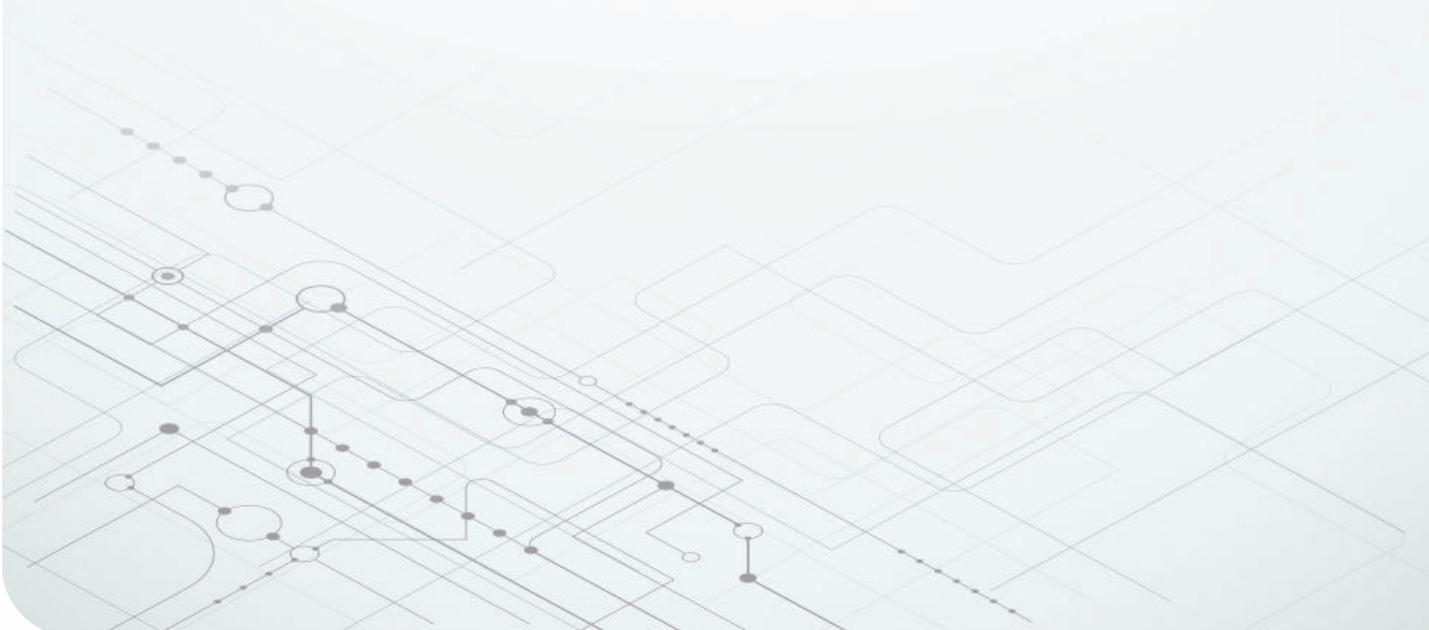


# Speed Up Development With C2000™ Real-Time MCUs Using SysConfig

---



**Nima Eskandari**  
Applications Engineer  
C2000 Real-Time MCUs



# This white paper explains how the C2000 SysConfig graphical user interface (GUI) tool is developed to facilitate the development process for designers.

---

## At a glance

---



1

### **C2000 SysConfig system initialization code generation can speed up your software development**

Reliable and pre-verified code generated by the C2000 SysConfig tool is the main feature that facilitates software development for designers. Device configuration errors are caught by tool and the developer is notified of the unsupported setting.



2

### **Integrated PinMux tool and PinMux initialization code generation closes the gap between hardware and software designers**

The C2000 SysConfig tool has built-in support for solving the device PinMux against system requirements, and provides device visualization aids.



3

### **Portable device initialization and code generation allows developers to create flexible and easily modifiable code**

The device configuration developed in C2000 SysConfig can be easily ported between device families. Also modifications in the device configuration can be easily propagated through the rest of the already developed application code.

Designers utilizing C2000™ Real-Time MCU can configure their device using C2000 SysConfig that generates reliable code and visual aids to simplify and speed up the development process. C2000 SysConfig catches incorrect device initialization settings which significantly facilitates the debugging process.

Developing embedded software for initializing an MCU can be difficult for inexperienced and experienced designers. New embedded devices with more advanced features and peripherals enter the market every year, making the task of configuring the device much more demanding. Small mistakes in configuring the device and its peripherals can cause the development process to slow down and frustrate the designers.

With advancements in embedded technology, both the size and capabilities of the embedded devices have grown. This makes the task of device resource management very important. Knowing which peripherals and which device pins are used and which ones are available for further development is useful for both hardware and software designers.

Selecting the correct family of devices for a specific application can also be difficult because the requirement for the system can change. Portability of initialization settings and embedded software between device families eases the minds of developers, because they can reuse their previous investment in software development when migrating between different devices.

This white paper explores how the C2000 SysConfig tool can simplify the tasks that embedded hardware and software developers face when designing a system with C2000 real-time MCUs.

### **C2000 SysConfig system initialization code generation can speed up your software development**

Developing initialization code is the first step in software development for an embedded application. This can be a tedious task and many mistakes are made due to small coding errors and not following the exact instructions given in the device's technical reference

manual. Once the device is correctly initialized, the designer can continue on to develop the software for their specific application. Confidence in the correctness of the device initialization is much higher when the code developed has been previously verified by device experts. C2000 real-time MCUs can be initialized through C2000 SysConfig which generates reliable and pre-verified code for configuring your device. Device configuration errors are caught by the tool, and the developer is notified of the unsupported settings. C2000

SysConfig tool can also configure the device PinMux and visualize the device pins for each package.

C2000 SysConfig is delivered through C2000Ware (C2000 real-time MCU software development kit) and can be used with Code Composer Studio™ (CCS) IDE's built-in SysConfig (System Configuration) tool or with any other supported IDE through the SysConfig tool's standalone version.

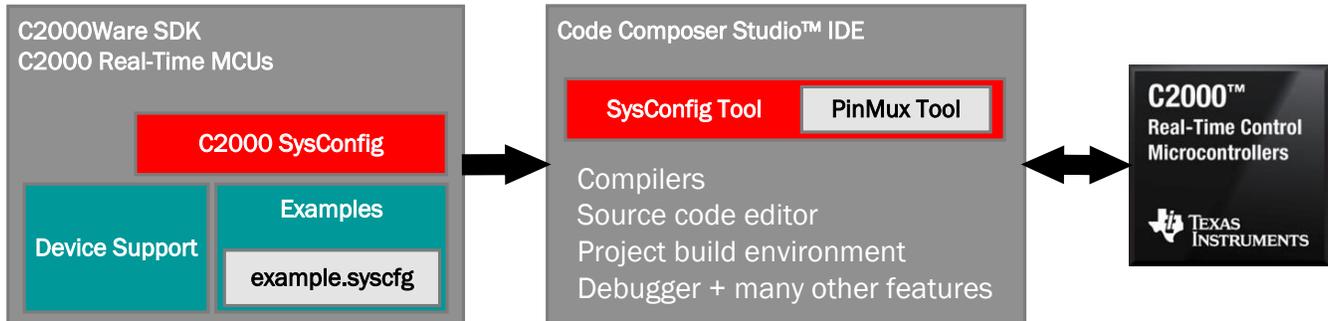


Figure 1. C2000 SysConfig in C2000Ware.

## Peripheral configuration

The C2000 SysConfig allows designers to configure their peripherals through the SysConfig GUI.

Device peripherals are listed in the C2000 SysConfig tool so the designer is aware of the peripherals available in their specific device package. The configurable options for each peripheral is listed, which allows the designer to see all the different available modes. The device level inter-connects are displayed in the tool that shows the available list of signals for each MUX previously described only in the technical documentation.

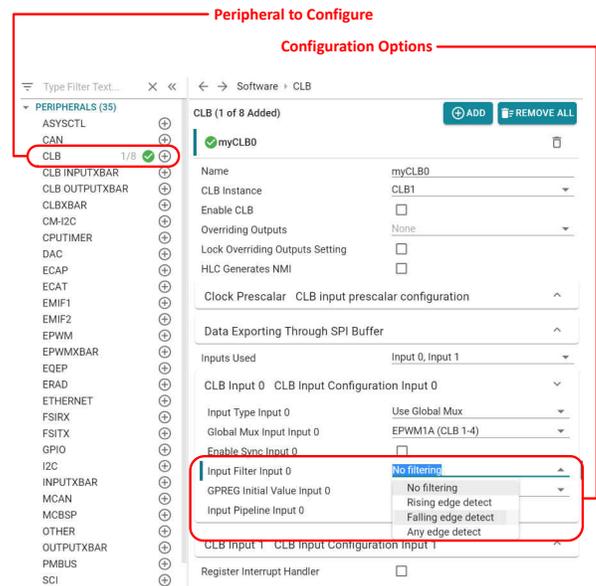


Figure 2. Peripheral configuration.

More complicated peripherals such as the Configurable Logic Block (CLB), which is capable of creating custom logic inside the device, or the Dual Code Security Module (DCSM), which is used of securing the customer's intellectual property, are also included in the C2000 SysConfig ecosystem. These add-on tools will show up

automatically in the tool and the designer has the options of using them in their application. The additional auto-generated artifacts from these tools will be presented to the designer seamlessly.

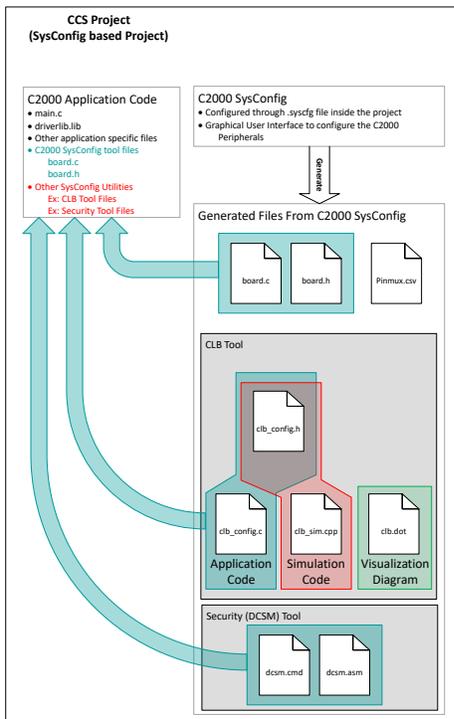


Figure 3. C2000 SysConfig CCS project overview.

## Code generation

Automatic code generation allows developers to be confident in the correctness of their initialization code for their embedded application. As configurable options are changed, the designers can view the differences in the auto-generated code. This allows developers to understand how the embedded software changes when the requirements for the device configuration are altered.

C2000 SysConfig's auto-generated code is structured so that the designer can pick which section of the code they wish to add to their application. The designer can choose to add all of the auto-generated code or they can choose to use only the PinMux initialization feature, or they can choose to use only the initialization for a specific peripheral. This eliminates the concerns that an experienced designer would have for adopting this tool, when they already have existing and verified code. Designers who already have existing code, can compare their initialization code with the auto-generated one by C2000 SysConfig as a verification step since the auto-generated code was designed by C2000 real-time MCU experts.

**Changed Configurable**

**Automatic Code Generation**

CLB (1 of 8 Added) + ADD REMOVE ALL

myCLB0 🗑

Name	myCLB0
CLB Instance	CLB1
Enable CLB	<input type="checkbox"/>
Overriding Outputs	None
Lock Overriding Outputs Setting	<input type="checkbox"/>
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

Input Type Input 0	Use Global Mux
Global Mux Input Input 0	EPWM1A (CLB 1-4)
Enable Sync Input 0	<input type="checkbox"/>
Input Filter Input 0	No filtering
GPREG Initial Value Input 0	0
Input Pipeline Input 0	<input checked="" type="checkbox"/>

CLB Input 1 CLB Input Configuration Input 1 ^

Register Interrupt Handler

```
board.c
48 48
49 49 }
50 50
51 51 void CLB_init(){
52 52 //myCLB0 initialization
53 53 CLB_setOutputMask(myCLB0_BASE,
54 54 (0 << 0), true);
55 55 CLB_enableOutputMaskUpdates(myCLB0_BASE);
56 56 CLB_disableSPIBufferAccess(myCLB0_BASE);
57 57 CLB_configSPIBufferLoadSignal(myCLB0_BASE, 0);
58 58 CLB_configSPIBufferShift(myCLB0_BASE, 0);
59 59 //myCLB0 CLB_IN0 initialization
60 60 CLB_configLocalInputMux(myCLB0_BASE, CLB_IN0, CLB_LOC
61 61 CLB_configGlobalInputMux(myCLB0_BASE, CLB_IN0, CLB_GL
62 62 CLB_configGPInputMux(myCLB0_BASE, CLB_IN0, CLB_GP_IN_
63 63
64 64 CLB_selectInputFilter(myCLB0_BASE, CLB_IN0, CLB_FILTE
65 65 - CLB_disableInputPipelineMode(myCLB0_BASE, CLB_IN0);
65 + CLB_enableInputPipelineMode(myCLB0_BASE, CLB_IN0);
66 66
67 67 //myCLB0 CLB_IN1 initialization
68 68 CLB_configLocalInputMux(myCLB0_BASE, CLB_IN1, CLB_LOC
69 69 CLB_configGlobalInputMux(myCLB0_BASE, CLB_IN1, CLB_GL
70 70 CLB_configGPInputMux(myCLB0_BASE, CLB_IN1, CLB_GP_IN_
71 71
72 72 CLB_selectInputFilter(myCLB0_BASE, CLB_IN1, CLB_FILTE
73 73 CLB_disableInputPipelineMode(myCLB0_BASE, CLB_IN1);
74 74
75 75 CLB_setGPREG(myCLB0_BASE,0);
76 76 CLB_disableCLB(myCLB0_BASE);
77 77 }
78 78
79 79
```

**Figure 4.** Code generation and difference identification.

## Error detection

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 which sometimes are missed by the designers.

It is common that the development process for configuring a device is slowed down due to errors in the designer'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 the user of the incorrect setup.

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

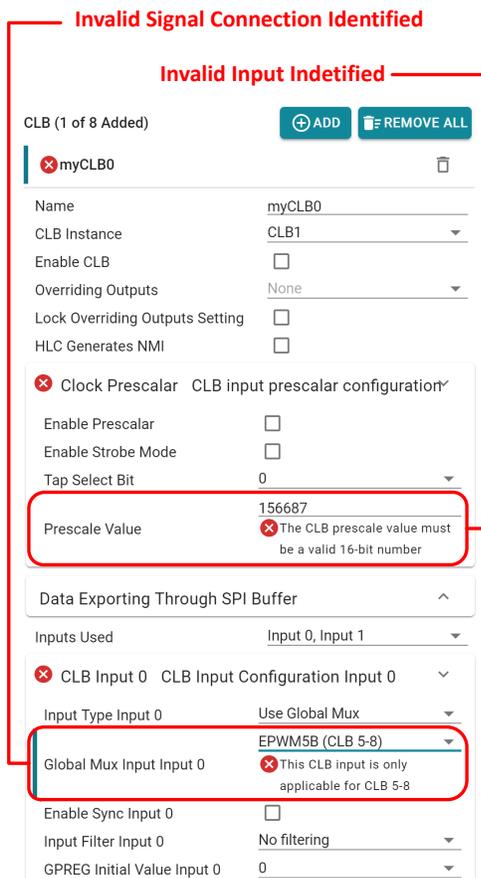


Figure 5. Error detection.

## Device level dependencies

Device level dependencies can be missed by developers. Not configuring all dependencies required for a peripheral to operate correctly is a common mistake. The C2000 SysConfig tool identifies the dependencies in the device and ensures that these dependencies are configured by the designer.

C2000 real-time MCUs are highly configurable and their inter-connections, which reduces signal chain latency and eliminates requirements for external components, create dependencies between peripherals inside the device. For example, the Analog-to-Digital Converter (ADC), the enhanced Pulse Width Modulator (ePWM) and Comparator Subsystem (CMPSS) could all be inter-connected to one another inside the device.

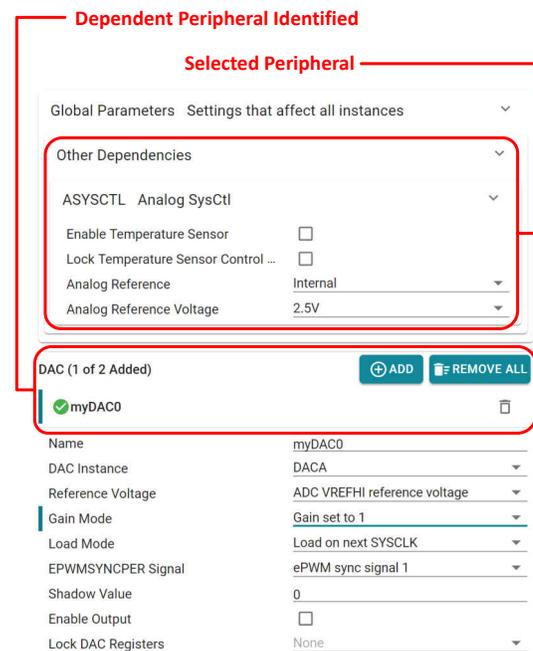


Figure 6. Identifying dependencies.

## Device level error detection

Error detection in the designer's 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.

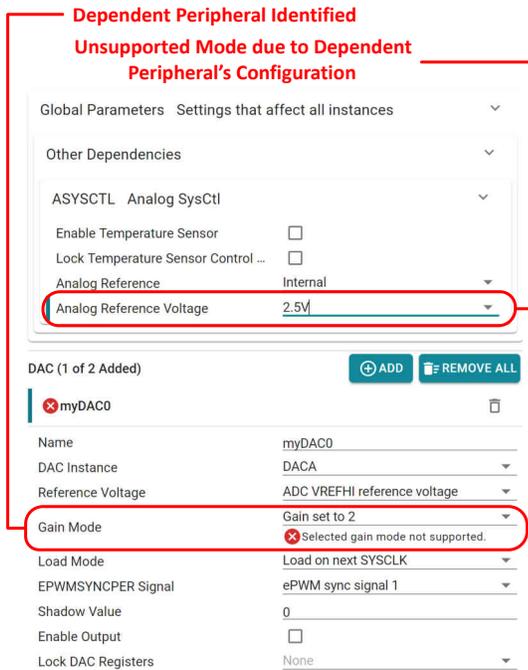


Figure 7. Detecting errors in dependencies.

## Integrated PinMux tool and PinMux initialization code generation closes the gap between hardware and software designers

Solving the device PinMux for a given application can be a difficult task. Resource management, knowing which peripherals and pins have been used and which ones are free, is extremely important in determining if the MCU is capable of all of the application's requirements. The C2000 SysConfig tool has built-in support for device PinMux, and can solve the device PinMux, and also auto-generates PinMux initialization code along with a summary CSV file. C2000 SysConfig will also visualize the device package and show the used and free package pins.

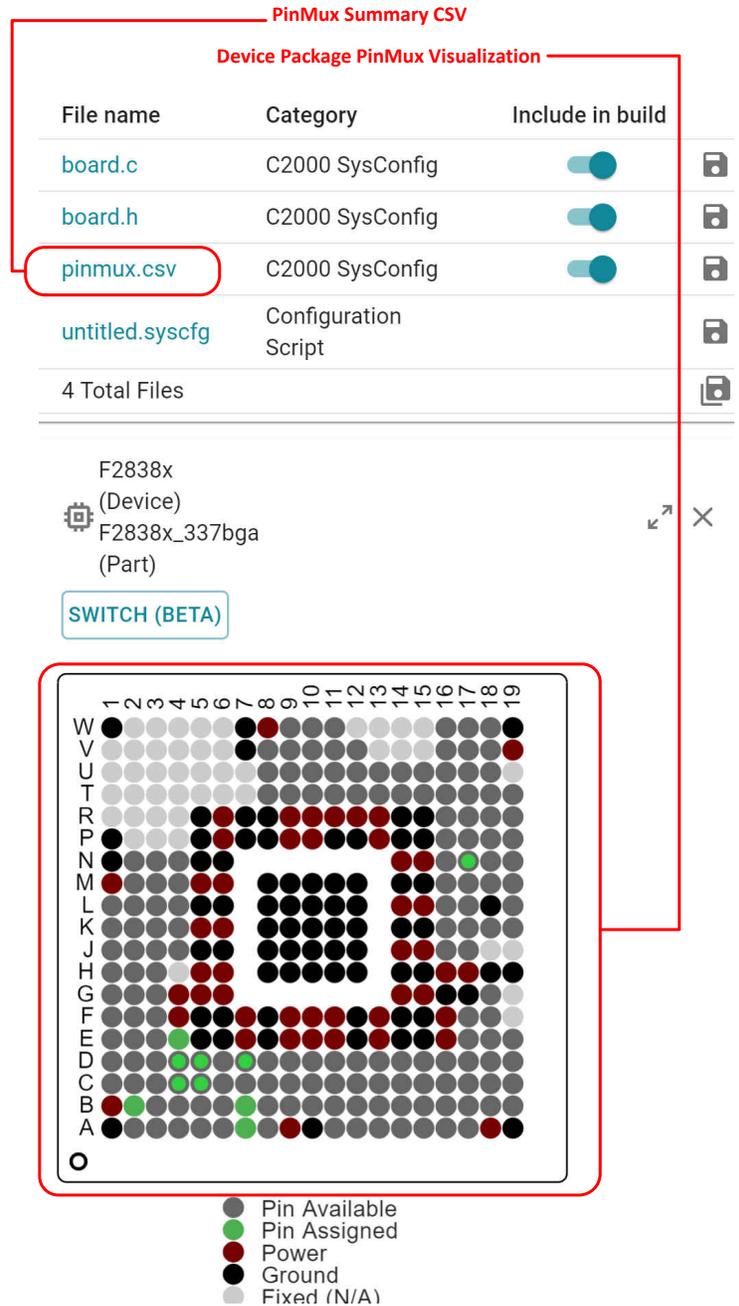


Figure 8. Device package PinMux support.

The pinmux.csv file bridges the gap between the hardware and software designer.

The Selected PinMux Option for the Application

Pin	Name	Selected Mode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
109	C9	GPIO161	GPIO161	EPWM9A			GPIO161				GPIO161		ESC_GPO28	GPIO161		ESC_TX0_DATA3		
110	D9	GPIO162	GPIO162	EPWM9B			GPIO162				GPIO162		ESC_GPO29	GPIO162		ESC_RX0_DV		
111	A8	GPIO163	GPIO163	EPWM10A			GPIO163				GPIO163		ESC_GPO30	GPIO163		ESC_RX0_CLK		
112	B8	GPIO164	GPIO164	EPWM10B			GPIO164				GPIO164		ESC_GPO31	GPIO164		ESC_RX0_ERR		
113	C8	GPIO0	GPIO0	EPWM1A			GPIO0		I2CA_SDA		GPIO0		CM-I2CA_5ESC_GPIO0	GPIO0		FSITXA_D0		
114	D8	GPIO1	GPIO1	EPWM1B		MFSRB	GPIO1		I2CA_SCL		GPIO1		CM-I2CA_5ESC_GPIO1	GPIO1		FSITXA_D1		
115	A7	GPIO2	I2CB_SDA	GPIO2	EPWM2A		GPIO2		OUTPUTXE_I2CB_SDA		GPIO2		ESC_GPIO2	GPIO2		FSITXA_CLK		
116	B7	GPIO3	I2CB_SCL	GPIO3	EPWM2B	OUTPUTXEMCLKRB	GPIO3		OUTPUTXE_I2CB_SCL		GPIO3		ESC_GPIO3	GPIO3		FSIRXA_D0		
117	C7	GPIO4	GPIO4	EPWM3A			GPIO4		OUTPUTXECANA_TX		GPIO4		MCAN_TX_ESC_GPIO4	GPIO4		FSIRXA_D1		
118	D7	GPIO5	MFSRA	GPIO5	EPWM3B	MFSRA	OUTPUTXEGPIO5		CANA_RX		GPIO5		MCAN_RX_ESC_GPIO5	GPIO5		FSIRXA_CLK		
119	A6	GPIO6	GPIO6	EPWM4A	OUTPUTXEXTSYNCO	GPIO6	EQEP3_A		CANB_TX		GPIO6		ESC_GPIO6	GPIO6		FSITXB_D0		
120	B6	GPIO7	GPIO7	EPWM4B	MCLKRA	OUTPUTXEGPIO7	EQEP3_B		CANB_RX		GPIO7		ESC_GPIO7	GPIO7		FSITXB_D1		
121	C6	GPIO88	GPIO88		EMIF1_A1	EMIF1_DQ	GPIO88				GPIO88		EMIF1_DQM1	GPIO88		ESC_TX0_DATA1		
122	D6	GPIO89	GPIO89		EMIF1_A1	EMIF1_DQ	GPIO89		SCIC_TX		GPIO89		EMIF1_CAS	GPIO89		ESC_TX0_DATA2		
123	A5	GPIO90	GPIO90		EMIF1_A1	EMIF1_DQ	GPIO90		SCIC_RX		GPIO90		EMIF1_RAS	GPIO90		ESC_TX0_DATA3		
124	B5	GPIO91	GPIO91		EMIF1_A1	EMIF1_DQ	GPIO91		I2CA_SDA		GPIO91		EMIF1_DQ_PMBUSA_5SSIA_TX	GPIO91		FSIRXF_D0	CLB_OUTP_SPID_SIMC	
125	C5	GPIO165	MDXA	GPIO165	EPWM11A		GPIO165				GPIO165		MDXA	GPIO165		ESC_RX0_DATA0		
126	A4	GPIO92	GPIO92		EMIF1_A1	EMIF1_BA	GPIO92		I2CA_SCL		GPIO92		EMIF1_DQ_PMBUSA_5SSIA_RX	GPIO92		FSIRXF_D1	CLB_OUTP_SPID_SOM	
127	D5	GPIO166	MDRA	GPIO166	EPWM11B		GPIO166				GPIO166		MDRA	GPIO166		ESC_RX0_DATA1		
128	B4	GPIO93	GPIO93		EMIF1_A2	EMIF1_BA	GPIO93		SCID_TX		GPIO93		PMBUSA_5SSIA_CLK	GPIO93		FSIRXF_CLI	CLB_OUTP_SPID_CLK	
129	C4	GPIO167	MCLKXA	GPIO167	EPWM12A		GPIO167				GPIO167		MCLKXA	GPIO167		ESC_RX0_DATA2		
130	A3	GPIO94	GPIO94		EMIF1_A21		GPIO94		SCID_RX		GPIO94		EMIF1_BA_PMBUSA_5SSIA_FSS	GPIO94		FSIRXF_D0	CLB_OUTP_SPID_STEN	
131	D4	GPIO168	MFSXA	GPIO168	EPWM12B		GPIO168				GPIO168		MFSXA	GPIO168		ESC_RX0_DATA3		
132	B3	GPIO95	GPIO95		EMIF2_A1	GPIO95					GPIO95			GPIO95		FSIRXG_D1	CLB_OUTPUTXBAR5	
133	C3	GPIO96	GPIO96		EMIF2_DQ	GPIO96	EQEP1_A				GPIO96			GPIO96		FSIRXG_CL	CLB_OUTPUTXBAR6	
134	A2	GPIO97	GPIO97		EMIF2_DQ	GPIO97	EQEP1_B				GPIO97			GPIO97		FSIRXH_D0	CLB_OUTPUTXBAR7	

Figure 9. PinMux summary table.

### Portable device initialization and code generation allows developers to create flexible and easily modifiable code

The device configuration set in C2000 SysConfig for a specific device family and package can be ported to other device families and packages. When migrating between device families, C2000 SysConfig will automatically update its code generation templates and output the correct embedded software for the new device family. This can simplify the task of porting device initialization code for designers who are migrating between device families.

C2000 SysConfig also allows designers to name their device resources to their application specific name. This allows the initialization code and the runtime application

code to be much more flexible to changes in PinMux and resource instances. For example, if the designer decides to change the GPIO number used for a task in the application, no change would be necessary to the runtime application code if the task GPIO was named through C2000 SysConfig.

In the case that some resources are not available when migrating between device families or device packages, the user is notified through warning and errors generated by the tool. If the tool is able to reassign a new resource, which can accomplish the same task, the designer is notified through a warning. If the tool is **NOT** able to reassign a new resource, the tool generates an error and forces the designer to either remove the requirement or reassign a new resource manually, which meets the same requirements.

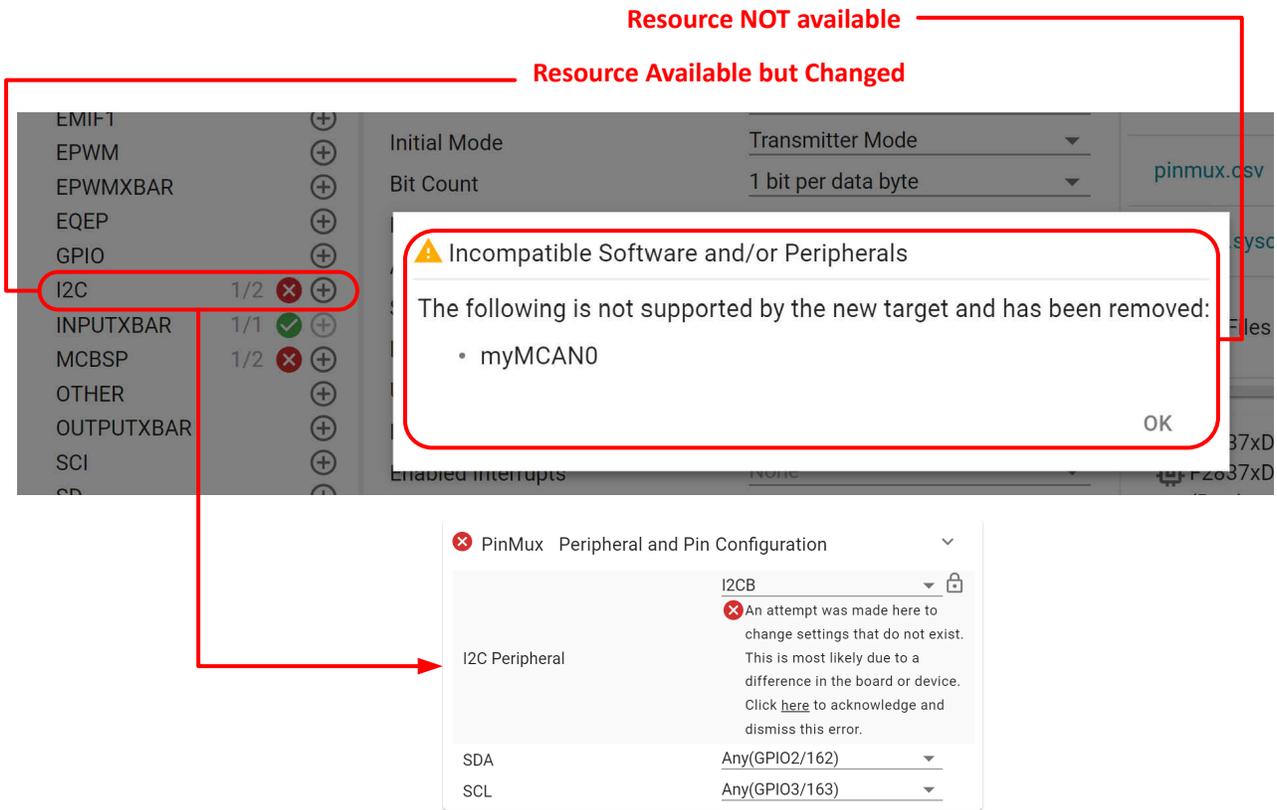


Figure 10. Migrating between device families and device packages.

## Conclusion

C2000 SysConfig is a powerful graphical user interface tool that configures the C2000 real-time MCUs and auto-generates embedded software, visualization diagrams, and debug artifacts that helps designers significantly 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.

## Additional resources

- **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 which do not have the built-in SysConfig tool
- Texas Instruments: **C2000 SysConfig**
  - Step by step instructions for C2000 SysConfig
- **C2000 SysConfig Lab 0**

**Important Notice:** The products and services of Texas Instruments Incorporated and its subsidiaries described herein are sold subject to TI's standard terms and conditions of sale. Customers are advised to obtain the most current and complete information about TI products and services before placing orders. TI assumes no liability for applications assistance, customer's applications or product designs, software performance, or infringement of patents. The publication of information regarding any other company's products or services does not constitute TI's approval, warranty or endorsement thereof.

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.  
All trademarks are the property of their respective owners.

## 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](http://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 © 2021, Texas Instruments Incorporated