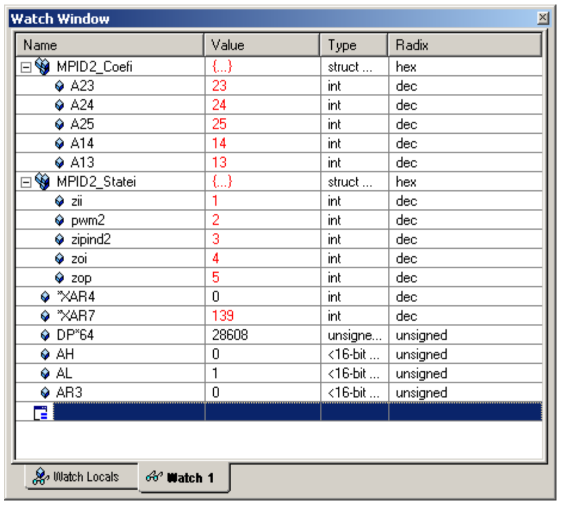
That disables the under voltage lockout so that the control algorithm will run. Then place a breakpoint at MPID2(); in ISR.c. Then from the IDE:

1. Project\Rebuild all,
2. Debugger\connect and
3. file\ Load Program… debug\DualSyncBuskBoost.out.

Be aware that IsSpice produces a ".out" file in the project. You must use the ".out" file produced by the code composer studio IDE located in the release or debug folder. Then

4. Debug\Go Main followed by
5. F5 to run the code.
6. Step into the assembly code, stopping at ADDB XAR4, #2.
7. Then open a watch window and enter values shown in Figure 3.

That initializes the values to something unique so that you can confirm the proper MAC operations as you step through the code. The comment before the MAC instruction in file "MPIDa.asm" indicates the state that should be pointed to by XAR4 and the coefficient pointed to by XAR7. Coefficients are pretty easy since they should step through in order. States are also straightforward when the matrix is being solved from left to right (pState++). However, the TI DSP doesn't do post decrement, only pre decrement so that *--XAR4 points to the state at *XAR4 and the one being use is at XAR4-1.


Watch Window

Name	Value	Type	Radix
MPID2_Coefi	{...}	struct ...	hex
A23	23	int	dec
A24	24	int	dec
A25	25	int	dec
A14	14	int	dec
A13	13	int	dec
MPID2_Statei	{...}	struct ...	hex
zii	1	int	dec
pwm2	2	int	dec
zipind2	3	int	dec
zoi	4	int	dec
zop	5	int	dec
*XAR4	0	int	dec
*XAR7	139	int	dec
DP*64	28608	unsigne...	unsigned
AH	0	<16-bit ...	unsigned
AL	1	<16-bit ...	unsigned
AR3	0	<16-bit ...	unsigned

Figure 3, initial values.

Once you are satisfied that the code is correct (This example has been tested and is correct) you can proceed to run in real time.

1 Comment out DISABLE_UVLO in file "configure.h" and
2. Remove Comment from SAFE_START

SAFE_START will turn the PWM's off after approximately 3 msec if the error signal is outside of limits specified in main.c.

References

[1] H.W.Bode, "Network Analysis and Feedback Amplifier Design"Princeton, NJ: Van'Nostrand, p. 10, 1945…. Or any modern text on network analysis.

[2] Power Specialist's App Note Book www.intusoft.com/lit/psbook.pdf pg 48,Average Models For Switching Converter

[3] AUTOMATING DSP DESIGN USING AUGMENTED SPICE SIMULATION, www.intusoft.com/DSPTechnicalArticles.htm

[4] SpiceNet help\PDF Files or www.intusoft.com/lit/GetStarted.pdf

[5] TMS320C28x CPU and Instruction Set Reference Guide http://focus.ti.com/lit/ug/spru430e/spru430e.pdf