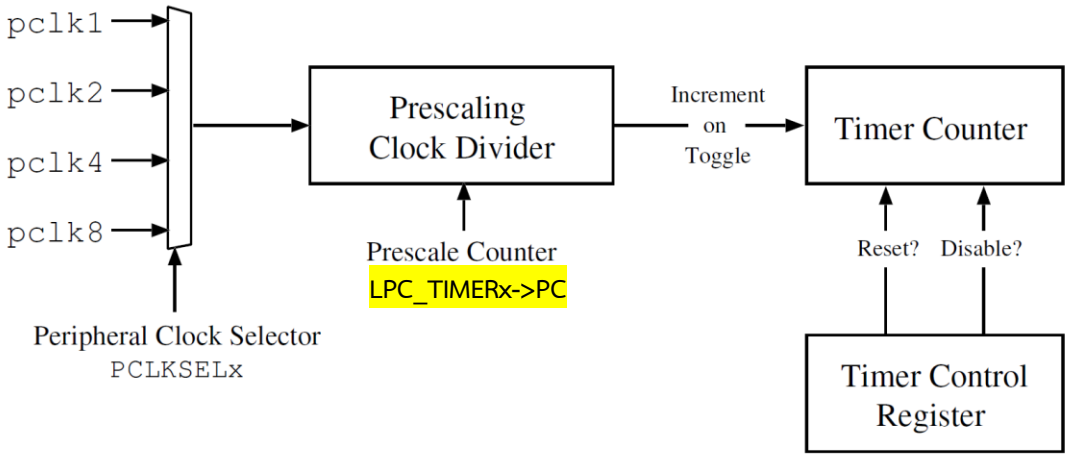
Name: ..... Student ID: .....
Name: ..... Student ID: .....

Laboratory 05
Introduction to Timer 1

1 Basic Description

Timers are peripherals within the LPC that are mainly internal, they use the CPU clock to keep track of time, and make that ability available to the user in a number of ways. Timers can send out periodic events, make very precise measurements, simply make the time available for your applications, among other things. There are four timers within the LPC, Tim0, Tim1, Tim2 and Tim3. All are identical, but can have options set independently, and can be used without interfering with each other.

2. Speed Settings



All timers are built around an internal *Timer Counter (TC)*, which is a 32 bit register that is incremented periodically. The rate of change is derived from the current speed of the CPU clock, which peripheral clock you have connected up and what the prescale counter is set to.

2.1 Powering Devices

Before we can get to choosing the peripheral clock, and setting the prescale register, we need to actually turn on the timer. On reset most of the LPC's peripherals are off, and are not being supplied power by the microcontroller. This can save a lot of energy, but a few core peripherals are turned on when the LPC starts, among these GPIO and Tim0 and Tim1.

Additionally, if you do not need Tim0 or Tim1, you can turn them off to save some power. Power control is considered a system feature, and is controlled by register LPC_SYSCTL->PCONP. Each bit in that register is assigned to a peripheral, with a 0 meaning unpowered, and a 1 meaning that the peripheral is powered.

```
void Chip_TIMER_Init(LPC_TIMER_T *pTMR);// Initialize a Timer

Chip_TIMER_Init(LPC_TIMER0);      // Initialize a first Timer
void Chip_TIMER_DeInit(LPC_TIMER_T *pTMR); // Shutdown a Timer

Chip_TIMER_DeInit(LPC_TIMER0);// Shutdown a first Timer

uint32_t timerFreq = Chip_Clock_GetSystemClockRate(); // Return system
                                                      clock rate
```

2.2 Setting the Prescale Counter

Once you have made sure your chosen timer is on, and is using the peripheral clock you want, you can move onto setting the prescale counter and actually using it to perform timing related tasks. The prescale counter is basically a 32 bit factor register inside a clock divider. You can set it using the LPC_TIMERx->PC register.

```
void Chip_TIMER_PrescaleSet(LPC_TIMER_T *pTMR, uint32_t prescale)

LPC_TIMER0->PC = 14; // Divide the incoming clock by 15.
```

4.2.4 Reset and Enable

Before the timer counter can actually start incrementing, there are two flags within the *Timer ControlRegister (TCR)*.

The enable flag, when set to one, disables the timer counter's ability to increment.

```
Chip_TIMER_Enable(LPC_TIMER0); // Enable the timer counter
Chip_TIMER_Disable(LPC_TIMER0);//Disable the timer counter, the coun-
                               ter's value is stuck whenever we
                               stopped it.
```

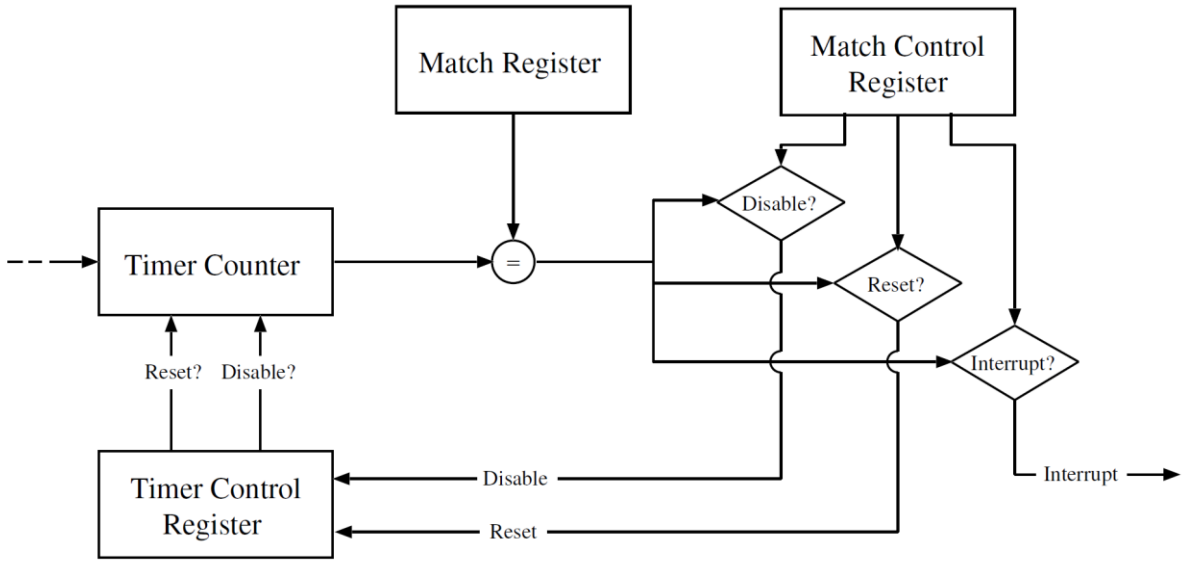
The reset function forces the timer counter's value to zero.

```
Chip_TIMER_Reset(LPC_TIMER0) // the counter's value is zero
```

Once you are done with setting those value properly, you can watch your timer counter increment.

```
uint32_t Chip_TIMER_ReadCount(LPC_TIMER_T *pTMR)
uint32_t Chip_TIMER_ReadPrescale(LPC_TIMER_T *pTMR)
```

2.3 Match Registers



Within each timer are four match registers, 32 bit registers which can store a specific match value. When the timer counter and match register are equal, certain events can be triggered.

Choosing the match value is just a matter of setting the match register, LPC_TIMERx->MRx.

2.3.1 Match Settings

There are three events a match can trigger, a reset of the timer counter, disabling the timer counter and sending an interrupt. The disable and reset events work by modifying the timer control register, and using its features to control the timer counter.

The match register's disable event sets the disable flag in the TCR to 1, requiring you to manually toggle it back. On the other hand, the reset event simply toggles the flag in the TCR, setting the timer counter to 0, but allowing it to continue incrementing,

To actually change which events are triggered on match requires manipulating the register at LPC_TIMERx->MCR.

```
Chip_TIMER_Reset(LPC_TIMER0);
Chip_TIMER_MatchEnableInt(LPC_TIMER0, 1); // Match Timer #2
Chip_TIMER_SetMatch(LPC_TIMER0,1, (timerFreq / TICKRATE_HZ));
Chip_TIMER_ResetOnMatchEnable(LPC_TIMER0, 1);
Chip_TIMER_Enable(LPC_TIMER0);
```

2.3.2 Match Interrupts

Like a GPIO interrupt, there are multiple triggers which all trigger an interrupt on the same channel. Each of the match registers can trigger interrupts, and so can other components of each timer.

To tell these interrupts apart, you can use each timer's *Interrupt Register (IR)*. When a component of the timer triggers the interrupt a flag in the IR is flipped and you can use that to determine what is already going on.

```
void TIMER0_IRQHandler(void)//Hooked function for IRQ TIMER0
{
      if (Chip_TIMER_MatchPending(LPC_TIMER0, 1)) {
            Chip_TIMER_ClearMatch(LPC_TIMER0, 1);
            // do whatever you want
      }
}


      /* Enable timer interrupt */
      NVIC_ClearPendingIRQ(TIMER0_IRQn);
      NVIC_EnableIRQ(TIMER0_IRQn);
```

3. Example program for Lab05

```
void TIMER0_IRQHandler(void)
{
      if (Chip_TIMER_MatchPending(LPC_TIMER0, 1)) {
            Chip_TIMER_ClearMatch(LPC_TIMER0, 1);
            // ….
      }
}

int main(void)
{
      uint32_t timerFreq;
      SystemCoreClockUpdate();
      Board_Init();

      Chip_TIMER_Init(LPC_TIMER0);
      timerFreq = Chip_Clock_GetSystemClockRate();

      Chip_TIMER_Reset(LPC_TIMER0);
      Chip_TIMER_MatchEnableInt(LPC_TIMER0, 1);
      Chip_TIMER_SetMatch(LPC_TIMER0, 1, (timerFreq / TICKRATE_HZ1));
      Chip_TIMER_ResetOnMatchEnable(LPC_TIMER0, 1);
      Chip_TIMER_Enable(LPC_TIMER0);

      NVIC_ClearPendingIRQ(TIMER0_IRQn);
      NVIC_EnableIRQ(TIMER0_IRQn);

      while (1) {
            __WFI();    // Wait for Interrupt
      }
      return 0;
}
```

4. Experiment

1) Write a program that make LED 1 Flash at frequency 5Hz using timer.

2) Write a program to produce a square wave at frequency 1 kHz on one of the External Match Output pins. Measure its frequency with an oscilloscope.
   Hint. Since the External Match Outputs are shared pins, you need to set PINSEL and PINMODE Register. You also need to set External Match Register (EMR) to control functionality of this output.

5. Appendix

Further information on address pointer and register can be found in following files

lpc_chip_175x_6x.h Peripheral address and register set declarations
timer_17xx_40xx.h Standard timer register block structures
sysclt_17xx_40xx.h Clock and Power register block structures

6. Reference

[1]UM10360 LPC176x/5x User manual, NXP, April 2014
[2]Rohit Ramesh, Introduction to Embedded Systems Development.
[3]Rob Toulson and Tim Wilmshurst, Fast and Effective Embedded Systems Design, Elsevier, 2012