SMART ARM-based Microcontrollers

AT12859: USB Host MSC Class For SAM S70/E70/V70

APPLICATION NOTE

Introduction

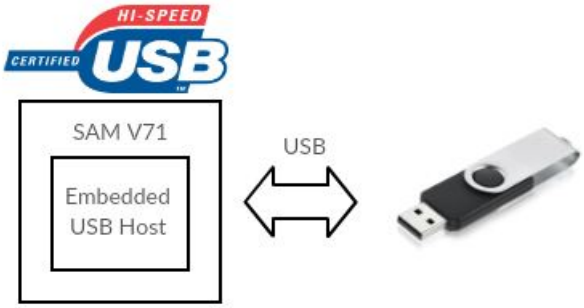
This document introduces the USB Host MSC stack available in ASF. The aim of this document is to describe how to start with the existing ASF example application and easy way to integrate a USB embedded Host application on a new or existing project.

Table of Contents

1. USB Introduction

The Universal Serial Bus (USB) is a technology that allows the user to connect a USB device such as mouse, keyboard, flash drive, etc. to a USB Host (PC/computer).

Figure 1-1. USB Host MSC



The following three main components are necessary for USB Communication:

1. USB Host (Computer/PC, etc.)
2. USB Devices.
3. USB Cable that links the USB device with the USB Host.

2. USB Host Introduction

2.1. Generic Description of USB Host

Normally, USB communication occurs between a host and a computer peripheral. The host is a PC or another computer with host-controller hardware. The USB Host initiates all communication on the bus.

The host computer contains a USB host controller hardware layer and a software layer.

USB hardware layer is responsible for:

- detecting the attachment and removal of USB devices
- monitoring device status and collecting activity statistics
- providing power to attached USB devices
- managing control and data flow between the USB host and USB devices

USB software layer is responsible for:

- handling USB devices and their connectivity
- USB devices enumeration and configuration
- loading appropriate device drivers
- managing the power on the bus
- managing the data transfer between the software and hardware

2.2. SAM V71 Peripheral Support for USB Host

- Compatible with the USB 2.0 specification
- Supports High-speed (480Mbps), Full-speed (12Mbps), and Low-speed (1.5Mbps) communication
- Ten Pipes/Endpoints
- 4096 bytes of Embedded Dual-Port RAM (DPRAM) for Pipes/Endpoints
- Up to three Memory Banks per Pipe/Endpoint (not for Control Pipe/Endpoint)
- Flexible Pipe/Endpoint configuration and management with dedicated DMA channels
- On-Chip UTMI transceiver including Pull-ups/Pull-downs

3. Abbreviations

Table 3-1. Abbreviations

Abbreviation	Definition
ASF	Atmel Software Framework
FS	Full Speed
HS	High Speed
LUN	Logical Unit Number
LS	Low Speed
MSC	Mass Storage Class
SCSI	Small Computer System Interface
SOF	Start Of Frame
UHC	USB Host Controller
UHD	USB Host Descriptor
UHI	USB Host Interface
UHS	USB High Speed
USB	Universal Serial Bus
ZLP	Zero Length Packet

4. USB Host MSC Architecture in ASF

The USB Host stack is divided into three parts.

1. The USB Host Controller (UHC), providing USB chapter 9 compliance.
2. The USB Host Interface (UHI), providing USB Class Compliance.
3. The USB Host Driver (UHD) provides the USB interface.

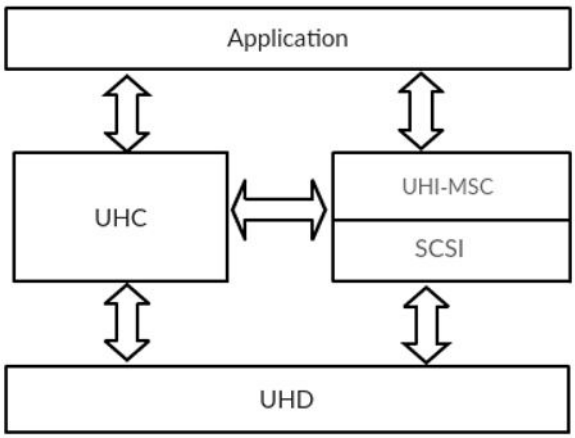
USB Host Controller (UHC): This layer implements services for the default control pipe and those services comply with Chapter 9 in the USB 2.0 Specification.

USB Host Interface (UHI): This layer includes implementation of class-specific requests.

The USB Host Driver (UHD): The layer includes drivers for low level code for programming USB controller for USB communication.

Note: The USB host drivers are implemented in full interrupt mode, thus this UHD is a perfect base to create a USB driver for third party's USB stacks.

Figure 4-1. USB MSC Architecture



The following table provides information about the organization of the files in the USB Host MSC example.

Table 4-1. USB Host MSC

Modules	Files	ASF Path	Description

Application	`main.c`
	`ui.c`
	`ui.h`
	`conf_ucb_host.h`
		`Example folder`
		`common/services/usb/class/hid/host/mouse/example/samv71q21_samv71_xplained_ultra`
			Main loop
			Set up hardware configuration
			USB Host Configuration

UHI – MSC Host class	`uhi_msc.c`
	`uhi_msc.h`
	`uhi_msc_mem.c`
	`uhi_msc_mem.h`
		`common/services/usb/class/hid/host/mouse/example2`	Standard MSC class implementation

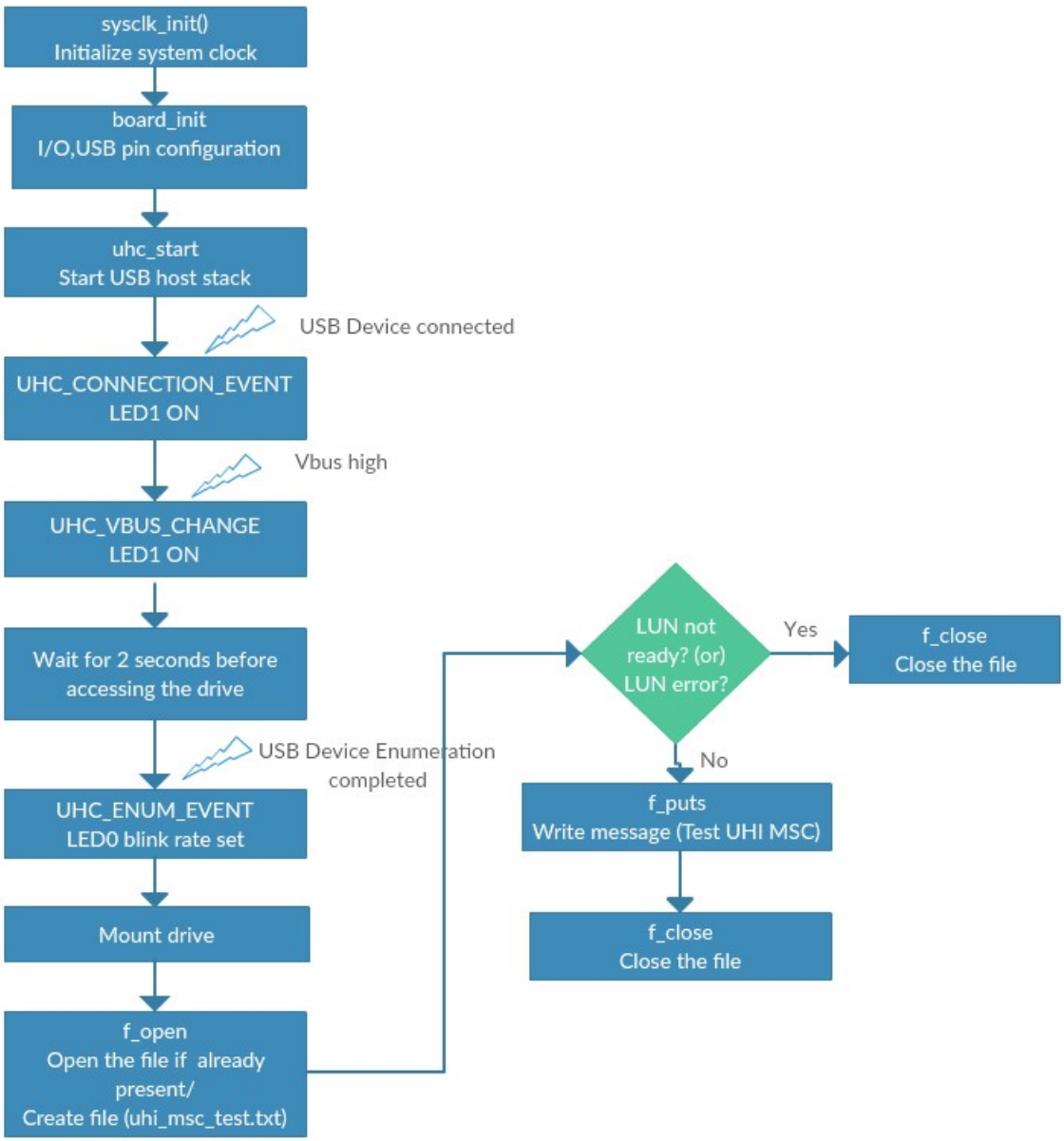
Modules	Files	ASF Path	Description
SCSI	sbc_protocol.c sbc_protocol.h usb_protocol_msc.h	common/services/usb/class/msc	SCSI Commands MSC class protocol constants
UHC	uhc.c uhc.h uhd.h uhi.h	common/services/usb/uhc	USB Host core
	usb_protocol.h usb_atmel.h	common/services/usb	USB Protocol constants USB VID, PID constants
UHD	usbhs_host.c usbhs_host.h usbhs_otg.h	ASF/sam/drivers/usbhs	USB Host drivers (Low level drivers of USB Host for SAM V71 device)

5. Understanding USB Host MSC Example in ASF

The ASF 3.28.1 or later provides the USB Host MSC class for SAM V71. This chapter provides the application overview. The firmware uses ASF 3.28.1 version. In this example application, the device system clock (i.e., the Processor clock HCLK) operates at 150MHz and the USB operates at USB High speed mode i.e., at 480MHz.

This example application configures SAM V71 device as USB MSC Host and USB mass storage device (e.g. pen-drive) is used to store the data. FatFS module is used to create file and write message to the file.

Figure 5-1. Example Application Flowchart



5.1. Hardware/Software Requirement

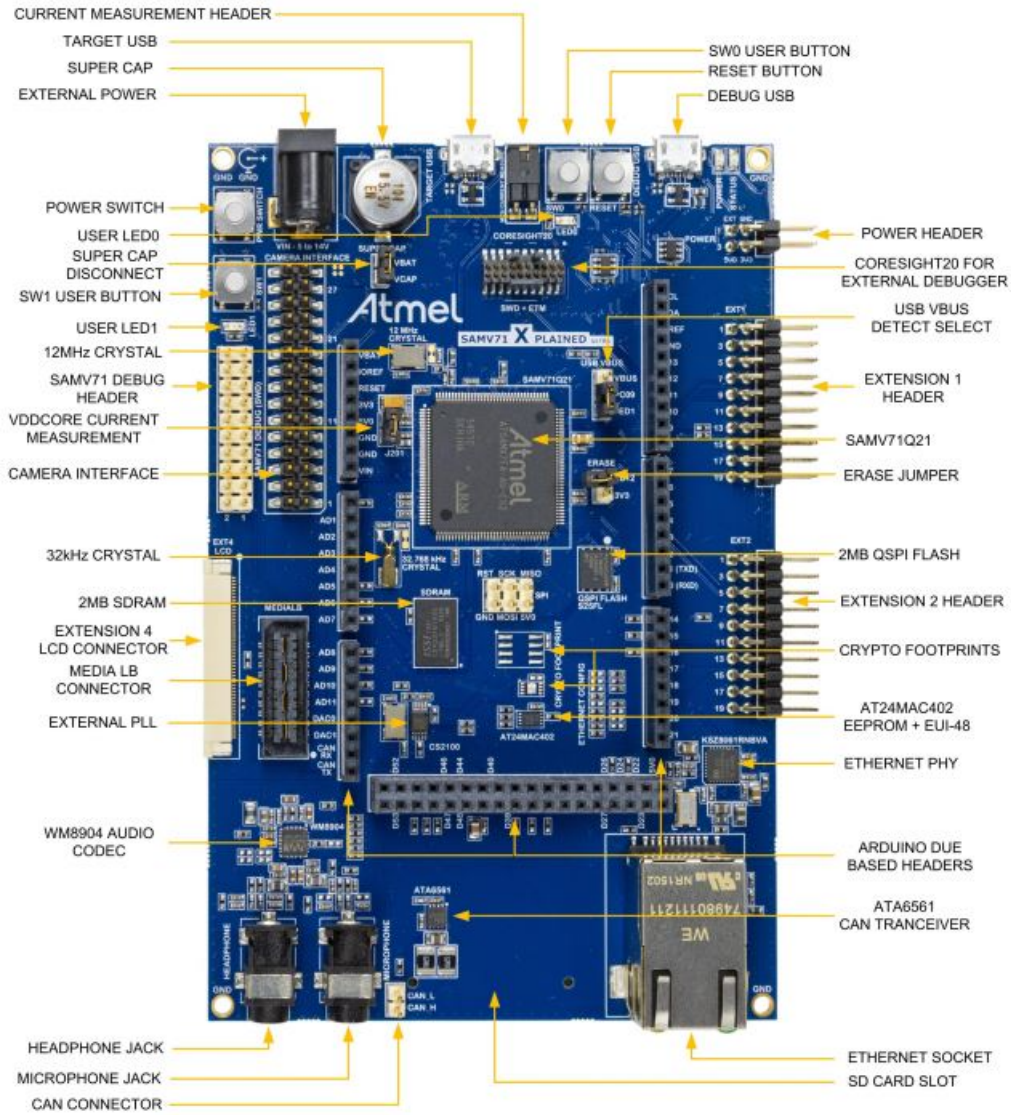
The following hardware and software environments are required to evaluate the examples.

5.1.1. Hardware Requirement

5.1.1.1. SAM V71 Xplained Ultra Evaluation Kit

The Atmel® | SMART™ SAM V71 Xplained Ultra evaluation kit is ideal for evaluating and prototyping with the Atmel SAM V71, SAM V70, SAM S70, and SAM E70 ARM® Cortex®-M7 based microcontrollers. Extension boards to the SAM V71 Xplained Ultra can be purchased individually. The ATSAMV71-XULT evaluation kit does not include extension boards. The detailed view of the SAM V71 Xplained Ultra Evaluation Kit is as follows.

Figure 5-2. SAM V71 Xplained Ultra Evaluation Kit



5.1.1.2. OTG Micro-USB Cable

5.1.1.3. USB Cable (Standard A to Micro-B USB)

5.1.1.4. USB Mass Storage

5.1.2. Software Requirement

5.1.2.1. Atmel Studio Version 7 or Later

Atmel Studio is the integrated development platform (IDP) for developing and debugging Atmel ARM Cortex-M processor-based microcontrollers and Atmel AVR microcontroller applications.

Atmel Studio can be downloaded from the following link: http://www.atmel.com/tools/ATMELSTUDIO.aspx.

5.1.2.2. Atmel Software Framework (ASF 3.28 or later)

The Atmel Software Framework (ASF) is an MCU software library providing a large collection of embedded for Atmel flash MCUs: megaAVR, XMEGA, UC3, and SAM devices.

ASF is integrated in the Atmel Studio IDE with a graphical user's interface or available as standalone for GCC, IAR compilers. ASF standalone can be downloaded from the following link: http://www.atmel.com/tools/avrsoftwareframework.aspx?tab=overview.

5.2. Hardware/Software Setup

5.2.1. Hardware Setup

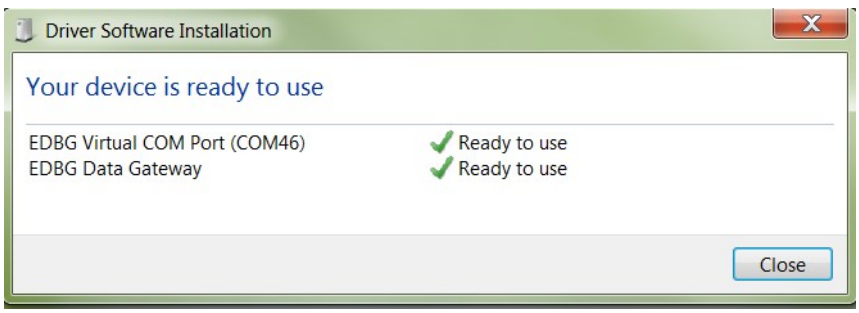
The SAM V71 Xplained Pro kit is used to run the application example. There are two USB ports on the SAM V71 Xplained Pro board; DEBUG USB and TARGET USB. For debugging using the Embedded debugger EDBG, the DEBUG USB port has to be connected in SAM V71 Xplained Pro with Debug USB connected as shown in the following figure.

Figure 5-3. SAM V71 Xplained Pro with Debug USB Connected

5.2.2. Software Setup

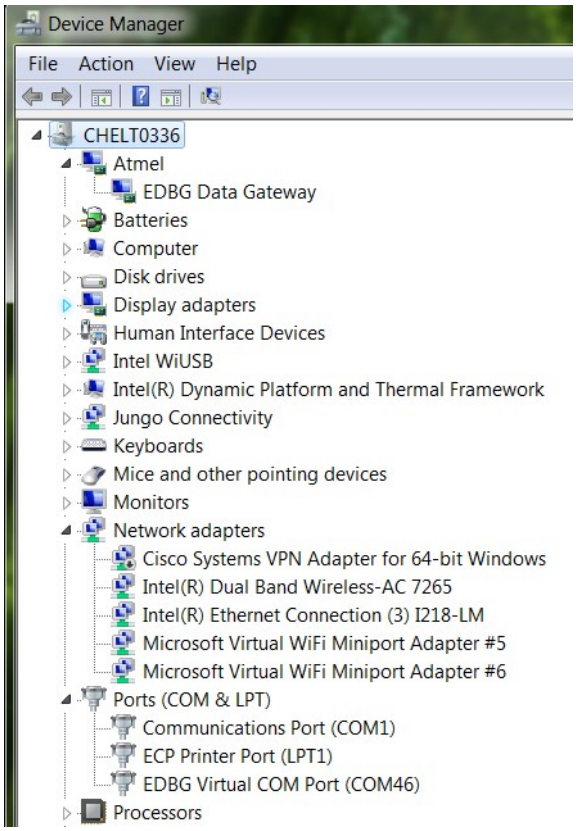
When the SAM V71 Xplained Pro kit is connected to a PC, the required drivers for the EDBG will be automatically installed. The following figure shows the **Driver Software Installation**.

Figure 5-4. SAM V71 Xplained Pro Driver Installation



If the driver installation is successful, the EDBG will be listed in the **Device Manager** window, which should display **EDBG Data Gateway** under Atmel, and **EDBG Virtaul COM Port** under **Ports (COM & LPT)** as shown in the following figure.

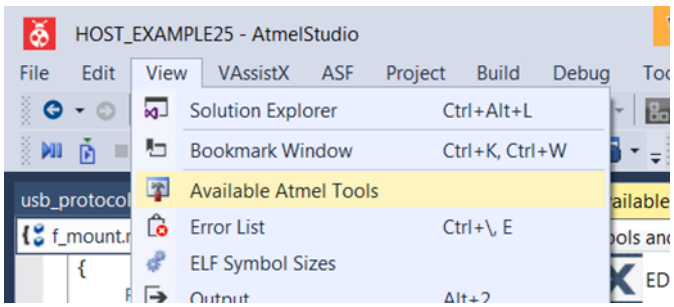
Figure 5-5. Successful EDBG Driver Installation



To ensure that the EDBG tool is detected in Atmel Studio, follow these steps:

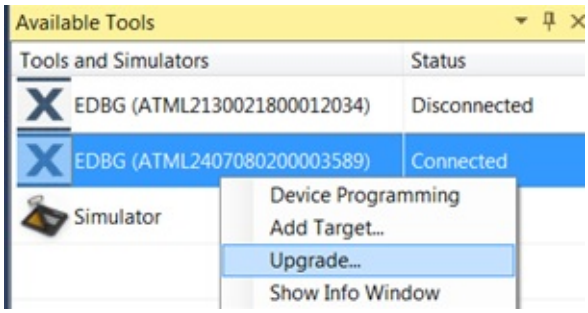
Open Atmel Studio 7 and go to Available Atmel Tools in **View** tab as shown in the following figure.

Figure 5-6. Opening Available Atmel Tools in Atmel Studio



The EDBG should be listed in the tools as “EDBG” and the tool status should display as “Connected”. This indicates that the tool is communicating successfully with Atmel Studio. If the tool is not displayed in **Available Atmel Tools**, disconnect the tool and reconnect again. To verify that the firmware of the EDBG tool is the latest, right click on the EDBG in **Available Atmel Tools** and select the option **Upgrade** as shown in the following figure.

Figure 5-7. Upgrading EDBG from Available Atmel Tools Window



If the firmware is not up-to-date, Atmel Studio will prompt for Upgrade. Click on the **Upgrade** button to upgrade the firmware. In case you get **Upgrade Failed** error, power cycle the tool and then try upgrading again.

5.3. Modules Description

5.3.1. Clock Configuration

The SAM V71 Xplained Pro board has a 12MHz crystal, connected to the PB08/PB09 pins, that is used as clock generator for the SAM V71 board. This application selects 3 to 20MHz crystal oscillator to the source of MAINCK, as it provides a more accurate frequency.

Table 5-1. External 12MHz Crystal

SAM V71 Pin	Function
PB08	XOUT

```
// ===== System Clock (MCK) Source Options
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_SLCK_RC
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_SLCK_XTAL
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_SLCK_BYPASS
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_MAINCK_4M_RC
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_MAINCK_8M_RC
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_MAINCK_12M_RC
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_MAINCK_XTAL
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_MAINCK_BYPASS
        #define CONFIG_SYSCLK_SOURCE         SYSCLK_SRC_PLLACK
        //#define CONFIG_SYSCLK_SOURCE       SYSCLK_SRC_UPLLCK

        // ===== Processor Clock (HCLK) Prescaler Options  (Fhclk = Fsys /
(SYSCLK_PRES))
        #define CONFIG_SYSCLK_PRES          SYSCLK_PRES_1
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_2
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_4
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_8
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_16
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_32
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_64
        //#define CONFIG_SYSCLK_PRES        SYSCLK_PRES_3

        // ===== System Clock (MCK) Division Options    (Fmck = Fhclk /
(SYSCLK_DIV))
        #define CONFIG_SYSCLK_DIV           2

        // ===== PLL0 (A) Options   (Fpll = (Fclk *PLL_mul) / PLL_div)
        // Use mul and div effective values here.
        #define CONFIG_PLL0_SOURCE          PLL_SRC_MAINCK_XTAL
        #define CONFIG_PLL0_MUL             25
        #define CONFIG_PLL0_DIV             1
```

```
        // ===== UPLL (UTMI) Hardware fixed at 480 MHz.
        // ===== USB Clock Source Options    (Fusb = FpllX /USB_div)
        // Use div effective value here.
        //#define CONFIG_USBCLK_SOURCE      USBCLK_SRC_PLL0
        #define CONFIG_USBCLK_SOURCE        USBCLK_SRC_UPLL
        #define CONFIG_USBCLK_DIV           1
```

The function `sysclk_init()` in `main.c` file sets up the clock system as specified in the `conf_clocks.h` file.

5.3.2. Clock Generator

The `pll_enable_source()` function enables 12MHz crystal oscillator and selects 12MHz as MAINCK clock by writing MOSCXTEN bit and the MOSCXTST in CKGR_MOR to enable the main oscillator. By doing so, the XIN and XOUT pins are automatically switched into Oscillator mode. The 12MHz Crystal oscillator is selected by writing the MOSCSEL bit in CKGR_MOR.

```
#ifdef CONFIG_PLL0_SOURCE
else if (CONFIG_SYSCLK_SOURCE == SYSCLK_SRC_PLLACK) {
            pll_enable_source(CONFIG_PLL0_SOURCE);
            pll_config_defaults(&pllcfg, 0);
            pll_enable(&pllcfg, 0);
            pll_wait_for_lock(0);
            pmc_mck_set_division(CONFIG_SYSCLK_DIV);
            pmc_switch_mck_to_pllack(CONFIG_SYSCLK_PRES);
    }
#endif
```

5.3.3. PLL Block

The 12MHz Crystal Oscillator is the source for PLL. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the parameters DIV (DIVA) and MUL (MULA). The factor applied to the source signal frequency is (MUL + 1)/DIV. `pll_enable()` function configures PLL MULA value as 24 and DIVA as 1.

So, PLLACK is 12MHz *(MUL+1)/DIV = 12MHz * (25)/1PLLACK = 300MHz

5.3.4. Master Clock Configuration

The `pmc_mck_set_division` function sets the MDIV value in CKGR_MCKR register. Setting MDIV value as 1 in CKGR_MCKR register divides the Master Clock by 2. The `pmc_switch_mck_to_plllalck` configures PLLACK from Clock generator as Master clock.

So, Master Clock (MCK) is 300MHz/2 = 150MHz.

5.3.5. USB Clock Configuration

This example application select UPLL output as the USB source clock by writing the USBS bit in PMC_USB. The source clock of the UTMI PLL is the 3 - 20MHz crystal oscillator. A 12MHz or 16MHz crystal is required to use the USB. The UPLL (UTMI) hardware frequency is fixed at 480MHz.

5.3.6. Board Configuration

5.3.6.1. USB Pin Configuration

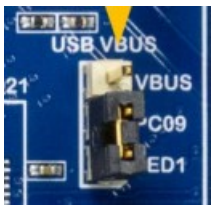
The SAM V71 Xplained Ultra has a Micro-USB connector for use with the SAM V71 USB module labeled as TARGET USB on the kit. In USB host mode VBUS voltage is provided by the kit and has to be enabled by setting the "VBUS Host Enable" pin low.

Table 5-2. USB Connections

Pin on SAM V71	USB Function
PC16	VBUS Host Enable (Active low)
HSDM	USB D-
HSDP	USB D+

There is a 1x3, 100mil pin-header marked VBUS in the kit. PC09 on the SAM V71 can be connected to either LED1 or to the target USB VBUS DETECT signal by placing a jumper between pin 1 and 2, or pin 2 and 3 respectively on this pin-header.

Figure 5-8. USB VBUS



USB VBUS DETECT is the target USB voltage divided by 1.64, when connected to the PC09 pin the signal can be used to detect power on the target USB connector.

Table 5-3. USB VBUS Selection

Pin	Function
1	LED1
2	PC09
3	USB VBUS DETECT

5.3.6.2. I/O Ports Configuration

The `board_init()` function configures the LED, Switch, and USB pins. This function configures pins PA23 and PC09 connected to LED as output, and set their default initial state to high (LED off) and configures PA09 as input, which is used for push button.

Table 5-4. Switch Functionality

SAM V71 pin	Function	Functionality
PA09	SW0	SW0 allows to enter the device in suspend mode with remote wakeup feature authorized. Only, SW0 can be used to wakeup USB device in suspend mode.

Table 5-5. LED Functionality

SAM V71 Pin	Function	Functionality
PA23	Yellow LED0	Blink when device is connected
PC09	Yellow LED1	ON when VBUS connected

5.3.7. USB Stack Configuration

The `uhc_start` function starts the USB Host Stack. The `uhd_enable` function enables the peripheral clock for USB and it configures asynchronous USB interrupts. The USB management is entirely managed by interrupts.

5.3.8. USB Host Callbacks

When the USB device is connected to the Host OTG cable, the `UHC_CCONNECTION_EVENT` is triggered and `main_usb_connection_event` callback is called.

In `main_usb_connection_event` function, LED1 in SAM V71 board is ON and enable Vbus. `UHC_SOF_EVENT` is triggered for each SOF token, and `main_usb_sof_event` callback is called and `main_usb_sof_counter` is incremented for each SOF token.

The `UHC_ENUM_EVENT` is triggered when the connected USB device enumeration is completed and `ui_usb_enum_event` callback is called where blink rate for LED0 is configured in `ui_device_speed_blink`.

When the device is enumerated successfully, Mount drive (`f_mount`) in the connected Logical Unit (Pendrive).

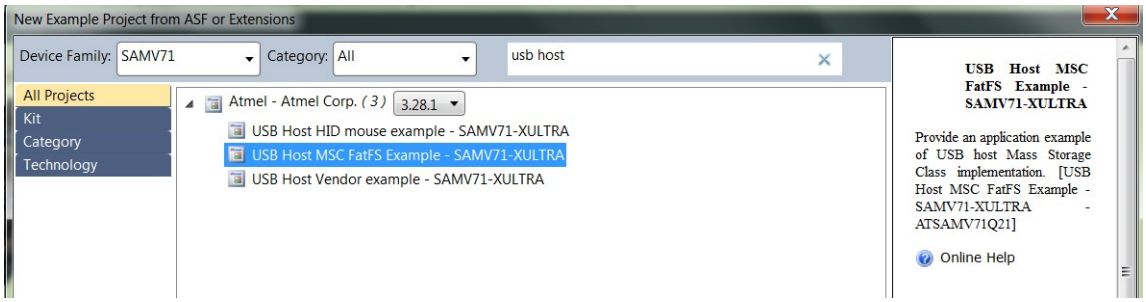
The `f_open` function creates a file on the disk with name `uhi_msc_test.txt` and `f_puts` function writes the message “Test UHI MSC” in the Logical unit connected.

```
/**
 * USB Host callbacks
 * @{
 */

//! To notify that the USB mode are switched automatically.
//! This is possible only when ID pin is available.
#define UHC_MODE_CHANGE(b_host_mode)        ui_usb_mode_change(b_host_mode)

//! To notify that the Vbus level has changed
//! Available only in USB hardware with Vbus monitoring.
#define UHC_VBUS_CHANGE(b_present)          ui_usb_vbus_change(b_present)

//! To notify that a Vbus error has occurred
//! Available only in USB hardware with Vbus monitoring.
#define UHC_VBUS_ERROR()                    ui_usb_vbus_error()

//! To notify that a device has been connected or disconnected.
#define UHC_CONNECTION_EVENT(dev,b_present) main_usb_connection_event(dev,b_present)

//! Called when a USB device or the host have wake up the USB line.
#define UHC_WAKEUP_EVENT()                  ui_usb_wakeup_event()

//! Called for each received SOF each 1 ms
//! Note: Available in High and Full speed mode
#define UHC_SOF_EVENT()                     main_usb_sof_event()

//! Called when a USB device configuration must be chosen.
//! Thus, the application can choose either a configuration number for this device
//! or a configuration number 0 to reject it.
//! If callback not defined the configuration 1 is chosen.
//#define UHC_DEVICE_CONF(dev)                uint8_t usb_device_conf(dev)

//! Called when a USB device enumeration is completed.
#define UHC_ENUM_EVENT(dev,b_status)        ui_usb_enum_event(dev,b_status)
```

5.4. Programming the Application

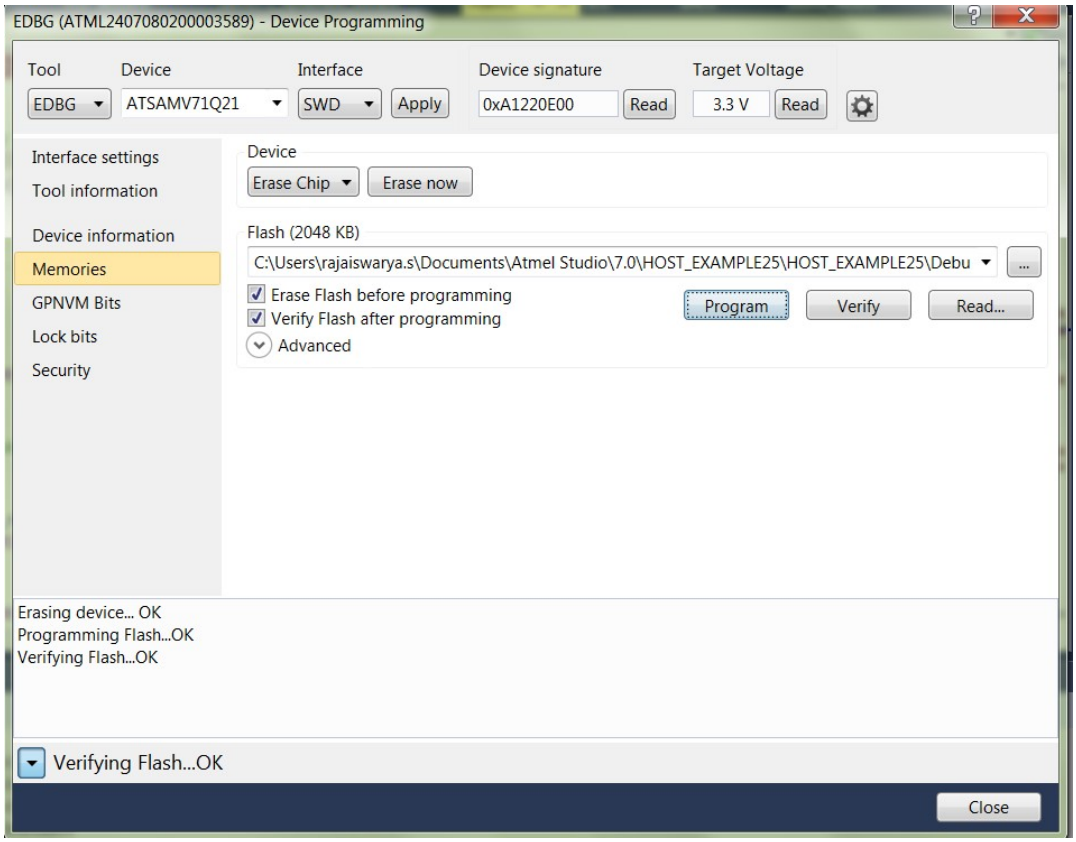
The firmware corresponding to this application is available in ASF 3.28 or later which is a part of Atmel Studio 7 or later. The following steps explain the execution of this application.

Figure 5-9. USB HOST MSC FatFS Example



1. To load the example project, go to File > New > Example Project.
2. Select **Device Family** as SAM V71 and type **USB Host** in the search box from **New Example Project.**
3. Select **USB Host MSC FatFS Example – SAMV71-XULTRA** example as shown in the preceding figure.
4. To compile the project, select Build > Build solution.
5. Open the **Device Programming** window, Tools > Device Programming.
6. Select tool as **EDBG**, device as **SAMV71Q21**, and interface as **SWD**, as shown in the following figure.
7. Give the path for `HOST_EXAMPLE.elf` file in **Memories** tab under Flash.
8. Click **Program.**

Figure 5-10. Programming Window



5.5. Result

After loading the firmware, When the USB MSC device is connected using OTG cable as shown in the following figure, the LED 1 is ON which indicates that Vbus is generated.

- LED 1 is continuously ON when the device is connected
- LED 0 blinks then the connected USB device is enumerated and USB is in idle mode;
  - The blink is slow (1s) with low speed device
  - The blink is normal (0.5s) with full speed device
  - The blink is fast (0.25s) with high speed device
- LED 1 is ON when a read or write access is on going
- LED 1 is ON when a LUN test is success
- LED 1 blinks when a LUN test is unsuccessful the `uhi_msc_test.txt` file will be created in the connected USB device
- Switch (SW0) allows to enter the device in suspend mode with remote wakeup feature authorized
- Only SW0 can be used to wakeup USB device from suspend mode

Figure 5-11. SAM V71 USB MSC Host Connection

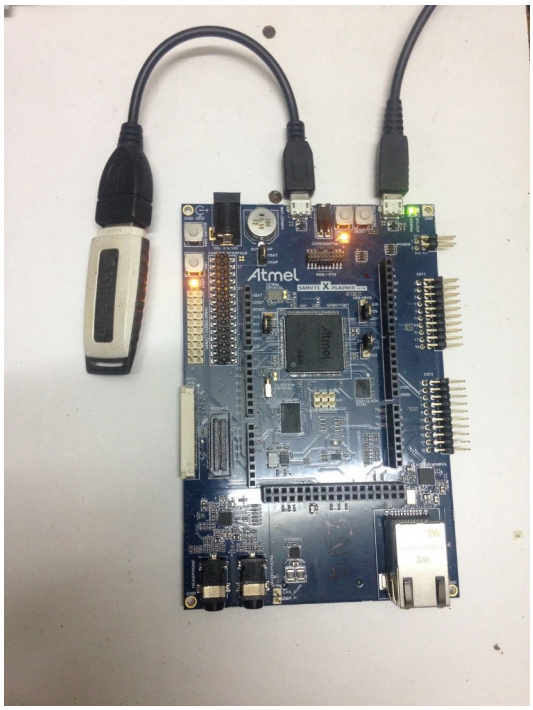
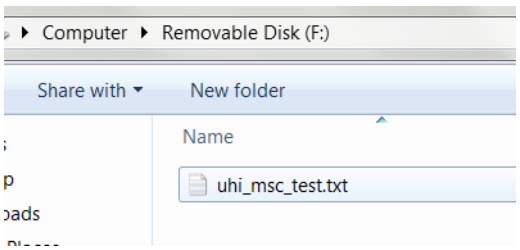


Figure 5-12. Result



5.6. Application Footprint

The application occupies the following resources when compiling with Optimization (-O1) and debug level Maximum (-g3) in ASF 3.28.1 with Atmel Studio 7.

```
Program Memory Usage : 26256 bytes 1.3 % Full
Data Memory Usage    : 12592 bytes 3.2 % Full
```

5.7. User Configuration

5.7.1. USB Full Speed Mode Configuration

By default, the example application available in the ASF is configured for USB High Speed mode. To configure USB Full speed mode in ASF, go to `conf_usb_host.h` file, comment the line `#define USB_HOST_HS_SUPPORT`.

```
#if (UC3A3||UC3A4)
# define USB_HOST_HS_SUPPORT
#elif (SAM3XA)
# define USB_HOST_HS_SUPPORT
#elif (SAMV71 || SAMV70 || SAME70 || SAMS70)
```

```
//# define USB_HOST_HS_SUPPORT
#endif
```

Uncommenting the `#define USB_HOST_HS_SUPPORT` configures the SAM V71 in USB full speed.

5.7.2. Configuring LED Blink Rate

The LED0 blink rate can be changed in variable `ui_device_speed_blink` inside the function `ui_usb_enum_event(uhc_device_t *dev, uhc_enum_status_t status)`. This function is available in the `ui.c` file, which is available in the following path `..\src\ASF\common\services\usb\class\msc\host\example2\samv71q21_samv71_xplained_ultra\ui.c`

```
void ui_usb_enum_event(uhc_device_t *dev, uhc_enum_status_t status)
{
        UNUSED(dev);
        ui_enum_status = status;
        switch (dev->speed) {
        case UHD_SPEED_HIGH:
            ui_device_speed_blink = 250;
            break;

        case UHD_SPEED_FULL:
            ui_device_speed_blink = 500;
            break;

        case UHD_SPEED_LOW:
        default:
            ui_device_speed_blink = 1000;
            break;
        }
        ui_test_done = false;
}
```

6. Adding USB Host MSC Feature to a Project

The USB Host MSC modules are available in Atmel Studio 7 ASF and can be imported into an Atmel Studio project. This section helps the user to gives the overview of USB Host MSC stack architecture in ASF and also describes the way to add an USB Host MSC in a project.

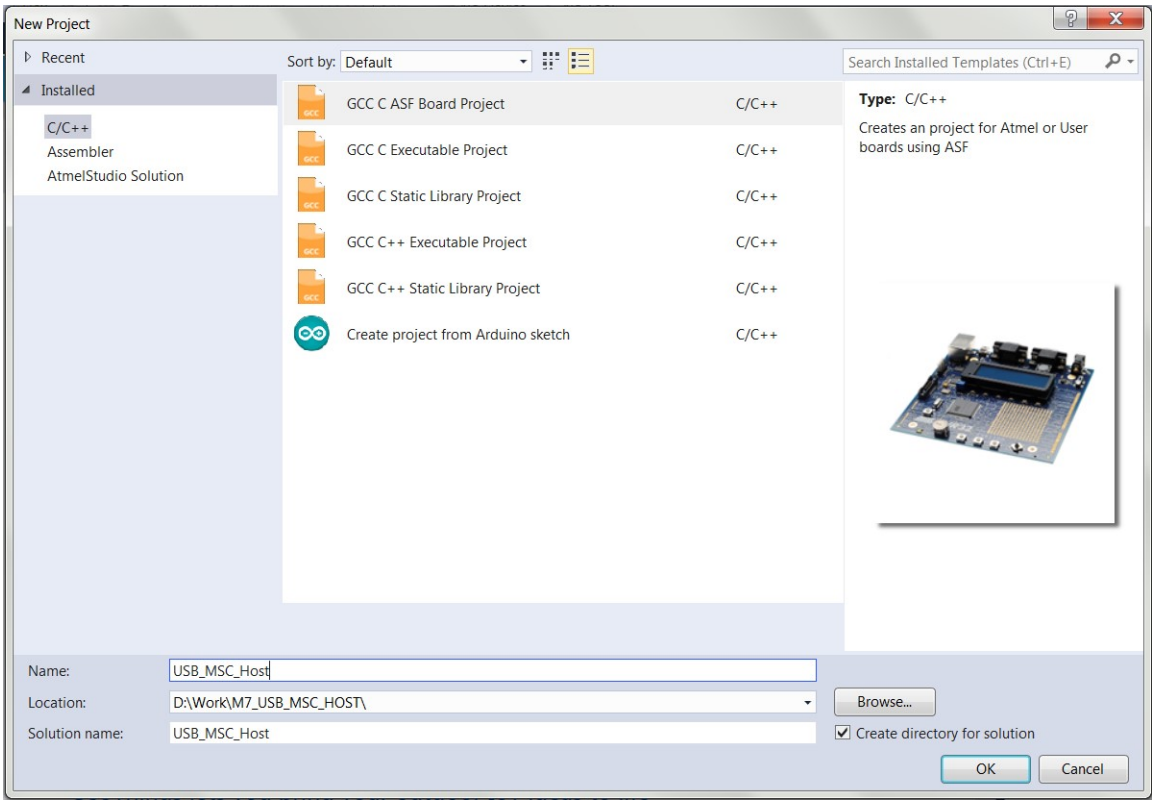
- Import USB Host MSC module to Atmel Studio project
- Configure USB parameters
- Call USB routines
- Add FatFS module

6.1. Import USB Host MSC Module

To import the USB Host MSC module to new project/ Existing project, follow the following instructions.

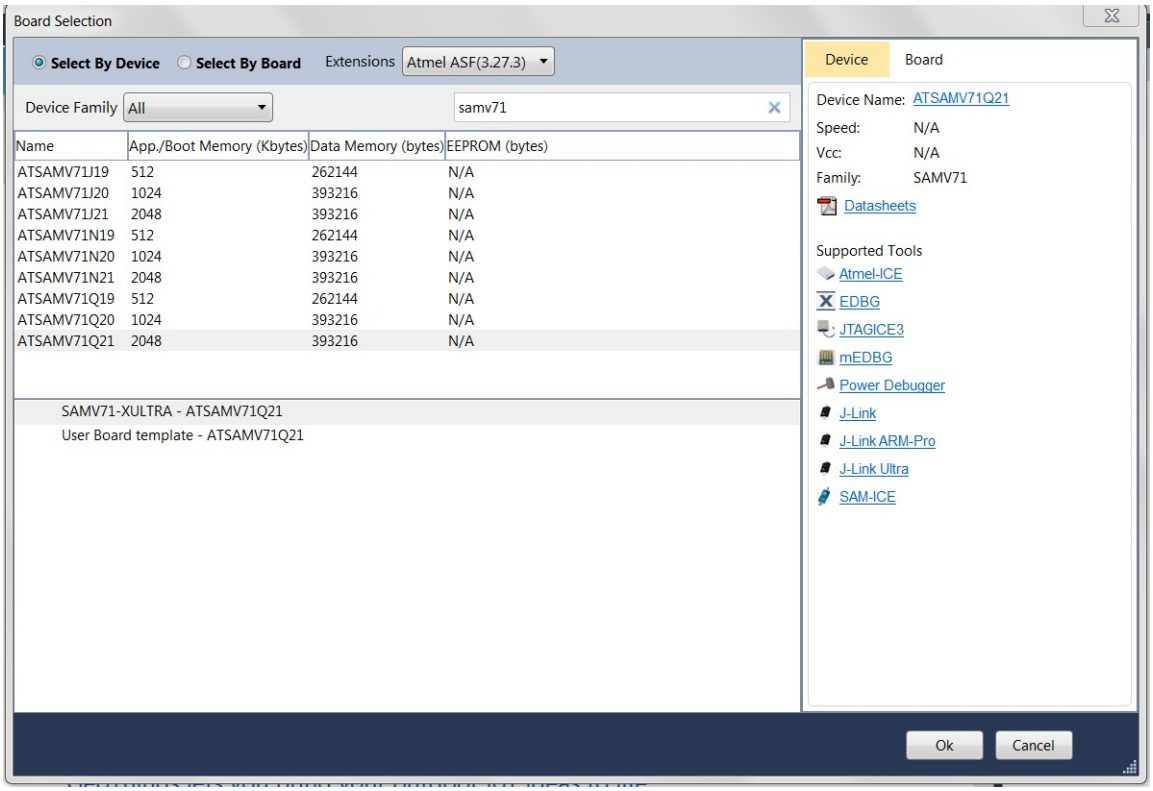
1. Open existing project (or) Create new project from File > New > Example Project. Select GCC C ASF Board Project.

Figure 6-1. Create Project



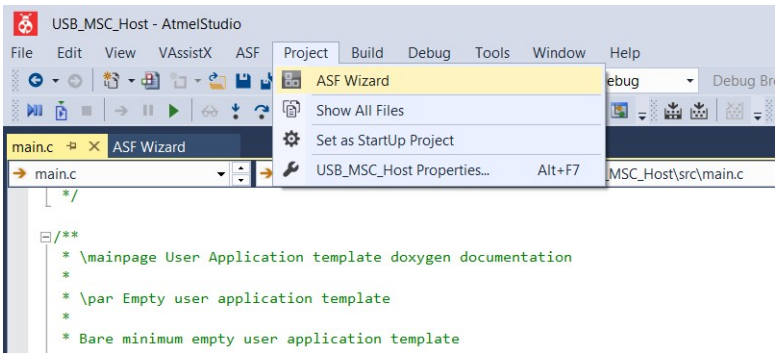
2. Select the device (ATSAMV71Q21) and board (SAMV71-XULTRA – ATSAMV71Q21) if the project is newly created.

Figure 6-2. Board-Device Selection



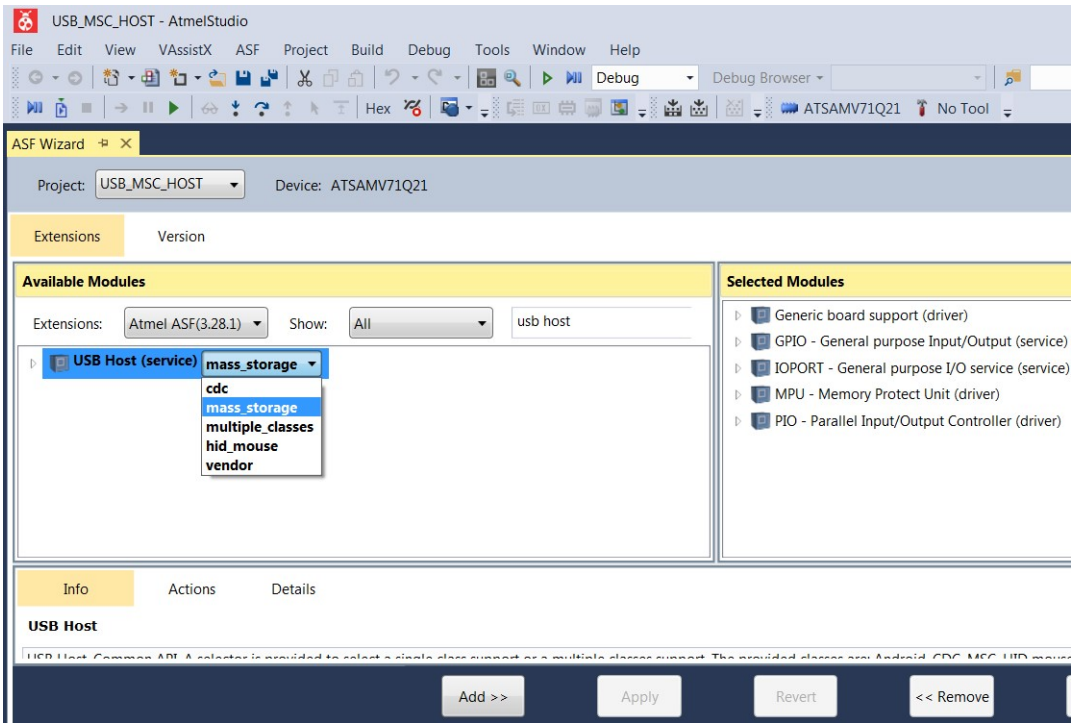
3. From Project Menu, select ASF Wizard.

Figure 6-3. ASF Wizard



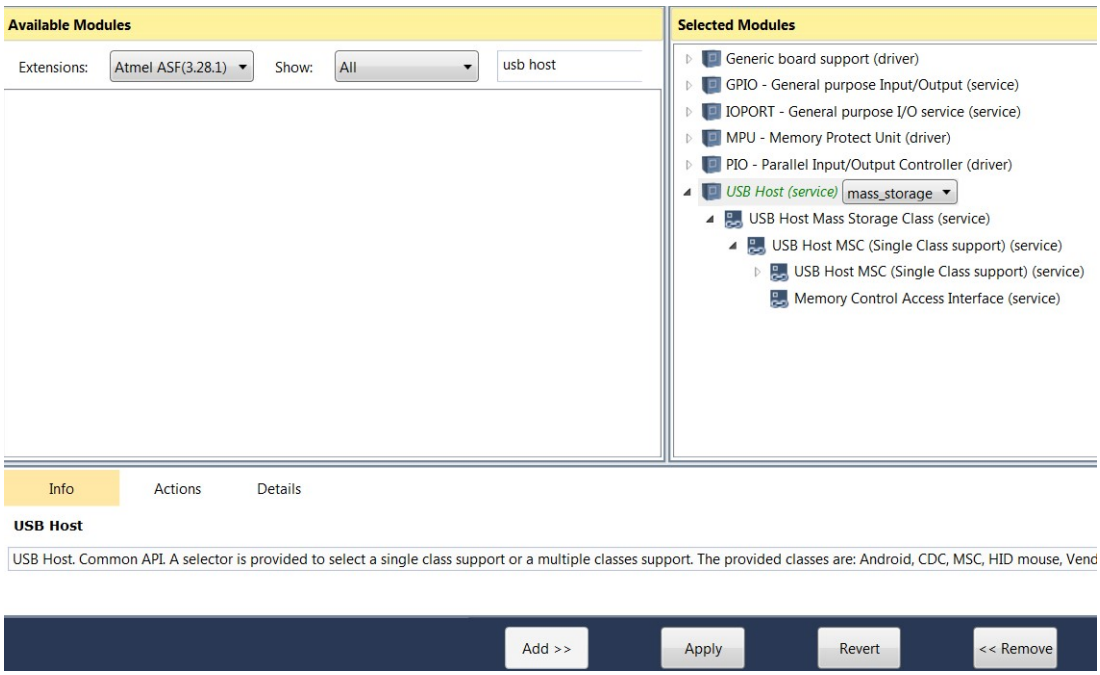
4. Search for USB Host in the search box. Select USB Host (Service) Mass Storage. Click **Add** and **Apply** to add the USB Host Mass Storage Service to the project.

Figure 6-4. Adding USB Host Mass Storage



5. This adds the USB Host MSC Stack to the application.

Figure 6-5. USB Host MSC (Service) - Configuration



6.2. USB Configuration

By adding USB Host (Service) for Mass Storage to the project, it will add the following configuration file to the project. The following table helps to understand the overview of changes needs to do in the project.

- `conf_access.h`

- conf_board.h
- conf_clock.h
- conf_sleepmgr.h
- conf_usb_host.h

Table 6-1. Configuration File

Configuration file	Contents	Changes
conf_usb_host.h	USB host configuration file	Authorize USB Host to run in High speed USB Host callbacks
conf_board.h	SAM V71 xplained Ultra board configuration/User board configuration	Configure USB pins Configure LUN (memory for data storage) Configure MEM, RAM interface
conf_clock.h	SAM V71 system clock configuration, USB clock configuration	No Change Defined by default
conf_sleepmgr.h	Chip specific sleep manager configuration • Initialize sleep manager service	No change
conf_access.h	Memory access control configuration file	No change
conf_fatfs.h	FatFS configuration file	Enable String function

The application's configurations are defined in conf_usb_host.h file in the application module. This file must be created for each USB Host application.

6.3. USB Host Configuration

The following table provides the possible USB Host Configuration

Table 6-2. USB HOST Configuration

Define Name	Type	Description	By default
USB_HOST_UHI	Array of UHI APIs	Define List of UHI supported by USB Host. #define USB_HOST_UHI UHI_MSC	Defined by default
USB_HOST_POWER_MAX	mA	Maximum current allowed on VBUS. #define USB_HOST_POWER_MAX 500	Defined by default
USB_HOST_HUB_SUPPORT(1)	Only defined	Authorize the USB HUB support.	Not defined by default
USB_HOST_HS_SUPPORT(1)	Only defined	Authorize the USB host to run in High Speed.	Not defined by default

Note: 1. Optional configuration. Comment the define statement to disable it (example: `// #define USB_HOST_X`).

In `conf_usb_host.h`, add the following line to configure SAM V71 for USB Host High speed.

```
#define USB_HOST_HS_SUPPORT
```

6.4. USB Host Drivers Configuration

In `conf_board.h`, add the following line to enable the USB lines.

```
/* Configure USB pins */
#define CONF_BOARD_USB_PORT
```

Configure memory (LUN) for data storage and its API by adding the following line in `conf_board.h`.

```
/*Configure LUN*/
#define USB_MASS_STORAGE_ENABLE

/*MEM <-> RAM interface*/
#define ACCESS_MEM_TO_RAM_ENABLED
```

To access the File system, `ACCESS_MEM_TO_RAM_ENABLED` has to be enabled.

6.5. USB Host Callback

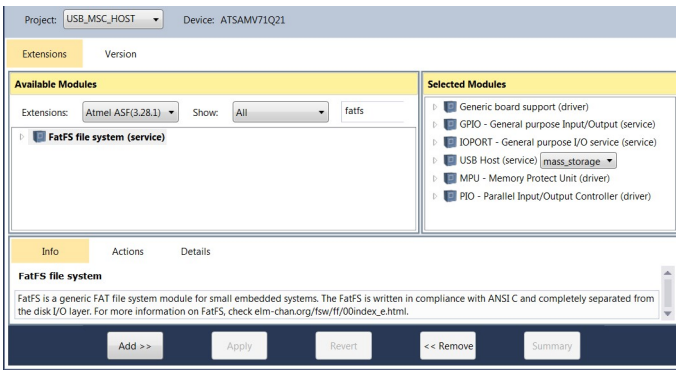
All UHC callbacks are optional and defined by the user in `conf_usb_host.h` file for each application. For this application, let us add a callback for the Start of Frame Event. SOF is sent for every 125µs (for USB High speed). This event is called for every SOF sent. This SOF event can be used as a Timer.

```
#define UHC_SOF_EVENT()         main_usb_sof_event()
```

6.6. Add FatFS Module

FAT file system is required to access a file in the connected mass storage memory on the USB stick/SD card. Add the FAT file system service through ASF wizard.

Figure 6-6. Adding FatFS Module



After clicking **Apply**, accept the FatFS File system license agreement to add FatFS drivers to the Project. More information about FatFS is available at elm-chan.org/fsw/ff/00index_e.html .

In conf_fatfs.h file, enable the string function. To enable the string function, set _USE_STRFUNC to 1 or 2. By enabling this, the f_puts function can be used.

```
  #define    _USE_STRFUNC    1    /* 0:Disable or 1-2:Enable */
/* To enable string functions, set _USE_STRFUNC to 1 or 2. */
```

6.7. Application Configuration

In case of a new project, the USB stack requires interrupts to be enabled, and clock and sleepmgr services to be initialized. Add the following line in the main.c file.

sysclk_init() function initializes system clock as described in the Clock Configuration section.

board_init() function initializes board configuration which is described in the Board Configuration section.

uhc_start() function starts the host mode.

Then it tries to create the file uhi_msc_.txt in the USB device. Add the following code snippet in main.c and program the device. This application creates the file in the connected USB Mass storage device.

```
#include <asf.h>
#include "conf_usb_host.h"
#include "string.h"

void main_usb_sof_event(void);

#define MAX_DRIVE       _VOLUMES
#define TEST_FILE_NAME "0:uhi_msc_test.txt"
#define MSG_TEST "Test UHI MSC\n"

typedef enum test_state {
    TEST_NULL,
    TEST_OK,
    TEST_NO_PRESENT,
    TEST_ERROR
} test_state_t;

static volatile uint16_t main_usb_sof_counter = 0;

static test_state_t lun_states[MAX_DRIVE];

static FATFS fs; // Re-use fs for LUNs to reduce memory footprint
static FIL file_object;

static char test_file_name[] = {
    TEST_FILE_NAME
};

/*! \brief Main function. Execution starts here.
 */
int main(void)
{
    sysclk_init();
    board_init();
    irq_initialize_vectors();
    cpu_irq_enable();

    // Initialize the sleep manager
    sleepmgr_init();

    // Start USB host stack
```

```
    uhc_start();

    // The USB management is entirely managed by interrupts.
    // As a consequence, the user application does only have :
    // - to play with the power modes
    // - to create a file on each new LUN connected
    while (true) {
        //sleepmgr_enter_sleep();
        if (main_usb_sof_counter > 2000) {
            main_usb_sof_counter = 0;
            volatile uint8_t lun;
            FRESULT res;
                // Mount drive
                memset(&fs, 0, sizeof(FATFS));
                res = f_mount(lun, &fs);
                if (FR_INVALID_DRIVE == res) {
                    // LUN is not present

                lun_states[lun] = TEST_NO_PRESENT;
                    continue;
                }
                // Create a test file on the disk
                test_file_name[0] = lun + '0';
                res = f_open(&file_object,
                        (char const *)test_file_name,
                        FA_CREATE_ALWAYS | FA_WRITE);
                if (res == FR_NOT_READY) {
                    // LUN not ready
                    lun_states[lun] = TEST_NO_PRESENT;
                    f_close(&file_object);
                    continue;
                }
                if (res != FR_OK) {
                    // LUN test error
                    lun_states[lun] = TEST_ERROR;
                    f_close(&file_object);
                    continue;
                }
                // Write to test file
                f_puts(MSG_TEST, &file_object);
                // LUN test OK
                lun_states[lun] = TEST_OK;
                f_close(&file_object);
            }

    }
}

void main_usb_sof_event(void)
{
    main_usb_sof_counter++;
}
```

7. References

SAM V71 datasheet: http://www.atmel.com/Images/Atmel-44003-32-bit-Cortex-M7-Microcontroller-SAM-V71Q-SAM-V71N-SAM-V71J_Datasheet.pdf

USB Host Stack: http://www.atmel.com/Images/doc8486.pdf

8. Revision History

Doc. Rev.	Date	Comments
42670A	02/2016	Initial document release

Atmel | Enabling Unlimited Possibilities®

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