APPLICATION NOTE

Atmel AT03088: Getting Started with SAM4E

Atmel 32-bit Microcontroller

Features

- Getting started with SAM4E device and tools
- Getting started with SAM4E-EK in Atmel Studio, IAR Embedded Workbench® for ARM® and SAM-BA®
- Getting started example in Atmel Software Framework (ASF)

Description

This applaiction note provides information on how to get start with the Atmel ARM Cortex®-M4 based SAM4E microcontroller. It will provide information on how to get datasheet, tools and software, and give a step-by-step instruction on how to load and buildup a single example project to SAM4E-EK.

Table of Contents

1. Get the Device Datasheet

Web page: www.atmel.com/sam4e

Documents: SAM4E Series Datasheet (Summary, Complete) (.pdf)

- Select the required device (ie. ATSAM4Ex) or and get the latest datasheet (.pdf file). There are two versions:
- Complete version (Full datasheet)
- Summary version (short version includes product features, package, pinout and order information)

2. Get the SAM4E Evaluation Kit

Web page: www.atmel.com/tools/SAM4E-EK.aspx

Get the kit: store.atmel.com

Document/file:

- SAM4E-EK production files: schematics (.pdf), gerber, BOM, test software
- SAM4E-EK User Guide (.pdf)

The SAM4E-EK is an evaluation kit featuring the SAM4E16E device BGA144 package with optional socket footprint, on board 12MHz and 32.769Hz crystal, a 2.8" TFT color LCD display with touch panel and backlight, One Ethernet physical transceiver layer with RJ45 connector, CAN port with driver, Mono/stereo headerphone jack output, QTouch® interfaces, Full speed USB device port, Serial Flash memory, NAND Flash memory, SD/MMC intergace, LEDs, push buttons, BNC connector for ADC input and DAC output, JTAG/ICE port, UART port with RS232 driver, USART port with RS232 driver multiplexed with RS485 function with driver.

The SAM4E-EK comes with a preloaded firmware which demonstrates the Ethernet capability of the product.

The SAM4E-EK User Guide introduces the SAM4E-EK and describes its development and debugging capabilities.

3. Get the Tools

The following tools are necessary for SAM4E development.

Atmel Studio 6.1: www.atmel.com/atmelstudio

IAR Embedded Workbench for ARM 6.50.3: www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/

SAM4E patch for IAR Embedded Workbench for eariler version of IAR: ewarm add-on v0.1.1.zip (provided with the application note)

Segger J-Link (v4.62 or above): www.segger.com/download_jlink.html

SAM-BA (v2.12): www.atmel.com/tools/atmelsam-bain-systemprogrammer.aspx

SAM4E patch for SAM-BA v2.12: sam-ba_2.12_patchsam4e.exe (provided with the application note)

ASF (v3.7.3 or above): www.atmel.com/tools/avrsoftwareframework.aspx

4. Get Started with Atmel Studio 6

4.1 Requirements

- Atmel Studio 6.1 (or above) installed
- Atmel Software Framework (ASF) v3.7.3 or above

- Segger J-Link installed
- SAM4E-EK board connected to Atmel Studio through SAM-ICE and powered on

4.2 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 SAM4E-EK in the latest ASF
- Pick a project in the list and then press OK
- Accept the license agreement and press Finish. Then the Atmel Studio will open the example
- Build the project: Build->Build Solution
- Load the code in SAM4E 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.

5. Get Started with IAR EWARM

5.1 Requirements

- ASF 3.7.3 or above standalone package installed
- IAR Embedded Workbench for ARM 6.50.3 installed
- SAM4E patch for IAR Embedded Workbench installed if using an earlier version of IAR
- Segger J-Link v.4.62 or above installed
- SAM4E-EK board connected to IAR Embedded Workbench for ARM through SAM-ICE and powered on

5.2 Load the Example

- Open the an example project file for SAM4E-EK
- Build the project: Project->Make
- Load the code in SAM4E 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.

6. Get Started with SAM-BA

6.1 Requirements

- Atmel Studio 6.1 (or above) installed
- ASF 3.7.3 or above standalone package installed
- SAM-BA v2.12 and SAM4E patch for it installed
- Segger J-Link v4.62 or above installed
- SAM4E-EK board connected to SAM-BA through SAM-ICE and powered on

6.2 Build the Binary File

- Open the Atmel Studio command line: Start->All Programs->Atmel->Atmel Studio 6.1 Command Prompt
- Change the directory where the a SAM4E-EK example makefile is
- 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, please refer to IAR C/C++ Development Guide for ARM provided by IAR Embedded Workbench for ARM.

6.3 Load the Example

- Open SAM-BA
- Select \jlink\ARM0 (could be ARM1, 2 or other number) as the connection
- Select at91sam4e16-ek as the target board. Then press Connect
- In SAM-BA GUI, choose Flash tab
- For Send File Name, choose the binary file (getting-started_flash.bin) generated previously
- Specify the address (0x400000), then press Send File
- For Scripts, select Boot from Flash (GPNVM1), then press Execute

Now the application has been programmed. To execute it, reset the board.

Besides J-Link, UART and USB can be used for the communication between SAM-BA and SAM4E, please refer to the chapter “SAM-BA Boot Program for SAM4E Microcontrollers” in SAM4E datasheet for the details.

7. The Getting-started Example

This section describes a simple example project that uses several important features present on SAM4E device, .

There are four main parts in this section:

- The specification of the getting-started example
- The introduction about relevant on-chip peripherals
- The introduction about relevant on-board components
- The implementation of the example

7.1 Specification

The getting-started example makes two LEDs on the board blink at a fixed rate. This rate is generated by using a timer for the first LED, and a Wait function based on a 1 ms tick for the second LED. The blinking can be stopped using two buttons - one for each LED.

7.2 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)

· Power Management Controller (PMC)

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 timebases, 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.

Using the NVIC is required to manage 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.

7.3 On-board Components

7.3.1 Buttons

The SAM4E-EK features five push-buttons, NRST, WAKU9, TAMP, SCROLL-UP and SCROLL-DOWN, connected to pins nRST, PA19, PA20, PA1 and PA2 respectively.

The NRST is used to reset the MCU usually, while the other four are used for general purpose, which can force a logical low level on the corresponding PIO line when pressed.

The Getting-started example uses WAKU9 and TAMPS buttons with the internal hardware debouncing circuitry embedded in the SAM4E.

7.3.2 LEDs

There are 4 LEDs on the SAM4E Evaluation Kit. D2, D3, D4 are used for general purpose, which are connected to PA0, PD20, and PD21 respectively. D5 is the power LED but can be controlled by software as well because it is connected to PD22.

Both D2 and D3 are used in the getting-started example.

7.3.3 COM Port (DBGU/UART)

UART0 of the SAM4E is connected to the COM port (DBGU/UART) on the SAM4E-EK.

7.4 Implementation

7.4.1 Startup

Most of the code in this program is written in C, which makes it easier to understand, more portable and modular. The C-startup code must:

· Provide vector table

· Initialize critical peripherals

· Initialize stacks

· Initialize memory segments

· Locate Vector Table Offset

These steps are described in the following paragraphs.

Note that there are two versions of c-startup code in Atmel Software Framework. One is for the IAR Embedded Workbench for ARM compiler and the other is for GNU GCC compiler. This application note will focus on the details of the GCC one.

7.4.1.1 Vector Table

The vector table contains the initialization value for the stack pointer (see “Initializing Stacks”) on reset, and the entry point addressed for all exception handlers. The exception numbers (see Table 7-1) define the order of entries in the vector table associated with the exception handler entries (see Table 7-2).

Table 7-1. Excetpion Numbers

Exception Number	Exception
1	Reset
2	Non-maskable Interrupt
3	Hard Fault
4	Memory Management
5	Bus Fault
6	Usage Fault
7-10	Reserved
11	SVCall
12	Debug Monitor
13	Reserved
14	PendSV
15	SysTick
16	External Interrupt 0
…	…
16 + N	External Interrupt N

Table 7-2. Vector Table Format

Word Offset	Description
0	Initial Stack Pointer
Exception Number	Exception using that Exception Number

On reset, the vector table is located at CODE partition. The table’s current location can be determined or relocated in the CODE or SRAM partitions of the memory map using the Vector Table Offset Register (VTOR). Details on the register can be found in the "Cortex-M4 TechnicalRef-erence Manual".

In the getting-started example, a full vector table looks like this:

The Full Vector Table in the getting-started example

```
const DeviceVectors exception_table = {

      /* Configure Initial Stack Pointer, using linker-generated symbols */
      (void*) (&_estack),
      (void*) Reset_Handler,
      (void*) NMI_Handler,
      (void*) HardFault_Handler,
      (void*) MemManage_Handler,
      (void*) BusFault_Handler,
      (void*) UsageFault_Handler,
      (void*) (0UL),          /* Reserved */
      (void*) (0UL),          /* Reserved */
      (void*) (0UL),          /* Reserved */
      (void*) (0UL),          /* Reserved */
      (void*) SVC_Handler,
      (void*) DebugMon_Handler,
      (void*) (0UL),          /* Reserved */
      (void*) PendSV_Handler,
      (void*) SysTick_Handler,

      /* Configurable interrupts */
      (void*) SUPC_Handler,   /* 0  Supply Controller */
      (void*) RSTC_Handler,   /* 1  Reset Controller */
      (void*) RTC_Handler,    /* 2  Real Time Clock */
      (void*) RTT_Handler,    /* 3  Real Time Timer */
      (void*) WDT_Handler,    /* 4  Watchdog/Dual Watchdog Timer */
      (void*) PMC_Handler,    /* 5  Power Management Controller */
      (void*) EFC_Handler,    /* 6  Enhanced Embedded Flash Controller */
      (void*) UART0_Handler,  /* 7  UART 0 */
      (void*) SMC_Handler,    /* 8  Static Memory Controller */
      (void*) PIOA_Handler,   /* 9  Parallel I/O Controller A */
      (void*) PIOB_Handler,   /* 10 Parallel I/O Controller B */
      (void*) PIOC_Handler,   /* 11 Parallel I/O Controller C */
      (void*) PIOD_Handler,   /* 12 Parallel I/O Controller D */
      (void*) PIOE_Handler,   /* 13 Parallel I/O Controller E */
      (void*) USART0_Handler, /* 14 USART 0 */
      (void*) USART1_Handler, /* 15 USART 1 */
      (void*) HSMCI_Handler,  /* 16 Multimedia Card Interface */
      (void*) TWI0_Handler,   /* 17 Two Wire Interface 0 */
      (void*) TWI1_Handler,   /* 18 Two Wire Interface 1 */
      (void*) SPI_Handler,    /* 19 Serial Peripheral Interface */
      (void*) DMAC_Handler,   /* 20 DMAC */
      (void*) TC0_Handler,    /* 21 Timer/Counter 0 */
      (void*) TC1_Handler,    /* 22 Timer/Counter 1 */
      (void*) TC2_Handler,    /* 23 Timer/Counter 2 */
      (void*) TC3_Handler,    /* 24 Timer/Counter 3 */
      (void*) TC4_Handler,    /* 25 Timer/Counter 4 */
      (void*) TC5_Handler,    /* 26 Timer/Counter 5 */
      (void*) TC6_Handler,    /* 27 Timer/Counter 6 */
      (void*) TC7_Handler,    /* 28 Timer/Counter 7 */
      (void*) TC8_Handler,    /* 29 Timer/Counter 8 */
      (void*) AFEC0_Handler,  /* 30 Analog Front End 0 */
      (void*) AFEC1_Handler,  /* 31 Analog Front End 1 */
      (void*) DACC_Handler,   /* 32 Digital To Analog Converter */
      (void*) ACC_Handler,    /* 33 Analog Comparator */
      (void*) ARM_Handler,    /* 34 FPU signals : FPIXC, FPOFC, FPUFC, FPIOC,
FPDZC, FPIDC, FPIXC */
      (void*) UDP_Handler,    /* 35 USB DEVICE */
      (void*) PWM_Handler,    /* 36 PWM */
      (void*) CAN0_Handler,   /* 37 CAN0 */
      (void*) CAN1_Handler,   /* 38 CAN1 */
      (void*) AES_Handler,    /* 39 AES */
      (void*) Dummy_Handler,
      (void*) Dummy_Handler,
      (void*) Dummy_Handler,
      (void*) Dummy_Handler,
      (void*) GMAC_Handler,    /* 44 EMAC */
      (void*) UART1_Handler    /* 45 UART */
};
```

7.4.1.2 Reset Exception

The handler of reset exception is responsible for starting up the application by performing the following actions:

Table 7-3. Reset Exception Actions

Action	Description
Initialize variables	Any global/static variables must be setup. This includes initializing the BSS variable to 0, and copying initial values from ROM to RAM for non-constant variables.
Set vector table	Optionally change vector table from Code area, value 0, to a location in SRAM. This is normally done to enable dynamic changes.
Enable FPU	Optionally enable FPU if __FPU_USED is defined
Branch to main()	Branch to the main() application.

7.4.2 System Clock Initialization

At the very beginning of the getting-started example main(), sysclk_init() is called to initialized the system clock of SAM4E. In this function, Power Management Controller (PMC) is set according to the clock configuration file, conf_clock.h.

In the conf_clock.h, the system clock source (CONFIG_SYSCLK_SOURCE) and system clock prescaler (CONFIG_SYSCLK_PRES) must be defined. In the case of the getting-started example, since the Phase Lock Loop block (PLLA) is used to multiply the frequency of the system clock, PLLA source, factor and divider are defined.

Clock Configuration

```
// ===== System Clock (MCK) Source Options
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_PLLACK

// ===== System Clock (MCK) Prescaler Options
#define CONFIG_SYSCLK_PRES          SYSCLK_PRES_2

// ===== PLL0 (A) Options
// Use mul and div effective values here.
#define CONFIG_PLL0_SOURCE          PLL_SRC_MAINCK_XTAL
#define CONFIG_PLL0_MUL             20
#define CONFIG_PLL0_DIV             1
```

As shown in the code above, the exteneral crystal oscillator (PLL_SRC_MAINCK_XTAL) is selected as the PLLA source (CONFIG_PLL0_SOURCE). The factor (CONFIG_PLL0_MUL) and divider (CONFIG_PLL0_DIV) are defined as 20 and 1 respectively. PLLA (SYSCLK_SRC_PLLACK) is chosen as the system clock source (CONFIG_SYSCLK_SOURCE), the prescaler of which (CONFIG_SYSCLK_PRES) is defined as 2.

So after calling sysclk_init() with this configuration, the system clock frequency (SYSCLK) is

SYSCLK = XTAL * MUL / DIV / PRES = 12MHz * 20 / 1 / 2 = 120MHz

Note that on the SAM4E-EK, 12MHz crystal oscillator is connected to XIN and XOUT pins.

7.4.3 Board Initialization

To contol the on-board components, buttons, LEDs and COM port in the case of the getting-started example, board_init() is called in the main(). With the conf_board.h, the corresponding pins are configured in the appropriate mode.

Board Configuration

```
/** Enable Com Port. */
#define CONF_BOARD_UART_CONSOLE
```

In board_init(), the pins connected to buttons are configured as input ports and the pins connected to LEDs are configured as output ports.

In the getting-started example, CONF_BOARD_UART_CONSOLE is predefined as above, which enables the COM port by configuring PA9 and PA10 as URXD0 and UTXD0 respectively.

7.4.4 Peripherals Configuration and Usage

7.4.4.1 UART

UART outputs the debug information via the COM port in the getting-started example. To display characters on PC terminal software correctly, several parameters must be configured before calling puts() and printf().

In SAM4E, the UART peripheral operates in asynchronous mode only and supports only 8-bit character handling (with parity) and 1 stop bit. No flow control is supported. So there are the baudrate and parity left to be configured.

UART Parameters

```
/** Baudrate setting */
#define CONF_UART_BAUDRATE   115200
/** Parity setting */
#define CONF_UART_PARITY     UART_MR_PAR_NO
```

In conf_uart_serial.h, the baudrate is set as 115200bps and no parity is used.

UART Configuration

```
const usart_serial_options_t uart_serial_options = {
      .baudrate = CONF_UART_BAUDRATE,
      .paritytype = CONF_UART_PARITY
};

/* Configure console UART. */
sysclk_enable_peripheral_clock(ID_UART0);
stdio_serial_init(UART0, &uart_serial_options);
```

In the above code, the peripheral clock for UART0 is enabled by calling sysclk_enable_peripheral_clock(). Then stdio_serial_init() configures the baudrate and the parity type.

7.4.4.2 SysTick

SysTick can be easily configured by calling SysTick_Config(). To generate 1ms period, the only parameter of this function should be system clock frequency / 1000.

SysTick Configuration

```
SysTick_Config(sysclk_get_cpu_hz() / 1000)
```

sysclk_get_cpu_hz() returns the current system clock frequency in Hz.

Then the SysTick interrupt will be triggered every 1ms. In the getting-started example, the SysTick interrupt handler SysTick_Handler() simply increases a global counter by 1 every time, which is used by the wait function to generate a specified period delay.

SysTick Interrupt Handler

```
volatile uint32_t g_ul_ms_ticks = 0;
void SysTick_Handler(void)
{
      g_ul_ms_ticks++;
}
```

Wait Function

```
static void mdelay(uint32_t ul_dly_ticks)
{
      uint32_t ul_cur_ticks;

      ul_cur_ticks = g_ul_ms_ticks;
      while ((g_ul_ms_ticks - ul_cur_ticks) < ul_dly_ticks);
}
```

Note that the global counter, g_ul_ms_ticks, is declared as a volatile variable. It prevents the compiler from optimizing the code casuing that the wait function does not work.

7.4.4.3 TC

SAM4E provides nine 32-bit TC channels, which could be used to measure frequency, count event, generate PWM wave and so on.

In the getting-started example, the TC channel 0 is configured to generate an interrupt per a quarter of a second.

Timer Counter Configuration

```
uint32_t ul_div;
uint32_t ul_tcclks;
uint32_t ul_sysclk = sysclk_get_cpu_hz();

/* Configure PMC */
pmc_enable_periph_clk(ID_TC0);

/** Configure TC for a 4Hz frequency and trigger on RC compare. */
tc_find_mck_divisor(4, ul_sysclk, &ul_div, &ul_tcclks, ul_sysclk);
tc_init(TC0, 0, ul_tcclks | TC_CMR_CPCTRG);
tc_write_rc(TC0, 0, (ul_sysclk / ul_div) / 4);

/* Configure and enable interrupt on RC compare */
NVIC_EnableIRQ((IRQn_Type) ID_TC0);
tc_enable_interrupt(TC0, 0, TC_IER_CPCS);

/** Start the counter if LED1 is enabled. */
if (g_b_led1_active) {
      tc_start(TC0, 0);
}
```

Before any configuration, TC peripheral clock is enabled. 2 necessary parameters, the TC divider, the tick value for the compare register (RC is used in the example), must be calculated to initialize the TC and the compare register. Then the program enables the TC channel 0 interrupt and the compare interrupt. In the end, it starts TC channel 0 and the counter starts ticking.

In the TC channel 0 interrupt handler, the LED status is toggled every time.

Interrupt Handler for TC Channel 0

```
volatile uint32_t ul_dummy;

/* Clear status bit to acknowledge interrupt */
ul_dummy = tc_get_status(TC0, 0);

/** Toggle LED state. */
ioport_toggle_pin_level(LED1_GPIO);
```

7.4.4.4 PIO

Besides toggling LEDs, in the getting-started example, PIO retrieves the button input. When a button is pressed, the level of the corresponding pin is changed. PIO detects the change and triggers an interrupt.

PIO Configuration for one button (one pin)

```
/* Configure Pushbutton 1 */
pmc_enable_periph_clk(PIN_PUSHBUTTON_1_ID);
pio_set_debounce_filter(PIN_PUSHBUTTON_1_PIO, PIN_PUSHBUTTON_1_MASK, 10);
/* Interrupt on rising edge  */
pio_handler_set(PIN_PUSHBUTTON_1_PIO, PIN_PUSHBUTTON_1_ID,
             PIN_PUSHBUTTON_1_MASK, PIN_PUSHBUTTON_1_ATTR, Button1_Handler);
NVIC_EnableIRQ((IRQn_Type) PIN_PUSHBUTTON_1_ID);
pio_handler_set_priority(PIN_PUSHBUTTON_1_PIO,
             (IRQn_Type) PIN_PUSHBUTTON_1_ID, IRQ_PRIOR_PIO);
pio_enable_interrupt(PIN_PUSHBUTTON_1_PIO, PIN_PUSHBUTTON_1_MASK);
```

The PIO peripheral clock is enabled at first so that the configuration below can take effect.

Usually in an application with the button inputs, there are some glitches on the input lines of the buttons. In PIO of SAM4E, the debouncing filter can be set to reject these unwanted pulses. In the getting-started example, if the period of a glitch is less than 10 slow clock cycles (slow clock frequency is 32768Hz in this case), the glitch will be ignored by PIO.

Pressing different button leads to different action, so there should be a specified handler for a specified button pressing. Before enabling the PIO interrupt and any pin interrupt, a handler, Button1_Handler, is set by calling pio_handler_set(). Also the condition to trigger a pin interrupt is chosen here.

In the getting-started example, two buttons control two LEDs in two ways. When WAKU button is pressed, the pin connected to LED D2 toggles its output. When TAMP button is pressed, the TC channel is stopped or restarted. In both ways, two LEDs stop or start blinking.

Button Pressing Process

```
static void ProcessButtonEvt(uint8_t uc_button)
{
      if (uc_button == 0) {
            g_b_led0_active = !g_b_led0_active;
            if (!g_b_led0_active) {
                  ioport_set_pin_level(LED0_GPIO, IOPORT_PIN_LEVEL_HIGH);
            }
        } else {
            g_b_led1_active = !g_b_led1_active;

            /* Enable LED#2 and TC if they were enabled */
            if (g_b_led1_active) {
                ioport_set_pin_level(LED1_GPIO, IOPORT_PIN_LEVEL_LOW);
                tc_start(TC0, 0);
            }
            /* Disable LED#2 and TC if they were disabled */
            else {
                ioport_set_pin_level(LED1_GPIO, IOPORT_PIN_LEVEL_HIGH);
                tc_stop(TC0, 0);
            }
        }
}

static void Button1_Handler(uint32_t id, uint32_t mask)
{
        if (PIN_PUSHBUTTON_1_ID == id && PIN_PUSHBUTTON_1_MASK == mask) {
            ProcessButtonEvt(0);
        }
}

static void Button2_Handler(uint32_t id, uint32_t mask)
{
        if (PIN_PUSHBUTTON_2_ID == id && PIN_PUSHBUTTON_2_MASK == mask) {
            ProcessButtonEvt(1);
        }
}
```

8. Revision History

Doc. Rev.	Date	Comments
42145A	06/2013	Initial release