Application Note

MSPM0Gx51x to MSPM33C32xx Software Migration Guide

Eric Rentschler

ABSTRACT

MSPM0Gx51x and MSPM33C32xx share many packages, features, and pinouts. As a result, it is possible to accommodate both MSPM0Gx51x and MSPM33C32xx using the same layout. This application note discusses software considerations when migrating between devices. The Hardware Migration Guide (SLAAES2) can also be used.

Table of Contents

Trademarks

1 Introduction

This document describes software considerations when migrating between MSPM0Gx51x and MSPM33C32xx.

Modern microcontrollers contain increasing amounts of features and peripherals. Thus migrating between devices can be a challenge. To simplify the process, this document highlights differences between the MSPM0Gx51x and MSPM33C32xx families of devices, but the same principles can be applied to any M0 and M33 device

This guide is most effectively used ahead of the migration or design process. Follow the steps below to address each aspect of compatibility considerations between MSPM0Gx51x and MSPM33C32xx. Having both datasheets available supports the migration process.

1. Select a target MSPM0Gx51x device and a target MSPM33C32xx device
2. Review feature level differences
3. Review pinout differences

2 Review of Software Differences Between MSPM0Gx51x and MSPM33C32xx

The MSPM0Gx51x family and MSPM33C32x family share some features and specifications, however the sections below describe differences that influence software migration.

2.1 Development Environments

Software Development Tools

The following tools are recommended to develop software for MSP products:

MSPM33 Software Development Kit: MSPM33SDK
Code Composer Studio
Example Code
Driver Library
SysConfig
CCS-Integrated SysConfig
Debugging Interface

2.2 Software Porting Flow

The below figure shows the steps of software migration from MSPM0Gx51x to MSPM33C class devices. The following sections describe the difference details between the deployment environments, and a brief overview of the differences in peripherals, in conjunction with SLAAES2.

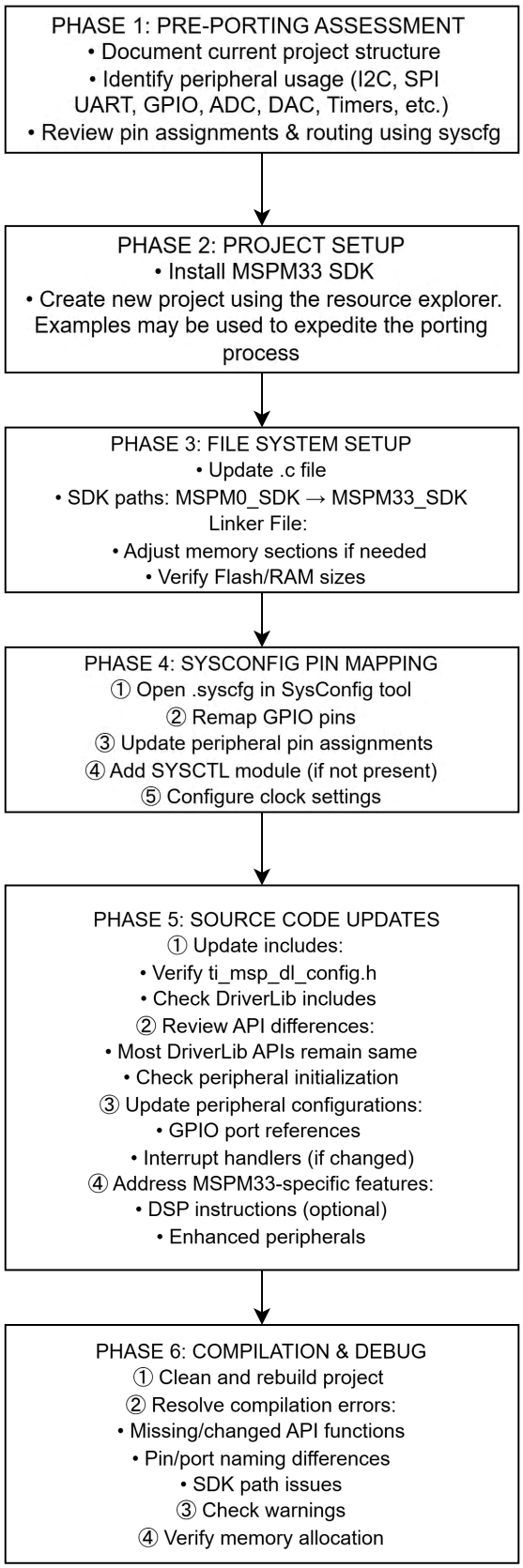

Figure 2-1. Software Porting Flow Diagram

This flow chart describes the migration process. Despite the similarities between the SDKs, the engineer is encouraged to implement one subsystem at a time, so performance can be verified at each step.

2.3 Peripheral Differences

The following sections describe the migration considerations for different peripherals.

2.3.1 Interrupt Differences

One of the most significant differences between the MSPM0SDK and MSPM33SDK is the formatting surrounding interrupts. The MSPM0G family is limited to 32 device interrupts, while the M33 core supports up to 496 interrupts, with each peripheral having a dedicated interrupt vector. This means that a GPIOA interrupt previously handled by GROUP_1_IRQHandler, are instead handled by GPIOA_IRQHandler. Additionally, any status APIs associated with groups are no longer applicable, and must be transferred to peripheral specific APIs.

In a majority of applications, interrupts can be one of the largest changes that must be made. The change is simple however, as interrupt requests (IRQs) are moved to the peripherals themselves, rather than utilizing one large group for all peripherals contained within them.

2.3.2 ADC Differences

The ADC peripheral in the MSPM33C family has a collection of notable changes when compared with the ADC12 in the MSPM0x51x. While both are 12-bit SAR ADC peripherals, the MSPM33C's High Speed Analog to Digital Converter (HSADC) has additional sequencing, oversampling, and post-processing features that must be considered when transitioning to the device.

The following can have minimal changes apart from naming conventions:

- Basic channel selection
- Power Management (enable/disable/reset)
- Basic interrupts
- Clock source selection

The following can require more significant redesign:

- Conversion sequencing
- Hardware averaging
- Result handling
- Interrupt configuration
- Triggering

The figure below displays how each feature differs between the two chips. If the code to be ported utilizes these features, many constants must be changed either using SysConfig or via definitions in the application code.

Peripheral Specification	ADC12 (MSPM0Gx51x)	HSADC (MSPM33Cx)
Conversion mode	Single sample, repeat single sample, sequence, repeat sequence	4 independent sequencers (SEQ1-4) with priority, preemption support
Triggering	Triggered with DL_ADC12_TRIG_SRC_SOFTWARE, DL_ADC12_TRIG_SRC_EVENT	Triggered per-sequencer using DL_HSADC_Trigger_TieLow_SW_Trig, DL_HSADC_Trigger_GEN_SUB_X
Post-Processing	Basic hardware averaging (2-128 samples), Simple divide-by options through SysConfig	4 Post Processing Blocks (PPB) with offset subtraction, two's complement, configurable oversampling, right-shift capabilities, trip comparators, error calculation
Results Storage	12 MEMRES registers (MEMRES0-11), simple FIFO mode	16 result registers (ADCRESULT0-15), 4 PPB result registers, per-sequencer FIFO with status, oversampling sum/count registers
Interrupts	DL_ADC12_INTERRUPT_MEM0_RESULT_LOADED, DL_ADC12_INTERRUPT_OVERFLOW, DL_ADC12_INTERRUPT_WINDOW_COMP_HI/LO	4 configurable interrupts, 4 dma interrupts, PPB trip interrupts per block, early interrupt generation with preemption

Peripheral Specification	ADC12 (MSPM0Gx51x)	HSADC (MSPM33Cx)
Sample Timers	2 sample timers (SCOMP0/1), Sample time = SCOMP +1 * CLK_DIV	Per-sequencer sample windows between 1 and 1472 cycles, simple cap reset options using VrefLo and VrefHi
Clock Configuration	DL_ADC12_CLOCK_DIVIDE_1 … DL_ADC12_CLOCK_DIVIDE 48, 48 total division options	DL_HSADC_CLOCK_DIVIDE_1_0, DL_HSADC_CLOCK_DIVIDE_2_0, DL_HSADC_CLOCK_DIVIDE_2_5... DL_HSADC_CLOCK_DIVIDE_8_5, from 1 to 8.5
Channels	DL_ADC12_INPUT_CHAN_0 through DL_ADC12_INPUT_CHAN_31	DL_HSADC_IN_0 through DL_HSADC_IN_31

2.3.3 Security Differences

Many of the security features between the MSPM0Gx51x and MSPM33Cx are virtually identical, with compatible wrappers. Below are highlighted necessary wrapper changes in each security peripheral.

AESADV Feature	MSPM0SDK	MSPM33SDK
Naming Conventions	DL_AESADV_* AESADV_REGS	The all functions and structures are identical, with the exception of adding HP at the end of "AESADV" DL_AESADVHP_* AESADVHP_Regs
Interrupt Register Structure	• aesadv->CPU_INT.IMASK • aesadv->CPU_INT.RIS • aesadv->CPU_INT.MIS • aesadv->CPU_INT.ICLR • aesadv->CPU_INT.IIDX • aesadv->DMA_TRIG_DATAIN.IMASK • aesadv->DMA_TRIG_DATAOUT.IMASK	• aesadv->INT_EVENT0.IMASK • aesadv->INT_EVENT0.RIS • aesadv->INT_EVENT0.MIS • aesadv->INT_EVENT0.ICLR • aesadv->INT_EVENT0.IIDX • aesadv->INT_EVENT1.IMASK • aesadv->INT_EVENT2.IMASK

2.4 Communications Peripherals

I2C, UART, and SPI are implemented through UNICOMM on the MSPM33Cx, as opposed to the individual peripherals on the MSPM0Gx51x. While operating each peripheral is similar, the wrapper implementations are different and neccesitate changes on any implementation of any UNICOMM enabled peripheral when transferring to MSPM33Cx. Below is a table enumerating the differences in wrappers between the MSPM0SDK and MSPM33SDK.

2.4.1 SPI Differences

SPI Peripheral Feature	SPI (MSPM0SDK)	UNICOMM SPI (MSPM33SDK)
Wrapper	Direct access to registers through SPI_Regs	SPI wrapped in UNICOMM accessed using UNICOMM_Inst_Regs->spi
Power Management	DL_SPI_enablePower(SPI0)	DL_SPI_enablePower(UNICOMM0)
Clock Source	• DL_SPI_CLOCK_BUSCLK • DL_SPI_CLOCK_MFCLK • DL_SPI_CLOCK_LFCLK	• DL_SPI_CLOCK_BUSCLK • DL_SPI_CLOCK_MFCLK • DL_SPI_CLOCK_LFCLK • DL_SPI_CLOCK_ASYNC_LFCLK • DL_SPI_CLOCK_ASYNC_SYSCLK • DL_SPI_CLOCK_ASYNC_HFCLK • DL_SPI_CLOCK_ASYNC_PLL
FIFO Threshold Levels	• DL_SPI_yX_FIFO_LEVEL_3_4_EMPTY • DL_SPI_yX_FIFO_LEVEL_1_2_EMPTY • DL_SPI_yX_FIFO_LEVEL_1_4_EMPTY • DL_SPI_yX_FIFO_LEVEL_ONE_ENTRY • DL_SPI_yX_FIFO_LEVEL_EMPTY	• DL_SPI_yX_FIFO_LEVEL_3_4_EMPTY • DL_SPI_yX_FIFO_LEVEL_1_2_EMPTY • DL_SPI_yX_FIFO_LEVEL_1_4_EMPTY • DL_SPI_yX_FIFO_LEVEL_ONE_ENTRY • DL_SPI_yX_FIFO_LEVEL_EMPTY • DL_SPI_yX_FIFO_LEVEL_ALMOST_EMPTY • DL_SPI_yX_FIFO_LEVEL_ALMOST_FULL
Preemption Additions	No preemption in MSPM0	• DL_SPI_INTERRUPT_DMA_PREIRQ_RX • DL_SPI_INTERRUPT_DMA_PREIRQ_TX • DL_SPI_INTERRUPT_LTOUT • DL_SPI_IIDX_PREIRQ_RX • DL_SPI_IIDX_PREIRQ_TX • DL_SPI_IIDX_LTOUT
Status Register Additions		• DL_SPI_getCurrentCDMODEValue(unicomm) • DL_SPI_isTXFIFOCleared(unicomm) • DL_SPI_isRXFIFOCleared(unicomm)
Suspend/Resume		• DL_SPI_enabledSuspend(unicomm) • DL_SPI_disableSuspend(unicomm)
Packing Feature	• DL_SPI_enablePacking(spi) • DL_SPI_disablePacking(spi) • DL_SPI_isPackingEnabled(spi)	Packing feature removed, all 32-bit operations work without packing enable

2.4.2 UART Differences

UART Peripheral Feature	UART (MSPM0SDK)	UNICOMM UART (MSPM33SDK)
Wrapper	Standalone UART Peripheral accessed via UART_REGS	UNICOMM_Inst_regs->uart
Clock Source Enhancements	• DL_UART_CLOCK_BUSCLK • DL_UART_CLOCK_MFCLK • DL_UART_CLOCK_LFCLK	• DL_UART_CLOCK_BUSCLK • DL_UART_CLOCK_MFCLK • DL_UART_CLOCK_LFCLK • DL_UART_CLOCK_ASYNC_LFCLK • DL_UART_CLOCK_ASYNC_SYSCLK • DL_UART_CLOCK_ASYNC_HFCLK • DL_UART_CLOCK_ASYNC_PLL
API Compatibility macros		High backwards compatability, all DL_UART_MAIN_* and DL_UART_EXTEND_* APIs are mapped to Unicomm
FIFO Management	• DL_UART_enableFIFOs() • DL_UART_disableFIFOs()	FIFO is always enabled, but APIs are still provided for backwards compatability. For example, calling DL_UART_isFIFOsENABLED() always returns true
FIFO Threshold levels	• DL_UART_yX_FIFO_LEVEL_3_4_EMPTY • DL_UART_yX_FIFO_LEVEL_1_2_EMPTY • DL_UART_yX_FIFO_LEVEL_1_4_EMPTY • DL_UART_yX_FIFO_LEVEL_ONE_ENTRY • DL_UART_yX_FIFO_LEVEL_EMPTY	• DL_UART_yX_FIFO_LEVEL_3_4_EMPTY • DL_UART_yX_FIFO_LEVEL_1_2_EMPTY • DL_UART_yX_FIFO_LEVEL_1_4_EMPTY • DL_UART_yX_FIFO_LEVEL_ONE_ENTRY • DL_UART_yX_FIFO_LEVEL_EMPTY • DL_UART_yX_FIFO_LEVEL_ALMOST_EMPTY • DL_UART_yX_FIFO_LEVEL_ALMOST_FULL
Preemption Additions	No compatible features on the MSPM0Gx51x	• DL_UART_IIDX_PREEMPT_INT_TX • DL_UART_IIDX_PREEMPT_INT_RX • DL_UART_INTERRUPT_LINE_TIMEOUT • DL_UART_IIDX_LINE_TIMEOUT
Register naming	• Control: CTLx • Line Control: LCRH • Status: STAT • TX Data: TXDATA • RX Data: RXDATA • Baud Rate: IBRD/FBRD	• Control: CTLx • Line Control: LCRH • Status: STAT • TX Data: TXDATA • RX Data: RXDATA • Baud Rate: IBRD/FBRD
Power Management	• DL_UART_enablePower(UARTx)	• DL_UART_enablePower(UNICOMM0) • DL_UNICOMM_enablePower(unicomm)

UART Peripheral Feature	UART (MSPM0SDK)	UNICOMM UART (MSPM33SDK)
Backup/Restore Configuration	• DL_UART_Main_backupConfig • DL_UART_Extend_backupConfig • DL_UART_Main_saveConfiguration() • DL_UART_Extend_saveConfiguration()	• DL_UART_backupConfig • DL_UART_saveConfiguration() • DL_UART_restoreConfiguration()
Analog Glitch Filter		No analog filter in MSPM33C.
Unmentioned Core Functionalities	UART_Regs_* used	UNICOMM_Inst_Regs_* used
API Compatibility macros		High backwards compatability, all DL_UART_MAIN_* and DL_UART_EXTEND_* APIs are mapped to Unicomm

2.4.3 I2C Differences

I2C Peripheral Feature	I2C (MSPM0SDK)	UNICOMM I2C (MSPM33SDK)
Wrapper	Direct register access via I2C_Regs, supports Controller and Target mode in the same peripheral	I2C Controller wrapped with UNICOMM Module, registers accessed through UNICOMM_Inst_Regs->i2cc/i2ct
Mode Selection	I2C Controller and Target in same peripheral DL_I2C_enableController(I2Cx), DL_I2C_enableTarget(I2Cx)	• Controller enable DL_I2CC_ENABLE(UNICOMMx) • Target enable DL_I2CT_ENABLE(UNICOMMx)
Register naming changes	• Control Register: MCTR • Target Address: MSA • Status Register: MSR • Timer Period: MTPR • Configuration: MCR	• Control Register: CTR • Target Address: SA • Status Register: SR • Timer Period: TPR • Configuration: CR
Clock Sources	• DL_I2C_CLOCK_BUSCLK • DL_I2C_CLOCK_MFCLK	• DL_I2Cx_CLOCK_BUSCLK • DL_I2Cx_CLOCK_MFCLK • DL_I2Cx_CLOCK_ASYNC_SYSCLK • DL_I2Cx_CLOCK_ASYNC_HFCLK • DL_I2Cx_CLOCK_ASYNC_PLL "I2Cx" refers to I2CC for controller or I2CT for target

I2C Peripheral Feature	I2C (MSPM0SDK)	UNICOMM I2C (MSPM33SDK)
FIFO Threshold Levels	• DL_I2C_TX_FIFO_LEVEL_EMPTY • DL_I2C_TX_FIFO_LEVEL_BYTES_x • DL_I2C_RX_FIFO_LEVEL_BYTES_x LEVEL_BYTES_x from 1 to 7	• DL_I2Cx_yX_FIFO_LEVEL_3_4_EMPTY • DL_I2Cx_yX_FIFO_LEVEL_1_2_EMPTY • DL_I2Cx_yX_FIFO_LEVEL_1_4_EMPTY • DL_I2Cx_yX_FIFO_LEVEL_ONE_ENTRY • DL_I2Cx_yX_FIFO_LEVEL_EMPTY • DL_I2Cx_yX_FIFO_LEVEL_ALMOST_EMPTY • DL_I2Cx_yX_FIFO_LEVEL_ALMOST_FULL For target configuraiton, I2CC uses TX and I2CT uses RX
Glitch Filtering	• DL_I2C_ANALOG_GLITCH_FILTER_WIDTH_X • DL_I2C_DIGITAL_GLITCH_FILTER_WIDTH_X	MSPM33C does not have an analog glitch filter. • DL_I2CC_DIGITAL_GLITCH_FILTER_WIDTH_X
API Function Name Changes	• DL_I2C_enableController() • DL_I2C_startControllerTransfer() • DL_I2C_transmitControllerData() • DL_I2C_receiveControllerData() • DL_I2C_getControllerStatus() • DL_I2C_fillControllerTXFIFO()	• DL_I2Cx_enable() • DL_I2Cx_startTransfer() • DL_I2Cx_transmitData() • DL_I2Cx_receiveData() • DL_I2Cx_getStatus() • DL_I2Cx_fillTXFIFO()
Interrupt Naming	• DL_I2C_INTERRUPT_CONTROLLER_RX_DONE • DL_I2C_INTERRUPT_CONTROLLER_TX_DONE • DL_I2C_INTERRUPT_CONTROLLER_RXFIFO_TRIGGER • DL_I2C_INTERRUPT_CONTROLLER_TXFIFO_TRIGGER • DL_I2C_INTERRUPT_CONTROLLER_NACK	• DL_I2Cx_INTERRUPT_RX_DONE • DL_I2Cx_INTERRUPT_TX_DONE • DL_I2Cx_INTERRUPT_RXFIFO_TRIGGER • DL_I2Cx_INTERRUPT_TXFIFO_TRIGGER • DL_I2Cx_INTERRUPT_NACK
Pre-Emptive DMA Interrupts	MSPM0 does not support pre-emptive DMA interrupts.	• DL_I2Cx_IIDX_PREEMPT_INT_TX • DL_I2Cx_IIDX_PREEMPT_INT_RX

3 Summary

In summary, this application notes supplies guidelines for migrating the software from the MSPM0Gx51x to the MSPM33C32xx. It is to be used along with the Hardware Migration Guide (SLAAES2) to perform a full migration between the two devices.

4 References

1. Texas Instruments, MSPM33C321x Mixed-Signal Microcontrollers, data sheet
2. Texas Instruments, MSPM33 C-Series 160MHz Microcontrollers, technical reference manual
3. Texas Instruments, MSPM0Gx51x Mixed-Signal Microcontrollers, data sheet
4. Texas Instruments, MSPM0 G-Series 80MHz Microcontrollers , technical reference manual
5. Texas Instruments, MSPM0Gx51x and MSPM33C32xx Hardware Migration Guide, hardware migration guide

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

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

Last updated 10/2025