freescale

Technical Support

MPC5643L

PWM triggered measurement concept

Introduction

The aim of the MPC5643L PWM triggered measurement concept is to introduce hardware subsystem concept of autonomous triggering of ADC measurement by PWM module in desired time intervals and automatic storing of measured data into buffer located in SRAM.

This autonomous measurement concept will offload the microprocessor's core and presents the very precise way how to achieve the ADC time critical measurement synchronized with PWM signal generated by FlexPWM module.

Contents

List of Tables

List of Figures

1 Document Management

Table 1 Revision history

Revision	Date	Author	Changes
0.1	3.7.2014	Peter Vlna	Initial draft
0.2	21.8.2014	Peter Vlna	Fixed header

Table 2 Distribution list

Name	Organization

Table 3 Reference Documents

Document Name	Document Number	Version

1.1 Acronyms

Acronym	Name
ADC	Analog to digital converted
CTU	Cross triggering unit
PWM	Pulse-width modulation
TGS	CTU trigger generator submodule
eDMA	Enhanced Direct Memory Access
TCD	Transfer control descriptor

2 PWM-ADC hardware concept

Hardware concept for PWM signal generation and ADC measurement builds on dedicated peripherals and DMA machine providing their interconnection and result storing mechanism.

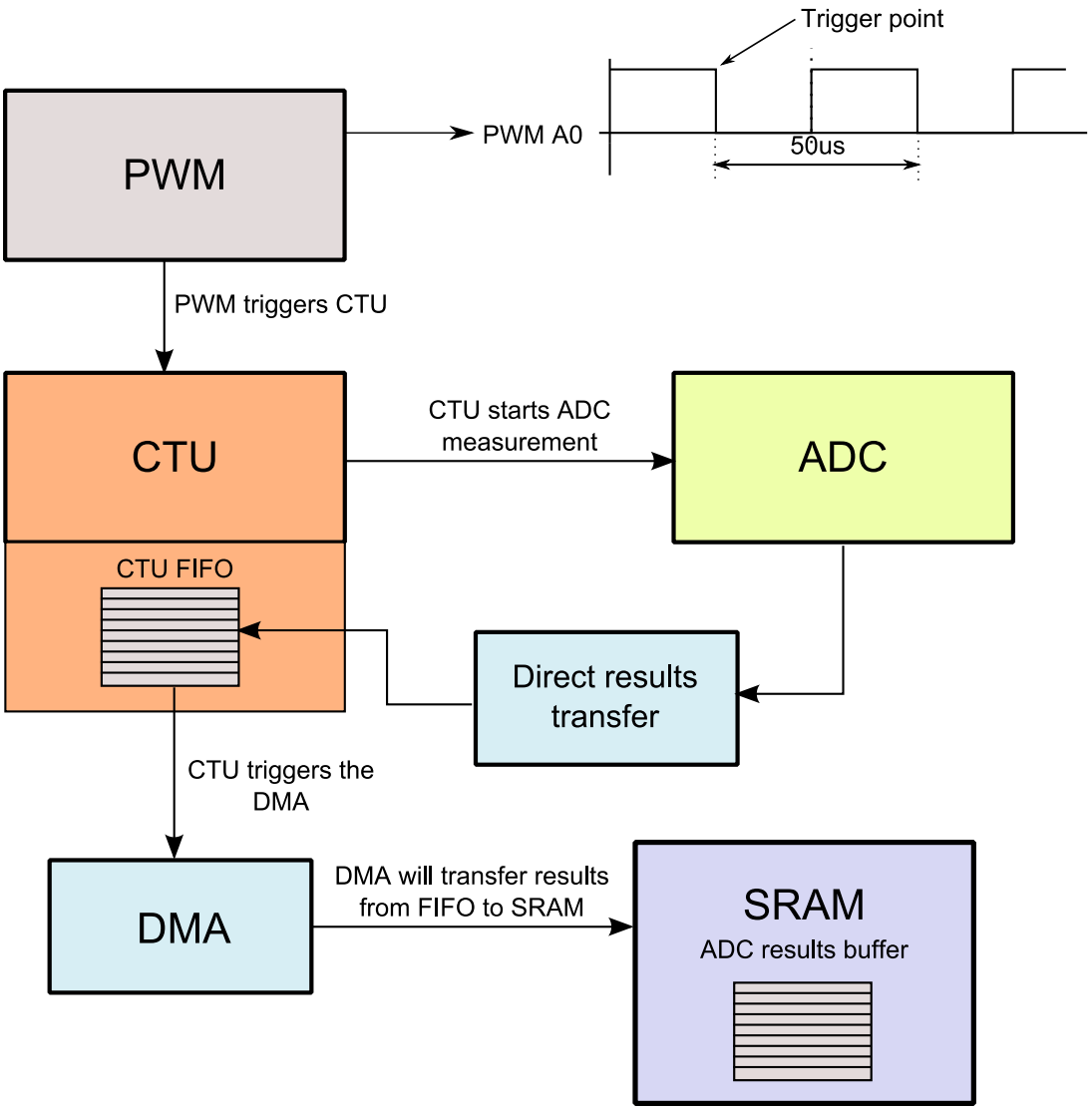


Figure 1 - PWM-ADC HW concept scheme

- FlexPWM
  - One of the FlexPWM modules is used to generate PWM signal. Rising edge of PWM signal provides the trigger point for CTU module as Figure 1 - PWM-ADC HW concept scheme represents.
- CTU
  - Cross Trigger Unit module provides triggers necessary for correct ADC measurement in desired time. CTU scheduler subunit also provides the “command list” for ADC which selects the channels to be converted on particular trigger.
  - ADC results can be stored in the channel relevant standard result register and/or in one of the 4 FIFOs: different FIFOs allow dispatching ADC results according to the type of acquisition. Each FIFO has its own interrupt line and DMA request signal.

- ADC
  - It contains one ADC measurement group of 16 channels. ADC is configured in CTU mode, so CTU can start ADC measurement.
  - Results of ADC conversions are handled by automatically by CTU-ADC system and after ADC conversion they are automatically stored in CTU FIFO.
- eDMA
  - DMA module is used for ADC measurement results transfer from CTU FIFO to desired location in SRAM.

3 PWM

The FlexPWM module is used to generate the trigger point for measurements as presented on Figure 2 - Trigger points for CTU.

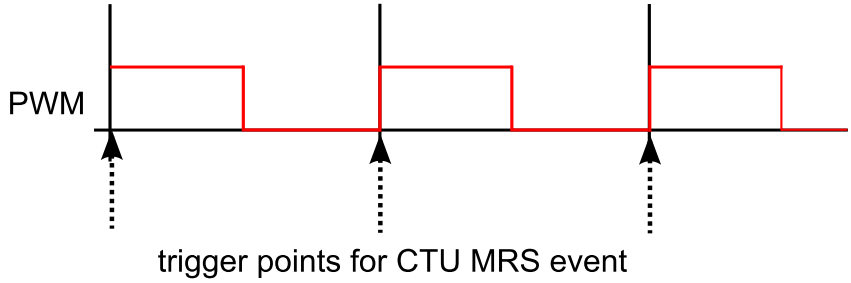


Figure 2 - Trigger points for CTU

The very important is to understand the interconnection between modules.

3.1 Configuration of PWM module

```
void PWM_Init (void)
{
    FLEXPWM 0.OUTEN.R = 0x100;        //PWM0[SUB0] A0 output enable
    FLEXPWM 0.SUB[0].CTRL2.B.CLK SEL = 2;  // PWM0[SUB0] AUX clock
    FLEXPWM_0.SUB[0].CTRL.B.PRSC = 0x0; /* Prescaler 1:1 */
```

First select the clock and prescaler for FlexPWM module as shown above in code example.

```
    FLEXPWM_0.SUB[0].VAL[0].R = 0x0;  /* 0 offset */
    FLEXPWM_0.SUB[0].VAL[1].R = 0x320;  /* 20kHz period */
    FLEXPWM_0.SUB[0].VAL[2].R = 0x0;
    FLEXPWM_0.SUB[0].VAL[3].R = 0x160; /* 50% duty cycle */
```

Configure the PWM module to generate the PWM signal via defining values for VAL[0] – VAL[6] registers.

```
    FLEXPWM_0.SUB[0].CTRL2.B.DBGEN = 0x1; /* Enable debug */
    FLEXPWM_0.SUB[0].CTRL2.B.INDEP = 0x1; /* Independant channels */
    FLEXPWM 0.SUB[0].DISMAP.R = 0; /* Fault dissbale mapping register */
    FLEXPWM_0.SUB[0].TCTRL.R = 0x4; /* Output Trigger Enables (OUT_TRIG_EN2 = 1 )*/
```

In order to be able trigger the CTU we need to select the output trigger (for example OUT_TRIG_EN2 =1). This will generate the trigger on VAL2 compare match.

These bits enable the generation of OUT_TRIG0 and OUT_TRIG1 outputs based on the counter value matching the value in one or more of the VAL0-5 registers. **VAL0, VAL2, and VAL4** are used to generate **OUT_TRIG0** and **VAL1, VAL3, and VAL5** are used to generate **OUT_TRIG1**. The OUT_TRIGx signals are only asserted as long as the counter value matches the VALx value.

Demonstration of interconnection of FlexPWM and CTU modules for our example is shown an Figure 4 - FlexPWM and CTU interconnection.

```
    FLEXPWM_0.MCTRL.B.LDOK = 0x1; /* Load OK */
    FLEXPWM_0.MCTRL.B.RUN = 0x1;  /* Run PWM */
}//PWM_Init
```

When the FlexPWM module is configured we can set load OK bit (LDOK) and start the PWM submodule.

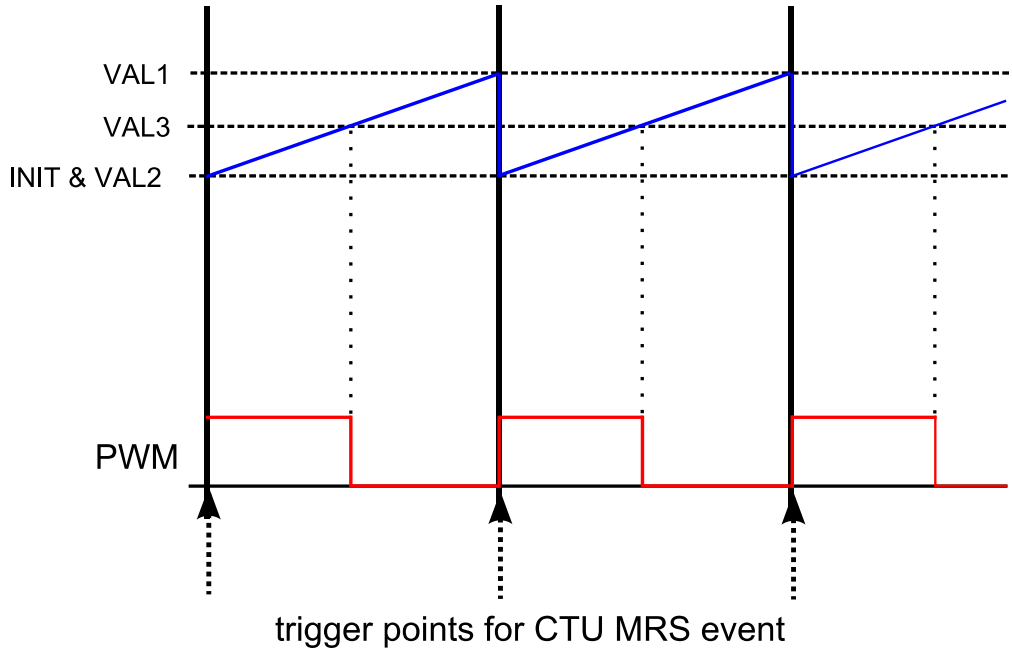


Figure 3 - generation of PWM signal

Interconnection between CTU and FlexPWM module are presented on picture below. Understanding of this interconnection is essential for correct cooperation of modules.

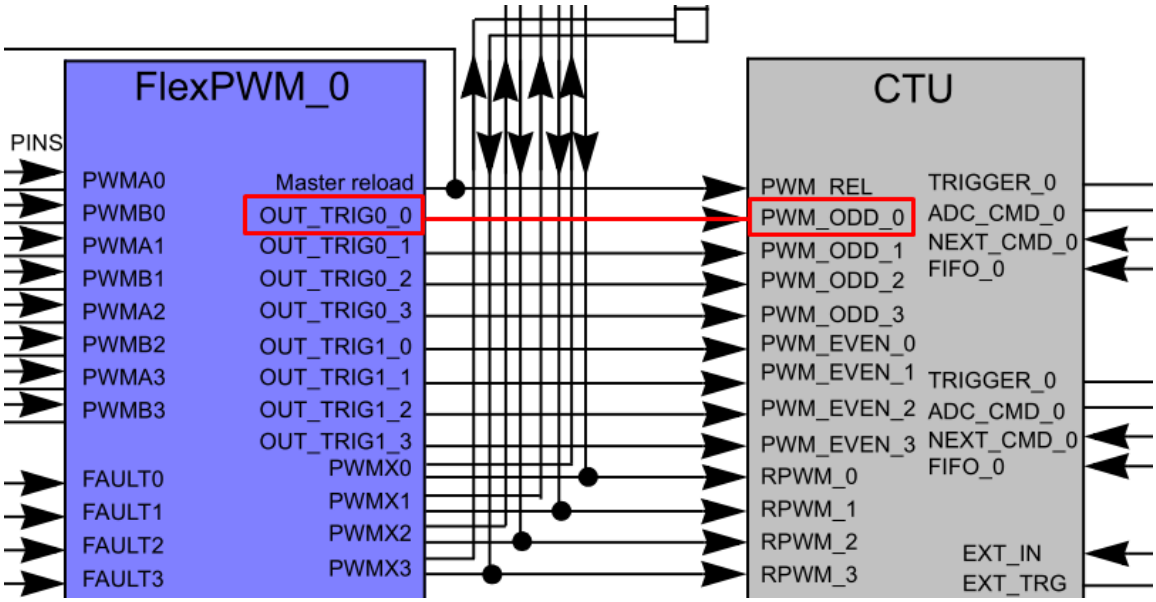


Figure 4 - FlexPWM and CTU interconnection

On compare of VAL2 match the FlexPWM will generate OUT_TRIG0_0 (for submodule 0 and VAL2 compare). This output trigger is necessary to connect to CTU module.

4 CTU

In PWM driven systems it is important to schedule the acquisition of the state variables with respect to PWM cycle.

Cross triggering unit is used to start ADC conversion in desired time intervals. It is separately connected to ADC and PWM to avoid any time delays during communication. It also applies direct channel for ADC measured data automatic transition into CTU FIFO buffers.

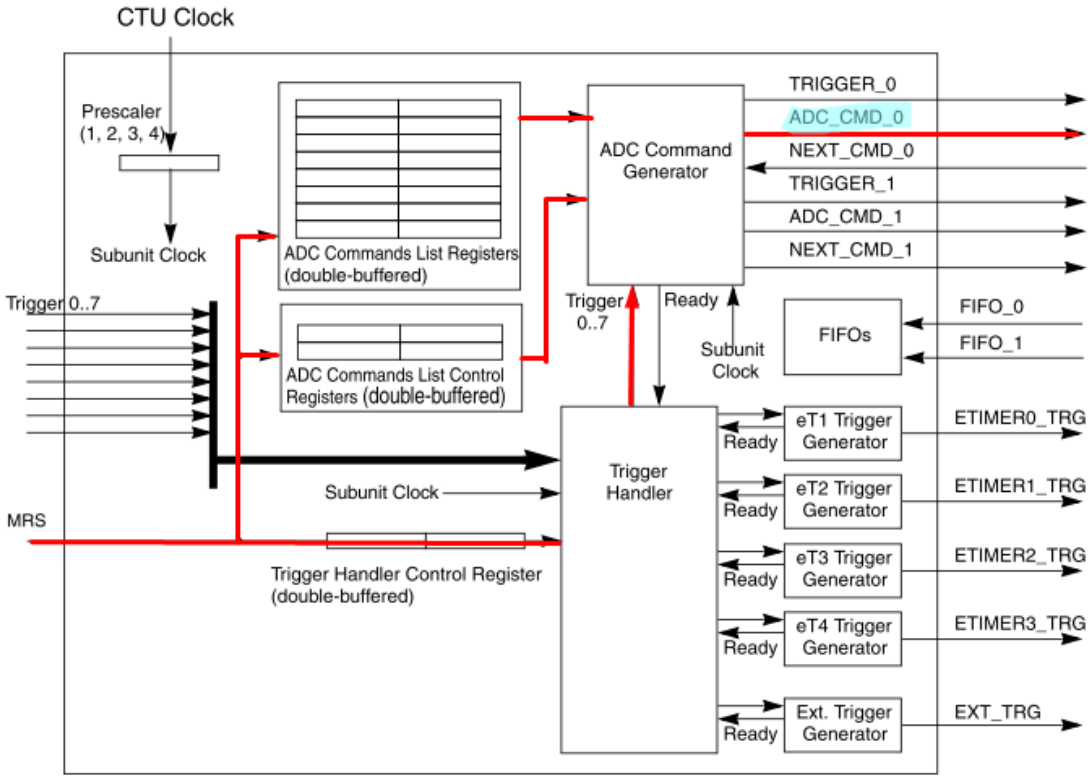
In previous chapter we defined that FlexPWM module will generate OUT_TRIG0_0 signal to CTU. On CTU side this signal is connected to PWM_ODD_0 input as shown on Figure 4 - FlexPWM and CTU interconnection.

Now it is necessary to select if we are going to use rising of falling edge of PWM as a trigger. This selection is done in Trigger generator subunit input selection register (TGSISR).

28 I1_FE	Input 1 PWM ch.0 odd Falling edge Enable (0: disabled - 1: enabled)
29 I1_RE	Input 1 PWM ch.0 odd Rising edge Enable (0: disabled - 1: enabled)
30 I0_FE	Input 0 PWM Reload Falling edge Enable (0: disabled - 1: enabled)
31 I0_RE	Input 0 PWM Reload Rising edge Enable (0: disabled - 1: enabled)

Figure 5 - Trigger generator subunit input selection register (TGSISR)

For our example we will select the **I1_RE = 1** which stands for PWM ch.0 odd Rising edge Enable. This means that on PWM VAL2 compare (which is in time 0 of PWM period) the CTU will receive the input trigger from FlexPWM module. On this input trigger CTU will start its internal counter (CTU clock).



As soon as the counter reach the value in trigger handler submodule compare registers TxCR (x = 0,1,..,7) the trigger even is generated and passed to CTU scheduler submodule.

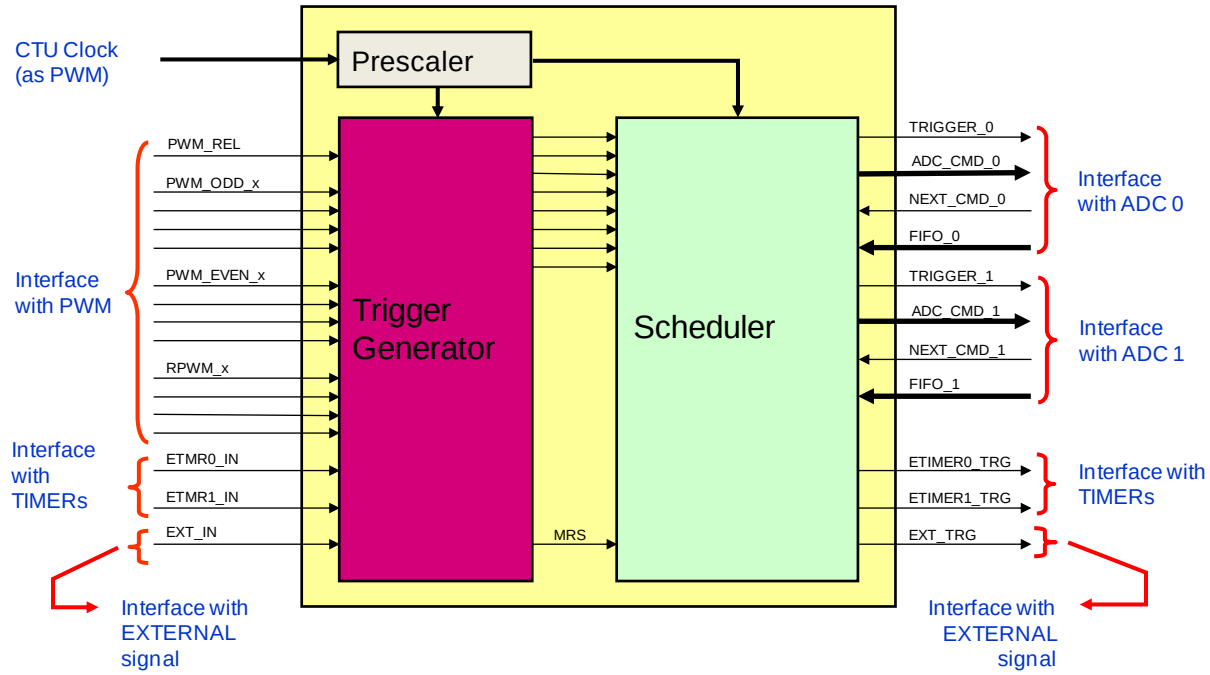


Figure 6 - CTU internal scheme

Number of commands is controlled by first command bit of the successive command sequence.

4.1 Configuration of CTU module

```
void CTU Init (void)
{
```

Select the input trigger source (master reload interval) for CTU module and clock for CTU module.

```
  CTU.TGSISR.B.I1_RE = 0x1;    /* Select the MRS for CTU - PWM reload - period 50usec */
  CTU.CTUCR.B.TGSISRRE = 0x1; /* TGS Input Selection Register Reload Enable */
  CTU.TGSCR.B.PRES = 0x0;      /* CTU prescaler is set to 1 */
```

Fill CTU trigger compare registers. (In our case only 1 register (T0CR) is filled with value 1). As soon as counter reaches value 1, the time trigger from TGS (trigger generator submodule) is linked to one ADC command sequence trigger. ADC command sequence trigger points to the first command in the command list.

```
  CTU.TCR[0].R = 0x1;          /* CTU Timer compare register 0 value */
```

Counter will stop and wait for reload trigger from PWM at value 0xFFFF.

```
  CTU.TGSCCR.R = 0xFFFF;       /* TGS Counter Compare Register */
```

Counter is starts to count from 0 after reload.

```
  CTU.TGSCRR.R = 0;            /* TGS Counter Reload Register */
```

Enables Trigger 0 and enables ADC command output.

```
  CTU.THCR1.B.T0E = 1;         /* Trigger 0 output enable */
  CTU.THCR1.B.T0ADCE = 1;      /* Trigger 0 ADC Command output enable */

  CTU.CLCR1.B.T0INDEX = 0;     /* Trigger 0 command list - first command address */
```

FIFO DMA control register (FDCR) setting.

```
  (*(volatile uint16 t *)0xFFE0C06C) = 1; /* CTU.FDCR = 1  - ebable DMA on FIFO (missing in
header file!)*/
  CTU.FCR.R = 0;               /* FIFO control register */
```

The example of ADC command sequence list is presented on Figure 7 - CTU ADC command sequence list. Select the ADC module and channels to be converted as shown below in example code. End the command list with the additional command with value 0x4000 to stop execution of next commands.

```
  /* CTU triggers ADC -  ADC command list */
  /* SU : ADC command list */
  CTU.CLR[0].R = 0x0;       /* ADC_0 module, channel 0 First command - CTU starts sending
commands to ADC after this command */
  CTU.CLR[1].R = 0x1;       /* ADC_0 module, channel 1 */
  CTU.CLR[2].R = 0x2;       /* ADC_0 module, channel 2 */
  CTU.CLR[3].R = 0x3;       /* ADC_0 module, channel 3 */
  CTU.CLR[4].R = 0x4;       /* ADC_0 module, channel 4 */
  CTU.CLR[5].R = 0x5;       /* ADC_0 module, channel 5 */
  CTU.CLR[6].R = 0x6;       /* ADC_0 module, channel 6 */
  CTU.CLR[7].R = 0x7;       /* ADC_0 module, channel 7 */
  CTU.CLR[8].R = 0x8;       /* ADC_0 module, channel 8 */
  CTU.CLR[9].R = 0x9;       /* ADC_0 module, channel 9 */
  CTU.CLR[10].R = 0xA;      /* ADC_0 module, channel 10 */
  CTU.CLR[11].R = 0xB;      /* ADC_0 module, channel 11 */
  CTU.CLR[12].R = 0xC;      /* ADC_0 module, channel 12 */
  CTU.CLR[13].R = 0xD;      /* ADC_0 module, channel 13 */
  CTU.CLR[14].R = 0xE;      /* ADC_0 module, channel 14 */
  CTU.CLR[15].R = 0xF;      /* ADC_0 module, channel 15 */
  CTU.CLR[16].R = 0x4000;   /* First command - this command is not send to ADC */
```

Set FIFO threshold for 0xE so the eDMA transfer will be triggered automatically when FIFO reached depth of 15, measured results will be transferred automatically by eDMA state machine.

```
  CTU.TH1.B.THRESHOLD0 = 0xE;        /* FIFO 0 Threshold. Maximum value of 15, as the threshold
value must be less than the number of FIFO 0 entries. */
```

In case the interrupt on FIFO overflow is needed enable also FIFO overflow interrupt in FCR register.

```
  CTU.FCR.B.FIFO_OVERFLOW_EN0 = 1;  /* FIFO 0 threshold Overflow interrupt enable - read FIFO 0
in this interrupt */
```

In order to guarantee the coherency, the reload of all double-buffered registers is enabled by setting GRE (General Reload Enable) bit in the CTU Control Register. The user must ensure that all intended double-buffered registers are updated before a new MRS occurrence.

```
  CTU.CTUCR.B.GRE = 1;              /* General Reload Enable */
}//CTU_Init
```

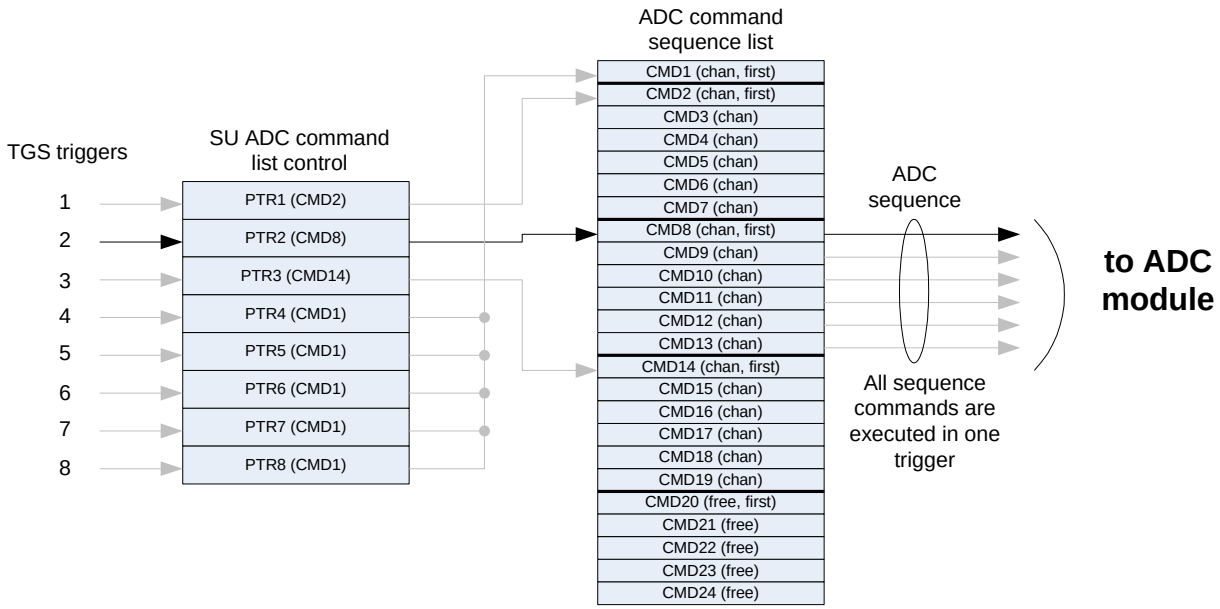


Figure 7 - CTU ADC command sequence list

5 eDMA

The eDMA takes care of transferring measured data stored in CTU FIFO into result buffer located in SRAM. This is done automatically as soon as CTU FIFO reaches its threshold.

5.1 Configuration of eDMA module

5.1.1 Configuration of eDMA module

The DMAERQL register provides a bit map for the implemented channels to enable the request signal for each channel. In our case we need to enable request from CTU FIFO_0 threshold overflow.

```
void DMA Init (void)
{
    //EDMA.CR.R   = 0x00000002; /* Enable debug mode */
    EDMA.EEIRL.R   = 0x0000; /* Error Interrupt disabled for all channels  */
    EDMA.ERQRL.R = 0x1;  /* Enable eDMA request 0 -> CTU FIFO 0 trigger */
}//DMA_Init
```

5.1.2 Configuration of eDMA TCD

```
void DMA TCD 0(void)
{
    (*(volatile uint16_t *)0xFFF4501C) = 0x0; //Clear DMA.TCD[0]0.WORD7
```

Select the source address for transfer. The data will be transferred from CTU FIFO_0.

```
    EDMA.TCD[0].SADDR = 0xFFE0C080;  /* Source Address - CTU FIFO_0 */
```

Select destination address where the data from CTU FIFO_0 will be transferred.

```
    EDMA.TCD[0].DADDR = (uint32 t) &ADC results[0];  /* Destination Address - SRAM */
    EDMA.TCD[0].SMOD = 0x0;   /* Source Address Modulo      */
    EDMA.TCD[0].DMOD = 0x0;   /* Destination Address Modulo */
    EDMA.TCD[0].SSIZE = 0x2;  /* Source Transfer Size: 32 bits*/
    EDMA.TCD[0].DSIZE = 0x2;  /* Destination Transfer Size: 32 bits*/
```

Read always from the same source address, because of FIFO buffer.

```
    EDMA.TCD[0].SOFF =  0x0;  /* Signed Source Addr Offset adjustment*/
```

Transfer 64 bytes in minor loop.

```
    EDMA.TCD[0].NBYTES = 0x40; /* Inner 'minor' byte count   */
    EDMA.TCD[0].SLAST = 0x0;  /* Last signed Source Address Adjust */
```

Set destination address offset to 4 bytes.

```
    EDMA.TCD[0].DOFF = 0x4;   /* Signed Destination Address Offset  */
```

After major loop completion decrement destination address by 64 bytes. This will set the destination address to beginning of the result buffer.

```
    EDMA.TCD[0].DLAST SGA = 0xFFFFFFC0; /* Signed Destination Address Offset -64  */
    EDMA.TCD[0].BITERE LINK  = 0x0;  /* Channel-to-channel linking on Minor Loop Complete:
Disabled*/
```

Used is one major loop.

```
    EDMA.TCD[0].BITER  = 0x1;  /* Current Major Iteration Count or Link Channel Number  */
    EDMA.TCD[0].CITERE_LINK  = 0x0;  /* Channel-to-channel linking on Minor Loop Complete:
Disabled*/
    EDMA.TCD[0].CITERLINKCH  = 0x0;  /* Channel Number for Channel-to-Channel Linking on Minor
Loop : Not Complete*/
    EDMA.TCD[0].CITER  = 0x0001;  /* Current Major Iteration Count or Link Channel Number  */
    EDMA.TCD[0].BWC  = 0x00;  /* Bandwidth control */
    EDMA.TCD[0].MAJORLINKCH = 0x00;  /* Major Channel number */
    EDMA.TCD[0].MAJORE_LINK  = 0x00;  /*Channel-to-channel Linking on Major Loop Complete:
Disabled*/
    EDMA.TCD[0].DONE  = 0x00;  /* Channel Done bit */
    EDMA.TCD[0].ACTIVE = 0x00;  /* Channel Active bit */
    EDMA.TCD[0].E_SG  = 0x00;  /* Enable Scatter/Gather: Disabled*/
    EDMA.TCD[0].D_REQ   = 0x00;  /* Disable the DMA channel when Done: Disabled*/
    EDMA.TCD[0].INT_HALF   = 0x00;  /* Interrupt on Half Major Count completion: Disabled*/
    EDMA.TCD[0].INT MAJ = 0x01;  /* Interrupt on major loop completion: Disabled*/
    EDMA.TCD[0].START = 0x00;  /* Explicit Channel Start bit  */
}//eDMA
```

6 eDMA Channel Mux (DMA_MUX)

The DMA_MUX allows to route 27 DMA peripheral sources (called slots) to 16 DMA channels.

The main configuration register is Channel Configuration Registers (CHCONFIG#n). Here it is possible to configure hardware trigger source for eDMA module. SOURCE specifies which DMA source, if any, is routed to a particular eDMA channel. In our case it is source FIFO1 which stands for CTU FIFO_0 eDMA trigger.

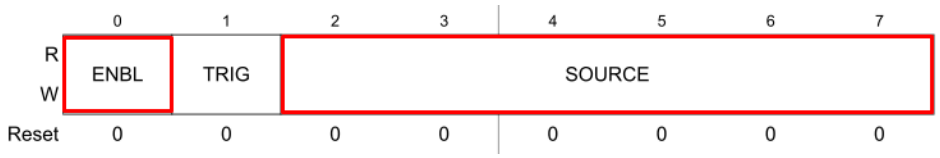


Figure 8 - Channel Configuration Registers (CHCONFIG#n)

DMA_MUX source slot #	Source module	Source resource
1	DSPI_0	DSPI_TFFF
2	DSPI_0	DSPI_RFDF
3	DSPI_1	DSPI_TFFF
4	DSPI_1	DSPI_RFDF
5	DSPI_2	DSPI_TFFF
6	DSPI_2	DSPI_RFDF
7	CTU	CTU
8	CTU	FIFO1
9	CTU	FIFO2
10	CTU	FIFO3
11	CTU	FIFO4
12	FlexPWM_0	comp_val
13	FlexPWM_0	capt
14	eTimer_0	Channel 0
15	eTimer_0	Channel 1
16	eTimer_1	Channel 0
17	eTimer_1	Channel 1
18	eTimer_2	Channel 0
19	eTimer_2	Channel 1
20	ADC_0	DMA
21	ADC_1	DMA
22	LINFlexD_0	Transmit
23	LINFlexD_0	Receive

Figure 9 - DMA_MUX source slot mapping (1 of 2)

DMA_MUX source slot #	Source module	Source resource
24	LINFlexD_1	Transmit
25	LINFlexD_1	Receive
26	FlexPWM_1	comp_val
27	FlexPWM_1	capt
28	Always Requestor	-
29	Always Requestor	-
30	Always Requestor	-
31	Always Requestor	-
32	Always Requestor	-
33	Always Requestor	-

Figure 10 - DMA_MUX source slot mapping (2 of 2)

6.1 Configuration of eDMA MUX

```
void DMA_MUX_Init(void)
{
```

First clear the channel configuration register.

```
    DMAMUX.CHCONFIG[0].R = 0x0000; /* Clear channel config register 0*/
```

Select the trigger source.

```
    DMAMUX.CHCONFIG[0].B.SOURCE = 8; /* Select MUX source for channel 0 -> CTU FIFO_1 */
```

Enable eDMA MUX channel.

```
    DMAMUX.CHCONFIG[0].B.ENBL = 1; /* DMA Channel Enable */

}//DMA_MUX_Init
```

7 Hints

- Do not forget to configure peripheral bridge (AIPS) to allow eDMA (master) access to CTU FIFO buffer.
- Ensure that whole CTU FIFO is read. To prevent results shifting inside results buffer in SRAM.s