

Application Note

C2000 SysConfig



Nima Eskandari

ABSTRACT

C2000 SysConfig is a powerful graphical user interface tool which configures the C2000 Real-Time Control MCUs and auto-generates embedded software, visualization diagrams, and debug artifacts. The reliable and validated initialization software generated by the C2000 SysConfig tool can significantly speed up development and help designers avoid lengthy debug sessions.

C2000 SysConfig and its PinMux tool closes the gap between hardware and software designers.

Table of Contents

1 Introduction	2
2 Getting Started With C2000 SysConfig	3
2.1 Example C2000 SysConfig in CCS.....	3
2.2 Other SysConfig Tools.....	6
3 C2000 SysConfig Overview	7
3.1 Modules.....	9
3.2 PinMux.....	10
3.3 Peripheral Initialization.....	13
3.4 Code Generation.....	14
3.5 Error Detection.....	16
3.6 SysConfig Script File.....	18
4 SysConfig Generated Files After the Project is Built	19
5 Application Code Based on C2000 SysConfig Initialization	20
6 Interrupt Support	21
7 Device-Specific Code Generation	24
8 Adding C2000 SysConfig Support to Existing Projects	25
9 Remove C2000 SysConfig Support from Projects	25
10 Standalone SysConfig Tool	26
11 Summary	26
12 References	27

List of Figures

Figure 1-1. C2000 SysConfig Overview.....	2
Figure 2-1. C2000 SysConfig Example Project in CCS.....	3
Figure 2-2. Launching SysConfig GUI.....	4
Figure 2-3. C2000 SysConfig GUI.....	4
Figure 2-4. Device View.....	5
Figure 2-5. CLB and DCSM (Security) Tool.....	6
Figure 3-1. CCS SysConfig Project Properties.....	7
Figure 3-2. SysConfig Basic Options.....	8
Figure 3-3. SysConfig Miscellaneous Options.....	8
Figure 3-4. C2000 SysConfig Modules and Resource Management.....	9
Figure 3-5. PinMux Submodule.....	10
Figure 3-6. PinMux Custom Use Case.....	10
Figure 3-7. Device Pin Representation.....	11
Figure 3-8. PinMux Summary CSV Auto-Generated File.....	12
Figure 3-9. <code>pinmux.csv</code> Auto-Generated File.....	13
Figure 3-10. SCI Configurable Options.....	13
Figure 3-11. Auto-Generated Content.....	14

Trademarks

Figure 3-12. Changed Configurable Code Generation DIFF.....	15
Figure 3-13. board.h File.....	15
Figure 3-14. Error Detection.....	16
Figure 3-15. Dependent Module Error Detection.....	17
Figure 3-16. SysConfig Script File.....	18
Figure 3-17. SysConfig Script Names.....	18
Figure 4-1. SysConfig Generated Content After the CCS Project is Built.....	19
Figure 5-1. Application Code Using C2000 SysConfig Initialization.....	20
Figure 6-1. Interrupt Register Checkbox.....	21
Figure 6-2. board.c: Interrupt Registration.....	22
Figure 6-3. board.h: Interrupt Registration.....	23
Figure 7-1. Device Specific Code Generation and Enhanced Portability.....	24
Figure 8-1. Enable SysConfig in the CCS Project.....	25
Figure 10-1. Standalone SysConfig Start Page.....	26

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

1 Introduction

C2000™ SysConfig is a graphical user interface (GUI) tool that allows you to configure your C2000 Real-Time Control MCUs. The following are the features supported in the tool:

- System initialization code generation: C2000 SysConfig generates initialization code for the C2000 device including peripheral initialization, interrupt initialization, PinMux initialization, and so forth.
- Device PinMux visualization: A visual representation of the device and all of its pins, a list of all possible PinMux options, your selected mode for each pin, and a summary PinMux CSV file is supported by the tool.
- Error detection: C2000 SysConfig is capable of catching configuration errors and notifying you of the incorrect setup.
- Device level dependency identification: C2000 SysConfig identifies the dependencies in the device and ensures that the dependent peripherals are all configured by you.
- Device level error detection: Device level configuration errors caused by dependent peripherals are caught and you are notified of the error.
- Portable device initialization: Device initialization settings are portable between device families and device packages.

Note

The device families supported are:

- F2807x, F2837xS, F2837xD
- F28004x
- F2838x
- F28002x
- F28003x

- Seamless support for other tools: Support for other SysConfig tools such as CLB Tool and the DCSM Tool.

C2000 SysConfig is available inside [C2000Ware](#) and requires the SysConfig Tool, which is delivered built-in with Code Composer Studio™ (CCS) IDE and is also delivered as a standalone tool for use with other IDEs.

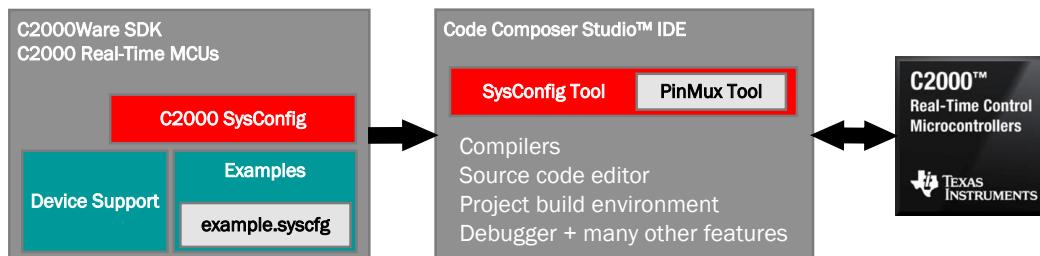


Figure 1-1. C2000 SysConfig Overview

2 Getting Started With C2000 SysConfig

The C2000 SysConfig support is built on top of the C2000 driverlib software layer. To get started, either start from an existing C2000 SysConfig based driverlib project or add C2000 SysConfig and driverlib support to an existing project.

Most driverlib examples in C2000Ware have either an *example.syscfg* file or you can add a file with the *.syscfg* extension. Double clicking and opening the *.syscfg* file launches the C2000 SysConfig tool.

2.1 Example C2000 SysConfig in CCS

To get started with C2000 SysConfig, let's import an existing example with C2000 SysConfig support.

1. Launch CCS and import the example: **clb_ex8_external_signal_AND_gate.projectspec**
 - a. Select Project → Import CCS Project
 - b. Browse to **C2000Ware_VERSION\driverlib\f2838x\examples\c28x\clb\CCS**
 - c. Select the **clb_ex8_external_signal_AND_gate.projectspec** project and import it
2. Inside your CCS project you should be able to see the *syscfg* file along with the rest of the application files.

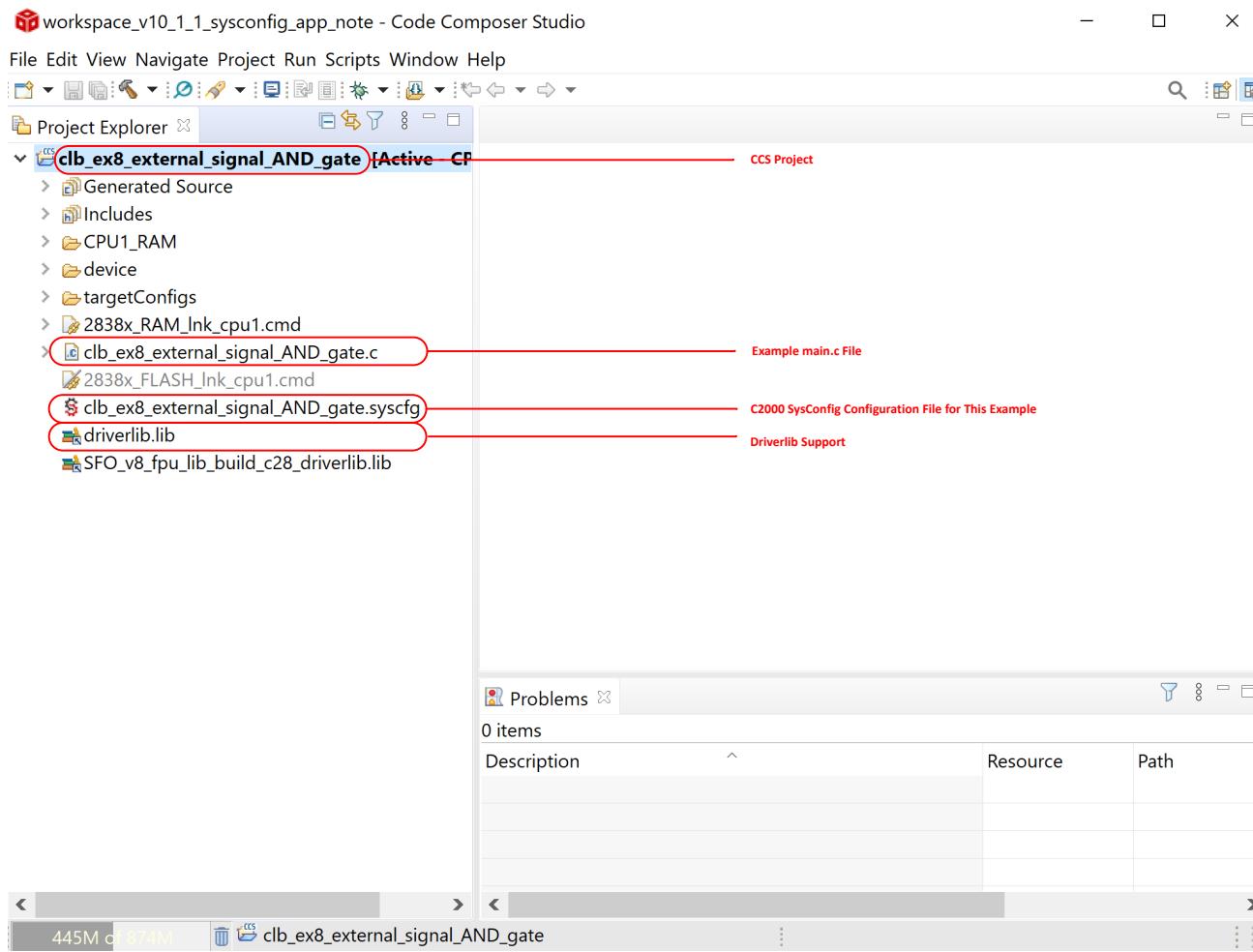


Figure 2-1. C2000 SysConfig Example Project in CCS

3. Double click on **clb_ex8_external_signal_AND_gate.syscfg** file and the C2000 SysConfig GUI will launch

Note

You can also right-click on the **syscfg** file, then select Open With → SysConfig Editor.

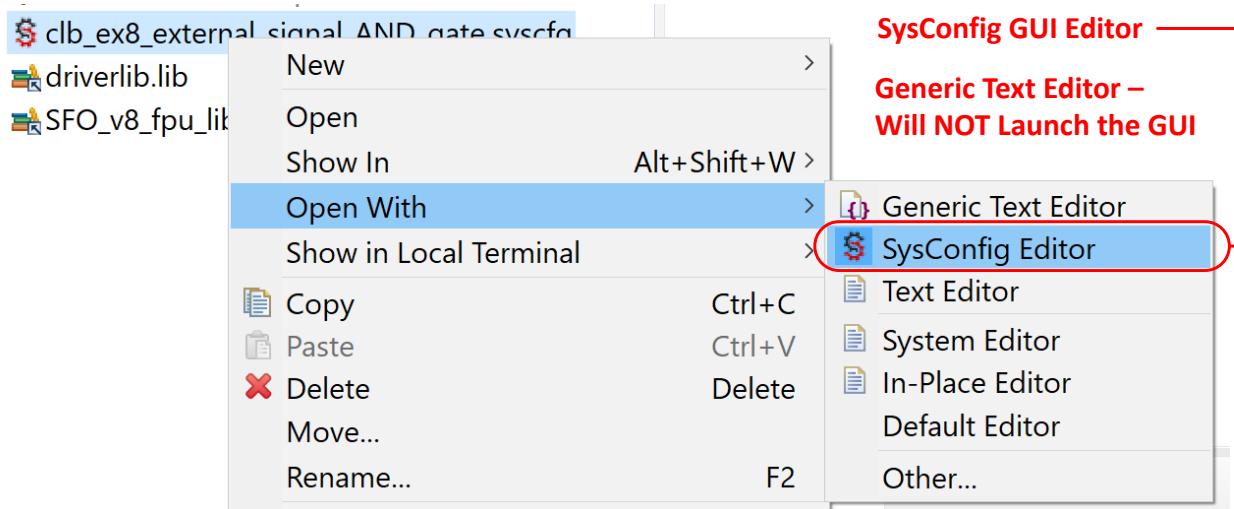


Figure 2-2. Launching SysConfig GUI

4. The C2000 SysConfig GUI should be launched inside CCS and should look similar to one shown in [Figure 2-3](#).

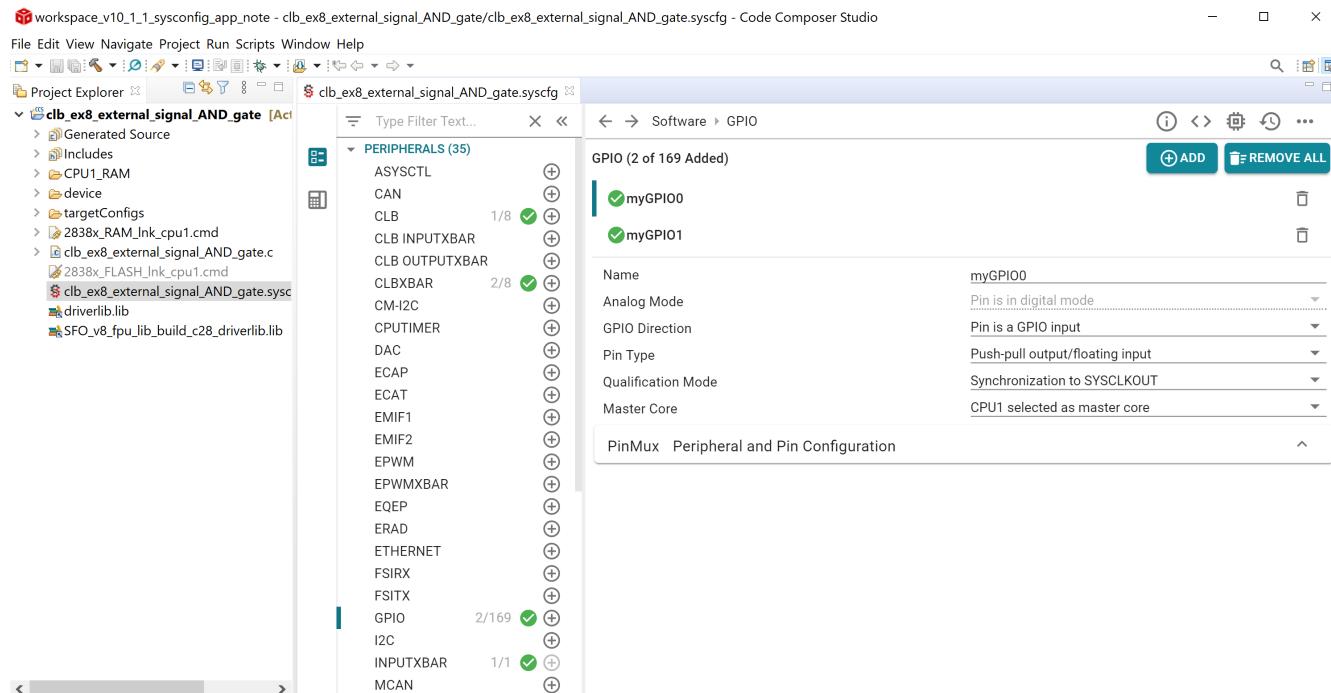


Figure 2-3. C2000 SysConfig GUI

5. Click the **Device View** button at the top right corner of the SysConfig GUI to see the device and package used for your project

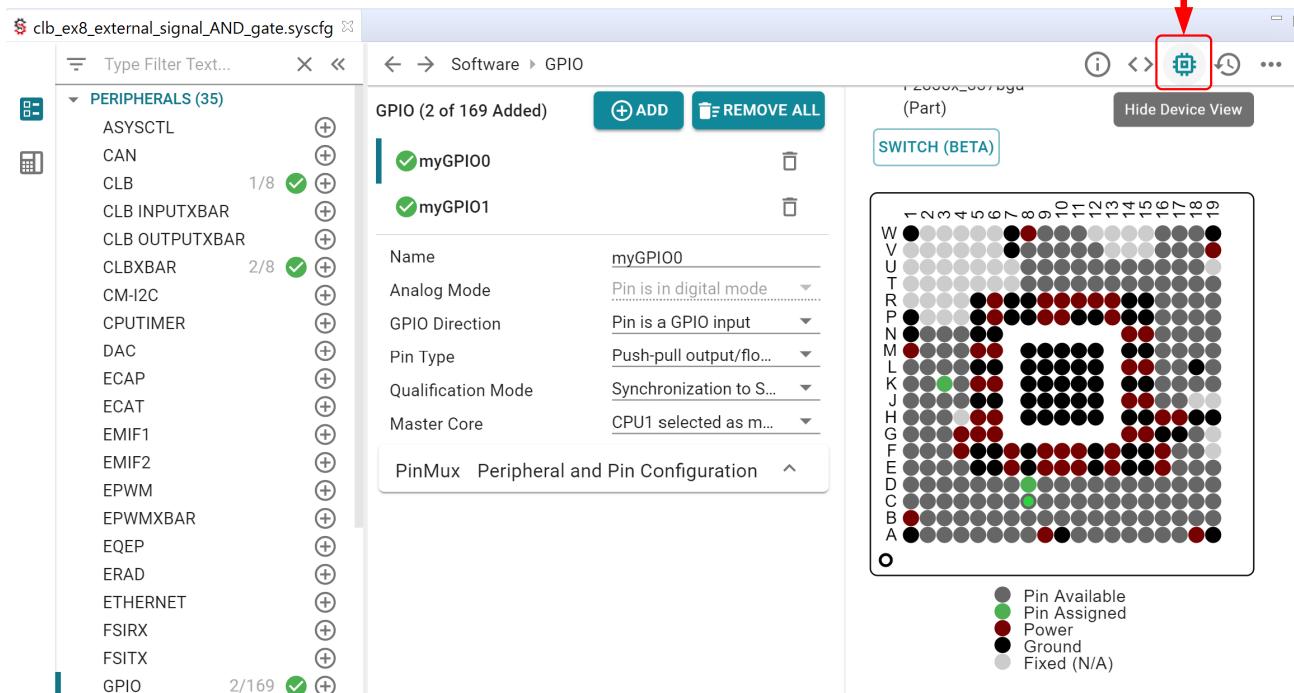


Figure 2-4. Device View

C2000 SysConfig support is added in the Project Properties. By default, this project was configured for F2838x family of devices and the selected device package is set to 337 BGA package. If the Project Properties for C2000 SysConfig support is not set up by default in your CCS project, the `syscfg` file will not launch the GUI successfully. Most driverlib projects have the Project Properties set up by default for C2000 SysConfig. If Project Properties are not set up correctly, [Section 8](#) describes how C2000 SysConfig can be added to a CCS project.

2.2 Other SysConfig Tools

Other SysConfig-based tools such as the CLB Tool and the Security (DCSM) Tool are seamlessly integrated with C2000 SysConfig. If these tools are supported by the device family selected by you, they will automatically show up as a new section under the modules panel of the SysConfig GUI.

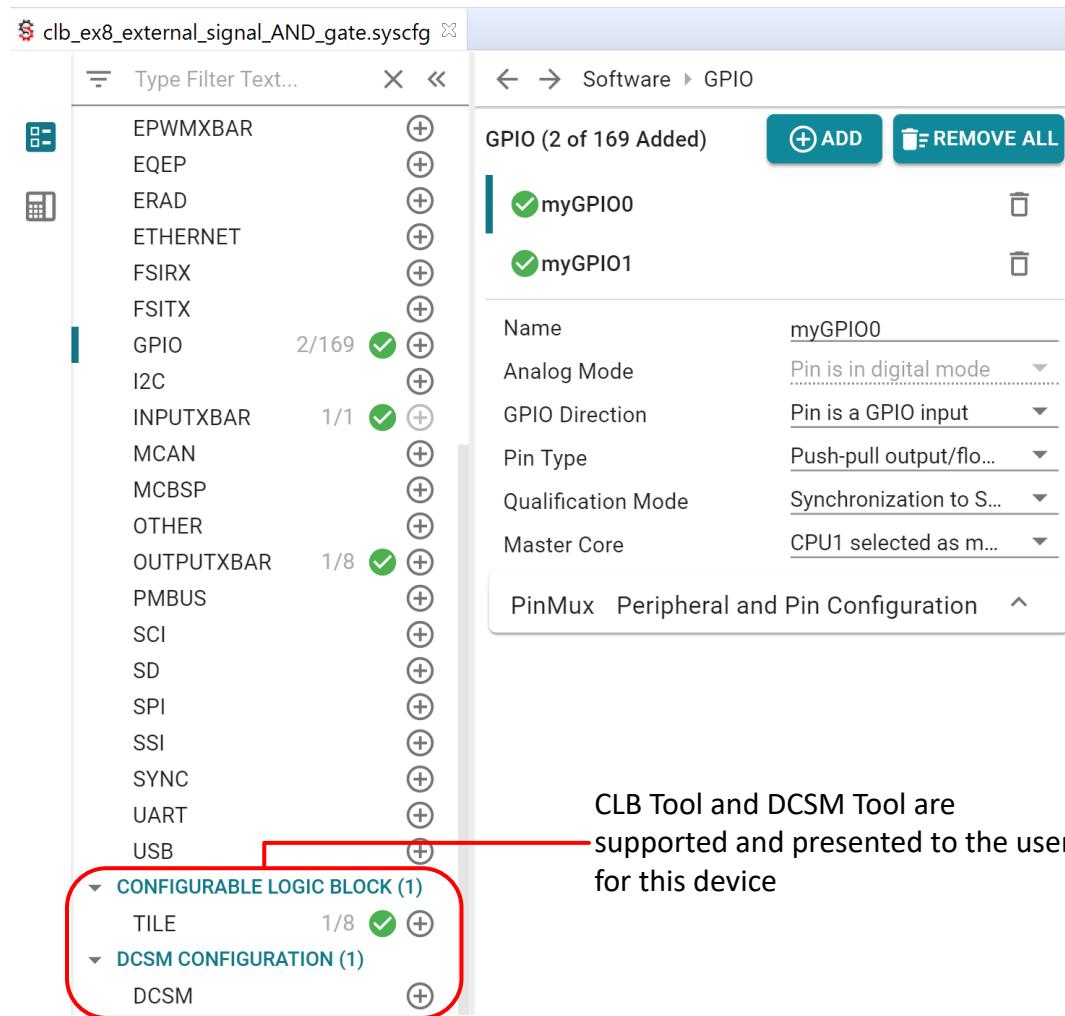


Figure 2-5. CLB and DCSM (Security) Tool

The CLB Tool and DCSM Tool can be used as standalone tools. You can modify the CCS Project Properties to use only the CLB Tool or the DCSM Tool.

For more information on the DCSM Tool, visit:

- [C2000™ DCSM Security Tool](#)

For more information on the CLB Tool, visit:

- [CLB Tool User's Guide](#)
- [Designing With the C2000™ Configurable Logic Block \(CLB\)](#)

3 C2000 SysConfig Overview

C2000 SysConfig begins with the `sdk.json` file which contains all of the information for the tool. The projects with C2000 SysConfig support built-in, already have the Project Properties set to point the CCS SysConfig GUI to the C2000 SysConfig content.

To view the SysConfig Project Properties in your CCS project:

1. Right-click on the project name and select **Properties**
2. Under the **Build** options, select **SysConfig** to view all SysConfig options

CCS SysConfig Project Options

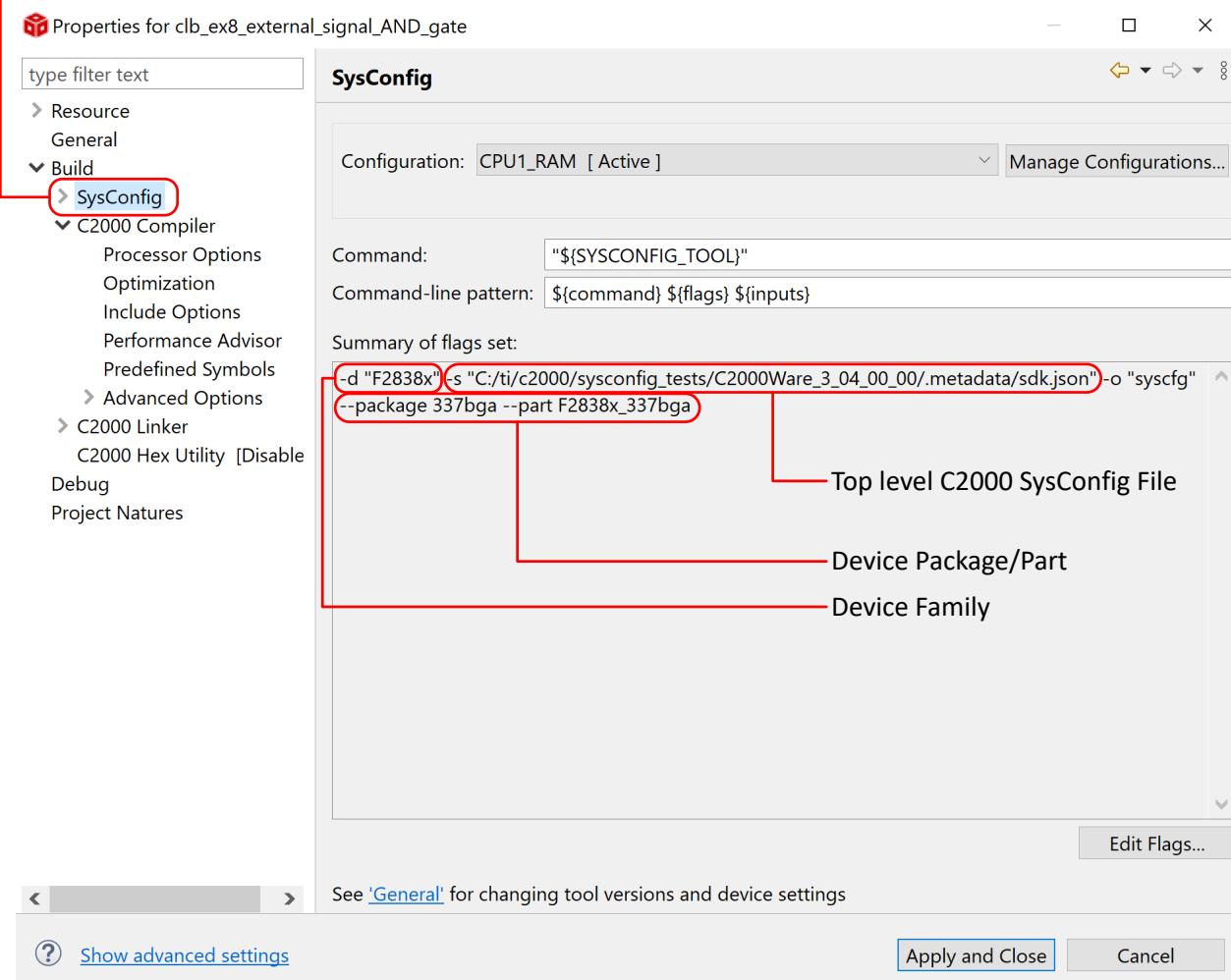


Figure 3-1. CCS SysConfig Project Properties

3. Select **Basic Options** to change/view the device family and top level SysConfig *sdk.json* file

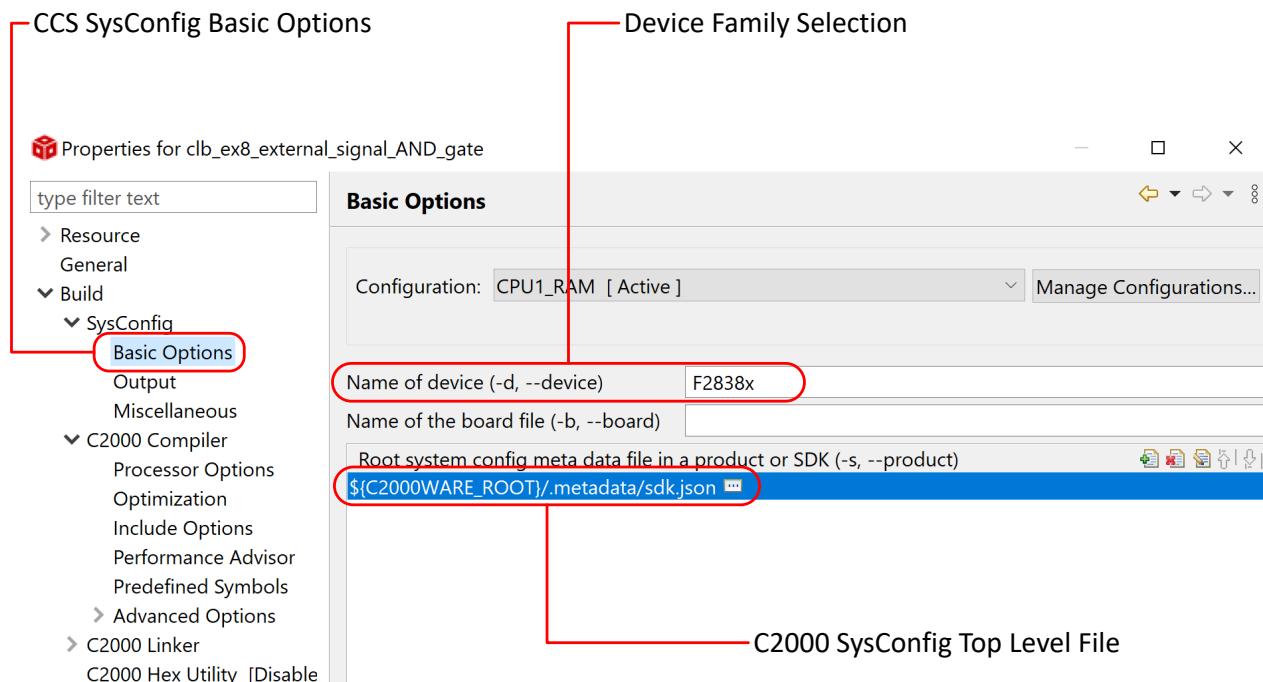


Figure 3-2. SysConfig Basic Options

4. Select **Miscellaneous** to change/view the device package/part

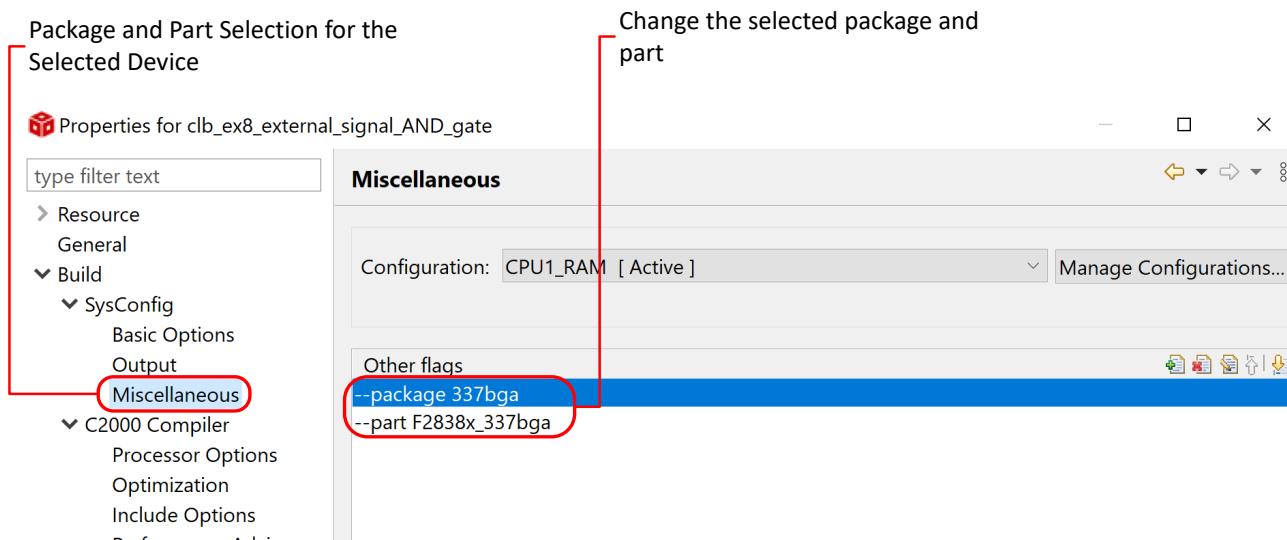


Figure 3-3. SysConfig Miscellaneous Options

3.1 Modules

The available modules/peripherals for each device/package is listed in the left panel of the C2000 SysConfig GUI. The number of each peripheral available for the device is shown in [Figure 3-4](#) as the modules are added to the application by you. This allows for a simple resource management by you.

Each module's description is shown in the middle panel (configurable options panels) and once the module is added, the description is minimized. The description can easily be expanded by clicking the question mark icon next to the name of the module (if available).

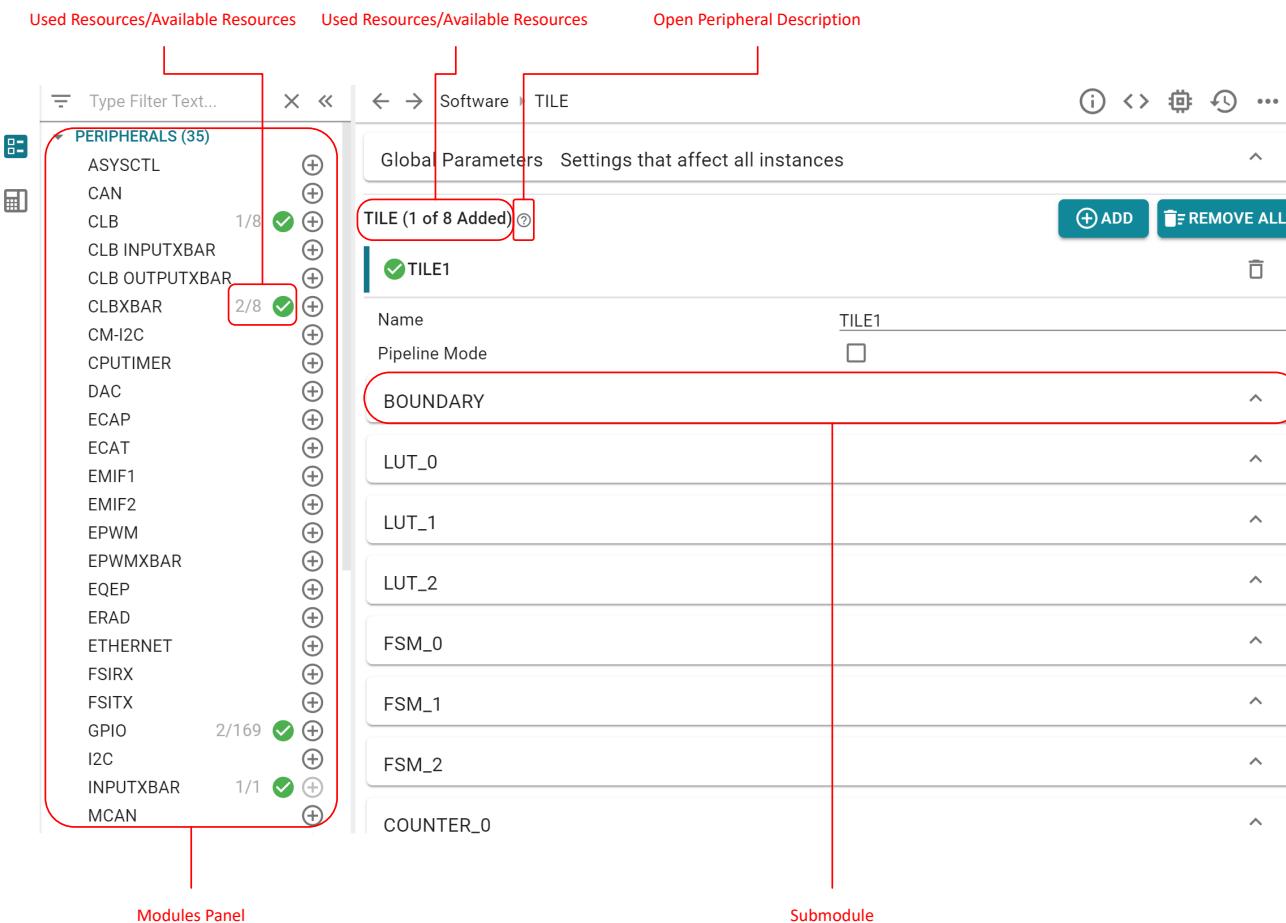


Figure 3-4. C2000 SysConfig Modules and Resource Management

3.2 PinMux

For each peripheral with PinMux options, there is a PinMux submodule available in the Configurable Options panel. Inside the PinMux submodule, there are configurable options for each pin of the peripheral along with the instance of the peripheral. The solution for the PinMux is shown as the selected option for each pin. You can **LOCK** the solution to ensure it does not change as more modules/peripherals are added.

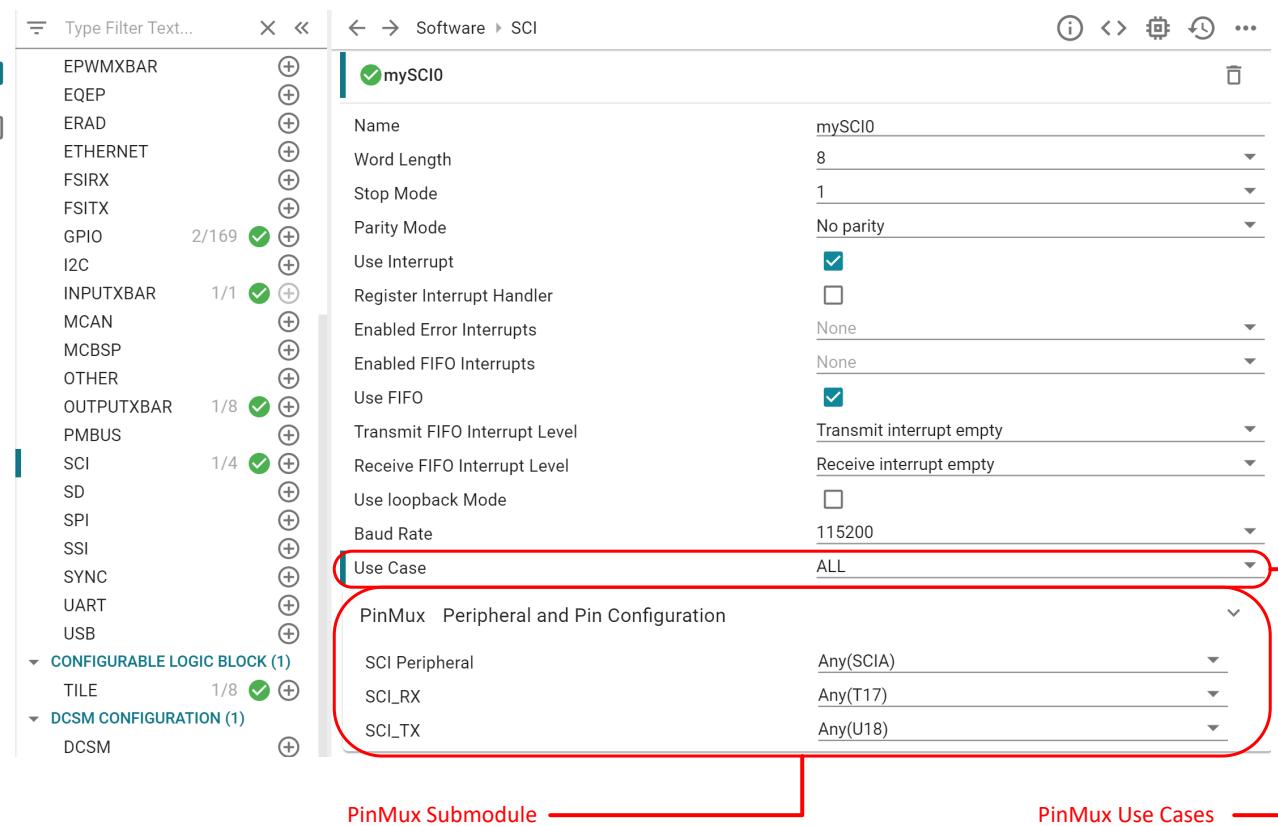


Figure 3-5. PinMux Submodule

Above the PinMux submodule, a configurable option named **Use Case** is also available to limit the available peripheral pins in the PinMux module. Selecting the **Custom** option for the **Use Case** adds a new configurable option named **Pins Used**, where you can select the peripheral pins for their specific custom use-case.

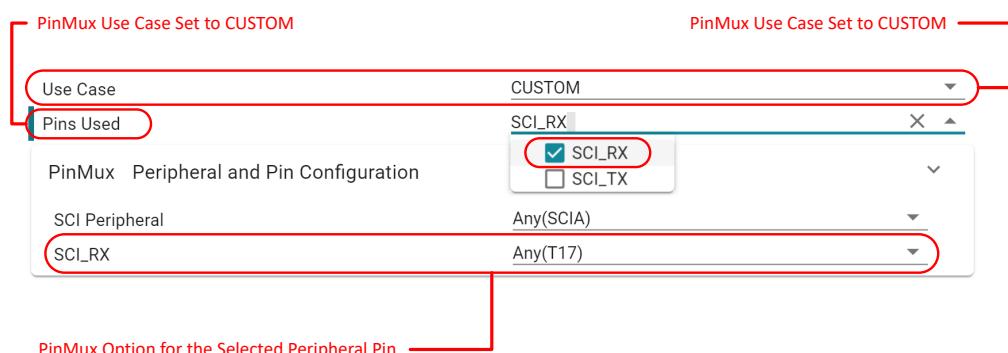


Figure 3-6. PinMux Custom Use Case

It is also possible to modify the PinMux submodule GUI to show not only the device pin names, but also the device GPIO number. To change the pin name representation:

- Click the three dots at the top right corner of the C2000 SysConfig GUI
- Select **Preferences and Actions**
- In the **Device Pin Labels** options, select the **Device Pin Name** option

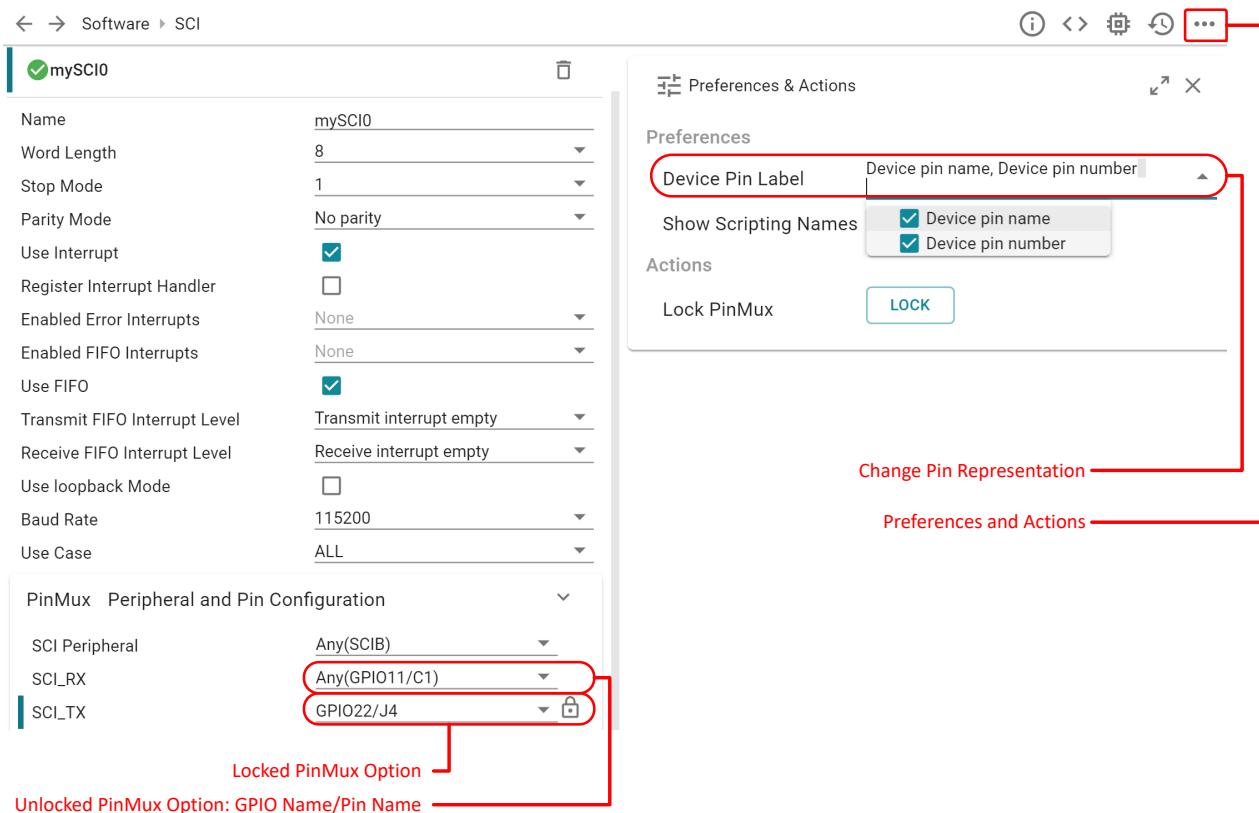


Figure 3-7. Device Pin Representation

The device PinMux summary is available inside the *pinmux.csv* auto-generated file.

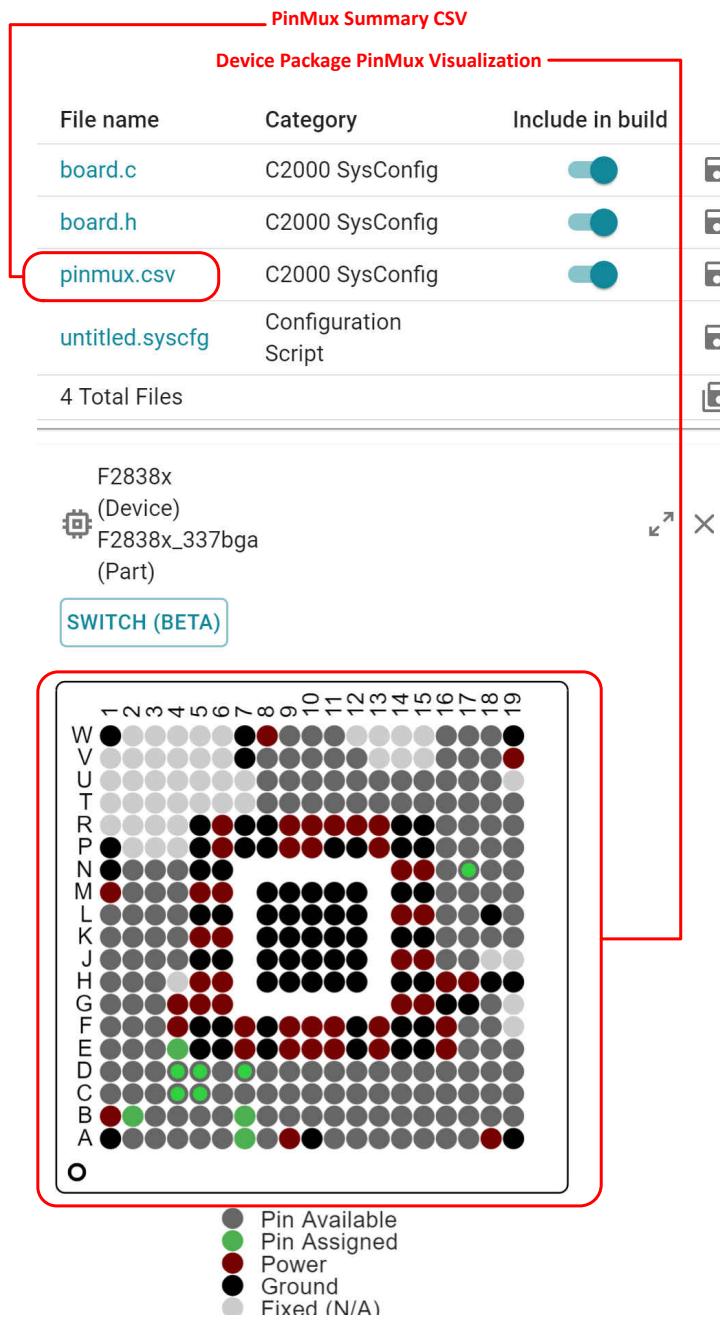


Figure 3-8. PinMux Summary CSV Auto-Generated File

The *pinmux.csv* file contains not only the selected PinMux options, but it also contains **ALL AVAILABLE** PinMux options for each pin.

3.4 Code Generation

The C2000 SysConfig auto-generates code and other debug or visualization artifacts to simplify the user's development process. The auto-generated content can be viewed inside SysConfig by clicking the **Show Generated Files** button at the top right corner.

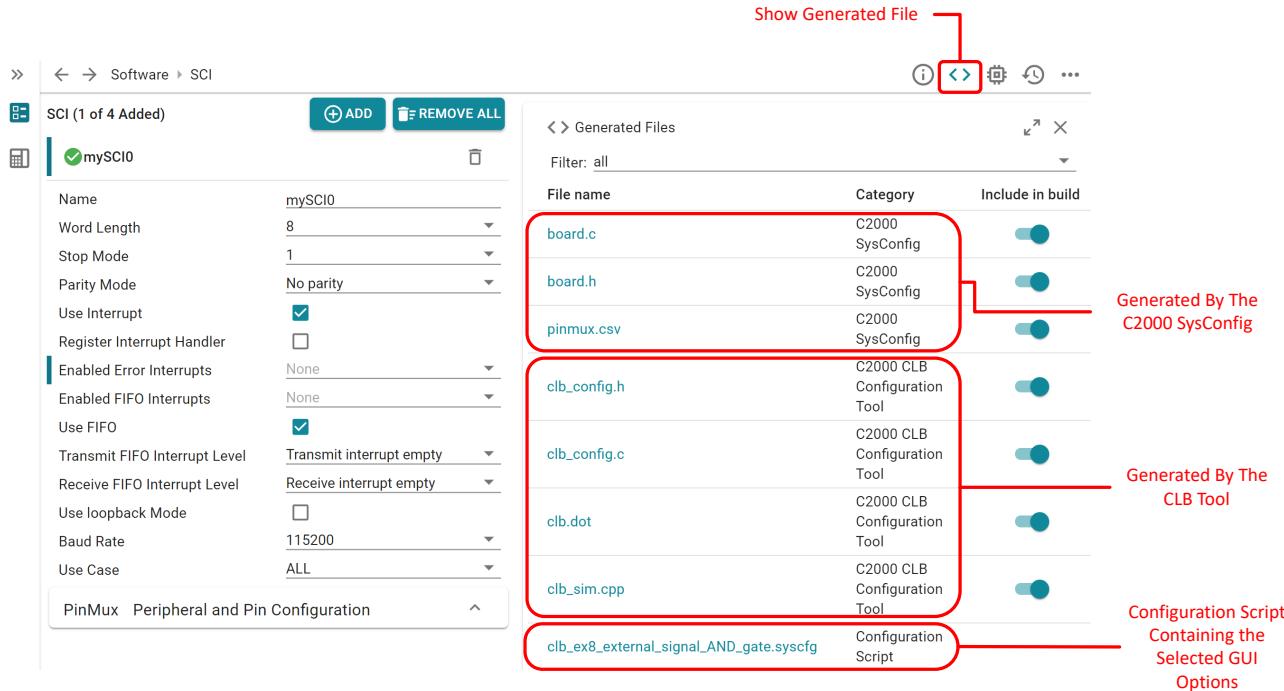


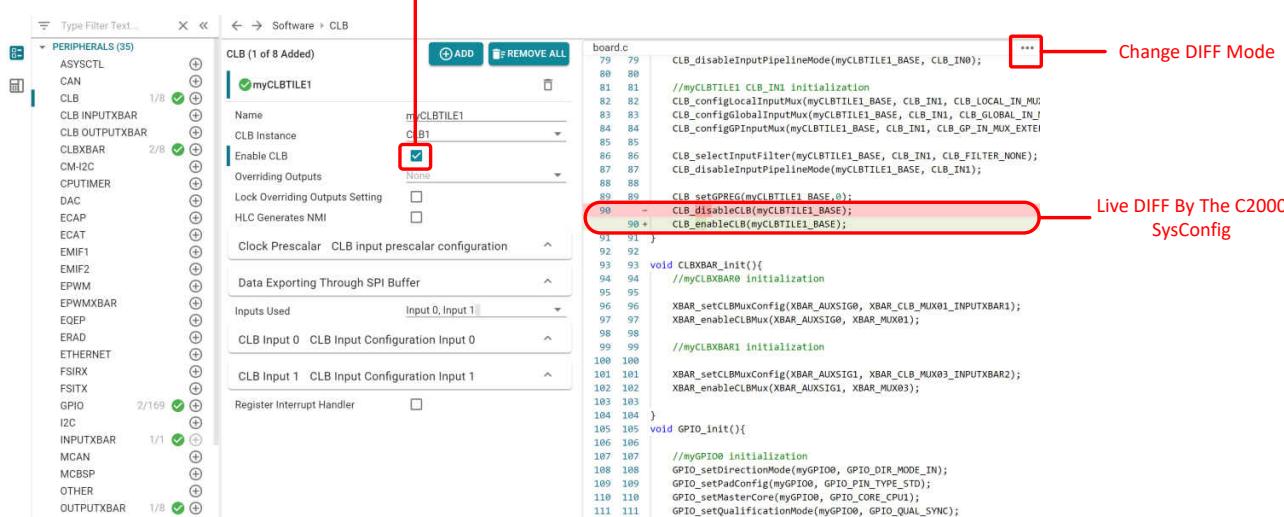
Figure 3-11. Auto-Generated Content

The content generated by the CLB Tool is removed when all instances of TILE used in the CLB Tool are removed from SysConfig. Similar to the CLB Tool, if the DCSM Tool to their design is added, new files generated by the DCSM tool will appear in the **Generated Files** panel.

To view a generated file and a live view of the updates made to the file as the configurable options are changed in the GUI, click on the file name in the **Generated Files** panel.

- Click on the *board.c* file in the **Generated Files** panel
- In the CLB module, check the **Enable CLB** box
- The *board.c* file should be updated with the changes made in the code marked with RED and GREEN
- You can pick the DIFF mode by clicking on the three dots at the top right corner of the GUI

Changed Configurable



PERIPHERALS (35)

- ASYCCTL
- CAN
- CLB
- CLB INPUTXBAR 1/8
- CLB OUTPUTXBAR 2/8
- CLBXBAR
- CM12C
- CPUTIMER
- DAC
- ECAP
- ECAT
- EMIF1
- EMIF2
- EPWM
- EPWMXBAR
- EQEP
- ERAD
- ETHERNET
- FSIRK
- FSITX
- GPIO 2/169
- I2C
- INPUTXBAR 1/1
- MCAN
- MCBSP
- OTHER
- OUTPUTXBAR 1/8

CLB (1 of 8 Added)

<input checked="" type="checkbox"/>	myCLBTILE1	+ ADD	- REMOVE ALL
Name		myCLBTILE1	
CLB Instance		CLB1	
Enable CLB		<input checked="" type="checkbox"/>	
Overriding Outputs		None	
Lock Overriding Outputs Setting			
HLC Generates NMI		<input type="checkbox"/>	
Clock Prescaler		CLB input prescaler configuration	
Data Exporting Through SPI Buffer			
Inputs Used		Input 0, Input 1	
CLB Input 0 CLB Input Configuration Input 0			
CLB Input 1 CLB Input Configuration Input 1			
Register Interrupt Handler		<input type="checkbox"/>	

```
board.c
79 79 CLB_disableInputPipelineMode(myCLBTILE1_BASE, CLB_IN0);
80 80 //myCLBTILE1_CLB_IN1_initialization
81 81 CLB_configLocalInputMux(myCLBTILE1_BASE, CLB_IN1, CLB_LOCAL_IN_MUX)
82 82 CLB_configGlobalInputMux(myCLBTILE1_BASE, CLB_IN1, CLB_GLOBAL_IN_MUX)
83 83 CLB_configInputMux(myCLBTILE1_BASE, CLB_IN1, CLB_GPIO_IN_MUX_EXTI)
84 84 CLB_selectInputFilter(myCLBTILE1_BASE, CLB_IN1, CLB_FILTER_NONE);
85 85 CLB_disableInputPipelineMode(myCLBTILE1_BASE, CLB_IN1);
86 86 CLB_setGPREn(myCLBTILE1_BASE, 0);
87 87 CLB_disableCLB(myCLBTILE1_BASE);
88 88 CLB_enableCLB(myCLBTILE1_BASE);
89 89 + CLB_disableCLB(myCLBTILE1_BASE);
90 90 - CLB_enableCLB(myCLBTILE1_BASE);
91 91 }
```

Figure 3-12. Changed Configurable Code Generation DIFF

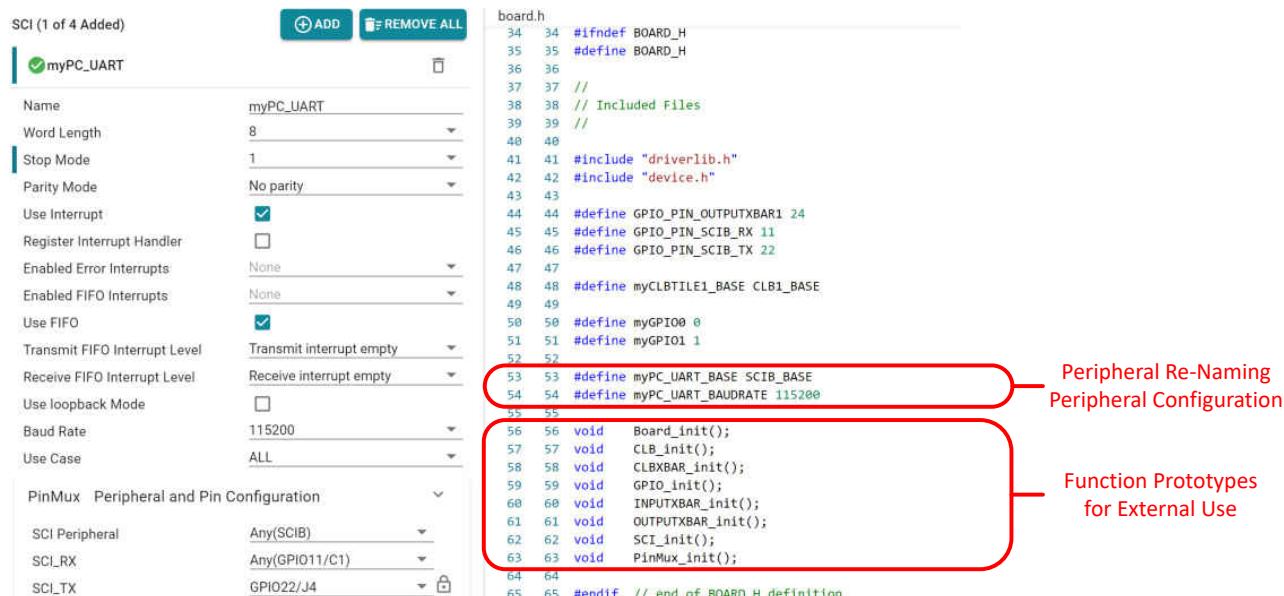
C2000 SysConfig generates two code files:

- **board.c**: This file contains the initialization code for each module including PinMux. An all inclusive function named **Board_init** is available which initializes all module. You can choose to call the individual **Module_init** functions in their application, or you can use the **Board_init** function to use all modules. You can also choose to add **PinMux_init** to your application code to use only the PinMux initialization feature of the tool.

Note

Board_init does not call **DC_DC_init**. **DC_DC_init** must be called before **Board_init**.

- **board.h**: This file contains the prototypes for all generated functions in the **board.c** file, along with the re-namings of the modules for their specific application. Also, the GPIO number assigned for each PinMux option along with more information on each peripherals configuration.



SCI (1 of 4 Added)

<input checked="" type="checkbox"/>	myPC_UART	+ ADD	- REMOVE ALL
Name		myPC_UART	
Word Length		8	
Stop Mode		1	
Parity Mode		No parity	
Use Interrupt		<input checked="" type="checkbox"/>	
Register Interrupt Handler		<input type="checkbox"/>	
Enabled Error Interrupts		None	
Enabled FIFO Interrupts		None	
Use FIFO		<input checked="" type="checkbox"/>	
Transmit FIFO Interrupt Level		Transmit interrupt empty	
Receive FIFO Interrupt Level		Receive interrupt empty	
Use loopback Mode		<input type="checkbox"/>	
Baud Rate		115200	
Use Case		ALL	
PinMux Peripheral and Pin Configuration			
SCI Peripheral		Any(SCIB)	
SCI_RX		Any(GPIO11/C1)	
SCI_TX		GPIO22/J4	

```
board.h
34 34 #ifndef BOARD_H
35 35 #define BOARD_H
36 36
37 37 //
38 38 // Included Files
39 39 //
40 40
41 41 #include "driverlib.h"
42 42 #include "device.h"
43 43
44 44 #define GPIO_PIN_OUTPUTXBAR 24
45 45 #define GPIO_PIN_SCIB_RX 11
46 46 #define GPIO_PIN_SCIB_TX 22
47 47
48 48 #define myCLBTILE1_BASE CLB1_BASE
49 49
50 50 #define myGPIO0 0
51 51 #define myGPIO1 1
52 52
53 53 #define myPC_UART_BASE SCIB_BASE
54 54 #define myPC_UART_BAUDRATE 115200
55 55
56 56 void Board_init();
57 57 void CLB_init();
58 58 void CLBXBAR_init();
59 59 void GPIO_init();
60 60 void INPUTXBAR_init();
61 61 void OUTPUTXBAR_init();
62 62 void SCI_init();
63 63 void PinMux_init();
64 64
65 65 #endif // end of BOARD_H definition
```

Figure 3-13. board.h File

3.5 Error Detection

One of the most important and useful features of the C2000 SysConfig is its ability to detect errors or missing requirements in your configuration.

Embedded devices often have many supported modes, but the device must be configured exactly as instructed by the technical documentation for each mode to operate correctly.

Also, the device silicon Errata documentation notes the unsupported modes that is sometimes missed. It is common that the development process for configuring a device is slowed down due to errors in the user's code. These errors could be due to mistakes in programming when transferring knowledge from the technical documentation into the application software. C2000 SysConfig is capable of catching configuration errors and notifying you of the incorrect setup.

Also similar to error generation, warnings are also generated as needed when a configuration is not necessarily wrong, but requires further attention.

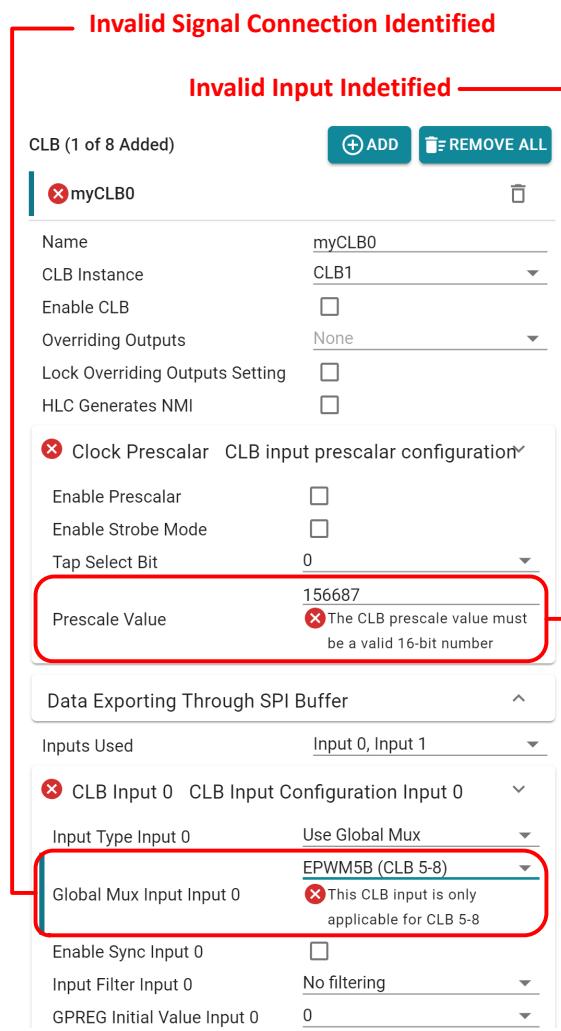


Figure 3-14. Error Detection

Error detection in your configuration is NOT limited to one peripheral at a time. Incorrect setups can be detected across dependent modules. This ensures that all dependent peripherals are configured correctly.

**Dependent Peripheral Identified
Unsupported Mode due to Dependent
Peripheral's Configuration**

Global Parameters Settings that affect all instances

Other Dependencies

ASYSCTL Analog SysCtl

- Enable Temperature Sensor
- Lock Temperature Sensor Control ...
- Analog Reference Internal
- Analog Reference Voltage 2.5V

DAC (1 of 2 Added) + ADD REMOVE ALL

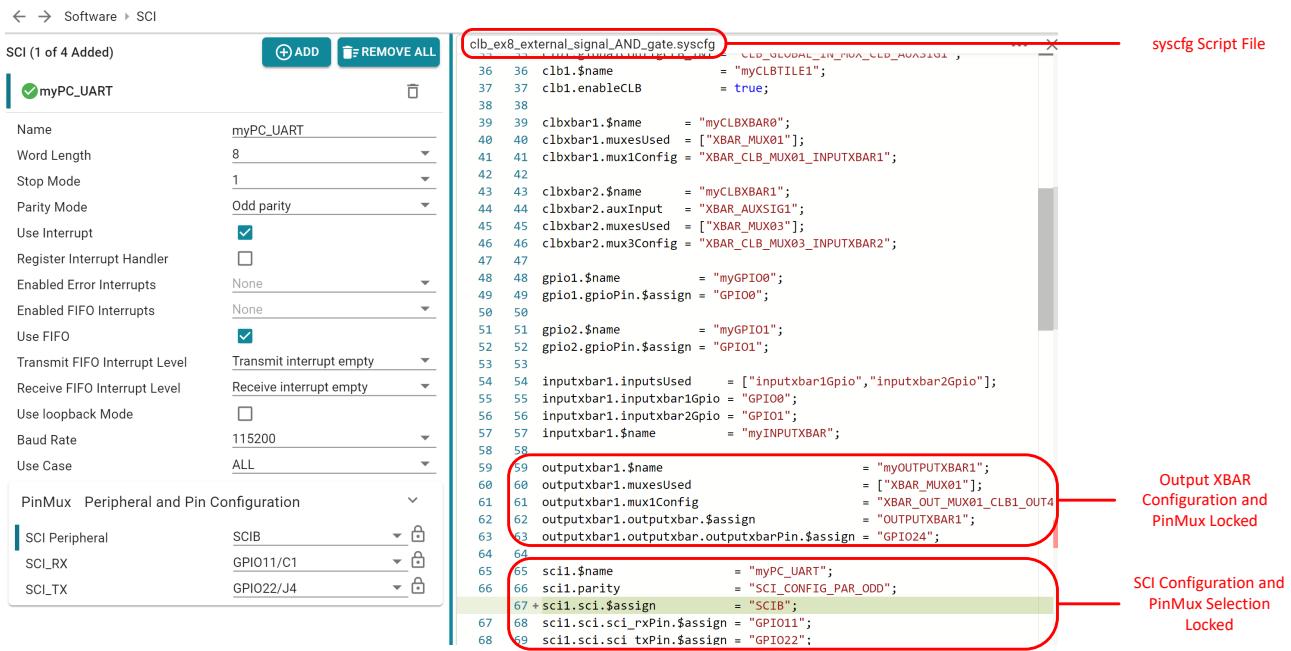
myDAC0

Name	myDAC0
DAC Instance	DACA
Reference Voltage	ADC VREFHI reference voltage
Gain Mode	Gain set to 2 ✖ Selected gain mode not supported.
Load Mode	Load on next SYSCLK
EPWMSYNCPER Signal	ePWM sync signal 1
Shadow Value	0
Enable Output	<input type="checkbox"/>
Lock DAC Registers	None

Figure 3-15. Dependent Module Error Detection

3.6 SysConfig Script File

Your settings for C2000 SysConfig and all other tools configured in the SysConfig GUI are saved in a `syscfg` file. The changes made to this file as you modify the configurable options can be viewed similar to any other auto-generated file in SysConfig. The SysConfig script saves the settings for all options selected by you.



The screenshot shows the C2000 SysConfig interface for the SCI peripheral. On the left, the configuration for `myPC_UART` is displayed, including fields for Name, Word Length, Stop Mode, Parity Mode, Use Interrupt, Register Interrupt Handler, Enabled Error Interrupts, Enabled FIFO Interrupts, Use FIFO, Transmit FIFO Interrupt Level, Receive FIFO Interrupt Level, Use loopback Mode, Baud Rate, and Use Case. Below this, the PinMux configuration is shown for SCI Peripheral (SCIB), SCL_RX (GPIO11/C1), and SCL_TX (GPIO22/J4). On the right, the generated `clb_ex8_external_signal_AND_gate.syscfg` script is shown. The script contains code for various components like CLB1, CLBXBAR1, CLBXBAR2, and GPIO pins, with specific names like `myPC_UART`, `myCLBXBAR0`, etc. Red boxes highlight specific sections of the script:

- A red box around the top of the script is labeled "syscfg Script File".
- A red box around the output XBAR configuration section (lines 59-63) is labeled "Output XBAR Configuration and PinMux Locked".
- A red box around the SCI configuration section (lines 65-69) is labeled "SCI Configuration and PinMux Selection Locked".

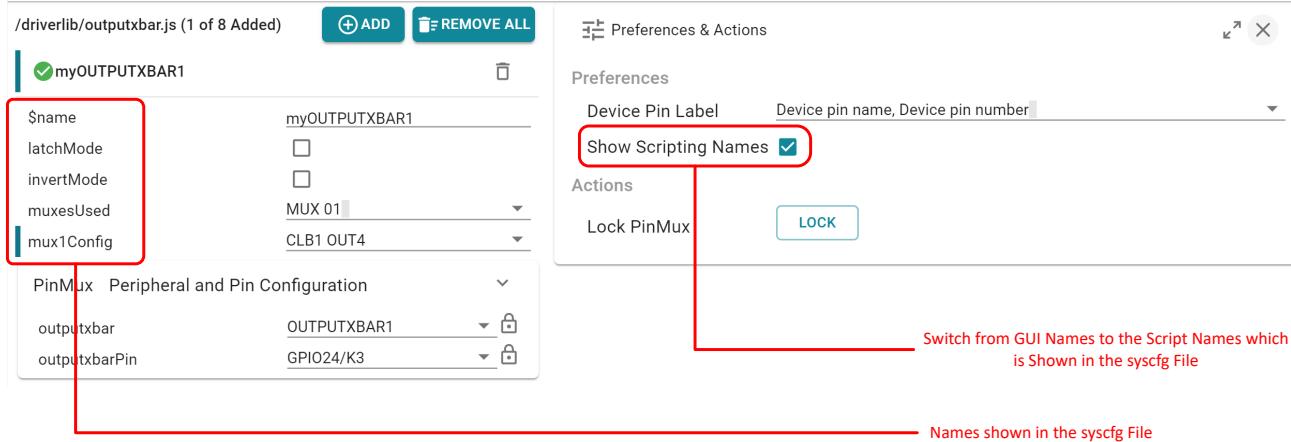
```

36 clb1.$name = "myCLBTITLE1";
37 clb1.enableCLB = true;
38
39 clbxbar1.$name = "myCLBXBAR0";
40 clbxbar1.muxesUsed = ["XBAR_MUX01"];
41 clbxbar1.mux1Config = "XBAR_CLB_MUX01_INPUTXBAR1";
42
43 clbxbar2.$name = "myCLBXBAR1";
44 clbxbar2.auxInput = "XBAR_AUXSIG1";
45 clbxbar2.muxesUsed = ["XBAR_MUX03"];
46 clbxbar2.mux3Config = "XBAR_CLB_MUX03_INPUTXBAR2";
47
48 gpio1.$name = "myGPIO00";
49 gpio1.gpioPin.$assign = "GPIO00";
50
51 gpio2.$name = "myGPIO1";
52 gpio2.gpioPin.$assign = "GPIO1";
53
54 inputxbar1.inputsUsed = ["inputxbar1Gpio", "inputxbar2Gpio"];
55 inputxbar1.inputxbar1Gpio = "GPIO0";
56 inputxbar1.inputxbar2Gpio = "GPIO1";
57 inputxbar1.$name = "myINPUTXBAR";
58
59 outputxbar1.$name = "myOUTPUTXBAR1";
60 outputxbar1.muxesUsed = ["XBAR_MUX01"];
61 outputxbar1.mux1Config = "XBAR_OUT_MUX01_CLB1_OUT4";
62 outputxbar1.outputxbar.$assign = "OUTPUTXBAR1";
63 outputxbar1.outputxbar.outputxbarPin.$assign = "GPIO24";
64
65 scii.$name = "myPC_UART";
66 scii.parity = "SCI_CONFIG_PAR_ODD";
67 +scii.sci.$assign = "SCIB";
68 scii.sci.sci_rxPin.$assign = "GPIO11";
69 scii.sci.sci_txPin.$assign = "GPIO22";

```

Figure 3-16. SysConfig Script File

The names shown in the script files for the configurable can be viewed in the GUI's configurable options by changing the setting inside the tool's **Preferences and Actions** panel.



The screenshot shows the C2000 SysConfig interface for the outputxbar peripheral. On the left, the configuration for `myOUTPUTXBAR1` is displayed, including fields for \$name, latchMode, invertMode, muxesUsed, and mux1Config. Below this, the PinMux configuration is shown for outputxbar (OUTPUTXBAR1) and outputxbarPin (GPIO24/K3). On the right, the "Preferences & Actions" panel is open. It shows the "Preferences" tab with a "Device Pin Label" field and a checked "Show Scripting Names" checkbox. The "Actions" tab has a "Lock PinMux" button. A red box highlights the "Show Scripting Names" checkbox, and another red box highlights the "LOCK" button. A red arrow points from the "Names shown in the syscfg File" text at the bottom right to the "Show Scripting Names" checkbox. Another red arrow points from the "Switch from GUI Names to the Script Names which is Shown in the syscfg File" text at the bottom right to the "LOCK" button.

Names shown in the syscfg File

Switch from GUI Names to the Script Names which is Shown in the syscfg File

Figure 3-17. SysConfig Script Names

4 SysConfig Generated Files After the Project is Built

After you have completed the C2000 SysConfig configuration and built the CCS project, the content that was shown in the **Generated Files Panel** inside SysConfig is now available in the build configuration directory, under the folder named `syscfg`.

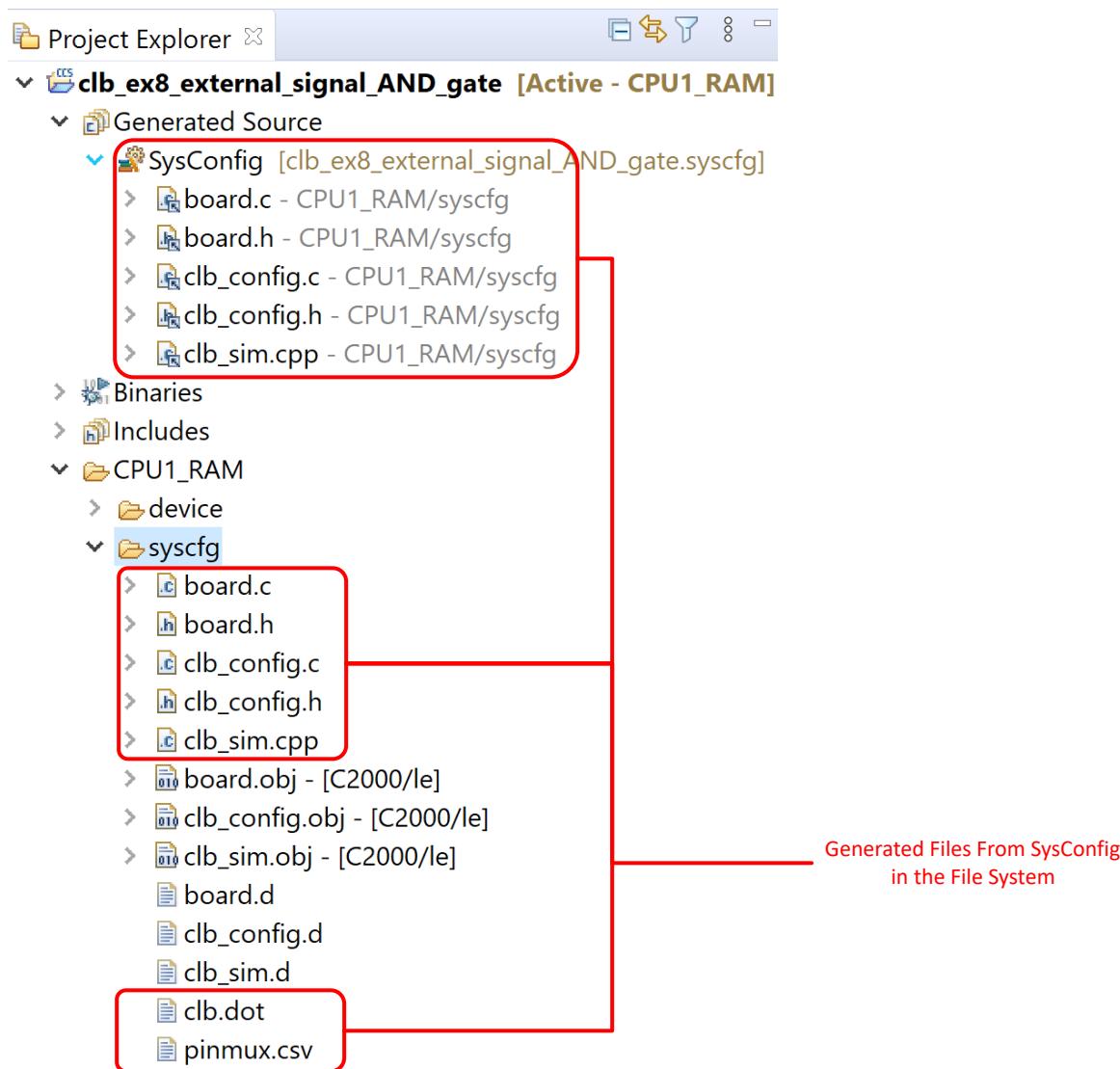


Figure 4-1. SysConfig Generated Content After the CCS Project is Built

In the Generated Files by SysConfig, the source files are compiled and the header files can be used in application software because the `syscfg` folder is automatically added to the include path during compile time. In most applications, the `board.h` is added by a `#include` statement and the function `Board_init` is called in `main`. The files generated in the file system are read-only and any changes made to them will be overwritten during the next project build.

5 Application Code Based on C2000 SysConfig Initialization

The first step in using the C2000 SysConfig initialization in an application is calling the **Board_init** or any of the other **Module_init** functions in the application code. The most common use-case is calling the **Board_init** after the **Device_init** function call to initialize all modules configured inside the C2000 SysConfig tool.

After the device initialization calls are completed, any further driverlib function call for using each module should be done using the new application specific name of the module.

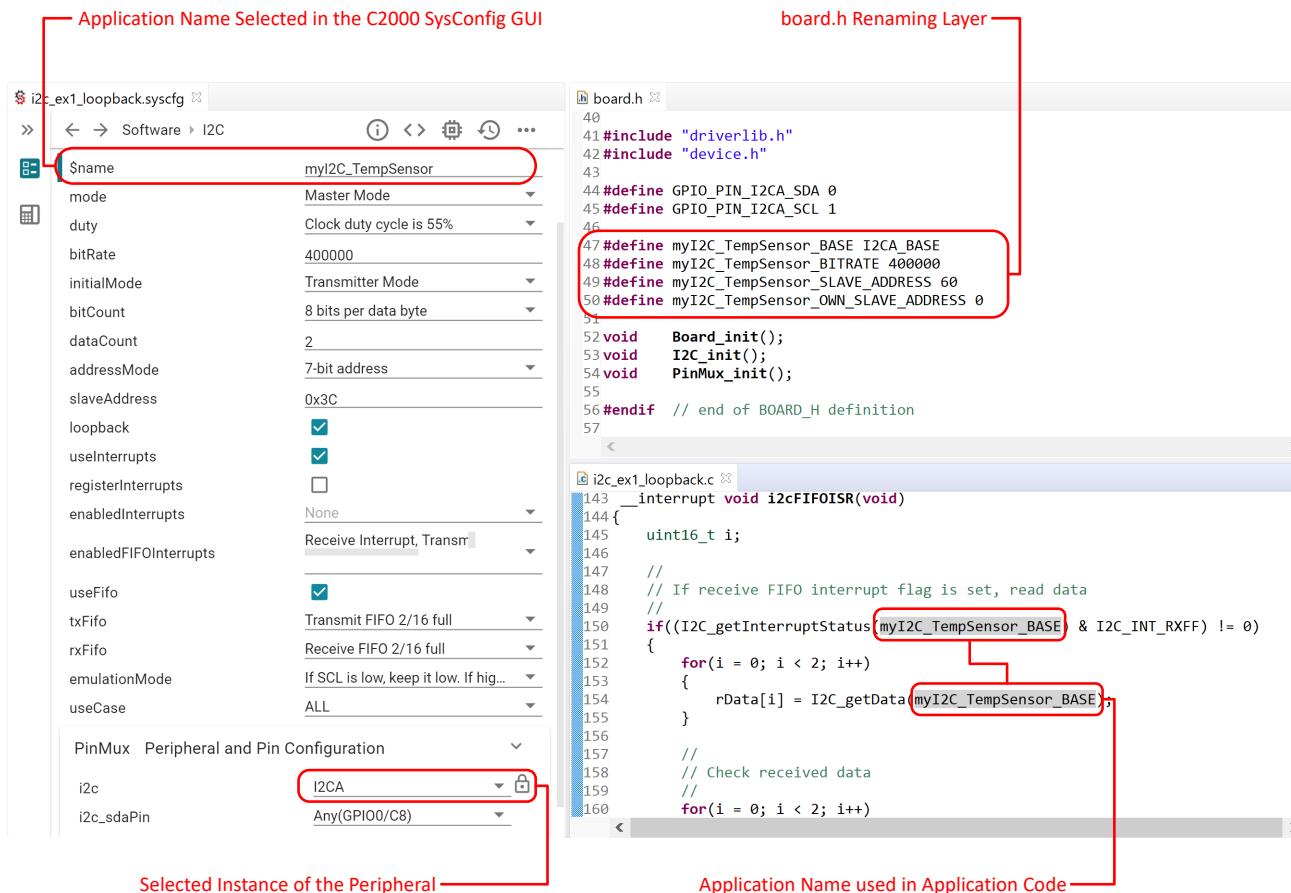


Figure 5-1. Application Code Using C2000 SysConfig Initialization

In the example above, the selected instance of the I2C peripheral is I2CA. You can switch to another instance of I2C by changing the GUI to select I2CB, and everything else is automatically taken care of if the application code uses the name assigned to the peripheral in the **\$name** configurable option.

6 Interrupt Support

C2000 SysConfig has support for registering interrupts and configuring both CPU interrupts and the PIE module. Each module, that has an associated interrupt, has a configurable option that determines whether or not you want to register an interrupt handler. When this option is checked, a submodule for configuring and registering the interrupt appears in the GUI.

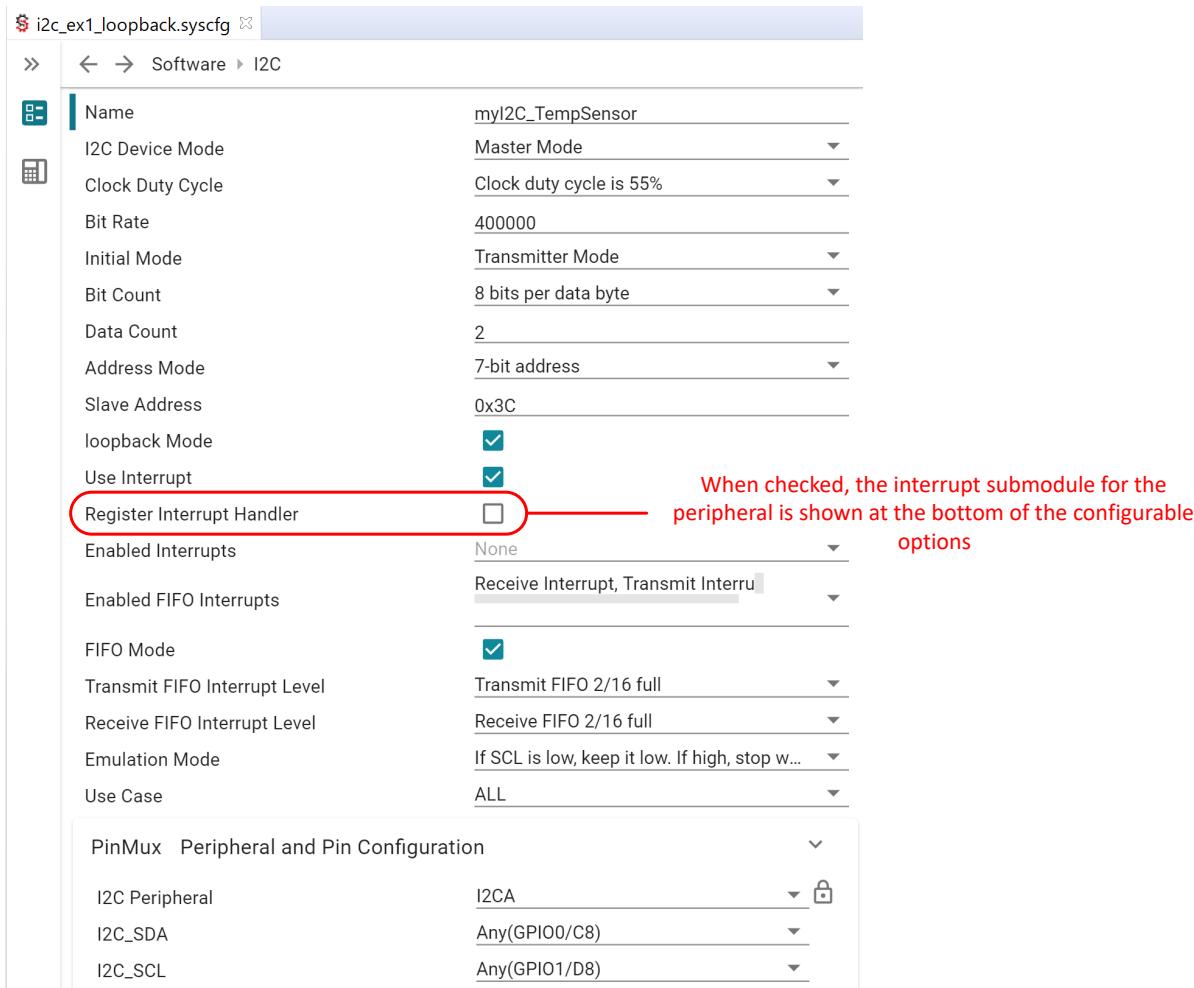
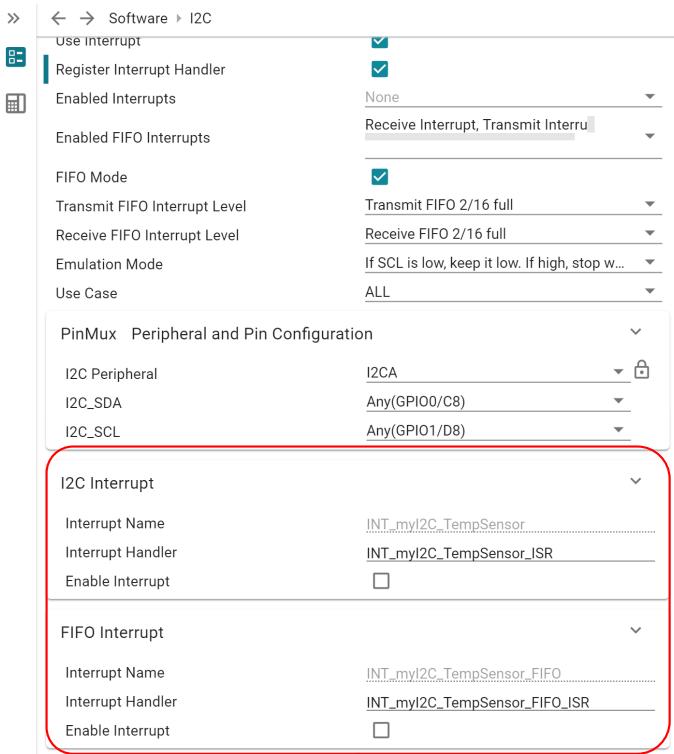


Figure 6-1. Interrupt Register Checkbox

Once one or more modules have selected to use C2000 SysConfig interrupt registration, the interrupt code generation is activated and the *board.h* and *board.c* files are updated with the interrupt code.



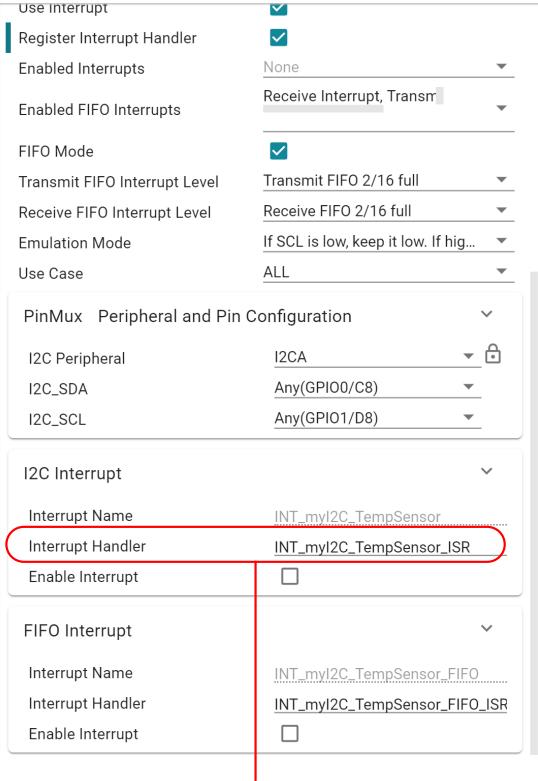
When checked, the interrupt submodule for the peripheral is shown at the bottom of the configurable options

```
board.c
50 // ...
51 // ...
52 // ...
53 // ...
54 // ...
55 // ...
56 // ...
57 void I2C_init(){
58     //myI2C_TempSensor initialization
59     I2C_disableModule(myI2C_TempSensor_BASE);
60     I2C_initMaster(myI2C_TempSensor_BASE, DEVICE_SYSCLK_FREQ, 48);
61     I2C_setConfig(myI2C_TempSensor_BASE, I2C_MASTER_SEND_MODE);
62     I2C_setSlaveAddress(myI2C_TempSensor_BASE, 60);
63     I2C_enableLoopback(myI2C_TempSensor_BASE);
64     I2C_setOwnSlaveAddress(myI2C_TempSensor_BASE, 0);
65     I2C_setBitCount(myI2C_TempSensor_BASE, I2C_BITCOUNT_8);
66     I2C_setDataCount(myI2C_TempSensor_BASE, 2);
67     I2C_setAddressMode(myI2C_TempSensor_BASE, I2C_ADDR_MODE_7BIT);
68     I2C_enableIFO(myI2C_TempSensor_BASE);
69     I2C_clearInterruptStatus(myI2C_TempSensor_BASE, I2C_INT_RXFF);
70     I2C_setFIFOInterruptLevel(myI2C_TempSensor_BASE, I2C_FIFO_RX);
71     I2C_enableInterrupt(myI2C_TempSensor_BASE, I2C_INT_RXFF | I2C_INT_TX);
72     I2C_setEmulationMode(myI2C_TempSensor_BASE, I2C_EMULATION_ST);
73     I2C_enableModule(myI2C_TempSensor_BASE);
74 }
75
76
77
78
79 + void INTERRUPT_init(){
80 +
81     // Interrupt Settings for INT_myI2C_TempSensor
82     Interrupt_register(INT_myI2C_TempSensor, &INT_myI2C_TempSensor_ISR);
83     Interrupt_disable(INT_myI2C_TempSensor);
84
85     // Interrupt Settings for INT_myI2C_TempSensor_FIFO
86     Interrupt_register(INT_myI2C_TempSensor_FIFO, &INT_myI2C_TempSensor_FIFO_ISR);
87     Interrupt_disable(INT_myI2C_TempSensor_FIFO);
88 +
89
90 + }
```

board.c: Interrupt Registration Code

Figure 6-2. board.c: Interrupt Registration

Traditionally, you would have to determine the interrupt group for the peripheral interrupts in the PIE module. C2000 SysConfig automatically determines the interrupt group and creates a mapping for you in the *board.h* file.



Interrupt Handler Function Name

```

board.h
34 34 #ifndef BOARD_H
35 35 #define BOARD_H
36 36
37 37 //
38 38 // Included Files
39 39 //
40 40
41 41 #include "driverlib.h"
42 42 #include "device.h"
43 43
44 44 #define GPIO_PIN_I2CA_SDA 0
45 45 #define GPIO_PIN_I2CA_SCL 1
46 46
47 47 #define myI2C_TempSensor_BASE I2CA_BASE
48 48 #define myI2C_TempSensor_BITRATE 400000
49 49 #define myI2C_TempSensor_SLAVE_ADDRESS 60
50 50 #define myI2C_TempSensor_OWN_SLAVE_ADDRESS 0
51 51
52 +
53 +// Interrupt Settings for INT_myI2C_TempSensor
54 +#define INT_myI2C_TempSensor INT_I2CA
55 +#define INT_myI2C_TempSensor_INTERRUPT_ACK_GROUP INTERRUPT_ACK_GROUP8
56 +extern __interrupt void INT_myI2C_TempSensor_ISR(void);
57 +
58 +// Interrupt Settings for INT_myI2C_TempSensor_FIFO
59 +#define INT_myI2C_TempSensor_FIFO INT_I2CA_FIFO
60 +#define INT_myI2C_TempSensor_FIFO_INTERRUPT_ACK_GROUP INTERRUPT_ACK_GROUP8
61 +extern __interrupt void INT_myI2C_TempSensor_FIFO_ISR(void);
62 +
63 void Board_init();
64 void I2C_init();
65 +void INTERRUPT_init();
66 void PinMux_init();
67
68 #endif // end of BOARD_H definition

```

Interrupt Rename to the Application Name
Interrupt ACK GROUP automatically Identified
Interrupt Handler Prototype:
User must define the handler function in application code

Figure 6-3. board.h: Interrupt Registration

7 Device-Specific Code Generation

C2000 SysConfig generates device-specific code given the same configuration in the `syscfg` file. This makes configuration inside `syscfg` more portable across device families than application initialization C code. For example, the same `syscfg` file using the INPUT X-BAR module generates different code for different device family as shown in [Figure 7-1](#). The figure shows how the same `syscfg` configuration for F2838x and F2837xD devices generates different code that is compatible for that device family.

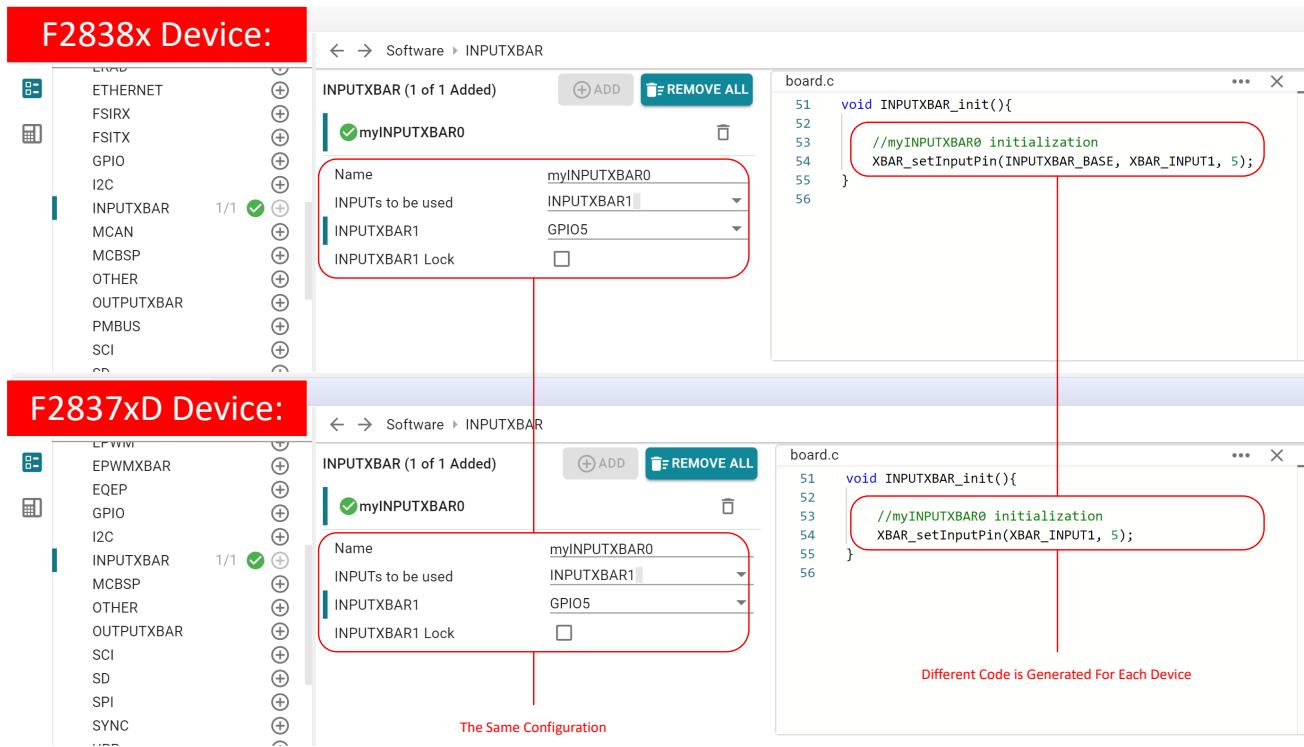


Figure 7-1. Device Specific Code Generation and Enhanced Portability

8 Adding C2000 SysConfig Support to Existing Projects

Most driverlib based examples in C2000Ware are delivered with built-in support for C2000 SysConfig. Even if the example does not have a `syscfg` configuration file, the project properties for C2000 SysConfig are most likely already configured.

Follow these steps to check if the C2000 project properties are set up for C2000 SysConfig development:

1. Right-click on the project and select **Properties**.
2. In the left panel of the project properties, under the **Build** category, check to see if the **SysConfig** options are available.
3. If the **SysConfig** option is available under **Build**, SysConfig is enabled for your project.

If the SysConfig module is not enabled in your project, follow these steps to enable it:

1. Add an empty file of the `syscfg` file named `empty.syscfg` to the project by copying the file into the project or creating a new file in the project.
2. CCS will ask whether or not to enable SysConfig. Accept and select **Yes**.

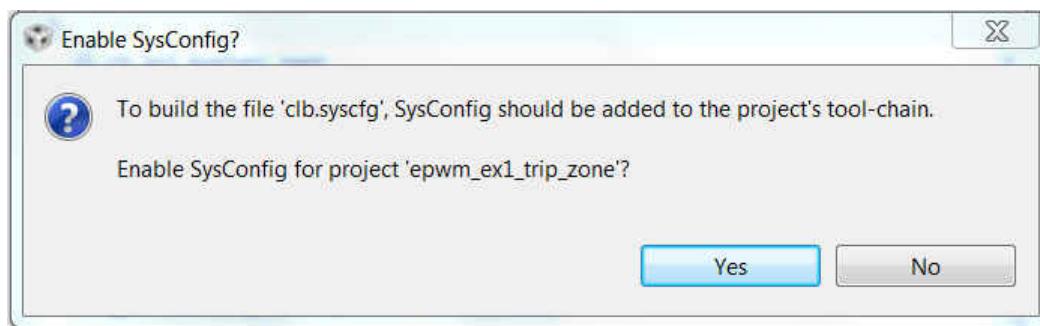


Figure 8-1. Enable SysConfig in the CCS Project

After the SysConfig feature has been enabled, you can change the settings inside the project properties to select C2000 SysConfig and choose your specific device package/part.

Detailed descriptions on how to configure the SysConfig project properties can be found at: [How to configure CCS Project Properties for C2000 SysConfig](#).

Note

C2000 SysConfig is built on top of C2000 driverlib software layer.

9 Remove C2000 SysConfig Support from Projects

Removing the C2000 SysConfig support from a project is very simple. When you delete the `syscfg` file from the project file system, SysConfig support is automatically skipped by the tool-chain. You can also right-click on the `syscfg` file and select **Exclude from Build**.

10 Standalone SysConfig Tool

You can choose to download a standalone version of the SysConfig tool instead of using the built-in CCS version. The standalone SysConfig version can be used with CCS or any IDE. The standalone SysConfig tool can be used to generate the configuration code; you can manually add the newly generated content to their C2000 project.

The standalone SysConfig tool is available for download at: [Standalone SysConfig Tool](#).

Once the standalone SysConfig tool is installed, you can launch the tool and select the C2000Ware at top folder to launch C2000 SysConfig.

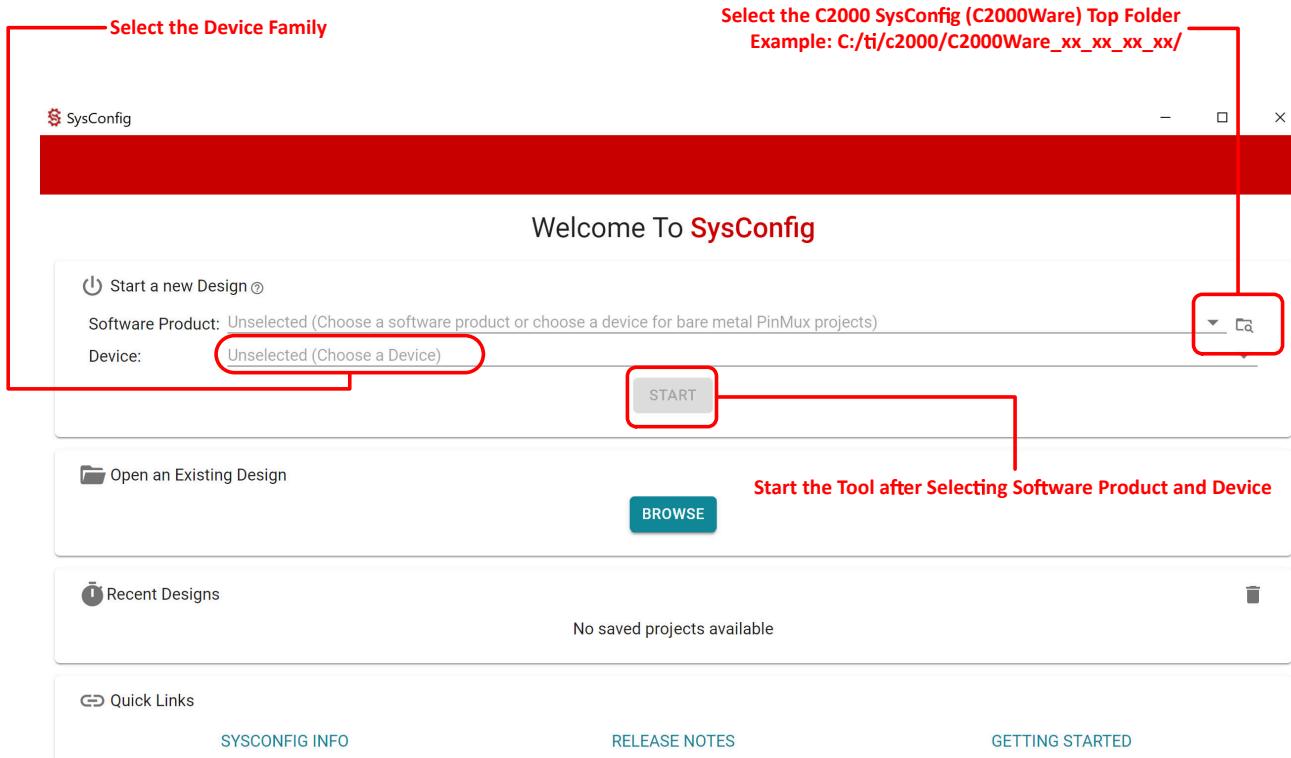


Figure 10-1. Standalone SysConfig Start Page

11 Summary

C2000 SysConfig is a powerful graphical user interface tool which configures the C2000 Real-Time MCUs and auto-generates embedded software, visualization diagrams, and debug artifacts that significantly help designers with their development process. The reliable and pre-validated initialization software generated by the C2000 SysConfig tool can speed up development and help designers avoid lengthy debug sessions.

12 References

- Texas Instruments: [*C2000™ DCSM Security Tool*](#)
- Texas Instruments: [*CLB Tool User's Guide*](#)
- Texas Instruments: [*Designing With the C2000™ Configurable Logic Block \(CLB\)*](#)
- [**TI Cloud Tools**](#)
 - [**SysConfig**](#)
 - [**Resource Explorer**](#)
- [**C2000Ware for C2000 MCUs**](#)
 - C2000 SysConfig and examples for C2000 real-time MCUs
- [**Code Composer Studio \(CCS\)**](#)
 - Integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processors
 - SysConfig tool is delivered integrated in CCS (built-in SysConfig support)
- [**SysConfig Standalone Version**](#)
 - SysConfig standalone version can be used alongside other IDEs that do not have the built-in SysConfig tool
- Texas Instruments: [*Speed up your software development by using C2000 SysConfig to configure your C2*](#)
- [**C2000 SysConfig Lab 0**](#)

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](#) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2022, Texas Instruments Incorporated