Application Note

Accelerating Development with SysConfig using MCU+SDK

Tushar Thakur, Anil Swargam, Soumya Tripathy

ABSTRACT

This application note explains the role of the SysConfig tool integration with MCU+SDK for AM243x, AM275x and AM6x devices. SysConfig simplifies bring-up by auto-generating source files for pin multiplexing, clock configuration, power domain configuration, driver configuration, board peripheral configuration, Region Address Translation (RAT), Memory Management Unit (MMU) and Memory Protection Unit (MPU) configuration.

The Application note also provides step-by-step guidance & example use cases and troubleshooting tips to help developers use SysConfig effectively with TI SOCs.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

SysConfig is an interactive configuration tool integrated with MCU+SDK that automates device initialization and driver setup for TI SoCs. This detects configuration conflicts, generates initialization files, and simplifies integration into custom software projects or MCU+SDK projects. Developers can use SysConfig to configure clocks, PinMux, MPU/MMU/RAT regions, and driver instances through an intuitive GUI or command-line interface.

The following are the features supported in the tool:

- **System Initialization:** SysConfig (CodeGen) tool generates initialization code for AM243x, AM275x and AM6x devices, covering peripheral setup, clock configuration, interrupt handling, PinMux configuration, and MPU, MMU and RAT settings. See *System Initialization* for details.
- **PinMux Visualization:** The tool provides a graphical view of the device and the pins, displaying all possible PinMux options and highlighting the user-selected mode for each pin. See *Example Sysconfig in CCS*(5) for details.
- **Error Detection:** SysConfig validates configurations and reports errors in case of incorrect setups. This automatically detects conflicts between pin assignments. See *Pin Conflict* for details.
- **Dependency Identification:** The tool identifies inter-module dependencies within the device and makes sure that required peripherals are configured consistently.
- **Resource Conflict Detection:** When a module depends on another peripheral, SysConfig checks for conflicts. If the dependent peripheral is already in use, the tool flags a resource conflict error. See *Resource Conflict* for details.

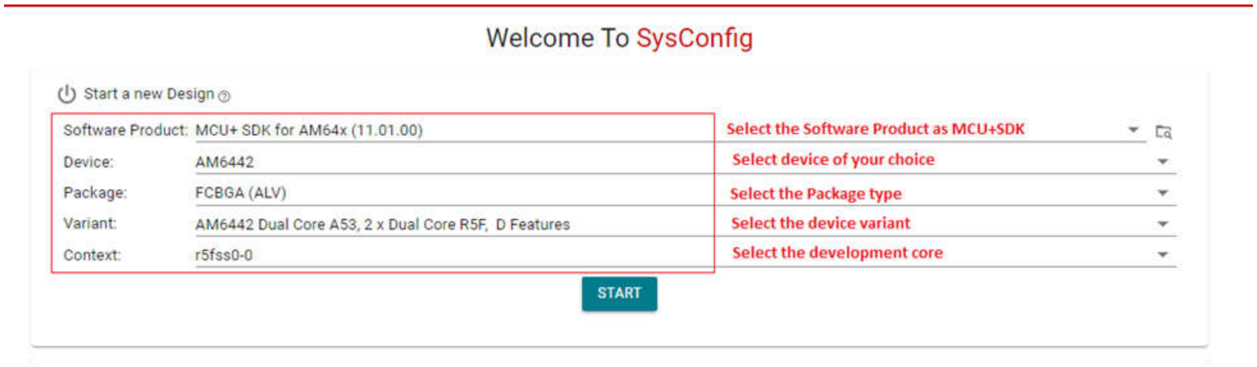
Note: The device families supported are:

- AM243x, AM64x
- AM62Lx
- AM62Ax
- AM62Dx, AM275x
- AM62Px, AM62x

1.1 SysConfig CodeGen Tool

The SysConfig CodeGen tool generates source and header files which are used with MCU SDK examples to achieve the above-mentioned functionalities. The CodeGen tool internally uses the Sciclient (TISCI) APIs provided by MCU+SDK to manage clocks, resets, and power domains through communication with the DMSC firmware.

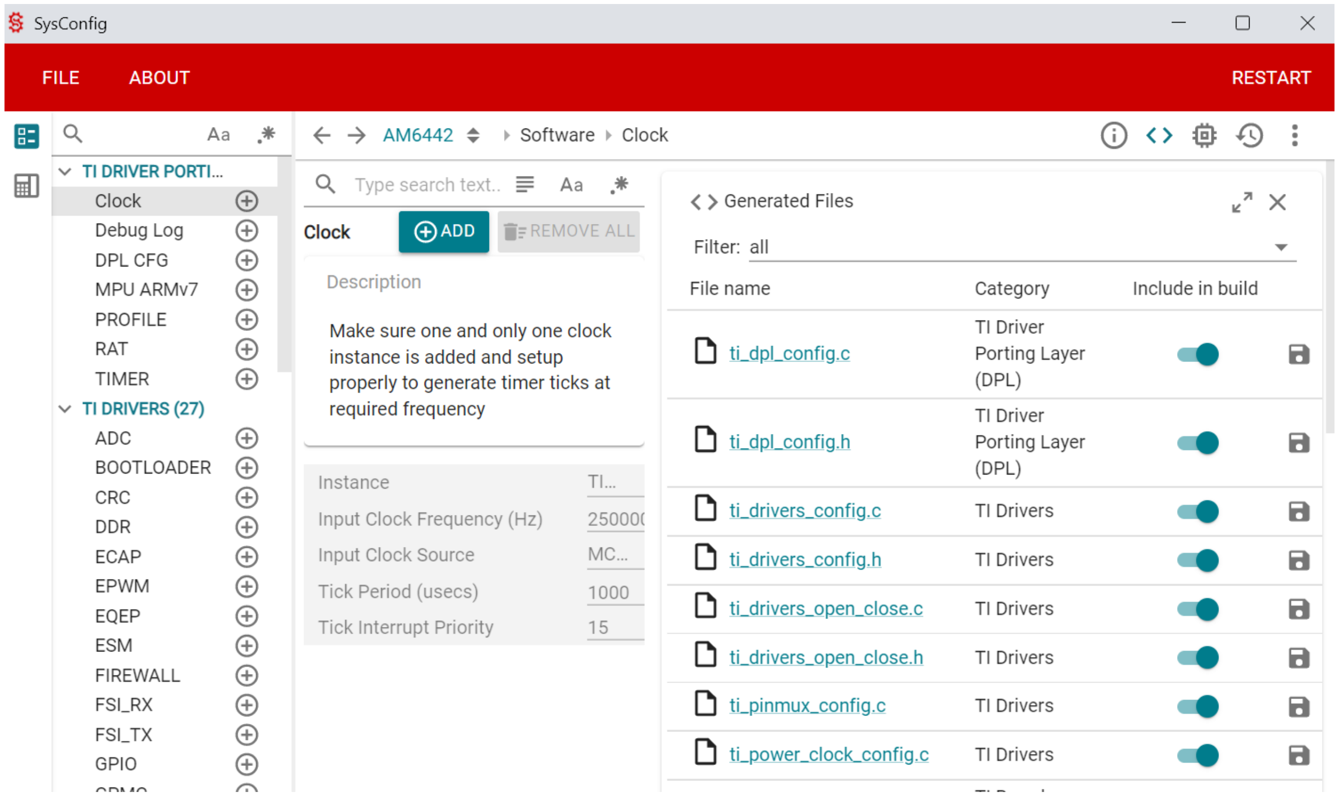
See the parameters in Sysconfig CodeGen Tool which are required to open the SysConfig CodeGen tool.



Software Product:	MCU+ SDK for AM64x (11.01.00)	Select the Software Product as MCU+SDK
Device:	AM6442	Select device of your choice
Package:	FCBGA (ALV)	Select the Package type
Variant:	AM6442 Dual Core A53, 2 x Dual Core R5F, D Features	Select the device variant
Context:	r5fss0-0	Select the development core

Figure 1-1. SysConfig CodeGen Tool

View of SysConfig CodeGen Tool shown in CodeGen Tool Generated Files.



Clock

Description

Make sure one and only one clock instance is added and setup properly to generate timer ticks at required frequency

Instance	TI...
Input Clock Frequency (Hz)	250000
Input Clock Source	MC...
Tick Period (usecs)	1000
Tick Interrupt Priority	15

Filter: all

File name	Category	Include in build
ti_dpl_config.c	TI Driver Porting Layer (DPL)	●
ti_dpl_config.h	TI Driver Porting Layer (DPL)	●
ti_drivers_config.c	TI Drivers	●
ti_drivers_config.h	TI Drivers	●
ti_drivers_open_close.c	TI Drivers	●
ti_drivers_open_close.h	TI Drivers	●
ti_pinmux_config.c	TI Drivers	●
ti_power_clock_config.c	TI Drivers	●

Figure 1-2. CodeGen Tool Generated Files

2 Getting Started Guide

The SysConfig tool is available both for both online and offline development.

Please refer to the link below to access the tool.

1. To Download the offline version of Sysconfig tool, see https://www.ti.com/tool/download/SYSCONFIG.
2. The online version of the tool can be accessed through https://dev.ti.com/sysconfig/#/start

Note

The online version of the tool is the latest version of SysConfig. Check the release note of the MCU SDK for compatibility issue while using the online SysConfig tool.

2.1 How to Launch SysConfig (GUI and Command-Line)

SysConfig tool can be launched either using the GUI (Graphical User Interface) or through CLI (Command Line Interface).

- To open the tool with the GUI, navigate to the Sysconfig directory and double click on **sysconfig_gui.bat** file.
- To open the SysConfig CodeGen tool via CLI follow the steps below.
  - Navigate to the example directory until the makefile is visible.
  - Run the command below. Use *make* for Linux and *gmake* for Windows.

```
> {gmake|make} -s syscfg-gui
```

- To run the Sysconfig tool through CLI, use the following command. The following command outputs all the files generated by the SysConfig CodeGen Tool.

```
> cd ${SysConfig_root} > sysconfig_cli.bat -s ${MCU+SDK_root}\.metadata\product.json -d AM6442 -o
${Project_path}\debug ${Project_path}\example.syscfg
```

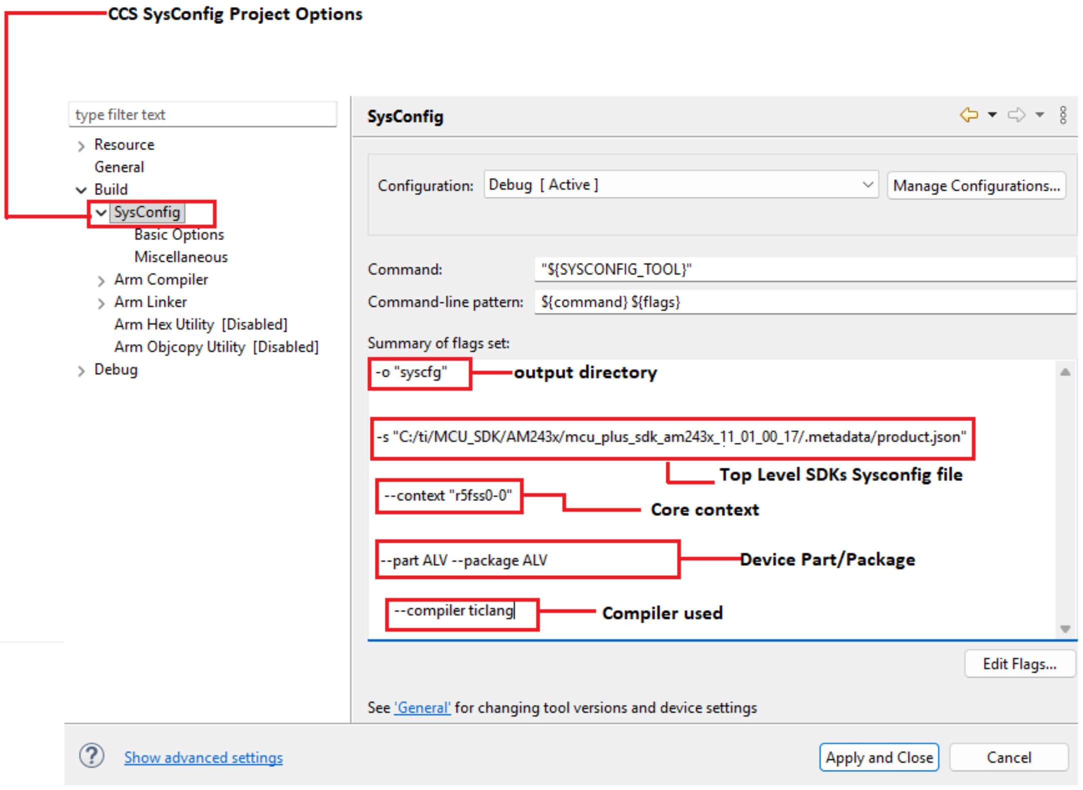
2.2 Integration with CCS and Makefile builds

The SysConfig tool begins with the **product.json** file which contains all the information for the CodeGen tool.

The information mentioned here is applicable to both SysConfig standalone tool and integrated with CCS.

To view the SysConfig project properties in your CCS project.

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



Configuration: Debug [Active]

Command: "${SYSCONFIG_TOOL}"

Command-line pattern: ${command} ${flags}

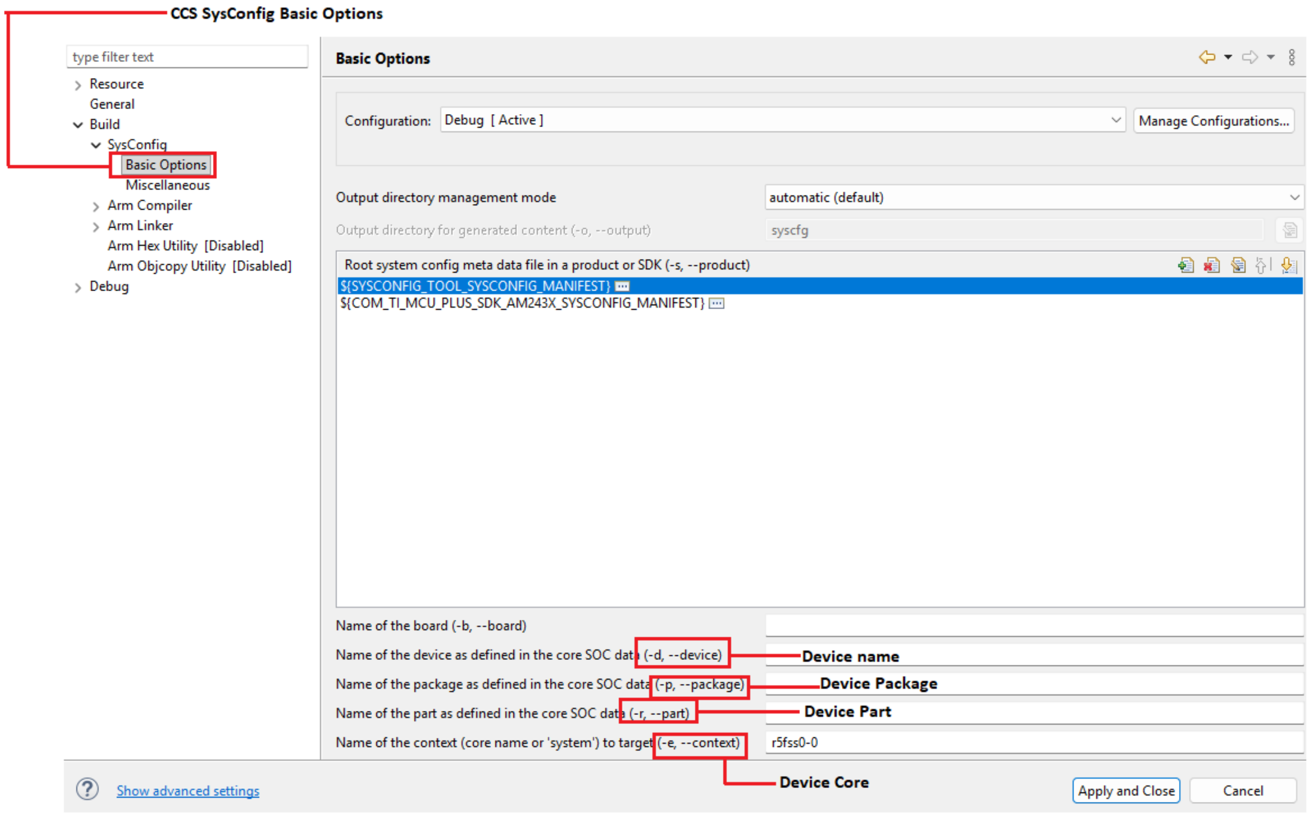
Summary of flags set:

```
-o "syscfg"
-s "C:/ti/MCU_SDK/AM243x/mcu_plus_sdk_am243x_11_01_00_17/.metadata/product.json"
--context "r5fss0-0"
--part ALV --package ALV
--compiler ticlang
```

See 'General' for changing tool versions and device settings

Figure 2-1. CCS SysConfig Project Properties

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



Configuration: Debug [Active]

Output directory management mode: automatic (default)

Output directory for generated content (-o, --output): syscfg

Root system config meta data file in a product or SDK (-s, --product)

${SYSCONFIG_TOOL_SYSCONFIG_MANIFEST}

${COM_TI_MCU_PLUS_SDK_AM243X_SYSCONFIG_MANIFEST}

Name of the board (-b, --board)

Name of the device as defined in the core SOC data (-d, --device)

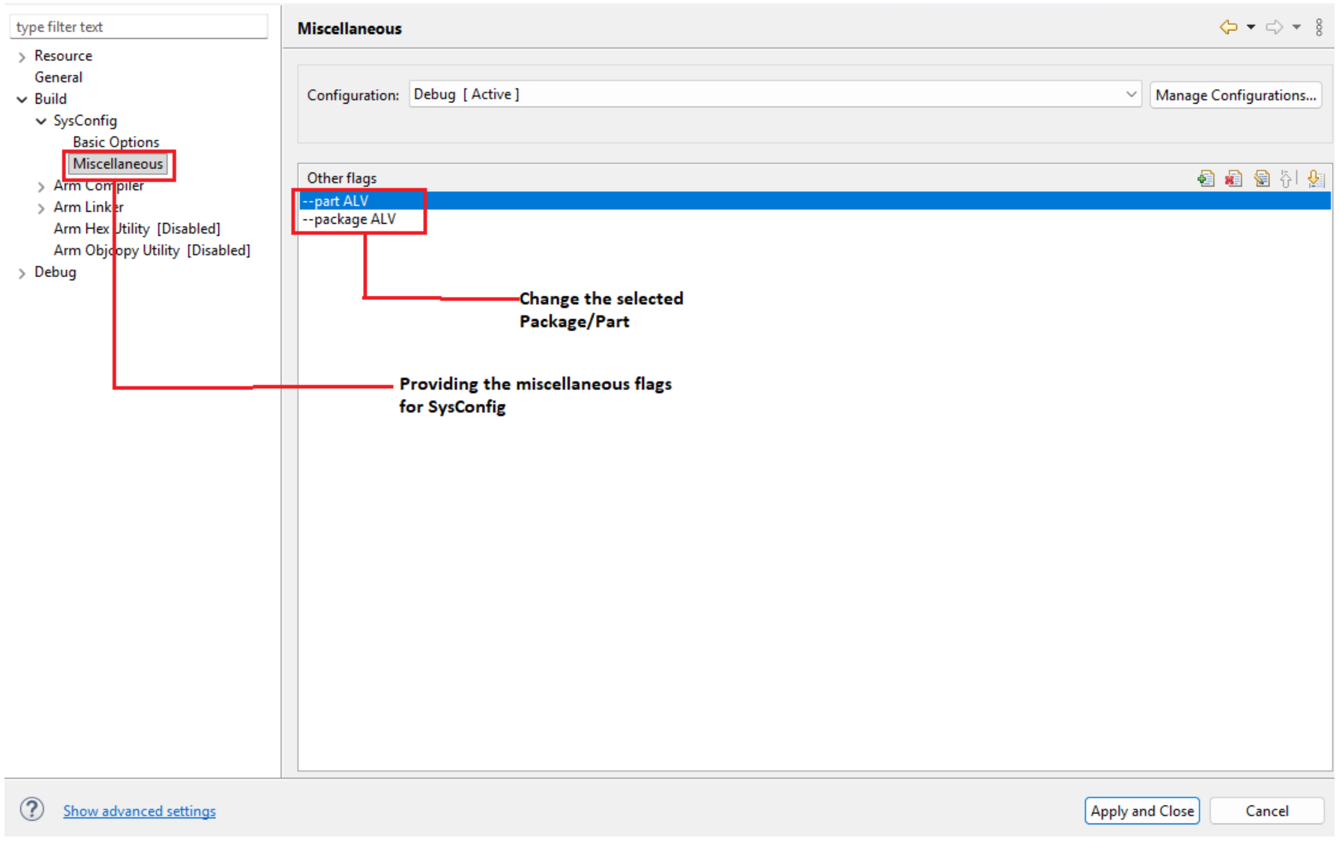
Name of the package as defined in the core SOC data (-p, --package)

Name of the part as defined in the core SOC data (-r, --part)

Name of the context (core name or 'system') to target (-e, --context): r5fss0-0

Figure 2-2. SysConfig Basic Options

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



--part ALV
--package ALV

Change the selected Package/Part

Providing the miscellaneous flags for SysConfig

Figure 2-3. SysConfig Miscellaneous Options

2.3 Location of SysConfig file in MCU SDK

2.3.1 Using Existing SysConfig File

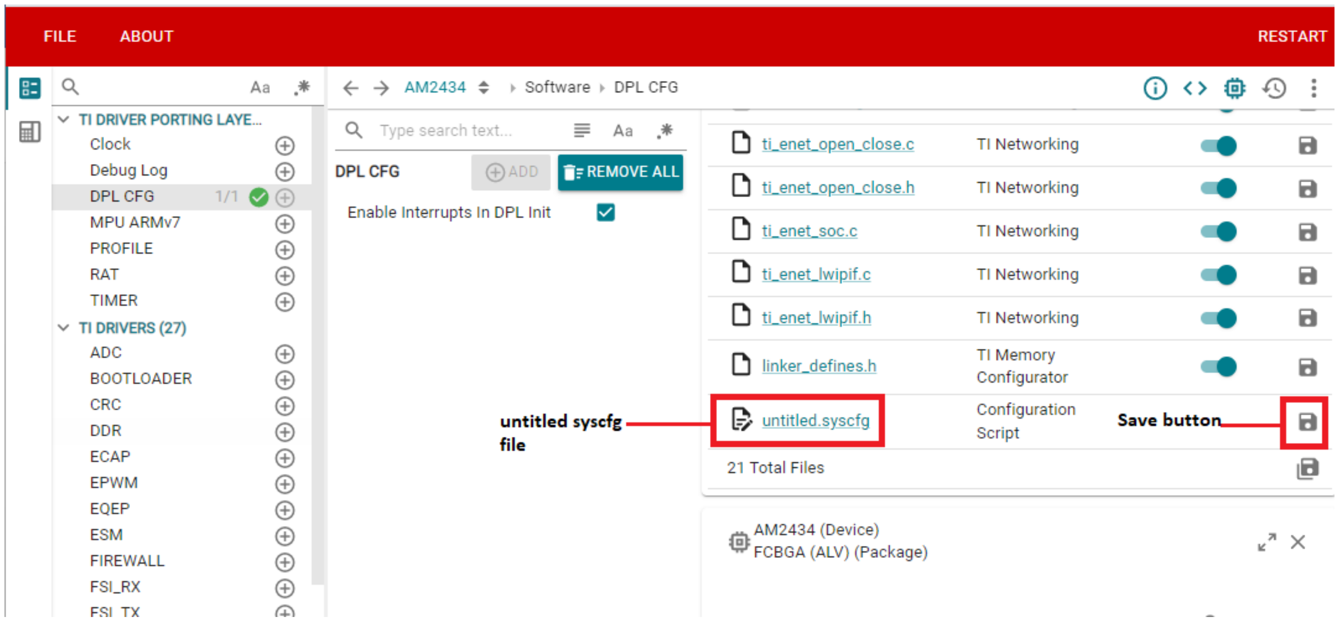
Every example provided in the MCU+SDK contains an **example.syscfg** file which contains details of which peripherals are configured and initialized through the SysConfig CodeGen Tool. The tool takes this file as input and generates the required output files for the configured peripherals.

The **example.syscfg** file is located at **${MCU+SDK}/examples/${name}/${device}/${core}/example.sysconfig**

2.3.2 Creating New SysConfig File

The **example.syscfg** file can be generated from scratch by opening the CodeGen Tool as specified in SysConfig CodeGen Tool.

The tool automatically generates the **untitled.syscfg** file which can be saved and then later used with the MCU SDK.



DPL CFG

Enable Interrupts In DPL Init ✓

ti_enet_open_close.c	TI Networking
ti_enet_open_close.h	TI Networking
ti_enet_soc.c	TI Networking
ti_enet_lwipif.c	TI Networking
ti_enet_lwipif.h	TI Networking
linker_defines.h	TI Memory Configurator
untitled.syscfg	Configuration Script

21 Total Files

Figure 2-4. Generated syscfg file

3 Example Sysconfig in CCS

3.1 Example I2C Read

To get started with SysConfig CodeGen Tool, import an existing example provided in the MCU SDK with SysConfig support.

1. Launch CCS and import the example: **i2c_read_r5fss0-0_nortos**
   a. Select **Project → Import CCS Project**
   b. Browse to **${MCU+SDK}\examples\drivers\i2c\i2c_read\am64x-evm\r5fss0-0_nortos**
   c. Select the project and import the project.
2. Inside the CCS project, the user can see the syscfg file along with the rest of the application files.

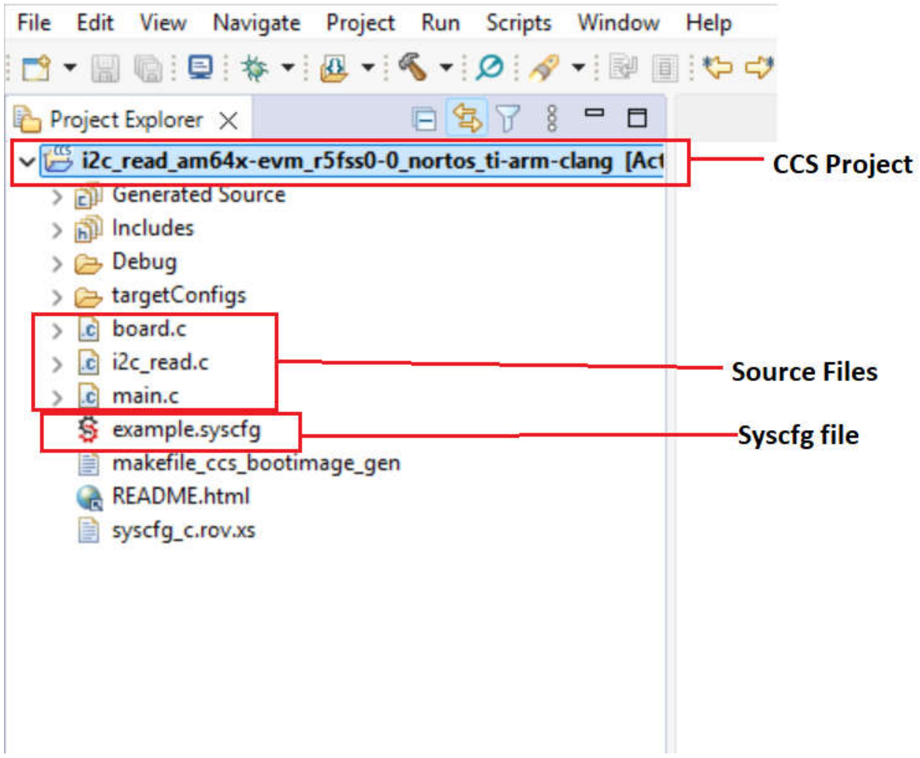


Figure 3-1. Example Project

3. Double click on **example.syscfg** file and the SysConfig GUI launches.

Note

Right-click on the syscfg file, then select Open With → SysConfig Editor.

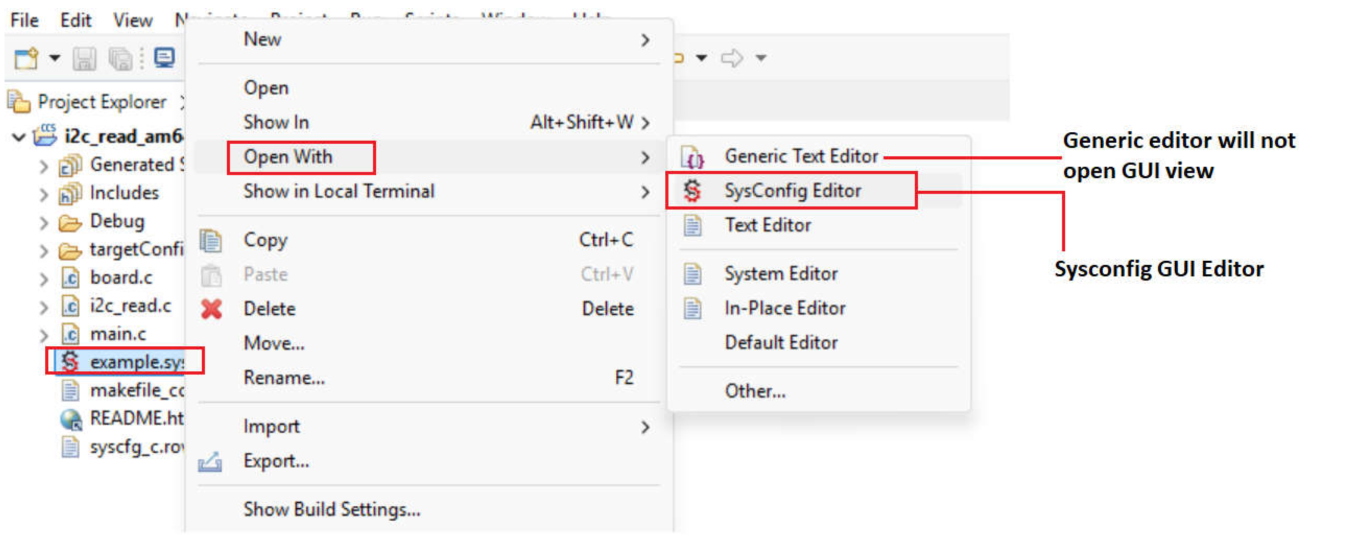
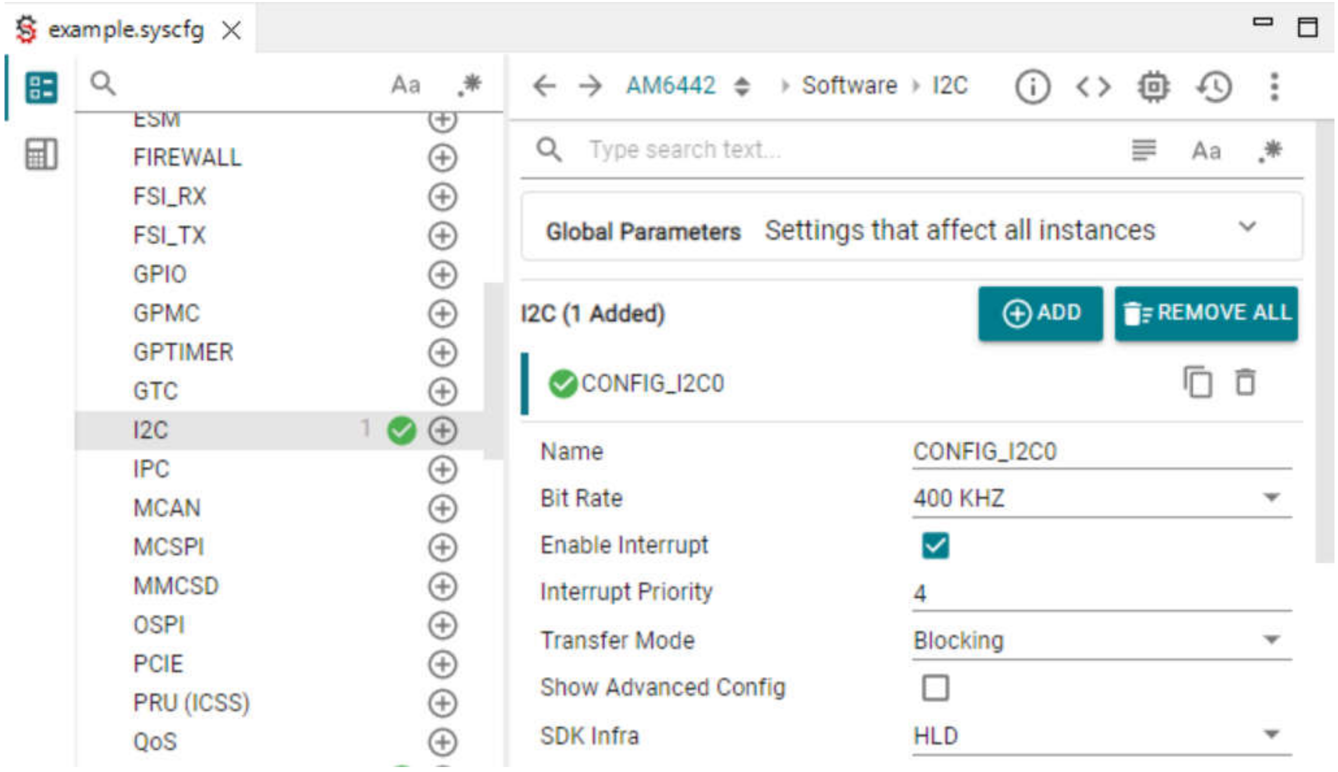


Figure 3-2. Sysconfig CCS GUI Editor

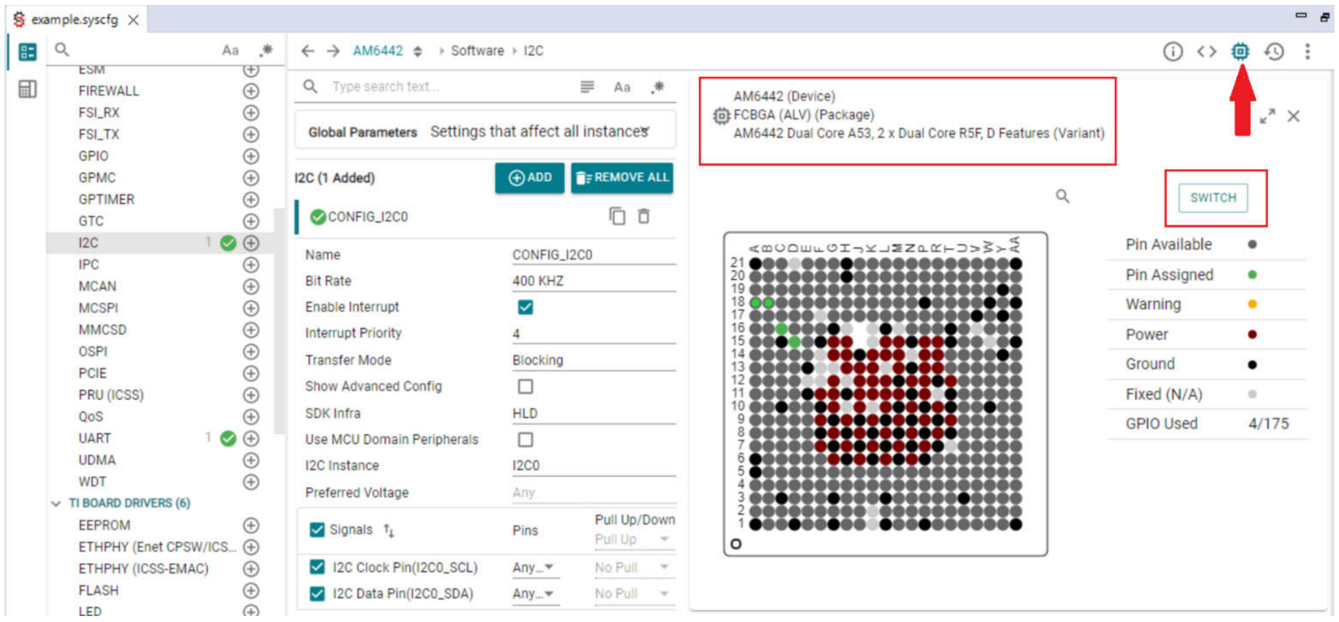
4. The SysConfig GUI must be launched inside CCS and looks similar to the one shown in Figure 3-3.



Name	CONFIG_I2C0
Bit Rate	400 KHZ
Enable Interrupt	✓
Interrupt Priority	4
Transfer Mode	Blocking
Show Advanced Config	☐
SDK Infra	HLD

Figure 3-3. Sysconfig CCS GUI View

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



Global Parameters Settings that affect all instances

I2C (1 Added)

CONFIG_I2C0

Name	CONFIG_I2C0
Bit Rate	400 KHZ
Enable Interrupt	✓
Interrupt Priority	4
Transfer Mode	Blocking
Show Advanced Config	☐
SDK Infra	HLD
Use MCU Domain Peripherals	☐
I2C Instance	I2C0
Preferred Voltage	Any

Signals	Pins	Pull Up/Down
		Pull Up
✓ I2C Clock Pin(I2C0_SCL)	Any...	No Pull
✓ I2C Data Pin(I2C0_SDA)	Any...	No Pull

AM6442 (Device)
FCBGA (ALV) (Package)
AM6442 Dual Core A53, 2 x Dual Core R5F, D Features (Variant)

Pin Available	●
Pin Assigned	🟢
Warning	🟠
Power	🔴
Ground	●
Fixed (N/A)	●
GPIO Used	4/175

Figure 3-4. Sysconfig Device View

SysConfig support is added to the Project Properties. By default, this project was configured for AM64x family of devices, and the selected device package is set to FCBGA (ALV) package. If the Project Properties for AM64x SysConfig support is not set up by default in the CCS project, the syscfg file does not launch the GUI successfully.

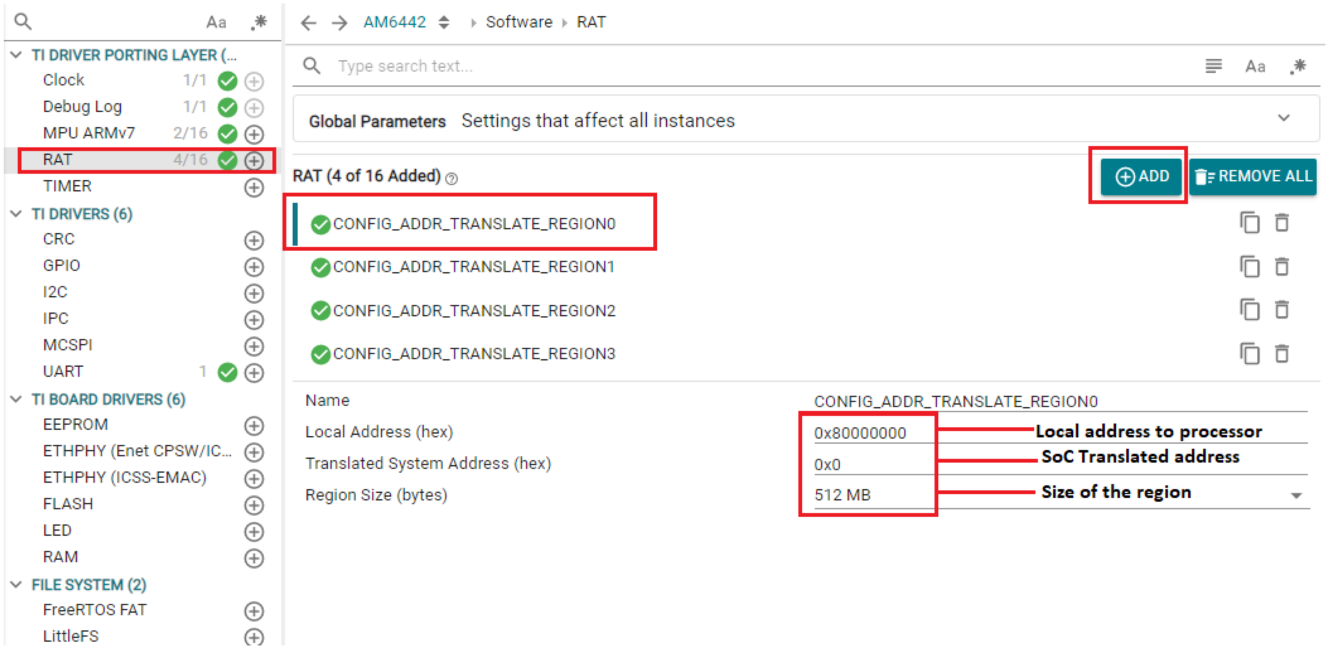
When using the standalone version of CodeGen tool (opened through CLI), the same steps are applicable for module configuration.

4 Common Application Configuration

4.1 RAT Configuration

The RAT stands for Region Based Address Translation. RAT module translates the 32-bit input address to 48-bit output address. In the AM243x and AM6x family of devices, the R5F/M4F cores in the MPU Sub System can only access 32-bit memory address. To overcome this limitation and access full memory view of SoC, RAT region is configured to access memory region higher than 32-bit memory address.

Figure 4-1 shows the RAT configuration view.



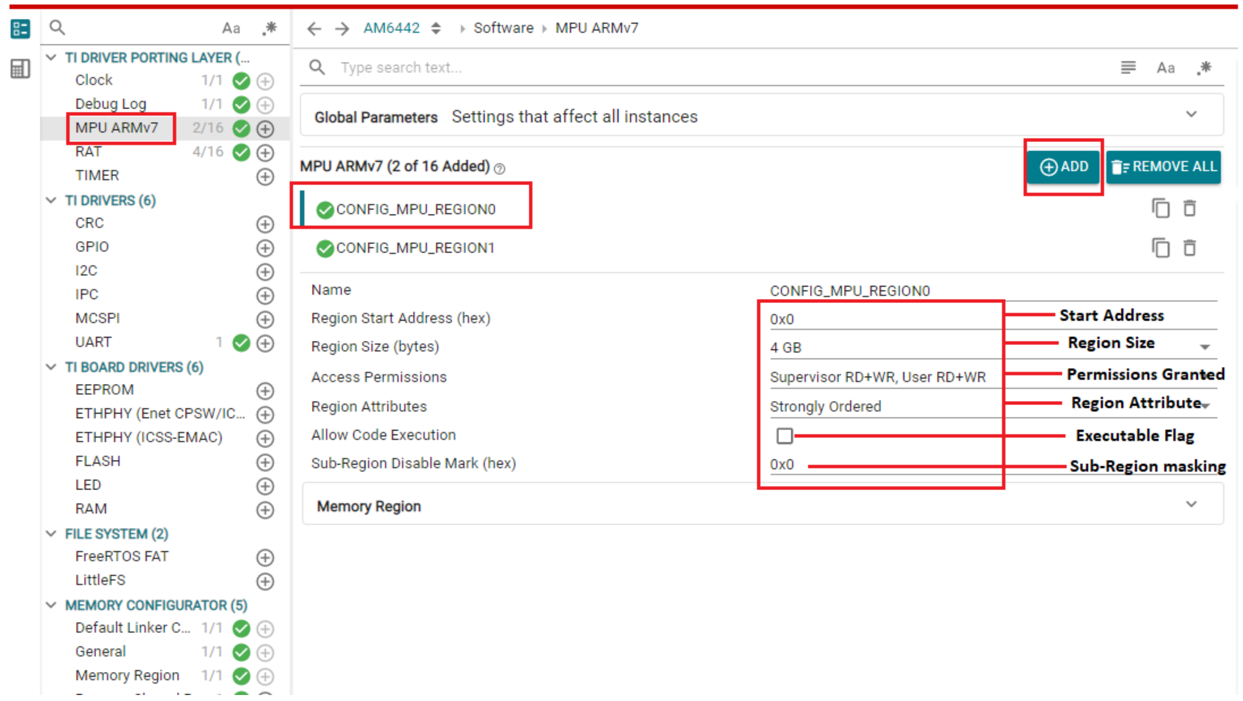
Name	CONFIG_ADDR_TRANSLATE_REGION0
Local Address (hex)	0x80000000
Translated System Address (hex)	0x0
Region Size (bytes)	512 MB

Figure 4-1. RAT Configuration

4.2 MPU Configuration

The MPU stands for the Memory Protection Unit. The MPU configuration allows user to configure the memory access permission(read/write/execute) with different privileges level. This also allows users to specify what attributes (cacheable/sharable/bufferable, and so on) a configured memory region can have.

Figure 4-2 shows the MPU Configuration view.



Name	CONFIG_MPU_REGION0
Region Start Address (hex)	0x0
Region Size (bytes)	4 GB
Access Permissions	Supervisor RD+WR, User RD+WR
Region Attributes	Strongly Ordered
Allow Code Execution	☐
Sub-Region Disable Mark (hex)	0x0

Figure 4-2. MPU Configuration

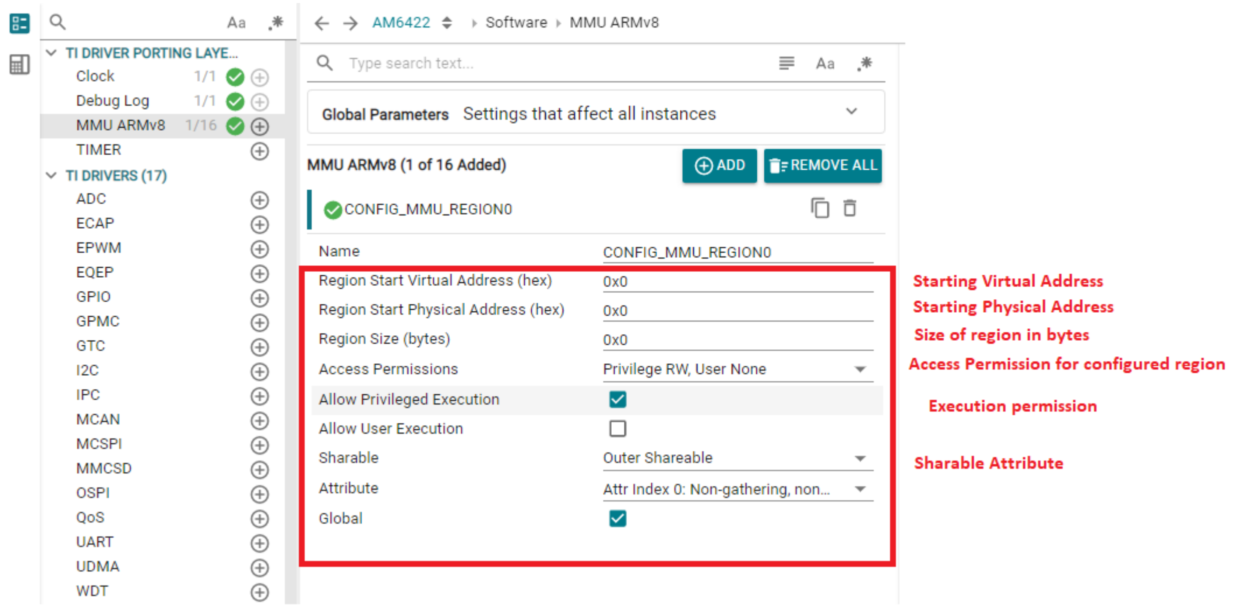
4.3 MMU Configuration

The MMU stands for the Memory Management Unit. The MMU Configuration allows virtual memory translation, memory protection, and cache management for the configured region.

The MPU is applicable for R5F or M4F cores and MMU is applicable for A53 cores and supports virtual address translation and cache management.

SysConfig provides separate views and configurations for MPU and MMU depending on the selected core context.

Figure 4-3 shows the MPU Configuration view.



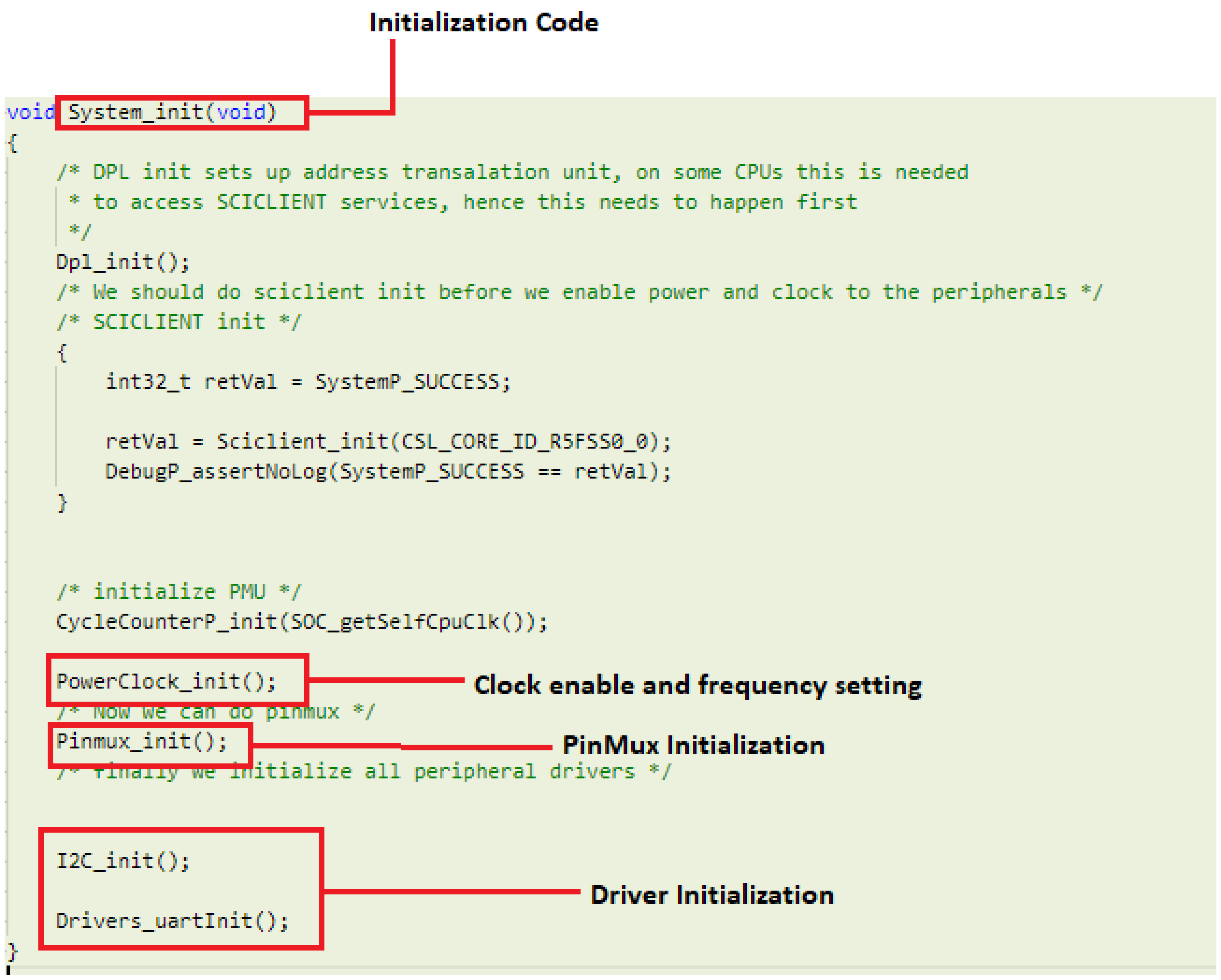
Name	CONFIG_MMU_REGION0
Region Start Virtual Address (hex)	0x0
Region Start Physical Address (hex)	0x0
Region Size (bytes)	0x0
Access Permissions	Privilege RW, User None
Allow Privileged Execution	✓
Allow User Execution	☐
Sharable	Outer Shareable
Attribute	Attr Index 0: Non-gathering, non...
Global	✓

Figure 4-3. MMU Configuration

4.4 System Initialization

The SysConfig CodeGen tool generates the code to enable clocks, do pinmux settings and driver initialization for configured peripheral.

Figure 4-4 shows the initialization code.



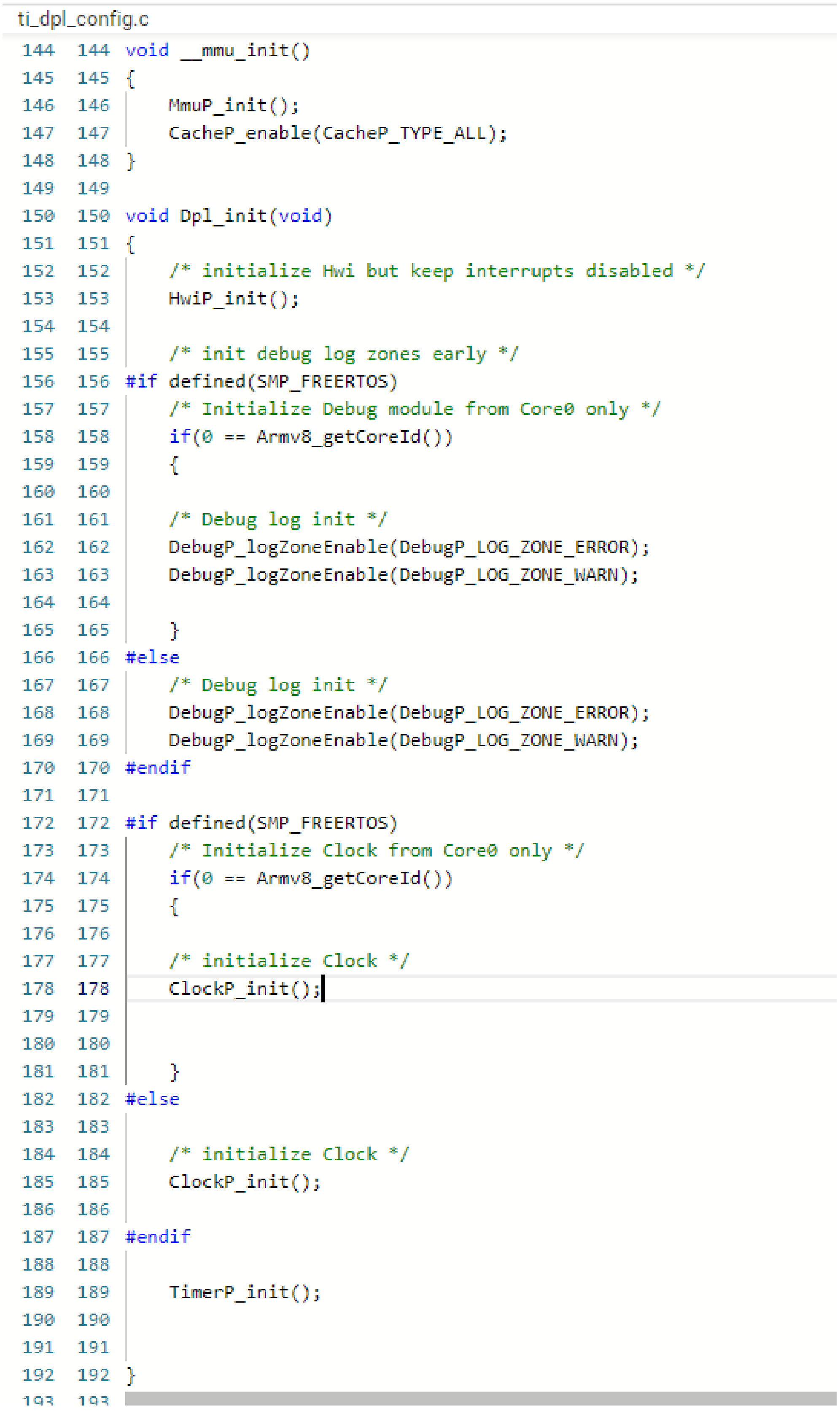
```
void System_init(void)
{
    /* DPL init sets up address transalation unit, on some CPUs this is needed
     * to access SCICLIENT services, hence this needs to happen first
     */
    Dpl_init();
    /* We should do sciclient init before we enable power and clock to the peripherals */
    /* SCICLIENT init */
    {
        int32_t retVal = SystemP_SUCCESS;

        retVal = Sciclient_init(CSL_CORE_ID_R5FSS0_0);
        DebugP_assertNoLog(SystemP_SUCCESS == retVal);
    }


    /* initialize PMU */
    CycleCounterP_init(SOC_getSelfCpuClk());

    PowerClock_init();
    /* Now we can do pinmux */
    Pinmux_init();
    /* finally we initialize all peripheral drivers */

    I2C_init();

    Drivers_uartInit();
}
```

Figure 4-4. System Initialization

The following sections discuss each of them in detail.

4.4.1 DPL Initialization

The CodeGen code generates the code for DPL initialization. DPL stands for Driver Porting Layer. The DPL initialization enables the Interrupts, initializes the system clock and does the timer initialization.



ti_dpl_config.c

```
void __mmu_init()
{
    MmuP_init();
    CacheP_enable(CacheP_TYPE_ALL);
}

void Dpl_init(void)
{
    /* initialize Hwi but keep interrupts disabled */
    HwiP_init();

    /* init debug log zones early */
#if defined(SMP_FREERTOS)
    /* Initialize Debug module from Core0 only */
    if(0 == Armv8_getCoreId())
    {

    /* Debug log init */
    DebugP_logZoneEnable(DebugP_LOG_ZONE_ERROR);
    DebugP_logZoneEnable(DebugP_LOG_ZONE_WARN);

    }
#else
    /* Debug log init */
    DebugP_logZoneEnable(DebugP_LOG_ZONE_ERROR);
    DebugP_logZoneEnable(DebugP_LOG_ZONE_WARN);
#endif

#if defined(SMP_FREERTOS)
    /* Initialize Clock from Core0 only */
    if(0 == Armv8_getCoreId())
    {

    /* initialize Clock */
    ClockP_init();


    }
#else

    /* initialize Clock */
    ClockP_init();

#endif

    TimerP_init();


}
```

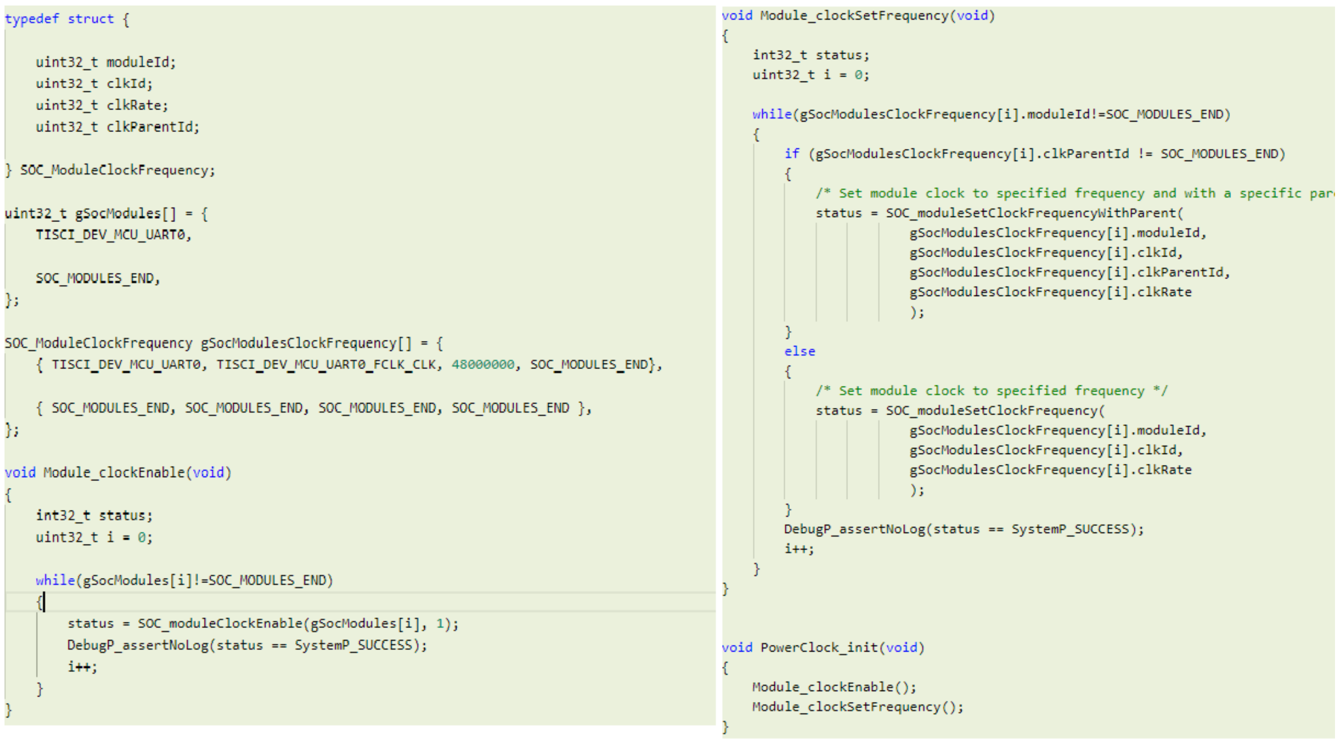
Figure 4-5. DPL Initialization

4.4.2 Clock Initialization

Before using a peripheral in the application, the peripheral must be properly initialized by enabling and configuring the respective clock settings. The clock is initialized by making call to TISCI APIs. This code required for clock configuration is auto generated by the tool when a peripheral is added.

When the user adds any module/peripherals in the CodeGen tool, the configured clock configuration automatically is populated to structure (for example, gSocModulesClockFrequency) used by TISCI APIs to set the frequency. The Module_clockEnable() API performs the power configuration for the module by enabling the LPSC gates for clocks.

Figure 4-6 shows the code generated by tool to enable and configure the clock for peripherals.



```
typedef struct {

    uint32_t moduleId;
    uint32_t clkId;
    uint32_t clkRate;
    uint32_t clkParentId;

} SOC_ModuleClockFrequency;

uint32_t gSocModules[] = {
    TISCI_DEV_MCU_UART0,

    SOC_MODULES_END,
};

SOC_ModuleClockFrequency gSocModulesClockFrequency[] = {
    { TISCI_DEV_MCU_UART0, TISCI_DEV_MCU_UART0_FCLK_CLK, 48000000, SOC_MODULES_END},

    { SOC_MODULES_END, SOC_MODULES_END, SOC_MODULES_END, SOC_MODULES_END },
};

void Module_clockEnable(void)
{
    int32_t status;
    uint32_t i = 0;

    while(gSocModules[i]!=SOC_MODULES_END)
    {
        status = SOC_moduleClockEnable(gSocModules[i], 1);
        DebugP_assertNoLog(status == SystemP_SUCCESS);
        i++;
    }
}
```

```
void Module_clockSetFrequency(void)
{
    int32_t status;
    uint32_t i = 0;

    while(gSocModulesClockFrequency[i].moduleId!=SOC_MODULES_END)
    {
        if (gSocModulesClockFrequency[i].clkParentId != SOC_MODULES_END)
        {
            /* Set module clock to specified frequency and with a specific par
            status = SOC_moduleSetClockFrequencyWithParent(
                        gSocModulesClockFrequency[i].moduleId,
                        gSocModulesClockFrequency[i].clkId,
                        gSocModulesClockFrequency[i].clkParentId,
                        gSocModulesClockFrequency[i].clkRate
                        );
        }
        else
        {
            /* Set module clock to specified frequency */
            status = SOC_moduleSetClockFrequency(
                        gSocModulesClockFrequency[i].moduleId,
                        gSocModulesClockFrequency[i].clkId,
                        gSocModulesClockFrequency[i].clkRate
                        );
        }
        DebugP_assertNoLog(status == SystemP_SUCCESS);
        i++;
    }
}


void PowerClock_init(void)
{
    Module_clockEnable();
    Module_clockSetFrequency();
}
```

Figure 4-6. Clock Configuration

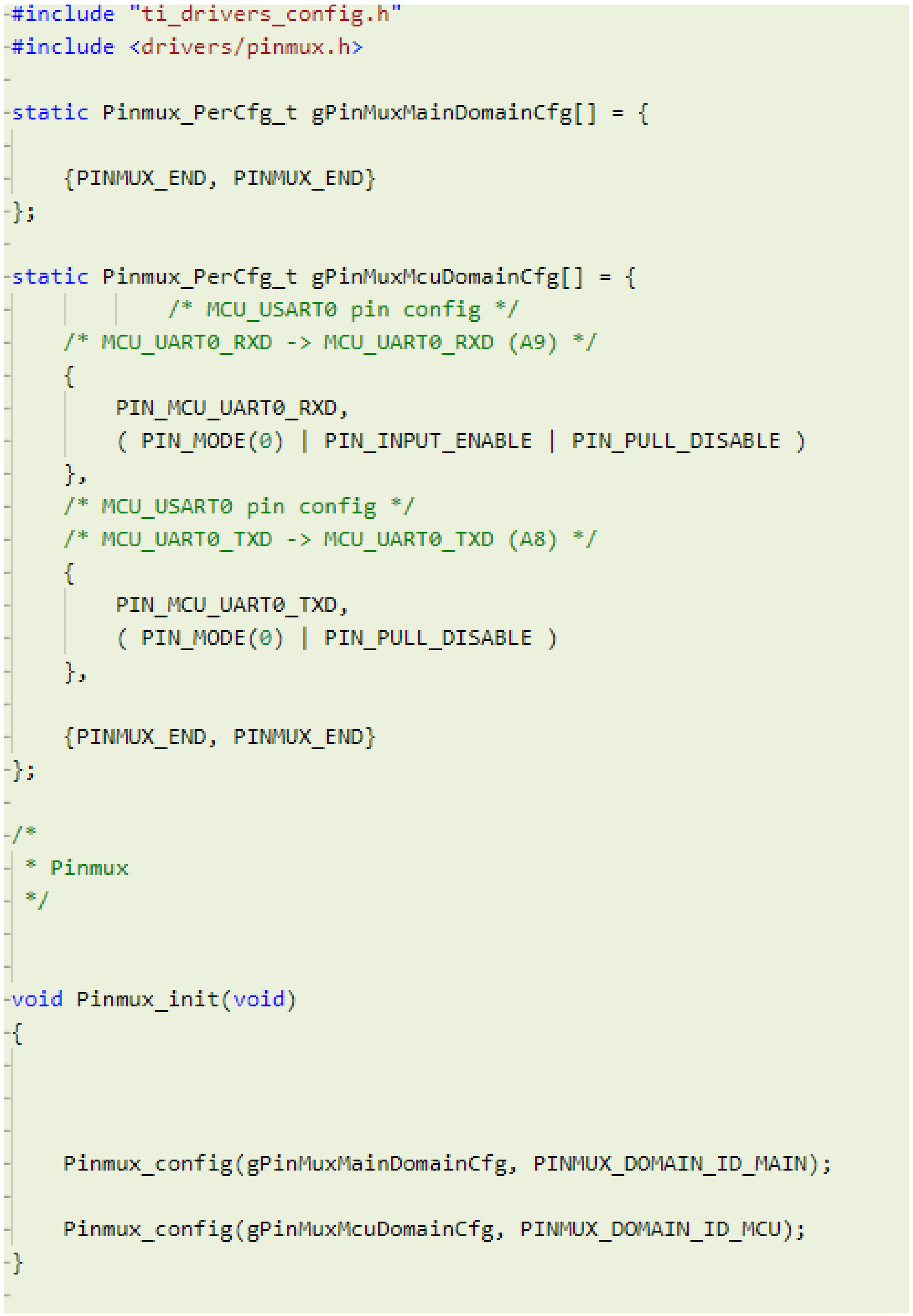
4.4.3 PinMux Configuration

The AM243x and AM6x family of devices share limited pins across multiple peripherals (UART, SPI, I2C, GPIO, etc.). Pin multiplexing (PinMux) selects which peripheral is connected to which physical ball and pin. Without correct PinMux setup, a peripheral cannot communicate with external devices. Conflicting assignments (for example, UART and I2C sharing the same pin) cause boot or runtime failures. All these conflicts can easily be avoided by using the CodeGen tool.

When a module and peripheral is added in the CodeGen tool, the pins required for the modules are exposed by the Sysconfig tool and corresponding code is generated. The user can also select the pin settings to be input enabled/disabled or can configure the pin to have pull up or down using the tool.

There are separate structures or sets of pins configured for MCU and MAIN domain peripherals.

Figure 4-7 shows the Pinmux Initialization generated code.



```
#include "ti_drivers_config.h"
#include <drivers/pinmux.h>

static Pinmux_PerCfg_t gPinMuxMainDomainCfg[] = {

    {PINMUX_END, PINMUX_END}
};

static Pinmux_PerCfg_t gPinMuxMcuDomainCfg[] = {
            /* MCU_USART0 pin config */
    /* MCU_UART0_RXD -> MCU_UART0_RXD (A9) */
    {
        PIN_MCU_UART0_RXD,
        ( PIN_MODE(0) | PIN_INPUT_ENABLE | PIN_PULL_DISABLE )
    },
    /* MCU_USART0 pin config */
    /* MCU_UART0_TXD -> MCU_UART0_TXD (A8) */
    {
        PIN_MCU_UART0_TXD,
        ( PIN_MODE(0) | PIN_PULL_DISABLE )
    },

    {PINMUX_END, PINMUX_END}
};

/*
 * Pinmux
 */


void Pinmux_init(void)
{



    Pinmux_config(gPinMuxMainDomainCfg, PINMUX_DOMAIN_ID_MAIN);

    Pinmux_config(gPinMuxMcuDomainCfg, PINMUX_DOMAIN_ID_MCU);
}
```

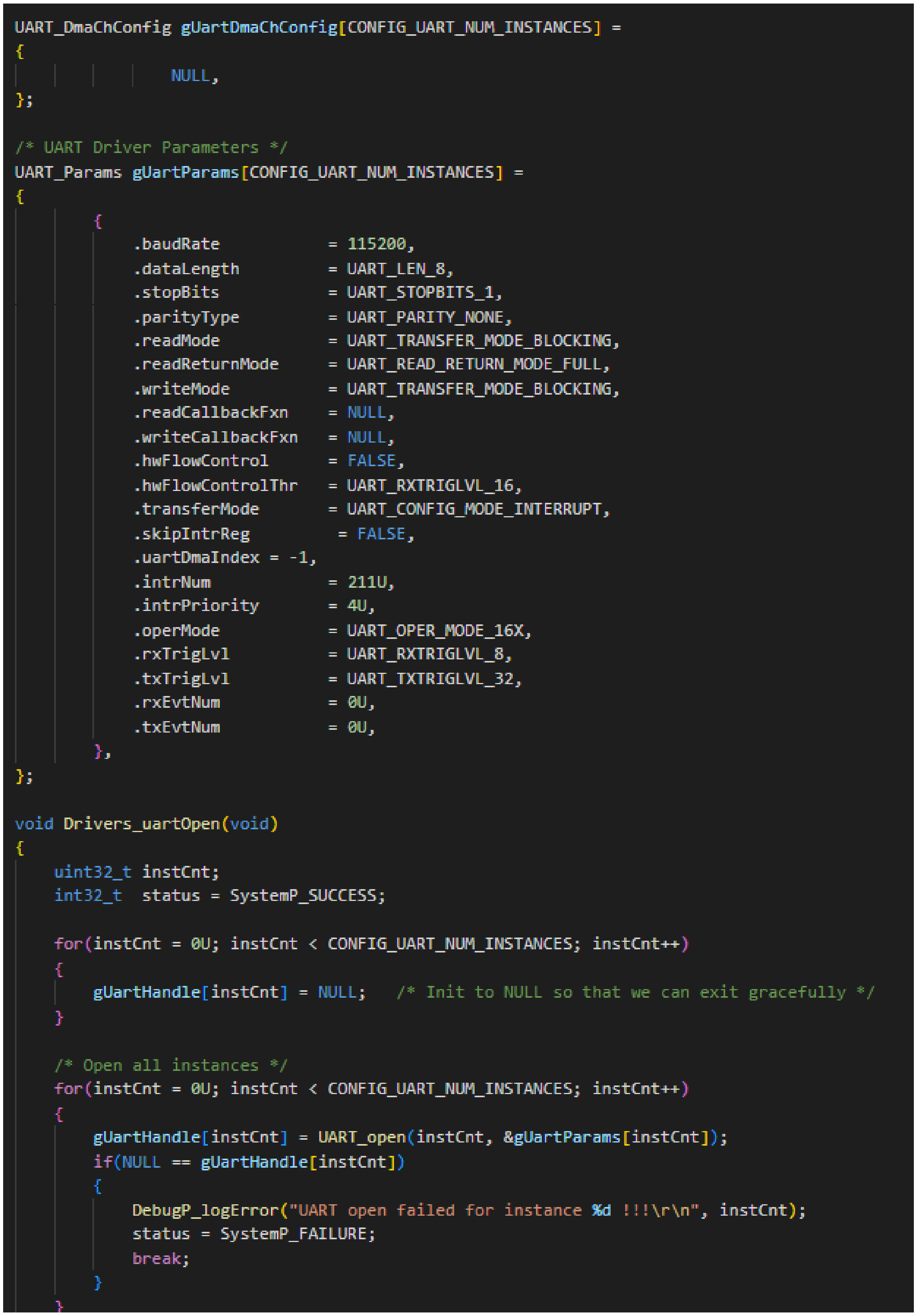
Figure 4-7. PinMux Initialization

4.4.4 Driver Initialization

To use any peripheral in an application, the driver initialization is required for proper functioning. Sysconfig CodeGen tool generates code for driver configuration for configured peripheral.

The tool uses Drivers_Init/Deinit(), Drivers_Open/Close() API as wrapper function to the actual driver initialization code. The drivers initialization code is coming from the drivers of SDK and is not auto generated by the CodeGen tool.

The CodeGen tool populate the required structure with configured values. The populated structure is used by the driver APIs of the SDK.



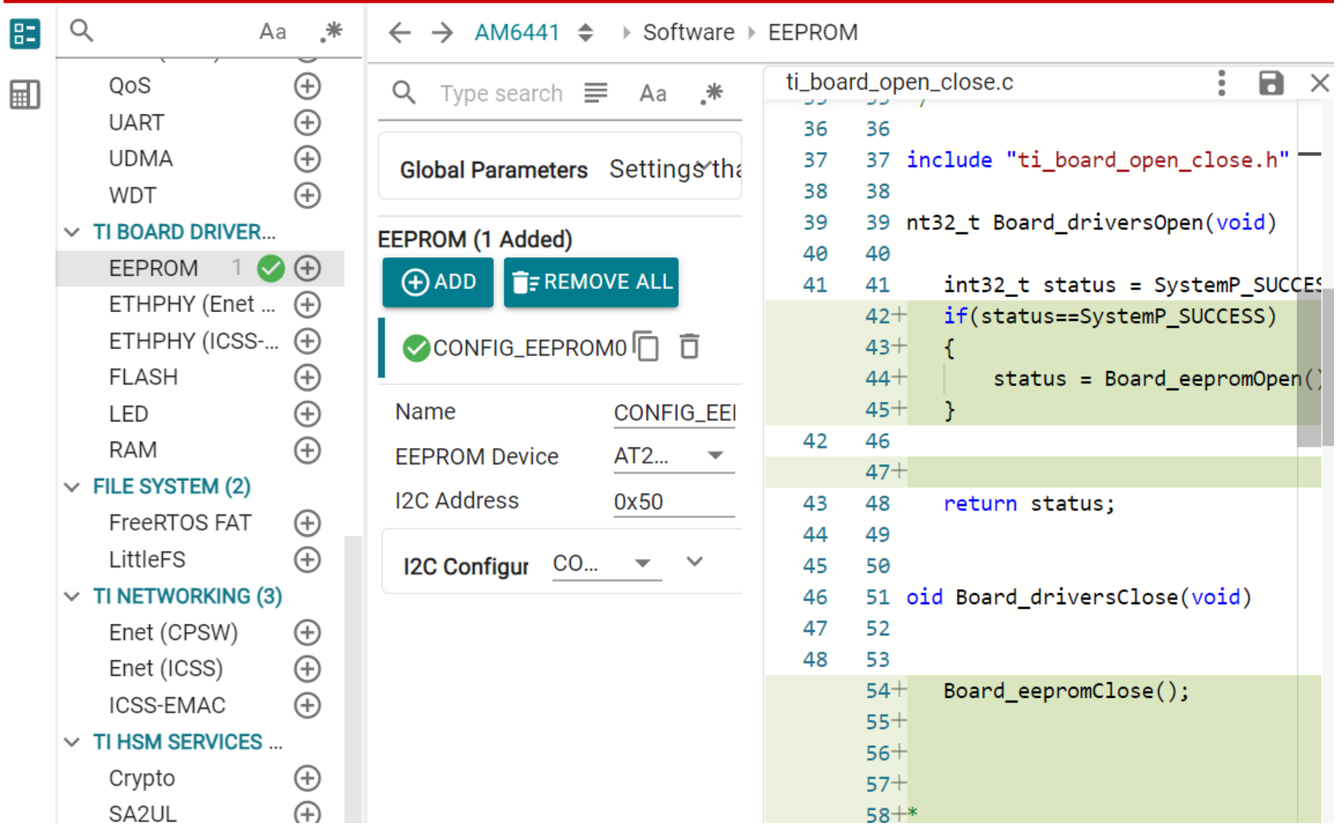
```
UART_DmaChConfig gUartDmaChConfig[CONFIG_UART_NUM_INSTANCES] =
{
                NULL,
};

/* UART Driver Parameters */
UART_Params gUartParams[CONFIG_UART_NUM_INSTANCES] =
{
        {
            .baudRate           = 115200,
            .dataLength         = UART_LEN_8,
            .stopBits           = UART_STOPBITS_1,
            .parityType         = UART_PARITY_NONE,
            .readMode           = UART_TRANSFER_MODE_BLOCKING,
            .readReturnMode     = UART_READ_RETURN_MODE_FULL,
            .writeMode          = UART_TRANSFER_MODE_BLOCKING,
            .readCallbackFxn    = NULL,
            .writeCallbackFxn   = NULL,
            .hwFlowControl      = FALSE,
            .hwFlowControlThr   = UART_RXTRIGLVL_16,
            .transferMode       = UART_CONFIG_MODE_INTERRUPT,
            .skipIntrReg         = FALSE,
            .uartDmaIndex = -1,
            .intrNum            = 211U,
            .intrPriority       = 4U,
            .operMode           = UART_OPER_MODE_16X,
            .rxTrigLvl          = UART_RXTRIGLVL_8,
            .txTrigLvl          = UART_TXTRIGLVL_32,
            .rxEvtNum           = 0U,
            .txEvtNum           = 0U,
        },
};

void Drivers_uartOpen(void)
{
    uint32_t instCnt;
    int32_t  status = SystemP_SUCCESS;

    for(instCnt = 0U; instCnt < CONFIG_UART_NUM_INSTANCES; instCnt++)
    {
        gUartHandle[instCnt] = NULL;   /* Init to NULL so that we can exit gracefully */
    }

    /* Open all instances */
    for(instCnt = 0U; instCnt < CONFIG_UART_NUM_INSTANCES; instCnt++)
    {
        gUartHandle[instCnt] = UART_open(instCnt, &gUartParams[instCnt]);
        if(NULL == gUartHandle[instCnt])
        {
            DebugP_logError("UART open failed for instance %d !!!\r\n", instCnt);
            status = SystemP_FAILURE;
            break;
        }
    }
```

Figure 4-8. Driver Configuration

4.4.5 Board Peripheral Initialization

The CodeGen tool also generates source files for Board Peripheral Initialization. This file contains the definition of API which is used to perform configured board driver initialization, and the file also provides the APIs to open and close the drivers.

The definitions of the generated APIs can be found at **ti_board_open_close.c** file.



Name	CONFIG_EEI
EEPROM Device	AT2...
I2C Address	0x50
I2C Configur	CO...

```
include "ti_board_open_close.h"

nt32_t Board_driversOpen(void)

    int32_t status = SystemP_SUCCES
    if(status==SystemP_SUCCESS)
    {
        status = Board_eepromOpen()
    }


    return status;


oid Board_driversClose(void)


    Board_eepromClose();
```

Figure 4-9. Board Peripheral Drivers

5 Output File

5.1 Files Generated by CodeGen Tool

The CodeGen tool of SysConfig generates C source and header files which are used during the application development to avoid errors and boost productivity. This generated file can be directly imported to application for driver initialization & configuration.

The files generated are listed below.

1. *ti_dpl_config.h* – contains the declaration for the DPL (Driver Porting Layer) initialization API.
2. *ti_dpl_config.c* – contains the code for DPL initialization. The DPL initialization includes initializing the Interrupt Controller, MMU & RAT configurations, Debug Logs & System tick initialization. The DPL initialization is done using the Kernel Level API call by generated code.
3. *ti_drivers_config.h* – contains the declaration for driver initialization APIs for all the configured drivers.
4. *ti_drivers_config.c* - contains code for the initialization of the configured peripheral drivers, clock initialization, PinMux settings and driver initialization. This file also contains global handles for configured peripherals.
5. *ti_drivers_open_close.h* – contains the declaration for drivers open/close API for configured peripheral along with required handler.
6. *ti_drivers_open_close.c* - contains code to Open/Close the driver for configured peripheral. This file also contains the handler with configured parameters required by the added peripheral.
7. *ti_pinmux_config.c* - contains the pinmux configuration required by the configured peripheral to achieve required functionality configured via GUI.
8. *ti_power_clock_config.c* – contains the code to enable the clock and modify the clock frequency for configured peripheral. The generated code uses the TISCI calls to configure the clock frequency.
9. *ti_board_config.h* – contains declaration for board specific driver configuration.
10. *ti_board_config.c* – contains definition for board specific driver configuration.
11. *ti_board_open_close.h* – contains declaration for board specific driver open/close APIs.
12. *ti_board_open_close.c* – contains definition for board specific driver open/close APIs.
13. *ti_enet_config.h* – contains the definition of all macros used by enet module.
14. *ti_enet_config.c* – contains definition for global structure and APIs required to provide enet functionality.
15. *ti_enet_open_close.h* - contains the declaration for enet open/close API along with the required utility API.
16. *ti_enet_open_close.c* – contains the definition for enet open/close API along with required structure definition.
17. *ti_enet_soc.c* – contains the definition for required structure and APIs for enet interrupt setup, clock frequency configuration and to set/get other necessary configurations.
18. *ti_enet_lwipif.h* – contains declaration of enet Lwip interface layer for driver callback.
19. *ti_enet_lwipif.c* - contains enet Lwip interface layer implementation for driver callback.

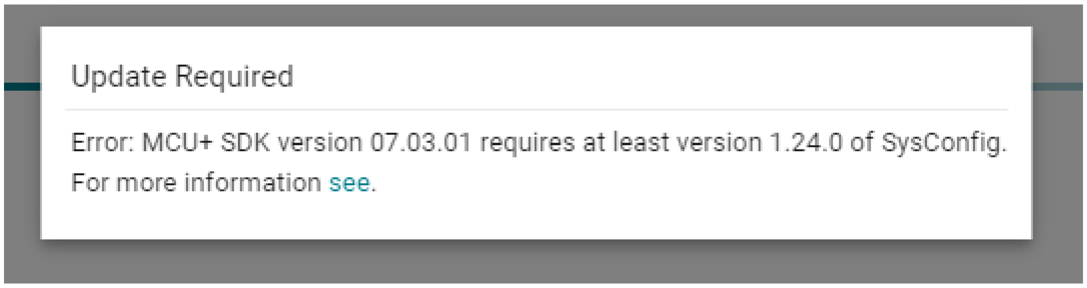
5.1.1 Debugging and Troubleshooting

When using SysConfig application if incorrect cliArgs arguments are passed, Sysconfig tool reports an error message. By looking at error messages we can identify the cause of the error. Apart from cliArgs error, there are other issues which may happen while using the tool.

The following sections discuss a few common issues which can show while using Sysconfig and see steps to fix them.

5.2 Version Mismatch

In Figure 5-1, the error message *Update Required* shows because of the version mismatch between the MCU SDK version and Sysconfig tool. The cliArgs used in the syscfg file is not correct and hence the tool reports errors while opening GUI view.


Update Required

Error: MCU+ SDK version 07.03.01 requires at least version 1.24.0 of SysConfig. For more information see.

Figure 5-1. Version Mismatch

To resolve the above issue, make sure the correct version of SysConfig is being used as mentioned in the MCU SDK documentation. Check the **product.json** file provided in the MCU SDK for version details.

In the previous example, SysConfig v1.23.0 and MCU+SDK v11.00 are used. The cliArgs used in the example.syscfg file are as follows which has incorrect MCU SDK version.

```
/** * These arguments were used when this file was generated. They will be automatically applied on
subsequent loads * via the GUI or CLI. Run CLI with '--help' for additional information on how to
override these arguments. *
                @cliArgs --device "AM64x" --part "Default" --package "ALV" --context "r5fss0-0" --
product "MCU_PLUS_SDK@07.03.01" * @v2CliArgs --device "AM6442" --package "FCBGA (ALV)" --variant
"AM6442-D" --context "r5fss0-0" --product
                "MCU_PLUS_SDK@07.03.01" * @versions {"tool":"1.21.2+3837"} */
```

After modifying the above cliArgs to have correct MCU SDK version in the example.syscfg file, the tool works as expected.

```
/** * These arguments were used when this file was generated. They will be automatically applied on
subsequent loads * via the GUI or CLI. Run CLI with '--help' for additional information on how to
override these arguments. *
                @cliArgs --device "AM64x" --part "Default" --package "ALV" --context "r5fss0-0"
--product "MCU_PLUS_SDK_AM64x@11.00.00" * @v2CliArgs --device "AM6442" --package "FCBGA (ALV)" --
variant "AM6442-D" --context "r5fss0-0" --product
                "MCU_PLUS_SDK_AM64x@11.00.00" * @versions {"tool":"1.21.2+3837"} */
```

Other issues can show *device variant not found*, *package/part number not found*, and so on. See Figure 5-2, Figure 5-3, and Figure 5-4, for details.

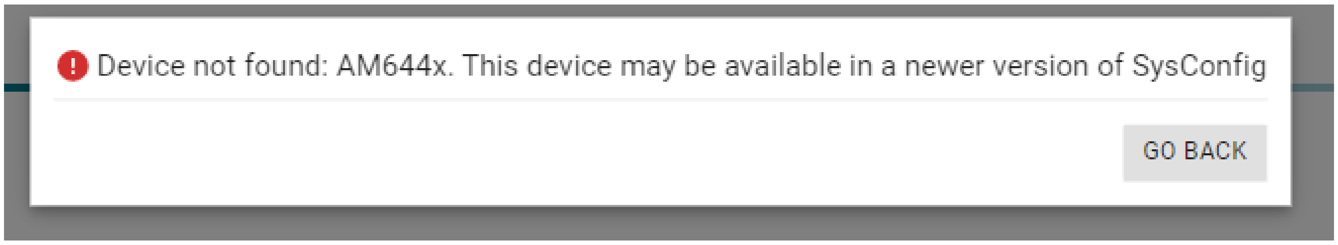

Device not found: AM644x. This device may be available in a newer version of SysConfig

Figure 5-2. Device Not Found

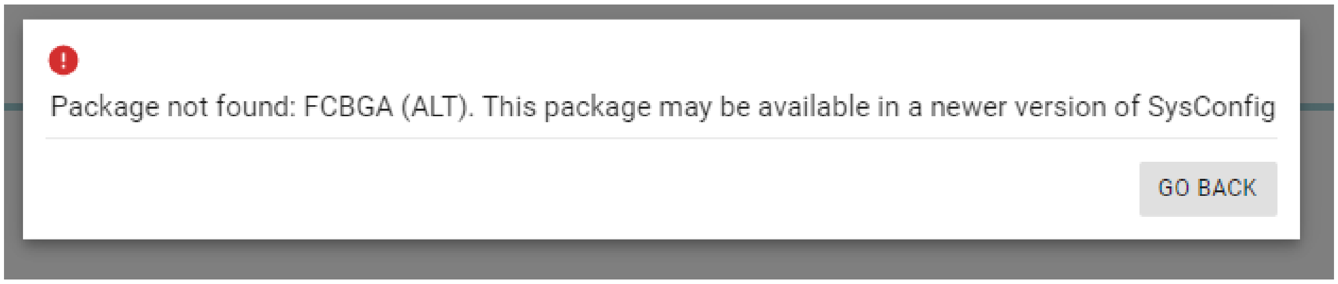
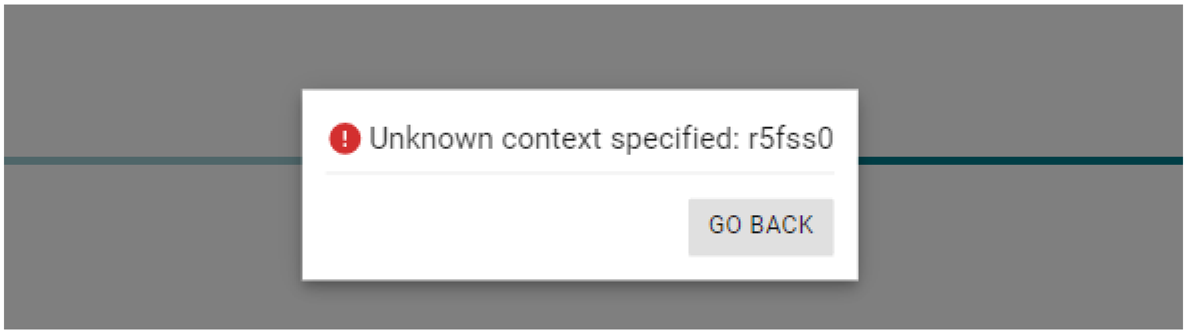

Package not found: FCBGA (ALT). This package may be available in a newer version of SysConfig

Figure 5-3. Package Not Found

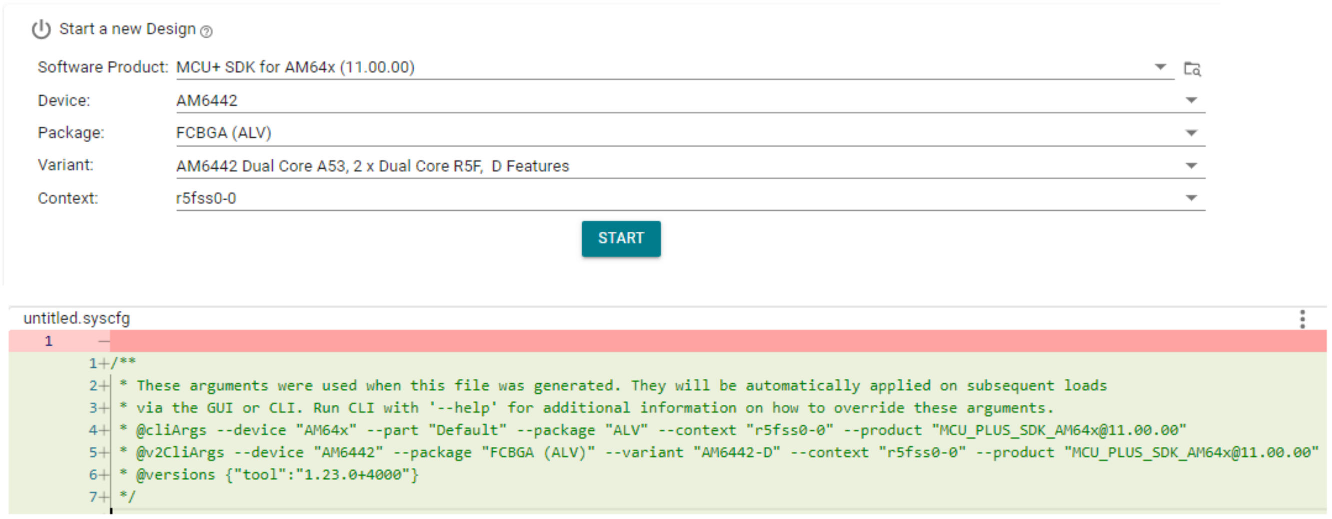


Unknown context specified: r5fss0

Figure 5-4. Unknown Context Specified

All the previous parameters must be correctly passed in the cliArgs of **example.syscfg** file. Passing incorrect argument will lead to one of the issues specified above.

If the user is still confused about what parameters to use in cliArgs, open the CodeGen tool with MCU SDK selected as software product and copy the cliArgs from the generated **untitled.syscfg** file.



Start a new Design

Software Product: MCU+ SDK for AM64x (11.00.00)
Device: AM6442
Package: FCBGA (ALV)
Variant: AM6442 Dual Core A53, 2 x Dual Core R5F, D Features
Context: r5fss0-0

untitled.syscfg

```
/**
 * These arguments were used when this file was generated. They will be automatically applied on subsequent loads
 * via the GUI or CLI. Run CLI with '--help' for additional information on how to override these arguments.
 * @cliArgs --device "AM64x" --part "Default" --package "ALV" --context "r5fss0-0" --product "MCU_PLUS_SDK_AM64x@11.00.00"
 * @v2CliArgs --device "AM6442" --package "FCBGA (ALV)" --variant "AM6442-D" --context "r5fss0-0" --product "MCU_PLUS_SDK_AM64x@11.00.00"
 * @versions {"tool":"1.23.0+4000"}
 */
```

Figure 5-5. SysConfig CodeGen cliArgs

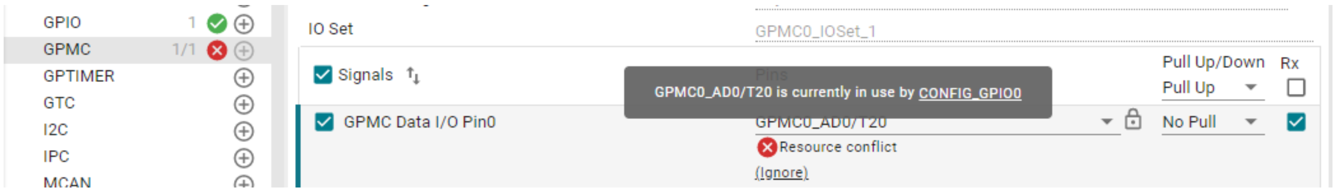
5.3 Resource Conflict

While using the CodeGen tool for development, it is simple to identify conflict and resolve them. The CodeGen tool detects different kinds of conflict which the user can have while doing manual configuration and the tools also pops up the error message for the same. The following sections discuss the types of conflict which can be identified and resolved using the tool.

5.3.1 Pin Conflict

The SysConfig tool reports an error when any of the pin is configured for more than one functionality.

For example, if user have configured GPIO pin (ball T20) and same pin (ball T20) to GPMC. The SysConfig tool reports a resource conflict issue as T20 pin is configured for multiple functionalities.



GPMC0_AD0/T20 is currently in use by CONFIG_GPIO0

Resource conflict

(Ignore)

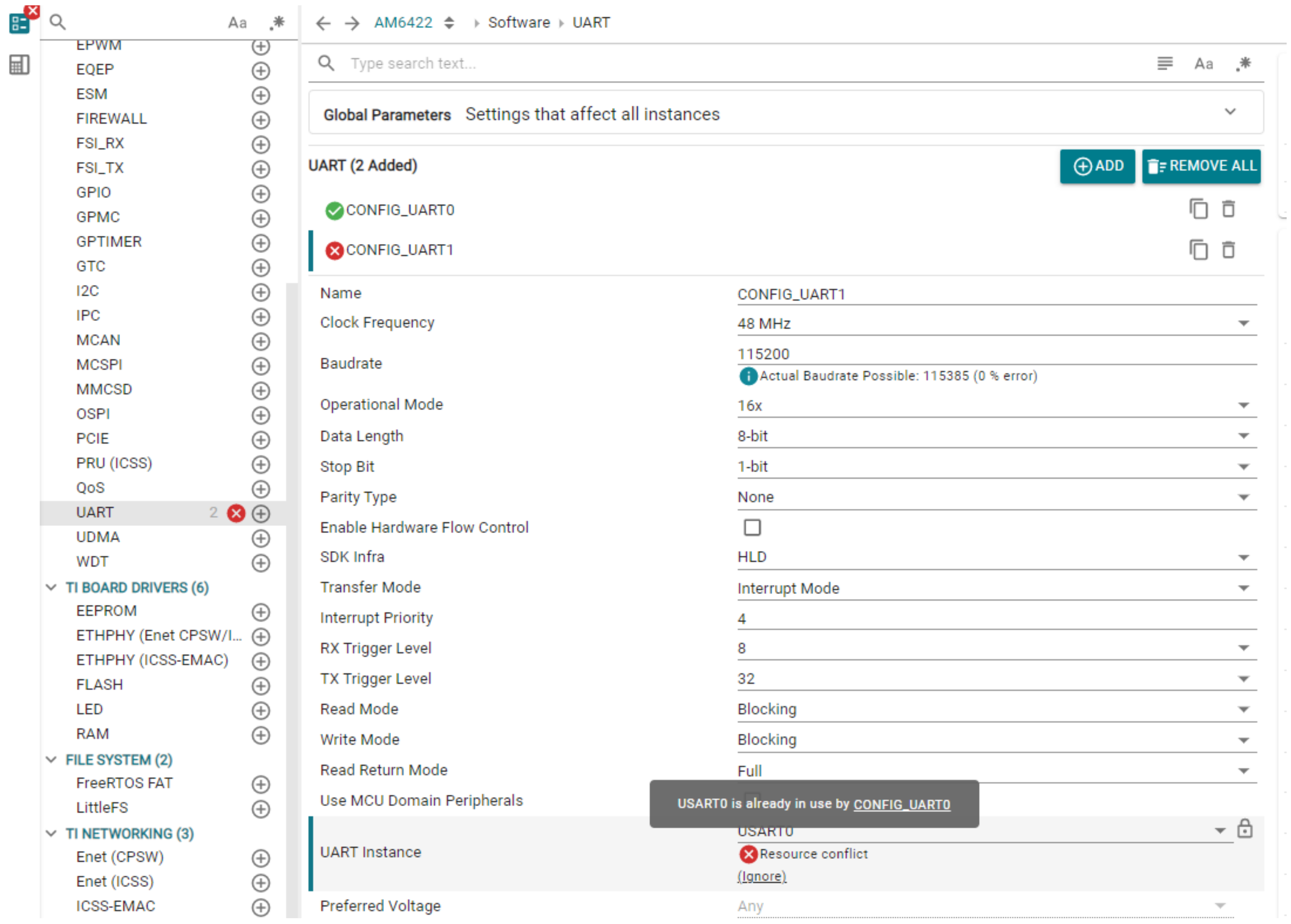
Figure 5-6. Pin Conflict

The issue can be resolved easily by removing the T20 pin either from GPIO or from GPMC peripheral.

5.3.2 Module Instance Conflict

The SysConfig tool reports an error when one of the instances under a specific module is configured more than once.

For example, if user have configured UART module and added two instances of UART. If two or more instances under UART module try to configure the same UART (say UART0) peripheral, the tool reports the instance conflict error.



Global Parameters Settings that affect all instances

UART (2 Added)

CONFIG_UART0
CONFIG_UART1

Name	CONFIG_UART1
Clock Frequency	48 MHz
Baudrate	115200 Actual Baudrate Possible: 115385 (0 % error)
Operational Mode	16x
Data Length	8-bit
Stop Bit	1-bit
Parity Type	None
Enable Hardware Flow Control	☐
SDK Infra	HLD
Transfer Mode	Interrupt Mode
Interrupt Priority	4
RX Trigger Level	8
TX Trigger Level	32
Read Mode	Blocking
Write Mode	Blocking
Read Return Mode	Full
Use MCU Domain Peripherals	
UART Instance	USART0 Resource conflict (Ignore)
Preferred Voltage	Any

USART0 is already in use by CONFIG_UART0

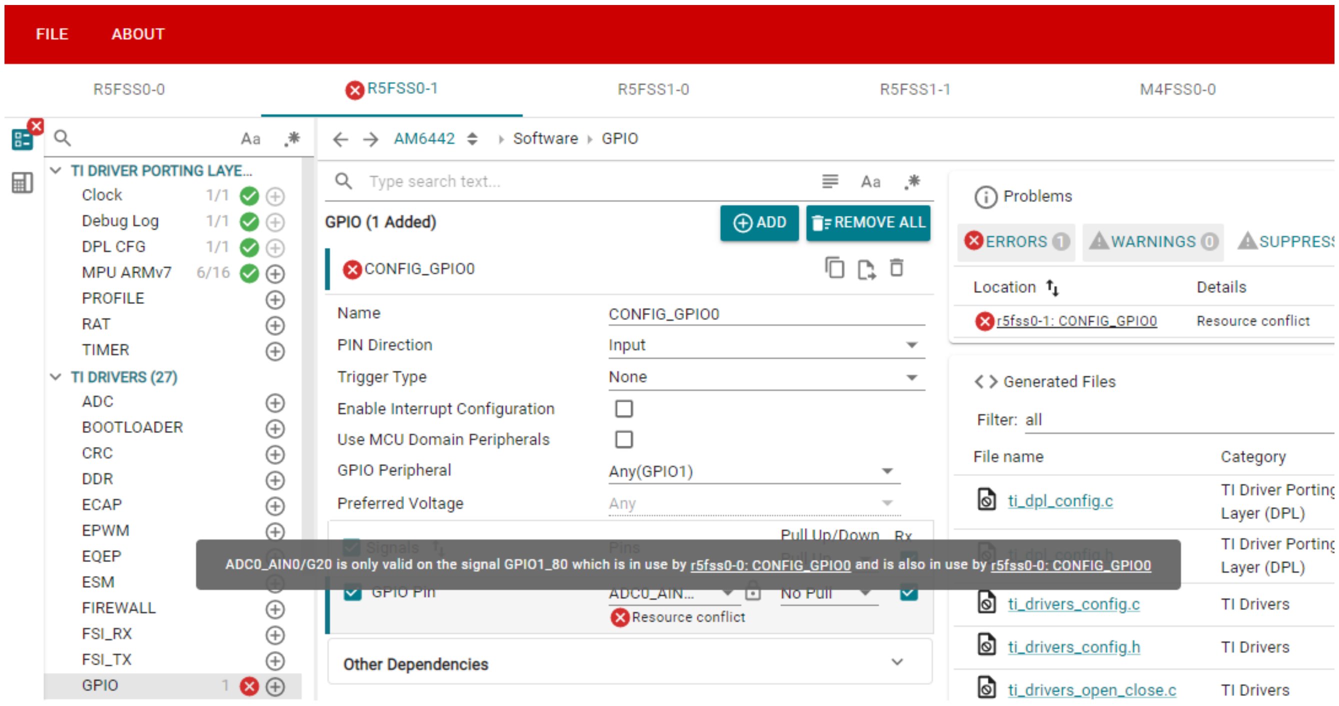
Figure 5-7. Instance Conflict

The issue can be resolved easily by configuring different UART for different instances.

5.3.3 Multicore Resource Conflict

When working with multicore projects, users can configure the same resource in two different cores. In such a scenario the tool automatically detects the cause of conflict between the cores and the pop-up error message.

For example, if a GPIO pin is configured for R5F0-0 core and the same pin is again configured for the R5F0-1 core in multicore project, resource conflict error is given by the tool.



TI DRIVER PORTING LAYE...
Clock 1/1
Debug Log 1/1
DPL CFG 1/1
MPU ARMv7 6/16
PROFILE
RAT
TIMER
TI DRIVERS (27)
ADC
BOOTLOADER
CRC
DDR
ECAP
EPWM
EQEP
ESM
FIREWALL
FSI_RX
FSI_TX
GPIO 1

GPIO (1 Added)
CONFIG_GPIO0

Name	CONFIG_GPIO0
PIN Direction	Input
Trigger Type	None
Enable Interrupt Configuration	☐
Use MCU Domain Peripherals	☐
GPIO Peripheral	Any(GPIO1)
Preferred Voltage	Any

GPIO Pin ADC0_AIN... No Pull
Resource conflict

ADC0_AIN0/G20 is only valid on the signal GPIO1_80 which is in use by r5fss0-0: CONFIG_GPIO0 and is also in use by r5fss0-0: CONFIG_GPIO0

Location	Details
r5fss0-1: CONFIG_GPIO0	Resource conflict

Filter: all

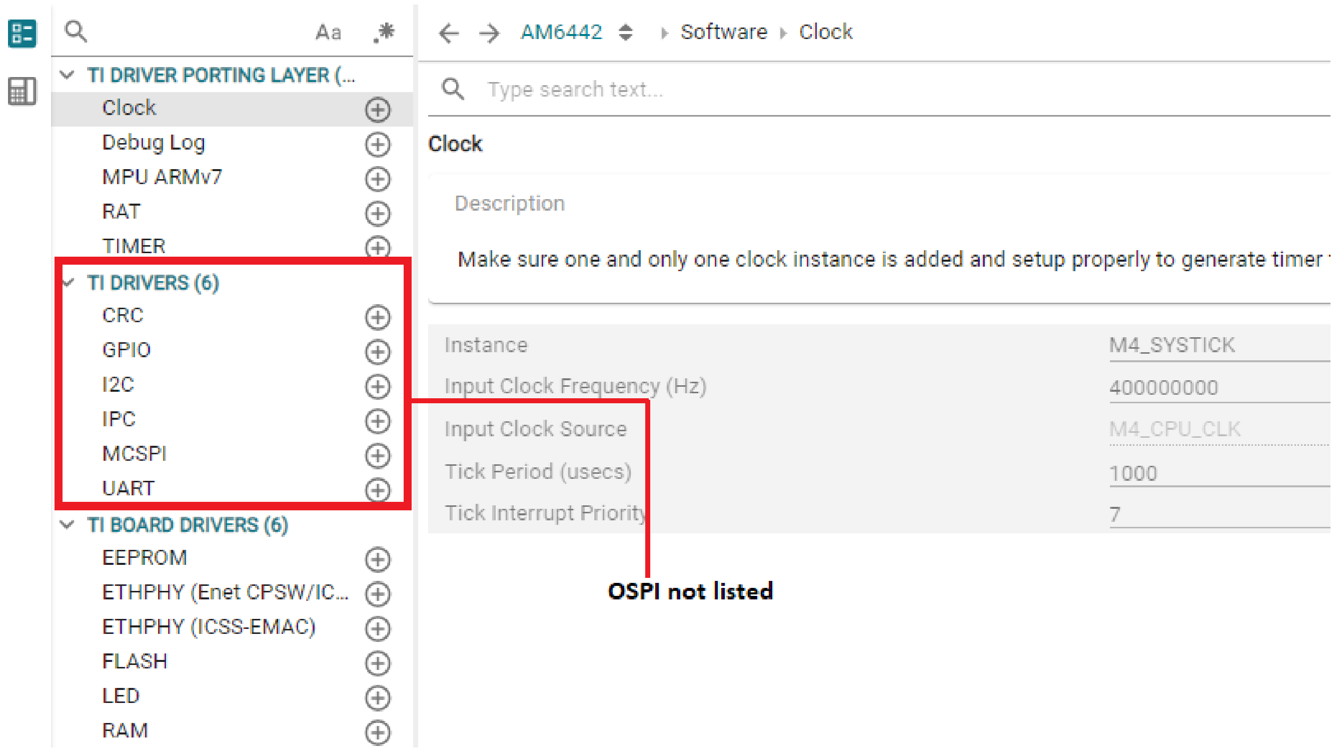
File name	Category
ti_dpl_config.c	TI Driver Porting Layer (DPL)
ti_dpl_config.h	TI Driver Porting Layer (DPL)
ti_drivers_config.c	TI Drivers
ti_drivers_config.h	TI Drivers
ti_drivers_open_close.c	TI Drivers

Figure 5-8. Multicore Resource Conflict

The issue can be resolved simply by configuring different GPIO pins for different cores.

5.4 Unsupported Drivers

In the following example, users must use the OSPI driver in the application, but the OSPI module is not present in the *TI Drivers* list of the tool. See Figure 5-9 for details.



TI DRIVER PORTING LAYER (...
Clock
Debug Log
MPU ARMv7
RAT
TIMER
TI DRIVERS (6)
CRC
GPIO
I2C
IPC
MCSPI
UART
TI BOARD DRIVERS (6)
EEPROM
ETHPHY (Enet CPSW/IC...
ETHPHY (ICSS-EMAC)
FLASH
LED
RAM

Clock

Description

Make sure one and only one clock instance is added and setup properly to generate timer

Instance	M4_SYSTICK
Input Clock Frequency (Hz)	400000000
Input Clock Source	M4_CPU_CLK
Tick Period (usecs)	1000
Tick Interrupt Priority	7

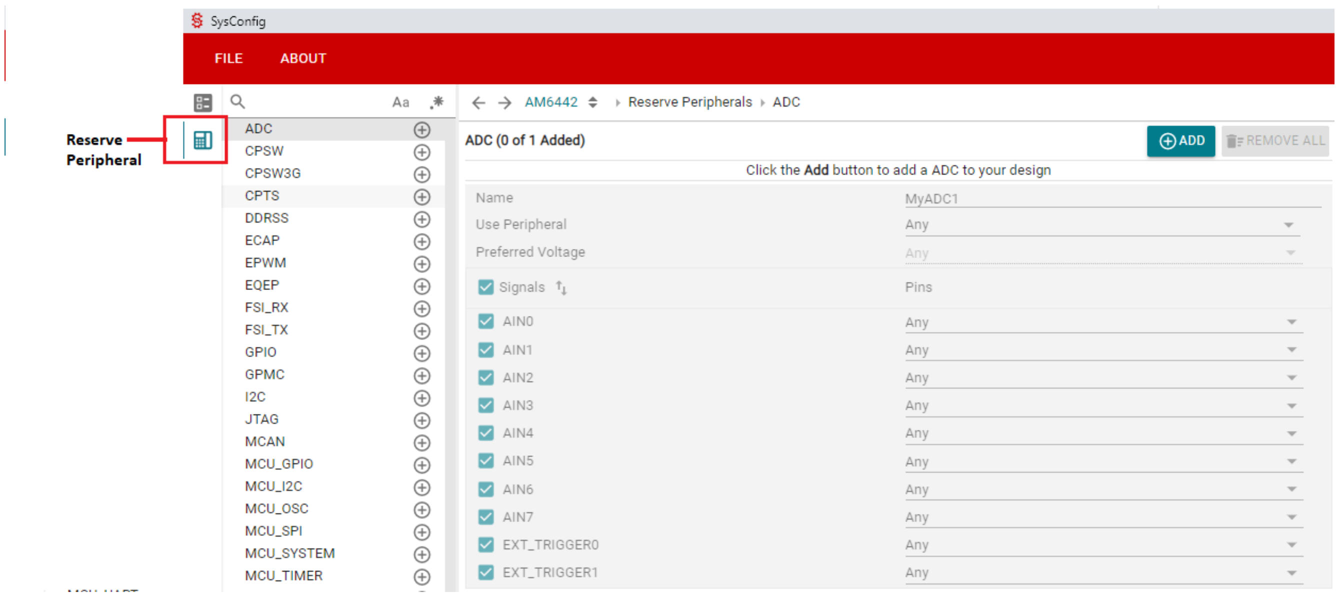
Figure 5-9. TI Drivers

The drivers which are not listed under the *TI Drivers* section of the tool are not supported by the MCU SDK. The drivers list in the tool can change with the different core combinations.

See the MCU SDK release notes for details on the supported driver list.

5.5 Use of Reserve Peripheral

The Reserved Peripherals tab is used to reserve any hardware resource that custom code can use, and the tab tells the SysConfig tool not to use that peripheral. The SysConfig tool does not generate any code for the peripherals configured under Reserved Peripheral. This tab must not be used for any peripherals that must be configured by the tool.



ADC (0 of 1 Added)

Click the **Add** button to add a ADC to your design

Name	MyADC1
Use Peripheral	Any
Preferred Voltage	Any
✓ Signals	Pins
✓ AIN0	Any
✓ AIN1	Any
✓ AIN2	Any
✓ AIN3	Any
✓ AIN4	Any
✓ AIN5	Any
✓ AIN6	Any
✓ AIN7	Any
✓ EXT_TRIGGER0	Any
✓ EXT_TRIGGER1	Any

Figure 5-10. SysConfig Reserve Peripheral Tab

6 Disclaimers and Intended Use

SysConfig core tool follows TI baseline quality development process. This means that there are no automotive or functional safety claims that can be made on code that is generated using SysConfig. The expectation is that it is the responsibility of the customer to perform standard qualification on generated code according to requirements of a particular standard.

7 Summary

SysConfig significantly accelerates software bring-up by auto-generating initialization and configuration code for TI SoCs.

The GUI and CLI interfaces minimize manual effort, verify configuration consistency, and improve productivity across multi-core projects.

By integrating SysConfig into the MCU+SDK workflow, developers can rapidly prototype and scale embedded systems with the reduced risk of configuration errors.

8 References

- TI Cloud Tools
  - SysConfig
  - Resource Explorer