AN2775

CPU Usage Demonstration Using DMAC for SAM L10 MCUs

Introduction

The Direct Memory Access Controller (DMAC) in Microchip® | SMART™ SAM L10/SAM L11 enables the transfer between memories and peripherals, and enables the off-loading of these tasks from the CPU. The DMAC enables high data transfer rates with minimum CPU intervention and frees up CPU time. It contains both a Direct Memory Access engine and a Cyclic Redundancy Check (CRC) engine. With access to all peripherals, the DMAC can handle automatic transfer of data between communication modules.

This document demonstrates the CPU usage when an application is executed with or without the DMA. The analog data from the light sensor is sampled with the ADC, and the data is sent to the USART. In this document, the CPU usage is calculated with, and without the DMA for the data transfer.

Features

This application covers the following peripheral features:

- DMA data transfer between:
  - Peripheral (ADC) to peripheral (USART)
  - Peripheral (ADC) to memory (SRAM)
  - Memory (SRAM) to memory (SRAM)
  - Memory (SRAM) to peripheral (USART)
- Transfer trigger sources
  - Software
  - Peripherals (ADC Result Ready, USART Data Register Empty)
- Multi buffer transfer modes by linking multiple descriptors
- Enabling three independent channels with automatic descriptor for each channel
- Fixed priority scheme within each priority level
- 1 to 256 KB data transfer in a single block transfer
- Multiple addressing modes
  - Static
  - Programmable increment scheme
- Transaction complete interrupt generation
- DMA event output
- Event System for direct peripheral-to-peripheral communication signaling
- Event triggered ADC conversion for accurate timing
- DMA transfer of conversion result
- CPU usage calculation using System Timer (SYSTICK)

Prerequisites

The solutions discussed in this document require a basic familiarity with the following tools:

- SAM L10/SAM L11 Xplained Pro Evaluation Kit with USB cable

This document covers the overview of the following peripherals:

- DMAC
- SERCOM – USART
- EVSYS
- ADC
- SYSTICK

Refer to the product data sheet for a detailed information on each of the peripherals.

Table of Contents

Introduction....1

1. Setup....5
   1.1. Hardware Setup....5
2. Direct Memory Access Controller (DMAC)....8
   2.1. Block Diagram....8
   2.2. Functional Description....8
3. Peripherals Overview....11
   3.1. Event System....11
   3.2. Analog-to-Digital Converter....11
   3.3. SERCOM - USART....11
   3.4. The System Timer (SYSTICK)....12
4. Example Implementation....13
   4.1. Peripheral-to-Peripheral Transfer with the DMAC (ADC to USART)....13
   4.2. Peripheral-to Memory and Memory-to-Peripheral Transfer with the DMAC (ADC to SRAM and SRAM to USART)....17
   4.3. Peripheral-to-Memory and Memory-to-Peripheral Transfer Without DMAC (ADC to SRAM and SRAM to USART)....20
   4.4. Logic Implementation Used to Calculate the CPU Utilization....21
5. Application Limitations....25
   5.1. USART Baud-Rate and ADC Sampling Frequency....25
   5.2. SRAM to SRAM Transfer Type....25
6. CPU Utilization Analysis Between Different Cases....26
   6.1. CPU Frequency Calculation....26
   6.2. CPU Idle Time Calculation from Result Observed....26
7. Execution of Application....29
8. References....30

The Microchip Web Site....31

Customer Change Notification Service....31

Customer Support....31

Microchip Devices Code Protection Feature....31

Legal Notice....32

Trademarks....32

Quality Management System Certified by DNV.....33

Worldwide Sales and Service.....34

1. Setup

This application was developed for SAM L10 Xplained Pro board. This chapter covers hardware and software setup required for testing this application.

1.1 Hardware Setup

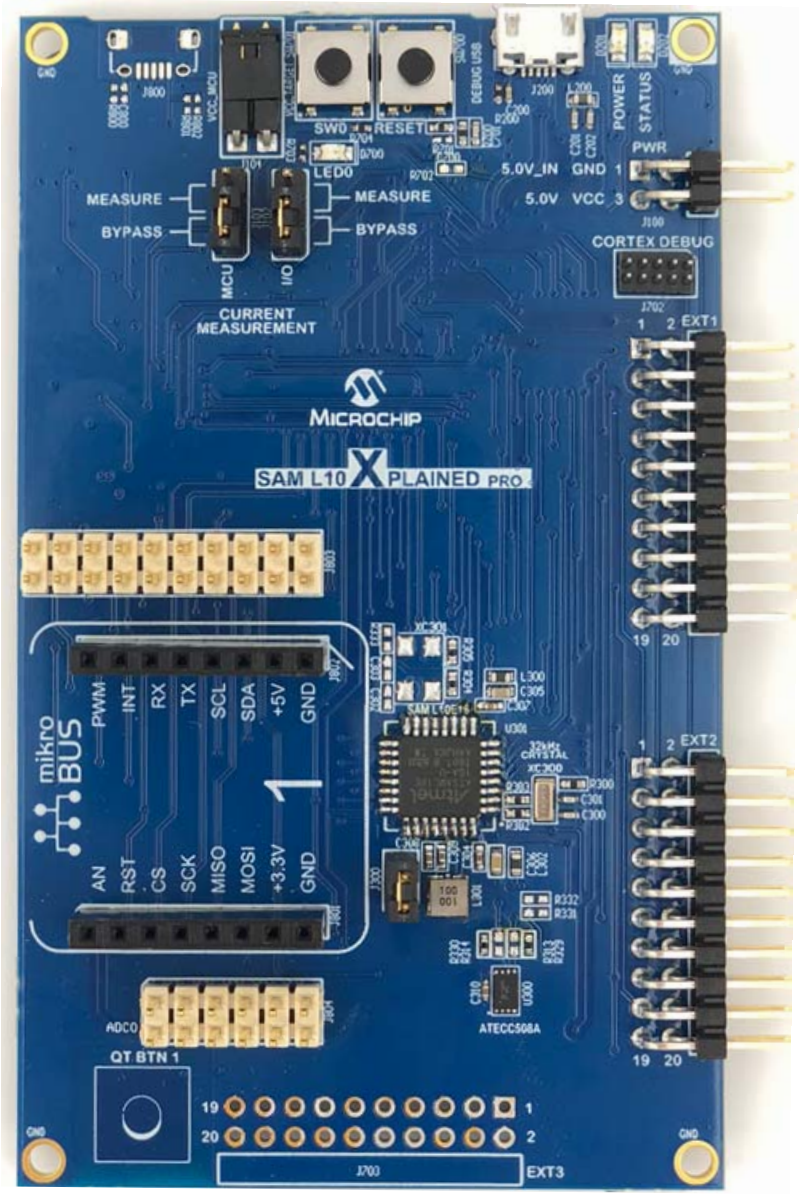
1.1.1 SAM L10 Xplained Pro

The Microchip SAM L10 Xplained Pro evaluation kit is a hardware platform to evaluate the ATSAML10E16A-AU microcontroller.

The evaluation kit offers a set of features that enables the ATSAML10E16A user to get started with the micro-controller peripherals instantly, and to understand the steps to integrate the device in a custom design.

The evaluation kit supports a wide range of Xplained Pro extension boards through Xplained Pro extension headers. The SAM L10 Xplained Pro has three such Xplained Pro extension headers, and EXT1 is used in this application.

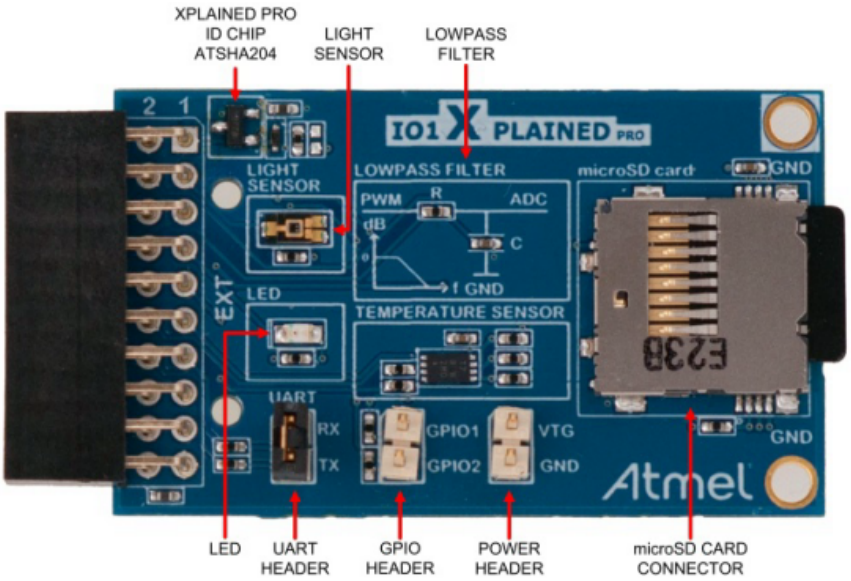
Figure 1-1. SAM L10 Xplained Pro Board



1.1.2 IO1 Xplained Pro Extension Board

The Microchip IO1 Xplained Pro extension board is a generic extension board for the Xplained Pro platform. It connects to any Xplained Pro standard extension header on any Xplained Pro MCU board. The extension board utilizes all functions available on the standard Xplained Pro extension header.

Figure 1-2. IO1 Xplained Pro Extension Board



The Microchip IO1 Xplained Pro has is designed to be connected to the Xplained Pro header marked EXT1 or EXT2. However, it is compatible with all Xplained Pro EXT headers available on an Xplained Pro board. The pin-out of the respective Xplained Pro evaluation kit is needed to find out which Xplained Pro EXT headers can be used. This document describes the demonstration using the EXT1 header.

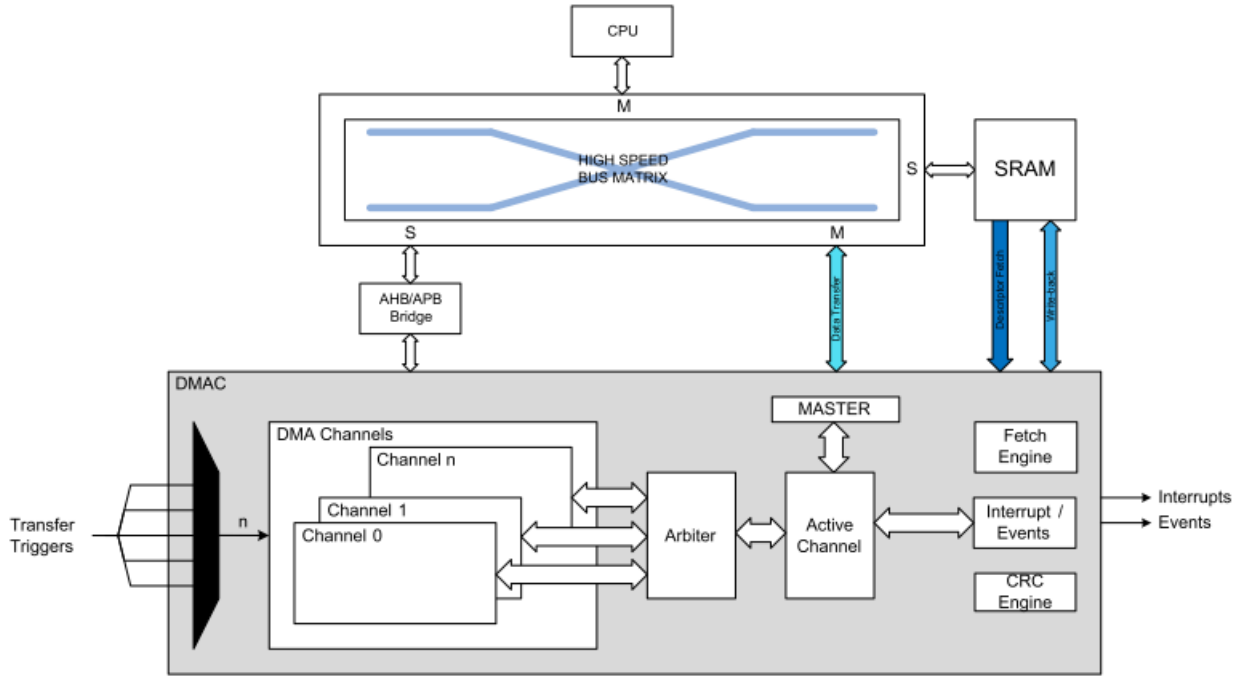
The IO1 Xplained Pro features a light sensor. Pin3 of the extension board is utilized for this purpose. The sensor data can be gathered from an ADC pin on the Xplained Pro MCU board. In the SAM L10 Xplained Pro kit, it is connected to the EXT1 header. This application utilizes a light sensor on the IO1 Xplained pro board as an analog input to the ADC.

2. Direct Memory Access Controller (DMAC)

This chapter covers the DMAC features and its operation relevant to this document. Refer to the product data sheet for a detailed description about its operation and configuration.

2.1 Block Diagram

Figure 2-1. DMAC Block Diagram



2.2 Functional Description

2.2.1 DMAC Basic Operation

The Direct Memory Access Controller (DMAC) contains both a Direct Memory Access engine and a Cyclic Redundancy Check (CRC) engine. The DMAC can transfer data between memories and peripherals, and allows for off-loading these tasks from the CPU. The DMAC enables high data transfer rates with minimum CPU intervention, and frees up CPU time. With access to all peripherals, the DMAC can handle the automatic transfer of data between communication modules. This allows the CPU to sleep for a longer time and reduce the power consumption.

A complete DMA read and write operation between memories and peripherals is called a DMA transaction. DMA reads data from the source address before writing to the destination address. New data is read when the previous write operation is completed. The transaction is initiated by a trigger and uses a DMA channel. The DMA trigger source can be application software, a peripheral, or events from Event System (EVSYS). Each read and write operation is done in blocks. The size of the transfer is controlled by the block transfer size and is configured in the software. The size of the block can be from 1 to 64K beats. The beat can be a byte, half-word or word.

2.2.2 DMAC Channels

The DMA implements eight channels, which enables eight independent transfers. Each DMA channel has an individual Transfer control descriptor setting that is stored in SRAM.

The transfer control descriptor defines the source and destination address, source and destination address increment settings, block transfer count, and optional event output condition selection.

Source and destination addressing can be static or incremental.

Dedicated I/O registers for each channel are available which control the trigger mode (peripheral/ software), peripheral trigger source type, event input actions, and channel priority level settings.

Dedicated write-back memory section is available for each active channel, to maintain the current transfer settings and status.

When enabling multiple channels, 4-level channel priority is supported, and fixed or round-robin scheme is available within each priority level.

2.2.3 DMAC Transfer Operation

A single transaction can be executed (using only one descriptor), or multiple transactions can be executed (using linked descriptor). Single or multiple block transfers can be enabled using the same DMA channel.

When the DMA peripheral and respective channel are enabled, the transfer will happen upon receiving the trigger request. The transfer type can be beat, block (group of beats together forms block), or transaction (group of blocks forms transaction).

The channel is automatically disabled when the DMA transfer is completed. If a single descriptor is defined for a channel, the channel will be disabled when a block transfer is completed. In the case of linked descriptors, the channel is disabled once the last descriptor is executed.

2.2.4 Other Features

Channel Suspend and Resume

The channel operation can be suspended or resumed at any time by software, or can be suspended when a selectable block transfer is complete.

Interrupt Request

Interrupt requests can be generated in the following scenario:

- A transaction is complete
- Selectable block transfer is complete
- DMA controller detects a bus error
- A channel operation is suspended

Event Input

The event input actions are available on the least significant DMA channels. The event can be programmed to trigger:

- Transfers
- Periodic transfers
- Conditional transfers
- To suspend or resume a channel operation

Event Output

Event output selection is available for the least significant DMA channels. Events can be generated during the following scenario:

- Each AHB data transfer is complete
- Selectable block transfer is complete
- Entire transaction is complete

3. Peripherals Overview

This section covers the overview of other peripherals relevant to this document. Refer to respective sections in the product data sheet for a detailed description about operation and configuration.

3.1 Event System

The Event System (EVSYS) allows autonomous, low-latency, and configurable communication between peripherals. Several peripherals can be configured to emit and respond to signals known as events.

The exact condition to generate an event, or the action taken upon receiving an event, is specific to each module. Peripherals that respond to events are called event users. Peripherals that emit events are called event generators. A peripheral can have many event generators, and can have many event users.

Communication is made without CPU intervention, and without consuming system resources such as bus or RAM bandwidth. This reduces the load on the CPU and other system resources, compared to a traditional interrupt-based system.

In this document, the EVSYS is configured to use ‘DMA channel 0 transfer complete’ (DMAC CH0) as the event generator, and ADC start conversion (ADC START) as event user. At the completion of the DMA transfer, the DMA will trigger an output event, and the event system upon receiving this event, triggers the ADC start conversion.

3.2 Analog-to-Digital Converter

The Analog-to-Digital Converter (ADC) converts analog signals to digital values. The ADC has up to 12-bit resolution, and is capable of converting up to 1MSPS. The input selection is flexible, and both differential and single-ended measurements can be performed. In addition, several internal signal inputs are available.

ADC measurements can be started by either application software or an incoming event from another peripheral in the device. Both internal and external reference voltages can be used.

The ADC may be configured for 8-bit, 10-bit, or 12-bit results, reducing the conversion time. ADC conversion results are provided as left-adjusted or right-adjusted, which eases calculation when the result is represented as a signed value. It is possible to use the DMA to move the ADC results directly to memory or peripherals when the conversions are done.

In this example, the ADC is configured for 8-bit resolution and uses the DMA to transfer the ADC result to the configured destination address (can be peripheral or memory). Event input from the DMA is used to trigger the next ADC conversion. A software trigger is used when the process is implemented without using the DMA.

3.3 SERCOM - USART

The SERCOM serial engine consists of a transmitter and receiver, baud-rate generator, and address matching functionality. The transmitter consists of a single write buffer and a shift register. The receiver consists of a two-level receive buffer and a shift register. The baud-rate generator is capable of running on the GCLK_SERCOMx_CORE clock or an external clock.

The Serial Communication Interface (SERCOM) can be configured to support a number of modes: I^2C, SPI, and USART. Once configured and enabled, all SERCOM resources are dedicated to the selected mode.

The Universal Synchronous and Asynchronous Receiver and Transmitter (USART) is one of the available modes in the Serial Communication Interface (SERCOM).

A data transmission is initiated by loading the DATA register with the data to be sent. The data in TxDATA is moved to the shift register when the shift register is empty and ready to send a new frame. When the shift register is loaded with data, one complete frame will be transmitted.

When the entire frame plus stop bits have been shifted out and there is no new data written to the DATA register,the Transmit Complete interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.TXC) is set, and the optional interrupt is generated.

The DATA register should only be written when the Data Register Empty flag in the Interrupt Flag Status and Clear register (INTFLAG.DRE) are set, which indicates that the register is empty and ready for new data.

The USART can generate a DMA request when the transmit buffer (TX DATA) is empty. The request is cleared when DATA is written.

In this application, EDGB CDC (SERCOM0) is utilized to transfer the ADC result data to the terminal.

3.4 The System Timer (SYSTICK)

The System Timer is a 24-bit timer that extends the functionality of both the processor and the NVIC. Refer to the ARM Technical Reference Manual for additional information, which is available for download from the following location www.arm.com.

The timer consists of the following registers:

- A control and status register (SYST_CSR). This configures the SYSTICK clock, enables the counter, enables the SYSTICK interrupt, and indicates the counter status.
- A counter reload value register (SYST_RVR). This provides the wrap value for the counter.
- A counter current value register (SYST_CVR).

When enabled, the timer counts down from the value in the SYST_CVR register. When the counter reaches zero, it reloads the value in the SYST_RVR register on the next clock edge. It then decrements on subsequent clocks. This reloading when the counter reaches zero is called wrapping. the interrupt can be enabled, which is triggered for each time counter wrap around.

In this application, the counter is loaded with the maximum count value, and is used to take a time stamp while calculating the CPU utilization. The SYSTICK runs at the processor clock as source.

4. Example Implementation

This section explains the application implementation in detail.

The objective of this document is to demonstrate the features and their configuration. In addition to that, CPU utilization is calculated, when implemented with or without the DMA. This highlights the DMAC usage in reducing the CPU load.

In the example implementation, the ADC converts the input analog signal to a digital value and the result is transferred to the USART. The light sensor in the IO1 Xplained Pro is used as an input to the ADC through the EXT1 header.

This document describes three different scenarios to cover the objective (i.e. with and without DMAC). Each scenario is implemented through a separate Atmel Start example project, and users will need to select the appropriate example project. The following sections explain each example project in detail.

4.1 Peripheral-to-Peripheral Transfer with the DMAC (ADC to USART)

The Atmel Start example project for this transfer type is *ADC DMAC USART*. In this case, the ADC result is written to the USART DATA register to illustrate the peripheral-to-peripheral DMA transfer type.

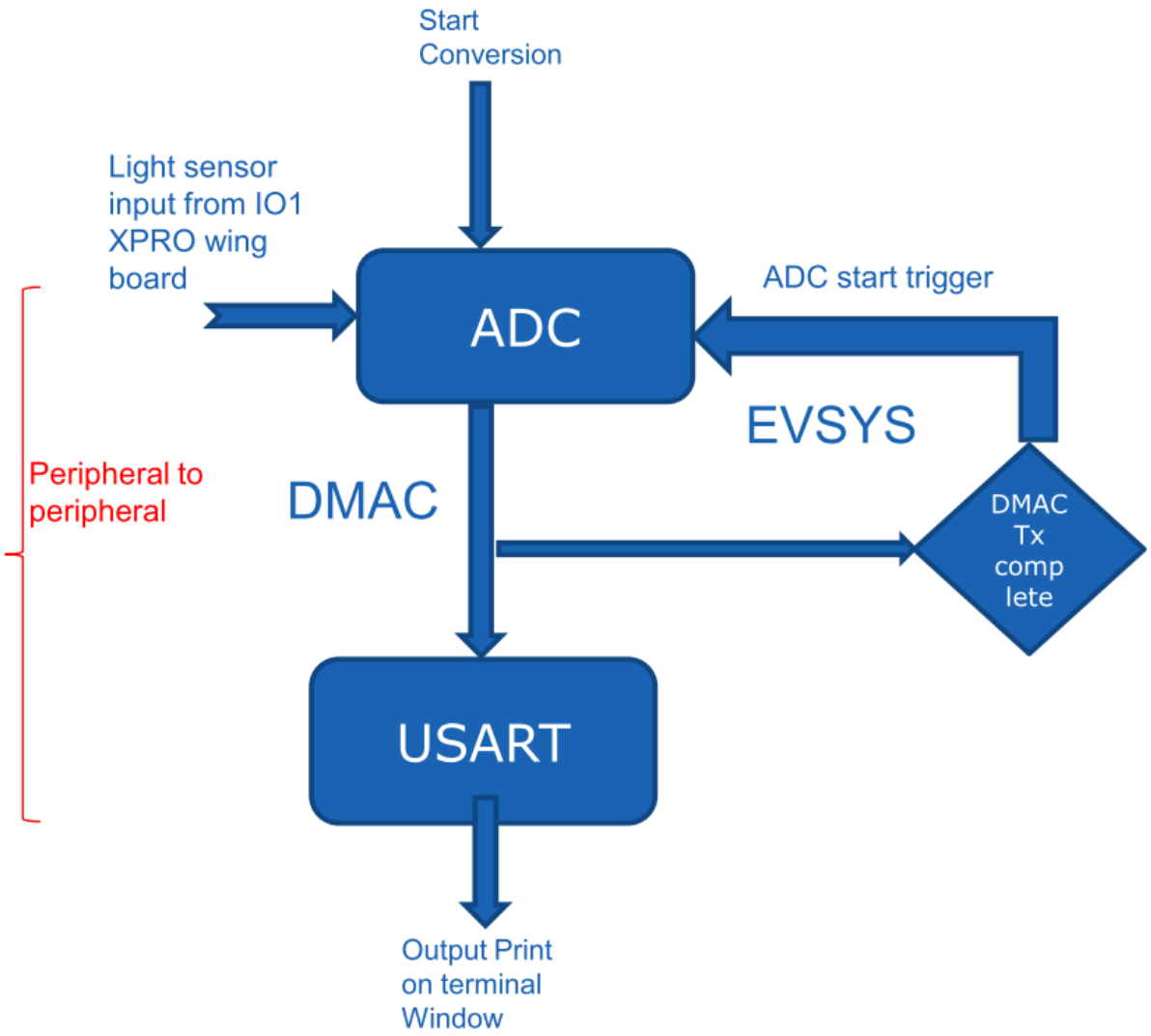
4.1.1 Application Configuration and Implementation

The DMAC is configured to trigger a data transfer to the configured destination address when the ADC RESULT is ready (peripheral trigger source). The configured destination address is the USART DATA register address, and the source address is the ADC RESULT register address. The DMA source and destination address is static in this case, as both the register addresses are fixed.

The ADC is configured as the event user, which will start conversion upon receiving an event signal from the DMAC through the Event System (EVSYS). The EVSYS is configured with the DMAC as the event generator, and the ADC as the event user.

The first ADC conversion is triggered by a software trigger. Once the input is sampled and the result is ready, it triggers a DMA transfer from the ADC RESULT to the USART DATA register. The next ADC conversion is triggered by the event signal from the DMA upon completing the transfer to the USART DATA register and the cycle continues. Overall the operation is done as shown in the following figure.

Figure 4-1. Peripheral to Peripheral Transfer with DMAC



The whole operation is performed using the DMAC and EVSYS without interrupting the CPU. The DMAC block transfer size is configured as 1024 bytes (BLOCK_COUNT), and an interrupt is configured to flag once the block transfer is complete. Once the block transfer is completed, the DMAC channel gets disabled automatically.

Oscillator Controller configuration:

The 16 MHz Internal Oscillator is selected and the frequency is set to 16 MHz.

- Configure the 16 MHz internal oscillator frequency to 16 MHz by setting the OSC16MCTRL.FSEL[1:0] to 0x11
- Enable the 16 MHz internal oscillator by setting the OSC16MCTRL.ENABLE to 1

GCLK configuration:

The GCLK0 is configured to use the 16 MHz Internal Oscillator (OSC16M) as the clock source.

- Configure the GCLK0 clock source to 16 MHz oscillator output by setting GENCTRL0.SRC[4:0] to 0x05 (OSC16M oscillator output)
- Enable GCLK0 by setting GENCTRL0.GENEN to 1
- The GCLK0 output is enabled to measure the CPU clock frequency. I/O pin PA27 is configured to output GCLK0
- Assign GCLK_IO[0] peripheral function to PA27 by setting PMUX13.PMUXO[3:0] to 0x07 (Peripheral Function H Selected)
- Set GENCTRL0.OE to 1 to output the Generator clock on GCLK_IO

MCLK configuration:

The GCLK0 is always the direct source for GCLK_MAIN, which is used by the MAIN clock module.

ADC Configuration:

ADC is configured to use GCLK0 (16MHz) as the clock source. The ADC conversion resolution is set to 8-bit and ADC reference to ½ VDDANA. The ADC clock is divided by 64 (ADC clock = 16MHz/64 = 250KHz). PA02 is selected as the positive input to the ADC and GND as the negative input.

- Select GCLK0 as the clock source to the ADC module by configuring PCHCTRL20.GEN[2:0] to 0x0 (Generic Clock Generator 0)
- Divide the input clock to the ADC peripheral by 64 by setting CTRLB.PRESCALER[2:0] to 0x05
- Set the ADC resolution to 8-bit by setting CTRLC.RESSEL[1:0] to 0x3
- Set the ADC reference voltage to ½ VDDANA by setting REFCTRL.REFSEL[3:0] to 0x02
- Select AIN0 as the positive input of the ADC by setting INPUTCTRL.MUXPOS[4:0] to 0x00 (ADC AIN0 pin)
- Select GND as the negative input of the ADC by setting INPUTCTRL.MUXNEG[4:0] to 0x18 (Internal ground)
- Assign AIN[0] peripheral function to PA02 by setting PMUX1.PMUXE[3:0] to 0x01 (Peripheral Function B Selected)
- ADC is configured to trigger a conversion on event input. The DMA transfer complete event will act as the event input.
- Trigger ADC conversion on event input by setting EVCTRL.STARTEI = 1

DMAC Configuration:

DMA channel 0 is configured to perform a beat (8-bit) transfer on ADC Result Ready trigger. An event is generated on each beat transfer. Since the source and destination addresses are static they are not incremented. The DMA channel will be disabled once the last block transfer in the transaction is completed.

- Select the DMA channel to configure (DMA channel 0 in this case) by setting CHID.ID[3:0] to 0x0

- Select the DMA trigger source as the ADC Result Ready by setting CHCTRLB.TRIGSRC[4:0] to 0x13 (ADC Result Ready Trigger)
- To perform a beat (8-bit) transfer on every trigger, set the trigger action in CHCTRLB.TRIGACT[1:0] to 0x2 (One trigger required for each beat transfer)
- Enable channel event output by setting CHCTRLB.EVOE = 1. This event output will be used by the ADC (through the event system) to start an ADC conversion.
- The DMA channel transfer descriptor registers are configured as configured as follows:
  - Source and destination address are static. BTCTRL.SRCINC =0 and BTCTRL.DSTINC = 0.
  - The beat size is configured to 1 byte. BTCTRL.BEATSIZE[1:0] = 0x0 (8-bit bus transfer).
  - To generate an event output when a beat (1-byte) transfer is complete, set the BTCTRL.EVOSEL[1:0] to 0x3 (Event strobe when beat transfer complete)
  - Configure the DMA channel 0 source address (SRCADDR[31:0]) to ADC result register
  - Configure the DMA channel 0 destination address (DSTADDR[31:0]) to the USART transmit register
  - Set the block transfer count to 1024 by setting BTCNT[15:0] to 1024
  - The next descriptor address for channel 0 must be set to NULL by setting DESCADDR to 0
- Enable the DMA channel 0 transfer complete interrupt by setting CHINTENSET.TCMPL to 1

Note: All transfer descriptors must reside in SRAM. The addresses stored in the Descriptor Memory Section Base Address (BASEADDR) and Write-Back Memory Section Base Address (WRBADDR) registers tell the DMAC where to find the descriptor memory section and the write-back memory section. As BASEADDR points only to the first transfer descriptor of channel 0, all first transfer descriptors must be stored in a contiguous memory section, where the transfer descriptors must be ordered according to their channel number. For more information, refer the SAM L10/L11 data sheet.

USART Configuration:

The SERCOM0 is configured for USART mode operation with the GCLK0 (16MHz) as the clock source to the SERCOM0 module. The USART RX and TX lines are mapped to PA25 (SERCOM0 PAD[3]) and PA24 (SERCOM0 PAD[2]) respectively.

The baud rate is set to 460800. Refer to USART Baud-Rate and ADC Sampling Frequency for details on why this baud is chosen.

Event System Configuration:

The Channel 0 of the Event System is configured with DMA channel 0 as the event generator and ADC Start Conversion as the event user.

- Select DMA channel 0 as the event generator for channel 0 of the Event System by setting CHANNEL0.EVGEN[5:0] to 0x26 (ADC Result Ready)
- Select asynchronous path for channel 0 of the event system by setting CHANNEL0.PATH[1:0] = 0x2 (Asynchronous path)
- Select ADC Start Conversion as the user of channel 0 of the event system, by setting USER14.CHANNEL[3:0] to 0x1 (A value of x in this field selects channel n = x-1)

I/O Pin Configuration:

I/O pin PA04 is configured as Digital Output. PA04 is used to measure the time taken by the CPU to increment the idle loop counter. This will then be used to calculate the CPU idle time.

I/O pin PA06 is configured as Digital Output. PA06 will be toggled in the interrupt service handlers, and is also used in calculating the CPU idle time.

I/O pin PA27 is configured to output the GCLK0 frequency (as described previously, under the GCLK configuration) and is used to measure the CPU clock frequency. The CPU clock frequency will be used to calculate the total execution time.

4.1.2 CPU Utilization Calculation

Once the MCU and the peripheral initialization is complete, the application takes a time stamp by reading the SYSTICK Current Value register and saves it to a variable (i.e., time_stamp1). It then starts the first ADC conversion using a software trigger (SWTRIG.START = 1). Subsequent conversions are triggered by the event generated by the DMA when the previous ADC result is transferred by the DMA to the destination (USART DATA register).

While the conversion is in progress, the application enters a while (1) loop and increments an idle loop counter (i.e., idle_loop_counter) and toggles the I/O pin PA04. The PA04 is probed using an oscilloscope to calculate the time taken to increment one idle loop counter and hence the total duration of time for which the CPU was idle.

Once all the 1024 ADC conversions are complete, another time stamp is taken (i.e., time_stamp2) in the DMA interrupt and a flag is set to indicate that the transfer is complete. Once the transfer is complete, the application calculates the total number of cycles taken to complete the transfer by taking the difference of the time stamp taken before the start of conversion (i.e., time_stamp1) and the time stamp taken when 1024 transfers are complete (i.e., time_stamp2). The idle loop counter value and the value of the total number of cycles to complete the transfer are printed on the terminal application running on the host computer.

Note: Refer to the CPU Utilization Analysis Between Different Cases for detailed description about the CPU utilization calculation from the results observed.

4.2 Peripheral-to Memory and Memory-to-Peripheral Transfer with the DMAC (ADC to SRAM and SRAM to USART)

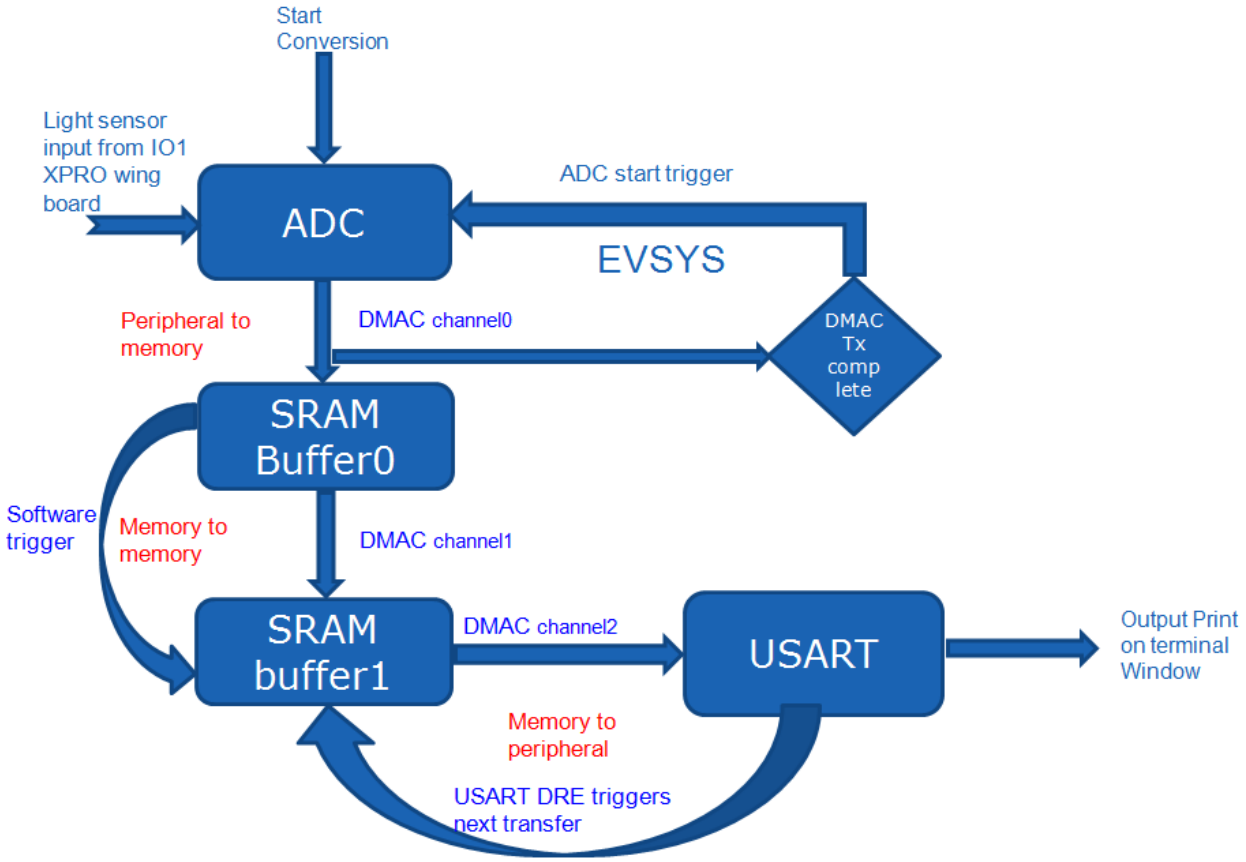
In this case, three DMAC channels have been used to demonstrate each transfer type. The purpose of having Memory-to-Memory type DMA transfer is for demonstration purposes only, and the application does not need this for its proper working.

Note: Refer to Atmel Start example document *ADC DMAC MEM MEM USART*.

4.2.1 Application Configuration and Implementation

In this example, three DMA channels are used. The DMA channel 0 is configured to transfer 1024 number of beats (1 beat = 1 byte) from the ADC RESULT register (peripheral) to the SRAM buffer (memory) using the ADC result ready as the trigger source. The DMA channel 1 is configured to transfer 1024 bytes from the SRAM buffer (memory) to another SRAM buffer (memory) using a software trigger. The DMA channel 2 is configured to transfer the 1024 bytes from the second SRAM buffer (memory) to the USART data register (peripheral) using USART DATA register is empty (DRE) as the trigger source. Overall the operation is done as shown in the following figure.

Figure 4-2. Peripheral to Memory and Memory to Peripheral Transfer with DMAC



DMA Channel 0 Configuration (Peripheral to Memory):

DMAC channel 0 is configured for a peripheral trigger from ADC RESULT ready. The source address is static as it is the ADC RESULT register. As the SRAM buffer will need to store 1024 bytes samples from ADC, the destination address is incremented. The event output is enabled on completing each beat transfer which is used to trigger the next ADC conversion.

- Select the DMA channel to configure (DMA channel 0 in this case) by setting CHID.ID[3:0] to 0x0
- Select the DMA trigger source as the ADC Result Ready by setting CHCTRLB.TRIGSRC[4:0] to 0x13 (ADC Result Ready Trigger)
- To perform a beat (8-bit) transfer on every trigger, set the trigger action in CHCTRLB.TRIGACT[1:0] to 0x2 (One trigger required for each beat transfer)
- Enable channel event output by setting CHCTRLB.EVOE = 1. This event output will be used by the ADC (through the event system) to start an ADC conversion.
- The DMA channel transfer descriptor registers are configured as configured as follows:
  - Source address is static and the destination address is incremented. BTCTRL.SRCINC = 0 and BTCTRL.DSTINC = 1
  - The beat size is configured to 1-byte. BTCTRL.BEATSIZE[1:0] = 0x0 (8-bit bus transfer).
  - To generate an event output when a beat (1-byte) transfer is complete, set the BTCTRL.EVOSEL[1:0] to 0x3 (Event strobe when beat transfer complete)
  - Configure the DMA channel ‘0’ source address (SRCADDR[31:0]) to ADC result register
  - Configure the DMA channel ‘0’ destination address (DSTADDR[31:0]) to the SRAM buffer

  - Set the block transfer count to 1024 by setting BTCNT[15:0] to 1024
  - Enable the DMA channel ‘0’ transfer complete interrupt by setting CHINTENSET.TCMPL to 1
  - The next descriptor address for channel ‘0’ must be set to NULL by setting DESCADDR to 0
- Enable the DMA channel ‘0’ transfer complete interrupt by setting the CHINTENSET.TCMPL to 1

Once 1024 bytes samples get transferred from the ADC to the SRAM buffer, the DMA block transfer complete interrupt handler is executed. From the interrupt handler the DMA channel 1 transfer is triggered.

DMA Channel 1 Configuration (Memory-to-Memory):

Channel 1 is configured with software trigger and transfer type is transaction, that is, once software triggers the transfer, all the ADC results stored in one SRAM buffer are transferred to another buffer stored in SRAM. Because both source and destination address are SRAM buffers, both source and destination addresses are incremented.

- Select the DMA channel to configure (DMA channel 1 in this case) by setting CHID.ID[3:0] to 0x1
- Select the DMA trigger source as software trigger by setting CHCTRLB.TRIGSRC[4:0] to 0x00 (Only software/event triggers)
- To transfer all 1024 bytes from one SRAM buffer to another on a single trigger, set the trigger action in CHCTRLB.TRIGACT[1:0] to 0x3 (One trigger required for each transaction)
- The DMA channel transfer descriptor registers are configured as configured as follows:
  - Source and destination address are incremented. BTCTRL.SRCINC = 1 and BTCTRL.DSTINC = 1
  - Set the address increment step size to 1 beat by setting BTCTRL.STEPSIZE[2:0] to 0x0
  - The beat size is configured to 1-byte. BTCTRL.BEATSIZE[1:0] = 0x0 (8-bit bus transfer)
  - Configure the DMA channel 1 source address (SRCADDR[31:0]) to the address of the source SRAM buffer
  - Configure the DMA channel 0 destination address (DSTADDR[31:0]) to the address of the destination SRAM buffer
  - Set the block transfer count to 1024 by setting BTCNT[15:0] to 1024
  - The next descriptor address for channel 1 must be set to NULL by setting DESCADDR to 0
- Enable the DMA channel 1 transfer complete interrupt by setting CHINTENSET.TCMPL to 1

Once 1024 bytes get transferred from source buffer in SRAM to the destination buffer in SRAM, the DMA block transfer complete interrupt handler is executed. From the interrupt handler the DMA channel 2 transfer is triggered to transfer the data from the destination buffer in SRAM to USART.

DMA Channel 2 Configuration (Memory-to-Peripheral):

Channel 2 is configured to have peripheral trigger and beat transfer type. A byte from SRAM buffer should be written to the USART DATA register whenever it is empty, that is, whenever USART DATA register is empty (DRE) and is ready for new data to be written, it triggers a DMA transfer from source to destination over the channel 2.

- Select the DMA channel to configure (DMA channel 2 in this case) by setting CHID.ID[3:0] to 0x2
- Select the DMA trigger source as SERCOM0 TX Trigger by setting CHCTRLB.TRIGSRC[4:0] to 0x05
- To perform a beat (8-bit) transfer on every trigger, set the trigger action in CHCTRLB.TRIGACT[1:0] to 0x2 (One trigger required for each beat transfer)
- The DMA channel transfer descriptor registers are configured as configured as follows:

  - Source address is incremented and the destination address is static. BTCTRL.SRCINC =1 and BTCTRL.DSTINC = 0.
  - Set the address increment step size to 1 beat by setting BTCTRL.STEPSIZE[2:0] to 0x0
  - The beat size is configured to 1-byte. BTCTRL.BEATSIZE[1:0] = 0x0 (8-bit bus transfer).
  - Configure the DMA channel 2 source address (SRCADDR[31:0]) to the address of the SRAM buffer
  - Configure the DMA channel 2 destination address (DSTADDR[31:0]) to the address of USART transmit register
  - Set the block transfer count to 1024 by setting BTCNT[15:0] to 1024
  - The next descriptor address for channel 2 must be set to NULL by setting DESCADDR to 0
- Enable the DMA channel 2 transfer complete interrupt by setting CHINTENSET.TCMPL to 1

Unlike other channels, this channel should be enabled at the end of channel 1 transfer complete. The reason is that the USART DRE is always set as no communication occurred previously. So, if this channel is enabled during the initialization, as USART DRE is already set, the DMA transfer will start immediately on channel 2 which results in wrong operation.

After the DMA channel 2 completes the transfer, a flag is set to indicate end of transfer and the time stamp (i.e., time_stamp2) is taken for CPU utilization calculation.

Other peripheral configurations, such as Clock, ADC, Event System, USART and I/O remain same as the Peripheral-to-Peripheral Transfer with DMAC.

4.2.2 CPU Utilization Calculation

The application takes the initial time stamp by reading the SYSTICK current value register and saves it to a variable (i.e., time_stamp1). It then starts the first ADC conversion using a software trigger (SWTRIG.START = 1). Subsequent conversions are triggered by the event generated by the DMA when the previous ADC result is transferred by the DMA to the destination (USART DATA register).

Similar to the Peripheral to Peripheral Transfer with DMAC, the application enters a while (1) loop and increments an idle loop counter and toggles the I/O pin PA04, while the transfer is in progress. Once all the 1024 ADC conversions are complete, another time stamp (i.e., time_stamp2) is taken in the DMA channel 2 interrupt handler and a flag is set to indicate that the transfer is complete.

Note: Refer to the CPU Utilization Analysis Between Different Cases for further information.

4.3 Peripheral-to-Memory and Memory-to-Peripheral Transfer Without DMAC (ADC to SRAM and SRAM to USART)

In this case, the application is implemented through interrupt handling without using the DMA. This is done to demonstrate the DMAC usage in reducing the CPU load.

Peripheral-to-Memory: The ADC results are stored in ADC driver buffer (i.e., Buffer1).

Memory-to-Memory: Once all the ADC results are available, the data is copied from the ADC driver buffer (Buffer1) to the application buffer (i.e., Buffer2).

Memory-to-Peripheral: The application buffer (Buffer2) is transmitted to USART.

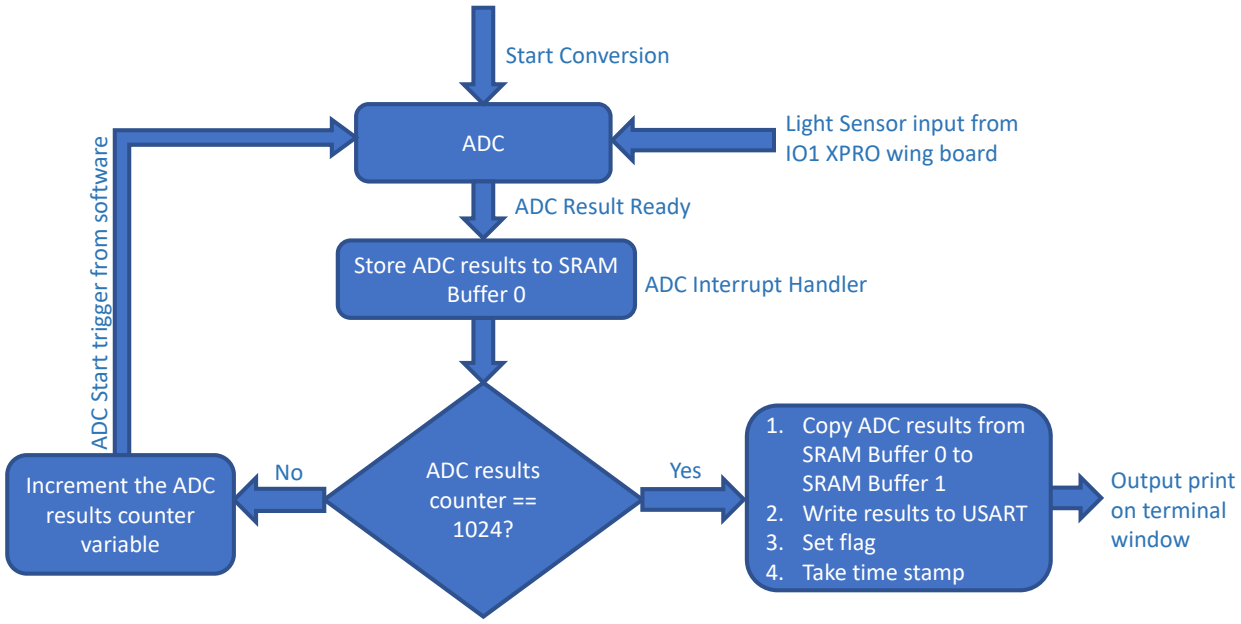
Note: Refer to the Atmel Start example *ADC NO DMAC MEM MEM USART*.

4.3.1 Application Configuration and Implementation

The Clock, USART and I/O configurations remain same as the Peripheral-to-Peripheral Transfer with DMAC. This example does not utilize the DMA and the Event System. The ADC configuration is same as the Peripheral-to- Peripheral Transfer with the DMAC except that the ADC is not configured to start a conversion on an event output by setting EVCTRL.STARTEI to '0'.

After the MCU and the peripheral initialization is complete, a software trigger in the main loop starts the first ADC conversion. Once the conversion is done, the ADC interrupt handler is called. In the interrupt handler, the ADC sample is saved in a buffer (i.e., SRAM Buffer 0) and a count variable indicating the number of ADC samples done is incremented. The next ADC conversion is triggered from interrupt handler. This continues until all 1024 samples are available. Once the count reaches 1024, ADC is disabled and further conversion is stopped. The ADC results are then read from the SRAM Buffer 0 and copied to another buffer (i.e., SRAM Buffer 1). The ADC results are then sent to USART and a flag is set to indicate that the transfer is complete. A time stamp is also taken now to find the CPU utilization. Overall the application flow would work as illustrated in the following figure.

Figure 4-3. ADC to SRAM and SRAM to USART Transfer without DMAC



4.3.2 CPU Utilization

The same logic is used to calculate the CPU utilization except that in this case, the interrupt is enabled and does not use the DMA. The number of conversions is counted by incrementing a counter. Once the counter reaches 1024, the ADC results are copied from the SRAM Buffer 0 to SRAM Buffer 1 and then transferred to USART. A time stamp is taken and `idle_loop_count` is noted to calculate the CPU utilization as shown in Logic Implementation Used to Calculate the CPU Utilization.

4.4 Logic Implementation Used to Calculate the CPU Utilization

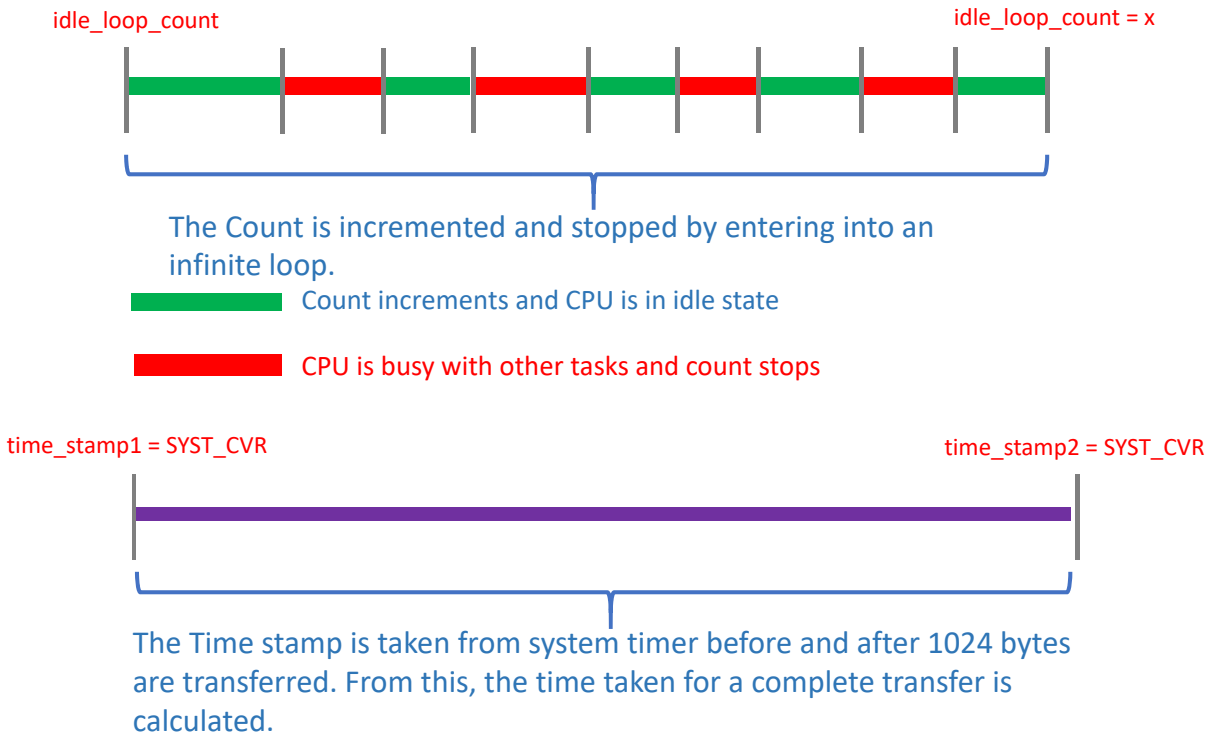
This section covers the logic implemented in this application to calculate the CPU utilization.

For calculating the CPU utilization, measure the total time taken for executing the data transfer routine. This is measured using the SYSTICK timer.

Measure the CPU idle time when executing the data transfer routine. This is measured by incrementing a variable (`idle_loop_count`) whenever the CPU is idle. The idle counter value is converted to time scale by multiplying the count value with the time taken to increment once.

Both the total time taken by the data transfer routine and the idle counter is measured for a fixed number of data transfers as shown in the following figure. In this test, it is a 1024 byte transfer.

Figure 4-4. CPU Utilization Calculation



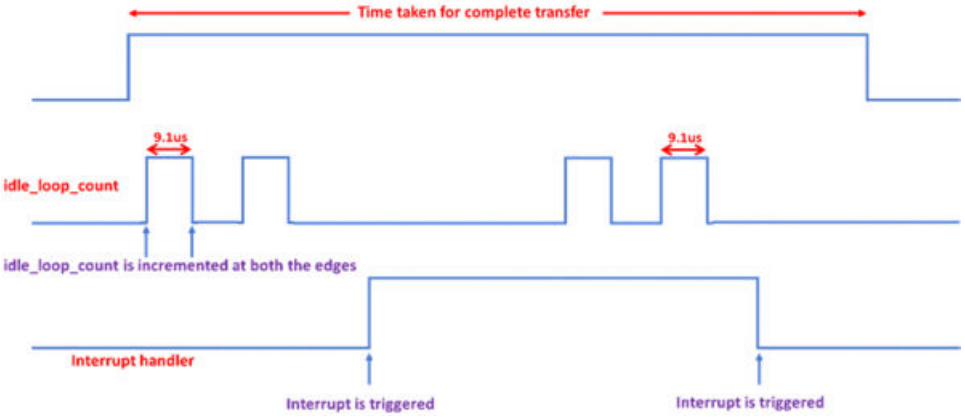
The number of cycles taken (`cycles_taken`) to complete the transaction can be calculated from the time stamp taken using the SYSTICK. As the SYSTICK runs the processor clock, the time taken for the total transaction can be calculated from the cycles taken, and the CPU clock frequency of the application is as follows.

Time taken to complete transaction = (cycles_taken/CPU clock frequency)

The `idle_loop_count` represents the number of times the code enters idle task. This can be used to derive the time that the CPU is idle during a complete transaction. To convert this count value to time scale, the time taken for each count increment should be known.

For this purpose, in the application code, two separate port pins are toggled in the idle loop and in the interrupt handler. Whenever the code enters the interrupt handler, the idle loop count stops and the pin toggled inside the idle loop stays at the same level. When the code comes out of the handler, the pin toggled inside the handler stays at same level, and the idle loop pin starts the toggle. The time taken for single toggling is calculated after removing the time taken for handler execution. This is shown in the following figure.

Figure 4-5. Calculation of Time Taken for a Single idle_loop_count Increment

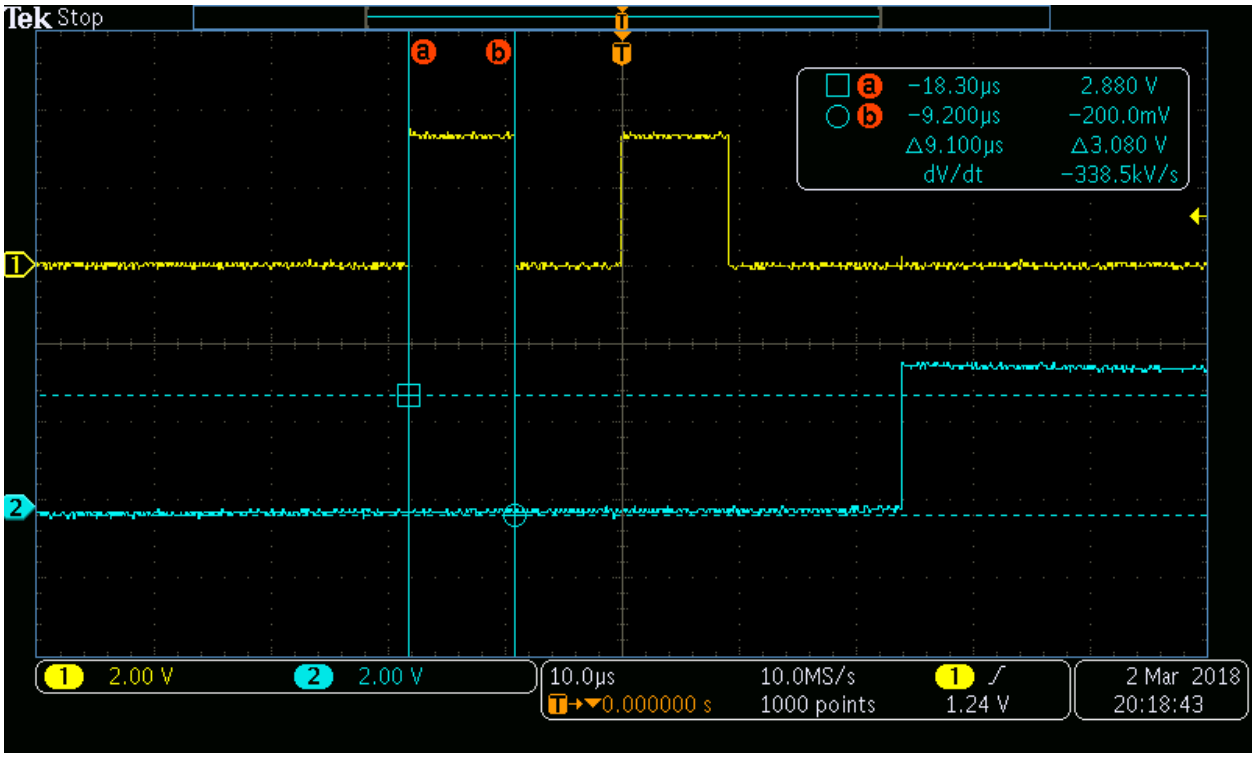


From the oscilloscope, the time taken for each count increment is calculated to be ~9.1µs, as shown in the following figure. The count value when multiplied with the pulse width (i.e., 9.1 µs) will give the time the CPU spends inside the idle task.

Note:

1. The width of the `idle_loop_count` pulse is the time taken to increment one idle count value when there is no interrupt triggered. The `idle_loop_count` pulse is output on the PA04 and PA06 is toggled in the interrupt handler.
2. The CPU idle time can be calculated by multiplying the `idle_loop_count` with the `idle_loop_count` pulse width.

Figure 4-6. Oscilloscope Shot of Idle Task



The CPU utilization analysis for each case is described in CPU Utilization Analysis Between Different Cases.

Note: The ideal expectation is that the idle loop count should be more when using the DMA than when not using it. When using the DMA, the CPU is not interrupted and the idle task can be executed in parallel. But in practice, this cannot be the case. The reason is that, the DMA will take less time to complete the transfer. In case of using the interrupt method, it takes more time to complete the transaction. Therefore, the time that code can spend for the ideal task would be lesser for the DMAC case and the `idle_loop_count` value can be lesser than when not using the DMA. To avoid such confusion, the time taken for completing the transfer is also taken using the system timer, and the ratio of both is used to calculate the CPU utilization.

5. Application Limitations

5.1 USART Baud-Rate and ADC Sampling Frequency

In the DMAC usage case, the ADC RESULT is directly written to the USART DATA register. The DMA triggers the next ADC conversion immediately once the data is written to the USART. This causes data loss on the terminal window if the USART baud rate is slower than the ADC conversion time. To avoid this, the ADC is configured with lowest possible frequency and the USART is configured with a higher baud-rate.

For the ADC:

The rate of conversion of the ADC clock depends on the `GCLK_ADC` (i.e. 16 MHz), and its pre-scalar, which is 64 in this case.

Therefore, the ADC clock frequency = 16 MHz/64 = 250 kHz ~= 4 µs.

Conversion time = 8 cycles (8-bit resolution) + 1 cycle (Sampling time) = 9 * 4 µs ~= 36 µs

For the USART:

In the product data sheet under *Baud Rate Equations*, the fbaud should be ≤ fref/S.

For the Asynchronous Arithmetic mode, the number of samples per bit is (S) = 16.

fref = 16 MHz

So, the maximum possible baud-rate = 16 MHz/16 = 1000000.

The baud-rate configured = 460800 (i.e. 460800 bits sent in = 1s).

For 10 bits, it takes = (10/460800) ~= 21.7µs.

Therefore, setting a 460800 baud rate is advisable, as the ADC sample is ready for every 36 µs. The USART would have sent the previous data in 21.7 µs, and waits for the next ADC result without any data loss.

Note:

1. The 10 bits come from the USART data frame = Start bit (1) + Data bit (8) + Stop bit (1).
2. Refer to the device data sheet for additional information on timing calculations on the ADC and USART.

5.2 SRAM to SRAM Transfer Type

For the ADC DMAC MEM MEM USART and the ADC NO DMAC MEM MEM USART examples, the transfer type Memory-to-Memory, that is, a copy of the ADC results from one SRAM buffer to another SRAM buffer is done for demonstration purposes. This application does not require this to work properly.

6. CPU Utilization Analysis Between Different Cases

After programming the firmware, the results can be seen in the terminal window. The result contains the ADC result data, number of cycles taken, and idle loop count in 'hex' format. This chapter explains how to derive the CPU usage for each case from the results observed. The calculation is done as explained in Logic Implementation Used to Calculate the CPU Utilization.

Note: The results shown in this document are taken with the following conditions. The resulting `idle_loop_count` and `cycles_taken` will vary with the optimizations, frequency or any change in the application code.

- Optimization set to zero (-O0)
- Configuration: Debug
- Port toggling function (`ENABLE_PORT_TOGGLE`) is enabled
- OSC16M internal oscillator is used with frequency selected as 16 MHz

6.1 CPU Frequency Calculation

To find the time taken, the CPU frequency needs to be known. This application runs at 16 MHz (OSC16M). The accuracy on the internal RC oscillator can be taken from the Electrical Characteristics section of the product data sheet. The accuracy of the RC in the board used for testing was calculated to be 15.98 MHz. This is done by giving the main clock, the GCLK0 (which runs at 16 MHz) output to the I/O pin.

Note:

1. The I/O Pin used here is PA27.
2. The GCLK0 output is enabled in all the three example projects.

6.2 CPU Idle Time Calculation from Result Observed

The following figures display the results of various cases used in the application.

Note: The ADC values are printed in Hex format. Therefore, the terminal emulator program must be configured to display the Hex values. Use the following steps to configure Tera Term to display Hex values:

1. Go to the installation folder of Tera Term and search for the file

   ```
   TERATERM.INI
   ```

   .
2. Open the

   ```
   TERATERM.INI
   ```

   file and set the Debug variable to ON.
3. Open Tera Term and setup the connection. Press the Shift + Esc keys twice to change the mode to the Hex format.

Figure 6-1. ADC DMAC USART Terminal Output

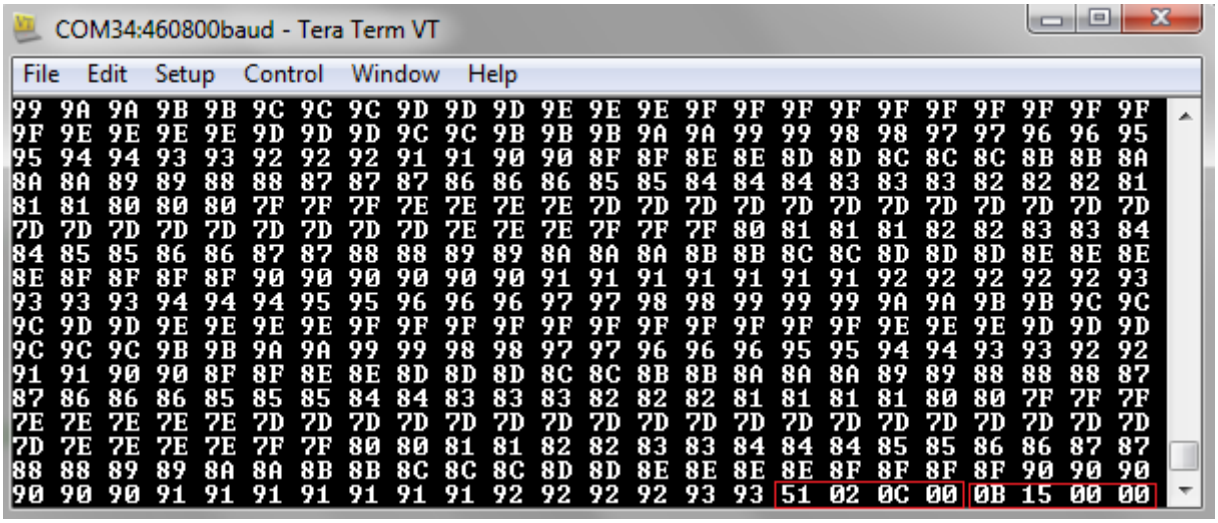


Figure 6-2. ADC DMAC MEM MEM USART Terminal Output

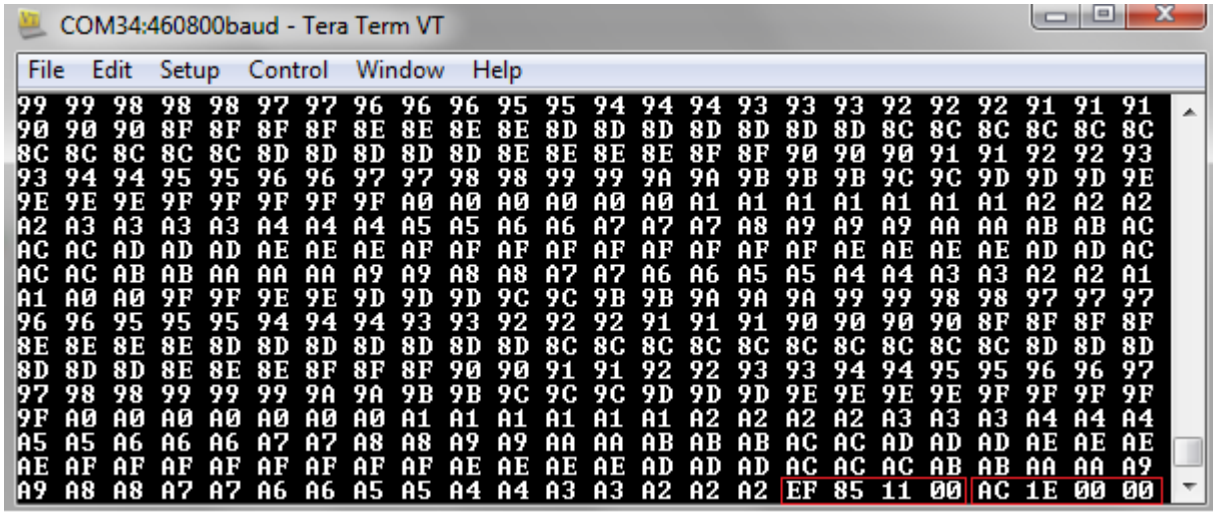
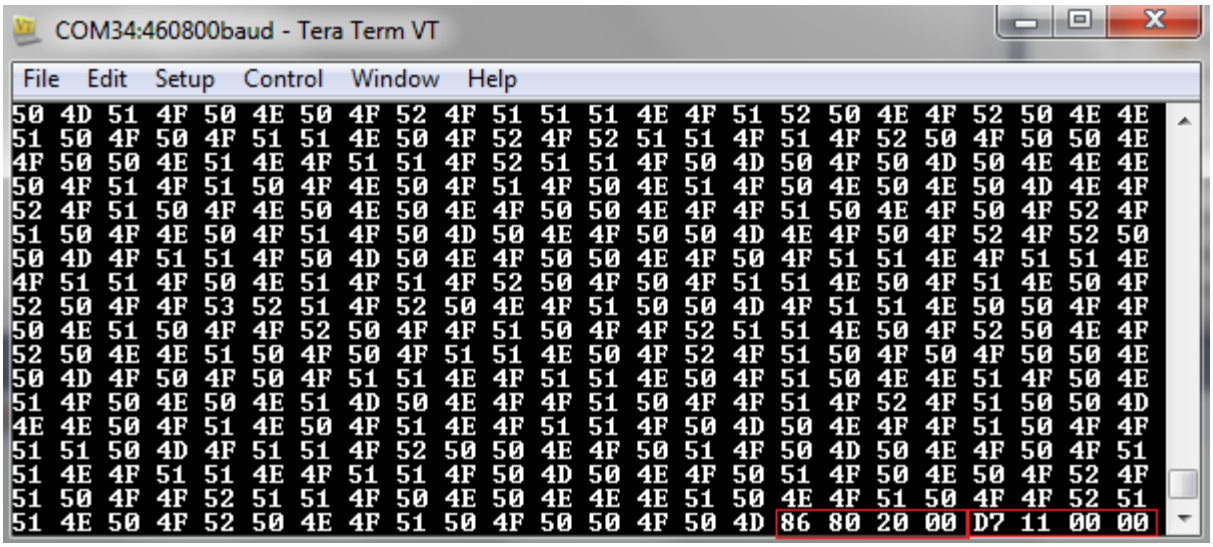


Figure 6-3. ADC NO DMAC MEM MEM USART Terminal Output



ADC DMAC USART:

The last eight bytes of data represent the `idle_loop_count` and the `cycles_taken` in big endian format shown in the following equations. The last four bytes of the result is the `idle_loop_count` and the next four bytes is the `cycles_taken`.

`idle_loop_count` = 0x0000150B = 5387d

`cycles_taken` = 0x000C0251 = 787025d

Time taken to complete the transaction = (cycles_taken/CPU clock frequency)

The time taken to complete the transaction = (787025/15.98) µs = 49.250 ms.

The time taken for each idle count is calculated to be 3.391 µs as in Logic Implementation Used to calculate the CPU Utilization.

Total CPU idle time = `idle_loop_count` * 9.1 µs = 5387 * 9.1 µs = 49.021 ms

Therefore, in a 49.250 ms transfer period, the CPU is idle for about 49.021 ms for the `ADC_DMAC_USART` case.

Similarly the calculation can be done for other cases as shown in the following table.

Table 6-1. CPU Usage Calculation

Case	Idle_Loop_Count	Cycles_Taken	Total Transfer Time (ms)	CPU Idle Time (ms)	CPU Idle Time (%)
ADC DMAC USART	0x0000150B	0x000C0251	49.250	49.021	99.53
ADC DMAC MEM MEM USART	0x00001EAC	0x001185EF	71.864	71.453	99.42
ADC NO DMAC MEM MEM USART	0x000011D7	0x00208086	133.294	41.559	31.178

When using the DMAC, the CPU is idle mostly during the data transfer. But without the DMAC, the CPU is in Idle mode only for a small portion of the data transfer, and the overall transfer time is also high.

Note: The table serves to provide an indication of performance improvement that can be gained by using the DMA, and is not intended to draw a comparison between the different use case scenarios.

7. Execution of Application

The example applications corresponding to this document can be downloaded from the following location: start.atmel.com. Follow these steps to download the example project.

1. Go to start.atmel.com and click on the *Browse Examples*tab.
2. *Search* for the desired project:
   - `ADC DMAC USART`
   - `ADC DMAC MEM MEM USART`
   - `ADC NO DMAC MEM MEM USART`
3. Click on **Open Selected Example** to view the software components in the project or click on **Download Selected Example** to download the example project.

8. References

ARM documentation on Cortex-M23 Core

- Cortex-M23 Processor Technical Reference Manual revision r1p0

Device data sheet

This data sheet contains the block diagrams of the peripherals and details about implementing firmware for the device. It also contains the electrical specifications and expected characteristics of the device.

This data sheet is available for download at https://www.microchip.com. Follow this path to download the document *Products > Microcontrollers & Microprocessors > 32-bit MCUs > SAM 32-bit MCUs > SAM L MCUs*.

Hardware tools user's guide

- SAM L10 Xplained Pro User's Guide and Schematics is available at the following location: http://www.microchip.com/DevelopmentTools/ProductDetails/dm320204
- For SAM L11 Xplained Pro User's Guide and Schematics is availabe at the following location: http://www.microchip.com/DevelopmentTools/ProductDetails/dm320205

The Microchip Web Site

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

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

Customer Change Notification Service

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

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

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

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

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

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

Legal Notice

Trademarks

ISBN: 978-1-5224-3535-8

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820