Application Note

Migration Guide From Microchip to MSPM0

ABSTRACT

This application note assists with migrating from Microchip's 8-bit AVR® 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 Microchip AVR® ecosystem to quickly ramp with the MSPM0 series of MCUs.

Table of Contents

1 MSPM0 Portfolio Overview 2
1.1 Introduction 2
1.2 Portfolio Comparison of Microchip AVR ATmega and ATiny MCUs to MSPM0 2
2 Ecosystem and Migration 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 7
3 Core Architecture Comparison 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 18
3.5 MSPM0 Operating Modes Summary and Comparison 19
3.6 Interrupt and Events Comparison 21
3.7 Debug and Programming Comparison 23
4 Digital Peripheral Comparison 24
4.1 General-Purpose I/O (GPIO, IOMUX) 24
4.2 Universal Asynchronous Receiver-Transmitter (UART) 25
4.3 Serial Peripheral Interface (SPI) 26
4.4 I2C 27
4.5 Timers (TIMGx, TIMAx) 28
4.6 Windowed Watchdog Timer (WWDT) 28
4.7 Real-Time Clock (RTC) 29
5 Analog Peripheral Comparison 29
5.1 Analog-to-Digital Converter (ADC) 29
5.2 Comparator (COMP) 31
5.3 Digital-to-Analog Converter (DAC) 32
5.4 Operational Amplifier (OPA) 33
5.5 Voltage References (VREF) 34
6 References 34

Trademarks

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 a ultra-low-power system architecture, allowing system designers to increase performance while reducing energy consumption.

MSPM0 MCUs offer a competitive alternative to Microchip's 8-bit AVR MCUs. This application note assists with migration from these Microchip MCUs to TI MSPM0 MCUs by comparing device features and ecosystems.

1.2 Portfolio Comparison of Microchip AVR ATmega and ATiny MCUs to MSPM0

Table 1-1. Comparison of the MSPM0 and Microchip 8-bit AVR MCUs

	Microchip ATmega 169A/ 329A Series	Microchip ATmega 48/88/168 Series	Microchip ATtiny 42x/82x Series	TI MSPM0 MSPM0Gx Series	TI MSPM0 MSPM0Lx Series	TI MSPM0 MSPM0Cx Series
Core / Frequency	AVR RISC 16/20MHz	AVR RISC 20MHz	AVR RISC 20MHz	CM0+ / 32-80MHz	CM0+ / 32MHz	CM0+ / 24MHz
Supply Voltage	1.8V to 5.5V	1.8V to 5.5V	1.8V to 5.5V	1.62V to 3.6V	1.62V to 3.6V	1.62V to 3.6V
Temperature	-40°C to 85°C	-40°C to 85°C	-40°C to 85°C	-40°C to 125°C	-40°C to 125°C	-40°C to 125°C
Memory	64KB to 16KB	16KB to 4KB	8KB to 4KB	128KB to 32KB	64KB to 8KB	16KB to 8KB
RAM	Up to 4KB	Up to 1KB	Up to 1KB	Up to 32KB	Up to 4KB	1KB
EEPROM	Up to 2KB	Up to 512B	128B	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)	54/69	23	12	Up to 60	Up to 28	Up to 18
Analog	1x 15ksps, 10-bit ADC 1x comparator	1x 15ksps, 10-bit ADC 1x comparator	1x 375ksps, 12-bit ADC 1x Comparator	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)	1x SPI 1x I2C 1x UART	1x SPI 1x I2C 1x UART	1x SPI 1x I2C 2x 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 8bit 1x 16bit	2x 8bit 1x 16bit	1x 16-bit type A 2x 16-bit type B	6x 16bit, 1x 32bit	4x 16bit	3x 16bit
Advance Timers	No	No	No	2x 16-bit Advanced	No	No
Hardware Accelerator	HW 2-cycle Multiplier	HW 2-cycle Multiplier	HW 2-cycle Multiplier	MATHACL	N/A	N/A
Security	No	No	CRC	CRC, TRNG, AES256	CRC	CRC
Low power	Active: 250µA/MHz Standby 15µA	Active: 281µA/MHz Standby 2.5µA	Active: 455µA/MHz Standby 0.7µA	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. Microchip Software Tool Equivalents for MSPM0

	Microchip	MSPM0
IDE	MPLAB X IDE	Code Composer Studio Theia™ IDE (CCS)
Software Configuration	MPLAB Code Configurator	SysConfig
Stand-alone programming	MPLAB IPE	UniFlash
Display/Demo GUI Editor	MPLAV Garniby Graphics Composer	GuiComposer

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, refer to the *MSPM0 SDK User's Guide*.

2.1.2 MPLAB X IDE vs Code Composer Studio IDE (CCS)

Code Composer Studio Theia IDE (CCS) is TI's equivalent of Microchip's MPLAB X IDE. CCS is a free Eclipse-based IDE that supports TI's microcontroller (MCU) 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 MPLAB Code Configurator 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 Microchips MPLAB Code Configurator. 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, refer to the MSPM0 SysConfig Guide.

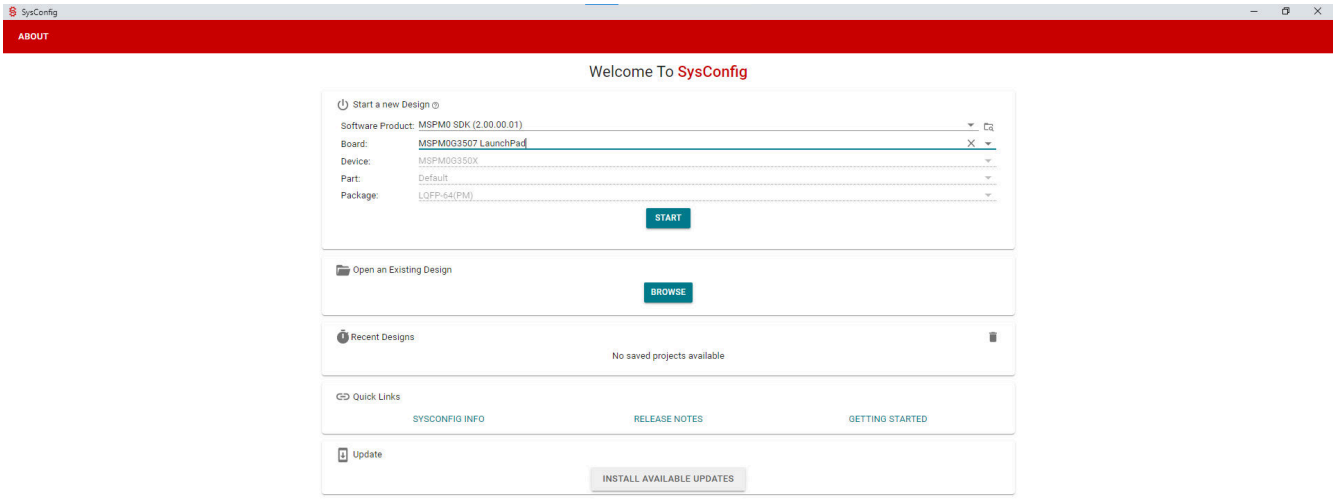

Figure 2-1. MSPM0 SysConfig

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

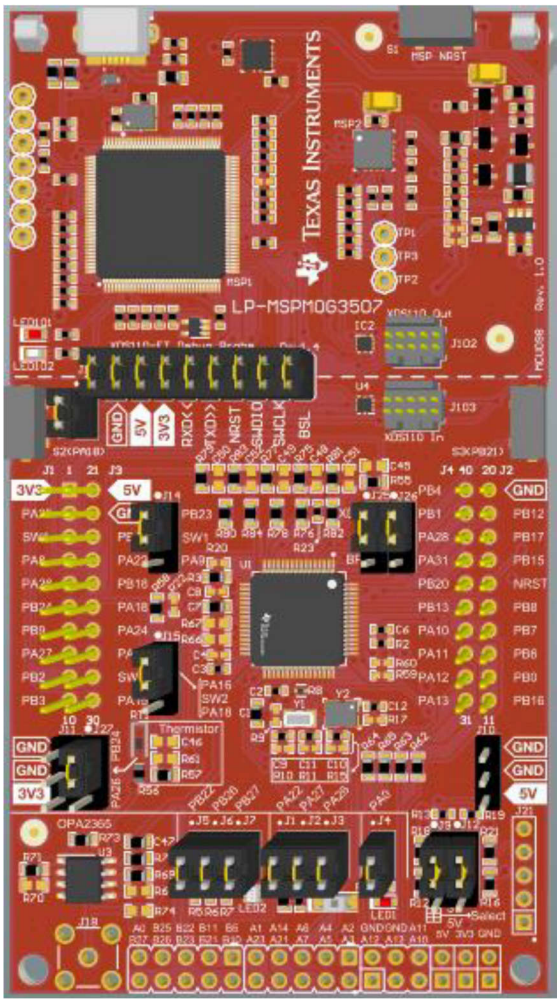

Figure 2-2. LP-MSPM0G3507 LaunchPad Development Kit

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, refer to 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.

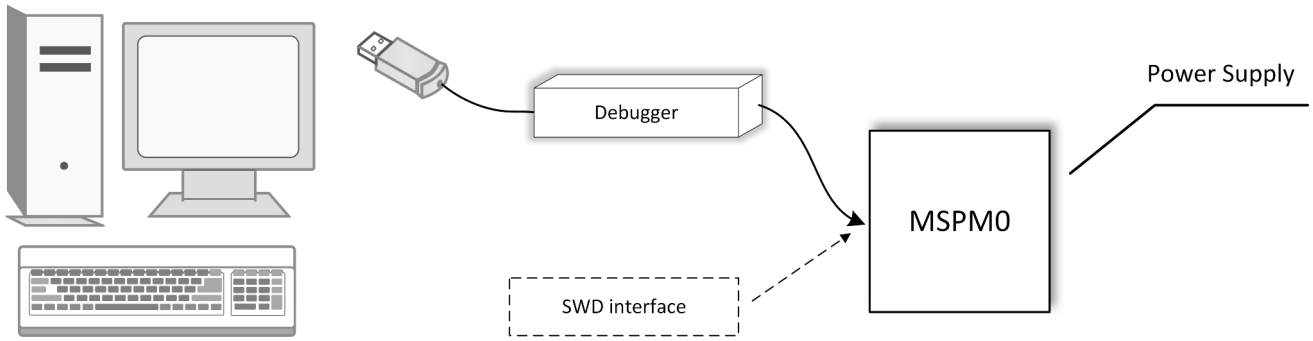

Figure 2-3. MSPM0 Debugging

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 Microchip to MSPM0. Finally, once the software ported, download and debug the application with our debugging tools.

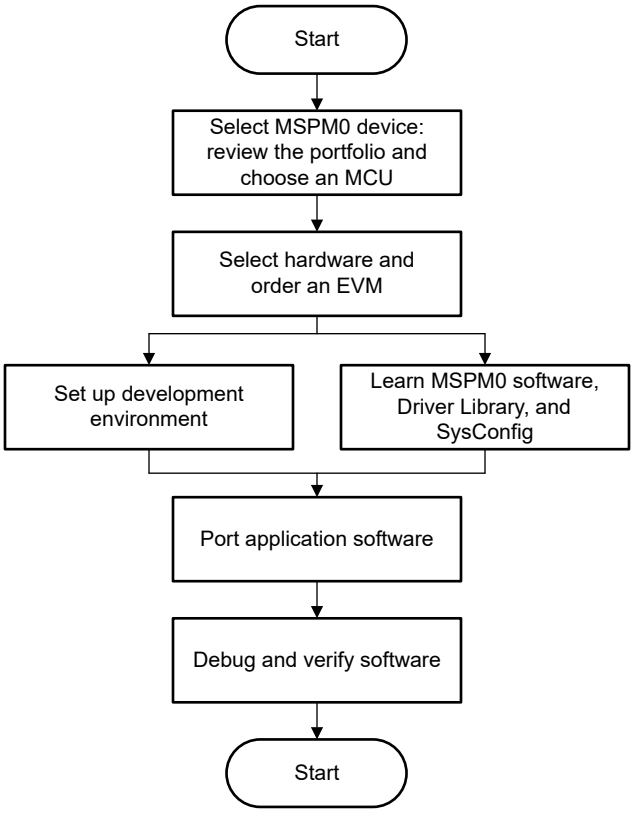

Figure 2-4. MSPM0 Migration Flowchart

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 Microchip to MSPM0, this description includes the steps to port a basic low-power UART application from a Microchip 8-bit device to an MSPM0 device using an existing UART example as the starting point. This example starts with a USART example for Microchip ATtiny and ATmega devices with a USART 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 using the product selection tool. When choosing a replacement for a Microchip ATtiny or ATmega 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 MSPM0C1104 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 MSPM0C1104 has a LaunchPad development kit (LP-MSPM0C1104) 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. Section 2.1 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.

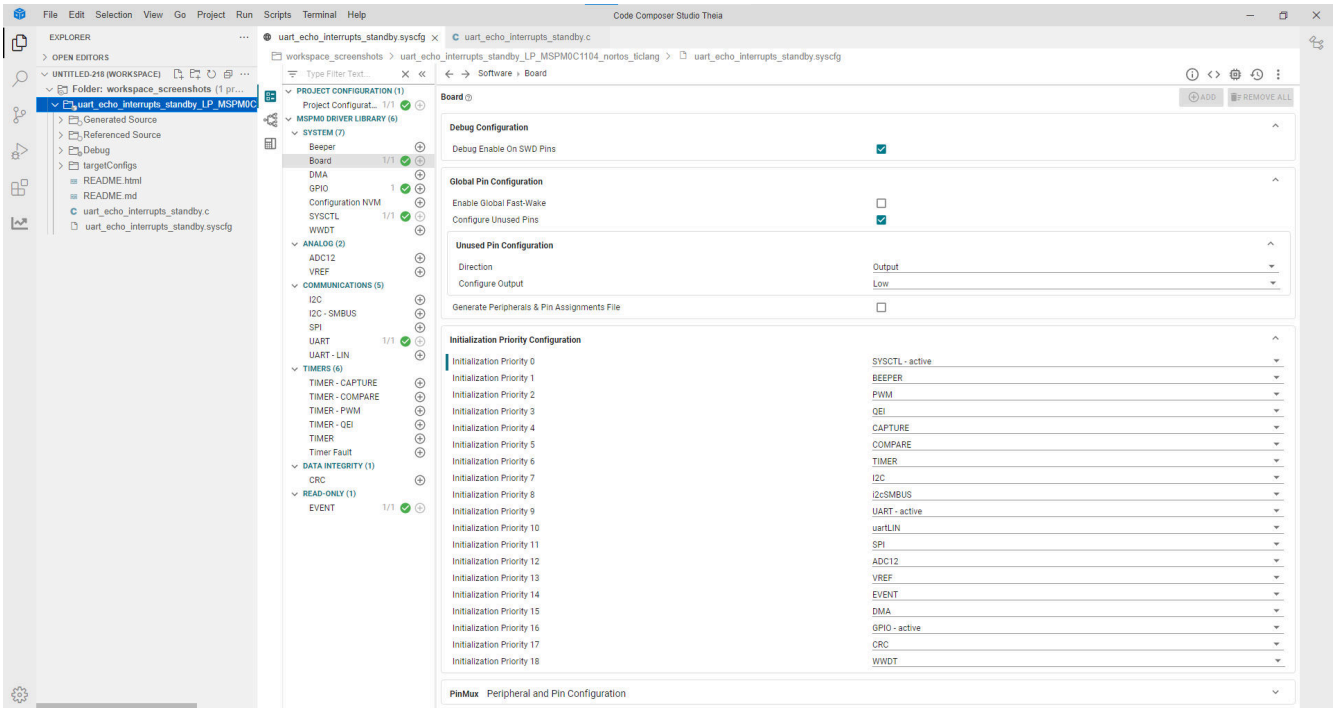

Figure 2-5. Code Composer Studio IDE

4. Software porting.

   When the environment is ready, start using the MSPM0 SDK. As mentioned, the MSPM0 SDK is similar to the MPLAB Harmony software package. The MSPM0 SDK offers different layers for software development. For equivalents to Microchip'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 low-power UART example from MPLAB Discover. 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 example project from Microchip's 'Getting Started with USART user guide.

      ```
      The following code continually sends the string “Hello World!”. A string is sent character
      by character. The ‘USART0_sendString’ function calls the ‘USART0_sendCharacter’ function for
      each character in “Hello Word!” string. Before sending each character, the ‘USART0_sendChar’
      function waits for the previous character transmission to be completed. This is done by
      polling the status register, until the data register empty flag, STATUS.DREIF, is set.
      ```

      The first step is to understand the main settings for the MCU. This is generally clock speeds and power policies. In this example, the general clock frequency is not specified because the only important setting is that the UART works in low power Stop0 mode. It states the low power UART clock is based on the 'HIS' or high-speed internal oscillator meaning there is no external crystal being used. The UART runs at 9600 baud, 8 data bits, 1 start and stop bit, no parity. No hardware flow control is used. The application side checks for an 'S' or 's' to be received and blinks an LED.

   b. Find the closest MSPM0 example.

      Next step is to understand any differences between the UART modules for ATmega/ATtiny 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!".

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.

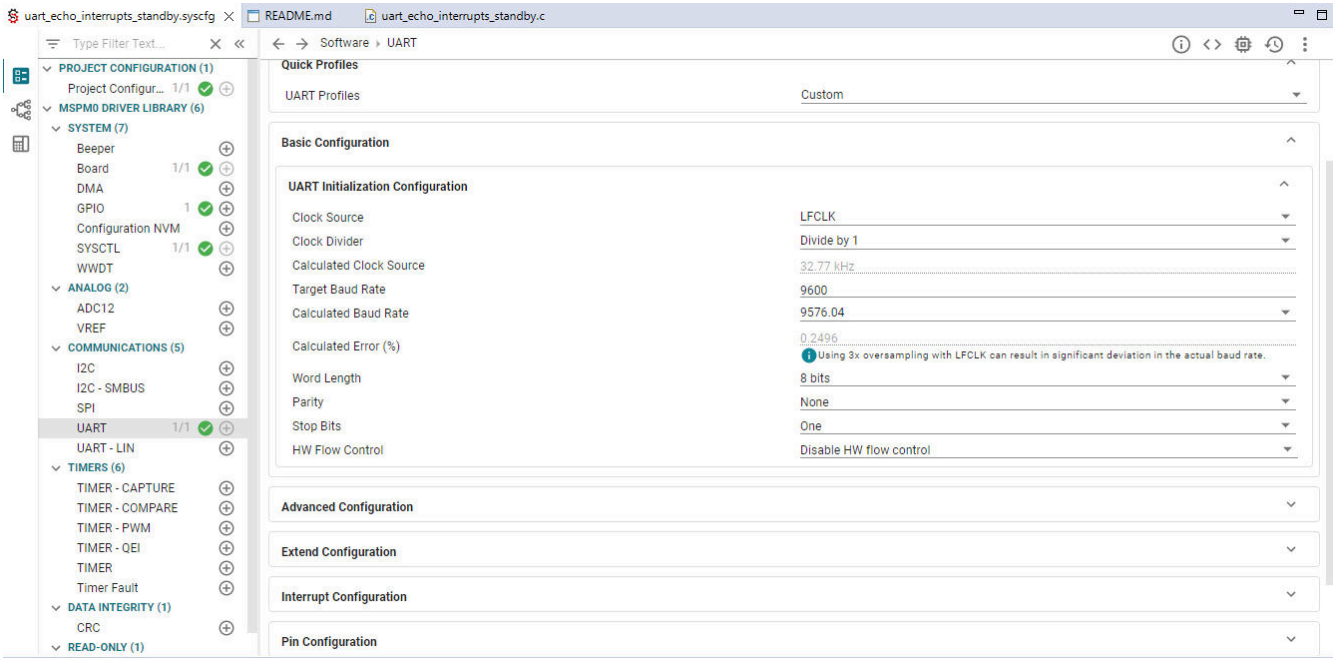

Figure 2-7. Sysconfig UART Tab

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

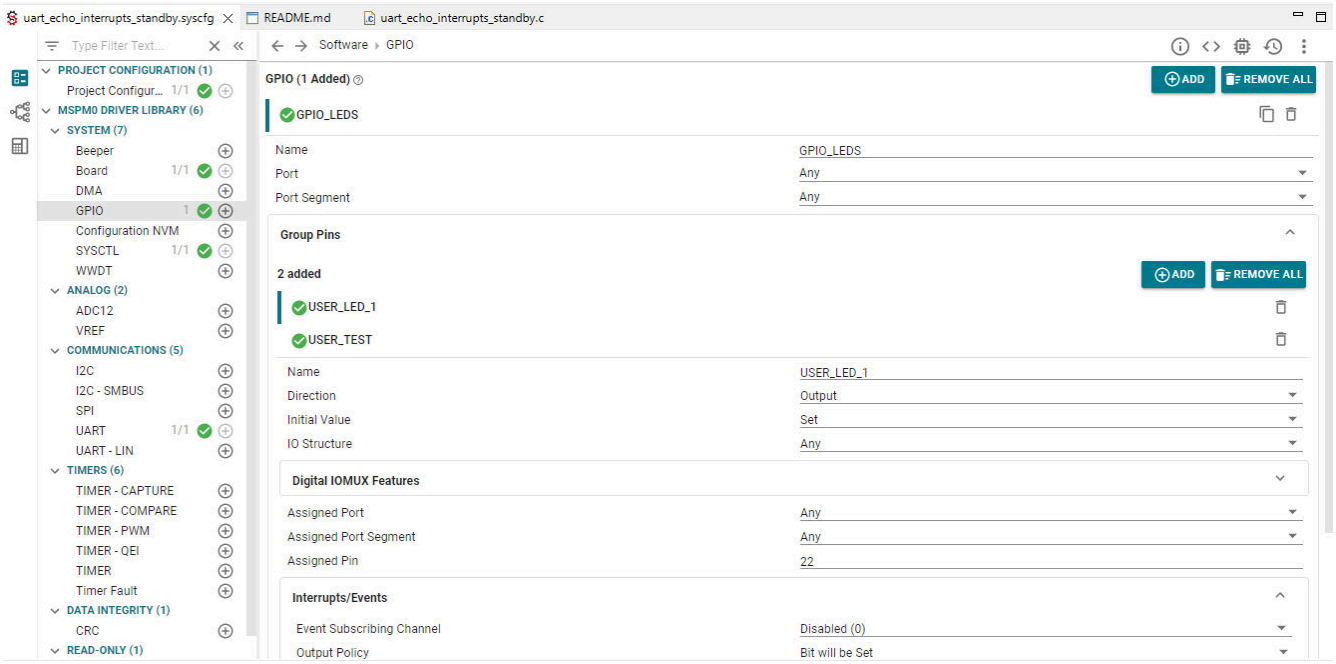

Figure 2-8. Sysconfig GPIO Tab

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 totoggle 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

```
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include "ti_msp_dl_config.h"

volatile uint8_t gEchoData = 0;
static uint8_t gMessage[12] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!'};

int main(void)
{
    SYSCFG_DL_init();

    NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);
    DL_SYSCTL_enableSleepOnExit();

    while (1) {
        __WFI();
    }
}

void UART_0_INST_IRQHandler(void)
{
    switch (DL_UART_Main_getPendingInterrupt(UART_0_INST)) {
        case DL_UART_MAIN_IIDX_RX:
            DL_GPIO_togglePins(GPIO_LEDS_PORT,
                GPIO_LEDS_USER_LED_1_PIN | GPIO_LEDS_USER_TEST_PIN);
            gEchoData = DL_UART_Main_receiveData(UART_0_INST);
            for(int i = 0; i < 12; i++){
                DL_UART_Main_transmitDataBlocking(UART_0_INST, gMessage[i]);
            }
            break;
        default:
            break;
    }
}

```

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-MSPM0C1104 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.

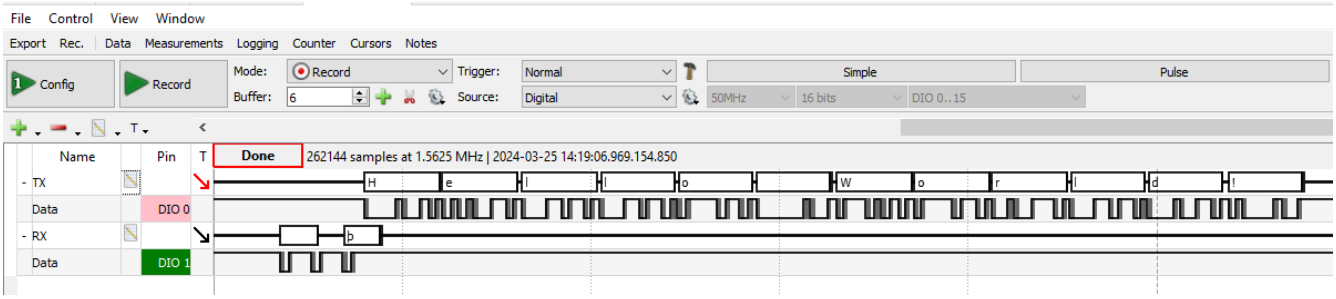

Figure 2-10. Logic Analyzer Capture

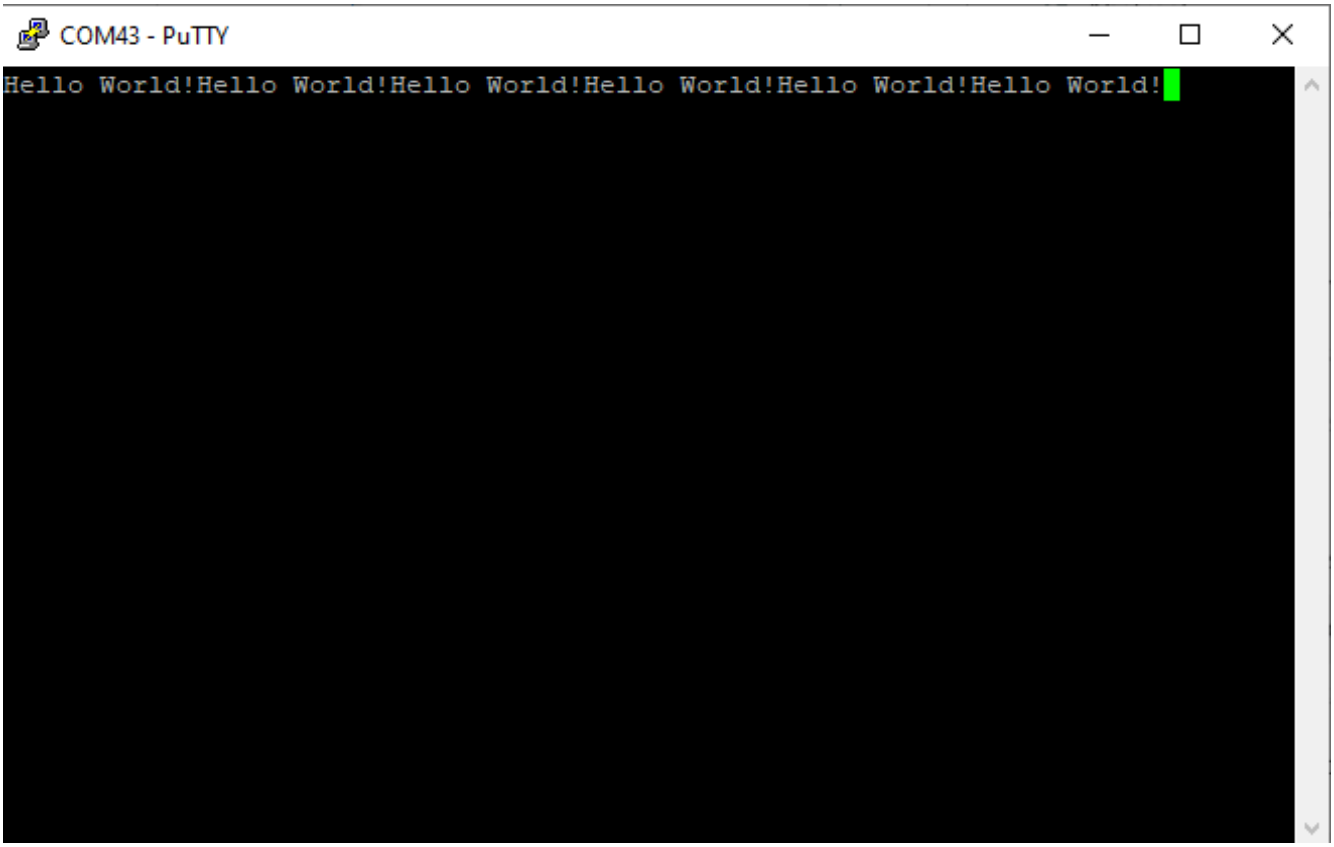

Figure 2-11. Serial Terminal

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 ATmega/ATtiny and MSPM0 family of parts are quite different from each other. These Microchip devices utilize a proprietary 8-bit CPU core, while the MSPM0 devices utilize an ARM M0+ 32-bit core. The table below gives a high-level overview of the general features of the CPUs in the MSPM0G and MSPM0L families compared to the ATtiny and ATmega devices. Section 3.6.1 provides 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	ATmega	ATtiny	MSPM0G	MSPM0L	MSPM0C
Architecture	Microchip 8-bit AVR	Microchip 8-bit AVR	Arm Cortex-M0+	Arm Cortex-M0+	Arm Cortex-M0+
Maximum MCLK	16MH	20MHz	32 up to 80MHz	32MHz	24MHz
CPU instruction cache	None	None	4x64 bit lines (32 bytes)	2x64-bit lines (16 bytes)	None
Processor trace capabilities	No	No	Yes, integrated micro trace buffer	No	No
Memory protection unit (MPU)	No	No	Yes	No	No
System timer (SYSTICK)	No	No	Yes - 24 bit	Yes - 24 bit	No
NVM prefetch	No	Yes	Yes	Yes	Yes
Hardware multiply	Yes	Yes	Yes	Yes	No
Hardware breakpoint / watchpoints	0	2 / 0	4/2	4/2	4/2
Boot routine storage	Flash (system memory)	Flash (system memory)	ROM	ROM	ROM
Bootstrap loader storage	Flash (system memory)	Flash (system memory)	ROM	ROM	No
Bootloader interface support(1) (2)	Available for all data interfaces	Available for all data interfaces	UART, I2C, user extendable	UART, I2C, user extendable	User defined
DMA	No	No	Yes - 7 ch	Yes - 3 ch	Yes - 1ch

(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 Microchip 8 bit family of MCUs feature nonvolatile Flash memory used for storing executable program code and application data.

Table 3-2. Comparison of Flash Feature

Features	ATmega	ATtiny	MSPM0G	MSPM0L	MSPM0C
Flash memory	4 - 64KB	4 - 32 KB	128KB to 32KB	64KB to 8KB	16KB to 8KB
Memory organization	Single Bank	Single Bank	Single Bank	Single Bank	Single Bank
Flash word size	8 bits	8 bits	64 bits plus 8 ECC bits	64 bits plus 8 ECC bits	64 bits plus 8 ECC bits
Programming resolution	8 bits	8 bits	Single word, 32-, 16-, or 8-bit (byte)	Single word, 32-, 16-, or 8-bit (byte)	Single word, 32-, 16-, or 8-bit (byte)
Erase	Page Erase Chip erase (all banks)	Page Erase Chip erase (all banks)	Erase Sector = 1KB Bank Erase (up to 256KB)	Erase Sector = 1KB Bank Erase (up to 256KB)	Erase Sector = 1KB Bank Erase (up to 256KB)
Write protection	Yes	Yes	Yes, static and dynamic	Yes, static and dynamic	Yes, static and dynamic

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

Features	ATmega	ATtiny	MSPM0G	MSPM0L	MSPM0C
Read protection	No	No	Yes	Yes	Yes
Flash memory read operations	8 bits	8 bits	64-bit Flash word size plus 8 ECC bits	64-bit Flash word size plus 8 ECC bits	64-bit Flash word size plus 8 ECC bits
Flash memory write operations	8 bits	8 bits	64-bit Flash word size plus 8 ECC bits	64-bit Flash word size plus 8 ECC bits	64-bit Flash word size plus 8 ECC bits
Error code correction (ECC)	No	No	8 bits for 64 bits	8 bits for 64 bits	8 bits for 64 bits
Prefetch	No	Yes	Yes	Yes	No
CPU instruction cache	N/A	N/A	Four 64-bit cache lines, 8x 32-bit instructions, or 16 x 16-bit instructions	Two 64-bit cache lines, 4x 32-bit instructions, or 8x 16-bit instructions	MSPM0Cxx: No cache

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.

3.2.3 Embedded SRAM

The MSPM0 and Microchip's 8-bit AVR MCU family feature SRAM used for storing application data.

Table 3-3. Comparison of SRAM Features

Feature	ATmega	ATtiny	MSPM0G	MSPM0L	MSPM0C
SRAM memory	512 B to 1 KB	512 B to 3 KB	32KB to 16KB	4KB to 2KB	1KB
			Select devices include SRAM parity and ECC. For more details, see the device data sheet.		
Access resolution	Byte	Byte	Byte, half-word (16-bits) or full word (32-bits)		
Parity check	No	No	Yes	Yes	No

MSPM0 MCUs include low-power high-performance SRAM with zero wait state access across the supported CPU frequency range of the device. SRAM can be used for storing volatile information such as the call stack, heap, and global data, in addition to code. The SRAM content is fully retained in run, sleep, stop, and standby operating modes, but is lost in shutdown mode. A write protection mechanism is provided to allow the application to dynamically write protect the lower 32KB of SRAM with 1KB resolution. On devices with less than 32KB of SRAM, write protection is provided for the entire SRAM. Write protection is useful when placing executable code into SRAM as it provides a level of protection against unintentional overwrites of code by either the CPU or DMA. Placing code in SRAM can improve performance of critical loops by enabling zero wait state operation and lower power consumption.

3.3 Power Up and Reset Summary and Comparison

Similar to Microchip 32 bit 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 3-4 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 3-4. Comparison of Power up

ATmega		ATtiny	MSPM0 Devices	
Modules governing power up and resets	Power Manager module, System Controller and Reset Module	Brown out detector, Reset Controller	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 or interrupts		Configurable BOR (Brownout Reset)	Can be configured as a reset or interrupt, with different voltage thresholds.

Microchip 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 3-5 shows the reset types in Microchip 8-bit AVR MCUs Devices. Table 3-6 gives a brief description of MSPM0 reset states. Figure 3-1 shows the relationship between all of the MSPM0 reset states.

Table 3-5. Reset Type in ATmega and ATtiny devices

ATmega and ATtiny Reset Types			
Reset Source	Power Supply Reset	User Reset	
Reset Name	Power on reset, brown out detect reset	External Reset	WDT Reset, software reset, Unified program and debug interface reset

Table 3-6. Comparison of Reset Domains

MSPM0 Reset States(1)	
POR	Typical triggers: POR voltage levels, SW trigger, NRST held low for >1s. Resets shutdown memory, re-enables NRST and SWD, triggers BOR
BOR	Typical triggers: POR or BOR voltage level, exit from shutdown mode. Resets PMU, VCORE, and associated logic. Triggers BOOTRST.
Boot reset (BOOTRST)	Typical triggers: BOR or software trigger, fatal clock failure, NRST held low for <1 s. Executes boot configuration routine. Resets majority of core logic and registers, including RTC, clock, and IO configurations.(2) SRAM power cycled and lost. Triggers SYSRST.
System reset (SYSRST)	Typical triggers: BOOTRST, BSL entry or exit, watchdog timer, software trigger, debug subsystem. Resets CPU state and all peripherals except RTC, LFCLK, LFXT, and SYSOSC frequency correction loop. Device enters RUN mode on exit.
CPU-only reset (CPURST)	Software and debug subsystem triggers only. Resets CPU logic only. Peripheral state are not affected.
RTC and associated clocks are reset through BOOTRST, BOR, or POR.(2)	

(1) Not all reset triggers 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 LFXT/LFLCK_IN configurations and IOMUX settings are NOT reset to allow RTC to maintain operation through external reset.

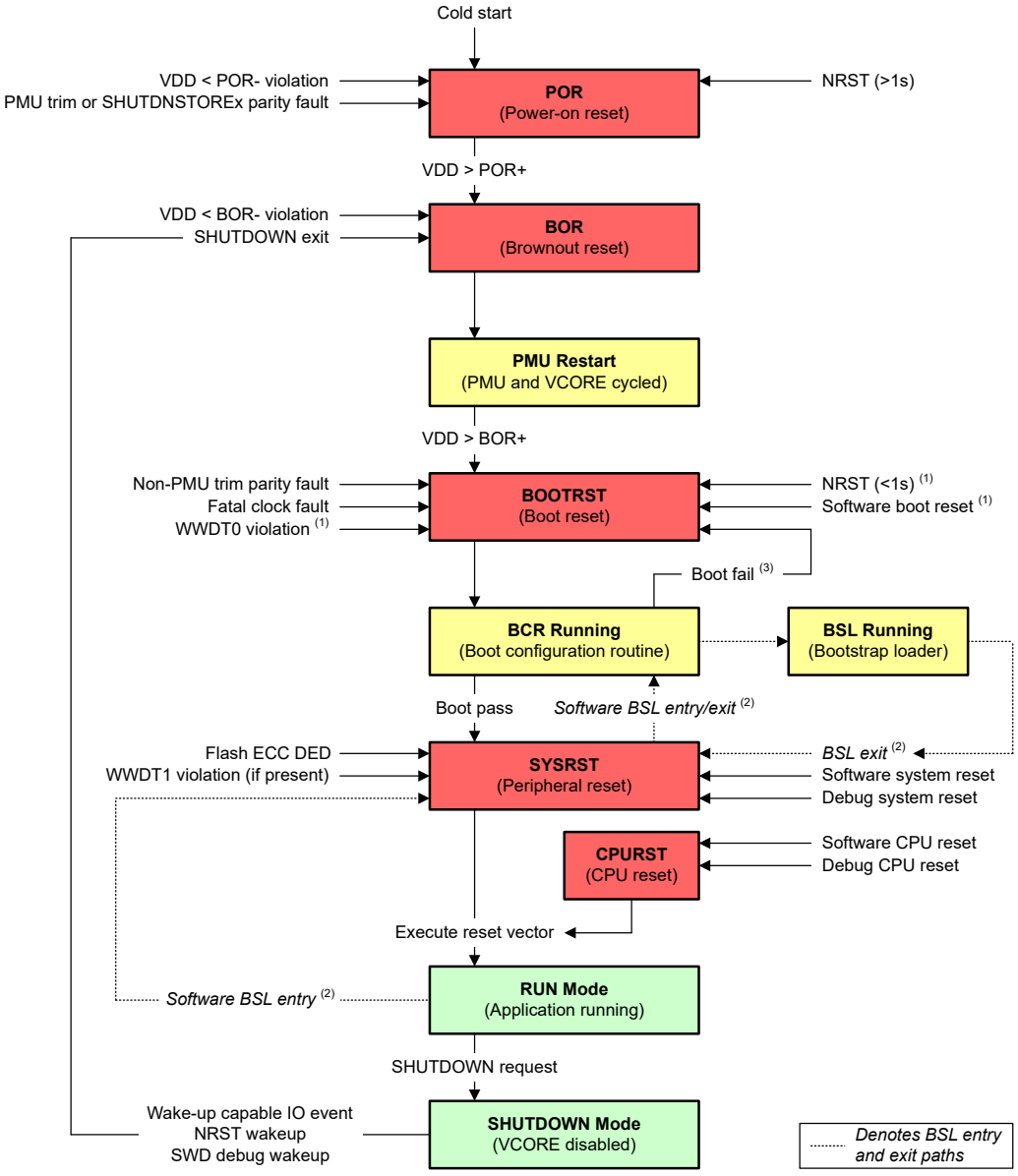

Figure 3-1. MSPM0 Reset Levels

3.4 Clocks Summary and Comparison

Microchip's 8-bit AVR 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 3-7. Oscillator Comparisons

ATMega	ATtiny	MSPM0
Calibrated Internal RC 8MHz	OSC20M	SYSOSC(1)
Full Swing Crystal	N/A	HFXT
External	External	HFCLK_IN (Digital Clock)
Internal 128kHz RC	OSCULP32K	LFOSC -32kHz
Low Frequency Crystal	XOSC32K	LFXT - 32kHz
N/A	N/A	LFCLK_IN
Low Power Crystal	N/A	LFXT - 32kHz

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

Table 3-8. Clock Comparison

ATmega	ATtiny	MSPM0G	MSPM0L/C
N/A	OSC20M	SYSOSC	SYSOSC
N/A	N/A	SYSPLLCLK1	N/A
N/A	N/A	SYSPLLCLK0	N/A
N/A	N/A	SYSPLLCLK2x (1)	N/A
CLK_cpu	CLK_CPU	BUSCLK(2)	BUSCLK(2)
CLK_cpu	CLK_CPU	BUSCLK(2)	BUSCLK(2)
CLK_flash	CLK_CPU	BUSCLK(2)	BUSCLK(2)
CLK_adc	CLK_PER	SYSOSC/ ULPCLK/ HFCLK	SYSOSC/ ULPCLK/ HFCLK
CLK_io	CLK_PER	BUSCLK(2)	BUSCLK(2)
CLK_async	CLK_RTC	LFCLK	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 3-9. Peripheral Clock Sources

Peripheral	ATmega	ATtiny Series	MSPM0G	MSPM0L/C
RTC	CLK_async	CLK_RTC	LFCLK (LFOSC, LFXT)	N/A
UART	CLK_io	CLK_PER	BUSCLK, ULPCLK,MFCLK, LFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK
SPI	CLK_io	CLK_PER	BUSCLK, MFCLK, LFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK
I2C	CLK_io	CLK_PER	BUSCLK, MFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK
ADC	CLK_adc	CLK_PER	ULPCLK, HFCLK, SYSOSC	SYSOSC/ ULPCLK
TIMERS	CLK_io	CLK_PER	BUSCLK, MFCLK, LFCLK	BUSCLK, ULPCLK,MFCLK, LFCLK

Table 3-9. Peripheral Clock Sources (continued)

Peripheral	ATmega	ATtiny Series	MSPM0G	MSPM0L/C
LPTIM 1/2 (TIMG0/1)	CLK_async	CLK_PER	LFCLK, ULPCLK, LFCLK_IN	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

Microchip 8-bit AVR devices have similar operating modes. Table 3-10 gives a brief comparison between Microchip and MSPM0 devices.

Table 3-10. Operating Modes Comparison Between Microchip 8-bit AVR and MSPM0 Devices

Microchip ATmega Series		Microchip ATtiny Series		MSPM0		
Mode	**Description**	**Mode**	**Description**	**Mode**	**Description**	
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
Idle	CPU stopped but all peripherals remain enabled	Idle	CPU stopped but all peripherals remain enabled	Sleep		
ADC noise reduction	CPU disabled, only certain peripherals enabled	N/A	N/A			N/A
Standby	Same as Power Down except OSC remains enabled	STANDBY	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	0	Sleep 0 + PD1 disabled
					1	Sleep 1 + SYSOSC gear shifted to 4MHz
					2	Sleep 2 + ULPCLK limited to 32kHz

Table 3-10. Operating Modes Comparison Between Microchip 8-bit AVR and MSPM0 Devices (continued)

Microchip ATmega Series		Microchip ATtiny Series		MSPM0		
Mode	**Description**	**Mode**	**Description**	**Mode**	**Description**	
Power Save	Same as Power Down except Timer/Counter2 can be enabled	Power Down	BOD, WDT, and PIT (a component of the RTC) are active	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
Power Down	CPU stopped, no clocks	N/A	N/A	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 3-10, 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 still allow 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 and/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, GPIO, and RTC; receive SPI, UART, and I2C; or trigger DMA transfers and ADC conversions, while sleeping in the lowest power modes. For specific 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 Microchip 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 Microchip 8-bit AVRs 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 3-11. 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 Microchip 8-bit AVRs, 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 4 programmable priority levels are available on both families of devices.

Table 3-11. Interrupt Comparison

NVIC Number	MCHP		MSPM0x	
	Interrupt/Exception	Priority	Interrupt/Exception	Priority
-	Reset	Fixed: -3	Reset	Fixed: -3
-	NMI Handler	Fixed: -2	NMI Handler	Fixed: -2
-	Hard Fault Handler	Fixed: -1	Hard Fault Handler	Fixed: -1
-	SVCall Handler	Selectable	SVCall Handler	Selectable
-	PendSV	Selectable	PendSV	Selectable
-	SysTick	Selectable	SysTick	Selectable
0	Window Watchdog Interrupt	Selectable	INT_GROUP0: WWDT0, DEBUGSS, FLASHCTL, WUC FSUBx, and SYSCTL	Selectable
1	Power Voltage Detector Interrupt	Selectable	INT_GROUP1: GPIO0 and COMP0	Selectable
2	RTC and Timestamp	Selectable	Timer G1 (TIMG1)	Selectable
3	Flash Global Interrupt	Selectable	UART3(1)	Selectable
4	RCC Global Interrupt	Selectable	ADC0	Selectable
5	EXTI0 and EXTI1 interrupt	Selectable	ADC1(1)	Selectable
6	EXTI2 and EXTI3 interrupt	Selectable	CANFD0(1)	Selectable
7	EXTI4-EXTI15 interrupt	Selectable	DAC0(1)	Selectable
8	UCPD1/UCPD2/USB	Selectable	Reserved	Selectable
9	DMA1 Channel 1	Selectable	SPI0	Selectable
10	DMA1 Channel 2 and 3	Selectable	SPI1(1)	Selectable
11	DMA1 Channel 4-6, and DMA2 Channel 1-5	Selectable	Reserved	Selectable
12	ADC and Comparator	Selectable	Reserved	Selectable
13	Timer 1 (TIM1), Break, Update, Trigger, and Commutation	Selectable	UART1	Selectable
14	TIM1 Capture Compare	Selectable	UART2(1)	Selectable
15	TIM2 global interrupts	Selectable	UART0	Selectable
16	TIM3 and TIM4 global interrupts	Selectable	TIMG0	Selectable
17	TIM6, LPTIM1, and DAC interrupts	Selectable	TIMG10(1)	Selectable
18	TIM6 and LPTIM2 global interrupts	Selectable	TIMA0(1)	Selectable
19	TIM14 global interrupts	Selectable	TIMA1	Selectable
20	TIM15 global interrupts	Selectable	TIMA2(2)	Selectable
21	TIM16 and FDCAN0 global interrupts	Selectable	TIMH0(1)	Selectable
22	TIM17 and FDCAN1 global interrupts	Selectable	Reserved	Selectable
23	12C1 global interrupts	Selectable	Reserved	Selectable
24	I2C2 and I2C3 global interrupts	Selectable	I2C0	Selectable

Table 3-11. Interrupt Comparison (continued)

NVIC Number	MCHP		MSPM0x	
	Interrupt/Exception	Priority	Interrupt/Exception	Priority
25	SPI1 global interrupts	Selectable	I2C1	Selectable
26	SPI2 and SPI3 global interrupts	Selectable	Reserved	Selectable
27	USART1 global interrupts	Selectable	Reserved	Selectable
28	USART2 and LPUART2 global interrupts	Selectable	AES(1)	Selectable
29	USART 3-6 and LPUART1 global interrupts	Selectable	Reserved	Selectable
30	CEC global interrupts	Selectable	RTC(1)	Selectable
31	AES and RNG global interrupts	Selectable	DMA	Selectable

(1) Only available in MSPM0G family of devices.
(2) TIMG4 on MSPM0L family of devices

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

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.

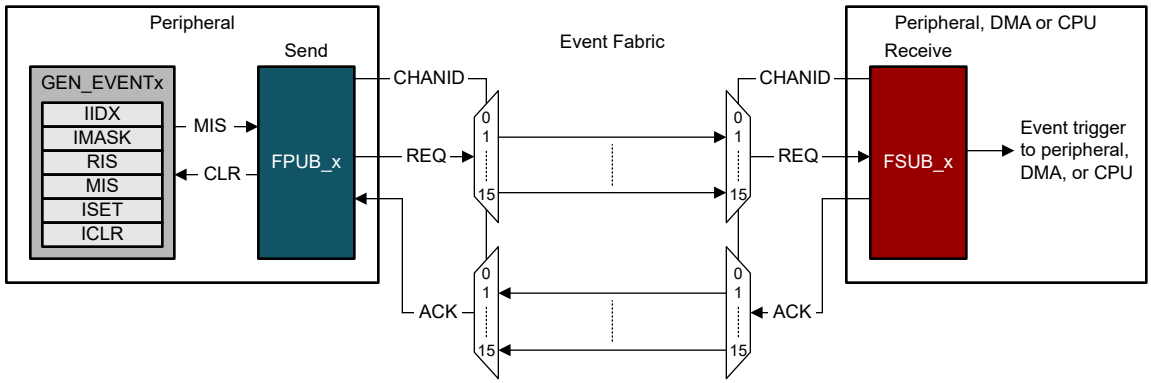

Figure 3-2. Generic Event Route

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 I^2C-to-UART bridge may previously have triggered a UART transmission upon receipt of an I^2C STOP, using an ISR to set a flag, or load the UART TX buffer directly. With the MSPM0 Event handler, the I^2C 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 MSPM0, while the Microchip 8-bit AVR devices is a one-wire Unified Program and Debug Interface (UPDI) . This interface is typically used during application development, and during production programming. Table 3-12 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 3-12. Program/Debug Interface Feature Comparison

	MCHP	MSPM0
Debug port	Arm SWD port (2-wire)	Arm SWD port (2-wire)
Break Point Unit (BPU)	Two hardware breakpoints; unlimited user software breakpoints	Four hardware breakpoints
Data Watch Unit (DWT)	Two watchpoints	Two watchpoints
Micro-Trace Buffer (MTB)	No	MTB support with 4 trace packets(1)
Low-power debug support	No	Yes
EnergyTrace support	No	EnergyTrace+ support (CPU states with power profiling)
Peripheral run support during debug	Yes	Yes
Debug interface locking	No	Can permanently disable debug capabilities, or can lock with password

(1) MSPM0Gxxxx devices only

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 3-13 shows a comparison of the different options and features between MSPM0 and Microchip ATmega and ATtiny device families.

Table 3-13. BSL Feature Comparison

BSL Features	Microchip ATmega Series	Microchip ATtiny Series	MSPM0
BSL started on blank device	Yes	No	Yes
Auto detection of programming interface	No	No	Yes
Security	Boot Lock bits ATmega88 and ATmega168 only	No	Secure boot options; CRC protections
Customizable	No	Yes	Yes, configurable invoke pin and plug-in feature
Invoke methods	Jump/call instruction	Jump/call instruction	1 pin high at BOOTRST, SW entry
Interfaces Supported			
UART	Yes	N/A	Yes
I2C	Yes	N/A	Yes
SPI	Yes(1)	N/A	Custom plug-in needed
CAN	Yes(1)	N/A	Plug-in planned(1)
USB	Yes(1)	N/A	No MSPM0 device with USB capability at this time.

(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 ATmega and ATtiny devices, with additional functionality. Microchip uses the term GPIO to refer to all the functionality responsible for managing the device pins. 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.

Together MSPM0 GPIO and IOMUX cover the same functionality as Microchip's GPIO. MSPM0 devices also offer a number of additional functions that are unavailable for Microchip ATmega and ATtiny devices.

Table 4-1. GPIO Feature Comparison

Feature	ATmega	ATtiny	MSPM0G, MSPM0L,MSPM0C
Output modes	Push-pull Open drain with pullup	Push-pull Open drain with pullup	Push-pull Open drain with pullup or pulldown
GPIO speed selection	Data unavailable	2.5ns rise time, 2.0ns fall time	ODIO pins: 120ns All others: 0.3*fmax = 3.75ns @ 80MHz
High-drive GPIO	Data unavailable	100mA combined per pin group	Equivalent, called High Drive IO (HDIO)
Input modes	Floating Pull-up Analog	Floating Pull-up Analog	Equivalent
Atomic bit set and reset	Yes	Yes	Equivalent
Alternate functions	Configured with configuration register	Configured with configuration register	Equivalent MSPM0 uses IOMUX
Wake-up	GPIO pin state change	GPIO pin state change	Equivalent
GPIO controlled by DMA	No	No	Yes
User controlled input filtering to reject glitches less than 1, 3, or 8 ULPCLK periods	No	No	Yes
User controllable input hysteresis	No	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)

Microchip's 8-bit MCUs and MSPM0 both offer peripherals to perform asynchronous (clockless) communication. In MSPM0 these UART peripherals come in two variants, one with standard features and one with advanced features. In Microchip's devices, the UART comes in one singular version. Table 4-2 shows a comparison of MSPM0's UART against Microchip's ATtiny and ATmega UARTs.

Table 4-2. UART Feature Set Comparison

Feature	ATmega	ATtiny	MSPM0
Hardware flow control	No	No	Yes
Continuous communication using DMA	No	No	Yes
Multiprocessor	No	No	Yes
Synchronous mode	Yes	Yes	No
Single-wire half duplex communication	Yes	Yes	Yes(1)
Wakeup from low-power mode	Yes	Yes	Yes
Data length	5, 6, 7, 8, 9	5, 6, 7, 8, 9	5, 6, 7, 8
Tx/Rx FIFO Depth	2	N/A	4
IrDA Support	No	Yes	Extended UART only
LIN Support	Client support on ATmega48/88 only	Client support	Extended UART only
DALI Support	No	No	Extended UART only
Manchester Code Support	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 Microchip 8-bit MCUs both support serial peripheral interface (SPI). Overall, MSPM0 and ATtiny/ ATmega SPI support is comparable with the difference listed in Table 4-3.

Table 4-3. SPI Feature Comparison

Feature	ATmega	ATtiny	MSPM0
Controller or peripheral operation	Yes	Yes	Yes
Data bit width (controller mode)	8 bits	8 bits	4 to 16 bit
Data bit width (peripheral mode)	8 bits	8 bits	7 to 16 bit
Maximum speed	8MHz	10MHz	MSPM0C: 12 MHz
			MSPM0L: 16MHz
			MSPM0G: 32MHz
Full-duplex transfers	Yes	Yes	Yes
Half-duplex transfer (bidirectional data line)	No	No	No
Simplex transfers (unidirectional data line)	Yes	Yes	Yes
Hardware chip select management	No	No	Yes
Programmable clock polarity and phase	Yes	Yes	Yes
Programmable data order with MSB-first or LSB-first shifting	Yes	Yes	Yes
SPI format support	No	No	Motorola, TI, MICROWIRE
Hardware CRC	No	No	No, MSPM0 offers SPI parity mode
TX FIFO depth	1	1	4
RX FIFO depth	1	2	4

SPI code examples

Information about SPI code examples can be found in the MSPM0 SDK examples guide .

4.4 I2C

MSPM0 and Microchip 8-bit MCUs both support I2C. In MSPM0, the I2C functionality is handled by the I2C module. In Microchip devices, this is slightly different. In ATmega169/329 devices, I2C is handled by the Universal Serial Interface. In the ATmega48/88, it is managed by the 2-Wire Serial Interface. In the ATtiny, the two wire interface or TWI handles I2C communications.

Table 4-4. I2C Feature Comparison

Feature	ATmega	ATtiny	MSPM0
Controller and target modes	Yes	Yes	Yes
Multi-controller capability	Yes	Yes	Yes
Standard-mode (up to 100 kHz)	Yes	Yes	Yes
Fast-mode (up to 400 kHz)	Yes	Yes	Yes
Fast-mode Plus (up to 1 MHz)	No	Yes	Yes
Addressing mode	7 bit	7 or 10 bit	7 or 10 bit
Peripheral addresses	1 Address	1 Address	2 addresses
General call	No for ATmega 169/329, yes for ATmega48/88	Yes	Yes
Programmable setup and hold times	No	No	No
Event management	No	No	Yes
Clock stretching	No for ATmega169/329, yes for ATmega48/88	Yes	Yes
Software reset	Yes	Yes	Yes
FIFO/Buffer	No	No	TX: 8 byte RX: 8 byte
DMA	No	No	Yes
Programmable analog and digital noise filters	N/A for Atmega169/329, Input filter not programmable for ATmega48/88	Input filter not programmable	Yes

I2C code examples

Information about I2C code examples can be found in the MSPM0 SDK examples guide.

4.5 Timers (TIMGx, TIMAx)

Microchip's ATmega and ATtiny and MSPM0 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	ATmega Timers	ATtiny Timers	MSPM0G Timers	MSPM0L and MSPM0C Timers
Resolution	16 bit	16 bit	16 bit, 32 bit	16 bit
PWM	Yes	Yes	Yes	Yes
Capture	Yes	Yes	Yes	Yes
Compare	Yes	Yes	Yes	Yes
One-shot	No	Yes	Yes	Yes
Up down count functionality	Yes	Yes	Yes	Yes
Power Modes	Yes	Yes	Yes	Yes
QEI support	No	No	Yes	No
Programmable pre-scalar	Yes	Yes	Yes	Yes
Shadow register mode	No	No	Yes	Yes
Events/Interrupt	Yes	Yes	Yes	Yes
Fault Event Mechanism	No	No	Yes	No
Auto reload functionality	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)

Microchip 8-bit MCUs and MSPM0 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 in time.

Table 4-6. WWDT Naming

Key	ATmega and ATtiny	MSPM0
Name	Watchdog Timer	Windowed watchdog timer
Abbreviated name (same order)	WDT	WWDT

Table 4-7. WDT Feature Comparison

Feature	ATmega	ATtiny	MSPM0
Window mode	No	Yes	Yes
Interval timer mode	No	No	Yes
LFCLK source	Yes	Yes	Yes
Interrupts	No	No	Yes
Counter resolution	14 bit for ATmega169/329, 11 bit for ATmega48/88	11 bit	25 bit
Clock divider	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 [1] and some of Microchip's 8-bit MCUs offer a real-time clock (RTC). Microchip's ATmega48/88 models, and ATtiny devices both contain RTC functionality, while the ATmega169/329 models do not. The ATmega48/88 models offer RTC functionality within one of their timer modules, while the ATtiny 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	ATmega48/88	ATtiny	MSPM0G
Power modes	Yes	Yes	Yes
Binary coded format	Yes	Yes	Yes
Leap year correction	No	No	Yes
Number of customizable alarms	1	1	2
Internal and External crystal	External	Yes	Yes
Crystal offset calibration	No	Yes	Yes
Prescaler blocks	Yes	Yes	Yes
Interrupts	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)

Microchip 8-bit AVR and MSPM0 both offer ADC peripherals to convert analog signals to a digital equivalent. Both device families feature either a 10-bit or 12-bit ADC. The following tables compare the different features and modes of the ADCs.

Table 5-1. Feature Set Comparison

Feature	Microchip ATMega	Microchip ATtiny	MSPM0
Resolution (Bits)	10	12	12/10/8
Conversion Rate (Msps) (12-bit)	15ksps	375ksps	MSPM0Gx - 4
			MSPM0Lx - 1.68
			MSPM0Cx - 1.5
Oversampling (Bits)	No	17	14
Hardware Oversampling	No	Yes	128x
FIFO	No	No	Yes
ADC Reference (V)	Internal: 1.1 AVCC	Internal: 1.024, 2.048, 2.500, 4.096 VDD	Internal:1.4,2.5 VDD
	External: $1.0 \le V_{REF} \le V_{DD}$	External: $1.0 \le V_{REF} \le V_{DD}$	External: $1.4 \le V_{REF} \le V_{DD}$
Operating Power Modes	Active, Idle, ADC noise reduction	Active, Idle, ADC noise reduction	Run, Sleep, Stop, Standby(1)
Auto Power Down	No	No	Yes
External Input Channels (2)	Up to 8	Up to 15	MSPM0Gx - up to 17
			MSPM0Lx/Cx up to 10
Internal Input Channels	Temperature Sensor, VREF, VBAT	No	Temperature Sensor, Supply Monitoring, Analog Signal Chain
DMA Support	Yes	No	Yes
ADC Window Comparator Unit	No	No	Yes

[1] Only MSPM0G devices support RTC.

Table 5-1. Feature Set Comparison (continued)

Feature	Microchip ATMega	Microchip ATtiny	MSPM0
Simultaneous Sampling	No	No	MSPM0Gx - Yes
			MSPM0Lx/Cx - No
Number of ADCs(3)	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	Microchip ATMega	Microchip ATtiny	MSPM0	Comments
Single Conversion Mode	YES	YES	Single Channel Single Conversion	ADC samples and converts a single channel once
Scan a Sequence of Channels	NO	NO	Sequence of Channels Conversion	ADC samples a sequence of channels and converts once.
Continuous Conversion Mode	YES	YES	Repeat Single Channel Conversion	Repeat single channel continuously samples and converts one channel
	NO	NO	Repeat Sequence of Channels Conversion	Samples and converts a sequence of channels then repeats the same sequence
Discontinuous Mode	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)

The Microchip 8-bit AVR MCU family and MSPM0 family of parts 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 ATmega family feature a single comparator with one pair of inputs, whereas the ATtiny family feature up to three pair of inputs. The MSPM0 comparator module provides a windowed comparator functionality using its DAC with two programmable levels. Both MSPM0 and AVR 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 AVR comparator modules compare feature-by-feature is included in Table 5-3.

Table 5-3. COMP Feature Set Comparison

Feature	ATmega	ATtiny	MSPM0G	MSPM0L
Available comparators	1	1	Up to 3	1
Number of positive and negative inputs	1 pos, 1 neg	Up to 4 pos, 3 neg	Up to 4 pos, 3 neg	Up to 2 pos, 2 neg
Output routing	No	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins
	Interrupt	Interrupt/Event Interface	Interrupt/Event Interface	Interrupt/Event Interface
Noninverting input sources	1 pos	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O Pins
			DAC12 output(1)	DAC8 output
			DAC8 output	OPA1 Output(2)
			Internal V_{REF}: 1.4 V and 2.5 V	
			OPA1 output(2)	
Inverting input sources	Multiplexed I/O Pins	Multiplexed I/O Pins	Multiplexed I/O pins	Multiplexed I/O Pins
	No	No	Internal temperature sensor	Internal temperature sensor
	Bandgap: 1.1V	Internal V_{REF}: 1.024V, 2.056V, 2.500, 4.096 and VDD	Internal V_{REF}: 1.4V and 2.5V	DAC8 output
	No	DAC8 output	DAC8 output	OPA0(3) output
	No	No	OPA0 output (3)	
Programmable hysteresis	No	None, small, medium, large	None, 10mV, 20mV, 30mV	None, 10mV, 20mV, 30mV
		No	Other values from 0V to V_{REF}/V_{DD} using DAC8	Other values from 0V to V_{DD} using DAC8
Register lock	No	No	Yes, some COMP registers (writes require key)	Yes, some COMP registers (writes require key)
Window comparator configuration	No	No	Yes	No (single COMP)
Input short mode	No	No	Yes	Yes
Operating modes	No	Active, low power	High speed, low power	High speed, low power
Fast PWM shutdowns	No	No	Yes (through TIMA fault handler)	No
Output filtering	No	No	Blanking filter	Blanking filter
			Adjustable analog filter	Adjustable analog filter
Output polarity control	No	Yes	Yes	Yes
Interrupts	Rising edge	Rising edge	Rising edge	Rising edge
	Falling edge	Falling edge	Falling edge	Falling edge
	Both edges	Both edges	Output ready	Output ready
Exchange inputs mode	No	No	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 Microchip AVR family of devices do not offer an integrated Digital-to-Analog converter (DAC) peripheral, but when migrating from AVR 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	MSPM0G
Resolution	12 bits (11 ENOB)
Output rate	1 MSPS
Output channels	1(1)
Data formats	8-bit right aligned, 12-bit right aligned, two's complement or straight binary
DMA integration	Yes
Output routing	External Pins
	Internal peripheral connections: OPA IN+, COMP IN+, ADC0
Internal reference voltage	Yes, 2.5V or 1.4V
External reference voltage	Yes
FIFO	Yes
Output buffer	Yes
Configurable output offset	Yes
Self-calibration mode	Yes
Automatic waveform generation	No
Sample and hold mode	No
Trigger sources	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 Microchip AVR family of devices do not offer an integrated Operational Amplifier (OPA) peripheral, but when migrating from AVR family 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 OPA code examples

Table 5-5. MSPM0 OPA Feature Set

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

The Microchip AVR and MSPM0 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	ATmega	ATtiny	MSPM0G	MSPM0L MSPM0C
Internal Reference (V)	1.1	1.024, 2.048, 2.500, 4.096, VDD	1.4, 2.5	1.4, 2.5
External Reference (V)	Up to VDD	Up to VDD	External: $1.4 \le V_{REF} \le V_{DD}$	External: $1.4 \le V_{REF} \le V_{DD}$
Output Internal Reference	No	No	Yes	Yes
Internally Connect to ADC	Yes	Yes	Yes	Yes
Internally Connect to DAC	N/A	N/A	Yes	No
Internally Connect to COMP	No	Yes	Yes	No
Internally Connect to OPA	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

- Texas Instruments: *MSPM0G350x Mixed-Signal Microcontrollers Data Sheet*
- Texas Instruments: *MSPM0L134x, MSPM0L130x Mixed-Signal Microcontrollers Data Sheet*
- Texas Instruments: *MSPM0C110x Mixed-Signal Microcontrollers Data Sheet*
- Texas Instruments: *Cybersecurity Enablers in MSPM0 MCUs*
- Texas Instruments: *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*
- Texas Instruments: *MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual*
- Texas Instruments: *MSPM0 C-Series 24-MHz Microcontrollers Technical Reference Manual*

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 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