# LPC1768 Timer Programming Tutorial

Posted By Umang Gajera Posted date: September 18, 2017 in: Embedded No Comments


In this discussion will go through ARM Cortex-M3 LPC1768 Timer Tutorial. In a previous LPC1768 programming tutorial we saw a blinky example using GPIO and harcoded delays, now its time to improvise and use precise delay using timers! LPC1768/LPC1769 has four 32-bit Timer blocks. Each Timer block can be used as a 'Timer' (like for e.g. triggering an interrupt every 't' microseconds) or as a 'Counter' and can be also used to demodulate PWM signals given as input.

Each Timer module has its own **Timer Counter(TC)** and **Prescale Register(PR)** associated with it. When a Timer is Reset and Enabled, the TC is set to 0 and incremented by 1 every **'PR+1'** clock cycles – where PR is the value stored in Prescale Register. When it reaches its maximum value it gets reset to 0 and hence restarts counting. Prescale Register is used to define the **resolution** of the timer. If **PR is 0** then TC is incremented every **1 clock cycle of the peripheral clock**. If **PR=1** then TC is incremented every **2 clock cycles of peripheral clock and so on**. By setting an appropriate value in PR we can make timer increment or count : every peripheral clock cycle or 1 microsecond or 1 millisecond or 1 second and so on.

Each Timer has **four 32-bit Match Registers** and **four 32-bit Capture Registers**. Timer 0,1,3 have two Match outputs while Timer 2 has four.

## Match Register

A Match Register is a Register which contains a specific value set by the user. When the Timer starts – every time after TC is incremented the value in TC is compared with match register. If it matches then it can Reset the Timer or can generate an interrupt as defined by the user. We are only concerned with match registers in this tutorial.

**Match Registers can be used to:**

- Stop Timer on Match(i.e when the value in count register is same as than in Match register) and trigger an optional interrupt.
- Reset Timer on Match and trigger an optional interrupt.
- To count continuously and trigger an interrupt on match.

## External Match Output

When a corresponding Match register(MRx) equals the Timer Counter(TC) the match output can be controlled using External Match Register(EMR) to : either toggle, go HIGH, go LOW or do nothing.

### Capture Register

As the name suggests it is used to Capture Input signal. When a transition event occurs on a Capture pin , it can be used to copy the value of TC into any of the 4 Capture Register or to genreate an Interrupt. Hence these can be also used to demodulated PWM signals. We are not going to use them in this tutorial since we are only concerned with using Timer block as a 'Timer'. We'll see them in a later tutorial.

## Registers used for LPC1768 Timer Programming

We will be using CMSIS based `lpc17xx.h` header file for programming. In CMSIS, all the Registers used to program and use timers are defined as members of structure(pointer) `LPC_TIMx` where x is the Timer module from 0 to 3. So for Timer1 we will use `LPC_TIM0` and so on. Registers can be accessed by de-referecing the pointer using "->" operator. For, example we can access TCR of Timer0 block as `LPC_TIM0->TCR`.

**Now lets see some of the main registers concerned mainly with timer operation.**

**1) PR : Prescale Register (32 bit)** – Stores the maximum value of Prescale counter after which it is reset.

**2) PC : Prescale Counter Register (32 bit)** – This register increments on every PCLK(Peripheral clock). This register controls the resolution of the timer. When PC reaches the value in PR , PC is reset back to 0 and Timer Counter is incremented by 1. Hence if PR=0 then Timer Counter Increments on every 1 PCLK. If PR=9 then Timer Counter Increments on every 10th cycle of PCLK. Hence by selecting an appropriate prescale value we can control the resolution of the timer.

**3) TC : Timer Counter Register (32 bit)** – This is the main counting register. Timer Counter increments when PC reaches its maximum value as specified by PR. If timer is not reset explicitly(directly) or by using an interrupt then it will act as a free running counter which resets back to zero when it reaches its maximum value which is 0xFFFFFFFF.

**4) TCR : Timer Control Register** – This register is used to enable , disable and reset TC. When bit0 is 1 timer is enabled and when 0 it is disabled. When bit1 is set to 1 TC and PC are set to zero together in sync on the next positive edge of PCLK. Rest of the bits of TCR are reserved.

**5) CTCR : Count Control register** – Used to select Timer/Counter Mode. For our purpose we are always gonna use this in Timer Mode. When the value of the CTCR is set to 0x0 Timer Mode is selected.

**6) MCR : Match Control register** – This register is used to control which all operations can be done when the value in MR matches the value in TC. Bits 0,1,2 are for MR0 , Bits 3,4,5 for MR1 and so on.. Heres a quick table which shows the usage:

**For MR0:**

- Bit 0: Interrupt on MR0 i.e. trigger an interrupt when MR0 matches TC. Interrupts are enabled when set to 1 and disabled when set to 0.
- Bit 1: Reset on MR0. When set to 1 , TC will be reset when it matched MR0. Disabled when set to 0.
- Bit 2: Stop on MR0. When set to 1 , TC & PC will stop when MR0 matches TC.

Similarly bits 3-5 , 6-8 , 9-11 are for MR1 , MR2 , MR3 respectively.

**7) IR : Interrupt Register** – It contains the interrupt flags for 4 match and 4 capture interrupts. Bit0 to bit3 are for MR0 to MR3 interrupts respectively. And similarly the next 4 for CR0-3 interrupts. when an interrupt is raised the corresponding bit in IR will be set to 1 and 0 otherwise. Writing a 1 to the corresponding bit location will reset the interrupt – which is used to acknowledge the completion of the corresponding ISR execution.

**8) EMR : External Match Register** – It provides both status and control of External Match Output Pins. First four bits are for EM0 to EM3. Next 8 bits are for EMC0 to EMC3 in pairs of 2.

- Bit 0 – EM0: External Match 0. When a match occurs between TC and MR0, depending on bits[5:4] i.e. EMC0 of this register, this bit can either toggle, go LOW, go HIGH, or do nothing. This bit is driven to MATx.0 where x=Timer number.
- Similarly for Bits 1, 2 & 3.
- Bits[5:4] – EMC0: External Match 0. The values in these bits select the functionality of EM0 as follows:
  - 0x0 – Do nothing
  - 0x1 – Clear the corresponding External Match output to 0 (MATx.m pin is LOW).
  - 0x2 – Set the corresponding External Match output to 1 (MATx.m pin is HIGH).
  - 0x3 – Toggle the corresponding External Match output.
- Similarly for Bits[7:6] – EMC1, Bits[9,8] – EMC2, Bits[11:10] – EMC3.

**Note:** Timer0/1/3 have only two Match outputs Pinned while Timer2 has four Match output Pinned. Hence EM2,EM3 and EMC2,EMC3 are not applicable for Timer0/1/3.

Now lets actually use a Timer and see it in action.

## Setting up & configuring Timers in LPC176x

To use timers we need to first configure them. We need to set appropriate values in CTCR , IR , PR registers and reset PC , TC registers. Finally we assign TCR = 0x01 which enables the timer.

I would recommend to use the following sequence for Setting up Timers:

1. Set appropriate value in `LPC_TIMx->CTCR`
2. Define the Prescale value in `LPC_TIMx->PR`
3. Set Value(s) in Match Register(s) if required
4. Set appropriate value in `LPC_TIMx->MCR` if using Match registers / Interrupts
5. Reset Timer – Which resets PR and TC
6. Set `LPC_TIMx->TCR` to `0x01` to Enable the Timer when required
7. Reset `LPC_TIMx->TCR` to `0x00` to Disable the Timer when required

Setting the Peripheral Clock for Timer: The input clock for timers can be set using peripheral clock selection registers **PCLKSEL0 & PCLKSEL1** .

- For Timer0 bits[3:2] in PCLKSEL0 are used.
- For Timer1 bits[5:4] in PCLKSEL0 are used.
- For Timer2 bits[13:12] in PCLKSEL1 are used.
- For Timer3 bits[15:14] in PCLKSEL1 are used.

These bits allows us to choose 4 different CCLK(SystemCoreClock in startup code) dividers to get the final PCLK as follows :

- [00] – PCLK = CCLK/4 (Default after reset)
- [01] – PCLK = CCLK
- [10] – PCLK = CCLK/2
- [11] – PCLK = CCLK/8

### LPC1768 Timer Prescaler Calculations:

The delay or time required for 1 clock cycle when PCLK = 'X' Mhz is given by :

$$T_{PCLK} = \frac{1}{PCLK_{Hz}} = \frac{1}{X * 10^6} \text{ Seconds}$$

It is also the maximum resolution Timer block can proved at a given PCLK frequency of X Mhz. The general formula for Timer resolution at X Mhz PCLK and a given value for prescale (PR) is as given below:

$$T_{RES} = \frac{PR+1}{PCLK_{Hz}} = \frac{PR+1}{X * 10^6} \text{ Seconds}$$

Hence, we get the **Formula for Prescaler (PR) for required Timer resolution ($T_{RES}$ in Secs)** at given **PCLK(in Hz)** frequency as:

$$PR = (PCLK_{Hz} * T_{RES}) - 1$$

$$PR = ((X * 10^6) * T_{RES}) - 1$$

Note that here, the resolution is also the time delay required to increment TC by 1.

Hence, Prescaler value for 1 micro-second resolution/ 1us time delay at 25 Mhz PCLK is,

$$PR_{1us} = (25Mhz * 1uS) - 1 = (25*10^6 * 10^{-6}) - 1 = 24$$

Prescale for 1 mS (milli-second) resolution at 25Mhz PCLK is,

$$PR_{1ms} = (25Mhz * 1ms) - 1 = (25*10^6 * 10^{-3}) - 1 = 24999$$

The maximum resolution of all the timers is 10 nano-seconds when using PCLK = CCLK = 100Mhz and PR=0 which is as follows,

$$T_{MAXRES} = [1 / (100Mhz)] = 10ns$$

Now lets implement to basic function required for Timer Operation:

1. `void initTimer0(void);`
2. `void delayMS(unsigned int milliseconds);`

### #1) initTimer0(void); [Used in Example #1]

> **Attention Plz! :** This function is used to setup and initialize the Timer block. Timer blocks use peripheral clock as their input and hence peripheral clock must be initialized before Timer is initialized. In our case it is assumed that LPC1768 CPU Clock (CCLK) is set at 100Mhz and Peripheral Clock (PCLK) is set to CCLK/4 i.e. 25Mhz. Note that these clocks are default which are configured by the startup code.

```
#define PRESCALE (25000-1)

void initTimer0(void)
{
   /*Assuming that PLL0 has been setup with CCLK = 100Mhz and PCLK = 25Mhz.*/
   LPC_SC->PCONP |= (1<<1); //Power up TIM0. By default TIM0 and TIM1 are enabled.
   LPC_SC->PCLKSEL0 &= ~(0x3<<3); //Set PCLK for timer = CCLK/4 = 100/4 (default)

   LPC_TIM0->CTCR = 0x0;
   LPC_TIM0->PR = PRESCALE; //Increment TC at every 24999+1 clock cycles
   //25000 clock cycles @25Mhz = 1 mS

   LPC_TIM0->TCR = 0x02; //Reset Timer
}
```

### #2) delayMS(unsigned int milliseconds); – LPC1768 Timer Delay Function

```
void delayMS(unsigned int milliseconds) //Using Timer0
{
   LPC_TIM0->TCR = 0x02; //Reset Timer

   LPC_TIM0->TCR = 0x01; //Enable timer

   while(LPC_TIM0->TC < milliseconds); //wait until timer counter reaches the desired de

   LPC_TIM0->TCR = 0x00; //Disable timer
}
```

## Real World LPC1768/LPC1769 Timer Examples with sample code

### Example 1) – Simple Blinky Example using Timer & GPIO

Now lets write a C/C++ blinky program which flashes a LED every half a second. Since 0.5 second = 500 millisecond we will invoke timer delay function 'delayMS' as `delayMS(500)`. The LED is connected to Pin P0.22.

```
/*(C) Umang Gajera - www.ocfreaks.com 2011-17.
More Embedded tutorials @ www.ocfreaks.com/cat/embedded/
LPC1768 Basic Timer example.*/

#include <lpc17xx.h>

#define PRESCALE (25000-1) //25000 PCLK clock cycles to increment TC by 1

void delayMS(unsigned int milliseconds);
void initTimer0(void);

int main(void)
{
   //SystemInit(); //called by Startup Code before main(), hence no need to call agai
   initTimer0(); //Initialize Timer0
   LPC_GPIO0->FIODIR = (1<<22); //Configure P0.22 as output

   while(1)
   {
      LPC_GPIO0->FIOSET = (1<<22); //Turn ON LED
      delayMS(500); //0.5 Second(s) Delay
      LPC_GPIO0->FIOCLR = (1<<22); //Turn LED OFF
      delayMS(500);
   }

   //return 0; //normally this wont execute ever
}
```

**Project Source for Example #1 on GitHub @** LPC1768 Timer Example 1 [Successfully tested on Keil uV5.23], **Download Project Zip @** Example_1.Zip

### Example 2) – Blinky example code using Timer & Match Outputs

In this C/C++ Example we will use Match outputs 0 and 1 of Timer 0 i.e. MAT0.0 and MAT0.1. MAT0.0 is pinned to P1.28 and MAT0.1 pinned to P1.29. You can connect LED to any of the Match output pin (P1.28 or P1.29) and see the code in action.

```
/*(C) Umang Gajera - http://www.ocfreaks.com 2011-17.
More Embedded tutorials @ http://www.ocfreaks.com/cat/embedded/
LPC1768 Timer example 2.*/

#include <lpc17xx.h>

#define PRESCALE (25000-1) //25000 PCLK clock cycles to increment TC by 1

void initTimer0(void);

int main (void)
{
   //SystemInit(); //called by Startup Code before main(), hence no need to call agai
   initTimer0(); //Initialize Timer0
   LPC_PINCON->PINSEL3 |= (1<<24) | (1<<25) | (1<<27) | (1<<26); //config MAT0.0(P1.2

   initTimer0();

   while(1)
   {
      //Idle loop
   }
   //return 0; //normally this won't execute
}

void initTimer0(void)
```

**Project Source for Example #2 on GitHub @** LPC1768 Timer Example 2 [Successfully tested on Keil uV5.23], **Download Project Zip @** Example_2.Zip

### Example 3) – LPC1768 Timer Interrupt Example Code

Here we will use a timer interrupt function which will be called periodically to blink an LED. The Timer Interrupt Service Routine (ISR) will toggle P1.29 every time it is called. If you are using C++ file then you will need to add `extern "C"` before the ISR definition or C++ linker won't be able to link it properly to generate finally binary/hex file. For C code this is not required.

**Attention Plz! :** Please take your time to go through "initTimer0()" function. Here I've setup LPC176x Timer Interrupt handler which gets triggered when value in TC equals the value in MR0. I will discuss Interrupts in an upcoming tutorial.

```
/*(C) Umang Gajera - http://www.ocfreaks.com 2011-17.
More Embedded tutorials @ http://www.ocfreaks.com/cat/embedded/
LPC1768 Timer example 3.
*/

#include <lpc17xx.h>

#define PRESCALE (25000-1) //25000 PCLK clock cycles to increment TC by 1

void initTimer0();

int main(void)
{
   //SystemInit(); //called by Startup Code before main(), hence no need to call agai
   LPC_GPIO1->FIODIR |= (1<<29); //set P1.29 as output
   initTimer0();

   while(1)
   {
      //Idle loop
   }
   //return 0; //normally this won't execute
}


void initTimer0(void)
```

**Project Source for Example #3 on GitHub @** LPC1768 Timer Example 3 [Successfully tested on Keil uV5.23], **Download Project Zip @** Example_3.Zip

Share this:

Share



Tags: lpc1768 tutorial

Like Tweet Share Share Share

« Previous
LPC1343 GPIO Programming Tutorial

Next »
LPC1768 PWM Programming Tutorial

**LPC1768 ADC Programming Tutorial**

**LPC1768 Timer Input Capture & Frequency...**

**LPC1768 UART Programming Tutorial**

**LPC1768 GPIO Programming Tutorial**

**LPC1768 PWM Programming Tutorial**

**LPC21 Progra Tutori**

## ABOUT THE AUTHOR

Umang Gajera

**Charlie Lee** • 2 years ago

LPC_SC->PCLKSEL0 |= (1<<3); //Set PCLK for timer = CCLK/4 = 100/4 (default)
If the setting is what we want it should correct to it as below
LPC_SC->PCLKSEL0 |= (1<<3); //Set PCLK for timer = CCLK/2 = 100/2

Thanks your post

Reply • Share ›

**Power_user_EX** Mod ➚ Charlie Lee • 2 years ago

Thanks for pointing out the error Charlie. I have made the corrections. For the examples given in the tutorial we want the timer PCLK running @ 25Mhz i.e. 1/4th of CCLK (100Mhz). Hence, the correction will be as follows:

```
LPC_SC->PCLKSEL0 &= ~(0x3<<3); //Set PCLK for timer = CCLK/4 = 100/4 (default)
```

Note that this is not required since timer PCLK is 1/4th by default after a System Reset.

2 1 • Reply • Share ›