**Freescale Semiconductor**
Application Note

Document Number: AN3396
Rev. 1, 02/2011

# MPC5500: A Decimation Solution with Zero CPU Loading

by: Geoff Emerson
David McMenamin
MCD Applications
East Kilbride, Scotland

**Contents**

## 1 Introduction

The MPC5500 microcontroller family provides a flexible feature set enabling a wide range of automotive and industrial applications to be realized. This application note describes a system created using some of these powerful features to perform decimation with no CPU loading during run time. The principles presented by this application note could also be applied in realizing other systems.

This application note describes a system that performs eQADC result data decimation. The decimation is achieved using zero CPU bandwidth at run time. Data produced by an analog-to-digital converter is moved to a coprocessors memory where this data can be processed. The processed data is then moved back to system RAM. The component parts of the system (eQADC, eDMA, eTPU, and eMIOS) as they pertain to the decimation system are described. The interaction between these subsystems is also described. The particular system described is a simple averaging decimation system. However, because the decimation calculations are performed by software on the eTPU coprocessor, a more complex filtering system could be implemented if desired. The example software referred to and used in the development of this application note is downloadable from www.freescale.com.

## 2 Overview of Decimation

Decimation is the process of effectively reducing the data rate. This is achieved by filtering and then down sampling results from a system. In this example, decimation is performed on the results of the eQADC.

To prevent aliasing when performing decimation, the highest frequency of the sampled signal must be less than half the post decimation sampling rate to ensure compliance with the Nyquist sampling theorem. If this is not the case, the incoming signal or captured results must be low-passed filtered prior to sampling to ensure it complies. Violating the Nyquist criteria results in aliasing of the signal.

Decimation can be performed to reduce the sampling rate so it can be used as an input to a slower system or to simply reduce the volume of data that has to be processed. The decimation factor is the ratio of the number of samples in to the number of samples out.

## 3 System Overview

Four subsystems (peripheral modules and coprocessor) are configured at system initialization by the CPU: eMIOS, eQADC, eTPU, and eDMA. These modules can then interact and operate independently of the CPU to realize the decimation system.

The eMIOS generates a regular sampling pulse that triggers the eQADC. The eQADC operates as instructed by commands stored in system RAM and supplied by the eDMA. The eDMA performs all data moves between the subsystem modules and system RAM. Two eDMA channels are used. One, Channel 0, provides conversion commands to the eQADC. A second, Channel 1, moves the data within the system. Channel 1 performs a chain of different transfers by reconfiguring itself using the scatter gather processing functionality built into the eDMA. This is covered in more detail in Section 4.3, “eDMA Subsystem”. The eTPU performs the decimation computation. An overview of the decimation system is shown in Figure 1.

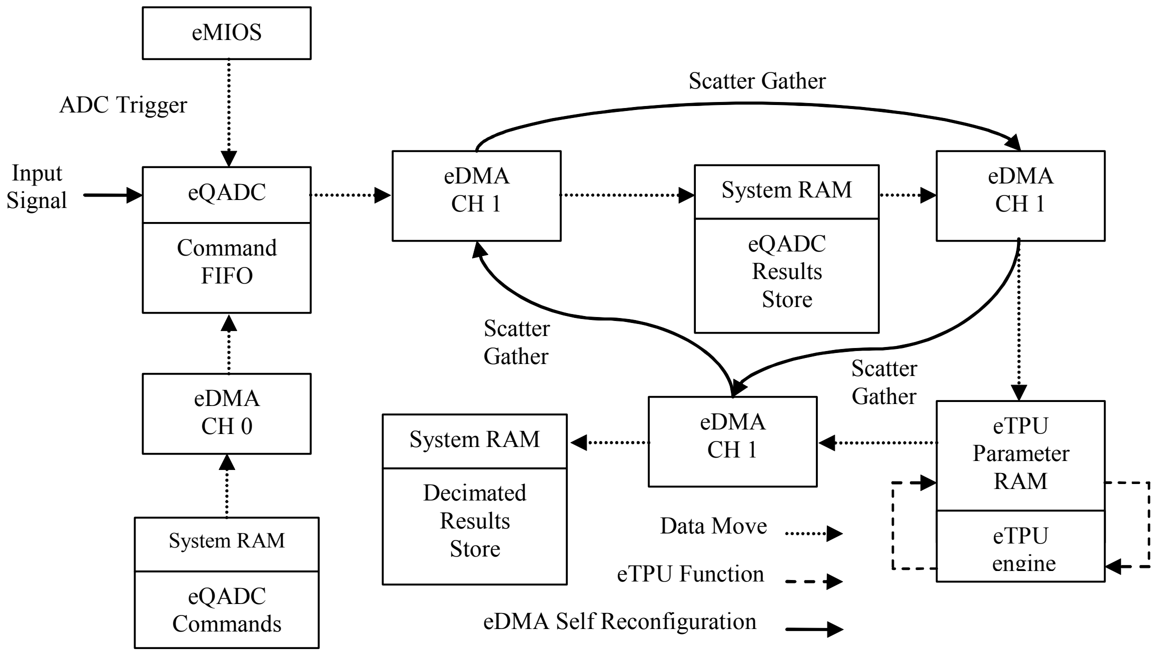

**Figure 1. Decimation System Overview**

# 4 Subsystems Overview

The decimation system consists of four subsystems described below.

## 4.1 eMIOS Subsystem

The eMIOS generates a regular pulse within the decimation system that triggers the analog-to-digital converter to sample at a regular frequency.

### 4.1.1 Overview of the eMIOS Hardware

The eMIOS is an evolution of the previously implemented MIOS module present in other Freescale products, such as the MPC500 family. The eMIOS module provides the ability to generate and measure timed events in various ways.

The MPC5500 features up to 24 eMIOS channels. Each of the 24 unified channels is identical and can operate independently of the other channels to provide a timed input or output function. Each eMIOS channel pin is multiplexed with GPIO functionality. The device pin must be configured so the eMIOS functionality is selected.

Each channel has its own counter used for reference when generating and measuring timed events. This counter is driven by a clock that can be scaled at module and channel level. Channel counters have a resolution of 24 bits. Up to four counter buses can be used to share a common time reference between the unified channels. Each unified channel supports a range of timing modes including: input pulse width measurement, output pulse width, and frequency modulation and output pulse width modulation. For full details on the module and the modes supported, refer to the relevant MPC5500 family member reference manual.

### 4.1.2 Triggering the eQADC

Within the decimation system, the eMIOS generates an output pulse width and frequency modulated (OPWFM) signal. The rising edge of this signal triggers the eQADC. When triggered, the eQADC performs a conversion and generates a result based on the command words present in a command FIFO.

The OPWFM signal is configured to have a frequency of 200 kHz. This is generated from an 80 MHz system frequency and a time base generated by the internal counter of the channel. In the example software provided, eMIOS channel 10 is used. The eMIOS channel's internal counter is driven at the system frequency of 80 MHz. This is achieved by configuring the eMIOS module prescaler and the channel prescaler to be 1. The period of the OPWFM signal is set to be 400 counts of the internal counter at 80 Mhz. This results in 200 kHz frequency. The duty cycle of the signal is configured to be 50 percent by setting the output to toggle at 200 counts of the internal counter. The resulting output signal is shown in Figure 2. The signal toggle count values (Duty and Period) are written to the A and B registers of the eMIOS channel. The output signal changes when the internal counter matches these values. On a match with the B register, the internal counter is also reset. In the example, the EDPOL bit in the eMIOS channel is left in its default state, which is 0. When the internal counter matches the A register, the output goes low. When it matches the B register, the output goes high.

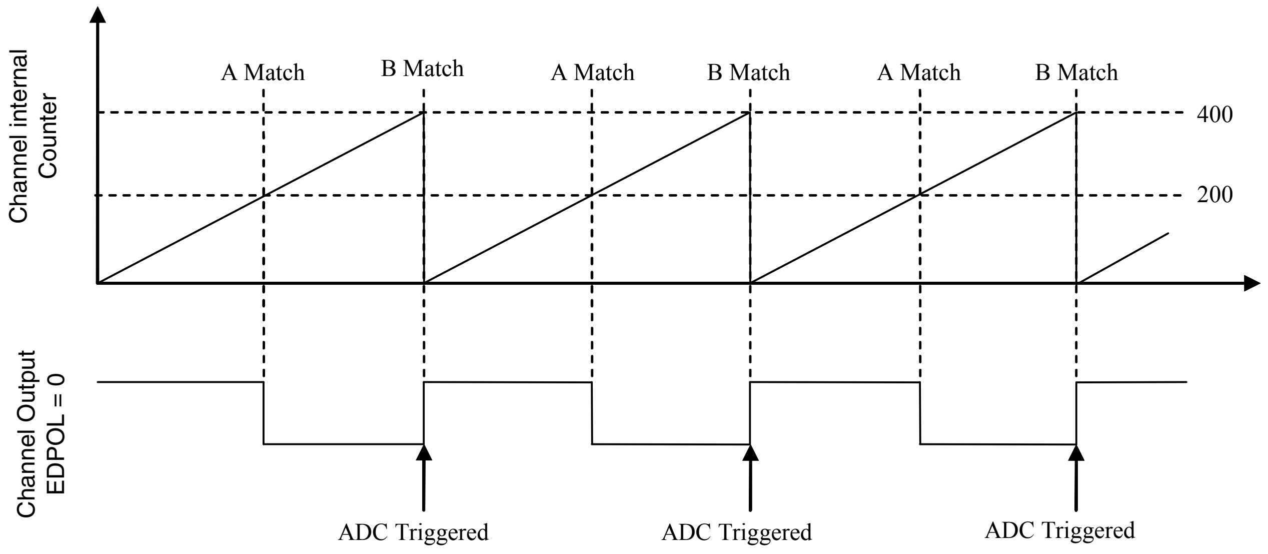

**Figure 2. eMIOS OPWFM Output Signal for Triggering the eQADC**

The eMIOS channel 10 output is routed internally to the eQADC module. This is configured in the eQADC trigger input select register, which is located within the system integration unit (SIU). The code to configure this register is shown below:

```
SIU.ETISR.R = 0x00200000; //emios10 drives the trigger
```

## 4.2 eTPU Subsystem

The eTPU subsystem is composed of the eTPU hardware along with the eTPU function software. These are described in the following sections.

### 4.2.1 Overview of the eTPU Hardware

The eTPU is a second-generation coprocessor module available on Freescale microcontrollers. It is based on its successful predecessor, the TPU. While the intended application of the eTPU is primarily time critical applications (e.g. engine or motor control), it is flexible enough to allow the micro-engine to be used to perform calculations unrelated to any timing functionality. The channel and timer hardware, which would typically be used to drive timed outputs or capture input signals, is not used in this particular system. Consequently, this hardware is not described here. The micro-engine is used to perform the decimation calculations.

An eTPU micro-engine has 32 timer channels associated with it. Many MPC5500 family members have a single or dual engine eTPU. Function code is executed by the micro-engine in response to an eTPU channel requesting service. One means of requesting service is by a host service request (HSR). This is achieved by setting the appropriate bit field in the host interface of the eTPU channel to be serviced. In the example decimation system, the eDMA writes to the host service request register.

Function code resides in the eTPU's shared code memory and is placed there at system initialization by software running on the host CPU. Function code is associated with a particular eTPU channel by populating the channel function select bit field in the channel's configuration register.

Functions may store and retrieve parameters from parameter RAM (PRAM). In the example decimation system, parameters in PRAM are accessed by the eDMA controller and by the eTPU decimation function.

For more information regarding the eTPU, refer to http://www.freescale.com/etpu.

### 4.2.2 Example eTPU Decimation Function

The decimation scheme chosen for the example decimation system is a simple averaging scheme. More complex schemes could be realized using the eTPU's multiply accumulate and divide unit (MDU). The MDU is an autonomous resource in the micro-engine that can carry out sequential multiply, multiply-accumulate fractional multiplication, and divide operations in parallel to the eTPU RISC core executing other non-MDU microinstructions.

When called by a host service request, the example eTPU decimator function simply sums four values stored in PRAM and then divides the sum by four. The result is then written to a different location in PRAM. This result is called the decimated value.

The example function is also responsible for maintaining a copy of the destination address parameter of the transfer control descriptor (TCD) used to move the decimated value from eTPU PRAM to the decimated value result queue (referred to as Decimator_Ping_Pong). The destination address pointer is incremented each time the function is called. If the pointer exceeds the allocated queue space, it is reset to the start of the queue.

The example function's single entry point is called using a host service request.

### 4.2.3 Parameter RAM Layout.

The Bytecraft eTPUC compiler optimizes how the function parameters are arranged in PRAM. Version 1.0.7.30 of the Bytecraft compiler has chosen the following arrangement for the function's parameters.

**Table 1. eTPU DRAM Layout**

| Address | Variable | Variable |
|---|---|---|
| eTPU Channel base address | (int16) Input_Values[0] | (int16) Input_Values[1] |
| eTPU Channel base address + 4 | (int16) Input_Values[2] | (int16) Input_Values[3] |
| eTPU Channel base address + 8 | (int16) Input_Values[4] | (int16) Input_Values[5] |
| eTPU Channel base address +12 | (int16) Input_Values[6] | (int16) Input_Values[7] |
| eTPU Channel base address + 16 | (int16) Decimated_value | (int16) Timestamp |
| eTPU Channel base address + 20 | (int32) TCD_Daddr | |
| eTPU Channel base address + 24 | (int24) Tcd_Doff | |
| eTPU Channel base address + 28 | (uint32) Ping_Pong_Address | |
| eTPU Channel base address + 32 | (int16) Ping_Pong_Length | |

Input_Values[0,2,4,6] are the result of the ADC conversion result. Input_Values[1,3,5,7] are the corresponding time stamp for each conversion.

The Bytecraft eTPUC compiler uses #pragma statements to write the layout information for these parameters to a file called etpu_decimate_auto.h. This file is used by application code to access the PRAM locations.

## 4.3 eDMA Subsystem

The eDMA performs all run time data moves within the system. Two eDMA channels are used. Channel 0 is used to provide conversion commands to the eQADC module, and Channel 1 is used to move data around the system. Because Channel 1 is used to do multiple data transfers, it reconfigures itself between transfers using the scatter gather processing feature of the eDMA.

### 4.3.1 Overview of the eDMA Hardware

An eDMA controller provides the ability to move data from one memory mapped location to another. After it is configured and initiated, the eDMA controller operates in parallel with and independently from the central processing unit (CPU), performing data transfers that would otherwise have to be managed by the CPU. This results in reduced CPU loading and a corresponding increase in system performance. Figure 3 illustrates the basic functionality provided by an eDMA controller.

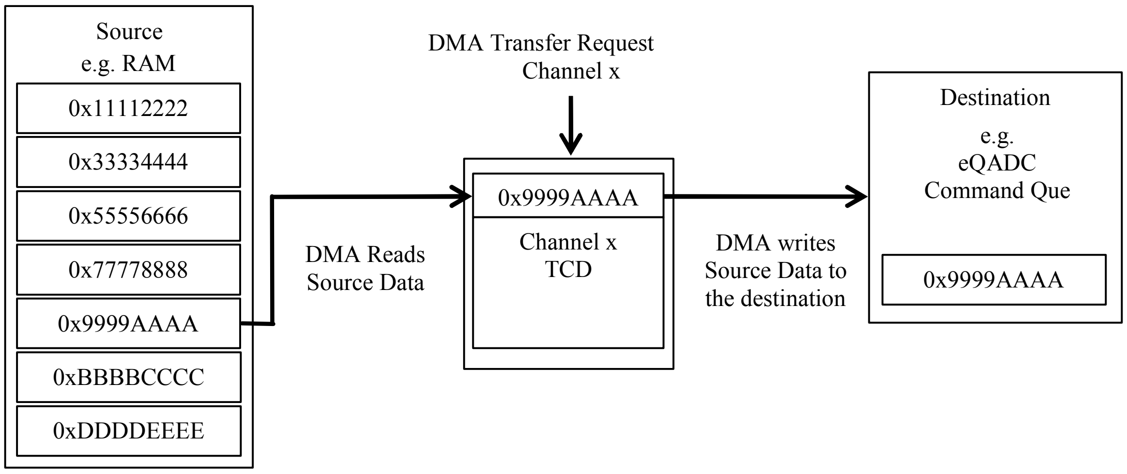

**Figure 3. eDMA Basic Functionality**

MPC5500 devices feature up to 64 eDMA channels. Each channel can be independently configured with the details of the transfer sequence to be executed. These details are specified in the channel transfer control descriptor (TCD) memory array.

In this example, eDMA transfers are activated in two ways:

- Events occurring in peripheral modules and off chip assert an eDMA transfer request.
- Software activation.

Peripheral module interrupt flags can be configured to cause an interrupt or an eDMA request. Each eDMA channel is linked to a specific source.

Each channel can generate an interrupt to indicate it has partially completed or fully completed a transfer. Interrupts can also be generated to indicate a transfer error has occurred. Scatter/gather processing is supported by each of the channels. This feature allows a channel to automatically load a new TCD into its registers. Multiple TCD descriptors can, therefore, be used with a single channel with no extra load on the CPU at run time.

To allow the eDMA and the CPU to operate simultaneously, a multi-master bus architecture is implemented. The MPC5500 multi-master bus has multiple master and slave nodes. The cross-bar switch forms the heart of this multi-master architecture. It links each master to the required slave device. If two masters attempt to access the same slave, an arbitration scheme eliminates bus contention.

### 4.3.2 Transfer Control Descriptors Description.

Channel 0 transfer control descriptor is configured such that every time the channel is triggered by the eQADC command FIFO fill flag, the eDMA reads a command word stored in system RAM and writes it to the push register for the command word within the eQADC. The eQADC is triggered by the eMIOS that causes the eQADC to trigger the eDMA to provide a new command word each time space becomes available on the FIFO. Sixty four command words, each 32-bits in length, are passed to the FIFO in this example. The command words are stored in system RAM in an array called CQUEUE. The eDMA wraps back to the beginning of the array after it has passed the last array member to the eQADC. Channel 0 TCD1 below lists the configuration used for Channel 0’s TCD.

**Table 2. Channel 0 TCD**

| TCD name | TCD_ADC_CQ |
|---|---|
| Purpose | Move 64 x commands from CQUEUE to PUSHR |
| Trigger | Command FIFO Fill Flag |
| Source Address | CQUEUE |
| Destination Address | PUSHR |
| Source / Destination Modulo | off |
| Source size | 0x2 → 32-bits |
| Source address offset | 0x4 → 32 bits |
| Destination Address offset | 0 |
| Nbytes – transferred each trigger | 0x4 → 32 bits |
| Last source address adjust | –4 bytes x 64 = –256 bytes |
| Citer | 64 |
| Biter | 64 |
| dsize | 0x2 → 32 |
| Half way interrupt | Off |
| Major loop interrupt | Off |
| Channel Linking | Off |
| Scatter Gather | Off |

Channel 1 is used to perform four different transfers. A separate transfer control descriptor is used for each of these transfers. The four different transfers are performed by enabling scatter gather processing. After the eDMA has completed one transfer, as defined by the TCD, it loads the TCD for the next transfer. The order the scatter gather processing uses the TCDs is shown in Figure 4.

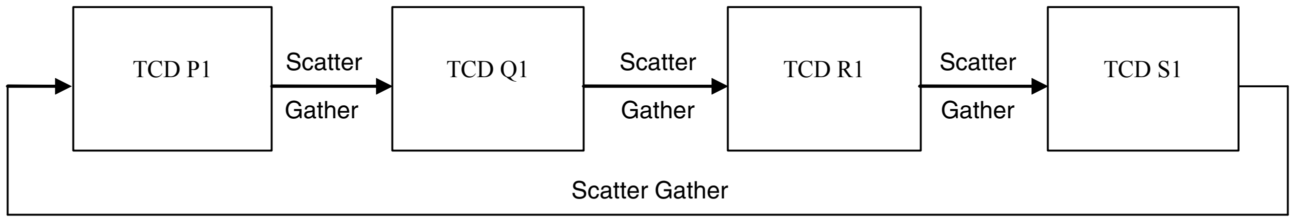

**Figure 4. TCD Scatter Gather Flow**

TCD P1 is processed first. It is used to move four eQADC results and corresponding timestamps into the eTPU's PRAM memory. Each transferred set consists of a 16-bit timestamp and a 16-bit conversion result. This results in 8 × 16 bits being moved. Data is moved each time the eDMA is triggered by the eQADC RPOP flag. After this channel has been triggered eight times (four results and four timestamps), the major loop is complete and scatter gather processing takes place.

TCD Q1 is then used to move the resultant decimated value, which has been previously calculated by the eTPU, from the eTPU's PRAM to the result array in System RAM. This transfer is triggered when the scatter gather processing updated the start bit in the eDMA channel. After the transfer has been performed, scatter gather processing loads TCD R1. Table 2 shows the settings used for TCD P1 and TCD Q1.

**Table 3. Channel 1 TCD P1 and Q1**

| TCD name | TCD P1 | TCD Q1 |
|---|---|---|
| Purpose | Move eQADC results and timestamps to eTPU Parameter RAM | Move decimation result from Parameter RAM into System RAM |
| Trigger | eQADC RPOP Flag | Software from P1 |
| Source Address | Result Pop Register | Parameter RAM |
| Destination Address | eTPU Rqueue | System RAM |
| Source / Destination Modulo | off | Off |
| Source size | 0x1 → 16-bits | 0x1 → 16-bits |
| Source address offset | 0 | 0 |
| Destination Address offset | 0x2 → 16 bits | 0 |
| Nbytes – transferred each trigger | 0x2 → 16 bits | 0x2 → 16 bits |
| Last source address adjust | 0 | 0 |
| Citer | 8 | 1 |
| Biter | 8 | 1 |
| Dsize | 0x1 → 16-bits | 0x1 → 16-bits |
| Half way interrupt | Off | Off |
| Major loop interrupt | Off | Off |
| Channel Linking | Off | Off |
| Scatter Gather | On TCD Q1 | On TCD R1 |

TCD R1 is used to update the destination address of TCD Q1. The latter defines the location where the next decimated result is stored in the results array. This location address has been calculated by the eTPU decimation function. This transfer is triggered when the scatter gather processing updated the start bit in the eDMA channel. The updated destination address is used next time TCD Q1 is loaded and the channel is triggered. After it is complete, R1 scatter gathers to TCD S1.

TCD S1 triggers the eTPU host service request. This host service request causes code to be executed on the eTPU that performs the decimation calculation and manages the destination address for TCD Q1. The DMA transfer is triggered when the scatter gather process updates the start bit. After this has completed, TCD P1 is loaded (by scatter gather processing) to start the cycle again.

**Table 4. Channel 1 TCD R1 and S1**

| TCD name | TCD R1 | TCD S1 |
|---|---|---|
| Purpose | Update destination address of TCD Q1 | Write the eTPU host service request |
| Trigger | Software from Q1 | Software from R1 |
| Source Address | RAM | Parameter RAM |
| Destination Address | TCD Q1 | Host Service Request |
| Source / Destination Modulo | off | Off |
| Source size | 0x4 → 32-bits | 0x4 → 32-bits |
| Source address offset | 0 | 0 |
| Destination Size | 0x4 → 32 bits | 0x4 → 32-bits |
| Nbytes – transferred each trigger | 0x4 → 32-bits | 0x4 → 32 bits |
| Last source address adjust | 0 | 0 |
| Citer | 8 | 1 |
| Biter | 8 | 1 |
| dsize | 0x2 → 32-bits | 0x2 → 32-bits |
| Half way interrupt | Off | Off |
| Major loop interrupt | Off | Off |
| Channel Linking | Off | Off |
| Scatter Gather | On TCD S1 | On TCD P1 |

**NOTE**

Ideally, TCD_Q1 would reside in eTPU PRAM and its destination address would be maintained by the eTPU decimation function. However, on all of the currently available MPC5500 family members, peripheral bridge A is configured so PRAM supports 32 bits accesses. The eDMA requires 64 bit accesses to correctly load a TCD. The workaround for this issue was to insert an additional scatter gather step using TCD_R1 that copies the destination address calculated by the eTPU decimation function from PRAM to TCD_Q1's destination address location is system RAM.

## 4.4 eQADC Subsystem

The enhanced queued analog-to-digital converter (eQADC) converts analog voltages to a digital representation of those voltages. Conversions produce 16 bit digital results and optionally timestamp information. The eQADC has six command FIFOs and six result FIFOs, all with eDMA support. Each command FIFO can be supplied with command messages from a queue in system RAM. Command messages may contain configuration commands or conversion commands. Conversion commands produce results sent to a selected result FIFO. Values from the result FIFO may, in turn, be transferred to a queue in system RAM under eDMA control.

Prior to being configured for use with command messages from a command queue in system RAM, the eQADC is initialized under eDMA control with command messages (configuration commands) that write to internal registers in the eQADC to enable the eQADC, set up the eQADC clock, conversion speed and timebase counter register. These configuration commands produce no results data.

In the example decimation system, a single command FIFO and a single result FIFO are used. The eQADC has two converters. In the example decimation system, only one of the converters, ADC0, is used. The basic movement of commands and results data is shown below.

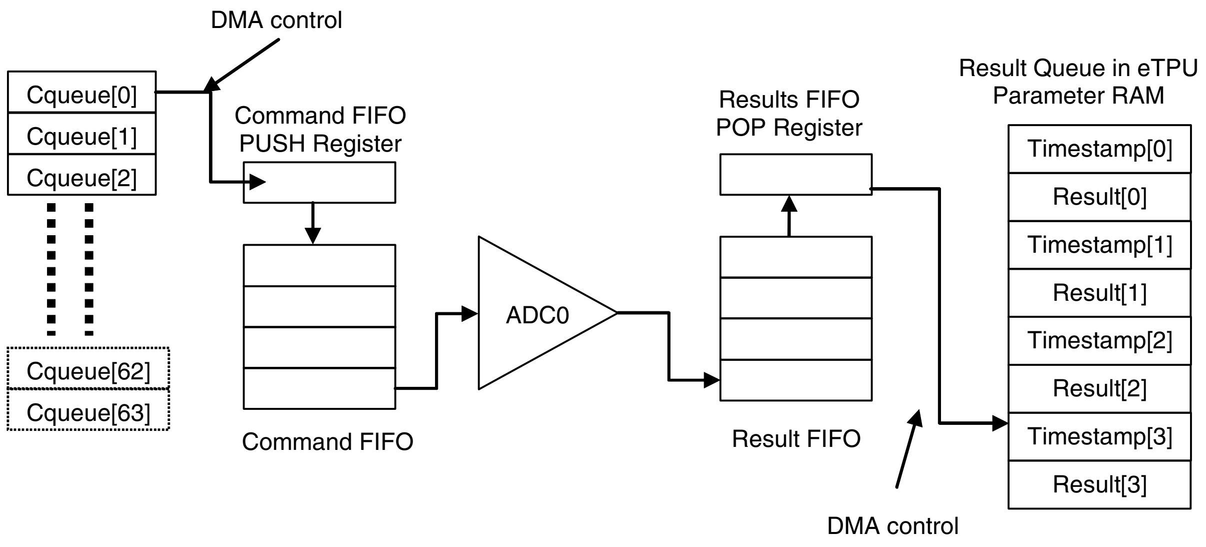

**Figure 5. Basic Data Flow in the eQADC Sub-system**

The eQADC's command FIFO has eDMA flags associated with it. One of these, the command FIFO fill flag, can signal to the eDMA that the CFIFO is not full. When this happens, the eDMA copies the next value in the command queue into the CFIFO push register. Commands propagate through the command FIFO before being used by the ADC converter.

When a conversion is triggered, a conversion command message is taken from the command FIFO and an analog-to-digital conversion is performed according to the instructions contained in the conversion command message. The result is sent to the selected results FIFO. In the example decimation system, timestamps are enabled so each conversion results in 2 × 16 bit values being sent to the results FIFO. The first value is the digital representation of the analog voltage that has been sampled. The second value is the timestamp that is the value of a timebase register at the time of the conversion.

The EQADC result FIFO also has eDMA flags associated with it. One of these, the result FIFO drain flag, can be used to signal to the eDMA that the result FIFO has new data ready to be copied to a queue in system RAM. In the example decimation system, the eDMA copies results data from the result FIFO pop register to a queue in eTPU parameter RAM

The eQADC conversions are configured to be triggered by the rising edge of an eMIOS signal. This signal's generation is dealt with in Section 4.1, "eMIOS Subsystem".

The eQADC has five internal reference voltages that can be converted. When Channels 40 to 43 are specified in the conversion command, a voltage is converted as shown in Table 5. VRH and VRL are externally provided reference voltages.

**Table 5. EQADC Channel Versus Converted Voltage**

| Channel | Converted voltage |
|---|---|
| 40 | VRH |
| 41 | VRL |
| 42 | (VRH-VRL)/2 |
| 43 | 0.75 * (VRH-VRL) |
| 44 | 0.25 * (VRH-VRL) |

In the example decimation system, the command queue consists of a sequence of 64 commands, which each convert one of the five internal eQADC voltages. The 64 commands produce 64 results that produce 16 decimated values when subsequently decimated with a decimation factor of four. In the example decimation system, a test case was used where sequences of the five internal voltages were converted. This meant the decimated values could be tested to ensure the decimation system was behaving as expected. In an application, the channel being converted would be connected to an external sensor.

## 5 Software Source

The software source for this example decimation system is available from Freescale in AN3396SW.zip. To create this build, several other software packages are required. These are AN2864SW (etpu utility functions) and mpc5500r19.zip (This is release 1.6 of mpc5500 device header files; files with a later revision will also be suitable.). The files listed in Section 6, "Build Structure," are the minimum set of files required to create the example decimator system. These software releases are available from www.freescale.com.

## 6 Build Structure

The example application contains the files shown. For the makefiles that come with the software release to work, they must be placed in these directories.

- decimate_software – AN3396SW
- decimate – API to decimate eTPU function and example application code
  - decimator_project.h – definitions used in the application software
  - decimate_etpu_example.c – example application
  - dma_init_for_eqadc_config.c – eDMA initialization for use with eQADC configuration routines.
  - eqadc_init.c – initialization routines for eQADC
  - etpu_decimate.h – eTPU decimation function API header file.
  - etpu_decimate.c – eTPU decimation function API routines.
  - crt0.s – assembly startup file
  - MPC5534_intram.dld – Linker file
  - makefile – makefile
- etpu_decimate_function etpuc sources and image file
  - etpuc.h – Standard include file for eTPU code.
  - etpuc_common.h – Standard include file for eTPU code.
  - etpuc_decimate.c – C source for the eTPU FPM function.
  - etpuc_decimate_function.c – Top level file for decimation function.
  - etpu_decimate_function.h – Image of eTPU code for host CPU (generate by eTPU compiler).
  - etpuc_decimate_function.cod – Debug information for the function.
  - etpuc_decimate_function.lst – Assembler listing file.
  - makefile – A makefile used to build the eTPUC code.
- mpc5500 - mpc5500r19.zip – device header files.
  - mpc5534.h – Register and bit field definitions for MPC5534.
  - mpc5534_vars.h – Variables that define some features of the MPC5534.
  - typedefs.h – Defines all of the data types for the mpc5534 header file.
- utils - AN2864SW – etpu utility functions
  - etpu_struct.h – contains a structure that defines the eTPU module.
  - etpu_util.c – contains the utilities functions.
  - etpu_util.h – This file contains function prototypes and defines for utility functions.
  - makefile – Makefile to build utility functions.
  - readme.txt.

Although this application note refers to MPC5534 and the software build in particular uses the device header file, the example decimation system software provided runs on any MPC553x/MPC555x/MPC556x family member, except for the MPC5561, which has no eTPU.

## 7 Conclusion

Two eDMA channels are used in the example decimation system. One channel is used to supply the eQADC with commands. A second channel is used to move the eQADC results data to eTPU parameter RAM. This second channel is also used to copy the decimated value to a results array in system RAM, write the eTPU host service request register, and copy the results array pointer from eTPU parameter RAM to system RAM. With the eMIOS triggering conversions every 5 us and the system configured to average four eQADC results, 12 percent of the bandwidth of the eDMA is used by this second channel. This bandwidth figure was derived theoretically and was also confirmed using measurements taken using Nexus Trace on MPC5534 silicon.

For comparison, an interrupt driven system was created using the CPU only to perform the same decimation function, including all data moves. The CPU bandwidth consumed by this system was eight percent. The CPU system performed decimation on batches of 200 samples.

This application note has shown the flexible feature set available in MPC5500 devices can be used to completely offload computational data processing tasks such as decimation from the CPU, which frees it up to do other tasks. The versatile eDMA module can be used to create complex chains of data movement between modules using self programming via the scatter gather function.

## 8 Glossary

Here is a list of potentially new terms and definitions.

eTPU – enhanced Time Processing Unit

eMIOS – enhanced Modular I/O Subsystem

eQADC – enhanced Queued analog-to-digital Converter

ADC – analog-to-digital Converter

RAM – Random Access Memory

TCD – Transfer Control Descriptor

CPU – Central Processing unit

OPWFM – Output Pulse Width and Frequency Modulation

PRAM – Parameter RAM

SIU – System Integration Unit

TPU – Time Processing Unit

MDU – Multiply Accumulate and Divide Unit

RISC – Reduced Instruction Set Computer

FIFO – First In First Out

## 9 Revision History

Table 6. Revision History

| Version | Changes |
|---|---|
| Rev. 0 | First public version of this document. |
| Rev. 1 | • In Table 3, TCD Q1 column, changed “Software from A1” to “Software from P1” and “On TCD X1” to “On TCD R1.”<br>• In Table 3, TCD P1 column, changed “On TCD B1” to “On TCD Q1.”<br>• In Table 4, changed “Q1” in the table title to “S1” and the “TCD Q1” column heading to “TCD S1.” |

THIS PAGE IS INTENTIONALLY BLANK

Document Number: AN3396
Rev. 1
02/2011