Freescale Semiconductor
Application Note

Document Number: AN4793
Rev. 0, 09/2013

Examples of using eTimer on Power Architecture devices

by: Tomas Kulig

1 Introduction

This application note describes how to use the Enhanced Motor Control Timer (eTimer) module and what is necessary to set in the device for using the eTimer. The base features of the eTimer are shown in four examples which were developed in a GreenHills project for RAM memory; generating periodical signal, generating periodical pulse, generating one-shot signal and measure signal parameters. This application note focuses on the eTimer module on the MPC5744P.

Contents

2 Implementation of eTimer in the device

There are three independent modules and each has six independent channels. All three modules are able to generate external signals and work with input signals.

For using eTimer we need to configure following:

2.1 CGM module - for live – set clock

There is no special divider and clock selector for the eTimers modules. The eTimers use the Motor Control clock which can be up to 160 MHz if the selector uses the PLL.
The selector CGM_AC0_SC can use internal oscillator, external oscillator (crystal) or PLL0. This selector is valid for ADC and SWG clock. Figure 2 shows the field description of CGM_AC0_DC0 register.

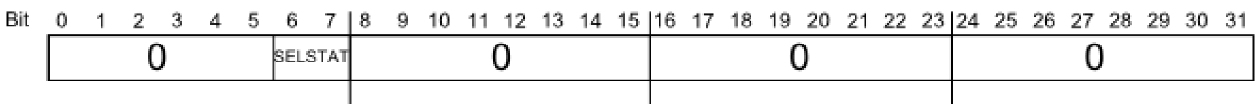

Figure 1. **MC_CGM_AC0_SC field description**

Where SELSTAT can be 0 - internal oscillator 16 MHz, 1 - external oscillator/crystal 8-40 MHz or 2 – PLL0.
The divider CGM_AC0_DC0 can enable/disable the clock and divide by 1 up to 16.

Warning: Use only odd DIV values (i.e., division factor of 2, 4, 6, 8, 10, 12, 14 or 16). Even values will cause incorrect device behavior. Figure 2 shows the field description of CGM_AC0_DC0 register.

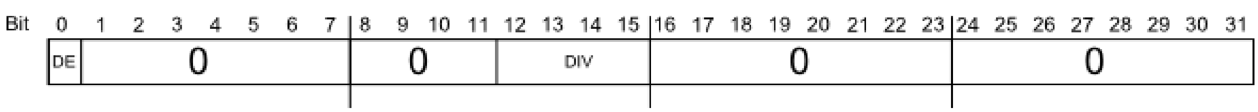

Figure 2. **MC_CGM_AC0_DC0 field description**

Where DE bit is for divider enable - 1/disable - 0. Div can be 0 up to 15. The motor control clock is divided by value “DIV+1”.

2.2 Enable clock in mode entry

PCTLs registers select the group for non- low- power mode and for low-power modes, each peripheral can be asserted only to one low-power group and one non-low-power group. Each group can be asserted for one or more modes. There are eight groups for non-low-power modes (RUN_PC0 – RUN_PC7) and eight groups for low-power modes (LP_PC0 – LP_PC7). See Figure 3 and code below which shows an example of enabling the clocks for the eTimers. All eTimer have clocks enabled in modes RUN0, RUN1, RUN2, RUN3 and DRUN. The eTimer0 has enable clock in modes STOP0 and HALT0 but eTimer1 and eTimer2 have enable clock only in STOP0 mode.

Example code:

```
//enable group RUN_PC0
MC_ME.RUN_PC[0].R          = 0xF8;  //enable DRUN, RUN3, RUN2, RUN1 and RUN0
//enable group LP_PC0
MC_ME.LP_PC[0].R           = 0x500 //enable STOP0 and HALT0
//enable group LP_PC1
MC_ME.LP_PC[1].R           = 0x400 //enable HALT0
//set peripherals for group RUN_PC0
MC_ME.PCTL247.B.RUN_CFG    = 0x0    //eTimer 0 - set group RUN_PC0 for enable clock
MC_ME.PCTL137.B.RUN_CFG    = 0x0    //eTimer 1 - set group RUN_PC0 for enable clock
MC_ME.PCTL245.B.RUN_CFG    = 0x0    //eTimer 2 - set group RUN_PC0 for enable clock
//set peripherals for group LP_PC0
MC_ME.PCTL247.B.LP_CFG     = 0x0    //eTimer 0 - set group LP_PC0 for enable clock
//set peripherals for group LP_PC1
MC_ME.PCTL137.B.LP_CFG     = 0x1    //eTimer 1 - set group LP_PC1 for enable clock
MC_ME.PCTL245.B.LP_CFG     = 0x1    //eTimer 2 - set group LP_PC1 for enable clock
```

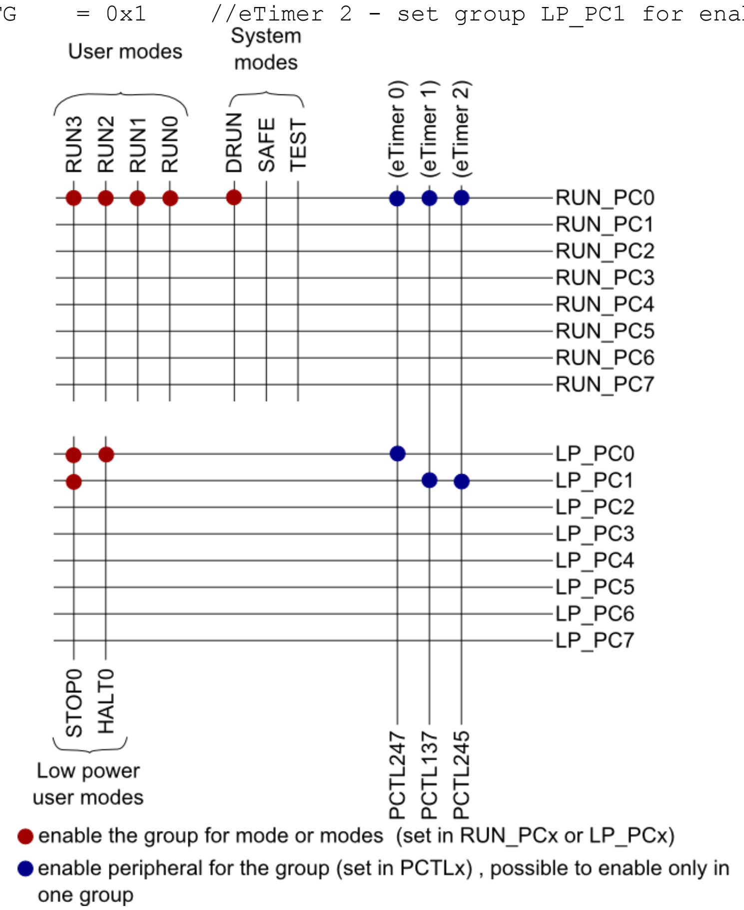

Figure 3. **Example of clock enable**

2.3 SIUL2 module

SIUL2 module provides communication with external world. Table 1, 2 and 3 show all pins which can be used for eTimers modules. MSCR is used for output direction and IMCR is used for input direction.

Table 1. eTimer 0 pins

channel	direction	MSCR/IMCR	SSS	PORT PIN	PINS	
					144 LQFP	257MAPBGA
0	O	0/-	1	A0	73	P12
	I	0/59	2	A0	73	P12
		58/59	1	D10	76	R16
1	O	1/-	1	A1	74	T14
	I	1/60	2	A1	74	T14
		59/60	1	D11	78	P17
2	O	2/-	1	A2	84	L14
	I	2/61	2	A2	84	L14
		80/61	1	F0	133	B6
3	O	3/-	1	A3	92	G15
	I	3/62	2	A3	92	G15
		62/62	1	D14	105	E17
4	O	4/-	3	A4	108	D16
		43/-	1	C11	80	P16
	I	4/63	3	A4	108	D16
		30/63	1	B14	64	P11
		43/63	4	C11	80	P16
		99/63	2	G3	104	E16
5	O	44/-	1	C12	82	M14
		77/-	1	E13	117	A11
	I	24/64	1	B8	47	P7
		44/64	3	C12	82	M14
		77/64	4	E13	117	A11
		100/64	2	G4	100	F16

Table 2. eTimer 1 pins

channel	direction	MSCR/IMCR	SSS	PORT PIN	PINS	
					144 LQFP	257MAPBGA
0	O	4/-	1	A4	108	D16
		47/-	2	C15	124	A8
	I	4/65	1	A4	108	D16
		47/65	2	C15	124	FA8
1	O	45/-	1	C13	101	E15

channel	direction	MSCR/IMCR	SSS	PORT PIN	PINS	
					144 LQFP	257MAPBGA
		48/-	2	D0	125	B8
	I	45/66	1	C13	101	E15
		48/66	2	D0	125	B8
2	O	16/-	2	B0	109	C16
		46/-	1	C14	103	F14
		49/-	2	D1	3	E3
	I	16/67	1	B0	109	C16
		46/67	2	C14	103	F14
		49/67	3	D1	3	E3
3	O	17/-	2	B1	110	C14
		50/-	2	D2	140	B4
		92/-	1	F12	106	D17
	I	17/68	1	B1	110	C14
		50/68	2	D2	140	B4
		92/68	3	F12	106	D17
4	O	14/-	2	A14	143	A3
		51/-	2	D3	128	A5
		56/-	2	D8	32	L4
		93/-	1	F13	112	A15
	I	14/69	1	A14	143	A3
		51/69	2	D3	128	A5
		56/69	3	D8	32	L4
		93/69	4	F13	112	A15
5	O	5/-	2	A5	14	H4
		15/-	2	A15	144	D3
		52/-	2	D4	129	B7
		78/-	1	E14	119	B10
	I	5/70	1	A5	14	H4
		15/70	2	A15	144	D3
		52/70	3	D4	129	B7
		78/70	4	E14	119	B10

Table 3. eTimer 2 pins

channel	direction	MSCR/IMCR	SSS	PORT PIN	PINS	
					144 LQFP	257MAPBGA
0	O	116/-	2	H4	-	F4
		128/-	1	I0	-	C6
	I	116/71	1	H4	-	F4

channel	direction	MSCR/IMCR	SSS	PORT PIN	PINS	
					144 LQFP	257MAPBGA
		128/71	2	I0	-	C6
1	O	119/-	2	H7	-	F2
		129/-	1	I1	-	T3
	I	119/72	1	H7	-	F2
		129/72	2	I1	-	T3
2	O	6/-	2	A6	2	D1
		122/-	2	H10	-	C7
		130/-	1	I2	-	D11
		152/-	2	J8	95	G16
	I	6/73	1	A6	2	D1
		122/73	2	H10	-	C7
		130/72	3	I2	-	D11
		152/73	4	J8	95	G16
3	O	7/-	2	A7	10	G4
		125/-	2	H13	-	A14
		131/-	1	I3	-	A10
	I	7/74	1	A7	10	G4
		125/74	2	H13	-	A14
		131/74	3	I3	-	A10
4	O	8/-	2	A8	12	H1
		126/-	2	H14	-	P13
		137/-	1	I9	-	L3
		152/-	1	J8	95	G16
	I	8/75	1	A8	12	H1
		126/75	2	H14	-	P13
		137/75	3	I9	-	L3
		152/75	4	J8	95	G16
5	O	9/-	2	A9	134	A4
		127/-	2	H15	-	C17
		138/-	1	I10	-	M3
		153/-	1	J9	16	K1
	I	9/76	1	A9	134	A4
		127/76	2	H15	-	C17
		138/76	3	I10	-	M3
		153/76	4	J9	16	K1

2.3.1 Set as input

Set the IMCR and MSCR register.

The IMCR registers select the input functionality of a pin and other parameters are set by the MSCR registers. The MSCR register manages the following main parameters of a pin:

- input buffer enable (IBE)
- output buffer enable (OBE)
- slew rate (SRC)
- output functionality (SSS).

```
IMCR[number of IMCR register].SSS.B =  SSS from table;        //select the input
functionality
MSCR[number of MSCR register].IBE.B =  0x1;                   //enable input buffer
MSCR[number of MSCR register].OBE.B =  0x0;                   //disable output buffer
MSCR[number of MSCR register].SRC   =  slew_rate;             //set slew rate
```

2.3.2 Set as output

Set the MSCR register.

```
MSCR[number of MSCR register].SSS.B =  SSS from table;        //select the output
functionality
MSCR[number of MSCR register].IBE.B =  0x0;                   //disable input buffer
MSCR[number of MSCR register].OBE.B =  0x1;                   //enable output buffer
MSCR[number of MSCR register].SRC   =  slew_rate;             //set slew rate
```

3 Generating periodical signal

Description: This function is for generating periodical signal with variable duty cycle.
Minimum steps of duty cycle variance given by motor control clock (motc_clk) are given in the Table 4.

Table 4: Minimum step of duty cycle

Minimal frequency		Maximum frequency		Minimum variance of duty cycle	
>	0	<=	motc_clk/1000	0.1	%
>	motc_clk/1000	<=	motc_clk/100	1	%
>	motc_clk/100	<=	motc_clk/10	10	%
>	motc_clk/10	<=	motc_clk/2	50	%

What is needed: 1 channel, 1 pad
Implementation: The COMP1 register is used for driving the duty cycle, the CMPLD1 register is used for driving the frequency of the signal. Figure 4 shows this. The output signal is set on a successful compare of COMP1 and cleared on successful compare of COMP2.

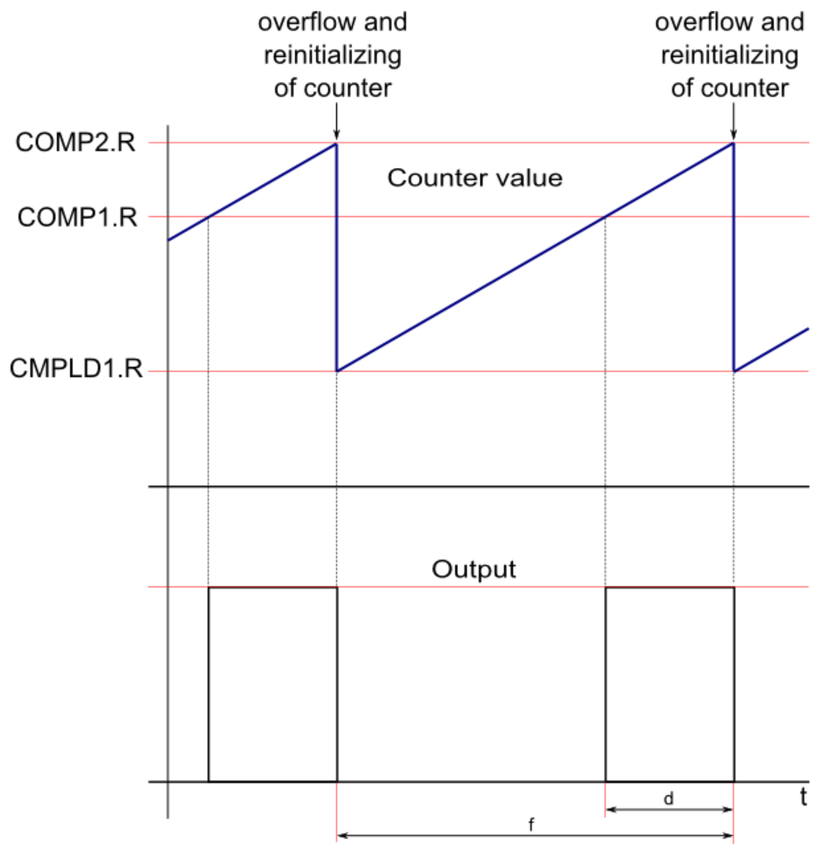

Figure 4. **Generating periodical signal waveforms**

d- duty cycle
f- frequency
motc_clk- motor control clock
eTimer_div- eTimer internal divider

1. Check if it is possible to generate the signal with parameters d and f with input frequency of module motc_clk:

```
motc_clk[Hz]/(MAX_DIVIDER*MIN_FREQ[Hz]) < f[Hz] <= motc_clk[Hz]/2
```

where, MAX_DIVIDER is value 128, it is the maximal internal eTimer divider
MIN_FREQ is value 65000. The maximum value of counter is 65535. The value 535 is reserve.
The left side of equation is low frequency board and the right side is the high frequency board.

Example:

```
motc_clk = 160 MHz
160*10^6/(128*65000) < f <= 160*10^6/2
19 Hz < f <= 80 MHz
```

It means device is able to generate signal form 20 Hz up to 80 MHz.

2. Set the internal eTimer divider:

```
eTimer_div >= motc_clk/(f*MIN_FREQ)
```

but DIV can be only following: 1, 2, 4, 8, 16, 32, 64 or 128

Example:
```
motc_clk = 160 MHz
f = 50 Hz

eTimer_div >= 160*10^6/(50*65000)
eTimer_div >= 49
```
The closest possible value is DIV = 64.

3. Set the registers for generating signal with parameter d and f:

a) Set up the signal parameters:
```
CTRL1.B.PRISRC = this is given by eTimer_div;     //See the table 5
```

Table 5: eTimer dividers values

eTimer_div	PRISRC value
1	24
2	25
4	26
8	27
16	28
32	29
64	30
128	31

```
CTRL1.B.CNTMODE = 0x1;        // count rising edges of primary source
CTRL1.B.LENGTH = 0x1;         // count until compare then reinitialize
CCCTRL.B.CLC1 = 0x7;          // reinitializing counter by value which is stored
                              in CMPLD1

COMP2.R = 0xFFFF;                       //
COMP1.R = 0xFFFF - (range*d)/1000;      //
CMPLD1.R = 0xFFFF - range + 1;          //
```

where range = motc_clk/(eTimer_div*frequency)

b) Output setting:
```
CTRL2.B.OEN = 0x1;            //output enable
CTRL2.B.OUTMODE = 0x8;          //set on successful compare on COMP1, clear on
                                successful compare on COMP2
```

4 Generating periodical pulse (signal which has long period but thin pulse)

Description: This function is for generating periodical signal with variable pulse width. One channel of eTimer is used for generating the pulse. Period is given by 1 or 2 channels. It depends on length of the period.
What is needed:

a) short period: 2 channel, 1 pad
b) long period: 3 channel, 1 pad, two chained channels are used for period

Implementation:
The Figure 5 shows the connection between the eTimer channels. The colors of blocks correspond with the waveform on figures 6 and 7.

Short period
Channel C defines the period of the signal that triggers the channel A which generates the pulse. The channel A COMP1 register is used for driving 50 % duty cycle and the CMPLD1 register is used for driving period of signal. The output signal is set on successful compares of COMP1 and cleared on successful compares of COMP2. The output signal of channel C is used as the secondary source for channel A (as a trigger). This signal is only inside the device (inside the module). The channel A COMP1 register is used for driving the width. The output signal is cleared on the secondary source input edge and set on the compare with COMP1. The output signal is inverted and then is routed to the pin (output). The waveforms are shown on the Figure 6.

Long Period
Channel C is used as a source of channel B which triggers the channel A which then generates the pulse. The channels B and C define the period of signal. The channels B and C COMP1 registers are used for driving 50 % duty cycle and CMPLD1 registers are used for driving period of signal. The output signals of both channels are set on successful compare of COMP1s and cleared on successful compares of COMP2s. The output signal of channel C is used as the source of clock for channel B. This signal is only inside the device (inside the module). The output signal of channel B is used as the secondary source for channel A (as a trigger). This signal is only inside the device (inside the module). The channel A COMP1 register is used for driving the width. The output signal is cleared on the secondary source input edge and set on compare with COMP1. The output signal is inverted and then is routed to the pin (output). The waveforms are shown on the Figure 7.

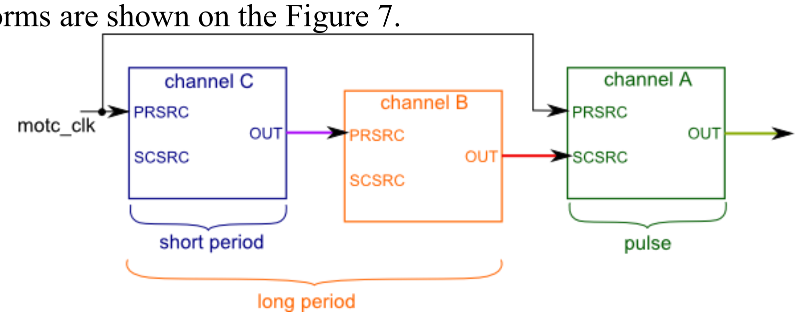

Figure 5. **Connection between channels**

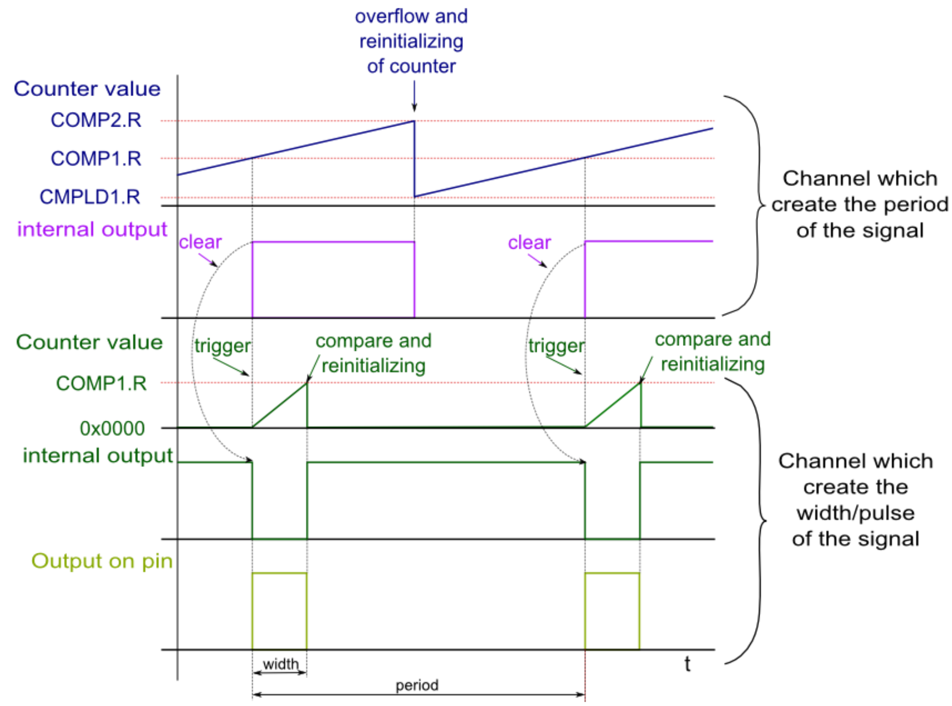

Figure 6. **Short period waveforms**

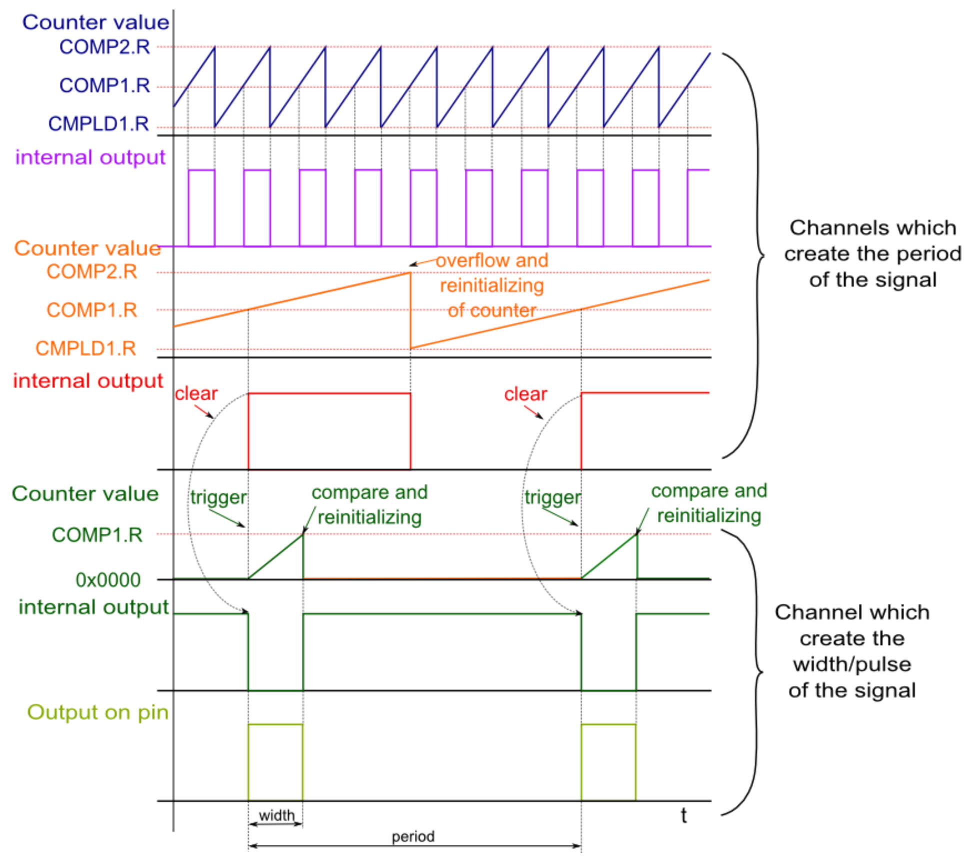

Figure 7. **Long period waveforms**

width -width of pulse
period - period of signal
range_b- range for channel of timer which defines the period – short period
range_c-range for channel of timer which defines the period, this is used only for longer period
div_pulse-value of channel divider which is used for pulse
div_period-divider of channel which is used for period
short x long period
motc_clk -motor control clock

4.1 Check if it is possible to generate the signal with parameters width and period with motc_clk input frequency

a) Check the width
1000000/motc_clk [kHz] < width [ns] < MIN_FREQ*MAX_DIVIDER*1000000/motc_clk [kHz],
where MAX_DIVIDER is value 128, it is the maximal internal eTimer divider
MIN_FREQ is value 65000, it is maximum value of counter (535 is reserve).
The left side of equation is the minimum period and the right side is the maximum period.

b) Check the period
2/motc_clk [kHz] < period [ms] < MIN_FREQ*MIN_FREQ*MAX_DIVIDER/motc_clk [kHz],
where MAX_DIVIDER is value 128, it is the maximal internal eTimer divider
MIN_FREQ is value 65000, it is maximum value of counter (535 is reserve).
The left side of equation is the minimum period and the right side is the maximum period.

It makes sense to check one more conclusion:
2*period [ms] *1000000 > width [ns]
Period must be at least twice bigger than width.

Decide if the signal has long or short period
```
short period < MIN_FREQ*MAX_DIV/motc_clk <= long period
```

Example:
```
motc_clk = 160 MHz
short period < 65000 * 128 / 160 [MHz] <= long period
short period < 52 [ms]  <= long period
```

4.2 Set registers for generating the pulse

The width of the pulse does not depend on the number of channels which are used for generating the period but the secondary source depends on this. It means that for short periods the output of channel C is used and for long periods the output of channel B is used as the secondary source. The first channel is used for short period (channel C) and the second channel (channel B) is used for long period.
Set the internal eTimer divider: div_pulse >= (width [ns] * motc_clk [GHz])/MIN_FREQ, but div can be only following: 1, 2, 4, 8, 16, 32, 64 or 128.

Example:

```
a) motc_clk = 160 MHz
   width = 750 ns
   div_pulse >= (750 * 0.16)/65000
   div_pulse >= 0
   The closest possible value is div_pulse = 1.
b) Set up the pulse parameters:
    CTRL1.B.PRISRC = this is given by div_pulse; //See the table 5 - eTimer_div =
    div_period
     CTRL1.B.CNTMODE = 0x6;//edge of secondary source triggers primary count till compare
     CTRL1.B.LENGTH = 0x1; //count until compare then reinitialize
     COMP2.R = 0xFFFF;      // here is not use this comparator set out of working area
     COMP1.R = (motor_clk [GHz] * width [ns])/div_pulse;
```

The secondary source of channel A (channel which generates the pulse) is given by the length of period generating signal. It is channel C for short period. It is channel B for long period.

Short:

```
CTRL1.B.SECSRC = 16 + number of channel C;  //for period is use one channel (channel C)
```

Long:

```
CTRL1.B.SECSRC = 16 + number of channel B;  //for this period is used two channels, here is
used the channel B which trigger channel A which create the pulse. The channel C is source of
primary clock for the channel B.
```

Output setting:

```
CTRL2.B.OEN = 0x1;                  //output enable
CTRL2.B.OUTMODE = 0x5;              // set on successful compare on COMP1, clear on secondary
source
```

Input edge:

```
CTRL2.B.OPS = 0x1;                  // inverted output
```

4.3 Set registers for generating short period

Only one channel of eTimer is used (channel C on Figure 5).

a) Set the internal eTimer divider:

```
div_period >= motc_clk [kHz] * period [ms]/ MIN_FREQ
```

Example:

```
motc_clk = 160 MHz
period = 20 ms

div_period >= (160000 * 160000)/65000
div_period >= 49
```

The closest possible value is div_period = 64.

b) Set up the period parameters:

```
CTRL1.B.PRISRC = this is given by div_period;        //See the table 5 - eTimer_div =
div_period
CTRL1.B.CNTMODE = 0x1;                               //count rising edges of primary source
CTRL1.B.LENGTH = 0x1;                                //count until compare then reinitialize
CCCTRL.B.CLC1 = 0x7;                         //reinitializing counter by value which is stored
in CMPLD1

COMP2.R = 0xFFFF;                                    //
COMP1.R = 0xFFFF - range_b/2;                                 //duty cycle is always 50 %
```

```
CMPLD1.R = 0xFFFF - range_b + 1;                          //
```
where range_b = motc_clk * period/div_period

c) Output setting

```
CTRL2.B.OUTMODE = 0x8;                                    //set on successful compare on COMP1,
clear on successful compare on COMP2
```
This signal is not routed on the output pin. It is used as trigger of channel which creates pulse.

4.4 Set registers for generating long period

There are two channels for generating period channel C and channel B (Figure 5). The Setting period has three level of latitude: internal eTimer divider (channel C), two counting values (channel C) and (channels B).

a) Determine ranges of both counters and divider for the channels C for given period and motor control:

```
period [ms]  = div_period * range_b * range_c/motc_clk [kHz]
where range_b and range_c is from 2 up to MIN_FREQ and div_period 1, 2, 4, 8, 16, 32, 64 or
128
```

Example:

```
motc_clk = 160 MHz
period = 250 ms
```

250 = div_period * range_b * range_c/160000
$4 * 10^7$ = div _period * range_b * range_c

Check the value of div_period:

range_b = range_c = MIN_FREQ
$4*10^7$ = div_period * MIN_FREQ2
div_period = $4 * 10^7/4225 * 10^6$ = 0,01 -> select dic_period = 1

range_b * range_c = 160000 * 250 / 1 = $4 * 10^7$
Chose range_b = 40000
range_c = $4 * 10^7$/range_b
range_c = $4 * 10^7/4 * 10^4$
range_c = 1000

b) Channel C – internal

```
CTRL1.B.PRISRC = this is given by div_period;    //See the table 5 - eTimer_div = div_period
CTRL1.B.CNTMODE = 0x1;                           //count rising edges of primary source
CTRL1.B.LENGTH = 0x1;                            //count until compare then reinitialize
CCCTRL.B.CLC1 = 0x7;                             //reinitializing counter by value which is stored
                                                 in CMPLD1

COMP2.R = 0xFFFF;                                 //
COMP1.R = 0xFFFF - range_b/2;                     //duty cycle is always 50 %
CMPLD1.R = 0xFFFF - range_b + 1;                  //
```

Output setting

```
CTRL2.B.OUTMODE = 0x8;            //set on successful compare on COMP1, clear on successful
                                  compare on COMP2
```

c) Channel B – trigger for the channel A which creates the pulse

```
CTRL1.B.PRISRC = this is given by div_period;    //See the table 5 - eTimer_div = div_period

CTRL1.B.CNTMODE = 0x1;               //count rising edges of primary source
CTRL1.B.LENGTH = 0x1;                //count until compare then reinitialize
CCCTRL.B.CLC1 = 0x7;                 //reinitializing counter by value which is stored in
                                     CMPLD1

COMP2.R = 0xFFFF;                    //
COMP1.R = 0xFFFF - range_c/2;        //duty cycle is always 50 %
CMPLD1.R = 0xFFFF - range_c + 1;     //
```

d) Output setting

```
CTRL2.B.OUTMODE = 0x8;            //set on successful compare on COMP1, clear on successful
                                  compare on COMP2
```

5 Generating one-shot signal

Description: This function generates a single short pulse which has two parameters: delay and width. The width expresses the width of the pulse and the delay expresses time between generating pulse start and the function trigger. It is possible to select the active level of pulse – high or low.
What is needed: 1 channel, 1 pad
Implementation: The COMP1 register is used for the delay and the COMP2 register is used for driving the width of the pulse.
Figure 8 shows this. The output signal is set on successful compare of COMP1 and cleared on successful compare of COMP2.
The first part is used to set the eTimer channel and the second part is used for generating the signal.

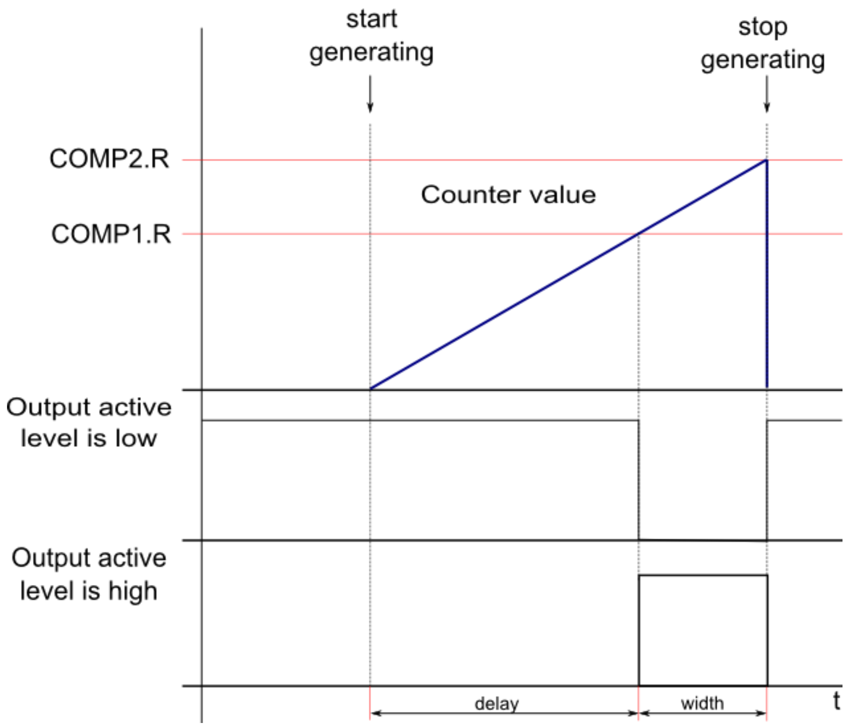

Figure 8. **Generating one-shot signal waveforms**

5.1 Check if it is possible to generate the signal with parameters width and delay with motc_clk input frequency

```
a)  check delay 2/motor_freq [MHz] < delay [us] < MIN_FREQ* MAX_DIVIDER/ motor_freq
    [MHz],Where MAX_DIVIDER is 128 and MIN_FREQ = 65000.
```

Example:

```
    motc_clk   = 160 MHz
    2/160 < delay [us] <128*65000/160
    25 ns < delay < 52 ms
b)  Check width
    width < delay
```

5.2. Setting of eTimer channel

a) Set the internal eTimer divider:

```
div_period >= motc_clk [MHz] * delay [us]/ MIN_FREQ
```

Example:

```
motc_clk   = 160 MHz
delay = 1 ms
div_period >= 160 [MHz] * 1000 [us]/ 65000
div_period >= 2.46
```

The closest possible value is div_period = 4.

b) Set up the period parameters:

```
CTRL1.B.PRISRC = this is given by div_period;       //See the table 5 - eTimer_div =
div_period
CNTR.R                = 0x0 ;                        //clear counter for the new use
COMP1.R = range - delay;                             //
COMP2.R = range + (width [us] *motc_clk [MHz]/div_period) - delay + width;
```

c) Where range = delay [µs]*motc_clk [MHz]/div_period

```
CTRL1.B.ONCE= 0x1;                        //count until compare and then stop
```

c) Output setting

```
CTRL2.B.OEN         = 0x1; - enable output
CTRL2.B.OPS         = output active level;        // 0 - low, 1- high
CTRL2.B.OUTMODE         = 0x4;            // toggle OFLAG output using alternating compare
                                            registers
```

5.3. Start generating:

```
CNTR.R              = 0x0;        // clear counter for the new use
CTRL1.B.CNTMODE     = 0x1;        // count rising edges of primary source/start generating
                                     signal
```

6 Measure signal parameters

Description: This function is for measuring signal frequency and duty cycle.
What is needed: 1 channel, 1 pad
Implementation: One channel of the eTimer is used for measuring the frequency and duty cycle. The function uses the capture functionality of the eTimer. The motor control clock is used as the primary source of clock and the input signal as secondary source. Its edges drive the capturing values of internal counter. The counter is counting repeatedly the primary source and captures its values on edges produced by the secondary source or input. The capture 1 register is set for capture the counter value on rising edge of signal and the capture 2 register is set for capture the counter value on falling edge of input signal. The capture registers have two-deep FIFO so they are able to capture two values. The frequency is calculated from the two values related to the rising edges and the duty cycle using the difference between the values related to the first rising edge and the first falling edge. The implementation is divided into two parts. First part is used for setting the eTimer channel and the second part is used for doing the measurement (start capturing and calculate the frequency and duty cycle). The Figure 9 shows the waveforms.

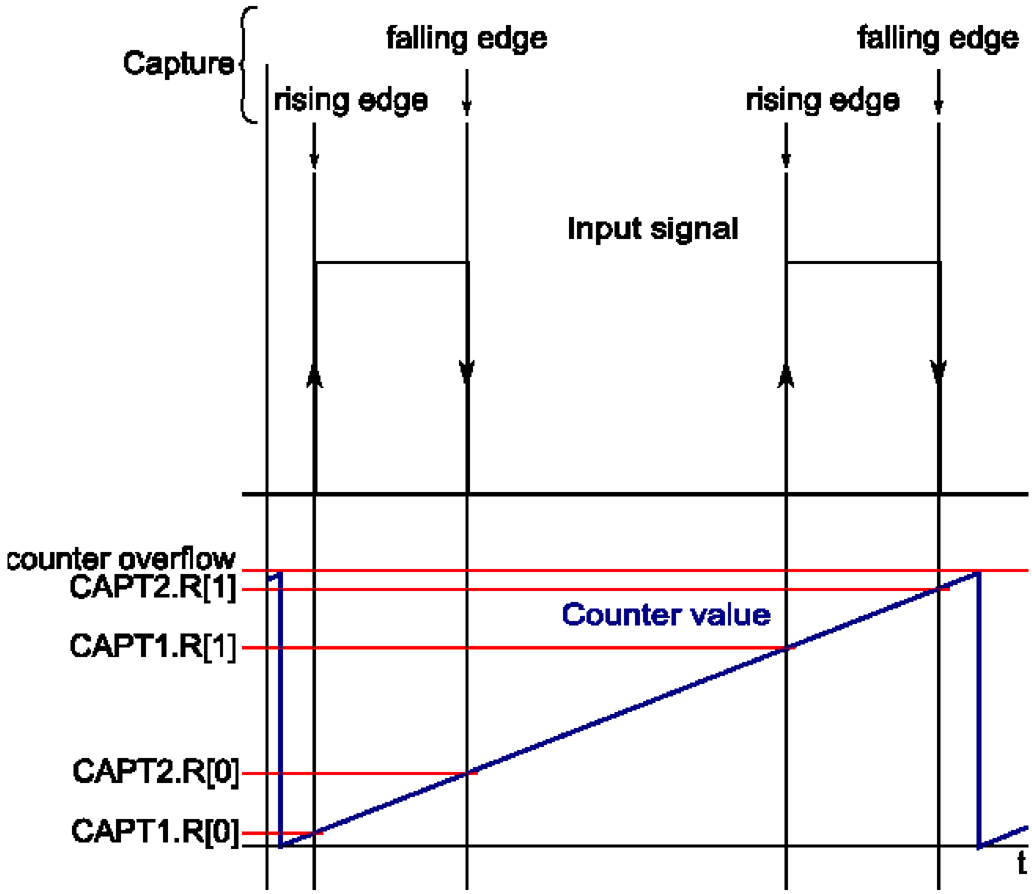

Figure 9. **Measurement signal waveforms**

The values in the brackets mean the position in the FIFO of the capture registers.

6.1 Calculate the signal parameters

a) Frequency:

```
f [kHz] = motc_clk [kHz]/(CAPT1.R[1] - CAPT1.R[0]),
```

where motc_clk is motor control clock.

Example:

The input signal was generated by function generator. Following setting has been used: 3 kHz, 11.26%, slewrates of falling and rising edge 150 ns.

```
motc_clk      = 160 MHz
CAPT1.R[1]    = 0xC642   = 50754
CAPT1.R[0]    = 0xF5F0  = 62960

f [kHz] = 160000/(50754-62960) = 160000/53329 = 3.0002 kHz  (3 kHz in device)
```

50754 – 62960 is equal 53329 because 16 bit unsigned format is used and the counter counts repeatedly. So the counter counts from 62960 to 65535 – overflow to 0 (maximum 16 bit value) and from 0 to 50754 so = 65535 – 62960 + 50754 = 53329.

b) Duty cycle

duty [per thousand] = ((CAPT2.R[0] - CAPT1.R[0])*1000)/(CAPT1.R[1] - CAPT1.R[0])

Example:

```
CAPT1.R[1]      = 0xC642  = 50754
CAPT1.R[0]      = 0xF5F0 = 62960
CAPT2.R[0]      = 0x0D66 = 3430
```

duty [per thousand] = ((3430- 62960)*1000)/(50754 - 62960) = 6005*1000/53329 = 112.6 (112 in device)

6.2 Calculate the signal parameters

```
CTRL1.B.PRISRC           = IP_BUS_DIVIDER[0];       //maximum resolution
CCCTRL.B.CPT1MODE = 0x2;    //capture counter by rising edge of secondary input (measure
                              signal)
CCCTRL.B.CPT2MODE = 0x1;    //capture counter by falling edge of secondary input (measure
                              signal)
CCCTRL.B.CFWM            = 0x2;    //capture flag set as soon as more than 3 values will be
                                     in FIFOs

CTRL1.B.LENGTH           = 0x0;             //continue counting to roll over
CTRL1.B.ONCE             = 0x0;             //count repeatedly
CTRL1.B.SECSRC    = channel;         //counter "channel" input pin is use for trigger the
                                       capturing - measuring signal is connect to this pin


CTRL1.B.CNTMODE   = 0x1;             //count rising edge of primary source
```

6.3 Start measurement (second part of implementation)

a) Measure

```
CCCTRL.B.ARM       = 0x1;        //enable/start capturing
 while ((STS.B.ICF1 == 0x0)||( STS.B.ICF2 == 0x0)); //wait for capture 2 cap1 values and 2 capt2
                                                      values
CCCTRL.B.ARM       = 0x0;        //disable/stop capturing

STS.B.ICF1   = 0x1;              //clear capture 1 flag
STS.B.ICF2   = 0x1;              //clear capture 2 flag
```

Read captures values from FIFOs:

```
measure[0]      = CAPT1.R;  //read first capture1 value
measure[1]      = CAPT1.R;  //read second capture 1 value
measure[2]      = CAPT2.R;  //read first capture2 value
measure[3]      = CAPT2.R;  //read second capture2 value
```

b) Calculate frequency

```
frequency [kHz]         = motor_freq [kHz]/(uint16_t)((measure[1] - measure[0]));
```

It is very important to use uint16_t data type for captured values because the counter rolls over and if the 16 bit unsigned data type is used the counter overflow is not important. It has no effect on value captured for frequency and duty. See “Calculate the signal parameters” for more details about this.

c) Calculate duty

```
duty            = (uint16_t)((measure[2] - measure[0]))*1000/(uint16_t)((measure[1] -
measure[0]));
```

Duty cycle is calculated in per thousand.

7 Description of the Green Hills project

The example codes use the eTimer 0 module. Before using any of the function it is necessary to set up the device for using the eTimer 0. The function Init_peripheral_eTimer() enables the clock for the eTimer 0 module. The function eTimer_CONFIG_PINS() sets the pin for communication with external world. Table 6 shows details about function which are in the project.

Table 6: Summary of functions which are in the project

Example name	Function name	Function parameters	unit	Description
Generating periodical signal	Generate_Signal	timer	[-]	which timer
		channel	[-]	which channel of timer
		frequency	[Hz]	frequency of the output signal
		duty	[‰]	duty of the output signal
		motor_freq	[kHz]	module frequency
Generating periodical pulse (signal which has long period but thin pulse)	Generate_Signal2	timer	[-]	which timer
		channel4period_b	[-]	base channel for period
		channel4period_0	[-]	output channel for period
		channel4pulse	[-]	channel create the pulses
		period	[ms]	period of the output signal
		width	[ns]	width of the pulse
		motor_freq	[kHz]	module frequency
Generating one-shot signal	Generate_OneShot_signal_set	timer	[-]	which timer
		channel	[-]	which channel of timer
		delay	[µs]	delay of the pulse
		width	[µs]	width of the pulse
		motor_freq	[kHz]	module frequency
		active_level	[-]	HIGH or LOW
	Start_Generate_OneShot_signal	timer	[-]	which timer
		channel	[-]	which channel of timer
Measure signal	Measure_signal	timer	[-]	which timer

Example name	Function name	Function parameters	unit	Description
parameters	_parameters_set	channel	[-]	which channel of timer
	Start_Measure _signal	timer	[-]	which timer
		channel	[-]	which channel of timer
		motor_freq	[kHz]	module frequency
		*frequency[1]	[Hz]	frequency of the measure signal
		*duty[1]	[‰]	duty of the measure signal

The project can be opened as follows:
"Location on computer"\ eTimer\build\ghs\blocks\eTimer\eTimer_sram.gpj

8 Reference

MPC5744PRM - Reference manual available at www.freescale.com

[1] These variables are returned by the function.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4793
Rev. 0, 09/2013