Freescale Semiconductor
Application Note

Document Number: AN4266
Rev. 1, May 2011

Using eTPU2 and eQADC for PWM Control on the MPC563xM

A PWM-Controlled LED Dimmer

by: Mong Sim
Applications Engineering
Austin, Texas
USA

Contents

1 Introduction

The MPC563xM is a low-cost, 32-Bit Qorivva microcontroller with a Power Architecture® core intended primarily for low-end engine management, such as a 4-cylinder gasoline-powered engine. In addition to the e200z335 Power Architecture core, the MPC563xM devices include an independent timing processor: the enhanced Timing Processor (eTPU2). Freescale offers a number of eTPU2 software drivers for various functions (available at www.freescale.com/etpu). The eTPU2 is typically used to control the timing of the spark and fuel type functions for the engine.

This application note uses the eTPU2 PWM driver from the eTPU2 general function set (Set 1, a set of eTPU(2) functions taken from Freescale application note AN2865, "MPC5500 & MPC5600 Simple Cookbook") to demonstrate an open-loop LED dimmer control system. The system, as shown in Figure 1, consists of:

- Input device (trim potentiometer)
- Acquisition device (enhanced analog-to-digital converter)
- Actuator (eTPU2)
- Output device (LED)

This example application combines two separate examples from AN2865: the eTPU2 PWM demo and the eQADC single software scan demo. This combined example measures the voltage on the eQADC input pin and translates that to a PWM duty cycle that is then sent to the eTPU2 PWM driver.

freescale semiconductor

In this example application the eQADC measures the voltage output from the accelerator and changes the current to the throttle control by varying the PWM duty cycle using the eTPU2, effectively changing the speed. This is just one of the many applications that the eQADC and eTPU2 can achieve together.

This example is designed for the low cost TRK-MPC5634M board (TRK), but can also be run on the MPC5634MKIT (EVB) with expanded capability. In the following sections, I will describe in detail the three main blocks that drive this application.

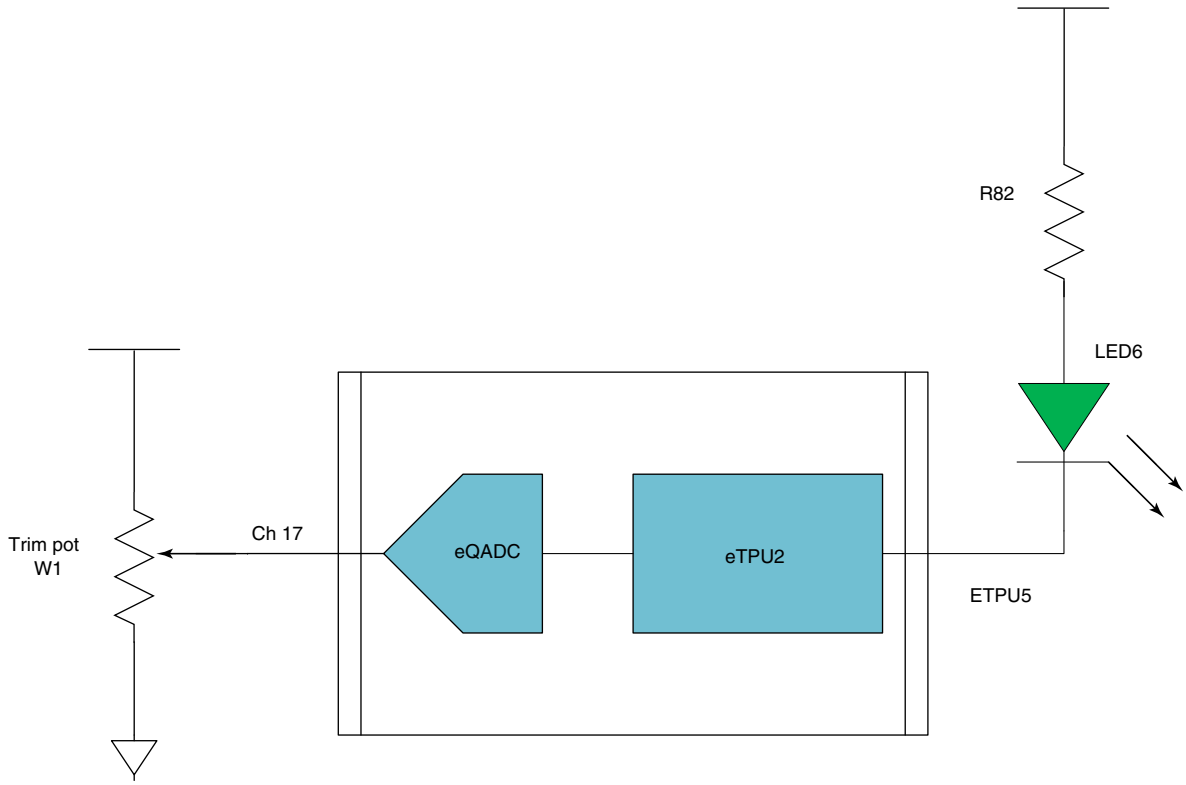

Figure 1. PWM-controlled LED dimmer

2 PLL: Initializing system clock — enhanced mode

2.1 Description

In this section, we will take a quick look at the PLL block diagram and the sub-blocks that are associated with the PLL. We will program the PLL divider and multiplier to achieve the desired operating frequency for the system clock, using the enhanced mode formula of the PLL. Figure 2 shows the block diagram of the PLL and how the different sub-logic blocks interact with one another.

Since both the MPC563xM family EVB and TRK boards come with an 8 MHz crystal, we will use the crystal as our input clock source to the PLL. We will program a 64 MHz system clock operating frequency.

The MPC563xM family also provides an output to verify the system clock frequency (at a clock rate that has been reduced by division) via the EMIOS channel 12. Please refer to Table 1 for more information on the EMIOS [12] clock signal. For a code example, please see the PLL-sysclk project in directory 563xM-CW in AN2865, "MPC5500 & MPC5600 Simple Cookbook."

Freescale recommends that new applications use the PLL in enhanced mode. The non-enhanced PLL mode is also available for backward compatibility.

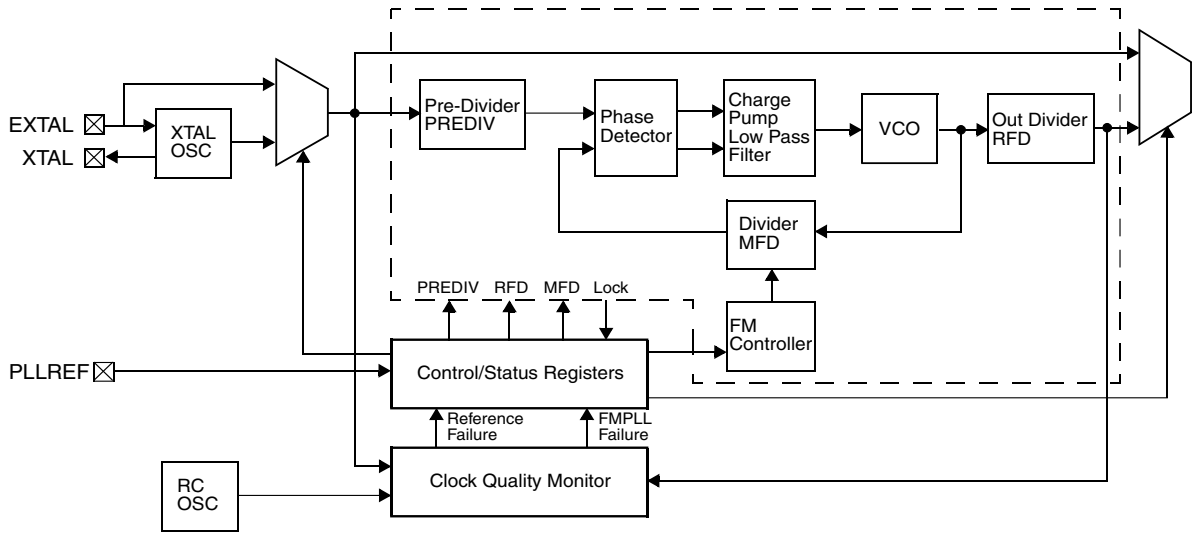

Figure 2. PLL enhanced mode example block diagram

Table 1. EMIOS Channel 12 Configuration

Signal	MPC563xM family				
	SIU PCR (values in hexadecimal)	Package pin number			header pin
		208 BGA	176 LQFP	144 LQFP	
EMIOS[12]	PCR191=0x60C	N10	76	63	EVB-PJ8-8
					TRK/P5 EMIOS12

2.2 Design

2.2.1 Design notes

In this example, after initializing the PLL predivider, divider, and multiplier, the PLL LOCK is tested by simply polling for the desired status to occur. In a real system, a maximum timeout mechanism would be implemented. If lock is not achieved in a maximum time, then an error message can be logged and the part can be reset.

Loss of lock and loss of clock detection are not enabled in this example. Because changing the predivider or multiplier can cause loss of lock, the loss-of-lock circuitry and interrupts would not normally be enabled until after these steps are executed.

In devices that do not exit reset with PLL enabled as system clock, it is good practice to initialize PLL registers with the multiplier and dividers before turning on the oscillator. Otherwise the reset default values may try to force the PLL to run beyond its frequency specifications.

Attempting to increase the frequency in one step may cause overshoot of the PLL beyond the maximum sysclk specification and could cause a brief sharp increase in current demand from the power supply. Therefore two frequency increases are commonly used. The second increase only changes the divider after the PLL feedback loop, which does not cause loss of lock because this divider only affects the feedback path.

2.2.2 Clock configuration pin and register control bits

In this application, we will configure the PLL in enhanced mode by programming the ESYNCR1 and ESYNCR2 registers. We will configure the clock configuration (CLKCFG) bit field of ESYNCR1 in normal mode, with crystal reference. Please refer to the Enhanced Synthesizer Control Register section in the reference manual for detailed information.

The CLKCFG 3-bit field (see the Modes of Operation section in the PLL chapter of the reference manual) is used to change the operating mode of the PLL. Bit 2 is not writable to zero while bit 1 is set. The reset state of bit 3 is determined by the state of the PLLREF pin. For our application example, PLLREF must be set to logic '1' to achieve normal mode operation with crystal reference. Table 2 lists the values of the CLKCFG bit field that are determined based on the state of the PLLREF pin.

Table 2. Clock configuration

Device	Pin	Purpose	Clock configuration bit affected by pin at reset	Clock configuration: reset value	Clock configuration: common value
MPC563xM	PLLREF	Determines if clock uses crystal or external reference	FMPLL_ESYNCR1 [CLKCFG2]	FMPLL_ESYNCR1 [CLKCFG] = 0b01x (Bypass mode with crystal or external reference)	FMPLL_ESYNCR1 [CLKCFG] = 0b111 (Normal mode with crystal reference)

2.2.3 PLL to system clock to peripheral clocks

The PLL logic provides the ability to divide and to multiply when configuring the PLL output and system clock to the desired operating frequency. Figure 2 shows the connections in the PLL block. Using the dividers and the multiplier with the formula as shown in Enhanced PLL calculations — MPC563xM will enable us to calculate the desired system clock frequency for our application.

2.2.4 Enhanced PLL calculations — MPC563xM

The system clock frequency in PLL enhanced mode is calculated using the formula shown in Figure 3.

$$F_{sys} = F_{ref} \times \frac{EMFD}{(EPrediv+1) \times 2^{(ERFD+1)}}$$

Figure 3. Equation for system clock frequency in PLL enhanced mode

In this example, we will set our system clock frequency to 64 MHz with these divider and multiplier settings:

Table 3. PLL divider and multiplier settings

EMFD	EPREDIV	ERFD
64	0	8

$$\frac{F_{sys}}{F_{ref}} = \frac{64MHz}{8MHz} = \frac{EMFD}{(EPrediv+1) \times 2^{(ERFD+1)}} = \frac{64}{(0+1) \times 8}$$

Figure 4. Setting the PLL to 64 Mhz (example)

Figure 4 illustrates an example of using an 8 MHz crystal frequency (F_{ref}) to generate a target frequency of 64 MHz (F_{sys}), using the divider and multiplier settings of Table 3.

2.3 Code

Here is the code for configuring the PLL in enhanced mode with an external crystal clock reference of 8 MHhz to produce a 64 MHz system clock frequency.

```
#include "mpc563m.h" /* Used for MPC563m devices */

...

FMPLL.ESYNCR2.R = 0x00000002;       // ERFD = 2: divides Fpll by 8 (2**(ERFD+1))
FMPLL.ESYNCR1.R = 0xF0000000 + 40; // EPREDIV = 0: divides Fxtal by 1 (EPREDIV+1)
                                    // EMFD = 64: multiply EPrediv = 1
                                    // CLKCFG = 7 for normal mode with crystal ref
                                    // EMODE = 1 to enable enhanced mode

while (!FMPLL.SYNSR.B.LOCK){}       // Wait for PLL to lock

FMPLL.ESYNCR1.R = 0xF0000000 + 64; // Do it in two steps, 40Mhz then 64Mhz
...
```

3 eQADC single software scan

3.1 Description

In this example application, we will use the eQADC analog channel 17 to sample the trim pot potential difference (the trim pot is hardwired to channel 17). We will configure the ADC clock to 2 MHz using the ADC clock prescaler. Command FIFO 0 and Read FIFO 0 are used in this application to send commands and retrieve data from the eQADC. ADC 0 is used in the example and will be configured in single-scan software trigger mode. Figure 5 provides a basic idea of the eQADC signal flow for the example.

Other features of the eQADC such as DMA, Queue, Interrupts, Calibration, etc., are not incorporated in this example.

Full accuracy is not possible because calibration is not implemented in this example. For more information, see Freescale application note AN2989, "Design, Accuracy, and Calibration of Analog to Digital Converters on the MPC5500 Family."

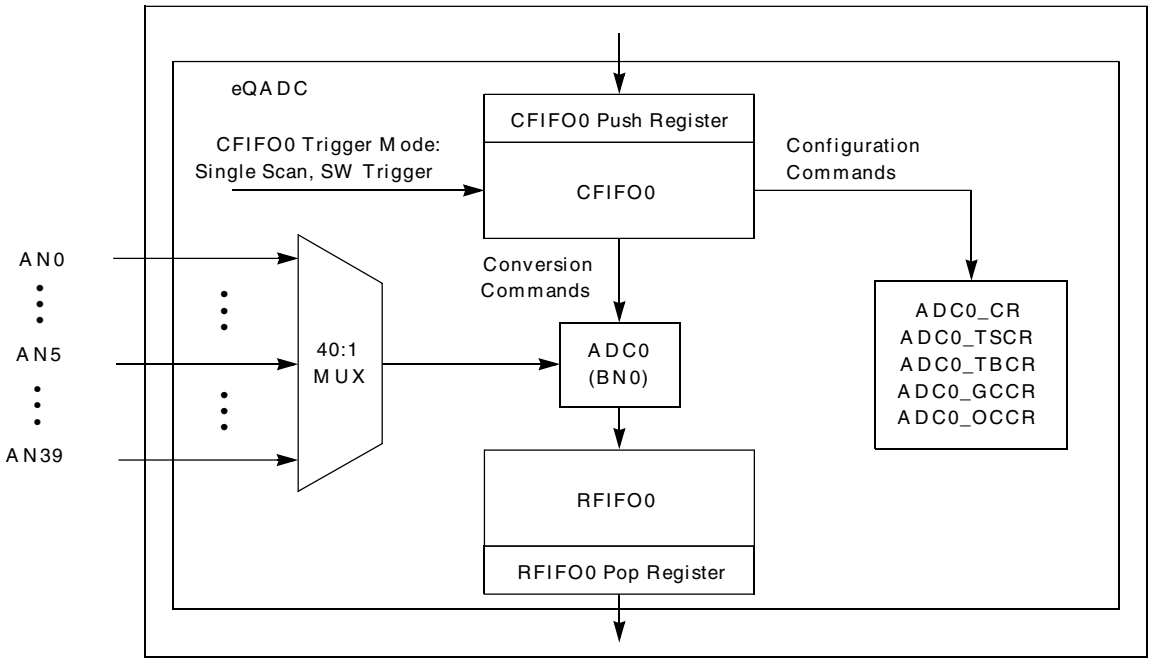

Figure 5. eQADC single software trigger example

3.2 Design

The system clock is configured to 64 MHz. Since the ADC clock frequency must not exceed 15 MHz, we will configure the ADC clock to 2 MHz using the system clock divide factor for ADC clock.

To achieve a 2 MHz ADC clock, we need to configure the ADC clock prescaler to 32 and the ADC ODD prescaler to 0 (64 MHz/32 = 2 MHz). Table 4 provides steps on how to setup the eQADC to perform a single software scan.

Table 4. eQADC single software scan

Step		Relevant bit fields	Pseudocode
Initialize ADC0	Determine Control Reg. value for ADC0: • Enable ADC0 • Prescaler = 32 ADC0 Control Reg = 0x800F	• ADC0_EN=1 • ADC0_CLK_PS = 0x0F (div 32)	—
	Send one write configuration command to CFIFO0: • End of queue (only sending one message here) • Select ADC0 (buffer number 0)	• EOQ = 1 • BN = 0	EQADC_CFPR[0] = 0x8080_0F01
	Configuration command is Write (not Read) ADC Control register value = 0x8001 ADC Control register address = 0x1	R/W = 0 (write) ADC_REGISTER = 0x8001 ADC_REG_ADDRESS = 1	
	Trigger CFIFO0 using single scan software mode (Send configuration command(s) to ADC0's registers)	MODE0 = 1, SSE0 = 1	EQADC_CFCR[0] = 0x0410
	Wait for end of queue flag for CFIFO0	wait for EOQF0 = 1	wait EQADC_FISR[EOQ] = 1
	Clear end of queue flag for CFIFO0	EOQF = 1	EQADC_FISR[EOQ] = 1
Send conversion command	Send one conversion command to CFIFO0: • Convert channel 5 • Use result FIFO0 • Use ADC0 (BN0) • Format is unsigned • Set EOQ	• CHANNEL_NUMBER = 5 • MESSAGE_TAG = 0 • BN = 0 • FMT = 0 • EOQ = 1	EQADC_CRPR[0] = 0x8000_0500
	Trigger CFIFO0 using single-scan software mode (Sends conversion command(s) to ADC 0)	MODE0 = 1, SSE0 = 1	EQADC_CFCR[0] = 0x0410
Read result	Wait for RFIFO0 drain flag to set	wait for RFDF0 = 1	Wait EQADC_FISR[RFDF]=1
	Read result from Result FIFO Pop register 0		read EQADC_RFPR[0]
	Clear flags for any subsequent use. (Note: Flags are cleared by writing a 1. Code here is for illustrative purposes, but actually causes all flags in the FISR register to clear because the compiler will read the current value from the register, OR in the "1", and write back the new value. Therefore existing flags at 1 are cleared. The proper way to clear a flag is to write to the entire register.		EQADC_FISR [RFDF, EOQF] = 1

3.3 Code

Here is the eQADC initialization code to initialize the eQADC for a single software trigger mode using the ADC0, plus the eQADC Read function to retrieve the result for the eQADC.

```
#include "mpc563m.h" /* Use proper include file such as mpc5510.h or mpc5554.h */

  static uint32_t Result = 0;         /* ADC conversion result  */
  static uint32_t ResultInMv = 0;     /* ADC conversion result in millivolts */

void initADC0(void) {
  EQADC.CFPR[0].R = 0x80800F01;       /* Send CFIFO 0 a ADC0 configuration command */
                                      /* enable ADC0 & sets prescaler= divide by 32*/
  EQADC.CFCR[0].R = 0x0410;           /* Trigger CFIFO 0 using Single Scan SW mode */
  while (EQADC.FISR[0].B.EOQF !=1) {} /* Wait for End Of Queue flag */
  EQADC.FISR[0].B.EOQF = 1;           /* Clear End Of Queue flag */
}

void SendConvCmd (void) {
  EQADC.CFPR[0].R = 0x80000500; /* Conversion command: convert channel 5 */
                                /* with ADC0, set EOQ, and send result to RFIFO 0*/
  EQADC.CFCR[0].R = 0x0410;     /* Trigger CFIFO 0 using Single Scan SW mode */
}

void ReadResult(void) {
  while (EQADC.FISR[0].B.RFDF != 1){}      /* Wait for RFIFO 0's Drain Flag to set*/
  Result = EQADC.RFPR[0].R;                       /* ADC result */
  ResultInMv = (uint32_t)((5000*Result)/0x3FFC);  /* ADC result in millivolts */
  EQADC.FISR[0].B.RFDF = 1;                /* Clear RFIFO 0's Drain Flag */
  EQADC.FISR[0].B.EOQF = 1;                /* Clear CFIFO's End of Queue flag */
  }

...

  initADC0();                   /* Enable ADC0 only on eQADC */
  SendConvCmd();                /* Send one conversion command */
  ReadResult();                 /* Read result from Trim Pot at Channel 17*/

...
```

4 eTPU2 PWM example

4.1 Description

This application uses the Set 1 eTPU2 functions from the Freescale website, www.freescale.com/etpu to build an image that can be loaded to the eTPU2 RAM by the CPU.

In this application, only the eTPU2 PWM function on eTPU2_A channel 5 is used to control the LED brightness. This channel is initialized at 1 kHz with a 25% duty cycle. The PWM duty cycle value is updated based on the trim pot value acquired by the eQADC. Figure 6 illustrates the signal flow in the eTPU2 block and shows an example PWM output (ETPUA5) from the eTPU2 block. The ETPUA5 is an external pin that needs to be configured to produce the desired PWM output. Table 5 provides header pin assignment for ETPUA5 for the TRK and EVB boards.

For further information, see Freescale application note AN2864, "General C Functions for the eTPU2."

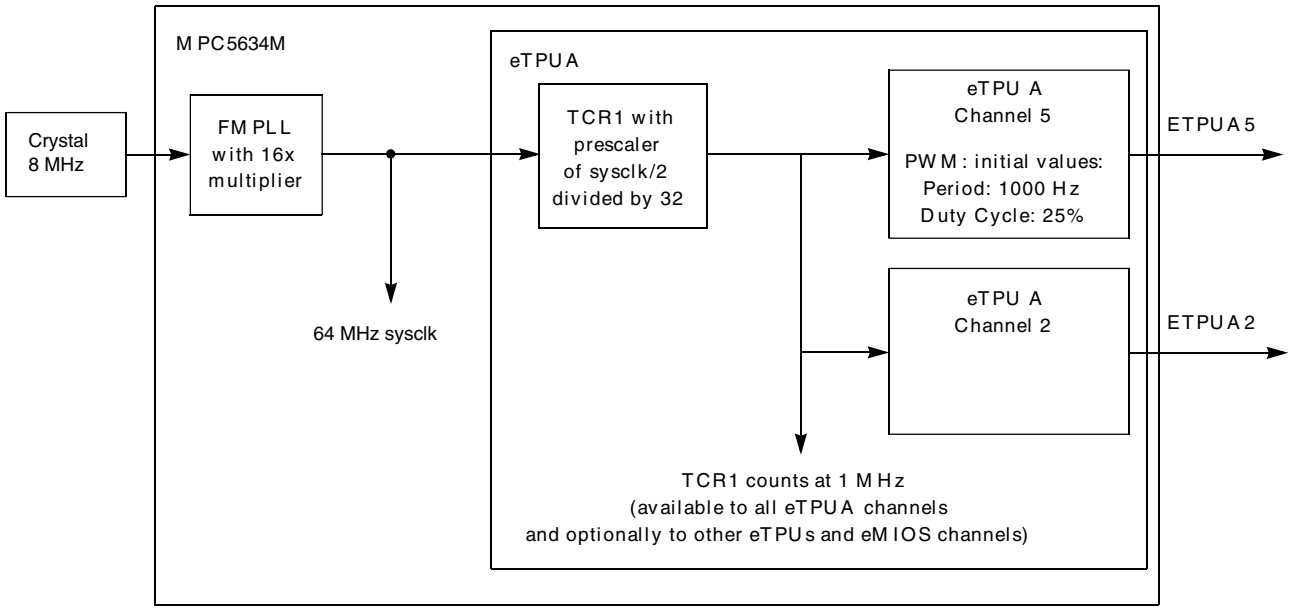

Figure 6. eTPU2 set 1 PWM function example

Table 5. eTPUA5 pin assignment

Signal	Function name	SIU PCR number	Package pin number	Header pin
eTPUA5	eTPUA5	119	P45	TRK-J39
				EVB-PJ9-6

4.2 Design

Timing resources used will include:

- sysclk = 64 MHz: assume 8 MHz crystal
- eTPU2 TCR1 clock: count at 1 MHz rate

4.2.1 Steps and pseudocode

Table 6 provides steps to initialize the eTPU2 and the function calls from the Set 1 PWM function example.

Table 6. Initialization: eTPU2 using Set 1 PWM function example

Step		Relevant bit field or structure	Pseudocode / function
Init eTPU2	Configure eTPUA for: • MISC not used • eTPUA Input filter clock divided by 8 • eTPUA Channel input filter uses 3 samples • eTPUA TCR1 = sysclk/2 prescaled by 32 • eTPUA TCR2 = sysclk/8 prescaled by 8 • eTPUB configurations as desired	etpu_config_t	fs_etpu_init
Init eTPUA[5] PWM	• Channel = eTPUA[5] • Priority = middle • Frequency = 1000 Hz • Duty cycle = 25% • Timebase = TCR1 • Timebase frequency = 1 MHz	PA = 3 OBE = 1 ODE =	fs_etpu_pwm_init
Configure pad	Configure Pad for eTPUA[5] output • Pad Assignment = eTPUA[5] • Output Buffer is enabled • Open Drain is not enabled		SIU_PCR[119] = 0x0E00
Start timers	Start all eTPU2 timers and eMIOS timers		fs_timer_start
Update eTPU2	• Channel = eTPUA[5] • Frequency = 2000 Hz • Duty cycle = Based on Trim Pot Value • Timebase = 1 MHz		fs_etpu_pwm_update

4.2.2 Files used in example

Table 7 is a summary of the files used for this project. All are available from the Freescale website, except for main.c and main.h, which are listed in the next section.

Note that the eTPU2 C compiler is not needed to make this example because precompiled eTPU2 code images are available on the Freescale website at www.freescale.com/etpu. However, all the source files used to build the set 1 ETPU2 functions are available and may be useful for reference.

Table 7. Files used in example

Provider	Type	File name	Description
Freescale	eTPU Set 1 library	etpu_pwm.c	Host application program interface for pwm function
		etpu_pwm.h	Header file for pwm function
		etpu_pwm_auto.h	Parameters automatically generated by eTPU compiler for pwm function
		etpu_set1.h	Code image and globals generated by eTPU compiler for all of set 1 ETPU functions
	eTPU utilities	etpu_util.c etpu_util.h etpu_struct.h	Host utilities to initialize eTPU, copy code image into code RAM, etc.

4.3 Code

4.3.1 eTPU2 initialization code

This code shows the sequence of how the eTPU2 is initialized in C.

```
#include "mpc563m.h" /* Use proper include file such as mpc5510.h or mpc5554.h */
#include "mpc563m_vars.h"       /* MPC563m specific variables */
#include "etpu_util.h"          /* useful utility routines */
#include "etpu_set1.h"          /* eTPU standard function set 1 */
#include "etpu_pwm.h"           /* eTPU PWM API */

/* User written include files */
#include "main.h"        /* include application specific defines. */

uint32_t *fs_free_param; /* pointer to the first free parameter */

...

  int32_t error_code;                   /* Returned value from etpu API functions */

  initSysclk();                        /* Initialize PLL to 64 MHz       */
                                       /* Initialize eTPU hardware       */
  fs_etpu_init (             my_etpu_config,
                (uint32_t *) etpu_code,
                             sizeof (etpu_code),
                (uint32_t *) etpu_globals,
                             sizeof (etpu_globals));
                                          /* Initialize eTPU channel ETPU_A[5] */
  error_code = fs_etpu_pwm_init  (5,      /*  Channel ETPU_A[5]             */
                                  FS_ETPU_PRIORITY_MIDDLE,
                                  1000,   /* Frequency = 1000 Hz            */
                                  2500,   /* Duty cycle = 2500/100 = 25%    */
                                  FS_ETPU_PWM_ACTIVEHIGH,
                                  FS_ETPU_TCR1,
                                  1000000);  /* Timebase (TCR1) freq is 1 MHz  */
```

```
  SIU.PCR[119].R = 0x0E00;           /* Configure pad for signal ETPU_A[5] output */
  fs_timer_start ();                        /* Enable all timebases           */

...
```

This routine updates the PWM duty cycle based on the trim pot value acquired by the eQADC.

```
error_code = fs_etpu_pwm_update (5,             /* Channel ETPU_A[5] */
                                 2000,          /* New frequency = 2KHz */
                                 DutyCycle*100, /* New duty cycle*/
                                 1000000);      /* Timebase (TCR1) freq = 1 MHz */
```

4.3.2 main.h listing

The main.h file includes the eTPU2 configuration structure used in the main program.

```
/* main.h based on gpio_example.h below */
 /*************************************************************************
 * FILE NAME: $RCSfile: gpio_example.h,v $   COPYRIGHT (c) FREESCALE 2004 *
 * DESCRIPTION:                                  All Rights Reserved   *
 * This file contains prototypes and definitions for the sample MPC5500   *
 * program using the the eTPU GPIO function.                              *
 *=======================================================================*
 * ORIGINAL AUTHOR: Jeff Loeliger (r12110)                                *
 * $Log: gpio_example.h,v $
 * Revision 1.1  2004/12/08 11:45:09  r47354
 * Updates as per QOM API rel_2_1
 *
 *.......................................................................*
 * 0.1   J. Loeliger  05/Sep/03    Initial version.                       *
 * 0.2   K Terry      29/Apr/04    mod'd for GPIO function test           *
 * 0.3                             Updated for new build structure.       *
 * 0.4   G. Emerson   2/Nov/04     Added etpu_config_t definition         *
 *************************************************************************/
 /* Rev 15/Mar/06 S. Mihalik : modified for eTPU PWM example  */
 /* Rev 15/Mar/06 S. Mihalik : modified for eTPU PWM example  */
 /* Rev 16/Jul/07 S. Mihalik : modified for 50 MHz sysclk, 1 MHz TCR1 */
 /* Rev 10/Aug/07 S. Mihalik: modified for 64 MHz sysclk, still 1 MHz TCR1 */

#include "etpu_util.h"

struct etpu_config_t my_etpu_config = {
  FS_ETPU_MISC_DISABLE,       /*MCR register*/

  FS_ETPU_MISC,               /*MISC value from eTPU compiler link file*/

                              /*Configure eTPU engine A*/
  FS_ETPU_FILTER_CLOCK_DIV8 +
  FS_ETPU_CHAN_FILTER_3SAMPLE +
  FS_ETPU_ENTRY_TABLE,

                              /*Configure eTPU engine A timebases*/
  FS_ETPU_TCR2CTL_DIV8 +
  ( 7 << 16) +                /*TCR2 prescaler of 8 (7+1)*/
  FS_ETPU_TCR1CTL_DIV2 +
  31,                         /*TCR1 prescaler of 32 (31+1) applied to sysclk/2*/
  0,
                              /*Configure eTPU engine B*/
  FS_ETPU_FILTER_CLOCK_DIV4 +
  FS_ETPU_CHAN_FILTER_3SAMPLE +
  FS_ETPU_ENTRY_TABLE,

                              /*Configure eTPU engine B timebases*/
```

```
  FS_ETPU_TCR2CTL_DIV8 +
  ( 7 << 16) +                  /*TCR2 prescaler of 8 (7+1)*/
  FS_ETPU_TCR1CTL_DIV2 +
  3,                            /*TCR1 prescaler of 4 (3+1)*/
  0
};
```

5 Main program

5.1 Description

As mentioned in the introduction, this example program uses the eQADC and the eTPU2 to control the brightness of an LED. The brightness of the LED depends on the setting of the trim potentiometer. We will look at how this is done in the next sections.

5.2 Design

Before we proceed to the example application, Table 8 shows the flow control of the main function.

Table 8. Main function flow control

Step		Type	Remarks
Initialize variables	error_code	int32_t	eTPU API return error code
	DutyCycle	uint16_t	This variable is passed to the eTPU API to change the duty cycle.
	State	init16_t	Ensures that the duty cycle of the PWM falls within a range of 1:99 or 99:1.
Initialize the system clock	initSysclk	function call	Set the system clock to 64 MHz.
Initialize eQADC	initADC0	function call	Enable the ADC0 only in the eQADC.
Initialize the eTPU	fs_etpu_init	function call	Enable the eTPU hardware.
Initialize the eTPU PWM function	fs_etpu_pwm_init	function call	Initialize the eTPU channel 5 frequency at 1 kHz and set duty cycle to 25%.
Set pad configuration	SIU.PCR[119]	0x0E00	Configure pad for signal ETPU_A[5] output.
While loop			Loop forever.
Read trim pot	SendConv	function call	Send a command to CFIF0 0 to read the trim pot potential difference.

Table continues on the next page...

Table 8. Main function flow control (continued)

Process result	ReadResult	function call	Read the eQADC result from RFIFO 0 and store the result in the variable Result (a global variable defined in main.c).
Update PWM duty cycle	fs_etpu_pwm_update	function call	Update the new PWM duty cycle.
Duty cycle setting control	"if" flow control	flow control	Compute the PWM duty cycle based on the result and ensure that the duty cycle falls within a range of 1:99 or 99:1. The duty cycle is computed using this equation: State = (16000 – (Result – 312) ÷ 160. The value 312 removes the noise and 16000 is the max return value for the eQADC.
End while loop			

We are now ready to run this example program.

5.3 Code

Here is the complete listing of main.c for this demonstration application.

```
#include "mpc563m.h"              /* Use proper include file */
#include "mpc563m_vars.h"         /* MPC563m specific variables */
#include "etpu_util.h"            /* useful utility routines */
#include "etpu_set1.h"            /* eTPU standard function set 1 */
#include "etpu_pwm.h"             /* eTPU PWM API */
#include "stm.h"

/* User written include files     */
#include "main.h"                 /* include application specific defines */

uint32_t *fs_free_param;          /* pointer to the first free parameter  */

void initSysclk (void) {

  FMPLL.ESYNCR2.R = 0x00000002;
  FMPLL.ESYNCR1.R = 0xF0000000 + 40;   /* Set to clock 40MHz */

  while (FMPLL.SYNSR.B.LOCK != 1) {}; /* Wait for FMPLL to LOCK */

  FMPLL.ESYNCR1.R = 0xF0000000 + 64;   /* Set to clock 64MHz */

}

static uint32_t Result     = 0;     /* ADC conversion result */
static uint32_t ResultInMv = 0;     /* ADC conversion result in millivolts */

void initADC0(void) {
  EQADC.CFPR[0].R = 0x80800F01;     /* Send CFIFO 0 a ADC0 configuration command */
                                    /* enable ADC0 & sets prescaler= divide by 32 */
  EQADC.CFCR[0].R = 0x0410;         /* Trigger CFIFO 0 using Single Scan SW mode */
```

```
  while (EQADC.FISR[0].B.EOQF !=1) {} /* Wait for End Of Queue flag */
  EQADC.FISR[0].B.EOQF = 1;           /* Clear End Of Queue flag */
}

void SendConvCmd (void) {
  EQADC.CFPR[0].R = 0x80001100;  /* Conversion command: convert channel 17 */
                                 /* with ADC0, set EOQ, and send result to RFIFO 0 */
EQADC.CFCR[0].R = 0x0410;       /* Trigger CFIFO 0 using Single Scan SW mode */
}

void ReadResult(void) {
  while (EQADC.FISR[0].B.RFDF != 1){}  /* Wait for RFIFO 0's Drain Flag to set */
  Result = EQADC.RFPR[0].R;            /* ADC result */
  ResultInMv = (uint32_t)((5000*Result)/0x3FFC);  /* ADC result in millivolts */
  EQADC.FISR[0].B.RFDF = 1;                        /* Clear RFIFO 0's Drain Flag */
  EQADC.FISR[0].B.EOQF = 1;                     /* Clear CFIFO's End of Queue flag */ }


void main ()
{
  int32_t  error_code;            /* Returned value from etpu API functions */
  uint16_t DutyCycle;
  int16_t  State;

  initSysclk();                   /* Initialize PLL to 64 MHz */
  initADC0();                     /* Enable ADC0 only on eQADC */

  /* Initialize eTPU hardware */
  fs_etpu_init (             my_etpu_config,
                (uint32_t *) etpu_code,
                             sizeof (etpu_code),
                (uint32_t *) etpu_globals,
                             sizeof (etpu_globals));

  /* Initialize eTPU channel ETPU_A[5] */
  error_code = fs_etpu_pwm_init  (5,      /*  Channel ETPU_A[5] */
                                  FS_ETPU_PRIORITY_MIDDLE,
                                  1000,             /* Frequency = 1000 Hz */
                                  2500,             /* Duty cycle = 2500/100 = 25%*/
                                  FS_ETPU_PWM_ACTIVEHIGH,
                                  FS_ETPU_TCR1,
                                  1000000);      /* Timebase (TCR1) freq is 1 MHz */

  SIU.PCR[119].R = 0x0E00;   /* Configure pad for signal  ETPU_A[5] output */
  fs_timer_start ();         /* Enable all timebases  */

  error_code = fs_etpu_pwm_update (5,        /* Channel ETPU_A[5] */
                                   2000,     /* New frequency = 2KHz */
                                   6000,     /* New duty cycle = 6000/100= 60% */
                                   1000000); /* Timebase (TCR1) freq = 1 MHz */

  DutyCycle = 1;          /* Limit the duty cycle from 1% to 99%          */
  State     = 0;          /* Initialized the state machine                */
  while(1)                /* Loop forever                                 */
  {

    SendConvCmd();        /* Send one conversion command */
    ReadResult();         /* Read result */

    error_code = fs_etpu_pwm_update (5,           /* Channel ETPU_A[5] */
                                     2000,        /* New frequency = 2KHz */
                                     DutyCycle*100, /* New duty cycle*/
                                     1000000);    /* Timebase (TCR1) freq = 1 MHz */

    State = (uint16_t)(16000 - (Result - 312));

    if (State <= 1)
    {
        DutyCycle = 1;                    /* force the duty cycle to 1:99 */
    }
```

```
    else
    {
        DutyCycle = (uint16_t)(State/160);  /* Compute the duty cycle */
        if (DutyCycle > 99)
        {
            DutyCycle = 99;                 /* force the duty cycle to 99:1 */
        }
    }

  }
}
```

6 Running the demo project

Unzip the project zip file, LED_Dimmer.zip. The project contains the subdirectories shown Figure 7.

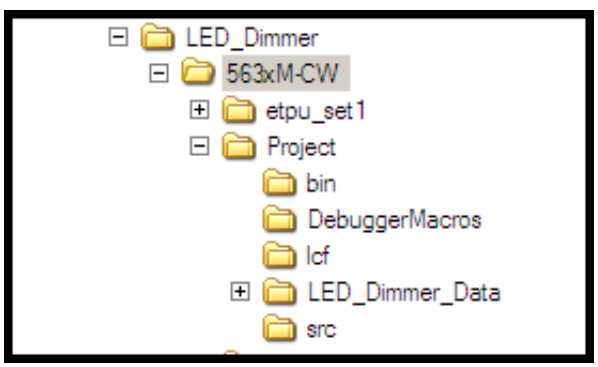

Figure 7. Subdirectories within the project folder

Double-click on the file "LED_Dimmer.mcp" to start loading the project with CodeWarrior, as shown in Figure 8.

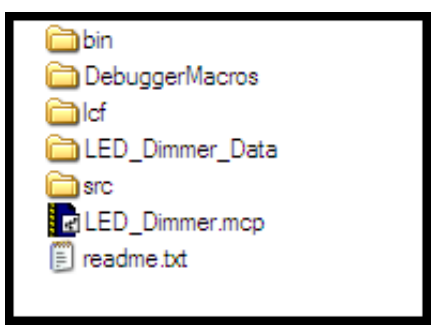

Figure 8. CodeWarrior project file

After CodeWarrior has loaded the project, click on the icon "Make" to compile the project. Make sure the board is set up with the P&E debugger attached. Click on the icon "Debug" to start loading the binary code to the board and execute the demo program as shown in Figure 9.

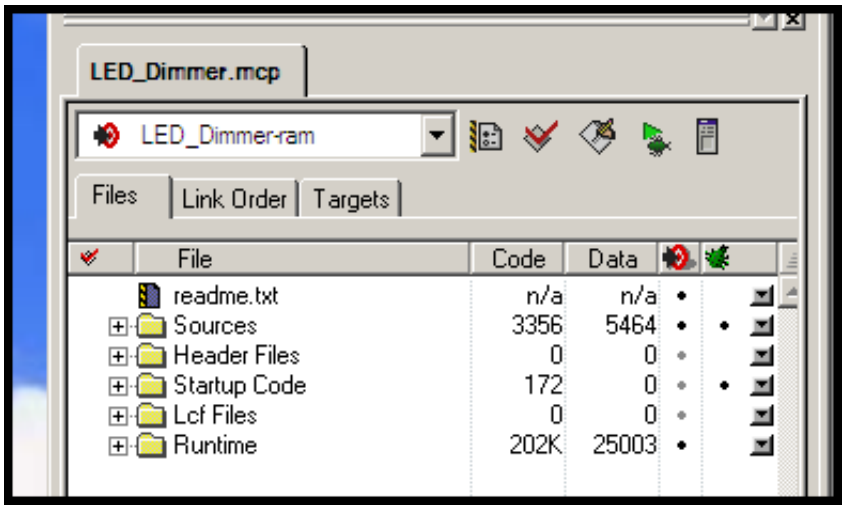

Figure 9. CodeWarrior IDE

Figure 10 and Figure 11 illustrate the output waveforms of the eTPU2 PWM signals at 99% and 1%, respectively. The PWM duty cycle is controlled by the trim pot. However, the software limits the duty cycle of the PWM to a ratio that varies from 1:99 to 99:1.

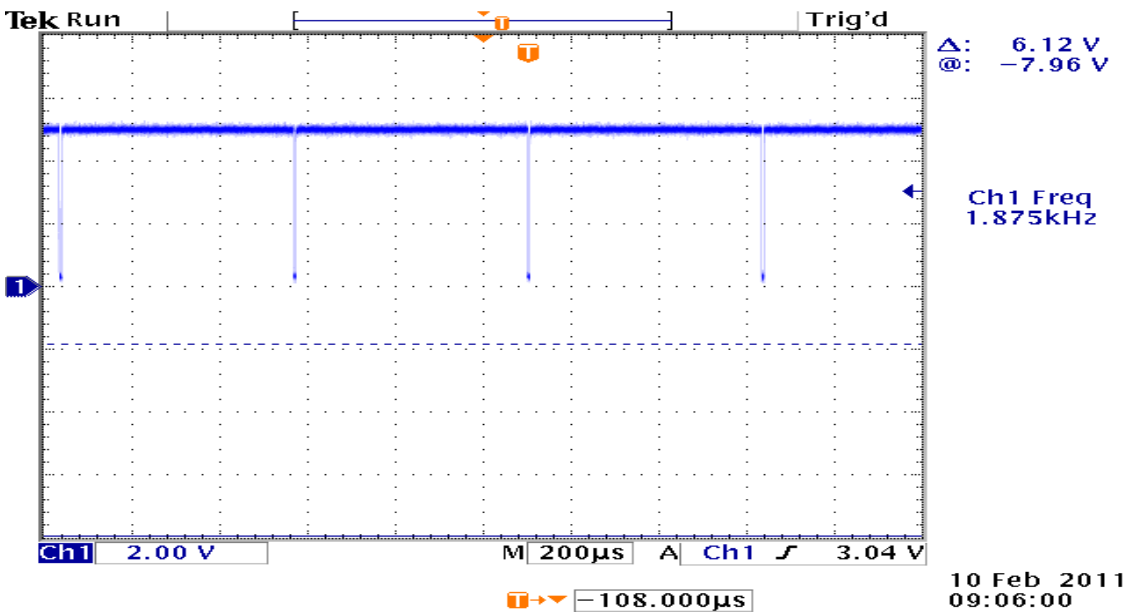

Figure 10. Output waveform at 99% duty cycle

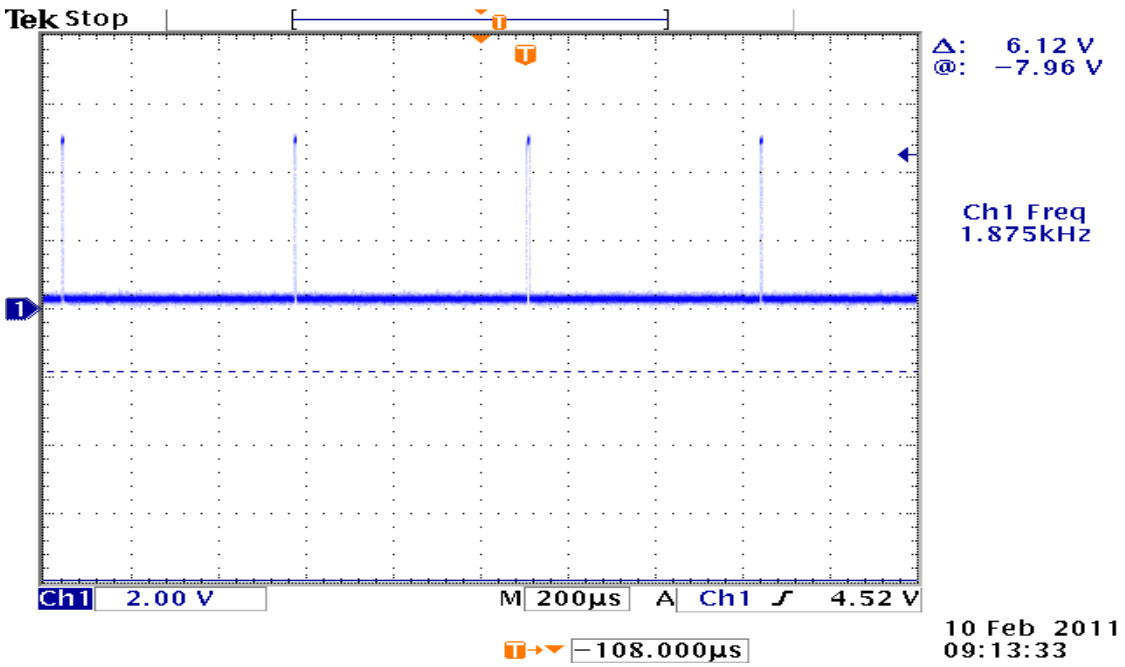

Figure 11. Output waveform at 1% duty cycle

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4266
Rev. 1, May 2011

freescale semiconductor