AT12874: Getting Started with SAM S70/E70

APPLICATION NOTE

Introduction

This application note provides information on how to get started with the Atmel® | SMART SAM S/E series, Atmel ARM® Cortex®-M7 based microcontroller. The application note will provide information on how to get datasheet, tools, and software, and give a step-by-step instruction on how to load and build a single example project with Xplained Ultra Evaluation Kit.

Glossary

ACC - Analog Comparator

ADC - Analog to digital converter

AFEC - Analog Front-End Controllers

BOD - Brown-out Detector

CAN - Controller Area Networks

DAC - Digital-to-Analog Controller

DMA - Direct Memory Access

EDBG - Embedded Debugger

HSMCI - High-speed Multimedia Card Interface

I^2C - Inter-Integrated Circuit

ICM - Integrity Check Monitor

IDE - Integrated Development Environment

ISI - Image Sensor Interface

MPU - Memory Protection Unit

NVIC - Nested Vectored Interrupt Controller

PIO - Parallel Input/Output (PIO) controller

POR - Power-on-Reset

QSPI - Quad I/O Serial Peripheral Interface

RASR - Region Attribute and Size Register

RBAR - Region Base Address Register

SCL - Serial Clock Line

SDA - Serial Data Line

SPI - Serial communication interface

SSC - Serial Synchronous Controller

SysTick - System Tick Timer

TC - Timer Counter (TC)

TCM - Tightly Coupled Memory

TRNG - True Random Number Generator

TWI - Two-Wire Interfaces

USART - Universal asynchronous receiver/transmitter

WDT - Watchdog Timer

Table of Contents

1. Pre-requisites

The software referenced in this application note requires several components:

- One SAM V71 Xplained Ultra Evaluation Kit
- One PC running Windows® 7
- One of the following development tools:
  - Atmel Studio 6.2 SP2 or higher
  - IAR Embedded Workbench® for ARM (later than V7.40.1)
  - Keil MDK-ARM (later than V5.12)
  - GNU Tools for ARM Embedded Processors (later than V4.8.4)

Note: MinGW (later than V0.6.2) is necessary for GNU.

- The following debugger:
  - EDBG (this unit offers SWD and USART port), that is part of SAM V71 Xplained Ultra Evaluation Kit
    - AtmelUSBInstaller.exe (later than 6.2.342)

In this document, examples are used to guide you in setting up development environments for these tools.

Even though the SAM S70/E70 MCU series support the following debuggers, they are not explained in this getting started document.

- SAM-ICETM (J-Link) (later than V8.0)
  - SEGGER J-Link software and documentation pack (later than V4.96)
- ULINKproTM and ULINK2TM for MDK

2. Get the Device Datasheet

Web page:

- SAM E: http://www.atmel.com/products/microcontrollers/arm/sam-e.aspx
- SAM S: http://www.atmel.com/products/microcontrollers/arm/sam-s.aspx

Datasheet document:

- SAM E70 Complete (.pdf) available at the link: http://www.atmel.com/products/microcontrollers/arm/sam-e.aspx?tab=documents
- SAM S70 Complete (.pdf) available at the link: http://www.atmel.com/products/microcontrollers/arm/sam-s.aspx?tab=documents

3. Get the SAMS70/E70 Kit

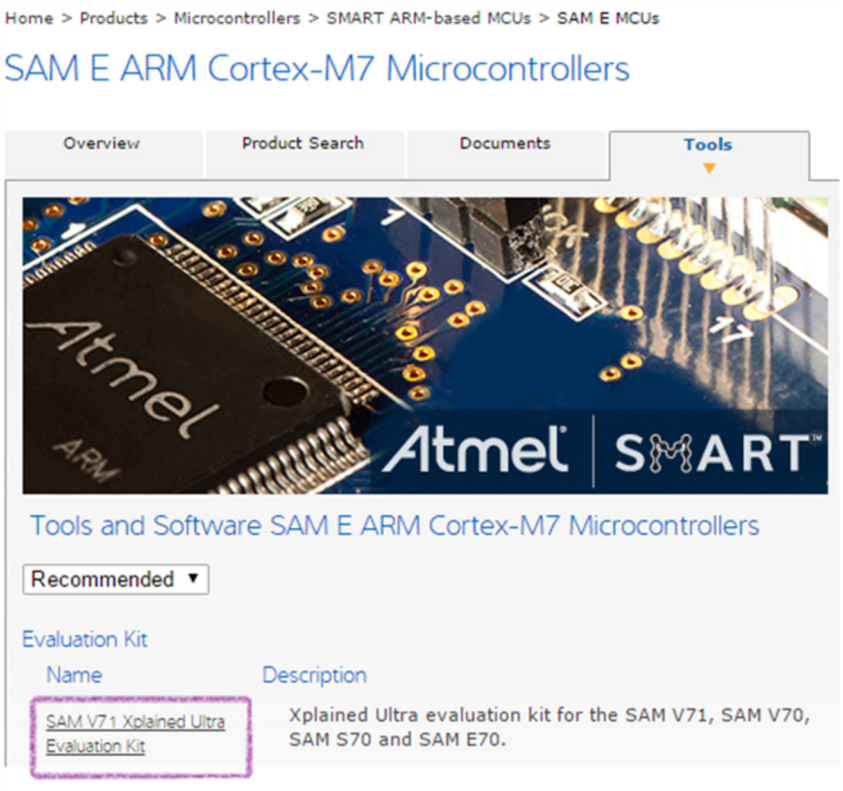
Web page:

- SAM E: http://www.atmel.com/products/microcontrollers/arm/sam-e.aspx?tab=tools
- SAM S: http://www.atmel.com/products/microcontrollers/arm/sam-s.aspx?tab=tools

To get the kit:

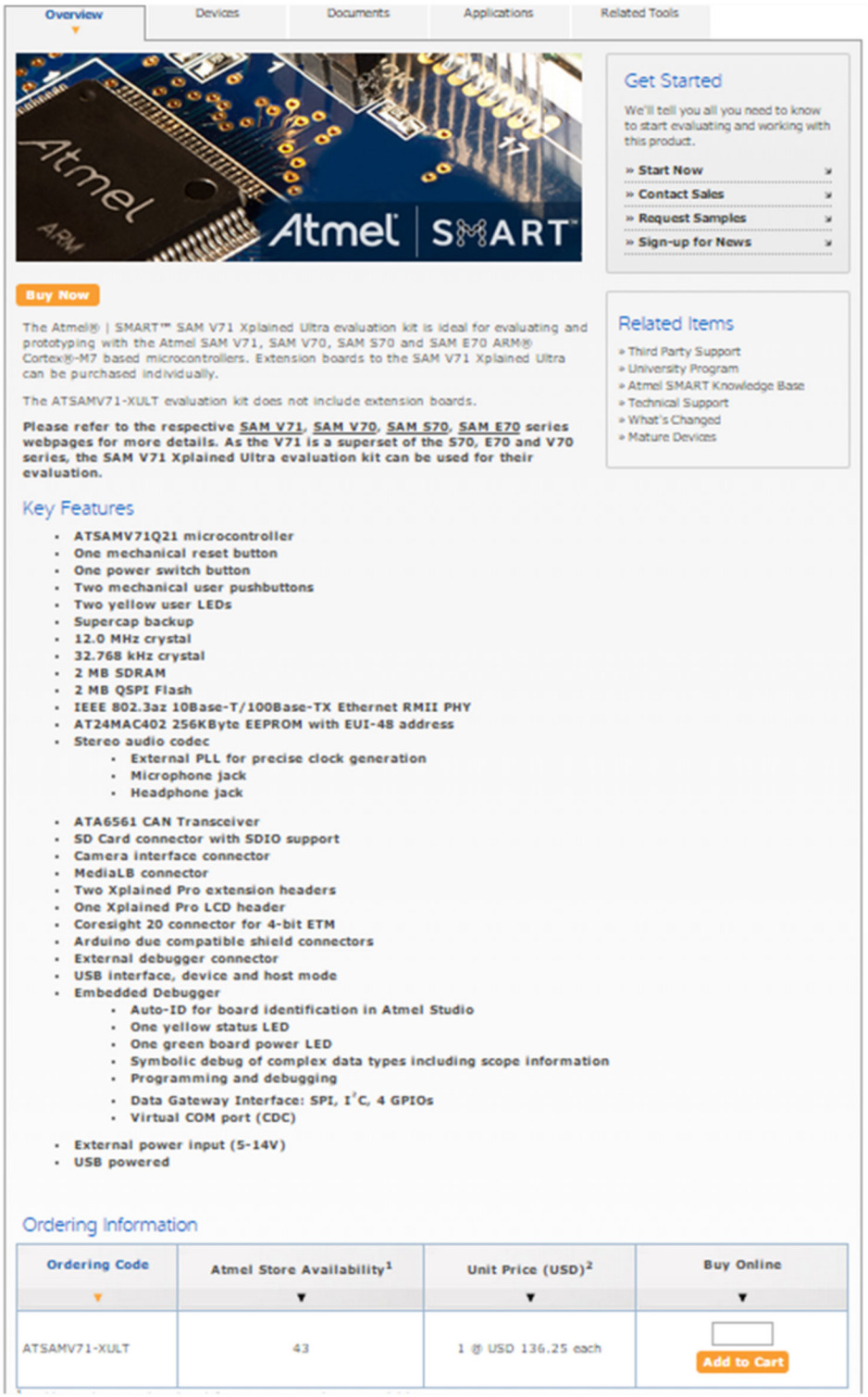
- Click on SAM V71 Xplained Ultra Evaluation Kit (marked in Purple box) as shown in the figure below:

Figure 3-1 SAM V71 Xplained Ultra Evaluation Kit in SAM E/S Product Site



It leads to the link http://www.atmel.com/tools/ATSAMV71-XULT.aspx and the following image appears (see the next figure):

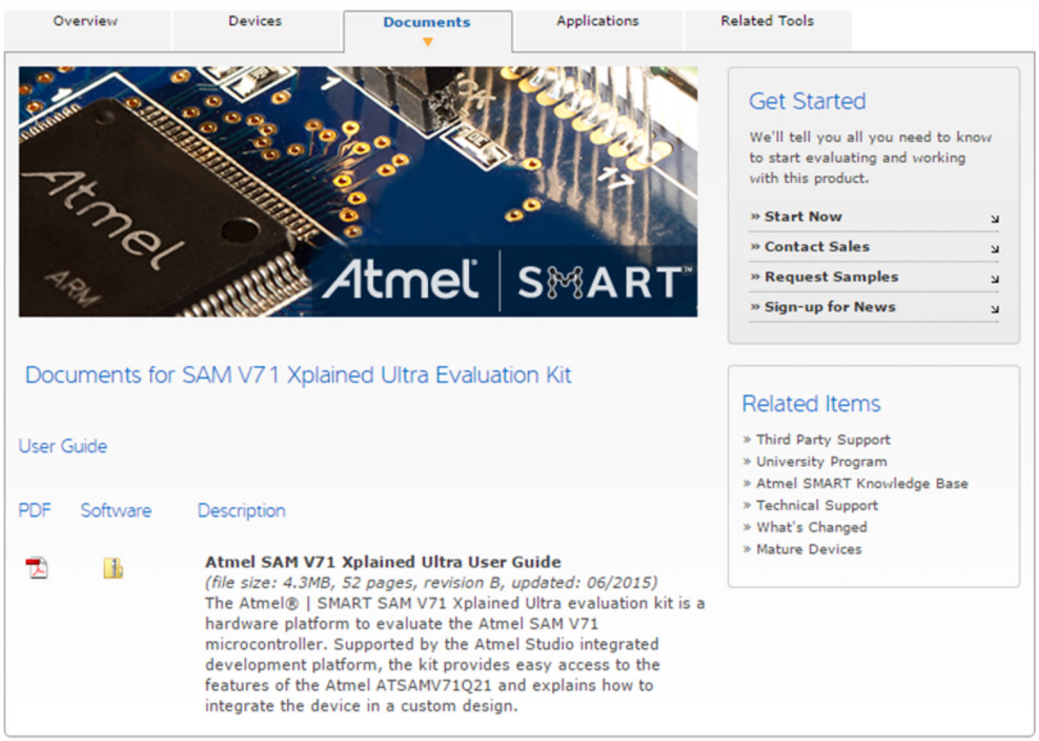
Figure 3-2 SAM V71 Xplained Ultra Evaluation Kit Link



The features supported by the kit are listed as shown in the figure above. The kit can be ordered by using "Buy Now" or entering the quantity in the ordering information box.

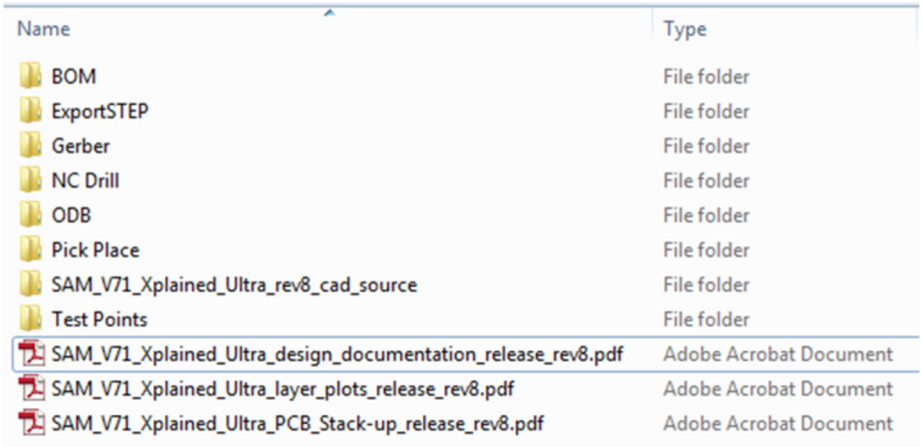
- The link http://www.atmel.com/tools/ATSAMV71-XULT.aspx?tab=documents leads to the user guide page as shown in the figure below:

Figure 3-3 SAM V71 Xplained Ultra Evaluation Kit: Documents Link



1. The pdf document in the link and as shown in the figure above is the SAM V71 Xplained Ultra User Guide. Inside the document the features supported by the kit with all extension header pinouts, other connectors (LCD Extension Connector, Arduino Connectors etc..), connectors silkscreen images, peripherals supported by the kit, Embedded Debugger Implementation and Known Issues, are highlighted.
2. The zip file in the user guide page (refer to the zip file below the software column), consists BOM, Gerber etc, as shown in the figure below:

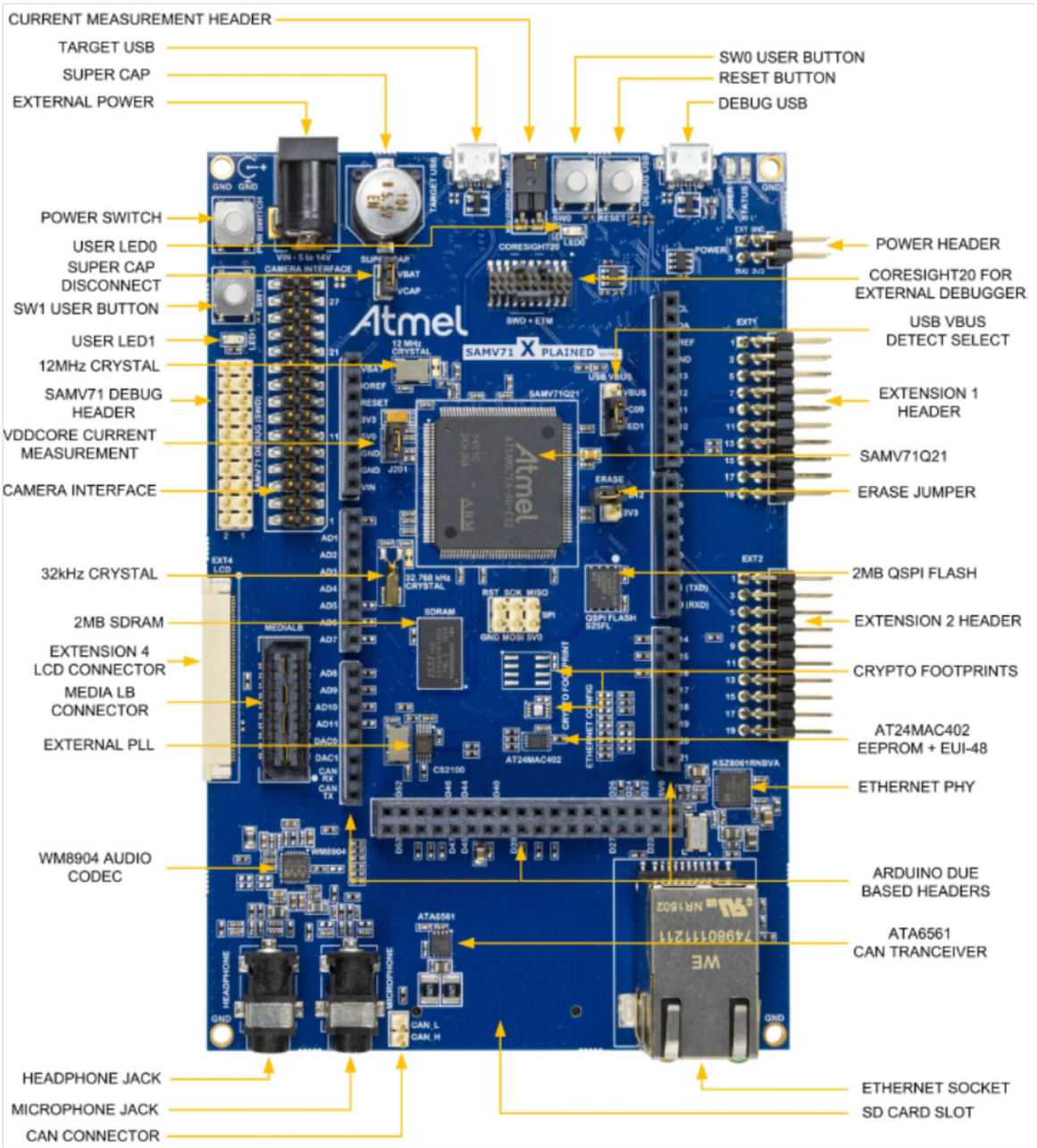
Figure 3-4 SAM V71 Xplained Ultra Evaluation Kit: User Guide Zip File Contents



Note: *This is the image of revision 8 folder contents. Some documents can be added or deleted to this package. Use the latest version from the web.*

- The SAM V71 Xplained Ultra board picture is shown in the figure below:

Figure 3-5 SAM V71 Xplained Ultra Evaluation Kit



4. Get the Tools

- Atmel Studio 6.2 SP2 or later: http://www.atmel.com/tools/atmelstudio.aspx?tab=overview
- Atmel Software Framework 3.25 or higher: http://www.atmel.com/tools/avrsoftwareframework.aspx?tab=overview
- IAR™ Embedded Workbench for ARM 7.40.1 or higher: www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/
- Keil MDK-ARM (v5.14 or higher): https://www.keil.com/download/product/
- Segger J-Link (v4.96 or higher): www.segger.com/download_jlink.html
- Atmel SAM-BA® (v2.15 or higher): http://www.atmel.com/tools/ATMELSAM-BAIN-SYSTEMPROGRAMMER.aspx
- Atmel SAM-ICE™ - a JTAG emulator for Atmel ARM-based MCUs: http://www.atmel.com/tools/ATMELSAM-ICE.aspx
- SAM V71/V70/E70/S70 Software Package: http://www.atmel.com/tools/samv71-samv70-same70-sams70-software-package.aspx. This link consists of the following releases. All the packages provides software drivers and libraries to build any application for SAM V71, SAM V70, SAM S70, and SAM E70 Cortex-M7 based MCUs. To know more about changes in the latest release, refer to the release package available in this page.
  - SAMV71-XULT IAR EWARM 7.40.1 Software Package 1.3: SAM V71 Software Package for EWARM requires an installation of IAR Systems Embedded Workbench for ARM version 7.40.1 or later.
  - SAMV71-XULT Atmel Studio Software Package 1.3: Atmel Studio Package adds support for the SAM V71 devices in Atmel Studio 6.2, and also package support for GNU Tools for ARM Embedded Processors version 4.8.4 or later.
  - SAMV71-XULT KEIL MDK 5.12 Software Package 1.3: SAM V71 Software Package for Keil MDK requires an installation of MDK version 5.12 or later.
  - SAMV71-XULT GNU Software Package 1.3: The SAM V71 SoftPack for GNU requires an installation of GNU Tools ARM Embedded version 4.8.4 or later.

Note: *To download the Atmel Software Framework 3.25 or SAM V71/V70/E70/S70 Software Package contents, log-in access needed using Atmel account.*

5. The Getting-started Example

This chapter describes a simple example project that uses several important features present on the SAM E70/S70/V70/V71 devices.

There are four main parts in this chapter:

1. The specification of the getting-started example.
2. The introduction about relevant on-chip peripherals.
3. The introduction about relevant on-board components.
4. The implementation of the various peripherals in the example.

5.1. Specification

5.1.1. Atmel Studio Program

The demonstration program makes the LED(s) on the board blink at a fixed rate. This rate is generated by using Time tick timer. The blinking can be stopped by using the push button.

5.2. IAR and Keil Program

The demonstration program makes two LEDs on the board blink at a fixed rate. This rate is generated by using Time tick timer. The blinking can be stopped by typing '1' or '2' in the console (one for each LED).

In the Atmel Studio program, IAR and Keil programs, the demo application can be controlled by using a terminal window.

- On the computer, open and configure a terminal application (e.g. Terminal on Microsoft® Windows) with these settings:
  - 115200 baud rates
  - Eight bits of data
  - No parity
  - One stop bit
  - No flow control
- Run the application
- Two LEDs should start blinking on the board. In the terminal window, the following text should appear (values depend on the board and chip used):
  - \code

Getting Started Example xxx --

xxxxxx-xx

Compiled: xxx xx xxxx xx:xx:xx --

- \endcode
  - Press '1' to start/stop the LED0 blinking and press '2' to start/stop LED1 blinking

While this software may look simple, it uses several peripherals, which make up the basis of an operating system. As such, it serves as a good starting point for someone wanting to become familiar with the SAM S70/E70 microcontroller series.

5.3. On-chip Peripherals

In order to perform the operations described previously, the getting-started example uses the following set of peripherals:

- Parallel Input/Output (PIO) controller
- Timer Counter (TC)
- System Tick Timer (SysTick)
- Nested Vectored Interrupt Controller (NVIC)
- Universal Asynchronous Receiver Transmitter (UART)

LEDs and buttons on the board are connected to standard input/output pins on the chip. The pins are managed by a PIO controller. In addition, it is possible to have the controller generate an interrupt when the status of one of its pins changes; buttons are configured to have this behavior.

The TC and SysTick are used to generate two timebase interrupts, in order to obtain the LED blinking rates. They are both used in interrupt mode:

- The TC triggers an interrupt at a fixed rate, each time toggling the LED state (on/off)
- The SysTick triggers an interrupt every millisecond, incrementing a variable by one tick. The Wait function monitors this variable to provide a precise delay for toggling the second LED state.

NVIC is required to manage the interrupts. It allows the configuration of a separate interrupt handler for each source. Three different functions are used to handle PIO, TC, and SysTick interrupts.

Finally, an additional peripheral is used to output debug traces on a serial line: the UART. Having the firmware send debug traces at key points of the code can greatly help the debugging process.

5.4. On-board Components

5.4.1. Buttons

SAM V71 Xplained Ultra contains three mechanical buttons. One button is the RESET button connected to the SAM V71 reset line and the others are generic user configurable buttons. When a button is pressed, it will drive the I/O line to GND.

Table 5-1 Buttons Description

SAM V71 pin	Function	Shared functionality
RESET	RESET	Trace, Shield, and EDBG
PA09	SW0	EDBG GPIO and Camera
PB12	SW1	EDBG SWD and Chip Erase

5.4.2. LEDs

There are two general purpose LEDs (yellow) on the SAM V71 Xplained Ultra board. They are wired to pins PA23 and PC9. Setting a logical low or high level on the corresponding PIO lines turns the LEDs on and off.

Table 5-2 LEDs Description

SAM V71 pin	Function	Shared functionality
PA23	Yellow LED0	EDBG GPIO
PC09	Yellow LED1	LCD and Shield

The example application uses both LEDs (PA23 and PC9).

5.4.3. COM Port (DBGU/UART)

On SAM V71, the default serial port, which is used to print debug information and monitor input, is USART1. The port uses pins PA21 and PB04 for the RXD1 and TXD1 signals, respectively.

SAM V71 pin	Function	Shared functionality
PA21	RXD1	UART Receive pin
PB04	TXD1	UART Transmit pin

5.4.4. Booting

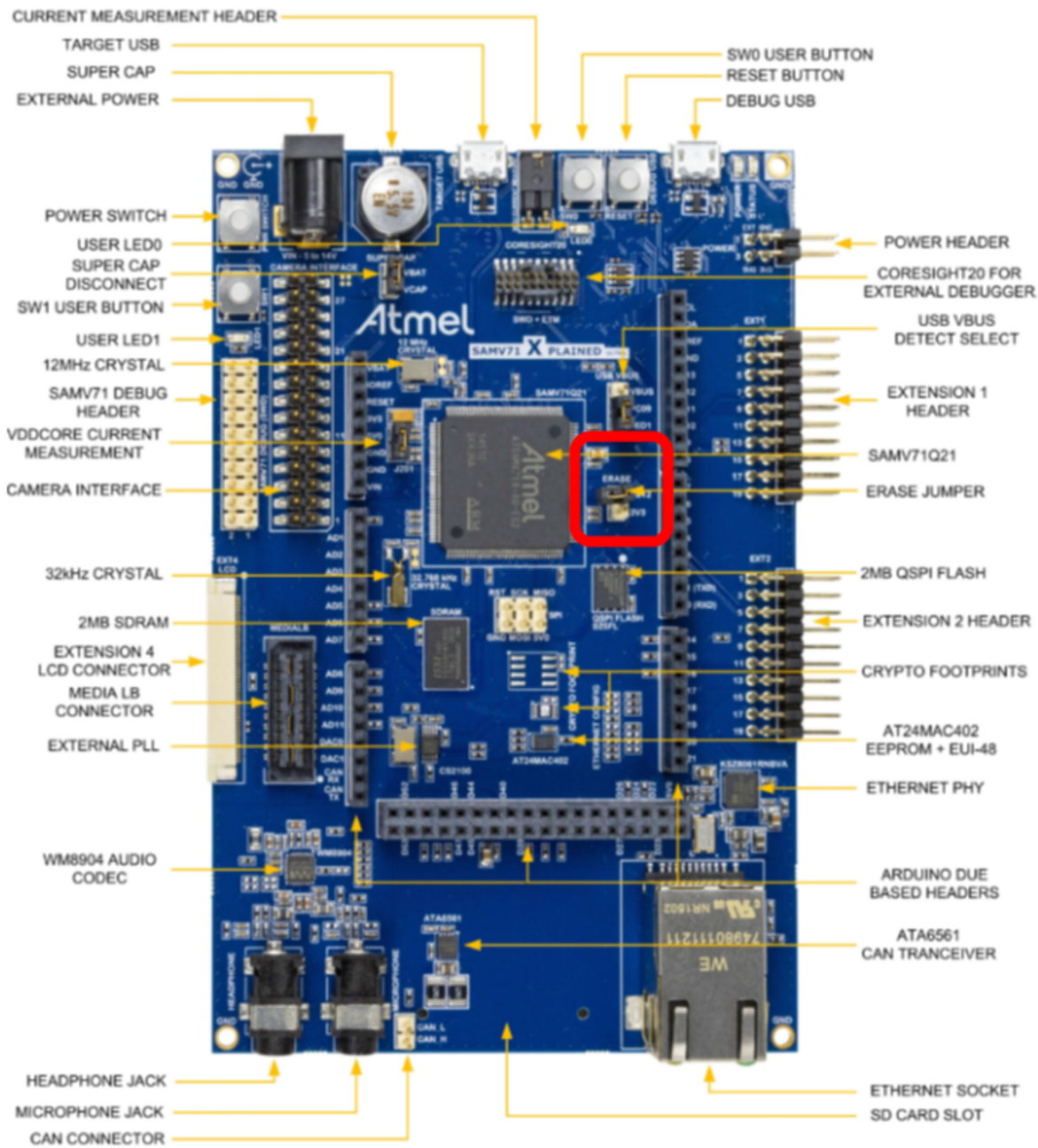
The SAM V71 devices feature up to 2048KB of embedded Flash and up to 384KB of internal SRAM. The Getting Started example can be compiled and downloaded to both the Flash and the SRAM.

The SRAM is accessible over the system Cortex-M bus at address 0x2040 0000 and the base address of the Flash is 0x0040 0000.

5.4.5. Erasing Flash

The user can close the ERASE jumper as mentioned in Figure 3-5, wait for at least seven seconds and then re-power the board to chip-erase SAM V71. The same jumper is highlighted in Figure 5-1.

Figure 5-1 ERASE Jumper on the SAM V71 Xplained Ultra Evaluation Kit



The ERASE jumper is used to reinitialize the Flash content (and some of its NVM bits) to an erased state (all bits read as logic level 1). It integrates a pull-down resistor of about 100kΩ to GND, so that it can be left unconnected for normal operations. The pin must be tied high for more than 220ms to perform a Flash erase operation, otherwise the ERASE operation is not taken into account.

To make sure that the erase operation is performed after power-up, the system must not reconfigure the ERASE pin as GPIO or enter Wait mode with Flash in Deep Power-down mode before the ERASE pin assertion time has elapsed. For more details, refer to the SAM E70/S70/V71 Series datasheet.

5.5. Implementation

As stated previously, the example defined above requires the use of several peripherals. It must also provide the necessary code for starting up the microcontroller.

5.5.1. Initialization Before 'main'

After the board is powered on, the ROM code will run and carry out the necessary initialization.

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

When downloading the application via the J-link GDB server, the users can set the registers of PC and stack pointer to 0x2040 0000 and 0x2040 0004 respectively in the GDB script for SAM V71 Xplained Ultra. Before running the application, the user might still want to:

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

These initialization requirements are described in the next sections.

Entry Point

For GNU toolchain, the PC points to the start address of Reset_Handler at the beginning.

For IAR and MDK, the PC points to the start address of _iar_program_start and Reset_Handler, respectively.

The purpose of the entry point is to:

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

Low-Level Initialization

Starting from the LowLevelInit interface, three toolchains share the program flow.

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

- The main oscillator and its PLL
- MPU
- TCM

The LowLevelInit function is shown as follows.

```
extern WEAK void LowLevelInit( void )
{
SystemInit();
SetupMemoryRegion();
#ifdef ENABLE_TCM
FLASHD_ClearGPNVM(8);
FLASHD_SetGPNVM(7);
TCM_Enable();
#else
TCM_Disable();
```

```
#endif
}
```

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

Low-Level Initialization: SystemInit

The main function of SystemInit is to complete the processor clock and master clock configuration.

After reset, the 4/8/12MHz fast RC oscillator is enabled with the 4MHz 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 SAM E70/S70/V71 Series datasheet.

In the example, the processor clock and master clock are 300MHz and 150MHz respectively by default. Example values on the SAM V71 Xplained Ultra (12MHz crystal):

```
#define SYS_BOARD_PLLAR (CKGR_PLLAR_ONE |\
CKGR_PLLAR_MULA(0x18U) |\
CKGR_PLLAR_PLLACOUNT(0x3fU) |\
CKGR_PLLAR_DIVA(0x1U))
#define SYS_BOARD_MCKR (PMC_MCKR_PRES_CLK_1 |\
PMC_MCKR_CSS_PLLA_CLK |\
PMC_MCKR_MDIV_PCK_DIV2)
```

Here:

- finput = 12MHz
- MAINCK = 12MHz
- PLLACK = MAINCK * MULA / DIVA = (12 * (0 × 18 + 1) / 1)MHz = 300MHz
- HCLK = PLLACK / PRES = (300 / 1)MHz = 300MHz
- MCK = PLLACK / PRES / MDIV = (300 / 1 / 2)MHz = 150MHz

In addition, the user must set the number of wait states of the embedded Flash depending on the system frequency. When MCK is 4MHz and FWS is 0, the number of cycles for Read/Write operations is 1.

In the example, defining FWS as 5 enables six cycles access, which is done as shown below:

```
EFC->EEFC_FMR = EEFC_FMR_FWS(5);
```

For more details, see the “Embedded Flash Wait State” table in the SAM V71 Series datasheet.

Low-level Initialization: Memory Protection Unit (MPU)

The SAM E70/S70/V71 devices supply MPU with 16 zones as a component for memory protection. The users can use the MPU to enforce privilege rules, separate processes, and enforce access rules.

The _SetupMemoryRegion function completes the memory mapping by setting the MPU Region Base Address Register (RBAR), the MPU Region Attribute, and the Size Register (RASR).

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, the MPU region is enabled by setting the MPU Control Register.

In the example, memory regions such as ITCM, Internal flash, DTCM, SRAM, peripheral memory, SDRAM, QSPI memory, and USBHS_RAM are all configured in this function

The SRAM, for example, is divided into two parts with the same attributes.

```
void MPU_SetRegion( uint32_t dwRegionBaseAddr, uint32_t dwRegionAttr )
{
MPU->RBAR = dwRegionBaseAddr;
MPU->RASR = dwRegionAttr;
}
void _SetupMemoryRegion( void )
{
uint32_t dwRegionBaseAddr;
uint32_t dwRegionAttr;
……
dwRegionBaseAddr = SRAM_PRIVILEGE_START_ADDRESS |
MPU_REGION_VALID |
MPU_DEFAULT_PRAM_REGION; //4
dwRegionAttr = MPU_AP_FULL_ACCESS |
INNER_NORMAL_WB_NWA_TYPE( NON_SHARABLE ) |
MPU_CalMPURegionSize(SRAM_PRIVILEGE_END_ADDRESS -
SRAM_PRIVILEGE_START_ADDRESS) |
MPU_REGION_ENABLE;
MPU_SetRegion( dwRegionBaseAddr, dwRegionAttr);
dwRegionBaseAddr = SRAM_UNPRIVILEGE_START_ADDRESS |
MPU_REGION_VALID |
MPU_DEFAULT_UPRAM_REGION; //5
dwRegionAttr = MPU_AP_FULL_ACCESS |
INNER_NORMAL_WB_NWA_TYPE( NON_SHARABLE ) |
MPU_CalMPURegionSize(SRAM_UNPRIVILEGE_END_ADDRESS -
SRAM_UNPRIVILEGE_START_ADDRESS) |
MPU_REGION_ENABLE;
MPU_SetRegion( dwRegionBaseAddr, dwRegionAttr);
……
/* Enable the memory management fault, Bus Fault, Usage Fault exception */
SCB->SHCSR |= (SCB_SHCSR_MEMFAULTENA_Msk |
SCB_SHCSR_BUSFAULTENA_Msk |
SCB_SHCSR_USGFAULTENA_Msk);
/* Enable the MPU region */
MPU_Enable( MPU_ENABLE | MPU_BGENABLE );
}
```

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

Low-level Initialization: Tightly Coupled Memory (TCM)

The SAM E70/S70/V71 devices embed the Tightly Coupled Memory (TCM) running at processor speed.

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).

After reset, the DTCM is enabled by default. ITCM is disabled and needs to be enabled by software. When enabled, the ITCM is located at 0x0000 0000, overlapping ROM or Flash depending on the general-purpose NVM bit 1 (GPNVM). The TCM configuration is done with GPNVM bits [8:7].

The user can program them through the “Clear GPNVM Bit” and “Set GPNVM Bit” commands of the EEFC User Interface.

ITCM	DTCM	SRAM for 384K	RAM-based SRAM for 256K	RAM-based GPNVM Bits [8:7]
0	0	384	256	0
32	32	320	192	1
64	64	256	128	2
128	128	128	0	3

Use the following codes to configure TCM to 32KB and enable it:

```
FLASHD_ClearGPNVM(8);
 FLASHD_SetGPNVM(7);
 TCM_Enable();
```

Accesses made to TCM regions when the relevant TCM is disabled and accesses made to the Code and SRAM region above the TCM size limit are performed on the AHB matrix, i.e., on internal Flash or on ROM depending on remap GPNVM bit.

Accesses made to the SRAM above the size limit will not generate aborts.

The Memory Protection Unit (MPU) can be used to protect these areas as mentioned in Low Level Initialization: Memory Protection Unit.

Note that internal SRAM and TCM share the same memory space, which means that when TCM is enabled, the size available as internal SRAM is correspondingly reduced.

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

5.5.2. Generic Peripheral Usage

Initialization

Most peripherals are initialized by performing the following actions:

- Disabling or reprogramming watchdog
- Enabling cache if necessary
- Enabling the peripheral clock in the PMC if necessary
- Enabling the control of the peripheral on PIO pins
- Enabling the interrupt source at the peripheral level

5.5.3. Disabling or Reprogramming Watchdog Timer (WDT)

Purpose

The Watchdog Timer (WDT) is used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow the clock to around 32kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

After a processor reset, the Watchdog peripheral is enabled by default with the value of watchdog counter value equal to 0xFFF, which corresponds to the maximum value of the counter with the external reset generation enabled (bit WDT_MR.WDRSTEN at 1 after a backup reset). The user can either disable the WDT by setting bit WDT_MR.WDDIS or reprogram the WDT to meet the maximum watchdog period the application requires.

Initialization

In the example, the user can disable WDT with the WDT_Disable function as follows.

```
WDT_Disable( WDT ) ;
```

The operation is done by setting WDT_MR as follows.

```
pWDT->WDT_MR = WDT_MR_WDDIS;
```

5.5.4. Enabling Cache If Necessary

Purpose

The SAM E70/S70/V71 devices support 16KB of ICache and 16KB of DCache with Error Code Correction (ECC). All caches are disabled at reset and enabling cache benefits the performance. The user can turn on ICache and DCache if necessary.

Initialization

The user can enable cache as follows.

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

The details of the SCB_EnableICache function is shown below as an example.

```
SCB->ICIALLU = 0; // Invalidate I-Cache
SCB->CCR |= SCB_CCR_IC_Msk;// Enable I-Cache
```

To make some regions cacheable, the following conditions should all be met:

- Enable cache as described above in the application
- Set the attributes of the relevant regions as cacheable in _SetupMemoryRegion function (refer to SectionLow-Level Initialization: Memory Protection Unit (MPU))

Cache Coherency

Enabling cache may cause breakdown when:

- Memory locations are updated by other agents in the system
- Memory updates made by the application code must be made visible to other agents in the system

For example, in a system with a DMA that reads memory locations held in the data cache of a processor, a breakdown of coherency occurs when the processor has written new data in the data cache, but the DMA reads the old data held in memory.

In situations where a breakdown in coherency occurs, the software must manage the caches by using cache maintenance operations. The Clean, Invalidate and Clean, and Invalidate operations can address these issues.

Take DCache as an example, these operations are realized in several functions such as:

```
static inline void SCB_InvalidateDCache();
static inline void SCB_CleanDCache ();
static inline void SCB_CleanInvalidateDCache ();
```

5.5.5. Using the Nested Vectored Interrupt Controller (NVIC)

Purpose

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

The NVIC supports up to 72 interrupts, each with up to eight 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.

Initialization

The SAM E70/S70/V71 uses hardware to save and restore key context state on exception entry and exit, and use 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.

Part of the constant is shown as follows:

```
__attribute__ ((section(".vectors")))
const DeviceVectors exception_table =
{
.pvStack = (void*) (&_estack),
 .pfnReset_Handler = (void*) Reset_Handler,
 .pfnNMI_Handler = (void*) NMI_Handler,
 .pfnHardFault_Handler = (void*) HardFault_Handler,
 ……
 .pfnSysTick_Handler = (void*) SysTick_Handler,
 ……
 .pfnTC0_Handler = (void*) TC0_Handler,
 ……
}
```

On reset, the processor initializes the vector table base address to an IMPLEMENTATION DEFINED address. The software can find the current location of the table, or relocate the table, by using the Vector Table Offset Register (VTOR) as shown below.

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

The _sfixed symbol points to the vectors section, which saves the vectors table. The SCB_VTOR_TBLOFF_Msk is equal to 0xFFF FFF8 on SAM V71 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 the 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 will be executed instead of the default interrupt handler.

2. Disable the interrupt in case 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(IRQn_Type IRQn);
   ```

3. Clear any pending interrupt, if any
   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.

5.5.6. Using the Timer Counter (TC)

Purpose

Timer Counters on SAM chips can perform several functions, e.g., frequency measurement, pulse generation, delay timing, and Pulse Width Modulation (PWM).

In this example, a single Timer Counter (TC) channel is going to provide a fixed-period delay. An interrupt is generated each time the timer expires, toggling the associated LED on or off. This makes the LED blink at a fixed rate.

Initialization

In order to reduce power consumption, most peripherals are not clocked by default. Writing the ID of a peripheral in the PMC Peripheral Clock Enable Register (PMC_PCERx) activates the peripheral clock.

The TC initialization sequence is the following:

1. Write the ID of the TC in the PMC Peripheral Clock Enable Register (PMC_PCERx):
   `PMC_EnablePeripheral(ID_TC0);`
2. Configure TC as 4Hz frequency by calling the function TC_FindMckDivisor, which will find the best MCK divisor. The best divisor depends on the timer frequency and MCK.
   `TC_FindMckDivisor( 4, BOARD_MCK, &div, &tcclks, BOARD_MCK );`
3. Configure the TC Channel Mode Register (TC_CMRx). TC channels can operate in different modes. In the example, set the TC in Capture mode by clearing the WAVE bit and enable RC Compare Trigger by setting the CPCTRG bit, , which is done in the internal TC_Configure function:
   `TC_Configure( TC0, 0, tcclks | TC_CMR_CPCTRG );`

4. Configure the interrupt whenever the counter reaches the value programmed in RC. The interrupt priority is level 0 as default. At the TC level, the RC Compare Interrupt is enabled by setting the CPCS bit of the TC Interrupt Enable Register (TC_IERx):

```
NVIC_ClearPendingIRQ(TC0_IRQn);
NVIC_EnableIRQ(TC0_IRQn);
TC0->TC_CHANNEL[ 0 ].TC_IER = TC_IER_CPCS ;
```

At the end of the sequence, the program starts the counter if LED1 is enabled as shown below:

```
if ( bLed1Active ) {
 TC_Start( TC0, 0 );
 }
```

Interrupt Handler

The interrupt handler for TC0 interrupt is ‘TC0_Handler’ and the main purpose is to toggle the state of the LED.

The first action to do in the handler is to acknowledge the pending interrupt from the peripheral. Otherwise, the latter continues to assert the IRQ line. In the case of a TC channel, acknowledging is done by reading the corresponding TC Status Register (TC_SRx). The code is shown below.

```
dummy = TC0->TC_CHANNEL[ 0 ].TC_SR;
```

It simply toggles the state (on or off) of one of the blinking LEDs by programming the PIO controller.

```
dummy = TC0->TC_CHANNEL[ 0 ].TC_SR;
```

Refer to Section ‘Controlling LEDs’ for more details.

5.5.7. Using the System Timer (SysTick)

Purpose

The system timer, SysTick, provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism.

This getting started example uses the SysTick to provide a 1ms time base. Each time the interrupt is triggered, a 32-bit counter is added. A Wait function uses this counter to provide a precise way for an application to suspend itself for a specific amount of time.

Initialization

Initialization is done with the following line of code:

```
SysTick_Config( Pck/1000);
```

In this example, the SysTick clock source is the processor clock / 8 and PCK = BOARD_MCK*2. The function initializes the System Timer and its interrupt, and starts the System Timer.

Interrupt Handlers

The handler for system timer is shown as follows:

```
static volatile uint32_t _dwTickCount = 0 ;
void SysTick_Handler( void )
{
 _dwTickCount ++;
```

```
.............
}
```

Using a 32-bit counter may not always be appropriate, depending on how long the system should stay up and on the tick period. In this example, a 1ms tick overflows the counter after about 50 days; this may not be enough for a real application. In that case, a larger counter can be implemented.

Wait Function

By using the global counter, it is very easy to implement a wait function taking a number of milliseconds as its parameter. Read the code for more details.

When called, the function first saves the current value of the global counter in a local variable. It adds the requested number of milliseconds, which has been given as an argument. Then, it simply loops until the global counter becomes equal to or greater than the computed value.

The interface can be called directly to wait for several milliseconds. In this example, it is called as follows to wait for 1000ms:

```
Wait(1000);
```

5.5.8. Using the Parallel Input/Output Controller (PIO)

Purpose

The SAM E70/S70/V71 devices support up to five PIO controllers and each one controls up to 32 lines. Each line can be assigned to one of four peripheral functions: A, B, C, or D.

In this example, the PIO controller manages two LEDs.

Configuring LEDs

The two PIOs connected to the LEDs must be configured as outputs, in order to turn them on or off. First, the PIOs control must be enabled in the PIO Enable Register (PIO_PER) by writing the value corresponding to a logical OR between the two LED IDs.

PIO direction is controlled by two registers; Output Enable Register (PIO_OER) and Output Disable Register (PIO_ODR). Since in this case the two PIOs must be output, the same value as before shall be written in OER.

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 the power consumption. This is done through the PIO Pull-Up Disable Register (PIO_PUDR).

In this example, LEDs are wired to pins PA23 and PC9. They are described in the following macros:

```
#define PIN_LED_0 {PIO_PA23, PIOA, ID_PIOA, PIO_OUTPUT_0, PIO_DEFAULT}
#define PIN_LED_1 {PIO_PC9 , PIOC, ID_PIOC, PIO_OUTPUT_0, PIO_DEFAULT}
```

Here is the code for LED configuration:

```
LED_Configure( 0 ) ;
LED_Configure( 1 ) ;
```

PIO_Configure will be called as shown below:

```
PIO_Configure( &pinsLeds[dwLed], 1 );
```

For LED0 wired to pin PA23, the program will run PIO_SetPeripheralA function.

When programming PIO, it is recommended to call PIO_Configure with proper parameters directly. So, the definition of relevant pins such as PIN_LED_0 and PIN_LED_1 is primary.

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, i.e., when the LED is ON according to PIO_PDSR, then it is turned off, and vice-versa.

The function is described below. PIO_GetOutputDataStatus returns the value of PIO_PDSR. The relevant interfaces which can be called directly are listed as follows:

```
unsigned char PIO_GetOutputDataStatus(const Pin *pin);
void PIO_Clear(const Pin *pin);
void PIO_Set(const Pin *pin);
```

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

5.5.9. Using the Serial Ports

Purpose

As mentioned before, the default serial port used as output is USART1 on SAM V71 Xplained Ultra.

Ensure that the board is detected as a COM device in the Device Manager once it is connected to the computer. If it is not detected, then run AtmelUSBInstaller.exe to make sure that the PC recognizes the port as a serial port and then connect the board with the PC via a Micro-AB USB cable.

The example application uses USART1 to print debug information and monitor input.

Initialization

The common interfaces which can be called directly are listed as below.

```
extern void DBG_PutChar( uint8_t c );
extern uint32_t DBG_GetChar( void );
```

Their purpose is outputting a character and inputting a character via the serial port respectively. At first, the program will check whether the console has been initialized by a static variable '_ucIsConsoleInitialized'. If not, DBG_Configure function will be called and finishes initialization and configuration.

```
if ( !_ucIsConsoleInitialized )
{
    DBG_Configure(CONSOLE_BAUDRSATE, BOARD_MCK);
}
```

It is configured with a baudrate of 115200, eight bits of data, no parity, one stop bit, and no flow control as default.

Redirecting printf

The function printf is redirected to the serial port in software package in 'dbg_console.h'.

For IAR toolchain, putchar is called by printf, so putchar is redefined by calling DBG_PutChar, which outputs a character on the serial port as shown below:

```
extern WEAK signed int putchar( signed int c )
{
DBG_PutChar( c ) ;
return c ;
}
```

For MDK toolchain, the relevant interface is fputc. It is performed as follows:

```
int fputc(int ch, FILE *f)
{
if ((f == stdout) || (f == stderr))
{
DBG_PutChar(ch) ;
          return ch ;
 }
 else
      {
return EOF ;
      }
}
```

For GNU toolchain, the relevant interface is _write. It is performed as follows:

```
extern int _write( int file, char *ptr, int len )
{
     int iIndex ;
for ( iIndex=0 ; iIndex < len ; iIndex++, ptr++ )
     {
         DBG_PutChar( *ptr ) ;
}
     return iIndex ;
}
```

Note: For IAR and MDK, the standard library functions are available in stdio.h.

6. Get Started with Atmel Studio 6

6.1. Requirements

- Atmel Studio 6.2 SP2 (or above version) installed
- Atmel Software Framework (ASF) 3.25 installed

Note: *If ASF is already installed the latest partpack can be downloaded from Atmel gallery link:* https://gallery.atmel.com/Products/Details/6f04539f-2222-477a-92c8-2ef62a28f09a

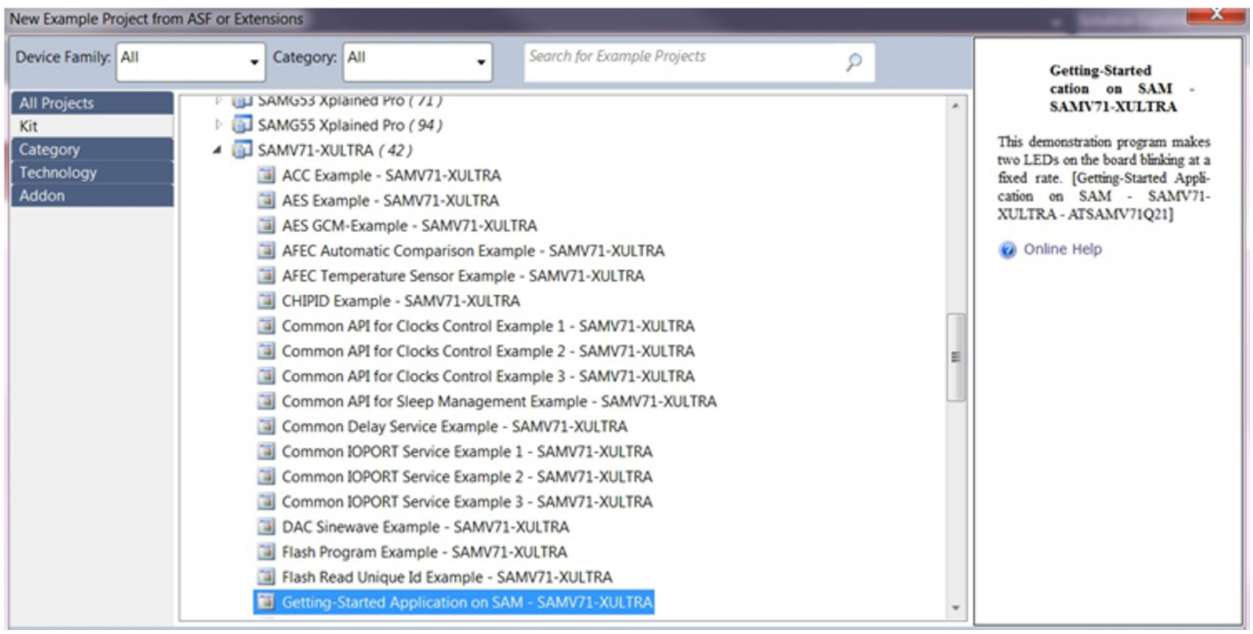
- SAM V71 Xplained Ultra Evaluation Kit connected to IAR Embedded Workbench running on PC and powered on

Note: *Connect a USB cable (Standard-A to Micro-B or Micro-AB) between the PC and the DEBUG USB port on the kit).*

6.2. Load the Example

- Launch Atmel Studio
- Open the example selection menu in ASF from Atmel Studio: File → New → Example Project from ASF...
- Select the "Kit" view and select SAM V71 Xplained Ultra Evaluation Kit in the latest ASF
- Select "Getting Started Application on SAM - SAMV71-XULTRA". This can be shown in Figure 6-1 Getting Started Project in Atmel Studio on page 27.

Figure 6-1 Getting Started Project in Atmel Studio



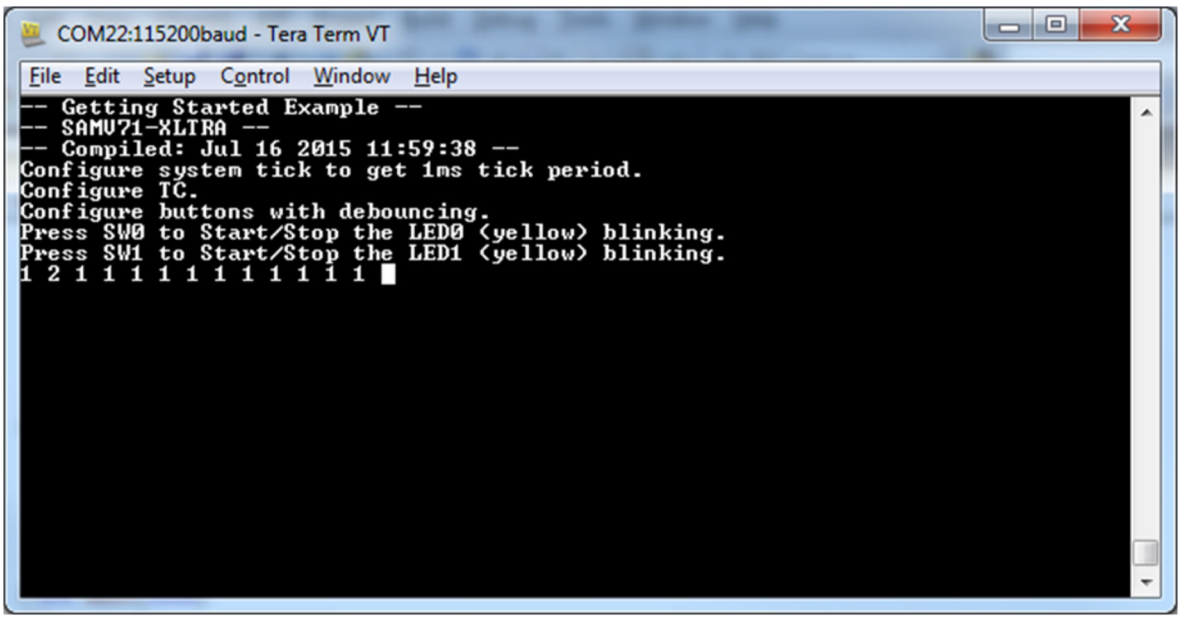
- Accept the license agreement (during the first time) and press Finish. Then the Atmel Studio will open the example.
- Build the project: Build → Build Solution.
- In the computer, open and configure a terminal application (e.g. Terminal on Microsoft Windows) with these settings:

- Port: Same port where EDBG Virtual COM Port is connected.

Note: *To know this port, go to your computer → Device Manager → Open Ports (COM & LPT) in on Microsoft Windows for example.*

- 115200 baud rates
- Eight bits of data
- No parity
- One stop bit
- No flow control
  - Load the code in SAM V71 Xplained Ultra Evaluation Kit and start debugging: Debug → Start Debugging and Break.
  - Now the application has been programmed and the debugger stops at the beginning of main(). To execute it, click on Debug → Continue.
  - The demonstration program makes one LED (LED0) on the board blink at a fixed rate
  - The blinking of LED0 can be stopped or re-started by using SW0 button
  - The blinking of LED1 can be stopped or re-started by using SW1 button. See Figure 6-2 Atmel Studio – Getting Started Project on Terminal Window on page 28.

Figure 6-2 Atmel Studio – Getting Started Project on Terminal Window



7. Get Started with IAR EWARM

7.1. Requirements

- IAR Embedded Workbench for ARM 7.40.1 or later version installed
- SAM V71 Xplained Ultra Evaluation Kit connected to IAR Embedded Workbench running on PC and powered on

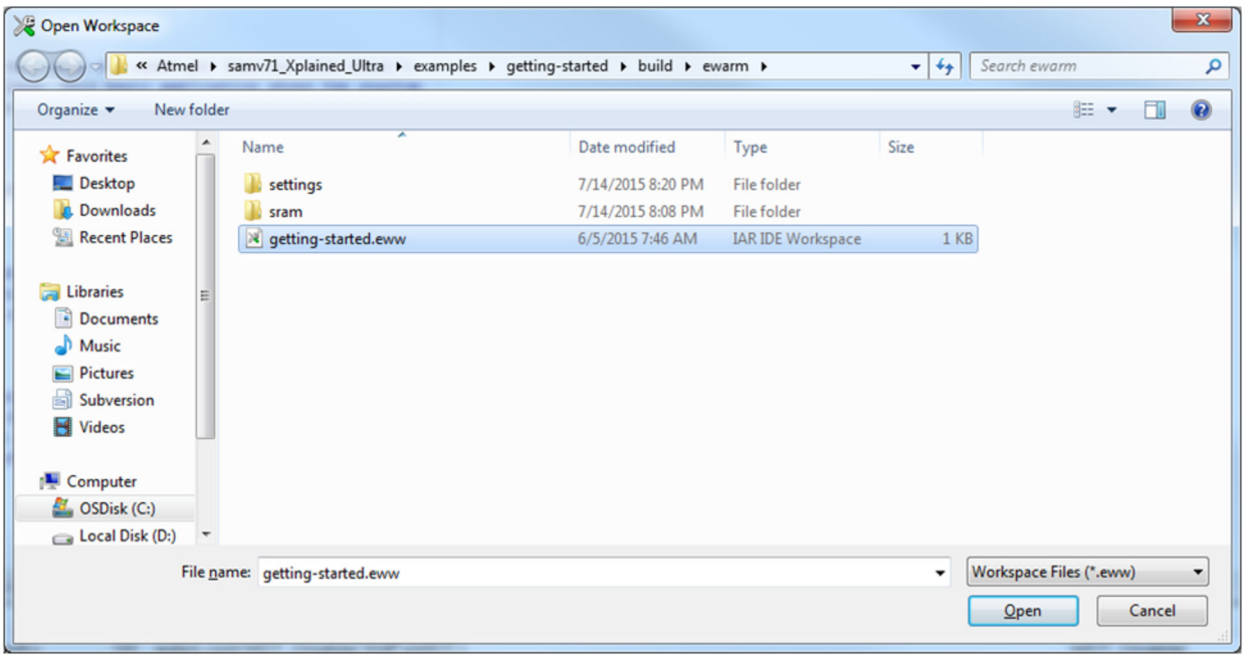
Note: *Connect a USB cable (Standard-A to Micro-B or Micro-AB) between the PC and the DEBUG USB port on the kit).*

7.2. Load the Example

Ensure that the SAM V71-XULT IAR EWARM 7.40.1 Software Package 1.3 is already downloaded as mentioned in Chapter 4.

- Open IAR Embedded Workbench
- Open the example project file for SAM V71 Xplained Ultra Evaluation Kit (from installation folder: arm\examples\Atmel\samv71_Xplained_Ultra\examples\getting-started\build\ewarm). See Figure 7-1 IAR – Getting Started Project Loading on page 29.

Figure 7-1 IAR – Getting Started Project Loading

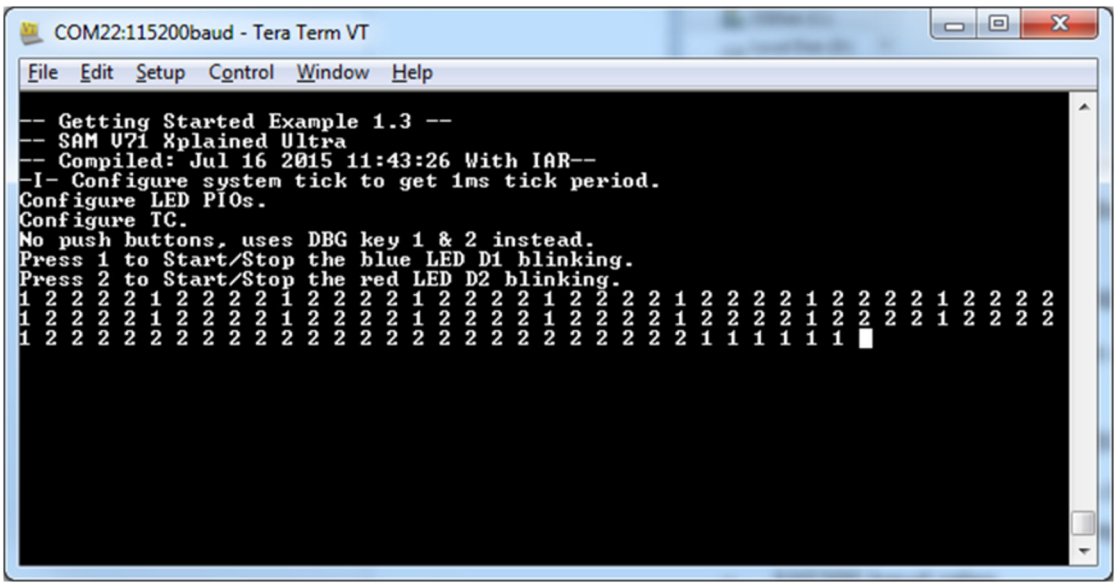


- In the Project → Options, Select Debugger option as 'CMSIS DAP'. Also inside CMSIS DAP → JTAG/SWD tab → select Interface as SWD.
- Build the project: Project → Rebuild All.
- In the computer, open and configure a terminal application (e.g. Terminal on Microsoft Windows) with these settings:
  - Port: Same port where EDBG Virtual COM Port is connected.

Note: *To know this port, go to your computer → Device Manager → Open Ports (COM & LPT) in on Microsoft Windows for example.*

- 115200 baud rates
- Eight bits of data
- No parity
- One stop bit
- No flow control
  - Load the code in SAM S70/E70 and start debugging: Project → Download and Debug.
  - Now the application has been programmed and the debugger stops at the beginning of main(). To execute it, click on Debug → Go.
  - The demonstration program makes two LEDs on the board blink at a fixed rate
  - The blinking can be stopped by typing '1' or '2' in the Hyperterminal console (one for each LED). See Figure 7-2 IAR – Getting Started Project on Terminal Window on page 30.

Figure 7-2 IAR – Getting Started Project on Terminal Window



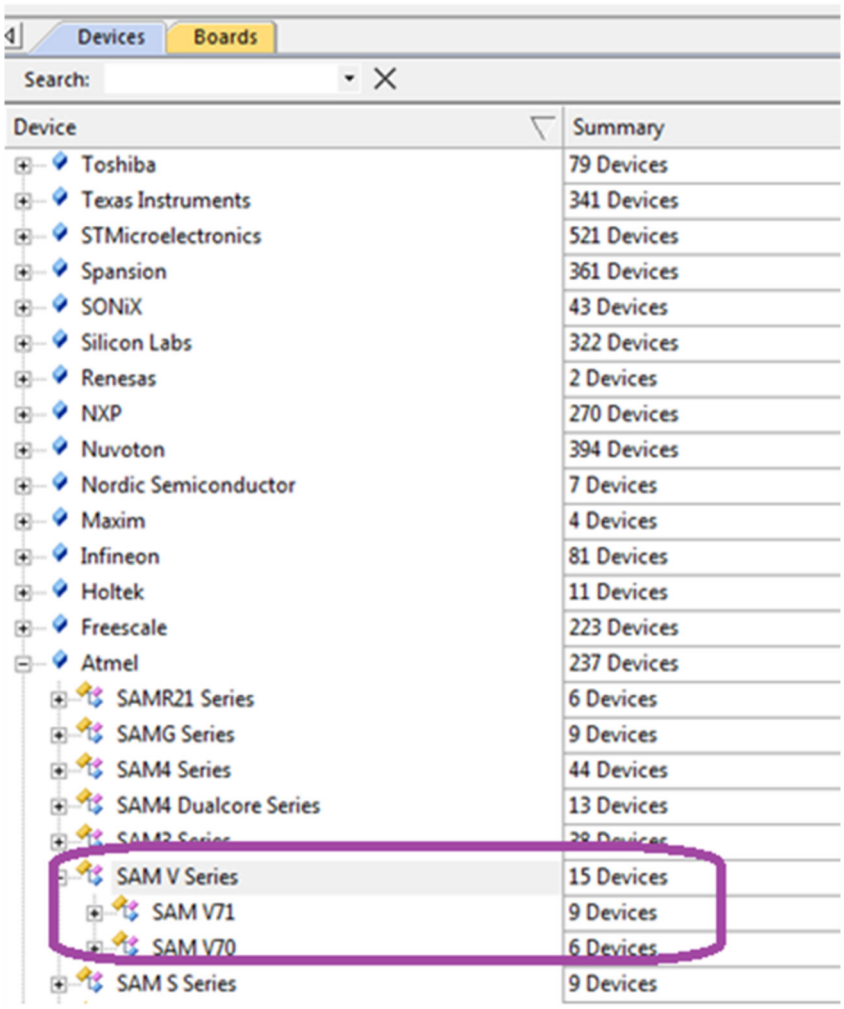
8. Get Started with KEIL MDK

8.1. Requirements

- Keil MDK version 5.14 or later version installed. The download link is: https://www.keil.com/download/product/.

Note: *The device support for Cortex M7 need to be updated by using the Pack Installer software, if SAM E/S/V is not supported in current version as shown below (refer to Purple box). See* Figure 8-1 Pack Installer → Atmel Device Selection *on page 31.*

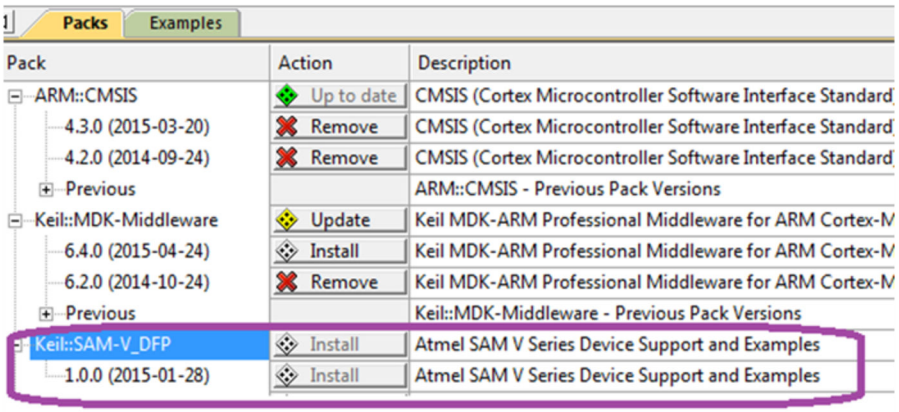
Figure 8-1 Pack Installer → Atmel Device Selection



Click on 'SAM V Series', so that the Keil::SAM-V_DFP download pack is available.

Figure 8-2 Pack Installer → Packs Installer for SAM V MCU on page 32

Figure 8-2 Pack Installer → Packs Installer for SAM V MCU



Once 'Install' is clicked, the Keil SAM-V_DFP 2.0.0. download pack is available at ..\Keil_v5\ARM\Pack\.Download

Click 'Keil.SAM-V_DFP.2.0.0.pack' to download it.

- SAM V71 Xplained Ultra Evaluation Kit connected to IAR Embedded Workbench running on PC and powered on

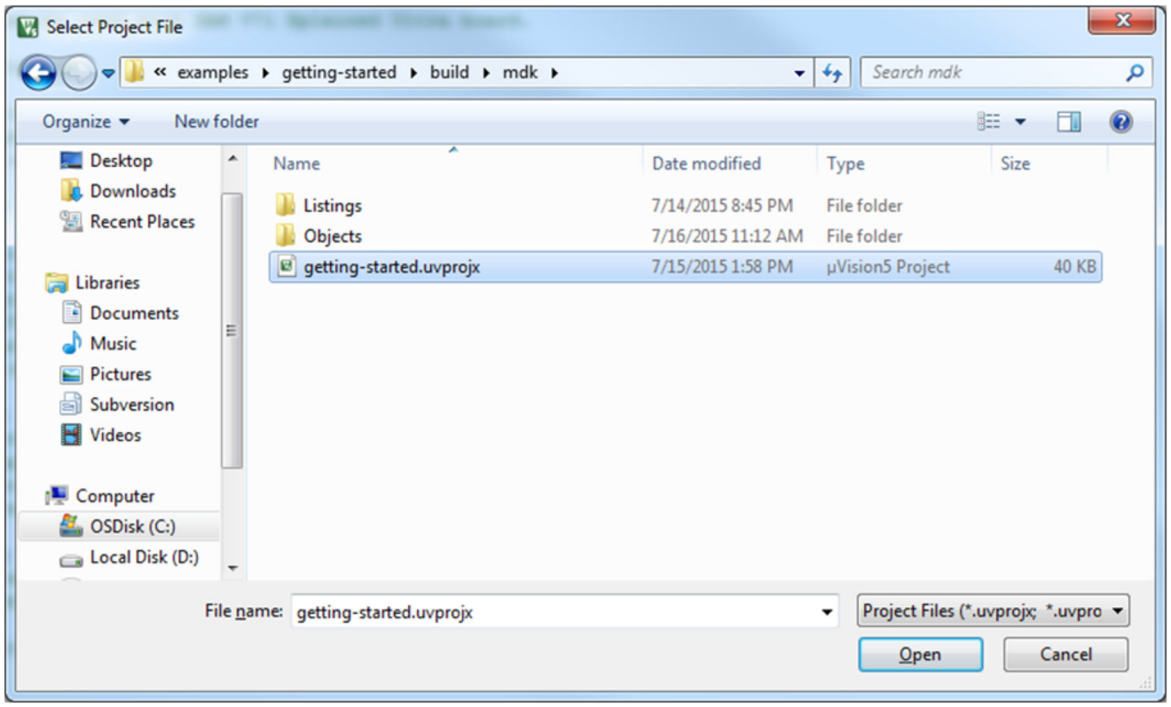
Note: *Connect a USB cable (Standard-A to Micro-B or Micro-AB) between the PC and the DEBUG USB port on the kit)*.

8.2. Load the Example

Ensure that the Keil MDK software package is downloaded as mentioned in Chapter 4.

- Open an example project file for SAM V71 Xplained Ultra Evaluation Kit from the installation folder → ARM\examples\Atmel\SAMV71_Xplained_Ultra\examples\getting-started path. See Figure 8-3 Keil MDK – Getting Started Project Loading on page 33.

Figure 8-3 Keil MDK – Getting Started Project Loading

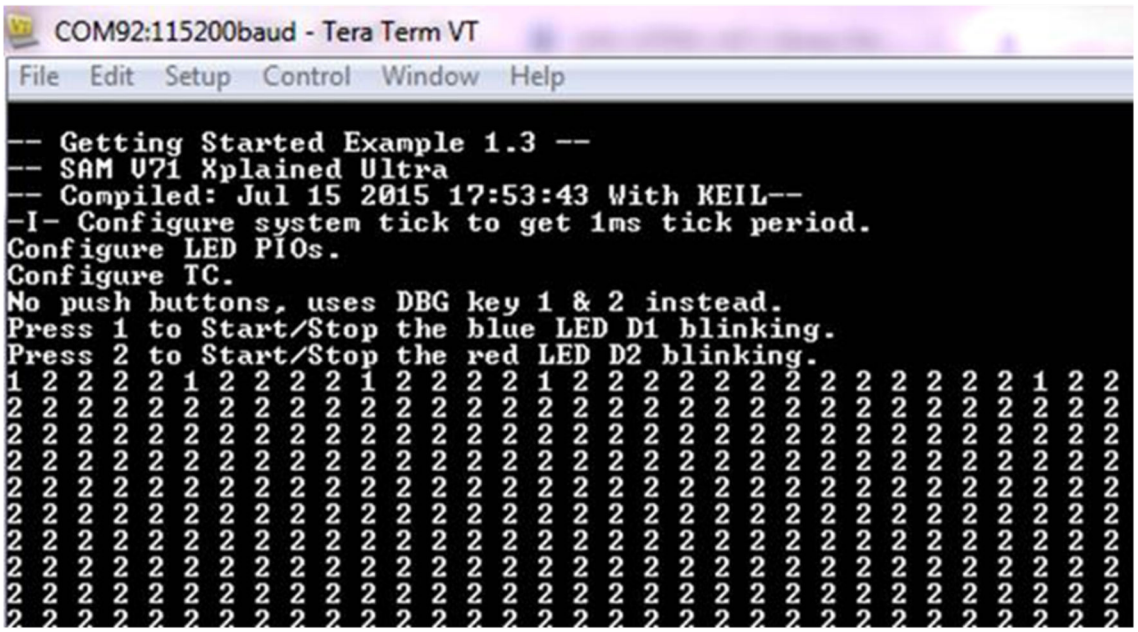


- Build the project: Project → Build Target.
- Load the code in SAMS70/ E70 and start debugging: Debug → Start/Stop Debug Session.
- In the computer, open and configure a terminal application (e.g. Terminal on Microsoft Windows) with these settings:
  - Port: Same port where EDBG Virtual COM Port is connected.

Note: *To know this port, go to your computer → Device Manager → Open Ports (COM & LPT) in on Microsoft Windows for example.*

- 115200 baud rates
- Eight bits of data
- No parity
- One stop bit
- No flow control
  - Now the application has been programmed and the debugger stops at the beginning of main(). To execute it, click on Debug → Run.
  - The demonstration program makes two LEDs on the board blink at a fixed rate.
  - The blinking can be stopped by typing '1' or '2' in the Hyperterminal console (one for each LED). See Figure 8-4 Keil MDK – Getting Started Project on Terminal Window on page 34.

Figure 8-4 Keil MDK – Getting Started Project on Terminal Window



9. Get Started with GNU Tools

To know more how to use the GNU tool for ARM embedded processor, refer to Section 3.1 of the document available in the link: http://www.atmel.com/images/atmel-44031-32-bit-cortex-m7-microcontroller-geting-started-sam-v71-microcontroller_applicationnote.pdf.

10. Get Started with SAM-BA

10.1. Requirements

- Atmel Studio 6.2 (or above version) installed
- Atmel Software Framework (ASF) 3.25 installed
- SAM-BA v2.15 (from the link: http://www.atmel.com/tools/atmelsam-bain-systemprogrammer.aspx)
- SAM V71 Xplained Ultra Evaluation Kit connected a PC and powered on

Note: *Connect a USB cable (Standard-A to Micro-B or Micro-AB) between the PC and the DEBUG USB port on the kit).*

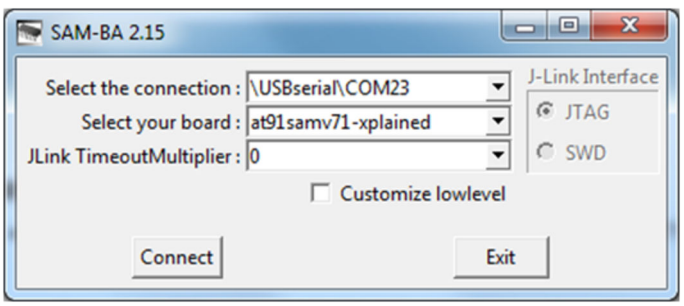
10.2. Build the Binary File

- Open the Atmel Studio command line: Start → All Programs → Atmel → Atmel Studio 6.2 Command Prompt.
- Change the directory where the example makefile is (\asf-standalone-archive-3.25.0.20\xdk-asf-3.25.0\sam\applications\getting-started\samv71q21_samv71_xplained_ultra\gcc)
- Type “make” and enter. Then the binary file (getting-started_flash.bin) will be generated in the directory.
- The binary file generated by IAR can be programmed by SAM-BA as well. About how to generate binary files by IAR, refer to IAR C/C++ Development Guide for ARM provided by IAR Embedded Workbench for ARM.

10.3. Load the Example

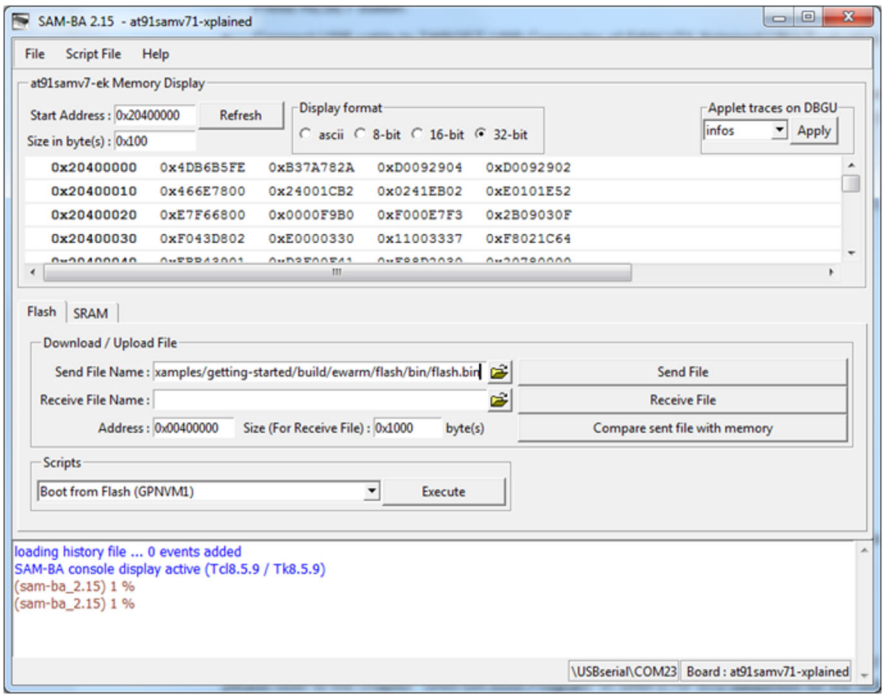
- Connect PB12 with 3.3V in ERASE connector of SAM V71 Xplained Ultra Evaluation Kit to erase the flash. Press the RESET button.
- Connect a USB cable to the TARGET USB Connector of SAM V71 Xplained Ultra Evaluation Kit and press the RESET button
- Now SAM V71 will get enumerated with PC if the SAM-BA driver is installed. If the driver is not installed the driver can be found from the SAM-BA installation path (default path: C:\Program Files (x86)\Atmel\sam-ba_2.15\drv).
- Open SAM-BA
- Select the port where the USB device is enumerated and then select samv71-xplained as the target board. Press Connect as shown below. See Figure 10-1 SAM-BA Connection Setting on page 36.

Figure 10-1 SAM-BA Connection Setting



- The SAM-BA GUI opens. In SAM-BA GUI, choose Flash tab.
- For Send File Name, choose the binary file (getting-started_flash.bin) generated previously. See Figure 10-2 SAM-BA Bin File Selection for Programming on page 37.

Figure 10-2 SAM-BA Bin File Selection for Programming



- Specify the address (0x400000) and press Send File
- For Scripts select Boot from Flash (GPNVM1) and then press Execute
- Now the application has been programmed. To execute it, reset the board.
- You can see the LED toggling indicating the flash is programmed

Besides J-Link, UART and USB can be used for the communication between SAM-BA and SAM E70/S70, refer to chapter “SAM-BA Boot Program” in the SAM S70/E70 datasheet for details.

To learn more about SAM-BA, refer to the AT91 ISP/SAM-BA user guide document available at the following link: http://www.atmel.com/images/6421b.pdf. Or simply use the SAM-BA user guide document located in C:\Program Files (x86)\Atmel\sam-ba_X.xx\doc.

11. Real Time Operating System Support

Following is a list of real time operating systems supported for SAMS70/ E70 MCUs.

- Keil RTX: www.keil.com/pack/Keil.SAM-V_DFP.pdsc
  - Application note reference: Migrating Application Code from ARM Cortex-M4 to Cortex-M7 Processors: http://www.keil.com/appnotes/files/apnt_270.pdf
- FreeRTOS: http://www.freertos.org/Atmel_SAMV7_Cortex-M7_RTOS_Demo.html
  - Application note reference: Atmel AT04056: Getting Started with FreeRTOS on Atmel SAM Flash MCUs
- NuttX: http://www.nuttx.org/Documentation/NuttX.html#at91samv71
- Segger embOS: https://www.segger.com/embos-for-cortex-m-cpus-and-rowley-compiler.html and https://www.segger.com/download_embos-arm-cortex-m-atmel-studio.html
- ExpressLogic ThreadX: http://rtos.com/downloads/threadx_demo/

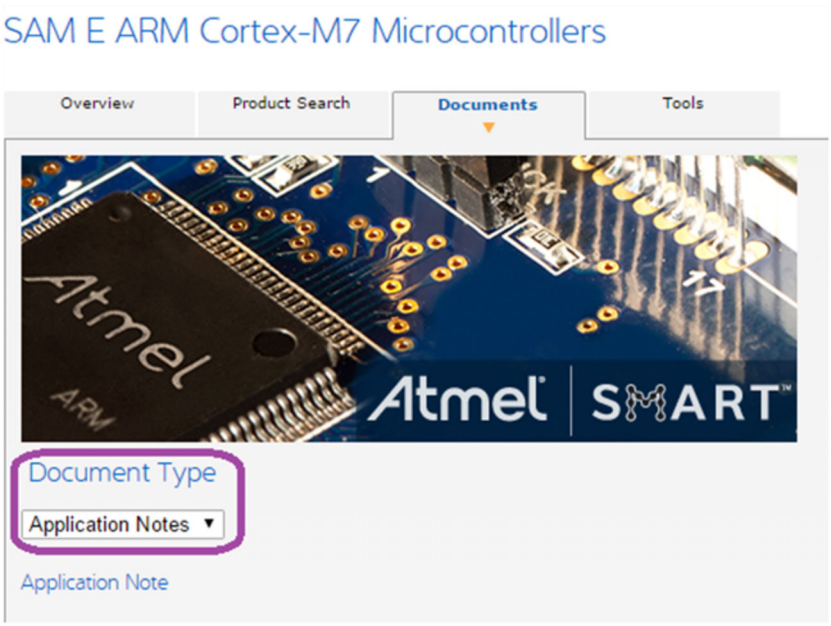
12. Application Notes

Following is a list of application notes supported for SAM S70/E70 MCUs from Atmel.

- AT04056: Getting Started with FreeRTOS on Atmel SAM Flash MCUs
- AT06015: Production Programming of Atmel Microcontrollers
- AT09331: ASF USB Stack Manual
- AT09332: USB Device Interface (UDI) for Communication Class Device (CDC)
- AT09333: USB Host Interface (UHI) for Communication Class Device (CDC)
- AT09334: USB Device Interface (UDI) for Human Interface Device Generic (HID Generic)
- AT09335: USB Device Interface (UDI) for Human Interface Device Keyboard (HID Keyboard)
- AT09336: USB Device Interface (UDI) for Human Interface Device Mouse (HID Mouse)
- AT09337: USB Host Interface (UHI) for Human Interface Device Mouse (HID Mouse)
- AT09338: USB Device Interface (UDI) for Mass Storage Class (MSC)
- AT09339: USB Host Interface (UHI) for Mass Storage Class (MSC)
- AT09340: USB Device Interface (UDI) for Vendor Class Device
- AT09341: USB Host Interface (UHI) for Vendor Class Device
- AT09423: SAM-BA Overview and Customization Process
- How to Optimize Usage of SAM V7x/E7x/S7x Architecture

To know more about existing application notes, select one of the links: http://www.atmel.com/products/microcontrollers/arm/sam-s.aspx?tab=documents or http://www.atmel.com/products/microcontrollers/arm/sam-e.aspx?tab=documents and select the document type as Application Notes as shown in the below figure (inside Purple Box). See Figure 12-1 SAM V71 Xplained Ultra Evaluation Kit Link on page 39.

Figure 12-1 SAM V71 Xplained Ultra Evaluation Kit Link



13. References

- SAM E70 Device Datasheet: http://www.atmel.com/Images/Atmel-11296-32-bit-Cortex-M7-Microcontroller-SAM-E70Q-SAM-E70N-SAM-E70J_Datasheet.pdf
- SAM S70 Device Datasheet: http://www.atmel.com/Images/Atmel-11242-32-bit-Cortex-M7-Microcontroller-SAM-S70Q-SAM-S70N-SAM-S70J_Datasheet.pdf
- Atmel Studio 6.2 sp2 Readme: http://www.atmel.com/Images/AStudio6_2sp2_1563-readme.pdf
- Atmel Software Framework 3.25 Release Notes: http://www.atmel.com/Images/asf-releasenotes-3.25.0.pdf
- SAM V71 Xplained Ultra: http://www.atmel.com/Images/Atmel-42408-SAMV71-Xplained-Ultra_User-Guide.pdf
- SAM V71-XULT Software Package 1.3 Release note: http://www.atmel.com/Images/samv71_softpack_release_note_v1.3.txt
- Smart SAM E70 SAM S70 Arm Cortex-M: http://www.atmel.com/Images/45131A-SAM-S70-E70_E_A4_021015_web.pdf
- Getting Started with SAM V71 Microcontrollers: http://www.atmel.com/images/atmel-44031-32-bit-cortex-m7-microcontroller-geting-started-sam-v71-microcontroller_applicationnote.pdf

14. Revision History

Doc Rev.	Date	Comments
42532A	09/2015	Initial document release.

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**