AN3213

Getting Started with the SAMRH71 Microcontroller

INTRODUCTION

This application note is aimed at helping the reader become familiar with the radiation-hardened Microchip Arm® Cortex® M7-based SAMRH71 microcontroller.

It describes in detail a simple project, which use several important features present on the SAMRH71. These includes how to set up the microcontroller prior to executing the application, as well as how to add the functionalities themselves. After going through this document, the reader should be able to successfully start a new project.

This document also explains how to set up and use MPLAB®, for compilation, and execution of a software project.

To use this document efficiently, the reader should be experienced in using the Arm® core. For more information about the Arm core architecture, please refer to the relevant reference documents available from the Arm Limited website.

REQUIREMENTS

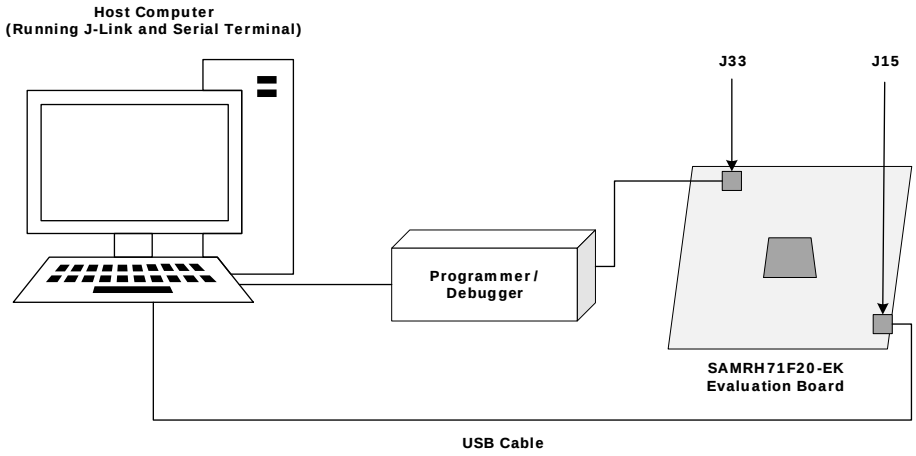
The cross-development environment is shown in the following figure.

The software referenced in this application note requires several components:

- SAMRH71F20-EK Evaluation Board
- Computer, running Microsoft Windows® 10 or later
- Microchip Programmer/Debugger Atmel-ICE (ATATMEL-ICE), J-32 Debug Probe (DV164232), ICD4 (DV164045), or PICkit4 (PG164140) with Debugger Adapter Board (AC102015)
- SEGGER J-Link software and documentation pack (version 6.50 or later)
- MPLAB (v5.35 or newer)

This document uses examples to guide you through setting up development environments for these tools.

SAMRH71 DEVELOPMENT ENVIRONMENT



TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@microchip.com**. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000000A is version A of document DS30000000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)

When contacting a sales office, please specify which device, revision of silicon and data sheet (include -literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com** to receive the most current information on all of our products.

Contents

1.0 GETTING STARTED

This section describes how to program a basic application that helps you to become familiar with the SAMRH71. It covers three main sections:

- the specification of the example (what it does, which peripherals are used);
- how to set up the hardware development environment;
- how to control relevant peripherals.

1.1 Features

The demonstration program makes an LED on the board turn on when the user presses the dedicated button. This switching takes place via an interrupt which is assigned to the push-button PB0.

While this software may appear basic, it uses several peripherals and as such it serves as a good starting point for familiarizing the user with the SAMRH71.

1.2 Peripherals

To perform the operations described in the previous section, the software example uses the following set of peripherals:

- Parallel Input/Output (PIO) controller
- Nested Vectored Interrupt Controller (NVIC)

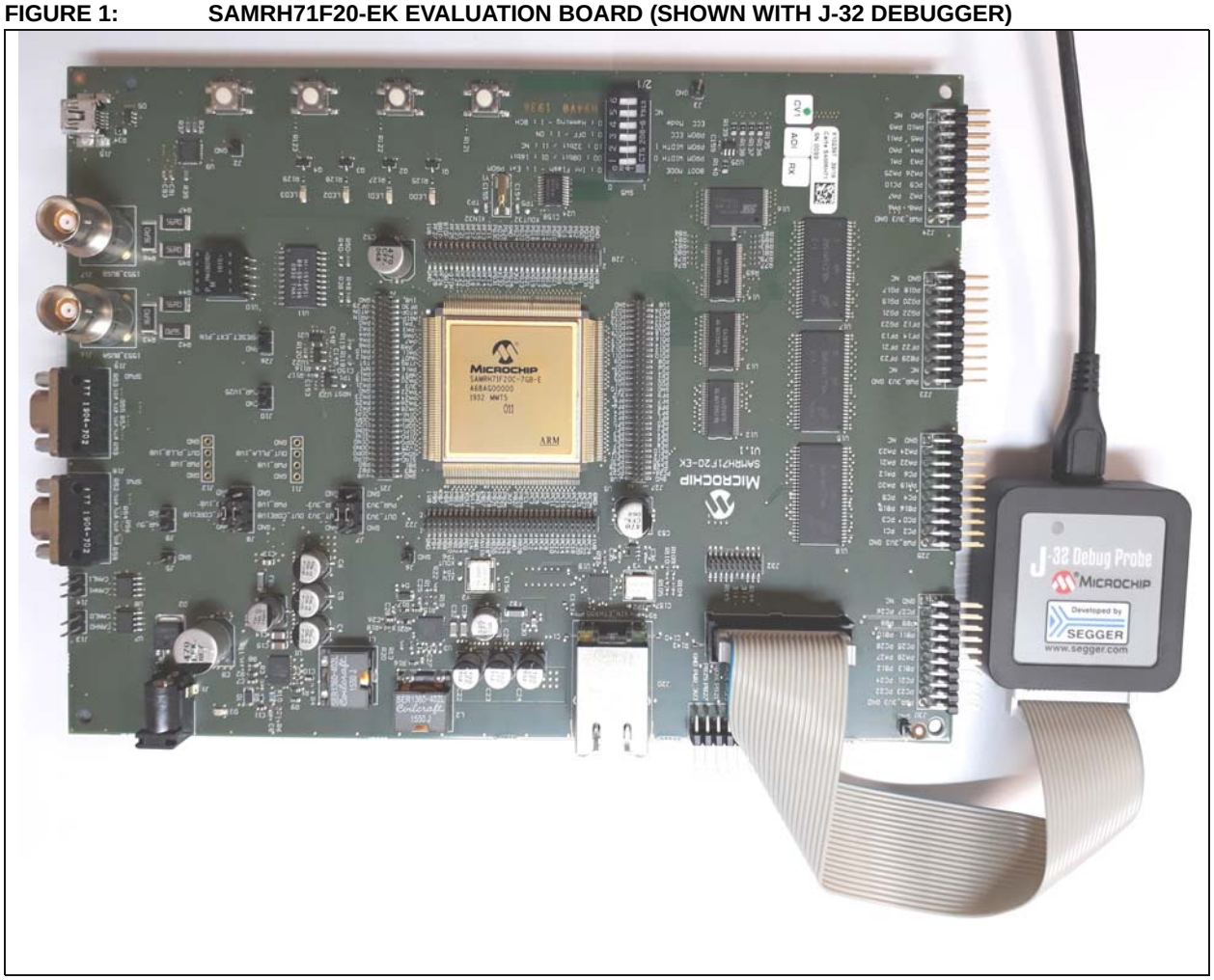
LEDs and buttons on the board are connected to standard input/output pins of the chip, which are managed by the chip's PIO controller. It is possible to have the controller generate an interrupt when the status of one of its pins changes, and one button pin is to have this behavior.

Using the NVIC is required to manage interrupts. It allows the configuration of a separate vector for each source. As can be seen in the code, a function is used to handle the PIO interrupt.

1.3 SAMRH71F20-EK Evaluation Board

The following image demonstrates an Evaluation Board.

FIGURE 1: SAMRH71F20-EK EVALUATION BOARD (SHOWN WITH J-32 DEBUGGER)



1.3.1 HARDWARE SETUP

To set up the hardware development environment.

1. Jumper and DIP switch settings: On the board's J7 header, verify that jumpers connect "DUT_3V3" to "PWR_3V3". On header J8, ensure that jumpers connect "DUT_CORE1V8" to "PWR_1V8". For this demonstration, set all of the switches of DIP switch SW5 to 0. (The default settings of SW5 are provided in Section 3.4.10 "DIP SWITCH" of the Evaluation Board User Guide.)
2. Power: Connect a 12V power supply to the board's J1 jack. While idle, the board draws just over 150 mA, but this is briefly exceeded when power is first applied. While executing the example code, the board may draw close to 350 mA.
3. Debugger tool connection: connect the debugger connector to the 20-pin connector J33 on the board, as shown in Figure 1.; connect the host PC to the debugger through a USB cable.
4. Serial port connection: A serial port is necessary for debug information display (and typed input) in the terminal window on the PC. For the required USB-to-UART Bridge, Virtual COM Port ("VCP") driver, visit the following link to Silicon Labs, and select the appropriate driver for your version of Windows: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers.
   Once installed, use the following COM settings:
   - 115200 baud rate
   - Eight bits of data
   - One stop bit
   - No parity
   - No flow control

Always apply an external reset following power-up, because there is no POR in the SAMRH71.

1.3.2 BOOTING

The SAMRH71 device features 128 Kbytes of embedded Flash and more than 1 Mbyte of internal SRAM. User applications can be compiled and downloaded to both memories; however, the example code support files have been prepared only for downloading into the Flash.

1.3.3 LEDs

There are four general-purpose LEDs (green): LED3 (PF20), LED2 (PF19), LED1 (PB23), and LED0 (PB19). These are toggled by driving their respective I/O lines. The example code toggles only LED0.

1.3.4 PUSH BUTTONS

There are three push buttons on the board, connected to I/O lines: PB2 (PC31), PB1 (PC30), and PB0 (PC29). The example code employs only PB0 for switching LED0 on and off.

1.4 Implementation

As stated previously, the examples mentioned above requires the use of several peripherals. It must also provide the necessary code for starting up the microcontroller. Both aspects are described in detail in this section, with commented source code when appropriate.

For filename references in this Implementation section, the command-line utilities "tree", "find", and "grep" are useful. These utilities are available in Linux, but also available for Windows through the Cygwin shell and MSL distribution shells (for example, Ubuntu shell for Windows).

See section Section 2.1, Get Started with MPLAB for the layout and explanation of example directories.

1.4.1 INITIALIZATION BEFORE `MAIN()`

Most of the code of an embedded application is written in C. This makes the program easier to understand, more portable and modular.

Prior to execution of application code, the following initialization should occur:

- Provide exception vectors
- Initialize critical peripherals
- Initialize memory segments

The overall sequence of events is shown in Example 1. The initialization requirements are described in the following sections.

EXAMPLE 1: RESET_HANDLER: NITIALIZATION PROCESS FLOW

```
Call optional appl-provided _on_reset()
function
Enable FPU if -mfloat-abi=softfp or -
mfloat-abi=hard
Initialize TCM ECC
Initialize FlexRAM ECC
Initialize data
Set vector-table base address in FLASH
Initialize C library
Enable Instruction Cache
Enable Data Cache
Call optional appl-provided _on_bootstrap()
function
Branch to main();
```

1.4.1.1 Entry Point

For MPLAB, the PC points to the start address of `Reset_Handler` at the beginning.

The purpose of the entry point is to:

- Set up a C environment
- Set the vector table base address
- Perform the low-level initialization
- Jump to the main application

1.4.1.2 Low-Level Initialization

The first step of the low-level initialization process is to configure critical peripherals:

- Main oscillator and its PLL
- MPU
- TCM

Among the files bundled with each SAMRH71 example in the Harmony Framework is a file named `startup_xc32.c`, under the example's `src/config/sam_rh71_ek/ directory`. This function steps through each of the actions listed in Example 1.

Also in the directory, `src/config/sam_rh71_ek/`, is file `initialization.c`, which accounts for:

- Identifies the main RC clock
- Configures PLL
- Configures MCK
- Disables WDT
- Configures SUPC
- Configures PMC
- Configures PIO

The following sections explain why these peripherals are considered critical, and detail the required operations to configure them properly.

1.4.1.3 Low-Level Initialization: `SYS_Initialize`

`SYS_Initialize` is found in file `src/config/sam_rh71_ek/initialization.c`.

The purpose of `SYS_Initialize` is to configure clocks and other peripherals. The first initialization involves the clocks.

Clock initialization occurs via function "CLK_Initialize", which calls the following functions (all contained in file "plib_clk.c"):

- `CLK_SlowClockInitialize();`
- `CLK_MainClockInitialize();`
- `CLK_PLLxClockInitialize();`
- `CLK_MasterClockInitialize();`
- `CLK_PeripheralClockInitialize();`

After reset, the Main RC oscillator is enabled with the 10 MHz frequency selected and it is selected as the source of MAINCK. MAINCK is the default clock selected to start the system.

The Main oscillator and its Phase Lock Loop A (PLLA) must be configured in order to run at full speed. Both can be configured in the Power Management Controller (PMC). For details, refer to the SAMRH71 data sheet.

In the example, the processor clock and master clock are 100 MHz and 50 MHz respectively, by default. Example values for the SAMRH71F20-EK Evaluation Board (10 MHz crystal) follow.

```
PMC_REGS->CKGR_PLLAR = CKGR_PLLAR_ONE_Msk |
CKGR_PLLAR_FREQ_VCO_VCO0 |
CKGR_PLLAR_PLLACOUNT(0x3f) |
CKGR_PLLAR_MULA(9) |
CKGR_PLLAR_DIVA(1);
```

1.4.1.4 Low-Level Initialization: Memory Protection Unit (MPU)

The SAMRH71 devices supply MPU with 16 zones as a component for memory protection. Users can use the MPU to enforce privilege rules, separate processes and enforce access rules.

The `MPU_Initialize` function (Example 2) completes the memory mapping by setting MPU Region Base Address Register (RBAR) and MPU Region Attribute and Size Register (RASR).

Within the `HarmonyFramework\csp\apps\` directory, see example "mpu" for more information. In that example, the `MPU_Initialize` function resides in file `plib_mpu.c`.

The MPU_RASR.ATTRS field defines the memory type, the cacheable and shareable properties, and the access and privilege properties of the memory region.

The System Handler Control and State Register is settled to enable memory management fault, bus fault, and usage fault exception.

At the end of the function, MPU region is enabled by setting MPU Control register.

In the example, memory regions such as ITCM, internal Flash, DTCM, SRAM, peripheral memory, and SDRAM are all configured in this function. The SRAM, for example, is divided into two parts with the same attributes.

The user can configure a new memory region or adjust the attributes of some regions in the function, such as the cacheable properties.

1.4.1.5 Low-Level Initialization: Tightly Coupled Memory (TCM)

The SAMRH71 device embeds Tightly Coupled Memory (TCM) running at processor speed. Within `HarmonyFramework\csp\apps\` directory see example "tcm" for further information. In that example, function `TCM_Enable` and other TCM functions reside in file `startup_xc32.c`.

ITCM is a single 64-bit interface, based at 0x0000 0000 (code region) and DTCM is composed of dual 32-bit interfaces interleaved, based at 0x2000 0000 (data region).

ITCM is disabled by default at reset. DTCM is enabled by default at reset with a fixed size of 256 Kbytes. When enabled, ITCM is located at 0x0000 0000, overlapping Flash. The Memory Protection Unit (MPU) can be used to protect these areas as mentioned in the section **"Low-Level Initialization: Memory Protection Unit (MPU)"**. The ITCM and DTCM are both ECC protected. By default, ECC is enabled. Before enabling ITCM and DTCM, all the data shall be aligned with their ECC in memory. For more details, refer to application note dedicated to TCM.

After carrying out all of the above initialization actions, the program can jump to the main application.

EXAMPLE 2: `MPU_Initialize` GENERIC PERIPHERAL USAGE

```
void MPU_Initialize(void)
{
    /*** Disable MPU            ***/
    MPU->CTRL = 0;
    /*** Configure MPU Regions  ***/
    /* Region 0 Name: region_nocache, Base Address: 0x21010000, Size: 1KB  */
    MPU->RBAR = MPU_REGION(0, 0x21010000);
    MPU->RASR = MPU_REGION_SIZE(9) | MPU_RASR_AP(MPU_RASR_AP_READWRITE_Val) | MPU_ATTR_NORMAL \
                | MPU_ATTR_ENABLE  ;
    /* Enable Memory Management Fault */
    SCB->SHCSR |= (SCB_SHCSR_MEMFAULTENA_Msk);
    /* Enable MPU */
    MPU->CTRL = MPU_CTRL_ENABLE_Msk  | MPU_CTRL_PRIVDEFENA_Msk;
    __DSB();
    __ISB();
}
```

1.4.1.6 Initialization

Most peripherals are initialized by performing the following actions:

- Disabling or reprogramming watchdog
- Disabling or flushing cache if necessary
- Configuring peripheral
- Selecting and enabling peripheral clock
- Enabling peripheral interrupt
- Re-enabling cache if necessary

1.4.2 DISABLING OR REPROGRAMMING WATCHDOG TIMERS (WDT RSWDT)

1.4.2.1 Purpose

The Watchdog Timer (WDT) is used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 32-bit down counter that allows a watchdog period of up to 16 seconds (slow clock around 32 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in Debug mode.

After a processor reset, the value of Watchdog Counter Value (WDV) is the maximum value of the counter with the external reset generation enabled (bit WDT_MR_WDRSTEN_Msk at 1 after a backup reset). This means that a default watchdog is running at reset, i.e., at power-up. The user can either disable the WDT by setting bit RSWDT_MR_WDDIS_Msk or reprogram the WDT to meet the maximum watchdog period the application requires.

1.4.2.2 Initialization

In the example, the user can disable the watchdog timer as follows:

```
WDT_REGS->WDT_MR = WDT_MR_WDDIS_Msk;
```

1.4.3 ENABLING CACHE IF NECESSARY

1.4.3.1 Purpose

The SAMRH71 device supports 16 Kbytes of ICache and 16 Kbytes of DCache with Error Code Correction (ECC). All caches are disabled at reset and enabling cache benefits the performance. The user can turn on I-Cache and D-Cache if necessary.

1.4.3.2 Initialization

The user can enable cache as follows:

```
SCB_EnableICache();
SCB_EnableDCache();
```

The details of the `SCB_EnableICache` function is shown in the following example.

```
__DSB();
__ISB();
```

```
SCB->ICIALLU = 0UL;
__DSB();
__ISB();
SCB->CCR |= (uint32_t)SCB_CCR_IC_Msk;
__DSB();
__ISB();
```

The Data Synchronization Barrier (DSB) and Instruction Synchronization Barrier (ISB) instructions flush data and instruction cache, respectively. To make some regions cacheable, do the following:

- Enable cache as described above in the application.
- Set the attributes of the relevant regions as cacheable in `MPU_Initialize` function (refer to the section **"Low-Level Initialization: Memory Protection Unit (MPU)"** for more information).

All the cache-related functions are contained in file `core_cm7.h`, within the `packs` subdirectory tree, under any example's `src` directory

1.4.4 USING THE NESTED VECTORED INTERRUPT CONTROLLER (NVIC)

1.4.4.1 Purpose

The NVIC provides configurable interrupt handling abilities to the processor. It facilitates low-latency exception and interrupt handling, and controls power management.

The NVIC supports up to 240 interrupts, each with up to 256 levels of priority. The user can change the priority of an interrupt dynamically. The NVIC and the processor core interface are closely coupled, to enable low-latency Interrupt processing and efficient processing of late arriving interrupts. The NVIC maintains knowledge of the stacked, or nested interrupts to enable tail-chaining of interrupts.

1.4.4.2 Initialization

The SAMRH71 uses hardware to save and restore key context state on exception entry and exit, and uses a table of vectors to indicate the exception entry points.

The vector table contains the initialization values for the stack pointer, and the entry point addresses of each exception handler. The vector table is defined as the constant of '*exception_table'* for GNU toolchain.

In file `interrupts.c.`, part of the constant is shown as follows:

```
__attribute__ ((section(".vectors")))
const DeviceVectors exception_table=
{
    /* Configure Initial Stack Pointer, using
linker-generated symbols */
    .pvStack = (void*) (&_stack),
    .pfnReset_Handler              = ( void *)
Reset_Handler,
    .pfnNonMaskableInt_Handler     = ( void *)
NonMaskableInt_Handler,
    .pfnHardFault_Handler          = ( void *)
HardFault_Handler,
.....
.pfnSysTick_Handler            = ( void * )
SysTick_Handler,
.....
.pfnTC0_CH0_Handler            = ( void * )
TC0_CH0_Handler,
.....
}
```

On reset, the processor initializes the vector table base address to an address 0x00000000. The software can find the current location of the table, or relocate the table, using the Vector Table Offset Register (VTOR) as shown below (see file startup_xc32.c).

```
pSrc = (uint32_t *) & __svectors;
SCB->VTOR = ((uint32_t) pSrc &
SCB_VTOR_TBLOFF_Msk);
```

The `__svectors` symbol points to the vectors section which saves the vectors table. The SCB_VTOR_TBLOFF_Msk is equal to 0xFFF FFF8 on SAMRH71 and bits [6:0] are RAZ (Read as Zero).

The VTOR holds the vector table address.

The processor and the NVIC prioritize and handle all exceptions. When handling exceptions, all exceptions are handled in Handler mode, and processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The vector is fetched in parallel to the state saving, enabling efficient interrupt entry.

Configuring an interrupt source requires six steps:

1. Implement interrupt handler if necessary.
   The first step is to re-implement the interrupt handler with the same name as the default interrupt handler in the vector table as just mentioned if necessary, so that when the corresponding interrupt occurs, the reimplemented interrupt handler is executed instead of the default interrupt handler.
2. Disable the interrupt if it was enabled.
   An interrupt triggering before its initialization completion may result in unpredictable behavior of the system. To disable the interrupt, the Interrupt Clear-Enable Register (ICER) of the NVIC must be written with the interrupt source ID to mask it. The following interface can be used directly:

   ```
   static inline void
   NVIC_DisableIRQ__STATIC_INLINE void
   __NVIC_DisableIRQ(IRQn_Type
   IRQn)(IRQn_Type IRQn);
   ```

3. Clear any pending interrupt.
   Setting the Interrupt Clear-Pending Register bit puts the corresponding pending interrupt in the

inactive state. It is also written with the interrupt source ID to mask it. The following interface can be used directly:

```
__STATIC_INLINE void
__NVIC_ClearPendingIRQ(IRQn_Type
IRQn)
```

4. Configure the interrupt priority.
   NVIC interrupts are prioritized by updating an 8-bit field within a 32-bit register (each register supporting four interrupts). Priorities are maintained according to the Armv7-M prioritization scheme. The following interface can be used directly:

   ```
   __STATIC_INLINE void
   __NVIC_SetPriority(IRQn_Type  IRQn,
   uint32_t priority)
   ```

5. Enable the interrupt at peripheral level.
6. Enable the interrupt at NVIC level.
   The interrupt source can be enabled, both on the peripheral (in a mode register usually) and in the Interrupt Set-Enable Register (ISER) of the NVIC. On the side of NVIC, the following interface can be called directly:

   ```
   __STATIC_INLINE void
   __NVIC_EnableIRQ(IRQn_Type IRQn)
   ```

Refer to `core_cm7.h` for more interfaces about NVIC which can be used directly.

1.4.5 USING THE PARALLEL INPUT/ OUTPUT CONTROLLER (PIO)

1.4.5.1 Purpose

The SAMRH71 supports several PIO controllers, each one controlling up to 32 lines. Each line can be assigned to GPIO or one of the four peripheral functions: A, B, C, or D.

In this example, the PIO controller manages single LED.

1.4.5.2 Configuring LEDs

The PIO connected to an LED must be configured as output, in order to turn it on or off. First, the PIOs control must be enabled in PIO Enable Register (PIO_PER).

PIO direction is controlled by two registers: Output Enable Register (PIO_OER) and Output Disable Register (PIO_ODR).

Note that there are individual internal pull-ups on each PIO pin. These pull-ups are enabled by default. Since they are useless for driving LEDs, they should be disabled, as this reduces power consumption. This is done in the PIO Pull-Up Disable Register (PIO_PUDR).

In this example, LED is wired to pins PB19. PB19 is represented by left-shift-by-19 operations in `PIOB_REGS` arguments, in file `plib_pio.h`.

```
/* Port B Peripheral function GPIO
configuration */
        PIOB_REGS->PIO_MSKR = 0x880000;
        PIOB_REGS->PIO_CFGR = 0x0;

 /* Port B Pin 19 configuration */
        PIOB_REGS->PIO_MSKR = 0x80000;
        PIOB_REGS->PIO_CFGR = (PIOB_REGS-
>PIO_CFGR & (PIO_CFGR_FUNC_Msk)) | 0x100;
```

When programming the PIO, it is recommended to use the Harmony Configurator; see Section 2.1, Get Started with MPLAB

1.4.5.3 Controlling LEDs.

LEDs are turned on or off by changing the level on the PIOs to which they are connected. After those PIOs have been configured, their output values can be changed by writing the pin IDs in the PIO Set Output Data Register (PIO_SODR) and the PIO Clear Output Data Register (PIO_CODR).

In addition, the PIO Pin Data Status Register (PIO_PDSR) indicates the current level on each pin. It can be used to create a toggle function, that is, when the LED is ON according to PIO_PDSR, then it is turned off, and vice-versa.

Refer to `plib_pio.c` or `plib_pio.h` for more interfaces that can be used directly.

2.0 RUNNING THE EXAMPLES

The required software tools for building the project and downloading the binary files are introduced in the **"Requirements"** section. The hardware development environment setup is described in **"Hardware Setup"**.

2.1 Get Started with MPLAB

MPLAB® X Integrated Development Environment (IDE) is a program that allows the user to configure, develop, and debug embedded designs for most of Microchip's microcontrollers (and DSPs). MPLAB relies on the XC8, XC16, and XC32 compilers, which must be enabled.

Integrated within MPLAB as a plugin, the MPLAB® Code Configurator (MCC) generates C code that is inserted into a project. MCC configures and enables a microcontroller's peripherals.

MPLAB® Harmony 3 is an extension of the MPLAB® ecosystem, for creating embedded firmware solutions for 32-bit Microchip devices. The MPLAB Harmony Configurator (for use with MPLAB Harmony v3) supports configuration and code generation for all MPLAB Harmony components (similar in functionality to the https://start.atmel.com/ tool for Atmel Studio).

The following sections focus only on what is needed from MPLAB and Harmony for the immediate requirements. For more information on these programs, see the documents listed in the **Section "References"** "References" section.

2.1.1 REQUIREMENTS

This section assumes the use of Windows; for Linux, MPLAB appears the same, but any references in this section to Windows tools, procedures, and directories differ. Note also that the steps described in Section 1.3.1, Hardware Setup must have been already executed, including step 3, "Debugger tool connection".

- Ensure that MPLAB v5.35 or later is installed. The installation process prompts for installation of the X compilers (XC8, XC16, XC32); the XC32 compiler should be selected, as it is used for all SAMRH71 projects. The installation process may also prompt for installation of the MPLAB Code Configurator, "MCC"; this should also be selected, as the following examples relies on MCC.
- When the MPLAB installation is completed:

a) Open MPLAB and from its top menubar, select **Tools** > **Plugins**.

b) In the resulting pop-up window, select the **Available Plugins** tab.

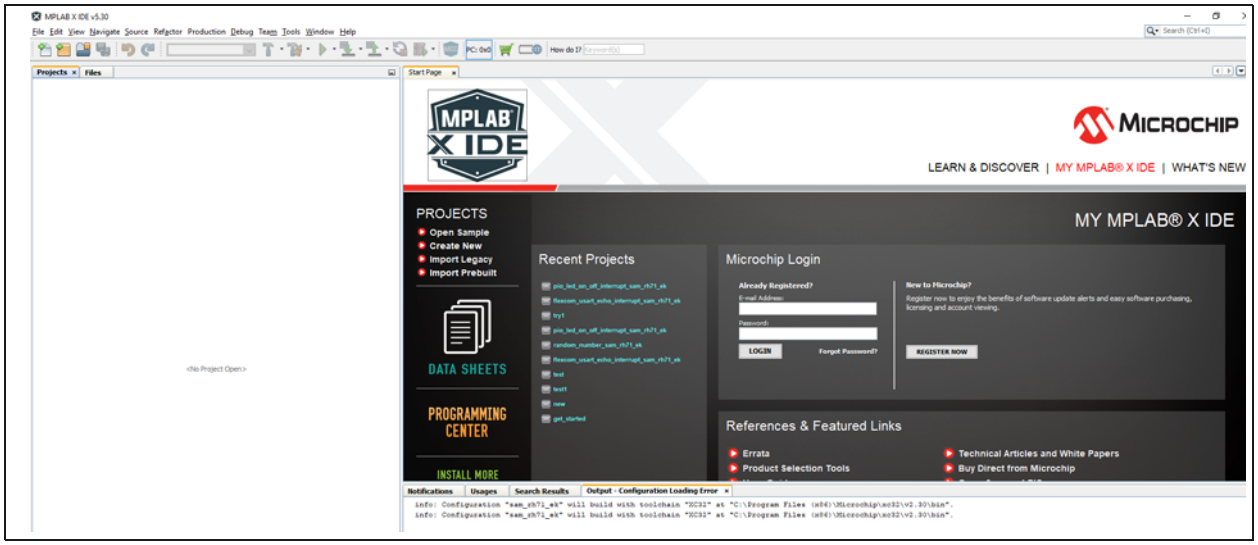
c) Select **Harmony Configurator 3** and click **Activate**. You can repeat this procedure at any time, for any of the plugins and their updates; you can also view installed plugins and deactivate them if desired.

2.1.2 LOAD EXAMPLE "PIO" IN MPLAB

Launch MPLAB and from the menubar select **File** > **Open** Project (or click the **Open Project** button, third from the left, under the menubar). By default, the HarmonyFramework" directory resides in your home directory; navigate to select file `HarmonyFramework\csp\apps\pio\pio_led_on_off_interrupt\firmwaresam_rh71_ek.X`

After the first opening of a project, in subsequent MPLAB sessions, the project is listed as a **Recent Project** in MPLAB's **Start Page** window; by default this is the upper-right window within MPLAB's main window. The project can then be opened simply by clicking on it within that **Recent Projects** list:

FIGURE 2: PROJECT VIEW IN MPLAB START PAGE



In the process of navigating to project sam_rh71_ek.X, one has to traverse many directories. Among the entries in the top-most directory, HarmonyFramework, the focus is on the csp directory, because almost all SAMRH71 examples reside under csp.

Harmony includes many examples, which are shared among various microcontrollers; to find all examples that works for the SAMRH71, execute a recursive search in HarmonyFramework for sam_rh71_ek. Windows search results in File Explorer may yield mixed results; if you have Cygwin or another command shell with Linux/UNIX commands, then in directory HarmonyFramework, try find . -name sam_rh71_ek, to find ~35 examples, most of which reside under csp.

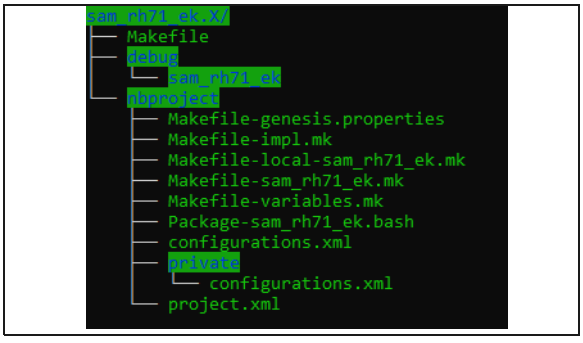
In directory csp, various files and directories for documentation, licensing, and so on are found. One of these directories, apps contains the examples that is going to be considered here, and in apps there are ~75 categories of examples. Only some of these categories currently reflect the SAMRH71; that is, a search for sam_rh71_ek yields about 25 names under this apps directory.

The pio example has just been opened (per the earlier step). The pio directory (in apps) contains two examples:

- pio_led_on_off_interrupt
- pio_led_on_off_polling

The pio_led_on_off_interrupt has been opened. The pio_led_on_off_interrupt directory contains a subdirectory firmware, in which there is an MPLAB X project directory for each of a few different microcontrollers, and in some cases a microcontroller may also have a project directory for the "IAR" IDE. Prior to building the pio project, use the tree shell command to locate the following contents in our project directory, pio_led_on_off_interrupt/firmware/sam_rh71_ek.X/:

FIGURE 3: SAM_RH71_EK CONTENT



In addition to these project files, the pio_led_on_off_interrupt/firmware/ directory has a src directory, which contains main_rh71.c, and the following directories:

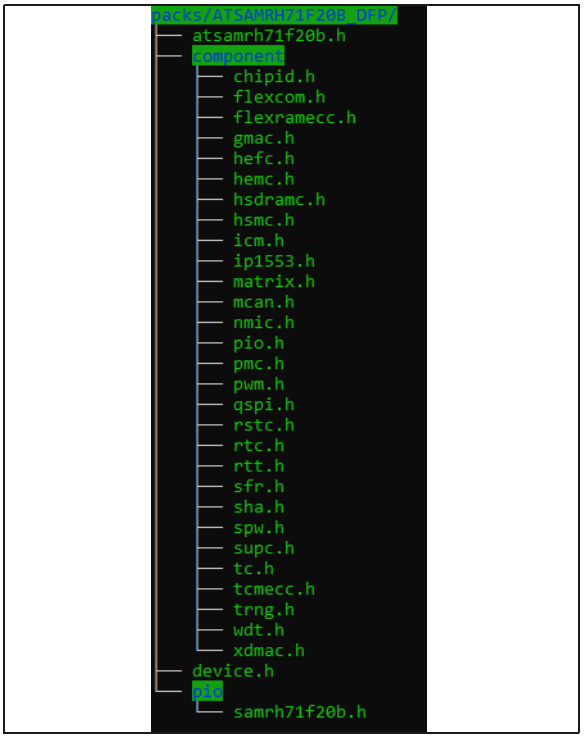
- arm: toolchain-specific header file(s)
- config: for each micro, a subdirectory with .c and .h configuration files
- packs: for each micro, a subdirectory with header files for that micro's peripherals

If one adds source code, it resides in src directory. If one adds or removes SAMRH71 peripherals to one's project, these changes are reflected in directory config, which initially looks like the following image.

FIGURE 4: CONFIG DIRECTORY CONTENTS

When one builds a project, all the build-related files appear under the sam_rh71_ek.X directory. Harmony may also update files under the src/ directory, including peripheral header files. In any case, all peripheral headers will remain present, as shown in the following image.

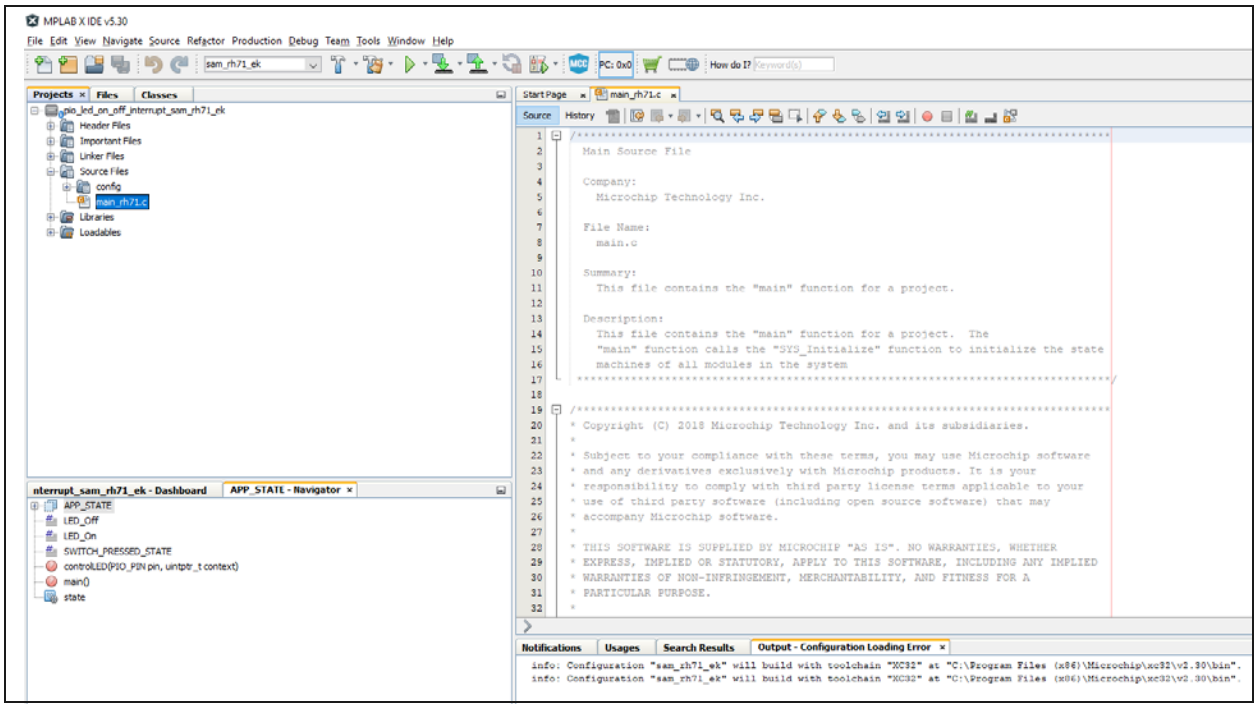
FIGURE 5: SAMRH71 PERIPHERAL HEADER FILES



With the project, `pio`, opened in MPLAB, notice its entry in the **Project** tab of the upper-left window. The other two tabs for this window are **Files** and **Classes**. The **Project** tab presents the files hierarchically of the project, by categories which are intuitive (for example. header files, linker files, source files), independent of the filesystem layout. The **Files** tab also presents the project files hierarchically, but according to their locations in the filesystem (as seen in the above "tree" images). The **Classes** tab presents a listing of the classes, functions, and datatypes of the project.

With the **Projects** tab selected, click the **+** icon to its left; this expands the content view to include the various kinds of files, which comprise the project (for example, header, linker, source). Click the **+** icon to the left of **Source Files** to expand the content view, and double-click **`main_rh71.c`**; this opens a new tab in MPLAB's upper-right window:

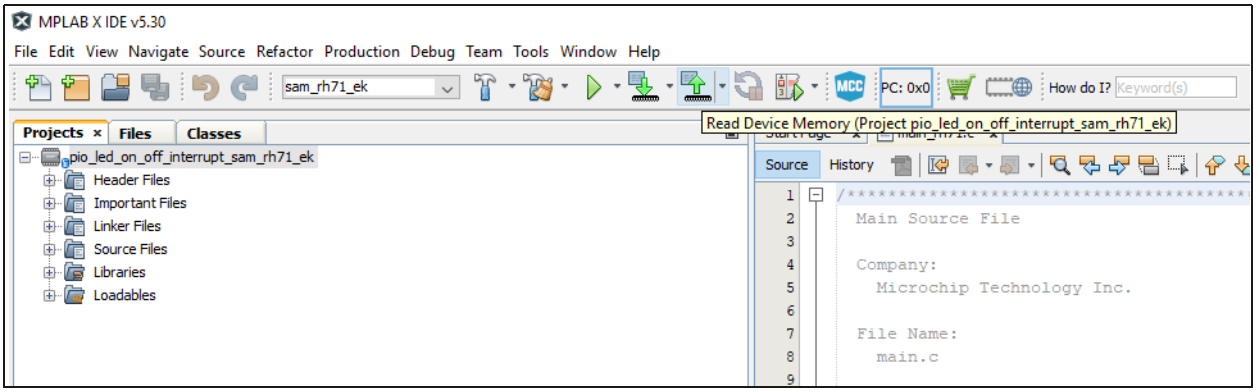
FIGURE 6: CONTINUE AND STOP DEBUGGING BUTTONS



Read the memory content, before you build the project. To do that, perform the following steps.

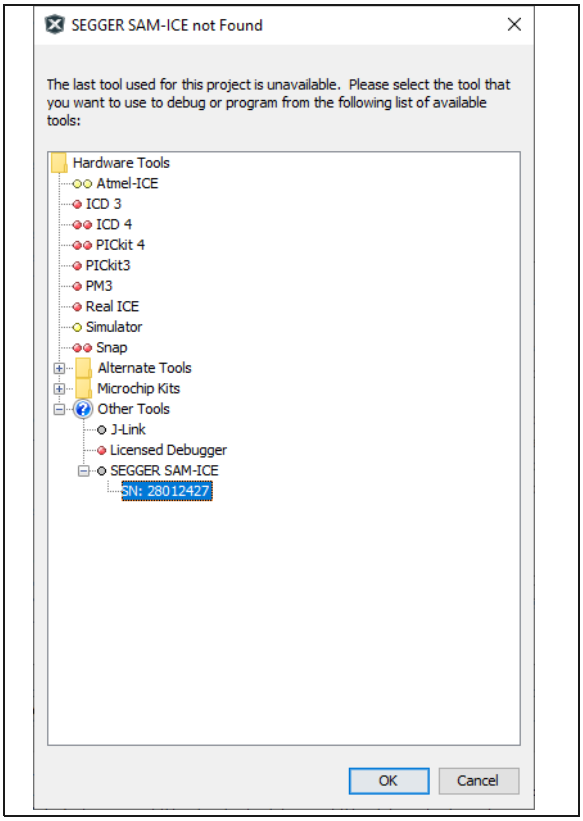
1. Below the main menu bar, click the little green up-arrow. You can hover the mouse over the icon to see the function of the icon. The requirement is to **Read Device Memory**.

FIGURE 7: READ MEMORY CONTENT



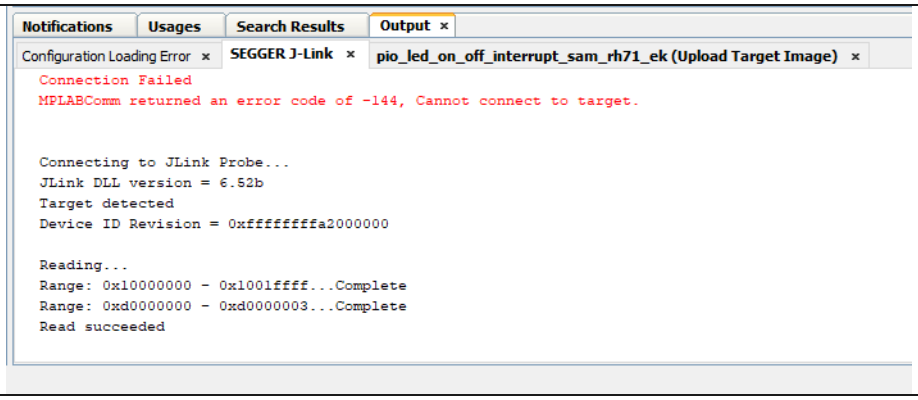
The first attempt to read the device might produce a pop-up window like the following image.

FIGURE 8: DEBUGGER NOT FOUND



2. Select the entry corresponding to your programmer/debugger; in the above case, it is the `S/N:` entry just below `SEGGER SAM-ICE`. For some older, SEGGER-based debuggers such as SAM-ICE.
3. In MPLAB, click the little green up-arrow (**Read Device Memory**), and the results appear in the notification window.

FIGURE 9: MEMORY STATUS



2.1.3 BUILD EXAMPLE "PIO" IN MPLAB

To build the project, perform the following steps.

1. In the top menu bar (below **Team**), click the hammer icon. A successful build result appears as shown in **FIGURE 11: "SUCCESSFUL BUILD MESSAGE"**.

FIGURE 10: BUILD PROJECT

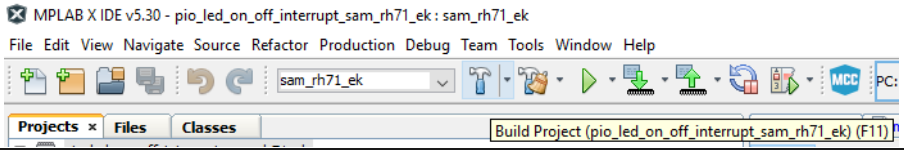
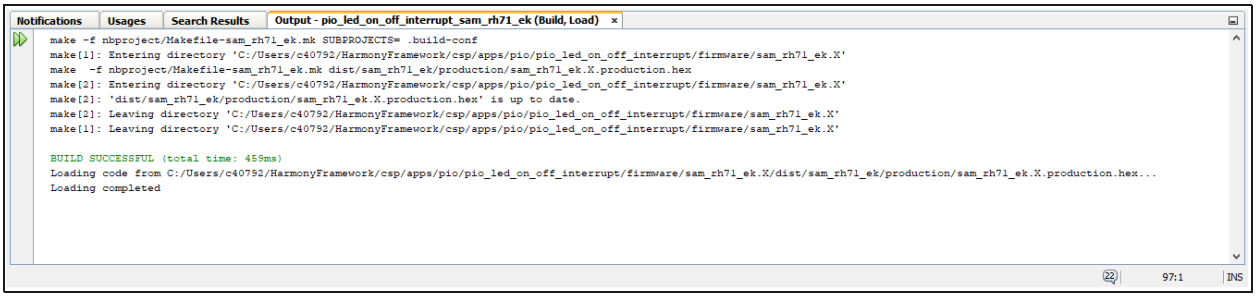


FIGURE 11: SUCCESSFUL BUILD MESSAGE



2.1.4 PROGRAM EXAMPLE "PIO" INTO DEVICE

To program the device, perform the following step.

1. To the left of the little green up arrow, click the little green down arrow to make the project and program the device. On successful programming, the following message should appear.

FIGURE 12: SUCCESSFUL PROGRAM MESSAGE

On the board, following successful programming, you should see **LED0** off, and the other three LEDs on. LED0 remains on for as long as one presses push-button PB0, based on an interrupt (there is another "pio" example, which relies on polling). If you have the PuTTY (or other) serial terminal open, there should be no activity therein, because this pio example does not use serial communication.

In file, `config/sam_rh71_ek/peripheral/pio/plib_pio.h` (shown in earlier directory image), the references to both the LED pin and the SWITCH ("PB") pin is found. The following instances should not be changed in our files. In MPLAB, such changes are made using the Harmony GUI.

- LED from PIO_PIN_PB19 to PIO_PIN_23 (LED1 on the board)
- SWITCH push-button from PIO_PIN_PC29 to PIO_PIN_PC30 ("PB1" on the board)

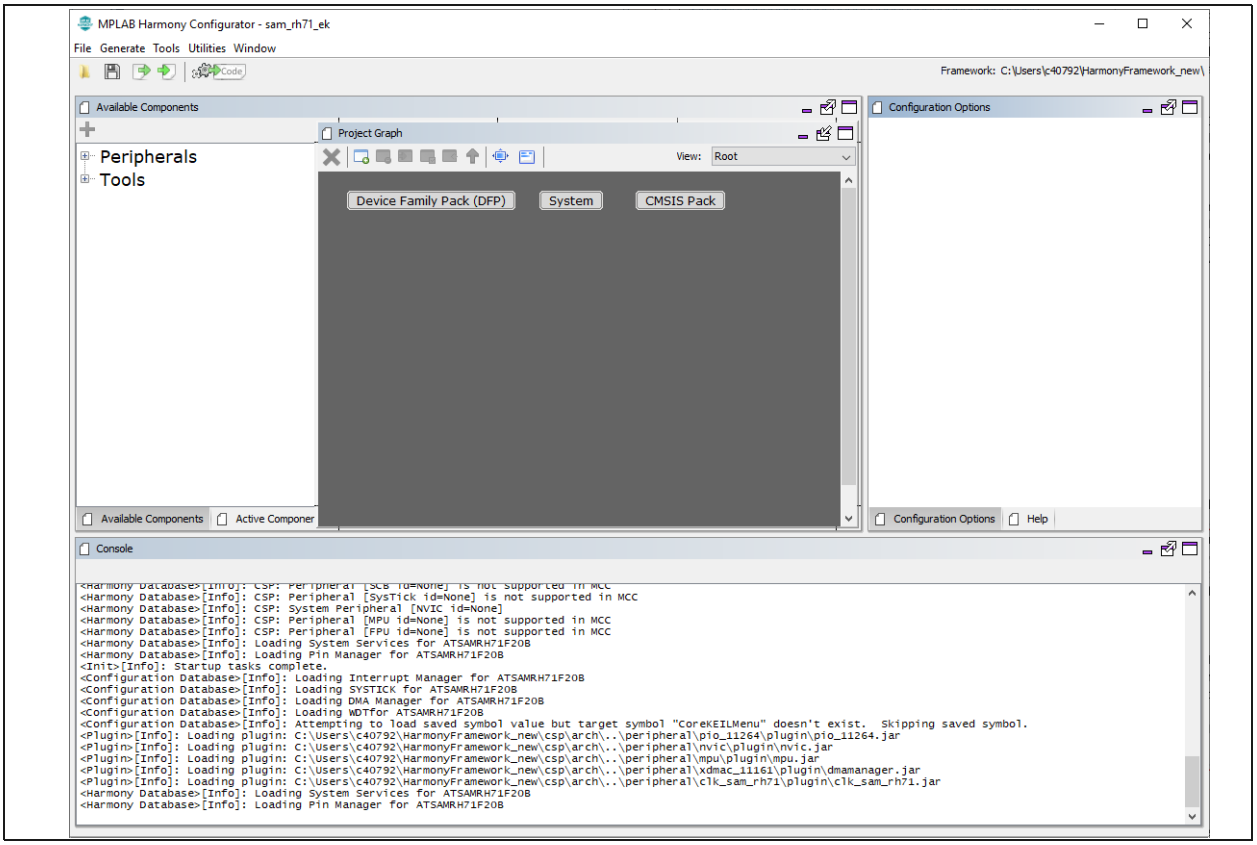
2.1.5 MODIFY "PIO" FUNCTIONALITY IN HARMONY

To change the PIO project functionality, perform the following steps.

2.1.5.1 Opening Save State File

1. With the PIO project still open, in MPLAB's menubar select **Tools** > **Embedded** > **MPLAB Harmony 3 Configurator**.
2. In the resulting **MPLAB Harmony Launcher** window, accept the default values for project paths and launch actions, by simply clicking **Launch**.
3. In the subsequent **Configuration Database Setup** window, accept the default values for DFP and CMSIS, by clicking **Launch**.
4. As it loads, Harmony might prompt for selection of desired packages (for example, **csp**, **net**, **usb**). For the example, **csp** is used.
5. If prompted to open a default **Saved State** file, click **Open**. The resulting window should appear like this.

FIGURE 13: HARMONY GUI

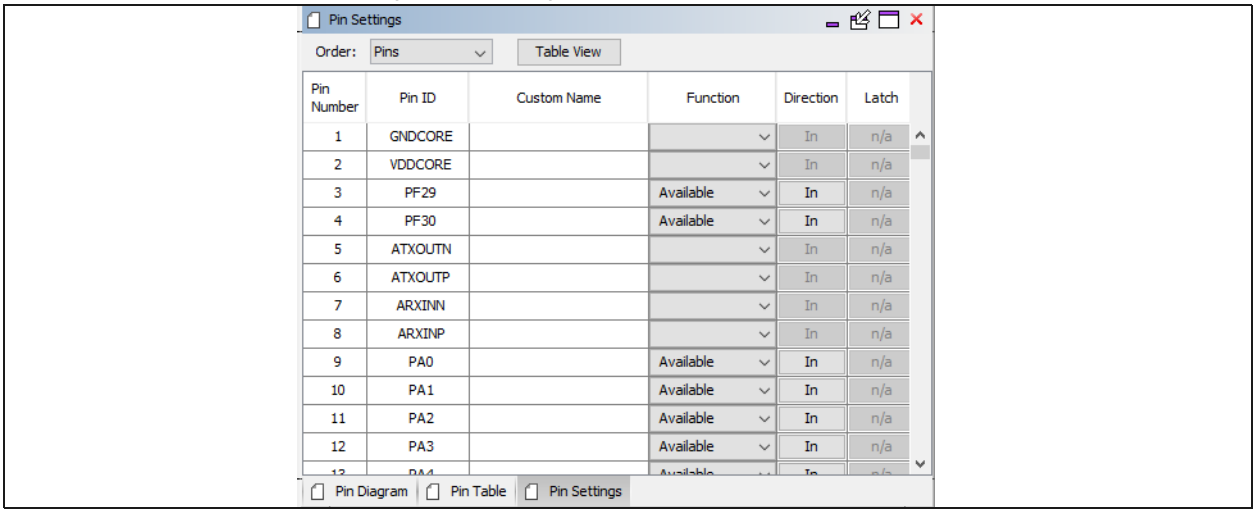


2.1.5.2 Configuring Pin

To configure the pin, perform the following steps.

1. In Harmony's (not MPLAB's) top menu bar, select **Tools** > **Pin Configuration**, which opens a window containing three tabs (along the bottom edge): **Pin Diagram**, **Pin Table**, and **Pin Settings**.

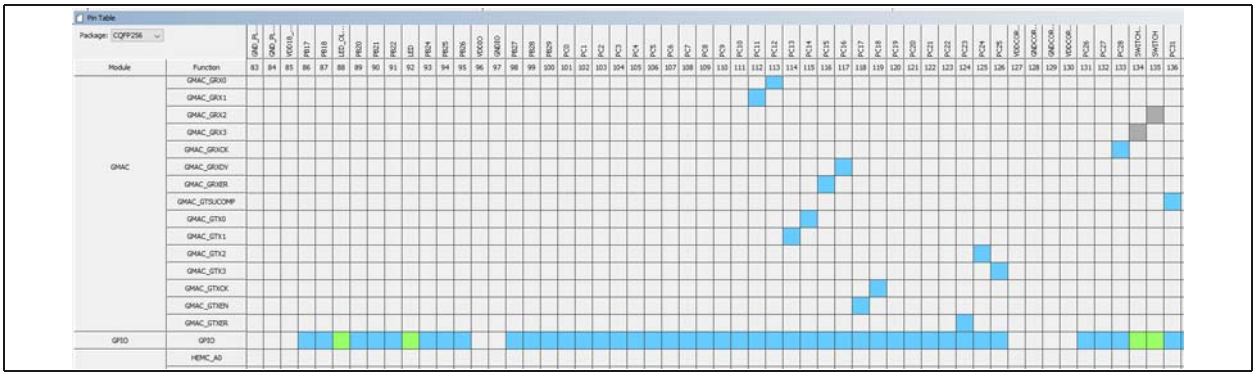
FIGURE 14: PIN DIAGRAM, PIN TABLE, AND PIN SETTINGS



Note: Maximize the window to easily navigate among the pins.

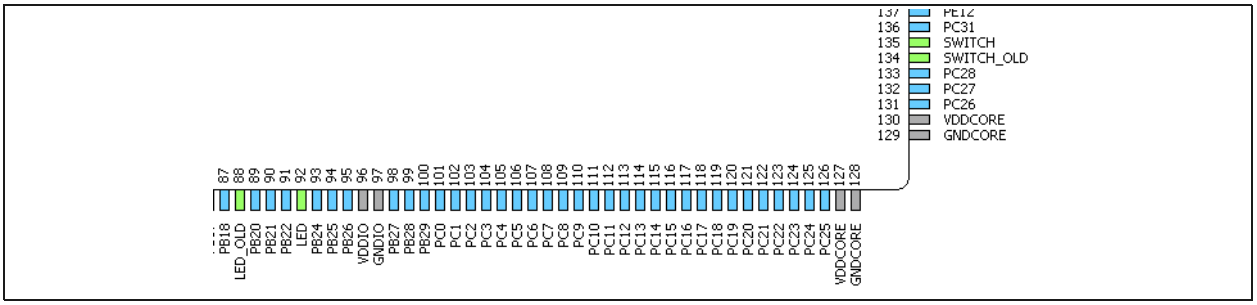
2. In the **Pin Settings** tab, scroll through the pins, down to **PB19** (indicated as **LED**, function **GPIO**, and direction **Out**), double-click the corresponding **Custom Name** cell and enter **LED_OLD** (this disassociateas PB19 from any references in `main.c`).
3. Locate PB23 and change its function from **Available** to **GPIO**, and then double-click the corresponding **Custom Name** cell and enter **LED**, and then change its direction from **In** to **Out**.
4. To assign a push-button control for the LED, locate PC29 and rename as **SWITCH_OLD**; locate **PC30** and change its function from **Available** to **GPIO** and enter **SWITCH** in the **Custom Name** cell. Select the **pull-up** and **debounce filter** options (as seen with "PC29").
5. After completing our changes in the **Pin Setting** tab, click the **Pin Table** and **Pin Diagram** tabs to see the color-coding for these two pins reflect their new, respective **Available** and **Assigned** properties; The following image demonstrates a portion of the resulting Pin Table (note the four green **assigned** boxes).

FIGURE 15: PIN TABLE



The following image is a portion of the resulting Pin Diagram (note the four green **assigned** pins).

FIGURE 16: PORTION of PIN DIAGRAM



2.1.5.3 Generating Project

To generate a project, perform the following steps.

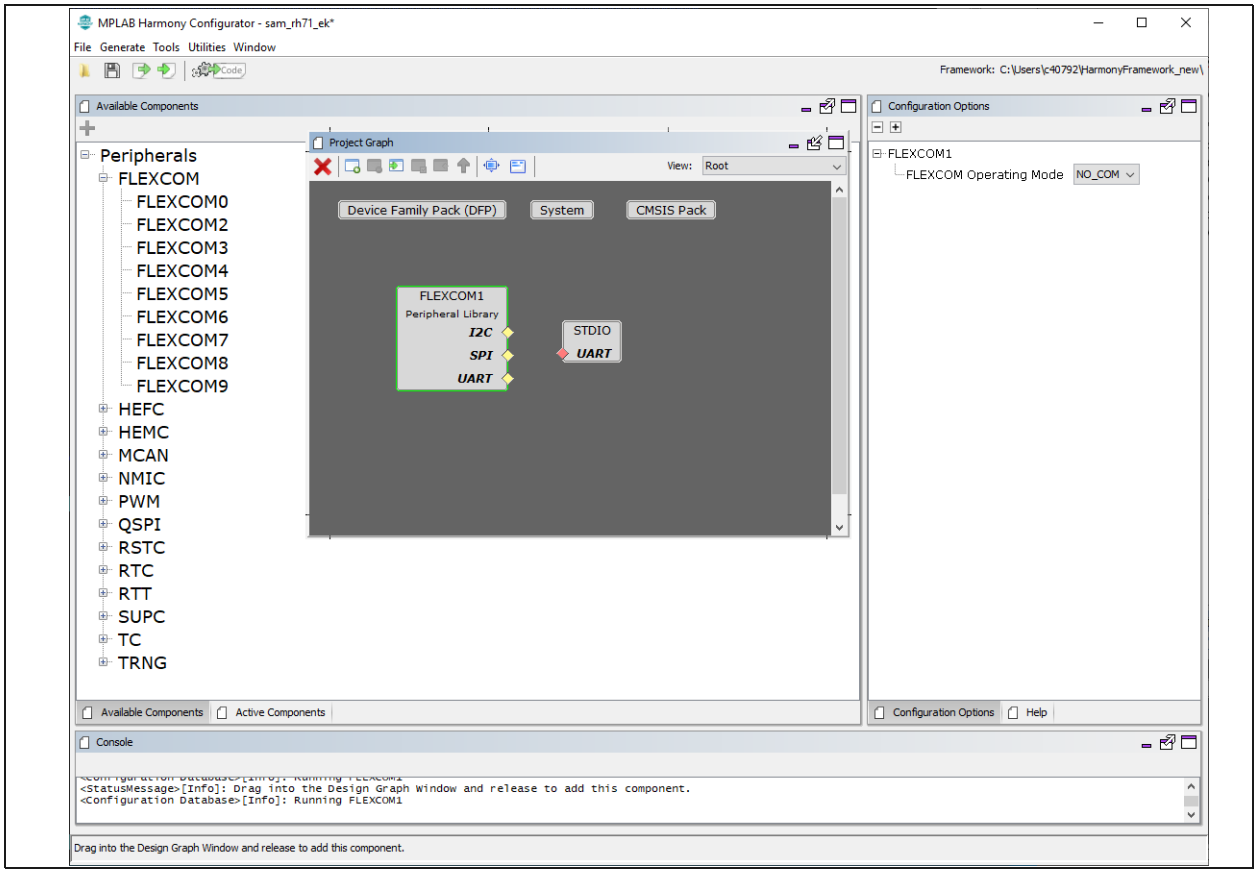
1. On the main menu of Harmony, select **Generate** > **Generate Code**, and click **Save As** when prompted by the resulting pop-up window (**File** > **Save State** also saves the configuration).
2. In the subsequent **Generate Project** window, accept the defaults by clicking **Generate**. If any code in `main_rh71.c` has changed as a result of this procedure, then **Before** and **After** panels appear, which highlight the difference. One can accept or reject the changes.

   Note: You must click **Generate Code** following any changes in Harmony, even if one's `main.c` will not be affected. **Generate Code** commits change to various different files, whereas **Save State** saves only the Harmony's graphical representation of those changes.
3. In the MPLAB (not Harmony) window, click the **Make and Program** (down-arrow) icon.
4. It is generally desirable to add `printf` statements for display in a serial terminal; add a `printf` just before the `while (true)` loop in `main_rh71.c`:

```
printf("\n\r HELLO WORLD!\n\r");
       while (true)
```

   To allow this added functionality, Check the Harmony Configurator GUI. Harmony configurator window displays various available components in the **Available Components** sub-window. Under **Tools**, **STDIO** is available.
5. Drag **STDIO** from the **Components** window into the central **Project Graph** window; once dropped, it appears as a small box, containing a UART port on its left side.
6. Under **Peripherals**, expand **FLEXCOM** and then drag the resulting **FLEXCOM1** entry into the **Project Graph** window; once dropped, it appears as a box, containing three ports on its right side: **I2C**, **SPI**, and **UART**.

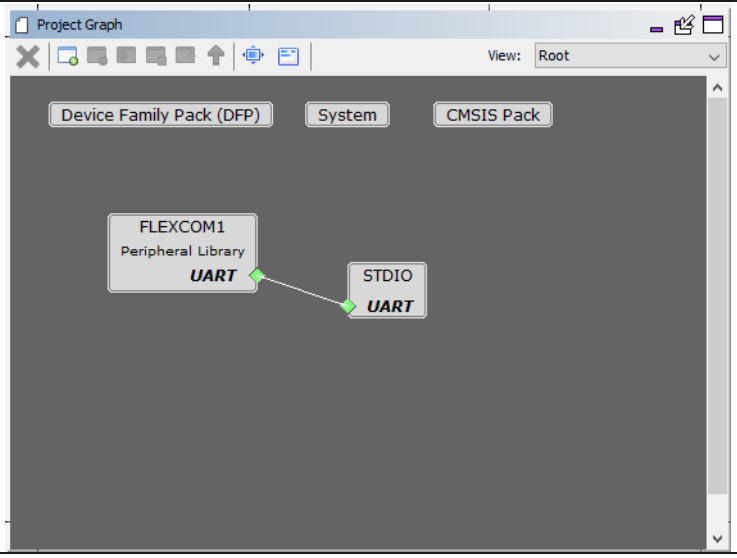
FIGURE 17: STUDIO IN PROJECT GRAPH



Note: When the components are dragged into the **Project Graph** window, they are no longer available in the **Available Components** window, because those resources have been allocated.

7. Click and hold the left mouse button on either of the two UART ports and drag the mouse to the other UART port, and release. A line now connects the two UART ports and the other **FLEXCOM1** ports disappear from the **FLEXCOM1** block as shown in the following image.

FIGURE 18: CONNECTING UART PORTS

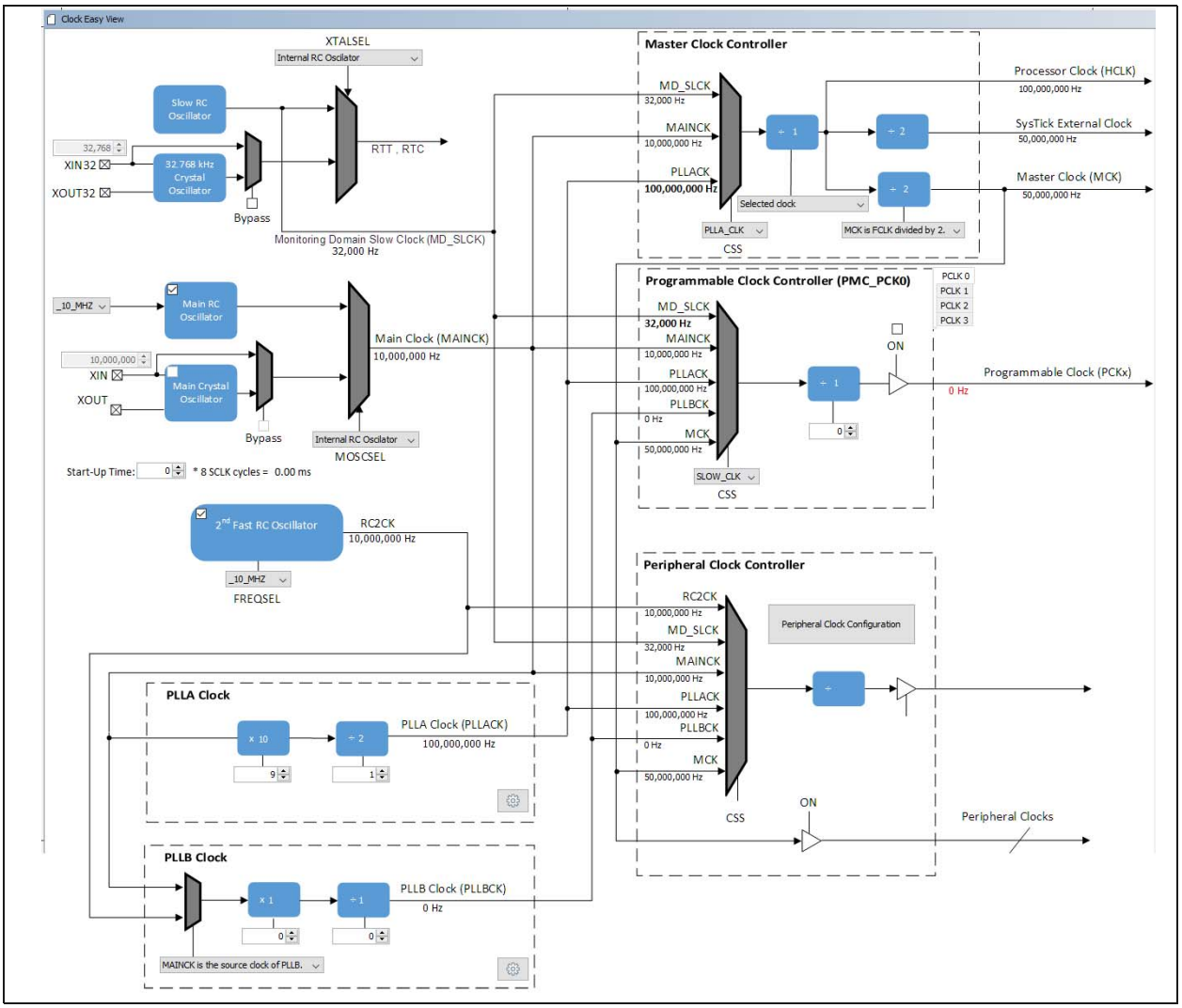


After connecting the two UART ports, click **FLEXCOM1** in the **Project Graph** window to update the **Configuration Options** window (upper-right) to reflect the properties of FLEXCOM1 (for example, Baud rate). Drill down to view all the sub-level entries.

8. On the main menu bar of Harmony Configurator, select **Tools** > **Clock Configuration**. The **Clock Easy View** window opens, which displays the full SAMRH71 clock system.

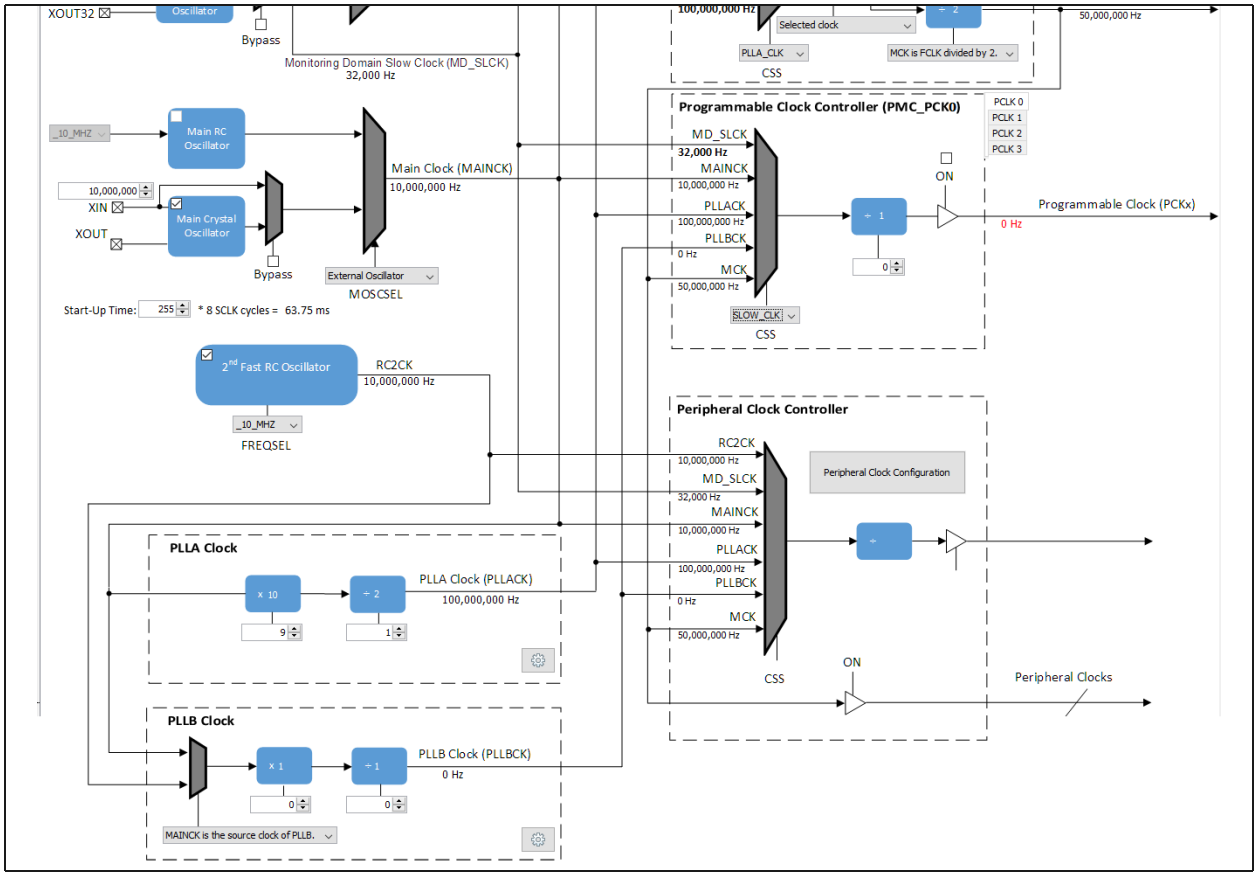
   Adjust as appropriate for the SAMRH71 board, in order to use the UART (correct values are needed for synchronization). The following image displays the default values.

FIGURE 19: CLOCK CONFIGURATOR WINDOW BEFORE EDIT



Update the main clock values to use the external oscillator, checking the main crystal oscillator box, and set the start-up time to **255** (otherwise the board might not properly come out of reset).

FIGURE 20: CLOCK CONFIGURATOR WINDOW AFTER EDIT



For inclusion of components, clock configuration, and pin configuration, it can help to first open a Harmony Framework example which uses the components one needs. Then, take screenshots of the clocking diagram, the **Project Graph** (for components), and any relevant pin configurations to compare with intended configurations.

If instead of using Harmony, a `printf` statement is added in `main.c` of the `pio` project (and FLEXCOM pins are configured directly in one's files), the project builds without error, but nothing is printed on PuTTY terminal. In this case, find an example, which uses `printf`, for example Harmony's trng example. Review of the trng clock, pin, and component configuration reveals that the clocking in the previous example differs from that of trng.Then, the clocking and start-up parameter (in Harmony's **Clock Easy View** window) has to be adjusted, as seen in above images to match what the trng example uses.

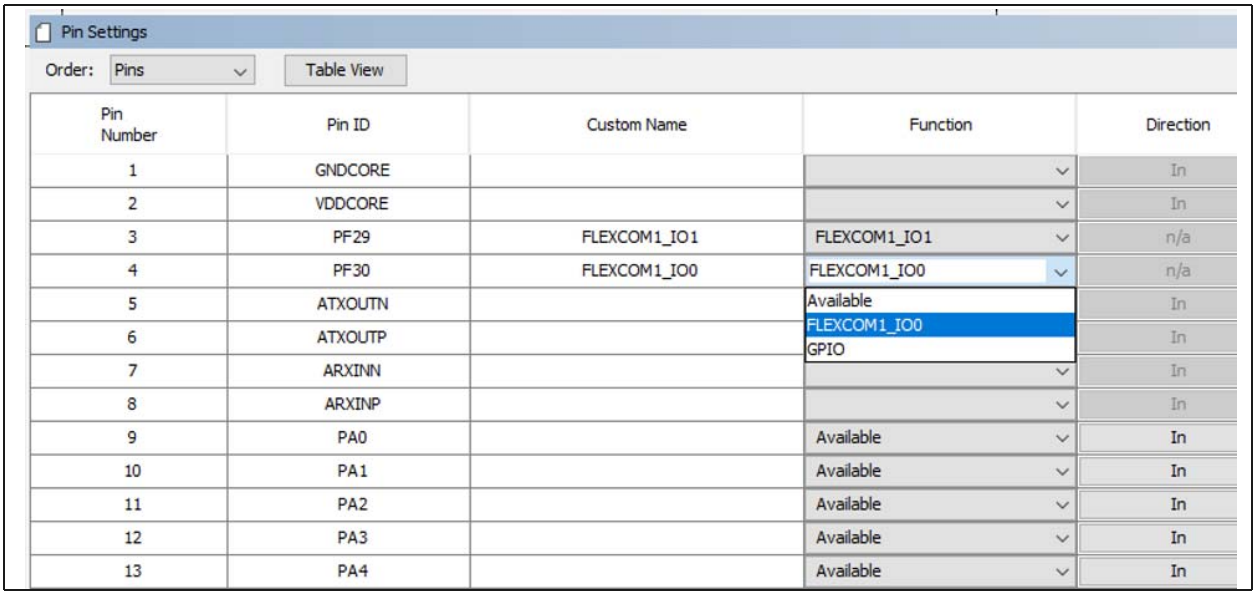
9. On Harmony's top menu bar, select **Tools** > **Pin Configuration**, which opens **Pin Settings** window.

Note: It is necessary to connect UART pins; otherwise you can not see anything printed on your serial terminal.

10. In the **Pin Settings** window, for **Pin Number 3** and **4**, in the **Function** list, select **FLEXCOM1_IO1** and **FLEXCOM_IO0** respectively as shown in the following image (each **Custom Name** cell updates automatically).

Note: These pins are physically connected on the SAMRH71 evaluation board (see the SAMRH71-EK User's Guide for further information on the board's connections of SAMRH71 pins).

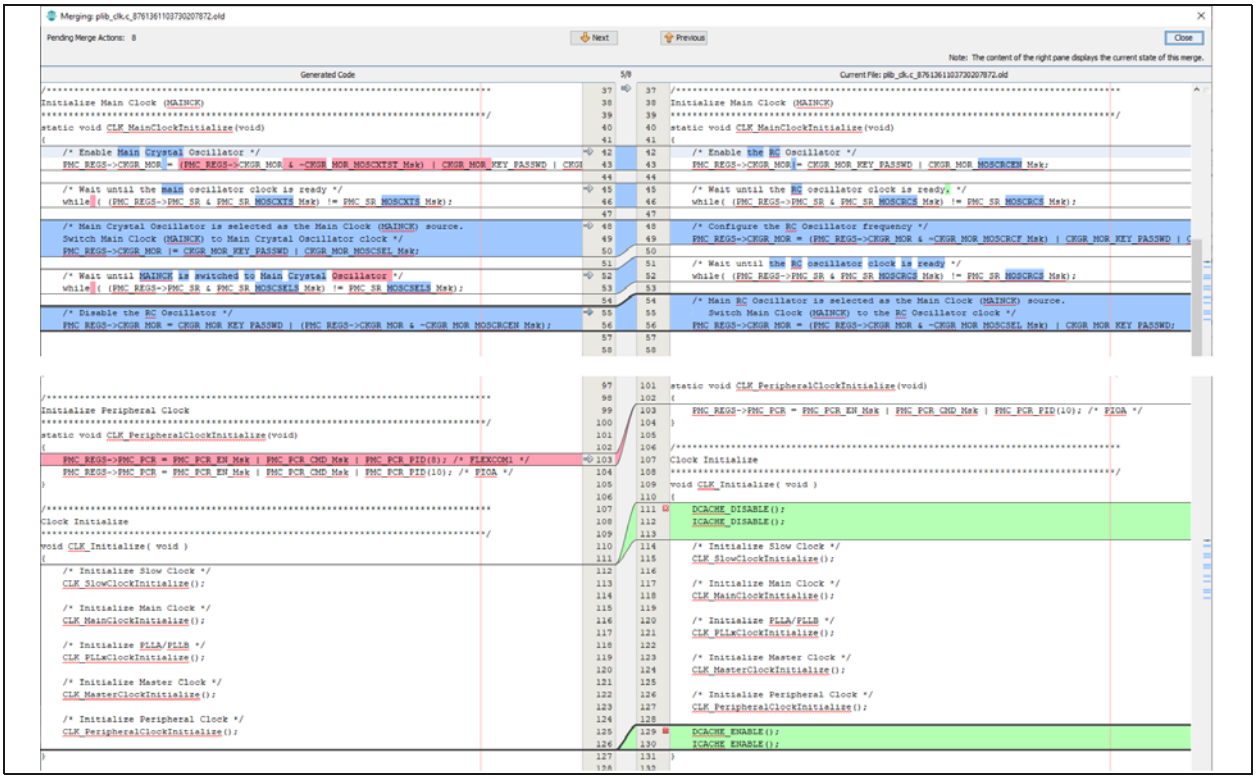
FIGURE 21: SETTING PINS



11. When all the desired configurations are done, on the top menu bar of Harmony, select **File** > **Save State** to save the current configuration; this includes the clocking, pins, components (and their connections), and any other configuration (that is, under **Tools**) options which have been modified.
12. Select **Generate** > **Generate Code** in order to generate new code which reflects the configuration changes.

When the LED and SWITCH pins were changed, Harmony did not display the changed code (although minor code change did actually occur, to reflect the changed pins). The clock changes result in more significant code changes, which Harmony presents for review. For review of the changes, two panels appear in a the **Merging** window.

FIGURE 22: CODE MERGE WINDOW



Changes in our code are reflected in the previous (edited) image. The generated code is in the left panel, and the old code is in the right panel. Notice that the earlier edits are in the right panel, highlighted in green. Each generated change is highlighted with a transfer arrow in the left pane, and one must click each transfer arrow in order to commit the change in the right-hand panel. One may instead click the one arrow in its own center column (seen above, between the panels), to commit all changes; however, this completely overwrites the files in the right panel; for example, our earlier cache disable/enable lines, which are highlighted in green, should be kept.

13. When finished, click **Close**. Then, back in MPLAB, select **Make and Program Device**.

This concludes the examination of the pio example. For information on debugging with MPLAB, a Microchip video series titled "MPLAB X IDE Advanced Debugging" is available on YouTube. The series consists of a brief introduction video and then 7 episodes, each episode running 5-8 minutes. Examples are presented in which line, address, data, sequence, and event breakpoints are used.

REFERENCES

SAMRH71 Device Data Sheet, DS60001593A, Microchip Technology Inc., 2019*

SAMRH71FK20 Evaluation Kit User's Guide, DS50002910A, Microchip Technology Inc., 2019.

Cortex M7 Technical Reference Manual (Revision F), Arm Limited, 2019 (www.arm.com)

IAR Embedded Workbench C-SPY Debugging Guide for Arm Cores (19th edition), IAR Systems, May 2019.

MPLAB X IDE User's Guide, DS50002027D, MPLAB Harmony Configurator User's Guide, DS(TBD). Microchip Technology Inc., 2018.

APPENDIX A: REVISION HISTORY

Revision B (March 2020)

- The USING THE SERIAL PORT section was removed.
- The Get Started with Atmel Studio 7 section was replaced by Get Started with MPLAB section. For more information, see **Section 2.1 “Get Started with MPLAB”**.
- The IAR Embedded Workbench section was removed.

Revision A (September 2019)

- Initial Release.

THE MICROCHIP WEBSITE

Microchip provides online support via our WWW site at www.microchip.com. This website is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the website contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip website at www.microchip.com. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the website at: http://microchip.com/support

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Trademarks

ISBN: 978-1-5224-5008-5

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820