



## ABSTRACT

This user's guide assists with migrating from the NXP's Arm® Cortex®-M0+ MCU platform to the Texas Instruments Arm Cortex-M0+ MSPM0 MCU ecosystem. This guide introduces the MSPM0 development and tool ecosystem, core architecture, peripheral considerations, and software development kit. The intent is highlight the differences between the two families and to leverage existing knowledge of the NXP Arm Cortex ecosystem to quickly ramp with the MSPM0 series of MCUs.

## Table of Contents

<b>1 MSPM0 Portfolio Overview</b> .....	2
1.1 Introduction.....	2
1.2 Portfolio Comparison of NXP M0 MCUs to MSPM0.....	2
<b>2 Ecosystem and Migration</b> .....	3
2.1 Software Ecosystem Comparison.....	3
2.2 Hardware Ecosystem.....	4
2.3 Debug Tools.....	5
2.4 Migration Process.....	6
2.5 Migration and Porting Example.....	6
<b>3 Core Architecture Comparison</b> .....	13
3.1 CPU.....	13
3.2 Embedded Memory Comparison.....	13
3.3 Power Up and Reset Summary and Comparison.....	15
3.4 Clocks Summary and Comparison.....	17
3.5 MSPM0 Operating Modes Summary and Comparison.....	19
3.6 Interrupt and Events Comparison.....	21
3.7 Debug and Programming Comparison.....	23
<b>4 Digital Peripheral Comparison</b> .....	25
4.1 General-Purpose I/O (GPIO, IOMUX).....	25
4.2 Universal Asynchronous Receiver-Transmitter (UART).....	25
4.3 Serial Peripheral Interface (SPI).....	26
4.4 I2C.....	27
4.5 Timers (TIMGx, TIMAx).....	27
4.6 Windowed Watchdog Timer (WWDT).....	28
4.7 Real-Time Clock (RTC).....	28
<b>5 Analog Peripheral Comparison</b> .....	29
5.1 Analog-to-Digital Converter (ADC).....	29
5.2 Comparator (COMP).....	30
5.3 Digital-to-Analog Converter (DAC).....	31
5.4 Operational Amplifier (OPA).....	32
5.5 Voltage References (VREF).....	33
<b>6 References</b> .....	34

## Trademarks

TI E2E™, Code Composer Studio™, LaunchPad™, EnergyTrace™, and BoosterPack™ are trademarks of Texas Instruments.

Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All trademarks are the property of their respective owners.

# 1 MSPM0 Portfolio Overview

## 1.1 Introduction

The MSP430™ MCUs have nearly 30 years of history as TI's classic microcontroller. The latest generation introduces the MSPM0 family. MSPM0 microcontrollers (MCUs) are part of the MSP highly-integrated ultra-low-power 32-bit MCU family based on the enhanced Arm Cortex-M0+ 32-bit core platform. These cost-optimized MCUs offer high-performance analog peripheral integration, support extended temperature ranges, and offer small footprint packages. The TI MSPM0 family of low-power MCUs consists of devices with varying degrees of analog and digital integration, allowing engineers to find the MCU that meets their project's needs. The MSPM0 MCU family combines the Arm Cortex-M0+ platform with ultra-low-power system architecture, allowing system designers to increase performance while reducing energy consumption.

MSPM0 MCUs offer a competitive alternative to NXP's M0 MCUs. This document compares device features and ecosystems to assist with migration from these NXP M0 MCUs to TI MSPM0 MCUs.

## 1.2 Portfolio Comparison of NXP M0 MCUs to MSPM0

**Table 1-1. Comparison of the MSPM0 and NXP M0 MCUs**

	32K1xx Series	KEA128x Series	KM35x Series	MSPM0Gx Series	MSPM0Lx Series	MSPM0Cx Series
Core / Frequency	CM0+ / 48MHz	CM0+ / 48MHz	CM0+ / 75MHz	CM0+ / 32-80MHz	CM0+ / 32MHz	CM0+ / 24MHz
Supply Voltage	2.7V to 5.5V	2.7V to 5.5V	2.7V to 3.6V	1.62V to 3.6V	1.62V to 3.6V	1.62V to 3.6V
Temperature	-40°C to 150°C	-40°C to 125°C	-40°C to 105°C	-40°C to 125°C	-40°C to 125°C	-40°C to 125°C
Memory	256 KB to 128KB	128KB	512KB	128KB to 32KB	64KB to 8KB	16KB to 8KB
RAM	Up to 25KB	Up to 16KB	Up to 64KB	Up to 32KB	Up to 4KB	1KB
EEPROM	Up to 2KB	Up to 256B	N/A	Emulated up to 32KB or entire FLASH if device has < 32KB	Emulated up to 32KB or entire FLASH if device has < 32KB	Emulated up to 32KB or entire FLASH if device has < 32KB
GPIO (max)	28/43/58	58/71	72/99	Up to 60	Up to 28	Up to 18
Analog	1x 1Msps, 12-bit ADC, 1x comparator, 1x8-bit DAC	1x 340Ksps, 12-bit ADC, 2x comparator, 2x6-bit DAC	4x AFE, 1x 4.75Msps, 16-bit ADC, 3x Comparator, 3x6-bit DAC	2x 4-Msps, 12-bit ADC, 3x high-speed comparator, 2x op amp, 1x general purpose amp, 1x 12-bit DAC	1x 1.68-Msps, 12-bit ADC, 1x high-speed comparator, 1x general purpose amp, 2x OPA	1x 1.5-Msps, 12-bit ADC
Communication (max)	1/2x SPI, 1x I2C, 2x UART, 1x FlexCAN, 1x FlexIO	2x SPI, 2x I2C, 3x UART, 1x MSCAN	3x SPI, 2x I2C, 5x UART	2x SPI, 2x I2C Fast+, 3x UART, 1x UART -LIN, 1x CAN-FD	1x SPI, 2x I2C Fast+, 1x UART, 1x UART (LIN)	1x SPI, 2x I2C Fast+, 1x UART (LIN)
Timers	2x 16bit	3x 16bit	6x 16-bit	6x 16bit, 1x 32bit	4x 16bit	3x 16bit
Advance Timers	No	No	No	2x 16-bit Advanced	No	No
Hardware Accelerator	N/A	N/A	RNGA	MATHACL	N/A	N/A
Security	CSEc	Unique ID, CRC	Unique ID, MMCAU, RNGA, PRCR	CRC, TRNG, AES256	CRC	CRC
Low power	Active: 422.92/447.92µA/MHz, Standby 26/27µA	Active: 236.170µA/MHz, Standby 2µA	Active: 165.07µA/MHz, Standby 6.5µA	Active: 96µA/MHz, Standby: 1.5µA	Active: 71µA/MHz, Standby: 1µA	Active: 71µA/MHz, Standby: 1µA

## 2 Ecosystem and Migration

MSPM0 MCUs are supported by an extensive hardware and software ecosystem with reference designs and code examples to get designs started quickly. MSPM0 MCUs are also supported by online resources, trainings with MSP Academy, and online support through the [TI E2E™ support forums](#).

### 2.1 Software Ecosystem Comparison

**Table 2-1. NXP Software Tool Equivalents for MSPM0**

	NXP	MSPM0
IDE	MCUXpresso IDE	<a href="#">Code Composer Studio™ Theia</a>
Software Configuration	MCUXpresso Code Configuration Tools	<a href="#">SysConfig</a>
Stand-alone programming	N/A	<a href="#">UniFlash</a>
Display/Demo GUI Editor	GUI Guider	<a href="#">GuiComposer</a>

#### 2.1.1 MSPM0 Software Development Kit (MSPM0 SDK)

The MSPM0 SDK delivers software APIs, examples, documentation, and libraries that help engineers quickly develop applications on Texas Instruments MSPM0+ microcontroller devices. Examples are provided to demonstrate the use of each functional area on every supported device and are a starting point for your own projects. Additionally, interactive MSP Academy trainings are included in the MSPM0 SDK to provide a guided learning path.

The examples folder is divided into RTOS and non-RTOS subfolders. These folders contain examples for each LaunchPad™ development kit and are organized categories such as lower-level DriverLib examples, higher-level TI Drivers examples, and examples for middleware such as GUI Composer, LIN, IQMath, and others. For details, see the [MSPM0 SDK User's Guide](#).

#### 2.1.2 MCUXpresso IDE vs Code Composer Studio IDE (CCS)

[Code Composer Studio Theia IDE \(CCS\)](#) is TI's equivalent of NXP's MCUXpresso IDE. Both CCS and MCUXpresso are free Eclipse-based IDE that supports their respective MCUs and embedded processor portfolios. CCS comprises a suite of tools used to develop and debug embedded applications including an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler and many other features. CCS is available as both a desktop or cloud-based IDE.

CCS integrates MSPM0 device configuration and auto-code generation from SysConfig as well as MSPM0 code examples and academy trainings in the integrated TI Resource explorer. CCS offers an all-in-one development tool experience.

In addition to CCS, MSPM0 devices are also supported in industry-standard IDEs listed in [Table 2-2](#).

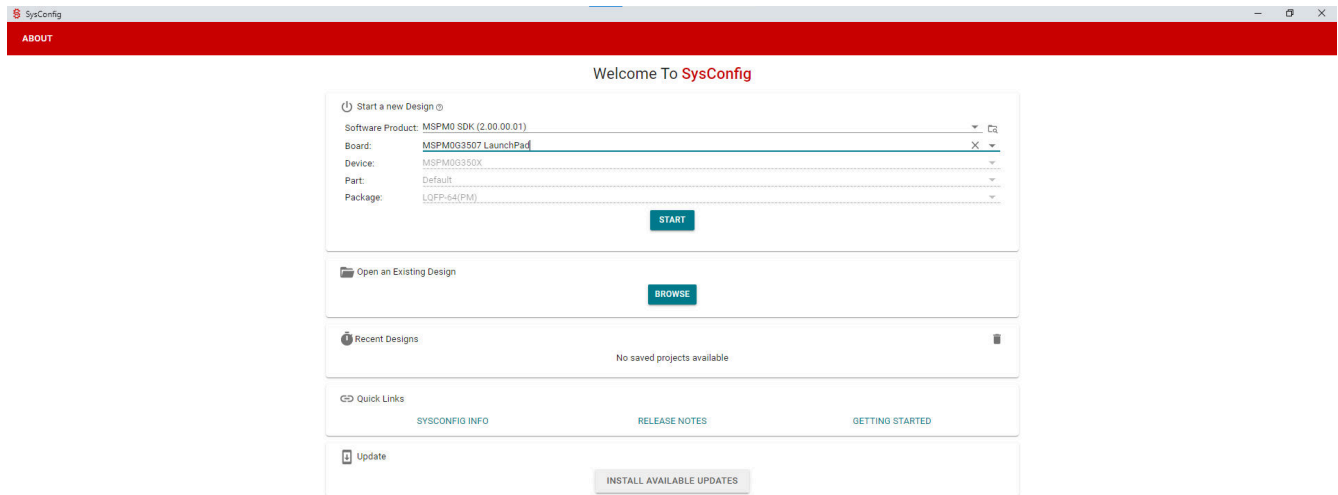
**Table 2-2. MSPM0 Supported IDEs**

IDE	MSPM0
CCS	✓
IAR	✓
Keil	✓

#### 2.1.3 MCUXpresso Code Configuration Tool vs SysConfig

SysConfig is an intuitive and comprehensive collection of graphical utilities for configuring pins, peripherals, radios, subsystems, and other components. It is TI's equivalent of NXP's MCUXpresso Code Configuration Tool. SysConfig helps manage, expose, and resolve conflicts visually so that you have more time to create differentiated applications. The tool's output includes C header and code files that can be used with MSPM0 SDK examples or used to configure custom software. SysConfig is integrated into CCS but can also be used as a standalone program.

For details, see the [MSPM0 SysConfig Guide](#).

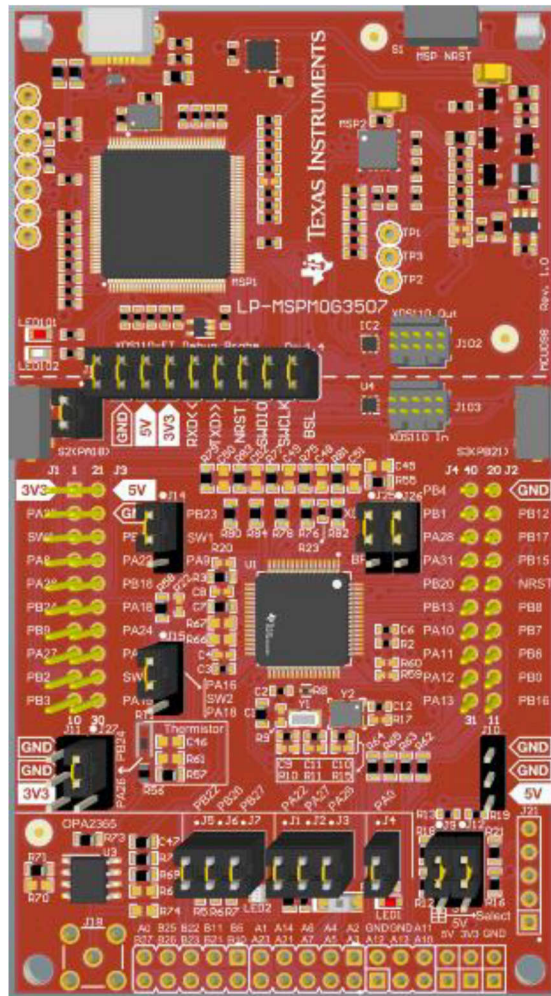


**Figure 2-1. MSPM0 SysConfig**

## 2.2 Hardware Ecosystem

LaunchPad development kits are the only evaluation modules for the MSPM0. LaunchPad kits are easy-to-use EVMs that contain everything needed to start developing on the MSPM0. This includes an onboard debug probe for programming, debugging, and measuring power consumption with EnergyTrace™ technology. MSPM0 LaunchPad kits also feature onboard buttons, LEDs, and temperature sensors among other circuitry. Rapid prototyping is simplified by the 40-pin BoosterPack™ plug-in module headers, which support a wide range of available BoosterPack plug-in modules. You can quickly add features like wireless connectivity, graphical displays, environmental sensing, and more.

- [LP-MSPM0G3507 LaunchPad development kit](#)
- [LP-MSPM0L1306 LaunchPad development kit](#)
- [LP-MSPM0C1104 Launchpad development kit](#)



**Figure 2-2. LP-MSPM0G3507 LaunchPad Development Kit**

### 2.3 Debug Tools

The debug subsystem (DEBUGSS) interfaces the serial wire debug (SWD) two-wire physical interface to multiple debug functions within the device. MSPM0 devices support debugging of processor execution, the device state, and the power state (using EnergyTrace technology). [Figure 2-3](#) shows the connection of the debugger.

MSPM0 support XDS110 and J-Link debugger for standard serial wire debug.

The Texas Instruments XDS110 is designed for TI embedded processors. XDS110 connects to the target board through a TI 20-pin connector (with multiple adapters for TI 14-pin and Arm 10-pin and Arm 20-pin) and to the host PC through USB2.0 High Speed (480 Mbps). It supports a wider variety of standards (IEEE1149.1, IEEE1149.7, SWD) in a single pod. All XDS debug probes support Core and System Trace in all Arm and DSP processors that feature an Embedded Trace Buffer (ETB). For details, see [XDS110 Debug Probe](#).

J-Link debug probes are the most popular choice for optimizing the debugging and flash programming experience. Benefit from record-breaking flash loaders, up to 3-MiB/s RAM download speed and the ability to set an unlimited number of breakpoints in the flash memory of MCUs. J-Link also supports a wide range of CPUs and architectures included Cortex M0+. For details, visit the [Segger J-Link Debug Probes page](#).

[Figure 2-3](#) shows a high-level diagram of the major functional areas and interfaces of the XDS110 probe to MSPM0 target.

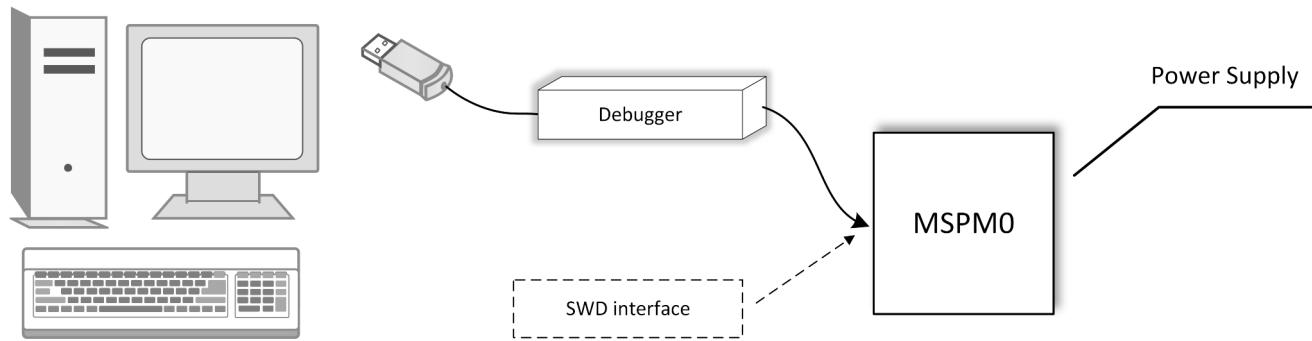


Figure 2-3. MSPM0 Debugging

## 2.4 Migration Process

The first step in migrating is to review the portfolio and choose the best MSPM0 MCU. After an MSPM0 MCU has been selected, choose a development kit. Development kits include a LaunchPad kit available for purchase and design files for a Target-Socket Board. TI also provides a free MSPM0 Software Development Kit (SDK), which is available as a component of [Code Composer Studio Theia IDE](#) desktop and cloud version within the TI Resource Explorer. Use the peripheral sections of this application note for help with porting software from NXP to MSPM0. Finally, once the software ported, download and debug the application with our debugging tools.

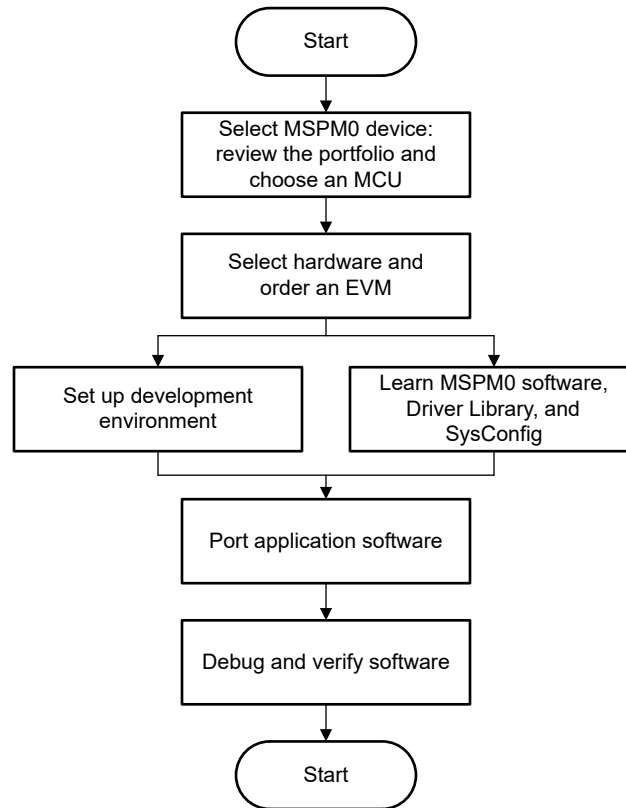


Figure 2-4. MSPM0 Migration Flowchart

## 2.5 Migration and Porting Example

To become more familiar with the TI ecosystem and explain how to best get started with MSPM0, this section describes the step-by-step migration process of a basic application.

To demonstrate the process of porting from NXP to MSPM0, this description includes the steps to port a basic low-power UART application from an NXP device to a MSPM0 device using an existing UART example as the starting point. This example starts with a UART example for an NXP KM35x device with a UART module.

1. Chose the right MSPM0 MCU.

The first step of migration is to choose the correct MSPM0 device for the application. To do this, the portfolio section of this guide can be used to choose a MSPM0 family. To narrow down to a specific device use the [product selection tool](#). When choosing a replacement for a NXP KM35x part, MSPM0 devices can match just about any functionality, so long as the correct replacement device is selected. It is important to ensure that the MSPM0 that you have selected has the peripheral set available for the code you want to migrate over. MSPM0 also offers many pin-to-pin scalable options, providing the ability to easily scale to larger or smaller memory devices without changing anything else in the system.

For purposes of this example, we have chosen the MSPM0G3507 as the best fit for his application.

2. Select hardware and order an EVM.

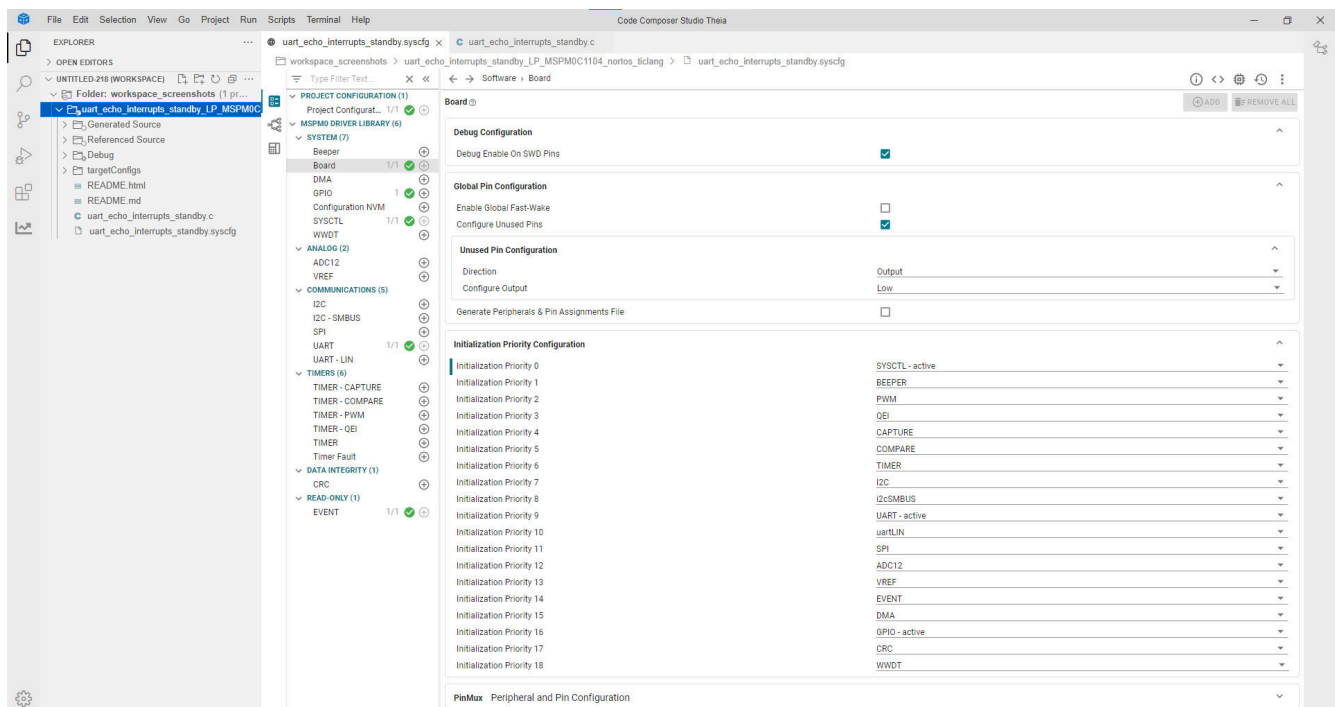
Using an evaluation module (EVM) can expedite the migration process. For the MSPM0 MCUs, a LaunchPad kit is the easiest hardware to begin on. LaunchPad kits are easy to use because they come with a built-in programmer and are designed to enable rapid development.

The MSPM0L1105 has a LaunchPad development kit ([LP-MSPM0G3507](#)) that can be used for porting the software.

3. Setup software IDE and SDK.

Before the software can be ported, a software development environment must be chosen and setup. [Figure 2-5](#) shows all of the IDEs supported by MSPM0. The migration and porting process is similar for any IDE that is chosen. The latest version of the [MSPM0 SDK](#) should be used.

For this example, TI's CCS-Theia is the chosen IDE.



**Figure 2-5. Code Composer Studio IDE**

4. Software porting.

When the environment is ready, start using the MSPM0 SDK. The MSPM0 SDK offers different layers for software development. For equivalents to NXP's device drivers, take a look at MSPM0's TI Drivers and Driverlib support. Most MSPM0 users find DriverLib level software is the best fit for their applications, so most MSPM0 software examples are also DriverLib based. This example uses DriverLib.

One option when porting a project is to try to replace each section of code with equivalent MSPM0 DriverLib APIs, but this is not generally the easiest path. Generally, it is best to first understand the application code being ported. Then start with the closest MSPM0 example project and modify it to match the original code functionality. This process is going to be shown below using a UART terminal example project from the MCUXpresso SDK\_2.x\_TWR-KM35Z75M, version 2.16.00.00 SDK. For more complex projects using many peripherals, this process is typically repeated for each peripheral.

- a. Understand the application.

The following description is from the NXP's SDK example "twrkm35z75m\_lpuart\_interrupt".

The lpuart\_functional\_interrupt example shows how to use lpuart driver functional API to receive data with interrupt method:  
In this example, one lpuart instance connect to PC, the board will send back all characters that PC send to the board.

The first step is to understand the main settings for the MCU. This is generally clock speeds and power policies. In this example, the device does not utilize a low power mode. The specified system clock frequency is 72MHz and is generated from the device's 'OSC' or System oscillator. The UART peripheral clock is derived from the OSC. It is divided down to allow the UART to operate at 15200 baud, 8 data bits, 1 start and stop bit, no parity. No hardware flow control is used.

- b. Find the closest MSPM0 example.

Next step is to understand any differences between the UART modules for KM35x and MSPM0 and then find the closest example in the MSPM0 SDK. This is easily accomplished by referring to the UART section in [Section 4](#). This section highlights differences between the UART modules and links to the UART-related MSPM0 SDK code examples. The closest example in the SDK for this example is probably [uart\\_echo\\_interrupts\\_standby](#) where the "UART RX/TX echos using interrupts while device is in STANDBY mode".

This MSPM0 example is similar, but not identical to the being ported. This example simply echoes the data received back on the device's TX pin. A small adjustment to the C code will allow us to match the original example.

- c. Import and modify the example.

Once a similar example is found, Open CCS and import the code example by going to Project > Import CCS Projects... and navigate it to the MSPM0 SDK example folder. Import the example. Here is the [uart\\_echo\\_interrupts\\_standby](#) example imported. This is a SysConfig project, so the main C file is simple. It first calls the SysConfig driverlib initialization which is a function autogenerated by SysConfig to configures the device. Then, it enables the UART interrupt. Finally, it goes to sleep waiting for any UART transaction. If it receives a UART transaction, it responds with "Hello World!".





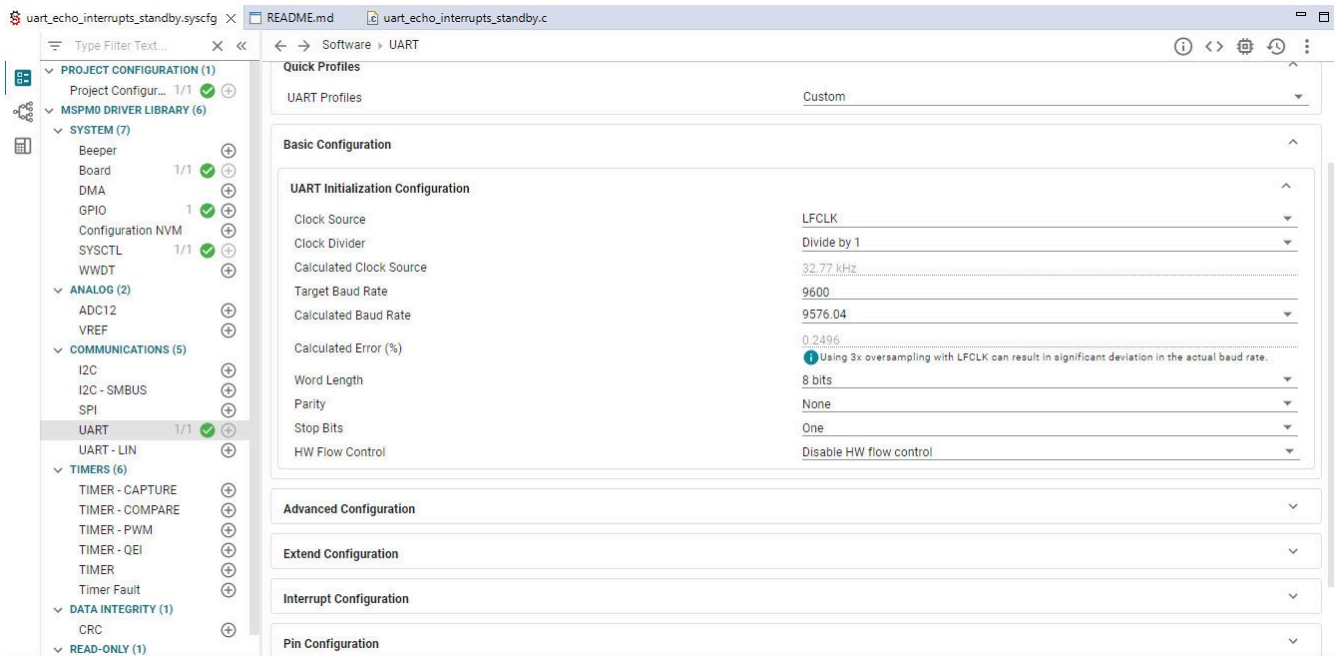
```

7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
  
```

**Figure 2-6. uart\_echo\_interrupts\_standby**

To see the SysConfig configuration, open the .syscfg file, which opens on the SYSCTL tab by default. For detailed guide on using SysConfig, see the [SysConfig Guide](#) in the in the MSPM0 SDK.

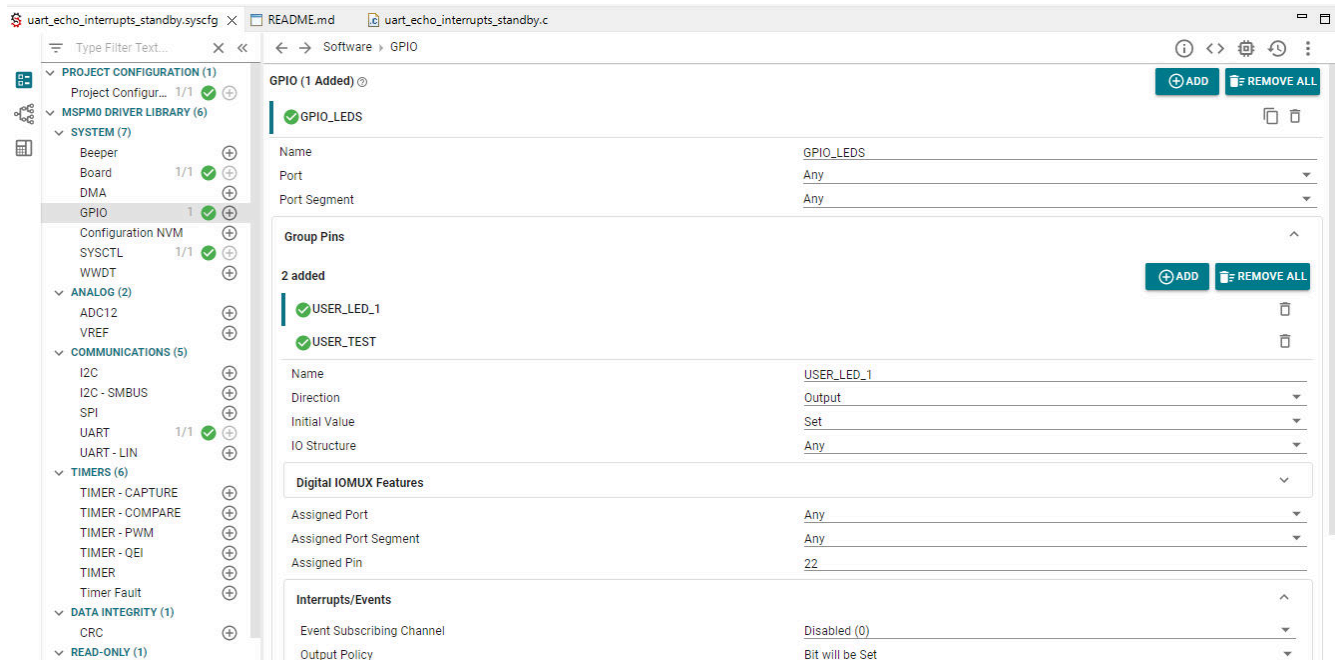
This example already has the UART peripheral set up, so there is no need to change any of the configurations found in this file. If desirable, it is possible to change the settings like the clock source, clock divider, the target baud rate, or others. For this migration demonstration, the configuration will be left the same.



Parameter	Value
<b>UART Initialization Configuration</b>	
Clock Source	LFCLK
Clock Divider	Divide by 1
Calculated Clock Source	32.77 kHz
Target Baud Rate	9600
Calculated Baud Rate	9576.04
Calculated Error (%)	0.2496
Word Length	8 bits
Parity	None
Stop Bits	One
HW Flow Control	Disable HW flow control
<b>Advanced Configuration</b>	
<b>Extend Configuration</b>	
<b>Interrupt Configuration</b>	
<b>Pin Configuration</b>	

**Figure 2-7. Sysconfig UART Tab**

This example also utilizes some GPIOs that are not featured in the NXP example. These GPIOs are simply used for debugging purposes, and one of them drives an LED. These can be left in for the sake of the demonstration, or removed if this is preferable.



**Figure 2-8. Sysconfig GPIO Tab**

When the project is saved and rebuilt, SysConfig updates the `ti_msp_dl_config.c` and `ti_msp_dl_config.h` files for the example. At this point, the example hardware configuration has been modified to match the full functionality of the original software being ported. The only remaining effort is application-level software to check for incoming UART bytes to toggle the LED and respond with "Hello World!". This is accomplished by editing a small amount of code in the `uart_echo_interrupts_standby.c` file.

```

uart_echo_interrupts_standby.syscfg  README.md  uart_echo_interrupts_standby.c X
26 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
27 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
28 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
29 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
30 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
31 */
32
33 #include "ti_msp_dl_config.h"
34
35 volatile uint8_t gEchoData = 0;
36 static uint8_t gMessage[12] = {'H', 'e', 'l', 'l', 'o', ' ', ' ', 'W', 'o', 'r', 'l', 'd', '!'};
37
38 int main(void)
39 {
40     SYSCFG_DL_init();
41
42     NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
43     NVIC_EnableIRQ(UART_0_INST_INT_IRQN);
44     DL_SYSCCTL_enableSleepOnExit();
45
46     while (1) {
47         __WFI();
48     }
49 }
50
51 void UART_0_INST_IRQHandler(void)
52 {
53     switch (DL_UART_Main_getPendingInterrupt(UART_0_INST)) {
54         case DL_UART_MAIN_IIDX_RX:
55             DL_GPIO_togglePins(GPIO_LEDS_PORT,
56                               GPIO_LEDS_USER_LED_1_PIN | GPIO_LEDS_USER_TEST_PIN);
57             gEchoData = DL_UART_Main_receiveData(UART_0_INST);
58             for(int i = 0; i < 12; i++){
59                 DL_UART_Main_transmitDataBlocking(UART_0_INST, gMessage[i]);
60             }
61             break;
62         default:
63             break;
64     }
65 }
66
  
```

**Figure 2-9. uart\_echo\_interrupts\_standby.c**

Two changes are made to the application code. First, the message array must be initialized so the device can respond to UART messages properly. For this, the following line is inserted below the initialization of gEchoData:

```
static uint8_t gMessage[12] = {'H', 'e', 'l', 'l', 'o', ' ', ' ', 'W', 'o', 'r', 'l', 'd', '!'};
```

The second step actually processes the data send using a for loop and a blocking version of the UART transmit function, so only one character is sent at a time and there is no data collision in the Tx buffer. This is accomplished by adding the below code to the UART RX ISR:

```
for(int i = 0; i < 12; i++){
    DL_UART_Main_transmitDataBlocking(UART_0_INST, gMessage[i]);
}
```

5. Debug and verify.

The following figures demonstrate the correct functionality of the code example. As shown in the first image, when a UART character is sent to the LP-MSPM0G3507 using a terminal program on a PC, the device responds with "Hello World!".

In the second image, logic analyzer captures show the device's RX and TX line, showing the incoming character, then the outgoing "Hello World!".

With each received character, the on-board LED will toggle on and off.

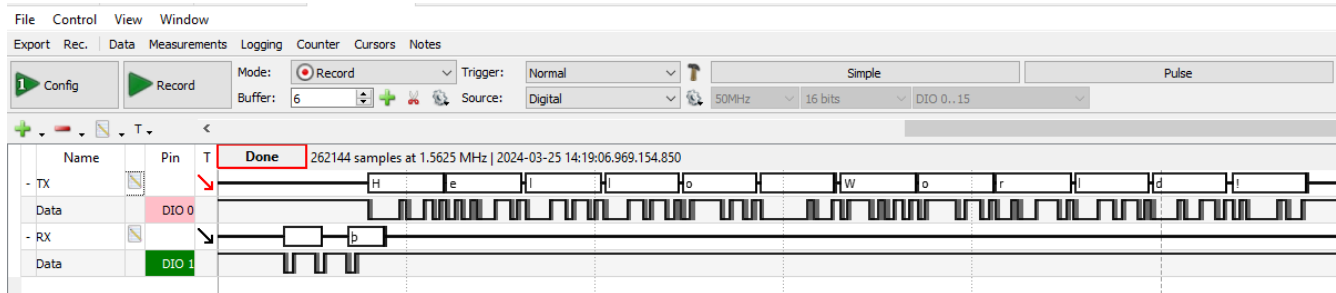


Figure 2-10. Logic Analyzer Capture

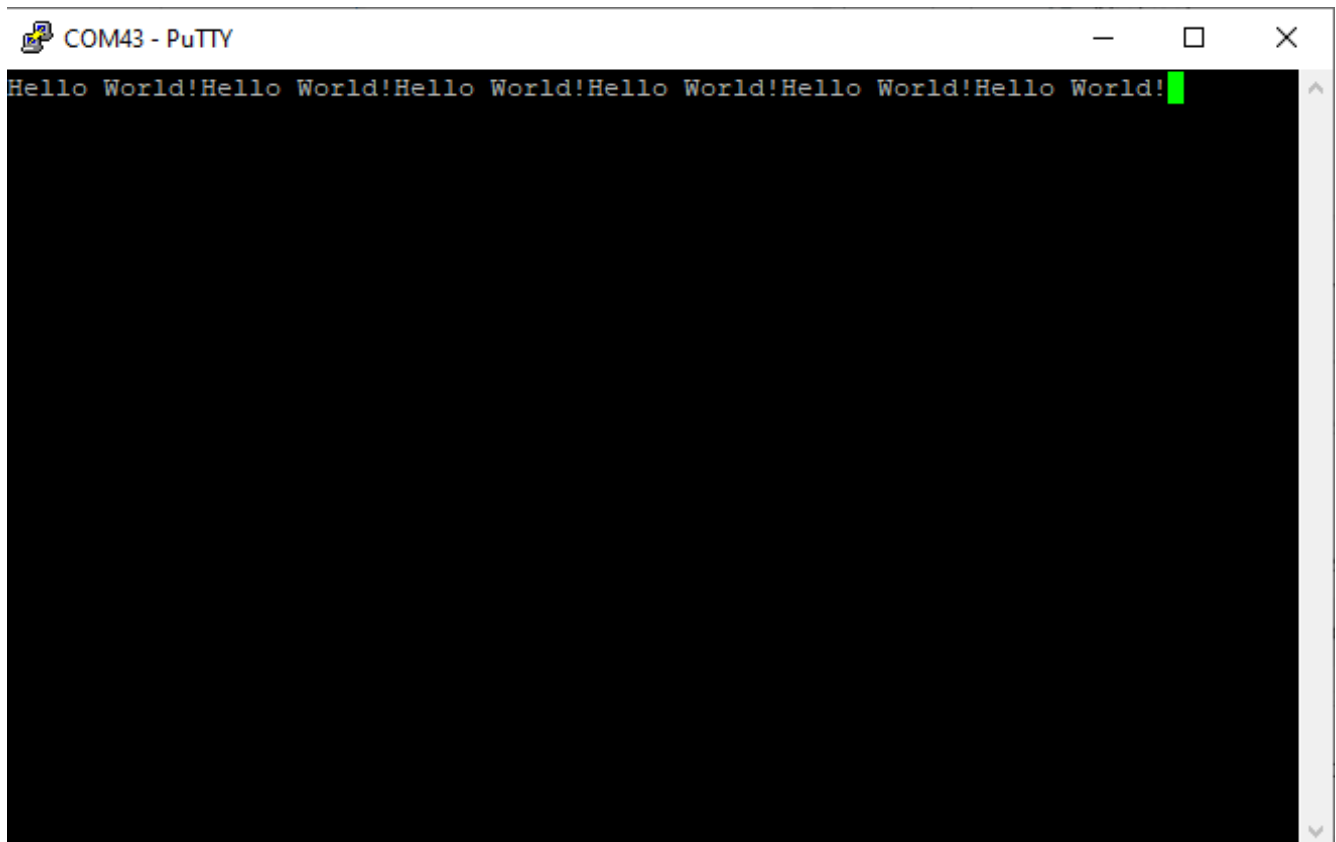


Figure 2-11. Serial Terminal

The software has successfully been ported! If this was just the first peripheral of many, continue to repeat this process and use SysConfig to combine each block.

## 3 Core Architecture Comparison

### 3.1 CPU

The NXP and [MSPM0 families](#) both utilize the Arm M0+ 32-bit core. [Table 3-1](#) provides a high-level overview of the general features of the CPUs in the MSPM0G, MSPM0L and MSPM0C devices compared to the NXP devices. [Section 3.6.1](#) provide a comparison of the interrupts and exceptions and how they are mapped in the Nested Vectored Interrupt Controller (NVIC) peripheral included in the M0 architecture for each device.

**Table 3-1. Comparison of CPU Feature Sets**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G	MSPM0L	MSPM0C
Architecture	Arm Cortex-M0+	Arm Cortex-M0+	Arm Cortex-M0+	Arm Cortex-M0+	Arm Cortex-M0+	Arm Cortex-M0+
Maximum MCLK	112MHz	20MHz	75MHz	32MHz - 80MHz	32MHz	24MHz
CPU instruction cache	Up to 4KB	No	64-bit	4 x 64-bit lines (32 bytes)	2 x 64-bit lines (16 bytes)	No
Processor trace capabilities	Yes, integrated micro trace buffer	No	Yes, integrated micro trace buffer	Yes, integrated micro trace buffer	No	No
Memory protection unit (MPU)	Yes	No	Yes	Yes	No	No
System timer (SYSTICK)	No	Yes	No	Yes (24 bit)	Yes (24 bit)	No
Hardware multiply	Yes	Yes	Yes	Yes	Yes	No
Hardware breakpoint / watchpoints	No	2/0	2/0	4/2	4/2	4/2
Boot routine	ROM	ROM	ROM	ROM	ROM	ROM
Bootstrap loader storage	Flash (system memory)	ROM	Flash (system memory)	ROM	ROM	Flash (system memory)
Bootloader interface support <sup>(1) (2)</sup>	Available for all data interfaces	Available for all data interfaces	Available for all data interfaces	UART, I2C, user extendable	UART, I2C, user extendable	User defined
DMA	Yes -16 ch	No	Yes - 4 ch	Yes - 7 ch	Yes - 3 ch	Yes - 1 ch

(1) For availability, see the device-specific data sheet.

(2) Other interfaces to be made available in later device releases.

### 3.2 Embedded Memory Comparison

#### 3.2.1 Flash Features

The MSPM0 and NXP family of MCUs feature nonvolatile Flash memory used for storing executable program code and application data.

**Table 3-2. Comparison of Flash Feature**

Features	S32K1xx	KEA128x	KM35	MSPM0G	MSPM0L	MSPM0C
Flash memory	Up to 2MB program/ 64KB FlexNVM data	64KB	Up to 512KB	128KB to 32KB	64KB to 8KB	16KB to 8KB
Memory organization	1 to 3 Blocks	Single Block		Single Block		
Block (bank) size	Up to 512KB	64KB	Up to 512KB	256KB		
Sector size	2KB	512B	2KB	1KB		
Flash word size	128 / 64 bits	64 bits		64 bits plus 8 ECC bits	64-bit, Refer to device-specific data sheet if ECC is supported	
Programming resolution	32-, 16-, or 8-bit (byte)		64-bit Flash word	64-bit Flash word, (32-, 16-, or 8-bit, see the device-specific Technical Reference Manual)		

**Table 3-2. Comparison of Flash Feature (continued)**

Features	S32K1xx	KEA128x	KM35	MSPM0G	MSPM0L	MSPM0C
<b>Erase</b>	Page Erase Chip erase (all banks)	N/A	Sector 2KB	Sector Erase 1KB / Bank Erase (up to 256KB)		
<b>Write protection</b>	Yes, static			Yes, static and dynamic		
<b>Read protection</b>	No	Yes	No	Yes		
<b>Flash memory read operations</b>	128-bit Flash word	128-bit Flash word	64-bit Flash word	64-bit Flash word size plus 8 ECC bits		
<b>Flash memory write operations</b>	Yes	Yes	Yes	64-bit Flash word size + 8 ECC bits	64-bit Flash word size, if ECC is supported, see the device-specific data sheet	
<b>FlexNVM (Data and EEPROM emulation)</b>	Read / Write 32-, 16-, or 8-bit	N/A	N/A	N/A		
<b>Error code correction (ECC)</b>	Yes	Yes	No	8 bits for 64 bits	Refer to device specific datasheet if ECC is supported	
<b>Prefetch</b>	Yes	No	No	Yes		No

In addition to the Flash memory features listed in the previous table, the MSPM0 Flash memory also has the following features:

- In-circuit program and erase supported across the entire supply voltage range
- Internal programming voltage generation
- Support for EEPROM emulation with up to 100 000 program/erase cycles on the lower 32KB of the Flash memory, with up to 10 000 program/erase cycles on the remaining Flash memory (devices with 32KB support 100 000 cycles on the entire Flash memory)

### 3.2.2 Flash Organization

The Flash memory is used for storing application code and data, the device boot configuration, and parameters that are preprogrammed by TI from the factory. The Flash memory is organized into one or more banks, and the memory in each bank is further mapped into one or more logical memory regions and assigned system address space for use by the application.

#### 3.2.2.1 Memory Banks

Most MSPM0 devices implement a single flash bank (BANK0). On devices with a single flash bank, an ongoing program/erase operation stalls all read requests to the flash memory until the operation has completed and the flash controller has released control of the bank. On devices with more than one flash bank, a program/erase operation on a bank also stalls read requests issued to the bank that is executing the program/erase operation but does not stall read requests issued to another bank. Therefore, the presence of multiple banks enables application cases such as:

- Dual-image firmware updates (an application can execute code out of one flash bank while a second image is programmed to a second symmetrical flash bank without stalling the application execution)
- EEPROM emulation (an application can execute code out of one flash bank while a second flash bank is used for writing data without stalling the application execution)

#### 3.2.2.2 Flash Memory Regions

The memory within each bank is mapped to one or more logical regions based upon the functions that the memory in each bank supports. There are four regions:

- **FACTORY** – Device Id and other parameters
- **NONMAIN** – Device boot configuration (BCR and BSL)
- **MAIN** – Application code and data
- **DATA** – Data or EEPROM emulation

Devices with one bank implement the FACTORY, NONMAIN, and MAIN regions on BANK0 (the only bank present), and the data region is not available. Devices with multiple banks also implement FACTORY,

NONMAIN, and MAIN regions on BANK0, but include additional banks (BANK1 through BANK4) that can implement MAIN or DATA regions.

### 3.2.2.3 NONMAIN Memory

The NONMAIN is a dedicated region of flash memory that stores the configuration data used by the BCR and BSL to boot the device. The region is not used for any other purpose. The BCR and BSL both have configuration policies that can be left at their default values (as is typical during development and evaluation) or modified for specific purposes (as is typical during production programming) by altering the values programmed into the NONMAIN flash region.

## 3.3 Power Up and Reset Summary and Comparison

Similar to NXP M0 devices, MSPM0 devices have a minimum operating voltage and have modules in place to make sure that the device starts up properly by holding the device or portions of the device in a reset state. [Table 3-3](#) shows a comparison on how this is done between the two families and what modules control the power up process and reset across the families.

**Table 3-3. Comparison of Power Up**

Feature	S32K1xx	KEA128x	KM35x	MSPM0 Devices	
Modules governing power up and resets	Reset Control Module	Reset Control Module	Reset Control Module	Module governing power up and resets	PMCU (Power Management and Clock Unit)
Voltage-Level Based Resets					
POR (Power-On Reset)	Complete device reset. First level voltage release for power up. Lowest voltage level for power down			POR (Power-On Reset)	Complete device reset. First level voltage release for power up. Lowest voltage level for power down.
BOR (Brownout Reset) with configurable levels	Programmable Threshold for triggering resets			Configurable BOR (Brownout Reset)	Can be configured as a reset or interrupt, with different voltage thresholds.

NXP defines different reset types, while MSPM0 devices have different levels of reset states. For MSPM0 devices, the reset levels have a set order, and when a level is triggered, all subsequent levels are reset until the device is released into RUN mode. [Table 3-4](#) gives a brief description of MSPM0 reset states. [Figure 3-1](#) shows the relationship between all of the MSPM0 reset states.

**Table 3-4. Comparison of Reset Domains**

S32K1xx, KEA128x, KM35 Reset Domains		MSPM0 Reset States <sup>(1)</sup>	
Power reset domain	Typical triggers are POR, BOR <sup>(3)</sup>	POR	existing content
No equivalent		Boot reset (BOOTRST) <sup>(2)</sup>	existing content
System reset domain	Typical triggers are external pin reset, low voltage detect, WDT reset <sup>(3)</sup>	System reset (SYSRST)	existing content
No equivalent		CPU-only reset (CPURST)	existing content
RTC domain	<sup>(3)</sup>		existing content

- (1) Not all reset triggers are described. Refer to the PMCU chapter of the device TRM for all available reset triggers.
- (2) If BOOTRST cause was through NRST or software trigger, RTC, LFCLK, and LFXTL/LFLCK\_IN configurations and IOMUX settings are NOT reset to allow RTC to maintain operation through external reset.
- (3) For specific reset triggers, see the Reset and Boot chapter of the NXP device-specific TRM.

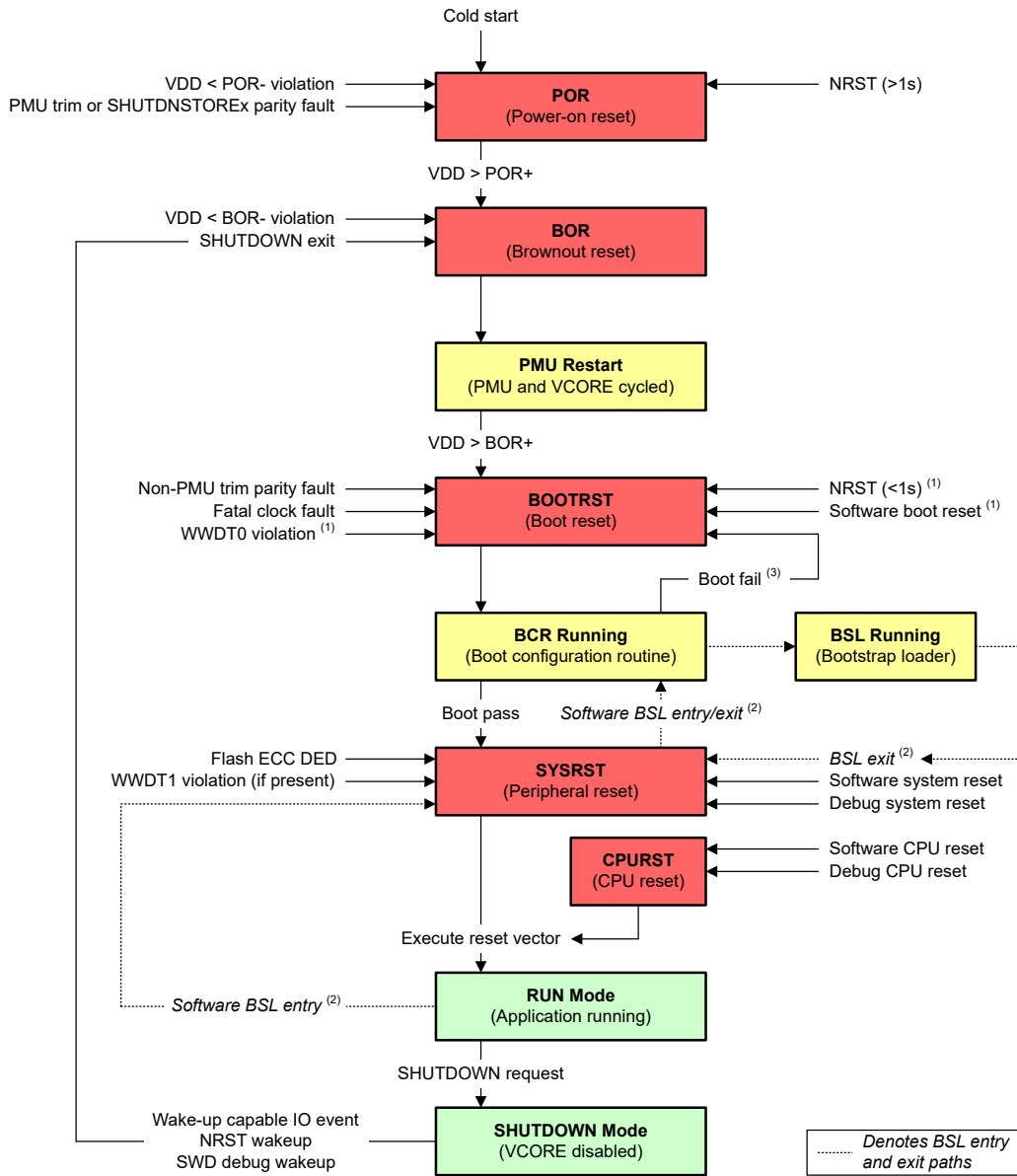


Figure 3-1. MSPM0 Reset Levels



### 3.4 Clocks Summary and Comparison

NXP's MCUs and MSPM0 both contain internal oscillators that source primary clocks. The clocks can be divided to source other clocks and be distributed across the multitude of peripherals.

**Table 3-5. Oscillator Comparisons**

Oscillator	S32K1xx	KEA128x	KM35x	MSPM0G/L	MSPM0C
Internal RC	Fast IRC, Slow IRC	N/A	Internal RC (4MHz)	SYSOSC <sup>(1)</sup>	SYSOSC (24MHz)
Full Swing Crystal	SOSC	OSCCLK		HFXT	N/A
Internal RC	LP0 128kHz	IRC 32kHz		LFOSC 32kHz	
Low Frequency Crystal	N/A	N/A	OSC32K	LFXT - 32kHz	N/A
Low Power Crystal	N/A	1kHz LPO	N/A	LFXT - 32kHz	N/A

(1) SYSOSC is programmable to be 32MHz, 24MHz, 16MHz, or 4MHz.

**Table 3-6. Clock Comparison**

Clock	S32K1xx	KEA128x	KM35	MSPM0G	MSPM0L/C
System	SOSC	ICSOUTCLK	MCGOUTCLK	SYSOSC (4-32MHz)	SYSOSC (24MHz)
	SPLLDIV1	N/A	MCGPLLCLK	SYSPLLCLK1	N/A
	SPLLDIV2	N/A	MCGFLLCLK	SYSPLLCLK0	N/A
		N/A		SYSPLLCLK2x1	N/A
Core / Bus Clock	CORE_CLK	ICSOUTCLK	MCGOUTCLK	BUSCLK/ULPCLK <sup>2</sup>	BUSCLK/ULPCLK <sup>2</sup>
Slow Internal Clock	SIRC (8MHz)	IRC (37.5kHz)	IRC (32kHz)	LFOSC (32kHz)	
Fast Internal Clock	FIRC (48MHz)	N/A	IRC (4MHz)	SYSOSC	
Low Power Clock	LPO (128 kHz)	LP0CLK	LPO (1kHz)	LFCLK (32kHz)	
RTC Clock	N/A	OSCERCLK	IRTC/OSC32KCLK	RTCCLK	N/A

1. SYSPLLCLK2x is twice the speed of the output of the PLL module and can be divided down.
2. BUSCLK depends on the Power Domain. For Power Domain 0, BUSCLK is ULPCLK. For Power Domain 1, BUSCLK is MCLK.

**Table 3-7. Peripheral Clock Sources**

Peripheral	S32K1xx	KEA128x	KM35x	MSPM0G	MSPM0L/C
RTC	LP01K_CLK, RTC_CLK	BUS_CLK,LPOCLK, CSIRCLK,OSCERCLK	EXTAL32	LFCLK (LFOSC, LFXT)	N/A
UART	BUSCLK,SOSCDIV2_CLK, FIRCDIV2_CLK,SPLLDIV2_CLK	BUSCLK	BUSCLK	BUSCLK, ULPCLK,MFCLK, LFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK
SPI		BUSCLK	BUSCLK	BUSCLK, MFCLK, LFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK
I2C	BUSCLK,SOSCDIV2_CLK, FIRCDIV2_CLK,SPLLDIV2_CLK	BUSCLK	BUSCLK	BUSCLK, MFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK
ADC		BUSCLK,OSCERCLK, ADACK	OSCERCLK,MCGPLLCLK	ULPCLK, HFCLK, SYSOSC	SYSOSC, ULPCLK
LPTIM 1/2		TIMER_CLK	LPO, OSCERCLK,MCGIRCLK, ERCLK32K	LFCLK, ULPCLK, LFCLK_IN	ULPCLK, LFCLK
TIMERS	SYS_CLK,SOSCDIV1_CLK, FIRCDIV1_CLK,SPLLDIV1_CLK	TIMER_CLK,ICSFFCLK	BUSCLK	BUSCLK, MFCLK, LFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK

The device-specific TRM for each family has a clock tree to help visualize the clock system. Sysconfig can assist with the options for clock division and sourcing for peripherals.

### 3.5 MSPM0 Operating Modes Summary and Comparison

MSPM0 MCUs provide five main operating modes (power modes) to allow for optimization of the device power consumption based on application requirements. In order of decreasing power, the modes are: RUN, SLEEP, STOP, STANDBY, and SHUTDOWN. The CPU is active executing code in RUN mode. Peripheral interrupt events can wake the device from SLEEP, STOP, or STANDBY mode to the RUN mode. SHUTDOWN mode completely disables the internal core regulator to minimize power consumption, and wake is only possible via NRST, SWD, or a logic level match on certain IOs. RUN, SLEEP, STOP, and STANDBY modes also include several configurable policy options (for example, RUN.x) for balancing performance with power consumption.

To further balance performance and power consumption, MSPM0 devices implement two power domains: PD1 (for the CPU, memories, and high performance peripherals), and PD0 (for low speed, low power peripherals). PD1 is always powered in RUN and SLEEP modes, but is disabled in all other modes. PD0 is always powered in RUN, SLEEP, STOP, and STANDBY modes. PD1 and PD0 are both disabled in SHUTDOWN mode.

Low-power mode code examples

Navigate to the SDK installation and find low-power mode code examples in examples > nortos > LP name > driverlib

#### 3.5.1 Operating Modes Comparison

NXP devices have similar operating modes. [Table 3-8](#) gives a brief comparison between NXP and MSPM0 devices.

**Table 3-8. Operating Modes Comparison Between NXP and MSPM0 Devices**

S32K1XX Series		KEA128x		KM35x		MSPM0		
Mode	Description	Mode	Description	Mode	Description	Mode	Description	
Run/High Speed Run	Full clocking and peripherals available	Run	Full clocking and peripherals available	Run	Full clocking and peripherals available	Run	0	Full clocking and peripherals available
							1	SYSOSC at set frequency; CPUCLK and MCLK limit to 32kHz
							2	SYSOSC disabled; CPUCLK and MCLK limit to 32kHz
VLPR	Reduced clock speed; Most peripherals remain enabled	N/A	N/A	N/A	N/A	N/A		
VLPS	CPU stopped; Some clocks and peripherals remain enabled	NORMAL WAIT	CPU stopped; peripherals individually enabled	NORMAL WAIT	CPU stopped; peripherals individually enabled	Sleep	0	CPU not clocked
							1	Same as Run1, but CPU not clocked
							2	Same as Run2, but CPU not clocked
STOP	CPU and system clocks stopped; Some peripherals remain enabled	NORMAL STOP	CPU and system clocks stopped; Some peripherals remain enabled	NORMAL STOP	CPU and system clocks stopped; Some peripherals remain enabled	Stop	0	Sleep 0 + PD1 disabled
							1	Sleep 1 + SYSOSC gear shifted to 4MHz
							2	Sleep 2 + ULPClk limited to 32kHz

**Table 3-8. Operating Modes Comparison Between NXP and MSPM0 Devices (continued)**

S32K1XX Series		KEA128x		KM35x		MSPM0	
Mode	Description	Mode	Description	Mode	Description	Mode	Description
						Standby	0 Lowest power with BOR capability; all PD0 peripherals can receive ULPClk and LFCLK at 32kHz; RTC available with RTCCLK
							1 Only TIMG0 and TIMG1 can receive ULPClk or LFCLK at 32kHz; RTC available with RTCCLK
						Shutdown	No clocks, BOR, or RTC. Core regulation off. PD1 And PD0 disabled. Exit triggers reset level BOR.

### 3.5.2 MSPM0 Capabilities in Lower Power Modes

As seen in [Table 3-8](#), MSPM0 peripherals or peripheral modes can be limited in availability or operating speed in lower power operating modes. For specific details, see the Supported Functionality by Operating Mode table found in the MSPM0 device-specific data sheet, for example:

[MSPM0G350x Mixed-Signal Microcontrollers data sheet](#)

[MSPM0L134x, MSPM0L130x Mixed-Signal Microcontrollers data sheet](#)

[MSPM0C110x Mixed-Signal Microcontrollers data sheet](#)

An additional capability of the MSPM0 devices is the ability for some peripherals to perform an Asynchronous Fast Clock Request. This allows MSPM0 device to be in a lower power mode where a peripheral is not active, but allows a peripheral to be triggered or activated. When an Asynchronous Fast Clock Request happens, the MSPM0 device has the ability to quickly ramp up an internal oscillator to a higher speed or temporarily go into a higher operating mode to process the impending action. This allows for fast wake up of the CPU from timers, comparator, general-purpose input/output (GPIO), and RTC; receive SPI, UART, and I2C; or trigger DMA transfers and ADC conversions, while sleeping in the lowest power modes. For details on implementation of Asynchronous Clock Requests as well as peripheral support and purpose, see the appropriate chapter in the device-specific MSPM0 TRMs.

[MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual](#)

[MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual](#)

[MSPM0 C-Series 24-MHz Microcontrollers Technical Reference Manual](#)

### 3.5.3 Entering Lower-Power Modes

Like the NXP devices, the MSPM0 devices go into a lower-power mode when executing the wait for event, `__WFE()`; , or wait for interrupt, `__WFI()`; , instruction. The low-power mode is determined by the current power policy settings. The device power policy is set by a driver library function. The following function call sets that power policy to Standby 0.

```
DL_SYSCTL_setPowerPolicySTANDBY0();
```

STANDBY0 can be replaced with the operating mode of choice. For a full list of driverlib APIs that govern power policy, see this section of the [MSPM0 SDK DriverLib API guide](#). Also see the following code examples that demonstrate entering different operating modes. Similar examples are available for every MSPM0 device.

## 3.6 Interrupt and Events Comparison

### 3.6.1 Interrupts and Exceptions

The MSPM0 and NXP MCUs both register and map interrupt and exception vectors depending on the device's available peripherals. A summary and comparison of the interrupt vectors for each family of devices is included in [Table 3-9](#). A lower value of priority for an interrupt or exception is given higher precedence over interrupts with a higher priority value. For some of these vectors the priority is user-selectable, and for others it is fixed.

For both MSPM0 and NXP MCUs, exceptions such as NMI, reset, and hard fault handlers are given negative priority values to indicate that they always have the highest precedence over peripheral interrupts. For peripherals with selectable interrupt priorities, up to four programmable priority levels are available on both families of devices.

**Table 3-9. Interrupt Comparison**

IRQ Number	NXP		MSPM0	
	Interrupt/Exception	Priority	Interrupt/Exception	Priority
-	Reset		Re	Fixed: -3
-	NMI Handler		NMI Handler	Fixed: -2
-	Hard Fault Handler		Hard Fault Handler	Fixed: -1
-	SVCALL Handler		SVCALL Handler	Selectable
-	PendSV		PendSV	Selectable
-	SysTick		SysTick	Selectable
0	DMA0: KM35x	Selectable	INT_GROUP0-6 [WWDT0, DEBUGSS, FLASHCTL, WUC FSUBx, and SYSCTL]: M0G	Selectable
			INT_GROUP0,2-6 [WWDT0, DEBUGSS, FLASHCTL, WUC FSUBx, and SYSCTL]: M0L	
			INT_GROUP0-6 [WWDT0, DEBUGSS, FLASHCTL, and SYSCTL]: M0C	
1	DMA1: KM35x	Selectable	INT_GROUP0-5 [GPIO0, GPIO1, COMP0, COMP1, COMP2, TRNG]: M0G	Selectable
			INT_GROUP0,2 [GPIO0, COMP0]: M0L	
			GPIO0: M0C	
2	DMA2: KM35x	Selectable	TIMG8: M0G, M0C	Selectable
			TIMG1: M0L	
3	DM3: KM35x	Selectable	UART3: M0G	Selectable
4	SPI0/SPI1/SPI2: KM35x	Selectable	ADC0: M0G	Selectable
			ADC: M0L, M0C	
5	FTMRE: KEA128x	Selectable	ADC1: M0G	Selectable
	PDB0: KM35x			
6	PMC: KE128x/KM35x	Selectable	CANFD0: M0G	Selectable
7	TMR0: KM35x	Selectable	DAC0: M0G	Selectable
	EXT IRQ: KE128x			
8	TMR1: KM35x	Selectable	Reserved	Selectable
	I2C0: KEA128x			

**Table 3-9. Interrupt Comparison (continued)**

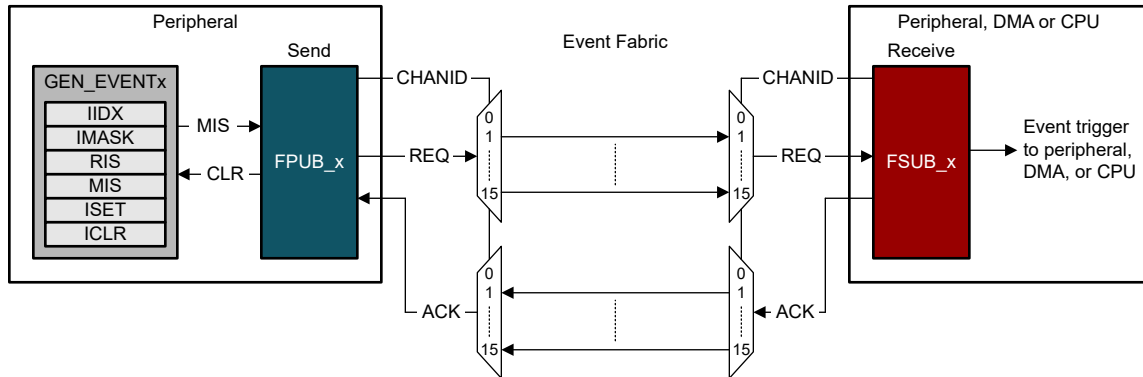
IRQ Number	NXP		MSPM0	
	Interrupt/Exception	Priority	Interrupt/Exception	Priority
9	TMR2: KM35x	Selectable	SPI0: M0G, M0L, M0C	Selectable
	I2C1: KEA128x			
10	TMR3: KM35x	Selectable	SPI1: M0G, M0L, M0C	Selectable
	SPI0: KEA128x			
11	PIT0/PIT1: KM35x	Selectable	Reserved	Selectable
	SPI1: KEA128x			
12	LLWU: KM35x	Selectable	Reserved	Selectable
	UART0: KEA128x			
13	FLASH: KM35x	Selectable	UART1: M0G, M0L	Selectable
	UART1: KEA128x			
14	ACMP0/ACMP1/ACMP2: KM35x	Selectable	UART2	Selectable
	UART2: KEA128x			
15	SLCD: KM35x	Selectable	UART0: M0G, M0L, M0C	Selectable
	ADC0: KEA128x			
16	ADC: KM35x	Selectable	TIMG0: M0G, M0L	Selectable
	ACMP0: KEA128x		TIMG14: M0C	
17	PTx: KM35x	Selectable	TIMG6: M0G	Selectable
	FTM0: KEA128x			
18	RNGA: KM35x	Selectable	TIMA0: M0G	Selectable
	FTM1: KEA128x		TIMG2: M0L	
			TIMA0: M0C	
19	UARTx: KM35x	Selectable	TIMA1: M0G	Selectable
	FTM2: KEA128x			
20	MMAU: KM35x	Selectable	TIMG7: M0G	Selectable
	RTC: KEA128x		TIMG4: M0L	
21	AFE_CH0: KM35x	Selectable	TIMG12: M0G	Selectable
	ACMP1: KEA128x			
22	AFE_CH1: KM35x	Selectable	Reserved	Selectable
	PIT_CH0: KEA128x			
23	AFE_CH2: KM35x	Selectable	Reserved	Selectable
	PIT_CH1: KEA128x			
24	AFE_CH3: KM35x	Selectable	I2C0: M0G, M0L, M0C	Selectable
	KBIO: KEA128x			
25	iRTC: KM35x	Selectable	I2C1: M0G, M0L, M0C	Selectable
	KBI1: KEA128x			
26	I2C0/I2C1: KM35x	Selectable	Reserved	Selectable
	Reserved: KEA128x			
27	LPUART0: KM35x	Selectable	Reserved	Selectable
	ICS: KEA128x			
28	MCG: KM35x	Selectable	AES: M0G	Selectable
	WDOG: KEA128x			
29	WDOG/EWM: KM35x	Selectable	Reserved	Selectable
	PWT: KEA128x			
30	LPTMR0/LPTRM1: KM35x	Selectable	RTC: M0G	Selectable
	MSCAN_RX: KEA128x			

**Table 3-9. Interrupt Comparison (continued)**

IRQ Number	NXP		MSPM0	
	Interrupt/Exception	Priority	Interrupt/Exception	Priority
31	XBAR: KM35x	Selectable	DMA : M0G, M0L, M0C	Selectable
	MSCAN_TX: KEA128x			

**3.6.2 Event Handler and Extended Interrupt and Event Controller (EXTI)**

The MSPM0 devices include a dedicated event manager peripheral, which extends the concept of the NVIC to allow digital events from a peripheral to be transferred to the CPU as interrupts, to the DMA as a trigger, or to another peripheral to trigger a hardware action. The event manager can also perform handshaking with the power management and clock unit (PMCU), to make sure that the necessary clock and power domain are present for triggered event actions to take place.



**Figure 3-2. Generic Event Route**

In the MSPM0 event manager, the peripheral that generates the event is known as a publisher, and the peripheral, DMA, or CPU that acts based on the publisher is known as the subscriber. The potential combinations of available publisher and subscriber are extremely flexible and can be used when migrating software to replace functionality previously handled by interrupt vectors and the CPU, to bypass the CPU entirely. For example, an I2C-to-UART bridge may previously have triggered a UART transmission upon receipt of an I2C STOP, using an ISR to set a flag, or load the UART TX buffer directly. With the MSPM0 Event handler, the I2C transaction complete event could trigger the DMA to load the UART TX buffer directly, and therefore eliminate the need for any action by the CPU.

To get more details on the use of the event handler in MSPM0, see the Events section of the [MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual](#) or the [MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual](#).

**3.7 Debug and Programming Comparison**

The Arm SWD 2-wire JTAG port is the main debug and programming interface for both MSPM0 and NXP . This interface is typically used during application development, and during production programming. [Table 3-10](#) compares the features between the two device families. For additional information about security features of the MSPM0 debug interface, see the [Cybersecurity Enablers in MSPM0 MCUs](#).

**Table 3-10. Program/Debug Interface Feature Comparison**

	S32K1xx	KEA128x	KM35x	MSPM0
<b>Debug port</b>	Arm SWD port (2-wire)			Arm SWD port (2-wire)
<b>Break Point Unit (BPU)</b>	Two hardware breakpoints; unlimited user software breakpoints			Four hardware breakpoints
<b>Data Watch Unit (DWT)</b>	Four watchpoints	Two watchpoints		Two watchpoints
<b>Micro-Trace Buffer (MTB)</b>	Yes	No	Yes	MTB support with 4 trace packets#none#
<b>Low-power debug support</b>	No	N/A	No	Yes
<b>EnergyTrace support</b>	N/A	N/A	N/A	EnergyTrace+ support (CPU states with power profiling)
<b>Peripheral run support during debug</b>	Yes	N/A	Yes	Yes
<b>Debug interface locking</b>	Yes	N/A	Yes	Can permanently disable debug capabilities, or can lock with password

### 3.7.1 Bootstrap Loader (BSL) Programming Options

The bootstrap loader (BSL) programming interface is an alternative programming interface to the Arm SWD. This interface offers programming capabilities only, and typically is utilized through a standard embedded communication interface. This allows for firmware updates through existing connections to other embedded devices in system or external ports. Although programming updates is the main purpose of this interface, it can also be utilized for initial production programming as well. [Table 3-11](#) shows a comparison of the different options and features between MSPM0 and NXP device families.

**Table 3-11. BSL Feature Comparison**

BSL Features	S32K1xx	KEA128x	KM35x	MSPM0
<b>BSL started on blank device</b>	No	N/A		Yes
<b>Auto detection of programming interface</b>	No	N/A		Yes
<b>Security</b>	Yes	N/A		Secure boot options; CRC protections
<b>Customizable</b>	No	N/A		Yes, configurable invoke pin and plug-in feature
<b>Invoke methods</b>	Jump/call instruction	N/A		1 pin high at BOOTRST, SW entry
<b>Interfaces Supported</b>				
<b>UART</b>	Yes	N/A		Yes
<b>I2C</b>	Yes	N/A		Yes
<b>SPI</b>	Yes <sup>(1)</sup>	N/A		Custom plug-in needed
<b>CAN</b>	Yes <sup>(1)</sup>	N/A		Plug-in planned <sup>(1)</sup>

(1) Pattern option availability is device dependent.



## 4 Digital Peripheral Comparison

### 4.1 General-Purpose I/O (GPIO, IOMUX)

MSPM0 GPIO functionality covers all of the features offered by the S32K1xx, KEA128x devices, and KM35x with additional functionality. NXP uses the term GPIO to refer to the pins that can be set, reset, or toggle. However, MSPM0 uses a slightly different nomenclature, namely:

- MSPM0 GPIO refers to the hardware capable of reading and writing IO, generating interrupts, and so forth.
- MSPM0 IOMUX refers to the hardware responsible for connecting different internal digital peripherals to a pin. IOMUX services many different digital peripherals, including, but not limited to, GPIO.

The MSPM0 GPIO and IOMUX modules cover the same functionality as NXP's GPIO, PORT, and TRGMUX/SMU modules. MSPM0 devices also offer several additional functions that are unavailable for NXP S32K1xx, KEA128x, and KM35x devices.

**Table 4-1. GPIO Feature Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G, MSPM0L,MSPM0C
<b>Output modes</b>	Push-pull Open drain with pullup or pulldown	Push-pull Open drain with pullup	Push-pull Open drain with pullup or pulldown	Push-pull Open drain with pullup or pulldown
<b>GPIO speed selection</b>	Synchronous path 1.5*tmax = 31.25ns @48MHz Asynchronous path: 50ns	10.2ns rise time, 9.5ns fall time	8.0ns rise time, 5.0ns fall time	ODIO pins: 120ns All others: 0.3*fmax = 3.75ns @ 80MHz
<b>High-drive GPIO</b>	12mA Per High drive port pin	20mA	Yes	Equivalent, called High Drive IO (HDIO)
<b>Input modes</b>	Floating Pull-up/Pull-down Analog	Floating Pull-up Analog	Floating Pull-up/Pull-down Analog	Equivalent
<b>Atomic bit set and reset</b>	Yes	Yes	Yes	Equivalent
<b>Alternate functions</b>	Configured with Signal Multiplexing Unit	Configured with PORT module	Configured with PORT module	Equivalent MSPM0 uses IOMUX
<b>Wake-up</b>	GPIO pin interrupt	N/A	Equivalent	Equivalent
<b>GPIO controlled by DMA</b>	Yes	No	Yes	Yes
<b>User controlled input filtering to reject glitches less than 1, 3, or 8 ULPCLK periods</b>	Equivalent	Equivalent	Equivalent	Yes
<b>User controllable input hysteresis</b>	Yes	Yes	No	Yes

GPIO code examples

Information about GPIO code examples can be found in the [MSPM0 SDK examples guide](#).

### 4.2 Universal Asynchronous Receiver-Transmitter (UART)

MSPM0 and NXP's M0 MCUs offer peripherals for asynchronous (clockless) communication. The S32K1xx and KM35x family offers a low-powered UART that supports basic UART, while the KEA128x and KM35x offer a regular UART module. In MSPM0, these UART peripherals come in two variants: one with standard features and one with advanced features. In NXP's devices, the UART comes in one singular version. [Table 4-2](#) shows a comparison of MSPM0's UART against NXP's S32K1xx, KEA128x, and KM35x UART.

**Table 4-2. UART Feature Set Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0
<b>Hardware flow control</b>	Yes	No	Yes	Yes
<b>Continuous communication using DMA</b>	Yes	No	Yes	Yes
<b>Multiprocessor</b>	No	No	No	Yes

**Table 4-2. UART Feature Set Comparison (continued)**

Feature	S32K1xx	KEA128x	KM35x	MSPM0
Synchronous mode	No	No	No	No
Single-wire half duplex communication	Yes	Yes	Yes	Yes <sup>(1)</sup>
Wakeup from low-power mode	Yes	Yes	Yes	Yes
Data length	7,8,9,10	8,9	8,9	5, 6, 7, 8
Tx/Rx FIFO Depth	4	N/A	8	4
IrDA Support	Yes	No	Yes	Extended UART only
LIN Support	Yes	Yes	Yes	Extended UART only
DALI Support	No	No	No	Extended UART only
Manchester Code Support	No	No	No	Extended UART only

(1) Requires reconfiguration of the peripheral between transmission and reception

#### UART code examples

Information about UART code examples can be found in the [MSPM0 SDK examples guide](#).

### 4.3 Serial Peripheral Interface (SPI)

MSPM0 and NXP's M0 MCUs both support serial peripheral interface (SPI). Overall, MSPM0 and S32K1xx/KEA128x/KM35x SPI support is comparable with the difference listed in [Table 4-3](#).

**Table 4-3. SPI Feature Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0
Controller or peripheral operation	Yes	Yes	Yes	Yes
Data bit width (controller mode)	8 bits	8 bits	8 to 16 bits	4 to 16 bits
Data bit width (peripheral mode)	8 bits	8 bits	8 to 16 bits	7 to 16 bits
Maximum speed	10MHz	12MHz	37.5 MHz	MSPM0C: 12 MHz
				MSPM0L: 16MHz
				MSPM0G: 32MHz
Full-duplex transfers	Yes	Yes	Yes	Yes
Half-duplex transfer (bidirectional data line)	Yes	No	Yes	No
Simplex transfers (unidirectional data line)	Yes	Yes	Yes	Yes
Hardware chip select management	Equivalent	Equivalent	Equivalent	Yes
Programmable clock polarity and phase	Yes	Yes	Yes	Yes
Programmable data order with MSB-first or LSB-first shifting	Yes	Yes	Yes	Yes
SPI format support	N/A	N/A	N/A	Motorola, TI, MICROWIRE
Hardware CRC	No	No	No	No, MSPM0 offers SPI parity mode
TX FIFO depth	4	N/A	8	4

**Table 4-3. SPI Feature Comparison (continued)**

Feature	S32K1xx	KEA128x	KM35x	MSPM0
RX FIFO depth	4	N/A	8	4

SPI code examples

Information about SPI code examples can be found in the [MSPM0 SDK examples guide](#).

#### 4.4 I2C

MSPM0 and NXP's M0 MCUs both support I2C. In both MSPM0 and NXP's M0 MCUs, the I2C functionality is handled by the I2C module.

**Table 4-4. I2C Feature Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0
Controller and target modes	Yes	Yes	Yes	Yes
Multi-controller capability	Yes	Yes	Yes	Yes
Standard-mode (up to 100 kHz)	Yes	Yes	Yes	Yes
Fast-mode (up to 400 kHz)	Yes	Yes	Yes	Yes
Fast-mode Plus (up to 1 MHz)	Yes	Yes	Yes	Yes
Addressing mode	7 or 10 bit	7 or 10 bit	7 or 10 bit	7 or 10 bit
Peripheral addresses	1 address (later)	1 address	1 address	2 addresses
General call	Yes	Yes	Yes	Yes
Programmable setup and hold times	Yes	Yes	Yes	No
Event management	No	No	No	Yes
Clock stretching	Yes	Yes	Yes	Yes
Software reset	Yes	Yes	Yes	Yes
FIFO/Buffer	Yes (Controller Only)	N/A	N/A	TX: 8 byte RX: 8 byte
DMA	Yes (Target Only)	No	No	Yes
Programmable analog and digital noise filters	No	Equivalent	Equivalent	Yes

I2C code examples

Information about I2C code examples can be found in the [MSPM0 SDK examples guide](#).

#### 4.5 Timers (TIMGx, TIMAx)

MSPM0 and NXP's M0 MCUs both offer various timers. MSPM0 offers timers with varying features that support use cases from low-power monitoring to advanced motor control.

**Table 4-5. Timer Feature Comparison**

Feature	S32K1xx Timers	KEA128x Timers	KM35x Timers	MSPM0G Timers	MSPM0L and MSPM0C Timers
Resolution	16 bit	16 bit	16 bit	16 bit, 32 bit	16 bit
PWM	Yes	Yes	Yes	Yes	Yes
Capture	Yes	Yes	Yes	Yes	Yes
Compare	Yes	Yes	Yes	Yes	Yes
One-shot	Yes	Yes	Yes	Yes	Yes
Up down count functionality	Yes	Yes	Yes	Yes	Yes

**Table 4-5. Timer Feature Comparison (continued)**

Feature	S32K1xx Timers	KEA128x Timers	KM35x Timers	MSPM0G Timers	MSPM0L and MSPM0C Timers
Power Modes	Yes	Yes	Yes	Yes	Yes
QEI support	No	No	No	Yes	No
Programmable prescaler	Yes	Yes	Yes	Yes	Yes
Shadow register mode	No	No	No	Yes	Yes
Events/Interrupt	Yes	Yes	Yes	Yes	Yes
Fault Event Mechanism	Equivalent	Equivalent	Equivalent	Yes	No
Auto reload functionality	Yes	Yes	Yes	Yes	Yes

Timer code examples

Information about timer code examples can be found in the [MSPM0 SDK examples guide](#).

## 4.6 Windowed Watchdog Timer (WWDT)

MSPM0 and NXP's M0 MCUs both offer Window Watchdog Timers. The window watchdog timer (WWDT) initiates a system reset when the application fails to check in during a specified window of time.

**Table 4-6. WWDT Naming**

Key	S32K1xx/ KEA128x/KM35x	MSPM0
Name	Watchdog Timer	Windowed watchdog timer
Abbreviated name (same order)	WDOG	WWDT

**Table 4-7. WDT Feature Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0
Window mode	Yes	Yes	Yes	Yes
Interval timer mode	No	No	No	Yes
LFCLK source	Yes	Yes	Yes	Yes
Interrupts	Yes	Yes	Yes	Yes
Counter resolution	16 bit	16 bit	32 bit	25 bit
Clock divider	Yes	Yes	Yes	Yes

WWDT code examples

Information about WWDT code examples can be found in the [MSPM0 SDK examples guide](#).

## 4.7 Real-Time Clock (RTC)

MSPM0<sup>1</sup> and some NXP's M0 MCUs offer a real-time clock (RTC). The S32K1xx/KEA128x/KM35x devices contain a dedicated RTC module. The real-time clock (RTC) module provides time tracking for the application, with counters for seconds, minutes, hours, day of the week, day of the month, and year in selectable binary or binary-coded decimal format.

**Table 4-8. RTC Feature Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G
Power modes	Yes	Yes	Yes	Yes
Binary coded format	No	No	No	Yes
Leap year correction	No	No	Yes	Yes
Number of customizable alarms	1	1	1	2
Internal or/and External crystal	N/A	N/A	Yes	Yes
Crystal offset calibration	No	No	No	Yes

<sup>1</sup> Only MSPM0G devices support RTC.

**Table 4-8. RTC Feature Comparison (continued)**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G
Prescaler blocks	Equivalent	Yes	No	Yes
Interrupts	Yes	Yes	Yes	Yes

RTC code examples

Information about RTC code examples can be found in the [MSPM0 SDK examples guide](#).

## 5 Analog Peripheral Comparison

### 5.1 Analog-to-Digital Converter (ADC)

MSPM0 and NXP's M0 MCUs both offer ADC peripherals to convert analog signals to a digital equivalent. [Table 5-1](#) compares the different features and modes of the ADCs.

**Table 5-1. Feature Set Comparison**

Feature	NXP S32K1xx	NXP KEA128x	NXP KM35x	MSPM0
Resolution (Bits)	12	12	16	12/10/8
Conversion Rate (Msps) (12-bit)	1.16	0.340	4.75	MSPM0Gx - 4
				MSPM0Lx - 1.68
				MSPM0Cx - 1.5
Oversampling (Bits)	No	No	No	14
Hardware Oversampling	No	No	No	128x
FIFO	No	Yes	No	Yes
ADC Reference (V)	Alternate: $2.7 \leq V_{REF} \leq V_{DDA} + 0.1$	Internal: $V_{SSA} \text{ to } V_{DDA}$	Alternate: $V_{SSA} \text{ to } V_{DDA}$	Internal: 1.4, 2.5 VDD
	Primary: $2.7 \leq V_{REF} \leq V_{DDA} + 0.1$	External: $V_{SSA} \text{ to } V_{DDA}$	External: $V_{SSA} \text{ to } V_{DDA}$	External: $1.4 \leq V_{REF} \leq V_{DD}$
Operating Power Modes	VLPR, STOP2, RUN	RUN, WAIT, STOP	RUN, WAIT, VLPS	Run, Sleep, Stop, Standby (1)
Auto Power Down	Equivalent	Equivalent	Equivalent	Yes
External Input Channels (2)	S32K118-Up to 16	Up to 16	Up to 16	MSPM0Gx - up to 17
	S32K116-Up to 13			MSPM0Lx/Cx up to 10
Internal Input Channels	Supply Monitoring	Internal gap, Temperature Sensor	Internal gap, Temperature Sensor	Temperature Sensor, Supply Monitoring, Analog Signal Chain
DMA Support	Yes	No	Yes	Yes
ADC Window Comparator Unit	No	No	No	Yes
Simultaneous Sampling	No	No	No	MSPM0Gx - Yes
				MSPM0Lx/Cx - No
Number of ADCs (3)	1	1	1	MSPM0Gx - 2
				MSPM0Lx/Cx - 1

(1) ADC can be triggered in standby mode, which changes the operating mode.

(2) The number of external input channels varies per device.

(3) The number of ADCs varies per device.

**Table 5-2. Conversion Modes**

Mode	S32K1xx	NXP KEA128x	NXP KM35x	MSPM0	Comments
Single Conversion Mode	Yes	Yes	Yes	Single Channel Single Conversion	ADC samples and converts a single channel once
Scan a Sequence of Channels	No	No	No	Sequence of Channels Conversion	ADC samples a sequence of channels and converts once.

**Table 5-2. Conversion Modes (continued)**

Mode	S32K1xx	NXP KEA128x	NXP KM35x	MSPM0	Comments
Continuous Conversion Mode	Yes	Yes	Yes	Repeat Single Channel Conversion	Repeat single channel continuously samples and converts one channel
	No	No	No	Repeat Sequence of Channels Conversion	Samples and converts a sequence of channels then repeats the same sequence
Discontinuous Mode	No	No	No	Repeat Sequence of Channels Conversion	Samples and converts a discontinuous set of channels. This can be done on MSPM0 by mapping the MEMCTRLx to different channels.

ADC code examples

Information about ADC code examples can be found in the [MSPM0 SDK examples guide](#).

## 5.2 Comparator (COMP)

MSPM0 and NXP's M0 MCUs both offer integrated comparators as optional peripherals on some devices. In the MSPM0 family, comparators are denoted as COMPx, where the 'x' final character refers to the specific comparator module being considered. The S32K1xx family features a COMP module with an internal single comparator with six to eight inputs, the KEA128x family supports up to four inputs, and the KM35x Up to six inputs. The MSPM0 comparator module provides a windowed comparator functionality using its DAC with two programmable levels. Both MSPM0 and NXP's M0 MCUs have multiple channels that can take inputs from various internal and external sources and can be used to trigger changes in power mode or truncate/control PWM signals. A summary of how the MSPM0 and NXP M0 comparator modules compare feature-by-feature is included in [Table 5-3](#).

**Table 5-3. COMP Feature Set Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G	MSPM0L
Available comparators	1	2	3	Up to 3	1
Number of positive and negative inputs	Up to 6 pos, up to 6 neg	Up to 4 pos, up to 4 pos	Up to 6 pos, up to 6 neg	Up to 4 pos, up to 3 neg	Up to 2 pos, up to 2 neg
	Up to 8 pos, Up to 8 neg				
Output routing	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins
	Interrupt	Interrupt	Interrupt	Interrupt/Event Interface	Interrupt/Event Interface
Noninverting input sources	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins
				DAC12 output <sup>(1)</sup>	DAC8 output
	DAC8 output	DAC6 output	DAC6 output	DAC8 output	OPA1 Output <sup>(2)</sup>
				Internal V <sub>REF</sub> : 1.4 V and 2.5 V	
			OPA1 output <sup>(2)</sup>		

**Table 5-3. COMP Feature Set Comparison (continued)**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G	MSPM0L
<b>Inverting input sources</b>	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O pins	Multiplexed I/O Pins
	No	No	No	Internal temperature sensor	Internal temperature sensor
	DAC8 output	DAC6 output	DAC6 output	Internal $V_{REF}$ : 1.4V and 2.5V	DAC8 output
	No	No	No	DAC8 output	OPA0 <sup>(3)</sup> output
	No	No	No	OPA0 output <sup>(3)</sup>	
<b>Programmable hysteresis</b>	None, 15mV/19mV, 23mV/34mV, 32mV/46mV	None, 15mV,20mV, 30mV	5mV, 10mV, 20mV, 30mV	None, 10mV, 20mV, 30mV	None, 10mV, 20mV, 30mV
				Other values from 0V to $V_{REF}/V_{DD}$ using DAC8	Other values from 0V to $V_{DD}$ using DAC8
<b>Register lock</b>	No	No	No	Yes, some COMP registers (writes require key)	Yes, some COMP registers (writes require key)
<b>Window comparator configuration</b>	Yes	No	Yes	Yes	No (single COMP)
<b>Input short mode</b>	No	No	No	Yes	Yes
<b>Operating modes</b>	VLPR (LS compare only), STOP1, STOP2, RUN	RUN, WAIT, STOP	RUN, WAIT, VLPS	High speed, low power	High speed, low power
<b>Fast PWM shutdowns</b>	No	No	No	Yes (through TIMA fault handler)	No
<b>Output filtering</b>	Low-pass filter	No	Yes	Blanking filter	Blanking filter
				Adjustable analog filter	Adjustable analog filter
<b>Output polarity control</b>	Yes	Yes	Yes	Yes	Yes
<b>Interrupts</b>	Rising edge	Rising edge	Rising edge	Rising edge	Rising edge
	Falling edge	Falling edge	Falling edge	Falling edge	Falling edge
	Both edges	Both edges	Both edges	Output ready	Output ready
<b>Exchange inputs mode</b>	Yes	Yes	Yes	Yes	Yes

- (1) Only on devices with DAC12 peripheral
- (2) Only on devices with OPA1 peripheral
- (3) Only on devices with OPA0 peripheral

**COMP code examples**

Information about COMP code examples can be found in the [MSPM0 SDK examples guide](#).

**5.3 Digital-to-Analog Converter (DAC)**

The S32K1xx, KEA128x, and KM35x MCU families offers a basic DAC inside its COMP module. However, when migrating from the M0 family to the MSPM0 family, you can make use of the MSPM0 internal 12-bit DAC to generate analog voltages. The MSPM0 family offers a 12-bit DAC peripheral to perform digital-to-analog conversion for various applications. In the [MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual](#), the MSPM0 series data sheets, and the MSPM0 SDK, the 12-bit DAC peripheral is referred to as the DAC12. This differentiates the DAC12 from the 8-bit DACs that are available for use with each comparator peripheral included in a given MSPM0 device. Those additional 8-bit DACs are covered in the comparator section of this document. This DAC12 peripheral is only available on the MSPM0G family of devices.

The features of the 12-bit DAC peripherals for the MSPM0G are summarized in [Table 5-4](#).

**Table 5-4. DAC Feature Set Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G
Resolution	8-bit	6-bit	6-bit	12 bits (11 ENOB)
Output rate	N/A	N/A	N/A	1 MSPS
Output channels	1	1	1	1 <sup>(1)</sup>
Data formats	N/A	N/A	N/A	8-bit right aligned, 12-bit right aligned, two's complement or straight binary
DMA integration	No	No	No	Yes
Output routing	Internal peripheral connection only: CMP	Internal peripheral connection only: CMP	DAC Output	External Pins
			Internal peripheral connection: CMP	Internal peripheral connections: OPA IN+, COMP IN+, ADC0
Internal reference voltage	Yes, 1V or VDDA	Yes, 1.2V or VDDA	Yes, 1.2V or VDDA	Yes, 2.5V or 1.4V
External reference voltage	No	No	No	Yes
FIFO	No	No	No	Yes
Output buffer	No	No	No	Yes
Configurable output offset	No	No	No	Yes
Self-calibration mode	No	No	No	Yes
Automatic waveform generation	No	No	No	No
Sample and hold mode	No	No	No	No
Trigger sources	Operation clock, trigger start signal	N/A	Trigger start signal	Internal dedicated sample time generator, DMA interrupts/events, FIFO threshold interrupts/events, two hardware triggers (available from event fabric)

(1) Dual DAC channels are planned for future MSPM0G devices.

#### DAC12 code examples

Information about DAC12 code examples can be found in the [MSPM0 SDK examples guide](#).

### 5.4 Operational Amplifier (OPA)

The NXP S32K1XX, KEA128x, and MK35x MCU families do not offer an integrated Operational Amplifier (OPA) peripheral, but when migrating from the S32K1XX, KEA128x and KM35x families to the MSPM0 family, you can make use of the MSPM0 internal OPAs to replace external discrete devices, or to buffer internal signals as necessary. The MSPM0 OPA modules are completely flexible and can, individually or in combination, replace many discrete amplifiers in sensing or control applications. The primary features of the MSPM0 OPA modules are included in [Table 5-5](#), and examples of common OPA configurations you can recreate are included in the OPA code examples mentioned below.

**Table 5-5. MSPM0 OPA Feature Set**

Feature	MSPM0 Implementation
Input type	Rail to rail (can be enabled or disabled)
Gain bandwidth	1MHz (low-power mode)
	6MHz (standard mode)



**Table 5-5. MSPM0 OPA Feature Set (continued)**

Feature	MSPM0 Implementation
Amplifier configurations	General-purpose mode
	Buffer mode
	PGA mode (inverting or noninverting)
	Differential amplifier mode
	Cascade amplifier mode
Input/output routing	External pin routing
	Internal connections to ADC and COMP modules
Fault detection	Burnout current source (BCS)
Chopper stabilization	Standard (selectable chopping frequency)
	ADC assisted chop
	Disabled
Reference voltages	Internal VREF (MSPM0G devices only)
	DAC12 (MSPM0G devices only)
	DAC8 (devices with COMP module only)

OPA code examples

Information about OPA code examples can be found in the [MSPM0 SDK examples guide](#).

### 5.5 Voltage References (VREF)

MSPM0 and NXP's M0 MCUs both have internal references that can be used to supply a reference voltage to internal peripherals and output to external peripherals.

**Table 5-6. Feature Set Comparison**

Feature	S32K1xx	KEA128x	KM35x	MSPM0G	MSPM0L MSPM0C
Internal Reference (V)	2.7, 5.5	2.7, 5.5	2.7, 3.6	1.4, 2.5	1.4, 2.5
External Reference (V)	Up to VDD	Up to VDD	Up to VDD	External: $1.4 \leq V_{REF} \leq V_{DD}$	External: $1.4 \leq V_{REF} \leq V_{DD}$
Output Internal Reference	No	No	No	Yes	Yes
Internally Connect to ADC	Yes	Yes	Yes	Yes	Yes
Internally Connect to DAC	Yes	Yes	Yes	Yes	No
Internally Connect to COMP	Yes	Yes	Yes	Yes	No
Internally Connect to OPA	N/A	N/A	N/A	Yes	No

For the MSPM0 VREF, you must enable the power bit, PWREN Bit0 (ENABLE).

VREF code examples

Code examples that use VREF can be found in the [MSPM0 SDK examples guide](#).

## 6 References

- [MSPM0 SDK User Guide](#)
- [SysConfig Guide for MSPM0](#)
- [LP-MSPM0G3507](#)
- [LP-MSPM0L1306](#)
- [MSPM0C1104](#)
- [TMDSEMU110-U](#)
- Texas Instruments: [MSPM0G350x Mixed-Signal Microcontrollers With CAN-FD Interface Data Sheet](#)
- Texas Instruments: [MSPM0L130x Mixed-Signal Microcontrollers Data Sheet](#)
- Texas Instruments: [MSPM0C110x, MSPS003 Mixed-Signal Microcontrollers Data Sheet](#)
- Texas Instruments: [MSPM0 G-Series 80MHz Microcontrollers Technical Reference Manual](#)
- Texas Instruments: [MSPM0 L-Series 32MHz Microcontrollers Technical Reference Manual](#)
- Texas Instruments: [MSPM0 C-Series 24-MHz Microcontrollers Technical Reference Manual](#)
- [MSPM0 Driver Library Overview](#)
- Texas Instruments: [Cybersecurity Enablers in MSPM0 MCUs](#)

## 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](https://www.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 © 2025, Texas Instruments Incorporated