Application Note

利用集成 SysConfig 的 MCU+SDK 加速开发进程

Tushar Thakur, Anil Swargam, Soumya Tripathy

摘要

本应用手册介绍了对于 AM243x、AM275x 和 AM6x 器件而言，SysConfig 工具在与 MCU+SDK 集成的过程中发挥的作用。SysConfig 通过自动生成引脚多路复用、时钟配置、电源域配置、驱动器配置、板级外设配置、区域地址转换 (RAT)、内存管理单元 (MMU) 及内存保护单元 (MPU) 配置的源文件，简化了系统启动流程。

本应用手册还提供分步指导、示例用例以及故障排除技巧，以帮助开发人员将 SysConfig 高效地用于 TI SOC。

内容

商标

1 简介

SysConfig 是一款与 MCU+SDK 集成的交互式配置工具，可自动为 TI SoC 完成器件初始化和驱动程序设置。该工具可检测配置冲突，生成初始化文件，并简化与自定义软件项目或 MCU+SDK 项目集成的流程。开发人员可以通过直观的 GUI 或命令行界面，使用 SysConfig 配置时钟、PinMux、MPU/MMU/RAT 区域和驱动程序实例。

该工具支持以下功能：

- **系统初始化：**SysConfig (CodeGen) 工具可为 AM243x、AM275x 和 AM6x 器件生成初始化代码，涵盖外设设置、时钟配置、中断处理、PinMux 配置以及 MPU、MMU 和 RAT 设置。有关详细信息，请参阅*系统初始化。*
- **PinMux 可视化：**该工具提供器件和引脚的图形化视图，显示所有可能的 PinMux 选项，并突出显示用户为每个引脚选择的模式。有关详细信息，请参阅 *CCS*(5) 中的示例 SysConfig。
- **错误检测：**SysConfig 会对配置进行验证，并在设置不正确时报告错误。该工具会自动检测引脚分配之间的冲突。有关详细信息，请参阅*引脚冲突。*
- **依赖关系识别：**该工具会识别器件内模块间的依赖关系，并确保所需外设的配置保持一致性。
- **资源冲突检测：**当某个模块依赖于另一个外设时，SysConfig 会检查冲突情况。如果依赖的外设已在使用中，该工具会标记资源冲突错误。有关详细信息，请参阅*资源冲突*。

注意：支持的器件系列包括：

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

1.1 SysConfig CodeGen 工具

SysConfig CodeGen 工具会生成源文件和头文件，这些文件与 MCU SDK 示例配合使用以实现上述功能。CodeGen 工具在内部使用 MCU+SDK 提供的 Sciclient (TISCI) API，通过与 DMSC 固件的通信来管理时钟、复位和电源域。

请参阅 SysConfig CodeGen 工具中的参数——这些参数是打开 SysConfig CodeGen 工具所必需的。

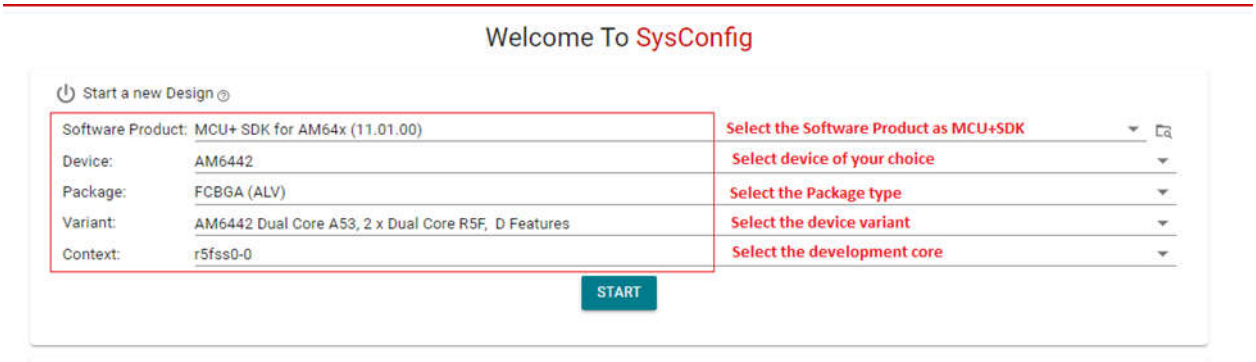

图 1-1. SysConfig CodeGen 工具

Codegen 工具生成的文件中显示的 SysConfig CodeGen 工具的视图。

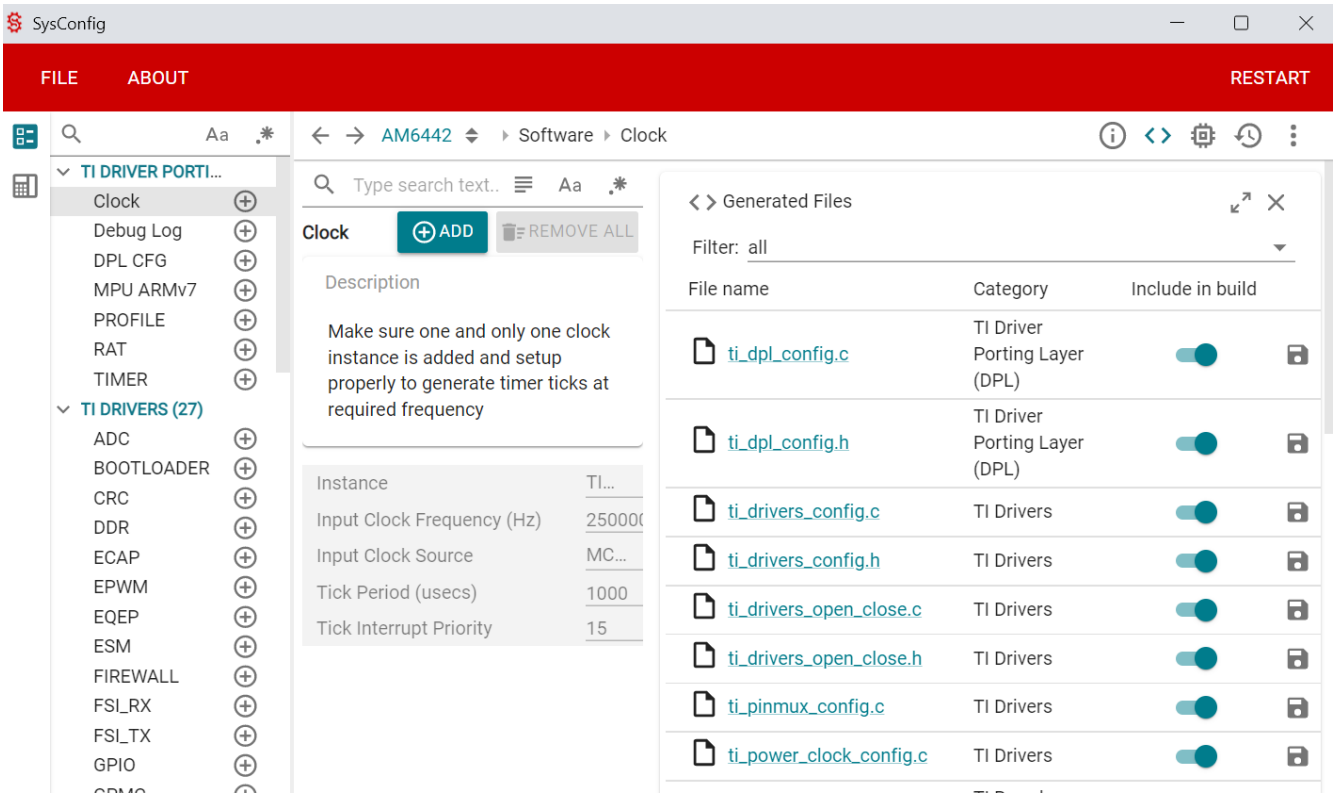

图 1-2. CodeGen 工具生成的文件

2 入门指南

SysConfig 工具可用于在线和离线开发。

请点击以下链接来访问该工具：

1. 要下载 SysConfig 工具的离线版本，请访问 https://www.ti.com/tool/download/SYSCONFIG。
2. 该工具的在线版本可通过 https://dev.ti.com/sysconfig/#/start 访问

备注

在线版本是 SysConfig 的最新版本。使用在线 SysConfig 工具时，请查看 MCU SDK 的版本说明，以确保版本兼容。

2.1 如何启动 SysConfig（GUI 和命令行）

SysConfig 工具可以通过 GUI（图形用户界面）或通过 CLI（命令行界面）启动。

- 若要使用 GUI 打开该工具，请导航至 SysConfig 目录并双击 **sysconfig_gui.bat** 文件。
- 要通过 CLI 打开 SysConfig CodeGen 工具，请执行以下步骤。
  - 导航到示例目录，直至 makefile 可见。
  - 执行以下命令。如果是 Linux，使用 *make*；如果是 Windows，则使用 *gmake*。

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

- 要通过 CLI 运行 SysConfig 工具，请使用以下命令。以下命令将输出 SysConfig CodeGen 工具生成的所有文件。

```
> 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 与 CCS 和 Makefile 构建系统的集成

SysConfig 工具的运行始于 **product.json** 文件，该文件包含了 CodeGen 工具所需的全部信息。

此处提到的信息适用于独立版 SysConfig 工具以及与 CCS 集成的工具。

查看 CCS 项目中的 SysConfig 项目属性。

1. 右键点击项目名称并选择 **Properties**（属性）。
2. 在 **Build**（构建）选项下，选择 **SysConfig** 以查看所有 SysConfig 选项。

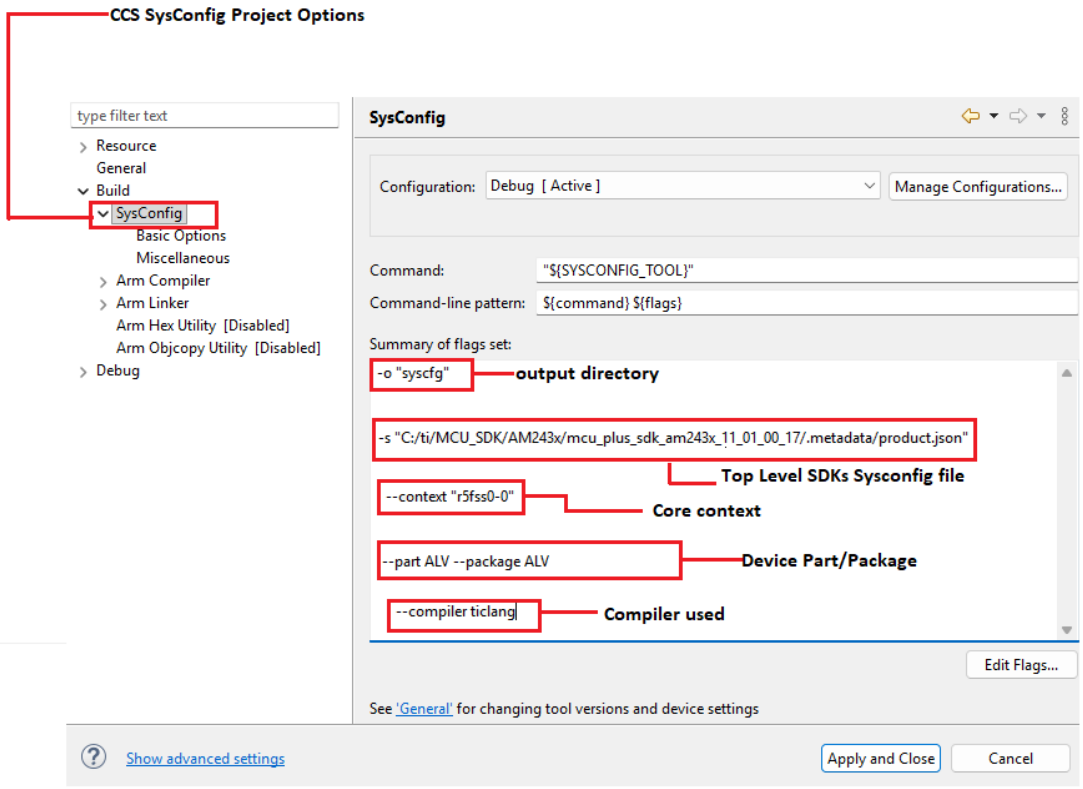

图 2-1. CCS SysConfig Project Properties

3. 选择 **Basic Options**（基本选项）可修改/查看器件系列和顶层 SysConfig **product.json** 文件。

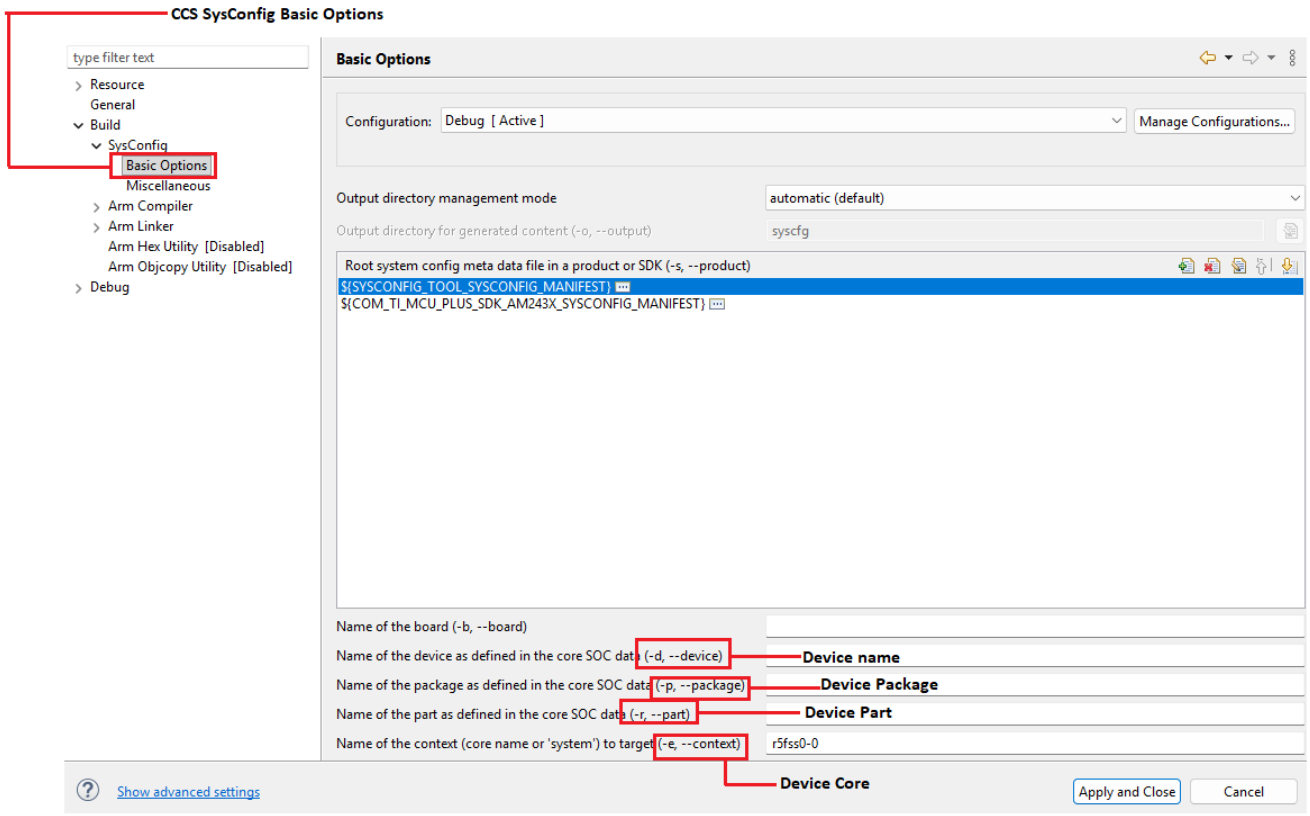

图 2-2. SysConfig Basic Options

4. 选择 **Miscellaneous**（其他）可修改/查看器件封装/零件。

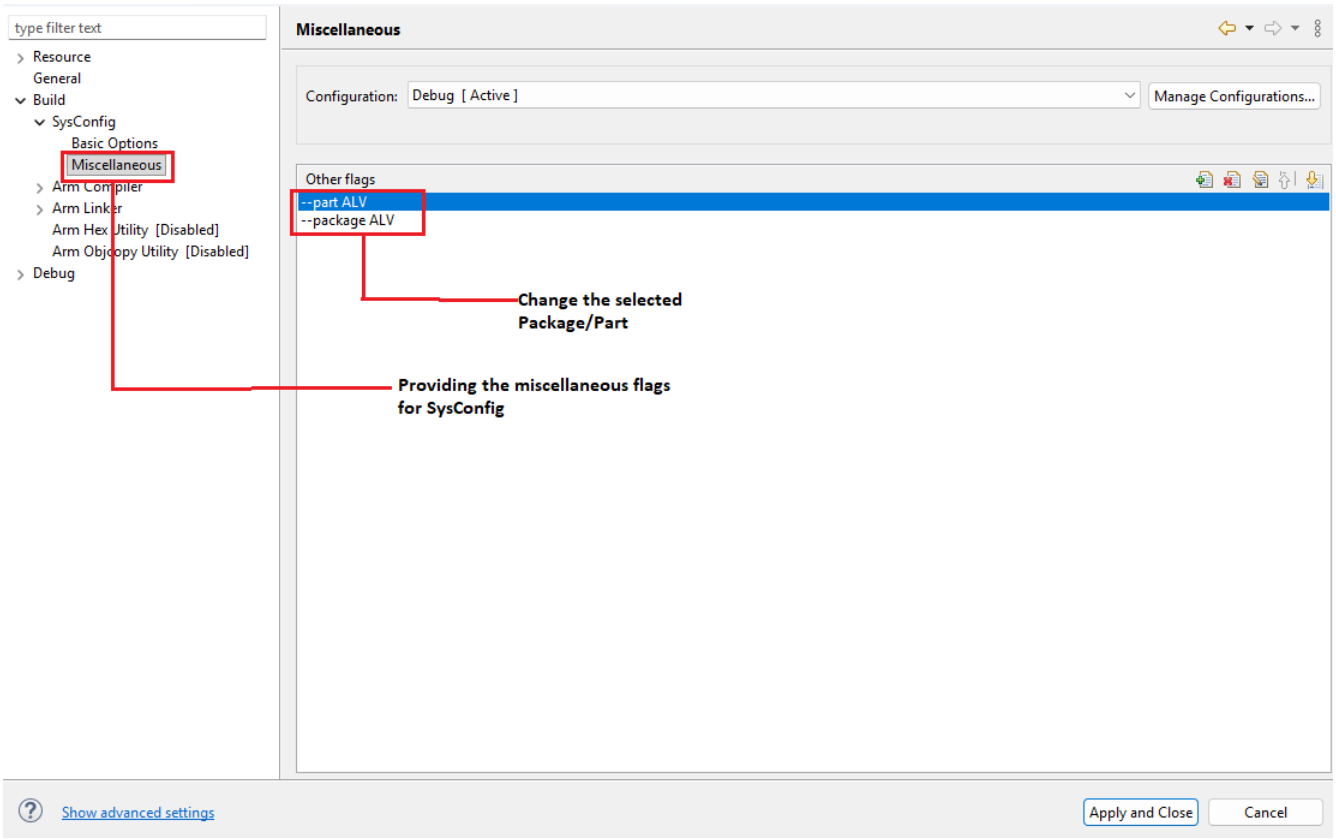

图 2-3. SysConfig Miscellaneous Options

2.3 MCU SDK 中 SysConfig 文件的位置

2.3.1 使用现有的 SysConfig 文件

MCU+SDK 中提供的每个示例都包含一个 **example.syscfg** 文件，该文件详细说明了通过 SysConfig CodeGen 工具配置和初始化的外设的详细信息。此工具将此文件作为输入，并为所配置的外设生成所需的输出文件。

example.syscfg 文件位于 **${MCU+SDK}/examples/${name}/${device}/${core}/example.sysconfig**

2.3.2 创建新的 SysConfig 文件

通过按照 SysConfig CodeGen 工具中的说明打开 CodeGen 工具，可以从头开始生成 **example.syscfg** 文件。

该工具会自动生成 **untitled.syscfg** 文件，该文件可在保存后用于 MCU SDK。

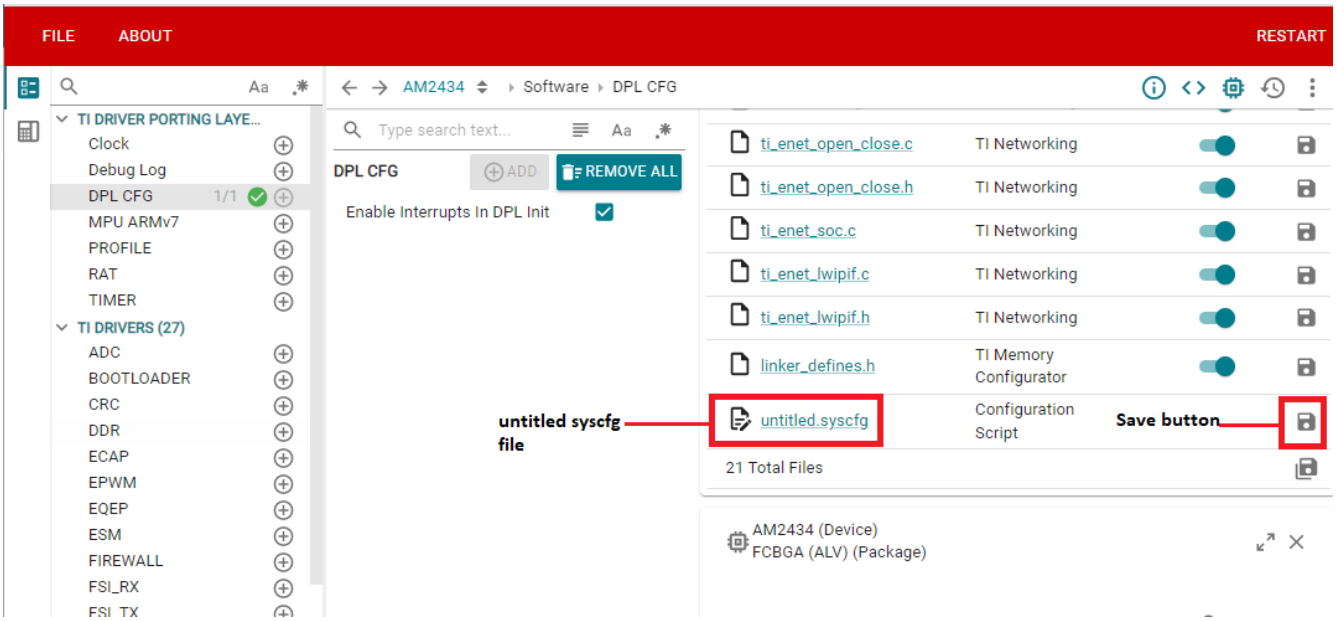

图 2-4. 生成的 syscfg 文件

3 CCS 中的示例 SysConfig

3.1 I2C 读取示例

若要开始使用 SysConfig CodeGen 工具，请导入 MCU SDK 中提供的支持 SysConfig 的现有示例。

1. 启动 CCS 并导入示例：**i2c_read_r5fss0-0_nortos**
   a. 选择 **Project（项目）→ Import CCS Project（导入 CCS 项目）**
   b. 访问 **${MCU+SDK}\examples\drivers\i2c\i2c_read\am64x-evm\r5fss0-0_nortos**
   c. 选择项目并导入该项目。
2. 在 CCS 项目中，用户可以看到 syscfg 文件与其他应用程序文件。

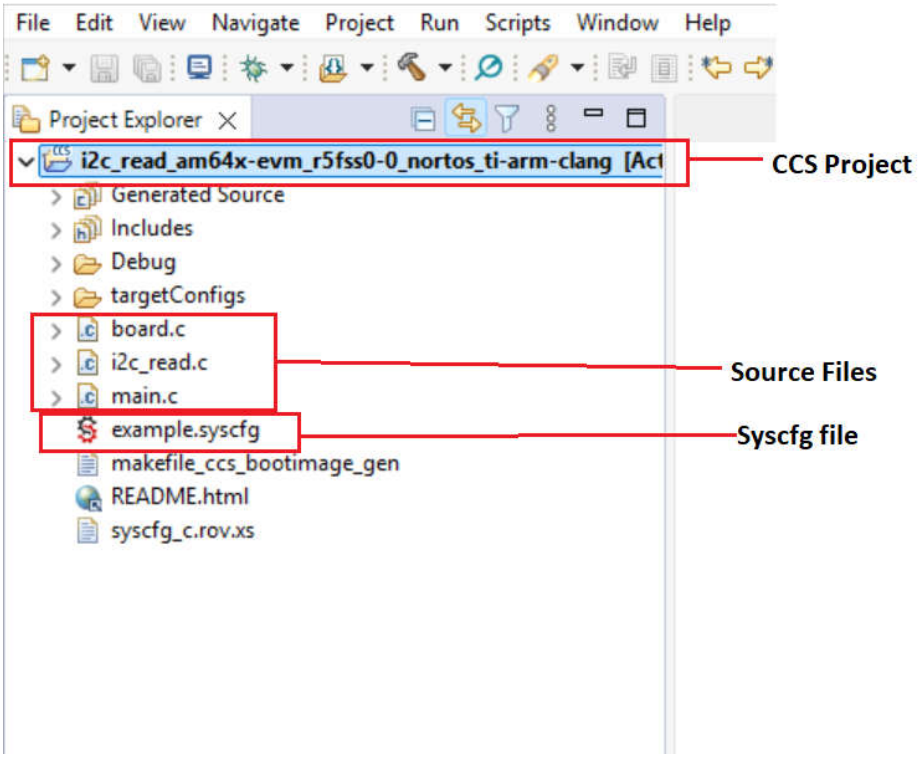

图 3-1. 示例项目

3. 双击 **example.syscfg** 文件，SysConfig GUI 将启动。

备注

右键点击 syscfg 文件，然后选择“Open With”→“SysConfig Editor”。

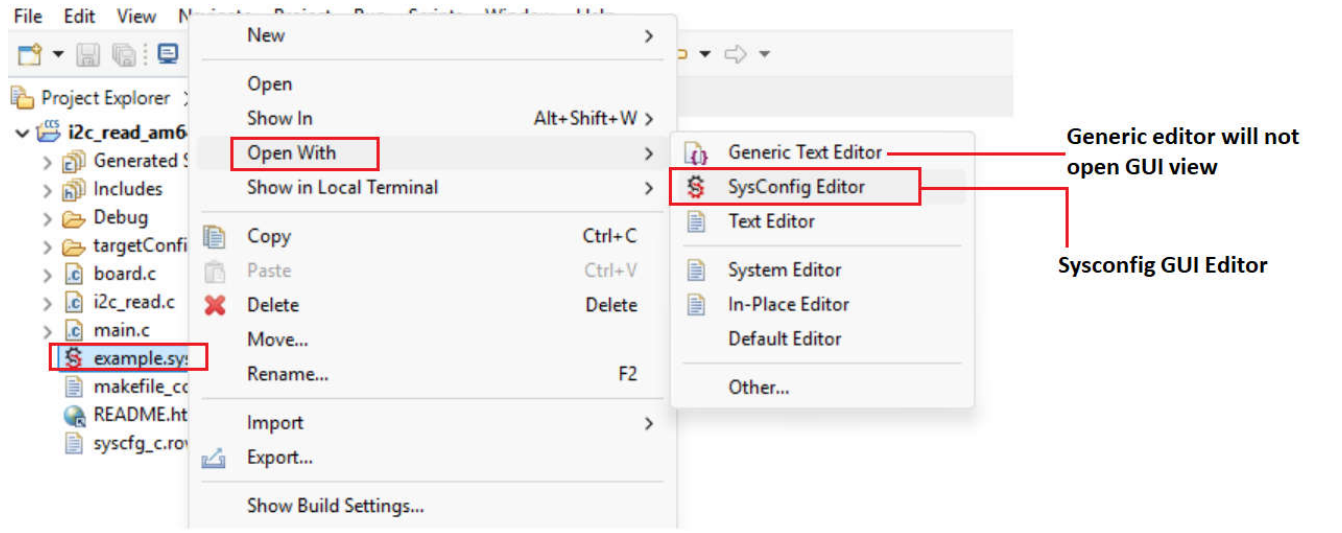

图 3-2. SysConfig CCS GUI 编辑器

4. SysConfig GUI 必须在 CCS 内启动，其外观与图 3-3 所示相似。

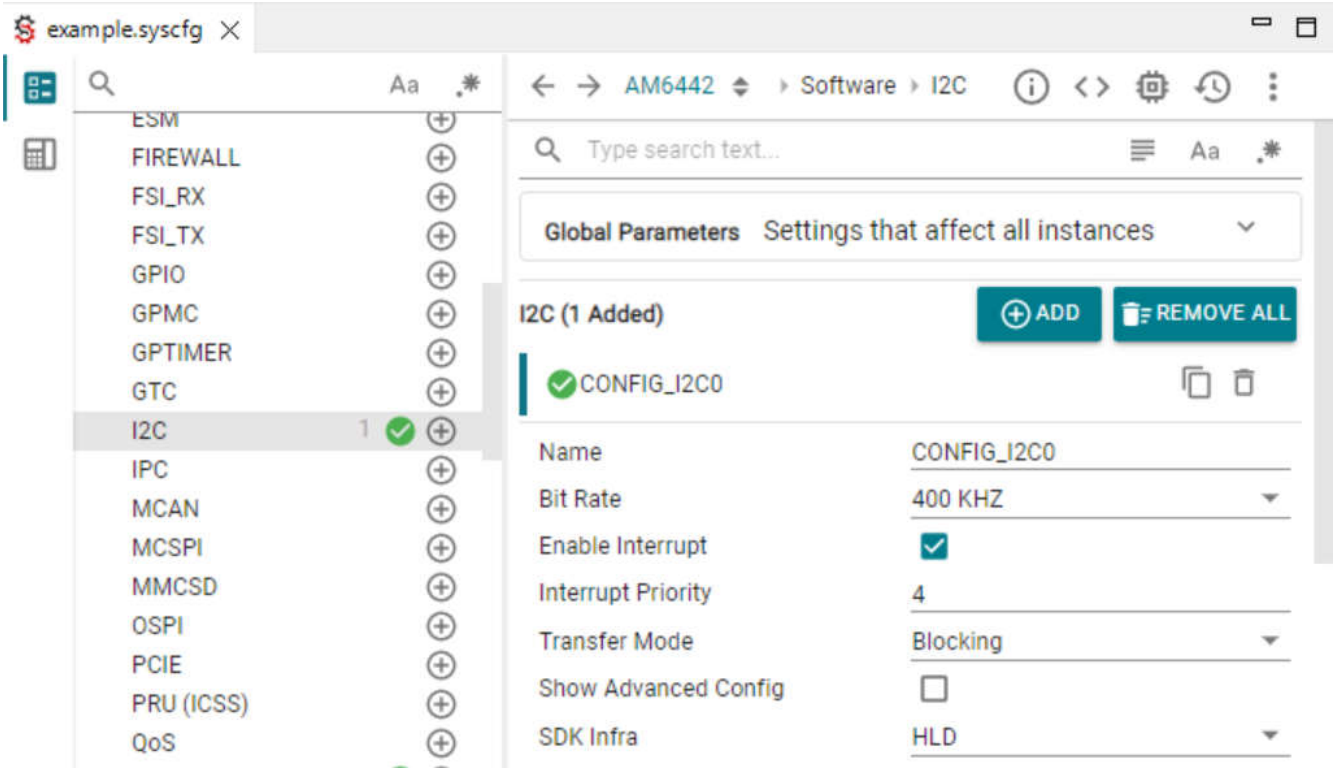

图 3-3. SysConfig CCS GUI 视图

5. 点击 SysConfig GUI 右上角的 **Device View**（器件视图）按钮，查看用于该项目的器件和封装。

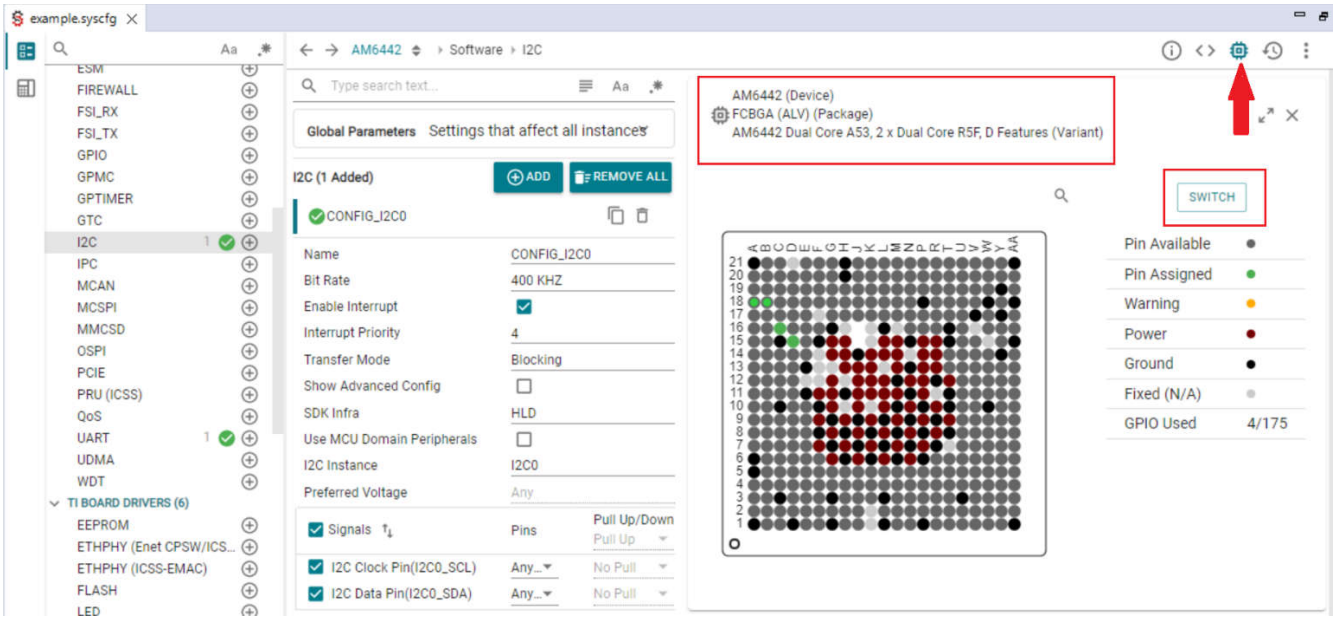

图 3-4. SysConfig 器件视图

“Project Properties”（项目属性）中已添加了 SysConfig 支持。默认情况下，此项目已针对 AM64x 系列器件完成配置，所选器件封装设置为 FCBGA (ALV) 封装。如果默认情况下，CCS 项目中未针对 AM64x SysConfig 支持设置项目属性，则 syscfg 文件将不会成功启动 GUI。

使用独立版的 CodeGen 工具（通过 CLI 打开）时，模块配置的步骤与此相同。

4 通用应用程序配置

4.1 RAT 配置

RAT 代表基于区域的地址转换。RAT 模块可将 32 位输入地址转换为 48 位输出地址。在 AM243x 和 AM6x 系列器件中，MPU 子系统中的 R5F/M4F 内核只能访问 32 位内存地址。为了突破此限制并访问 SoC 的完整内存视图，需将 RAT 区域配置为可访问高于 32 位内存地址的内存区域。

图 4-1 显示了“RAT 配置”的视图。

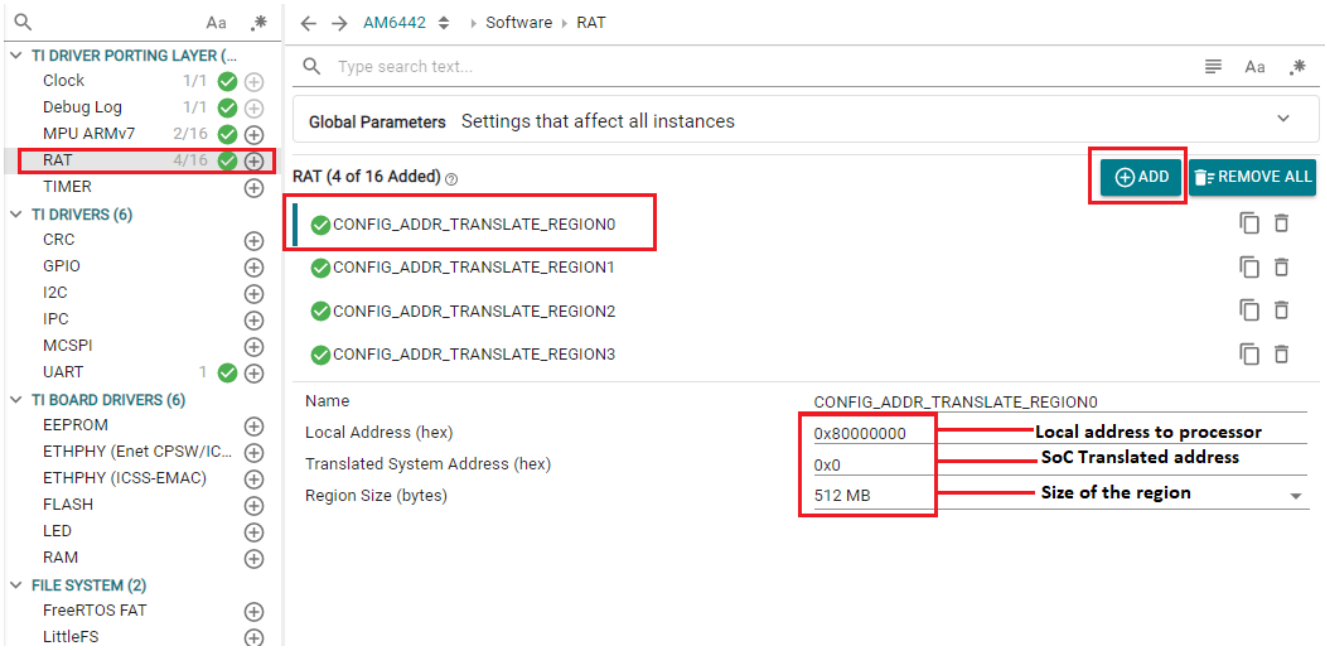

图 4-1. RAT 配置

4.2 MPU 配置

MPU 代表内存保护单元。通过 MPU 配置，用户可设置不同权限级别下的内存访问权限（读取/写入/执行）。此外，用户还可指定配置的内存区域具有的属性（可缓存/可共享/可缓冲等）。

图 4-2 显示了“MPU 配置”的视图。

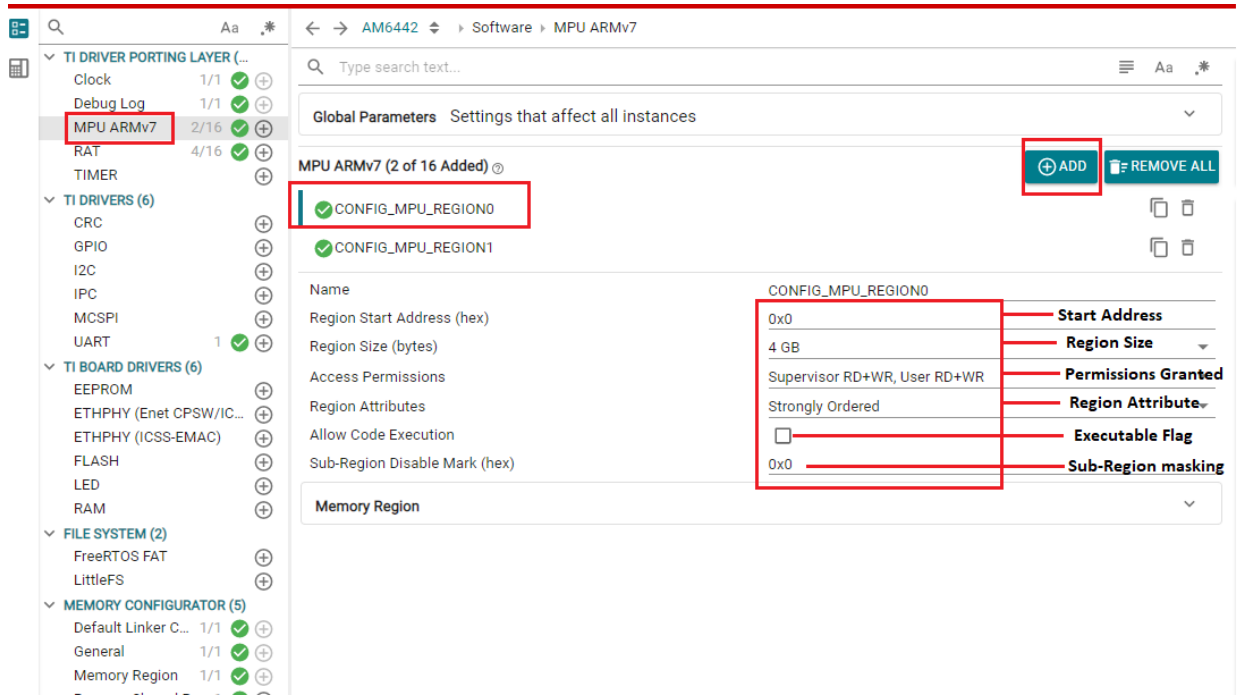

图 4-2. MPU 配置

4.3 MMU 配置

MMU 代表内存管理单元。MMU 配置支持对配置的区域进行虚拟内存转换、内存保护和高速缓存管理。MPU 适用于 R5F 或 M4F 内核，而 MMU 适用于 A53 内核并支持虚拟地址转换和高速缓存管理。SysConfig 会根据所选的内核上下文，为 MPU 和 MMU 提供独立的视图和配置。

图 4-3 显示了“MPU 配置”的视图。

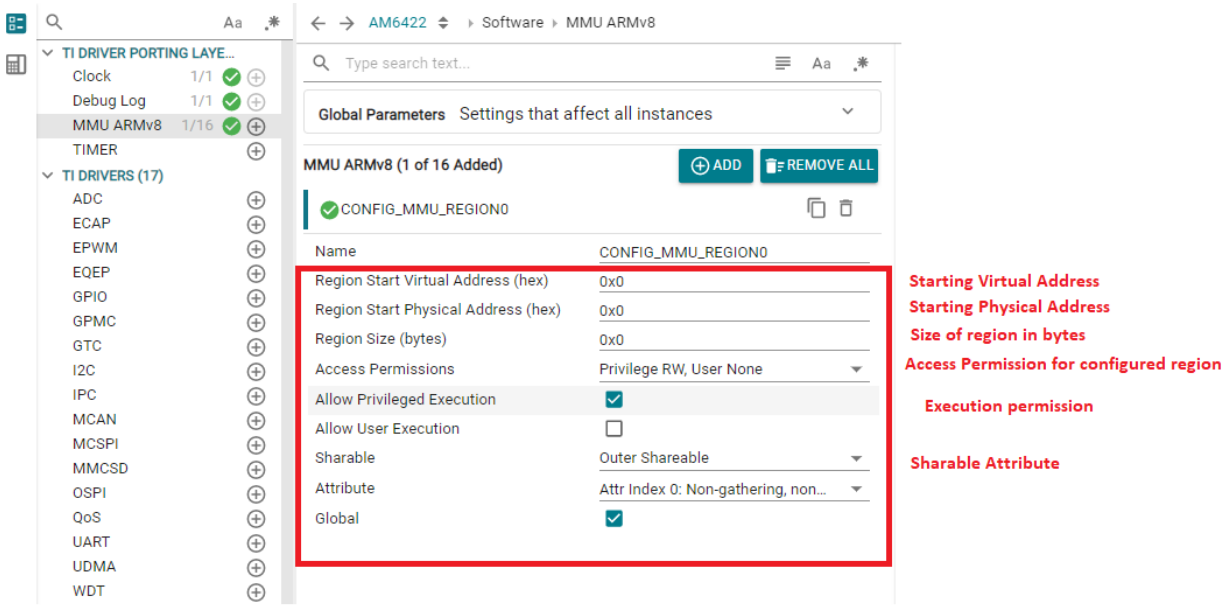

图 4-3. MMU 配置

4.4 系统初始化

SysConfig CodeGen 工具会针对已配置的外设生成用于启用时钟、执行 pinmux 设置和驱动程序初始化的代码。图 4-4 显示了用于初始化的代码。

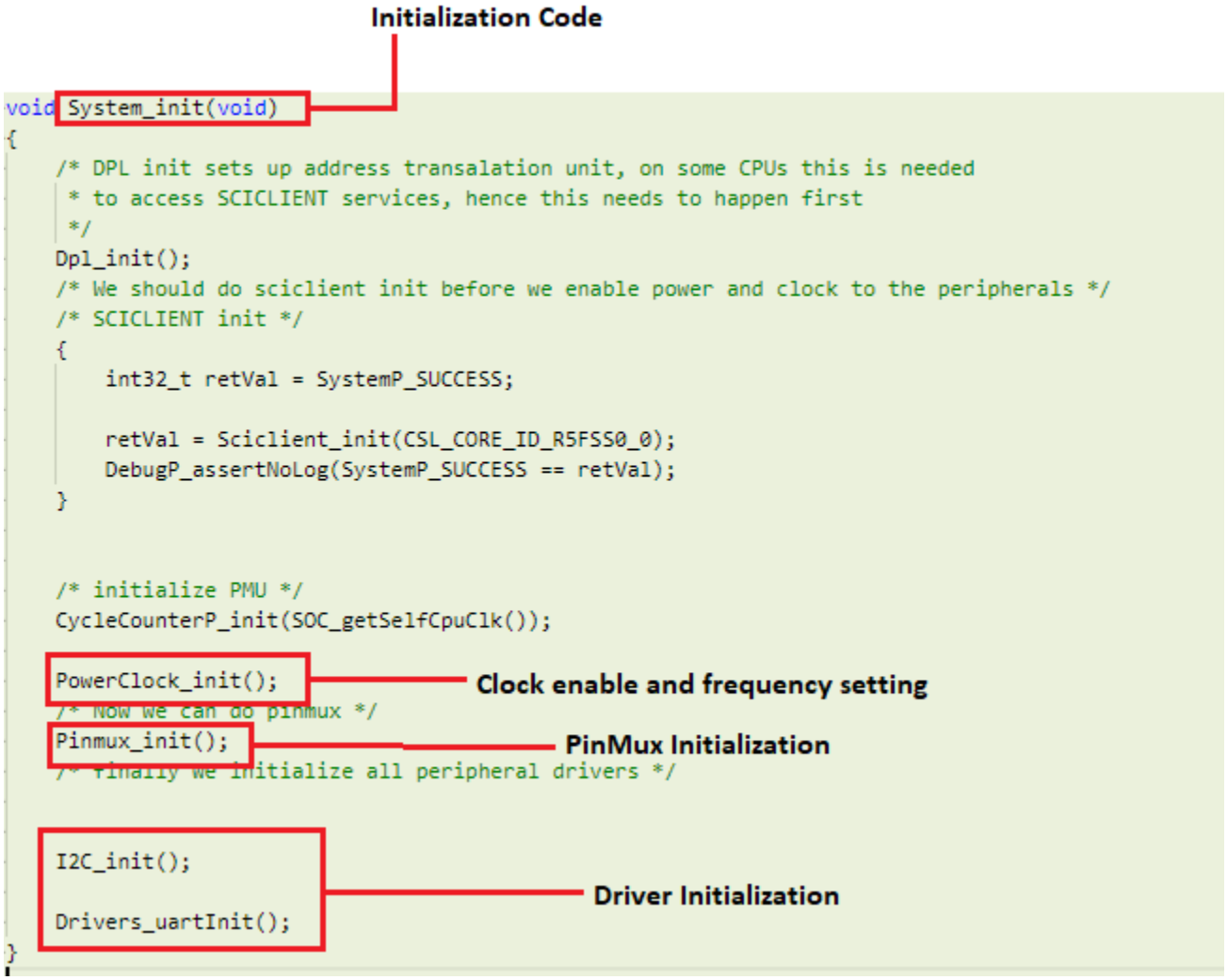

图 4-4. 系统初始化

以下章节将逐一详细说明。

4.4.1 DPL 初始化

CodeGen 代码会生成用于 DPL 初始化的代码。DPL 代表驱动程序移植层。DPL 初始化可启用 Interrupts（中断）、初始化系统时钟并完成定时器初始化。

ti_dpl_config.c

```
144 144 void __mmu_init()
145 145 {
146 146     MmuP_init();
147 147     CacheP_enable(CacheP_TYPE_ALL);
148 148 }
149 149
150 150 void Dpl_init(void)
151 151 {
152 152     /* initialize Hwi but keep interrupts disabled */
153 153     HwiP_init();
154 154
155 155     /* init debug log zones early */
156 156 #if defined(SMP_FREERTOS)
157 157     /* Initialize Debug module from Core0 only */
158 158     if(0 == Armv8_getCoreId())
159 159     {
160 160
161 161     /* Debug log init */
162 162     DebugP_logZoneEnable(DebugP_LOG_ZONE_ERROR);
163 163     DebugP_logZoneEnable(DebugP_LOG_ZONE_WARN);
164 164
165 165     }
166 166 #else
167 167     /* Debug log init */
168 168     DebugP_logZoneEnable(DebugP_LOG_ZONE_ERROR);
169 169     DebugP_logZoneEnable(DebugP_LOG_ZONE_WARN);
170 170 #endif
171 171
172 172 #if defined(SMP_FREERTOS)
173 173     /* Initialize Clock from Core0 only */
174 174     if(0 == Armv8_getCoreId())
175 175     {
176 176
177 177     /* initialize Clock */
178 178     ClockP_init();
179 179
180 180
181 181     }
182 182 #else
183 183
184 184     /* initialize Clock */
185 185     ClockP_init();
186 186
187 187 #endif
188 188
189 189     TimerP_init();
190 190
191 191
192 192 }
193 193
```

图 4-5. DPL 初始化

4.4.2 时钟初始化

在应用程序中使用外设之前，必须通过启用和配置相应的时钟设置来正确初始化该外设。时钟初始化通过调用 TISCI API 实现。添加外设时，该工具会自动生成时钟配置所需的代码。

当用户在 CodeGen 工具中添加任何模块/外设时，配置的时钟参数会自动填入 TISCI API 用于设置频率的结构体（例如 gSocModulesClockFrequency）中。Module_clockEnable() API 通过启用时钟的 LPSC 门来完成模块的电源配置。

图 4-6 显示了由工具生成的用于启用和配置外设时钟的代码。

```c
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++;
    }
}
```

```c
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();
}
```

图 4-6. 时钟配置

4.4.3 PinMux 配置

AM243x 和 AM6x 系列器件在多个外设（UART、SPI、I2C、GPIO 等）之间共享有限数量的引脚。引脚多路复用 (PinMux) 用于选择哪个外设连接哪个物理焊球和引脚。若未正确配置 PinMux，外设将无法与外部器件通信。分配冲突（例如，UART 和 I2C 共用同一引脚）会导致启动或运行时故障。使用 CodeGen 工具可以轻松避免所有此类冲突。

当在 CodeGen 工具中添加模块和外设时，SysConfig 工具会显示模块所需的引脚，并生成相应的代码。用户还可以使用该工具选择要启用/禁用输入的引脚设置，或者将引脚配置为具有上拉或下拉功能。

针对 MCU 和 MAIN 域外设，系统会配置单独的结构体或引脚组。

图 4-7 显示了“Pinmux 初始化”生成的代码。

```c
#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);
}
```

图 4-7. PinMux 初始化

4.4.4 驱动程序初始化

若要在应用程序中使用任何外设，需要进行驱动程序初始化才能确保其正常运行。SysConfig CodeGen 工具为所配置的外设生成驱动程序配置代码。

该工具使用 Drivers_Init/Deinit()、Drivers_Open/Close() API 作为实际驱动程序初始化代码的封装函数。驱动程序初始化代码源自 SDK 中的驱动程序，并非由 CodeGen 工具自动生成。

CodeGen 工具会使用配置的值填充所需结构体。这些填充的结构体由 SDK 的驱动程序 API 使用。

```
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;
        }
    }
```

图 4-8. 驱动程序配置

4.4.5 板级外设初始化

CodeGen 工具还生成用于板级外设初始化的源文件。此文件包含用于执行所配置板级驱动程序初始化的 API 定义，还提供用于打开和关闭驱动程序的 API。

生成的 API 定义可在 **ti_board_open_close.c** 文件中查看。

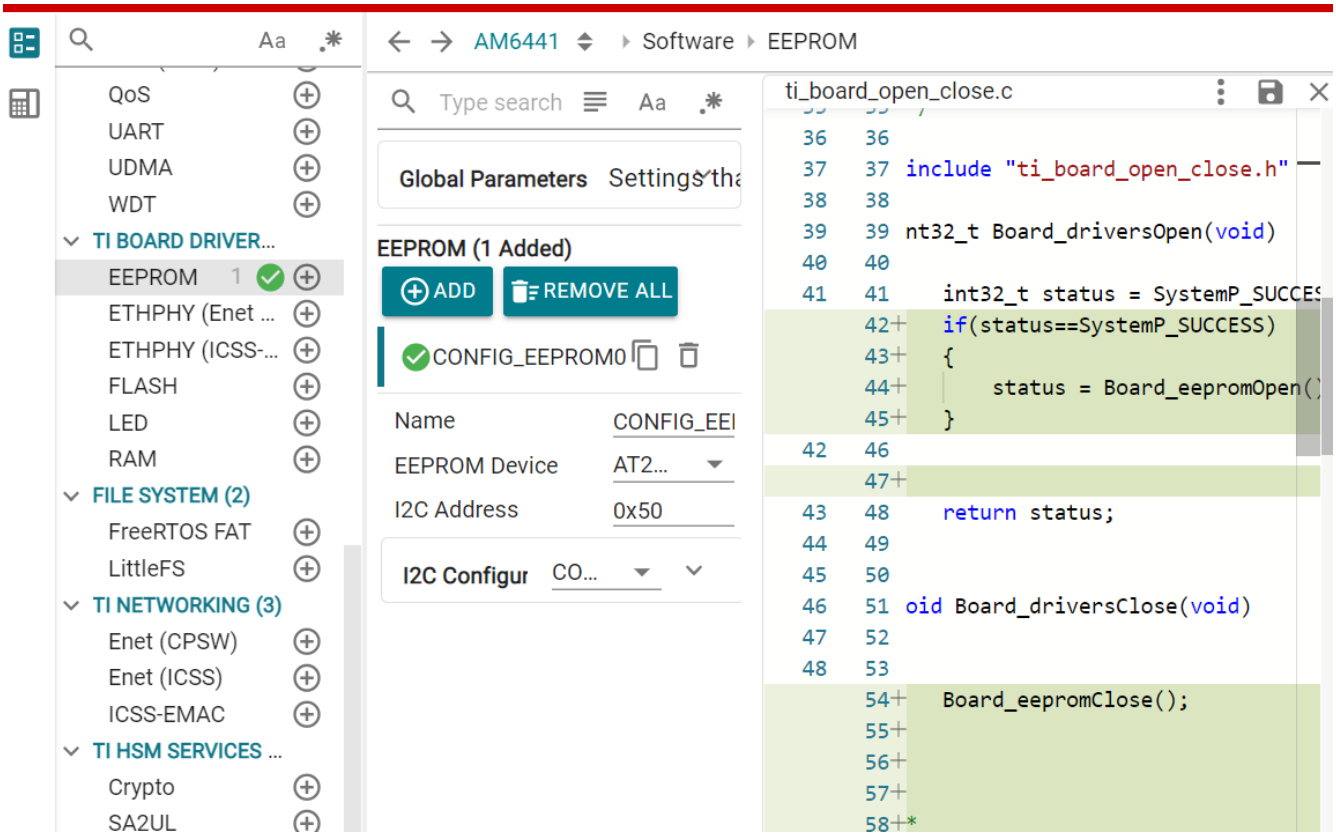

图 4-9. 板级外设驱动程序

5 输出文件

5.1 CodeGen 工具生成的文件

SysConfig 的 CodeGen 工具可生成 C 源代码文件和头文件，这些文件在应用程序开发期间用于避免错误并提升工作效率。生成的文件可以直接导入到应用程序中，用于完成驱动程序初始化和配置。

以下是该工具生成的文件。

1. *ti_dpl_config.h* – 包含 DPL（驱动程序移植层）初始化 API 的声明。
2. *ti_dpl_config.c* – 包含 DPL 初始化的代码。DPL 初始化涵盖中断控制器初始化、MMU 与 RAT 配置、调试日志与系统节拍初始化。DPL 初始化是通过由生成的代码调用的内核级 API 完成的。
3. *ti_drivers_config.h* – 包含所有所配置驱动程序的驱动程序初始化 API 的声明。
4. *ti_drivers_config.c* – 包含用于所配置外设驱动程序初始化、时钟初始化、PinMux 设置和驱动程序初始化的代码。此文件还包含所配置外设的全局句柄。
5. *ti_drivers_open_close.h* – 包含所配置外设的驱动程序打开/关闭 API 以及所需处理程序的声明。
6. *ti_drivers_open_close.c* – 包含所配置外设的打开/关闭驱动程序的代码。此文件还包含处理程序以及添加的外设所需的配置参数。
7. *ti_pinmux_config.c* – 包含所配置外设实现所需功能（通过 GUI 配置）所需的 pinmux 配置。
8. *ti_power_clock_config.c* – 包含用于启用所配置外设的时钟并修改时钟频率的代码。生成的代码通过 TISCI 调用来配置时钟频率。
9. *ti_board_config.h* – 包含板级专用驱动程序配置的声明。
10. *ti_board_config.c* – 包含板级专用驱动程序配置的定义。
11. *ti_board_open_close.h* – 包含板级专用驱动程序打开/关闭 API 的声明。
12. *ti_board_open_close.c* – 包含板级专用驱动程序打开/关闭 API 的定义。
13. *ti_enet_config.h* – 包含 enet 模块使用的所有宏的定义。
14. *ti_enet_config.c* – 包含实现 enet 功能所需的全局结构和 API 的定义。
15. *ti_enet_open_close.h* – 包含 enet 打开/关闭 API 以及所需实用程序 API 的声明。
16. *ti_enet_open_close.c* – 包含 enet 打开/关闭 API 的定义以及所需结构体的定义。
17. *ti_enet_soc.c* – 包含 enet 中断设置、时钟频率配置以及设置/获取其他必要配置所需的结构体与 API 的定义。
18. *ti_enet_lwipif.h* – 包含用于驱动程序回调的 enet Lwip 接口层的声明。
19. *ti_enet_lwipif.c* – 包含用于驱动程序回调的 enet Lwip 接口层的实现。

5.1.1 调试与故障排除

使用 SysConfig 应用程序时，如果传递了错误的 cliArgs 参数，SysConfig 工具会报告错误消息。通过查看错误消息，我们可以识别错误的原因。除了 cliArgs 错误外，在使用工具时可能还会出现其他问题。

以下章节将讨论在使用 SysConfig 时可能出现的一些常见问题以及解决这些问题的步骤。

5.2 版本不匹配

在 图 5-1 中，由于 MCU SDK 版本与 SysConfig 工具之间的版本不匹配，系统会显示错误消息 *Update Required*（需要更新）。syscfg 文件中使用的 cliArgs 错误，因此在打开 GUI 视图时该工具会报告错误。

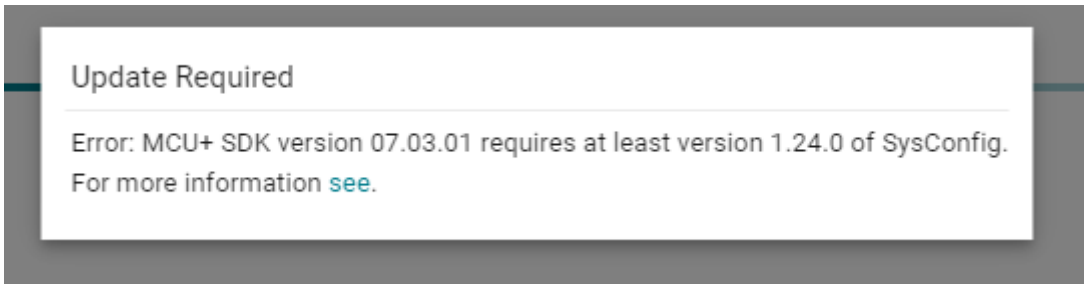

图 5-1. 版本不匹配

要解决上述问题，请确保按照 MCU SDK 文档中所述要求使用正确版本的 SysConfig。请查看 MCU SDK 中提供的 **product.json** 文件，了解版本详细信息。

前面的示例中使用的是 SysConfig v1.23.0 和 MCU+SDK v11.00。example.syscfg 文件中使用的 cliArgs 如下所示，其 MCU SDK 版本不正确。

```
/** * 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"} */
```

修改上述 cliArgs，在 example.syscfg 文件中设置正确的 MCU SDK 版本后，该工具即可正常运行。

```
/** * 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"} */
```

如果是其他问题，则可能会出现*未找到器件型号*、*未找到封装/零件编号*等提示。有关详细信息，请参阅 图 5-2、图 5-3 和 图 5-4。

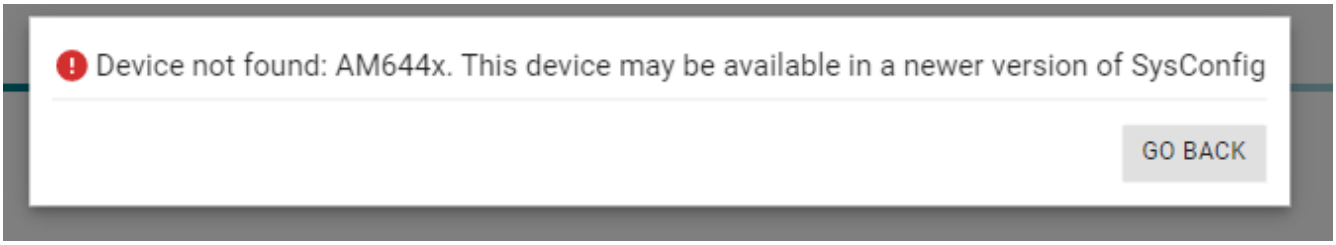

图 5-2. 未找到器件

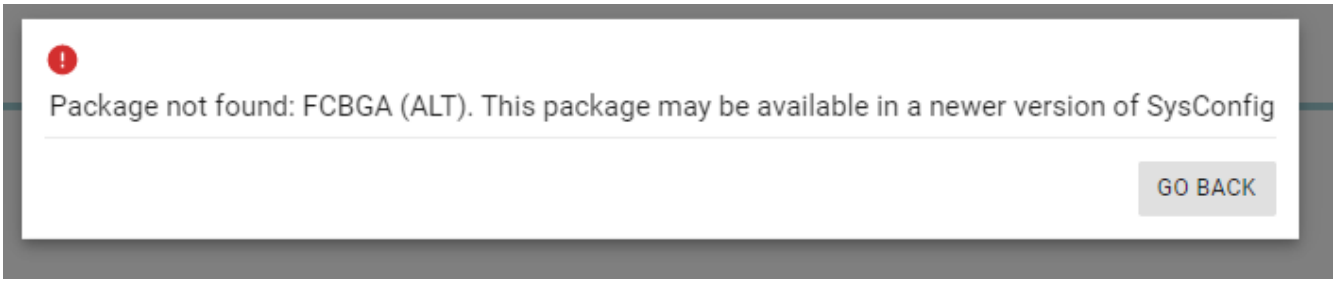

图 5-3. 未找到软件包

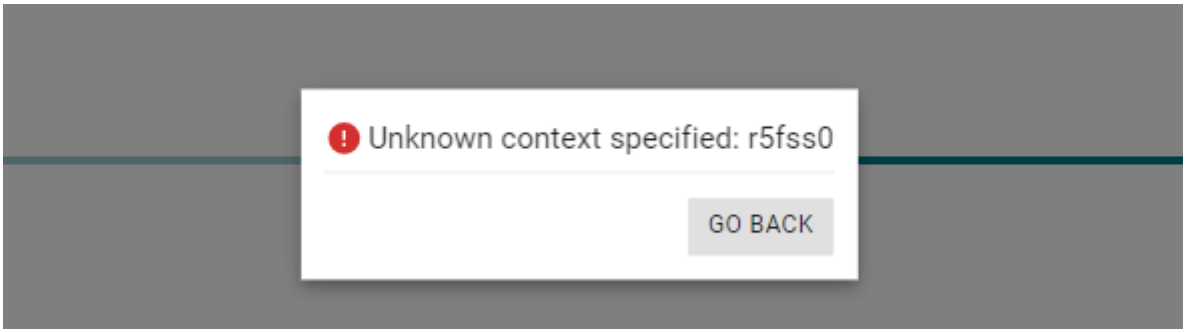

图 5-4. 未指定上下文信息

所有上述参数均必须在 **example.syscfg** 文件的 cliArgs 中正确传递。传递错误的参数将导致出现上述问题之一。

如果用户仍然对要在 cliArgs 中使用的参数感到困惑，请打开 CodeGen 工具并选择 MCU SDK 作为软件产品，然后从生成的 **untitled.syscfg** 文件复制 cliArgs。

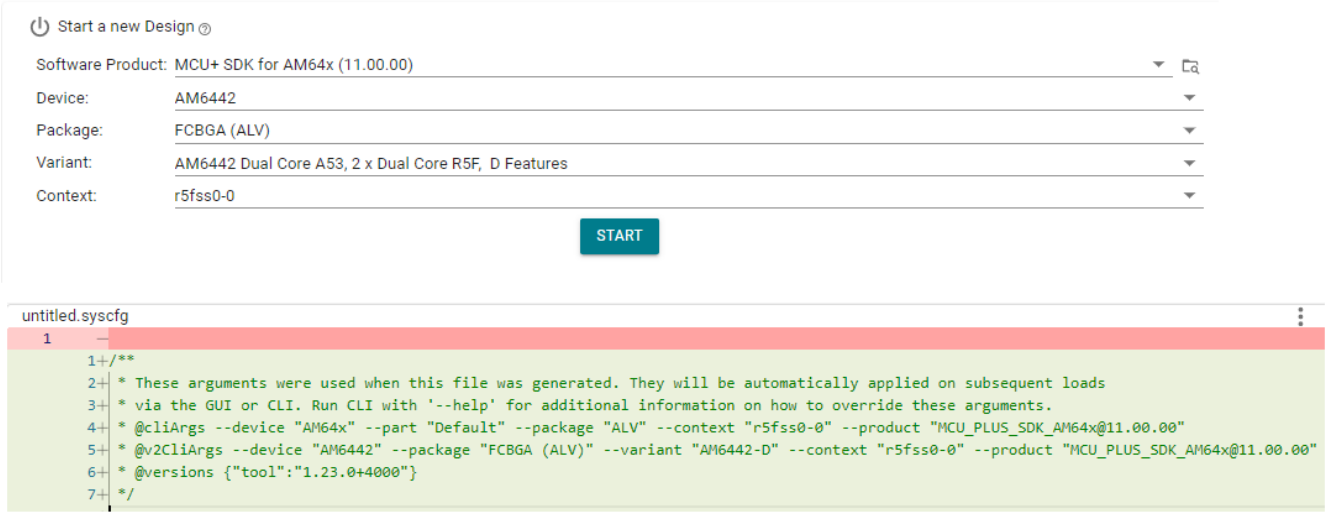

图 5-5. SysConfig CodeGen cliArgs

5.3 资源冲突

使用 CodeGen 工具进行开发时，可以轻松识别并解决冲突。CodeGen 工具会检测用户在进行手动配置时可能发生的各种冲突，并且还会弹出相应的错误消息。以下章节将详细讨论可通过该工具识别并解决的冲突类型。

5.3.1 引脚冲突

当为多个功能配置任一引脚时，SysConfig 工具会报告错误。

例如，如果用户将 GPIO 引脚（焊球 T20）和同一引脚（焊球 T20）配置给 GPMC。当为多种功能配置 T20 引脚时，SysConfig 工具会报告资源冲突问题。

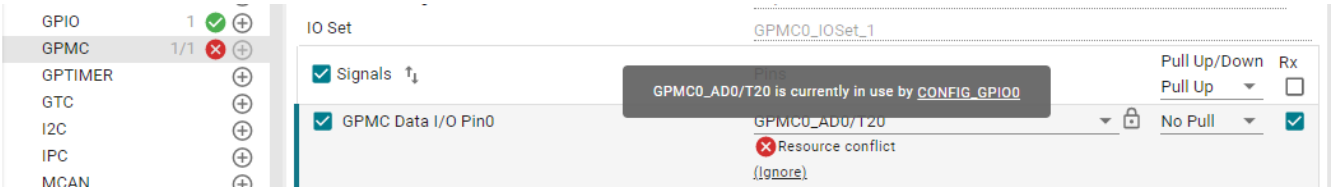

图 5-6. 引脚冲突

通过从 GPIO 或 GPMC 外设上移除 T20 引脚，即可轻松解决该问题。

5.3.2 模块实例冲突

当特定模块下的某个实例被配置多次时，SysConfig 工具会报告错误。

例如，如果用户已配置 UART 模块并添加了两个 UART 实例。如果 UART 模块下的两个或多个实例尝试配置同一个 UART（例如 UART0）外设，则该工具会报告实例冲突错误。

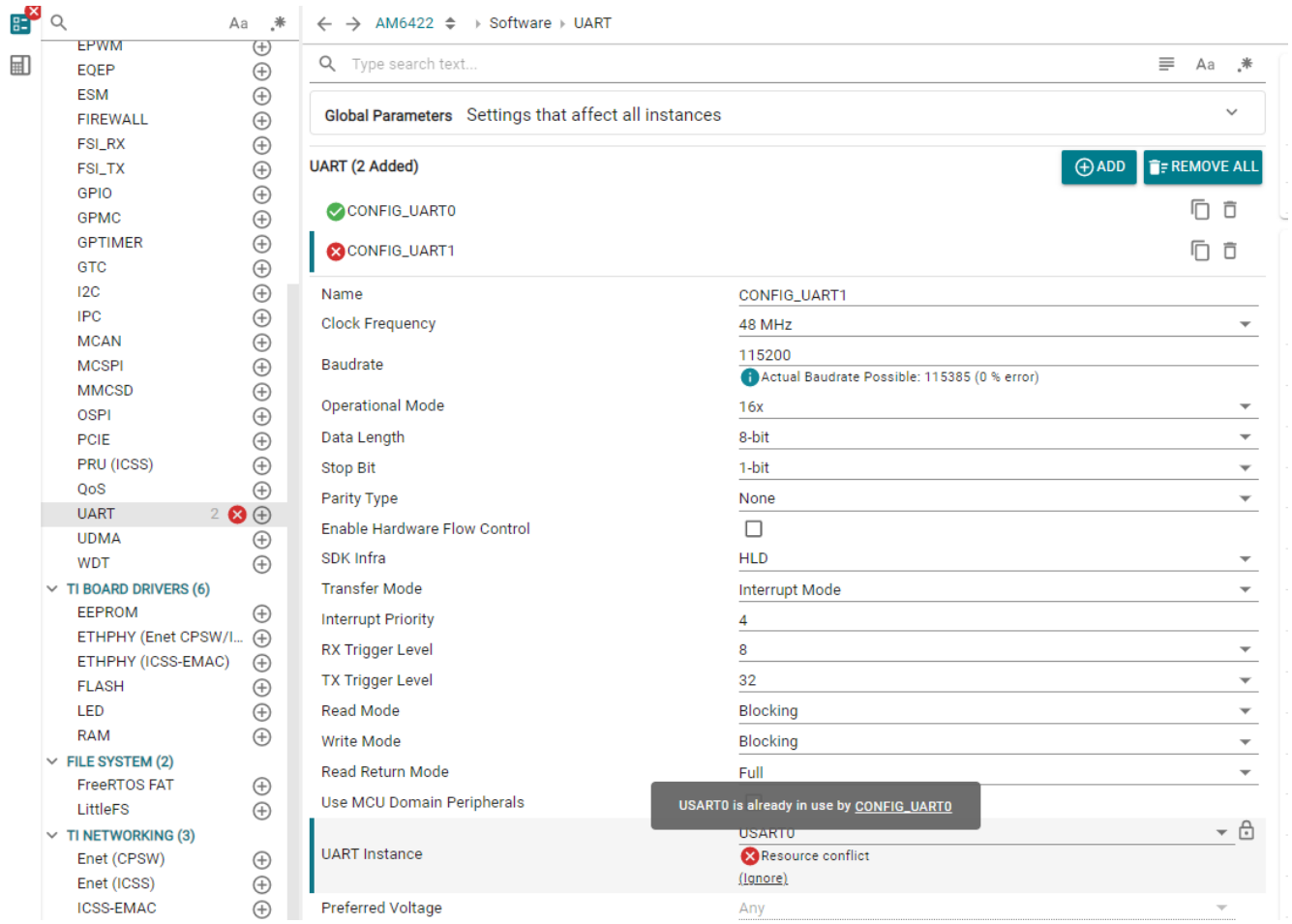

图 5-7. 实例冲突

通过为不同实例配置不同的 UART，即可轻松解决该问题。

5.3.3 多核资源冲突

在多核项目中，用户可以在两个不同的内核中配置相同的资源。在这种情况下，该工具会自动检测内核之间的冲突原因并弹出错误消息。

例如，如果在多核项目中，为 R5F0-0 内核配置了一个 GPIO 引脚，又为 R5F0-1 内核配置了同一引脚，则工具将提示资源冲突错误。

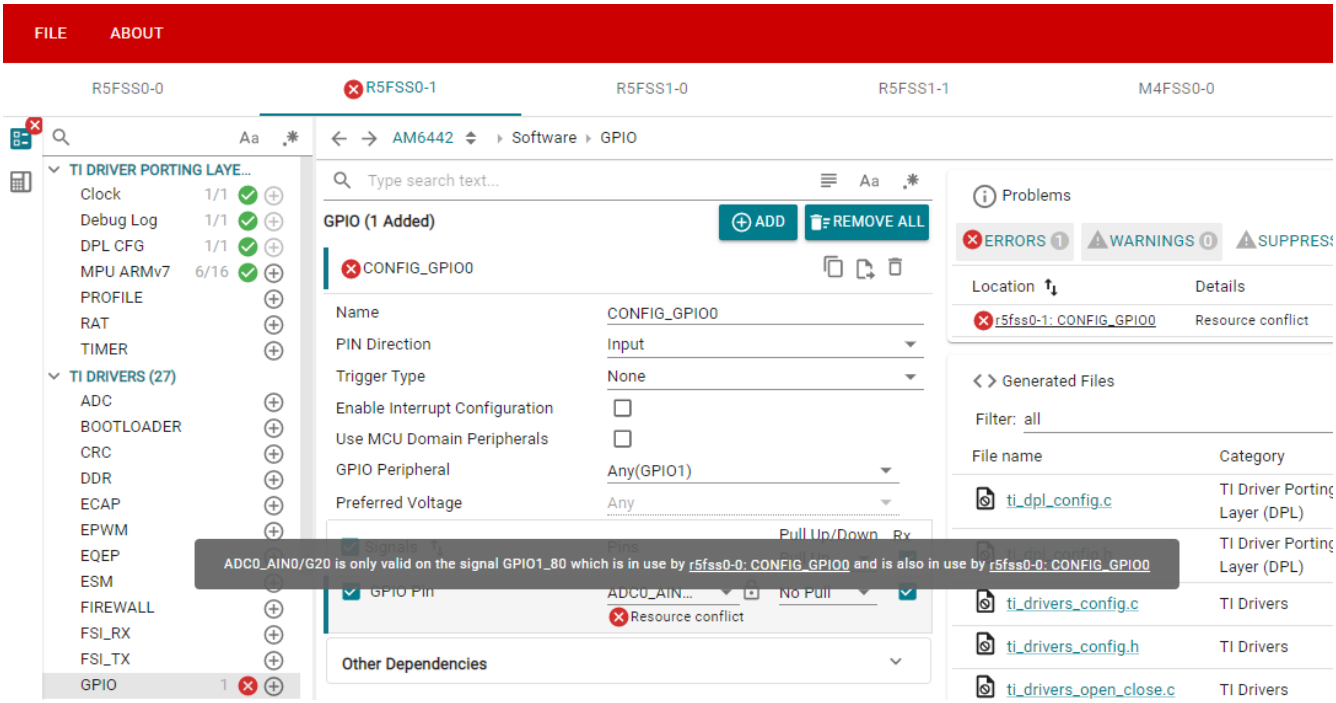

图 5-8. 多核资源冲突

只需为不同的内核配置不同的 GPIO 引脚，即可解决该问题。

5.4 不支持的驱动程序

在以下示例中，用户必须在应用程序中使用 OSPI 驱动程序，但工具的 *TI 驱动程序*列表并未包含 OSPI 模块。有关详细信息，请参阅 图 5-9。

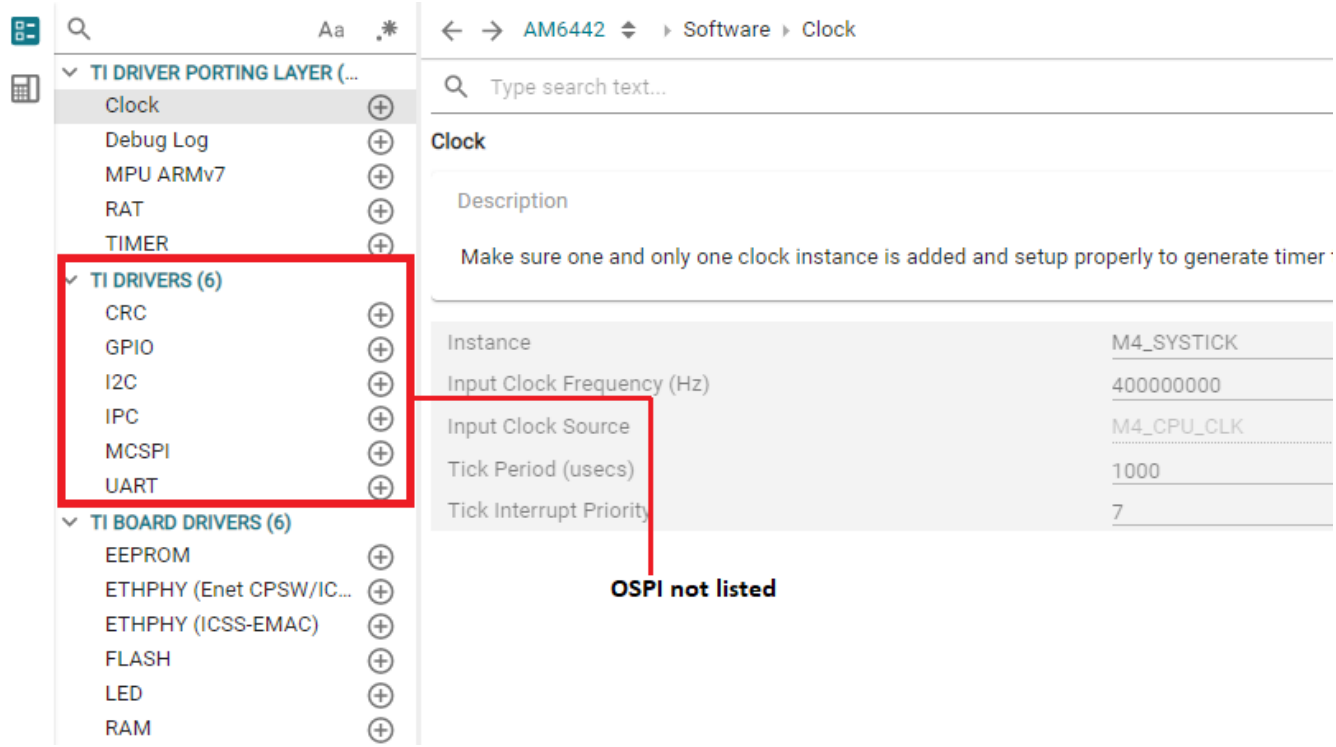

图 5-9. TI 驱动程序

MCU SDK 不支持该工具的 *TI 驱动程序*部分下未列出的驱动程序。该工具中的驱动程序列表可能随不同的内核组合而变化。

有关受支持的驱动程序列表的详细信息，请参阅 MCU SDK 版本说明。

5.5 使用“保留外设”

“保留外设”选项卡用于保留自定义代码可以使用的任何硬件资源，该选项卡会告知 SysConfig 工具不使用该外设资源。对于在“保留外设”下配置的外设，SysConfig 工具不会为其生成任何代码。此选项卡不得用于任何必须由该工具进行配置的外设。

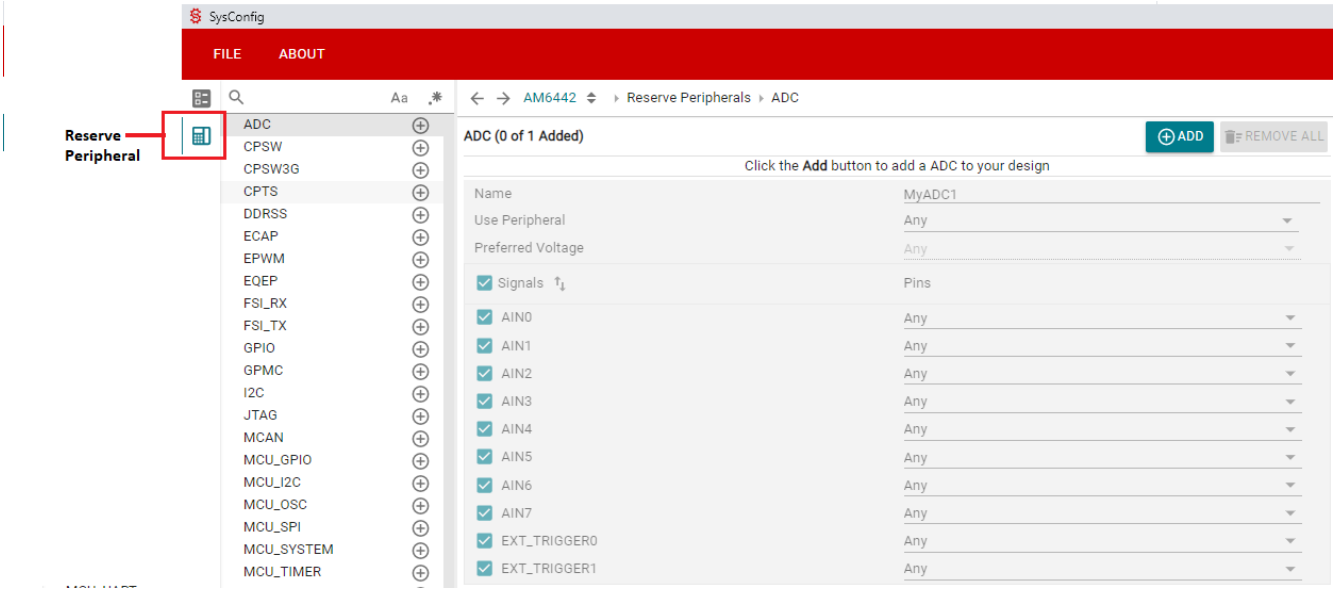

图 5-10. SysConfig“保留外设”选项卡

6 免责声明与预期用途

SysConfig 核心工具遵循 TI 基线质量开发流程。这意味着对于使用 SysConfig 生成的代码，无法做出任何有关汽车级或功能安全方面的声明。客户有责任根据特定标准的要求对生成的代码执行标准鉴定。

7 总结

SysConfig 通过自动为 TI SoC 生成初始化和配置代码，显著加快了软件启动速度。

GUI 和 CLI 界面可最大程度地减少手动操作，验证配置一致性，并提升多核项目的工作效率。

通过将 SysConfig 集成到 MCU+SDK 工作流程中，开发人员可以快速对嵌入式系统进行原型构建和扩展，同时降低配置错误的风险。

8 参考资料

- TI 云工具
  - SysConfig
  - Resource Explorer

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月