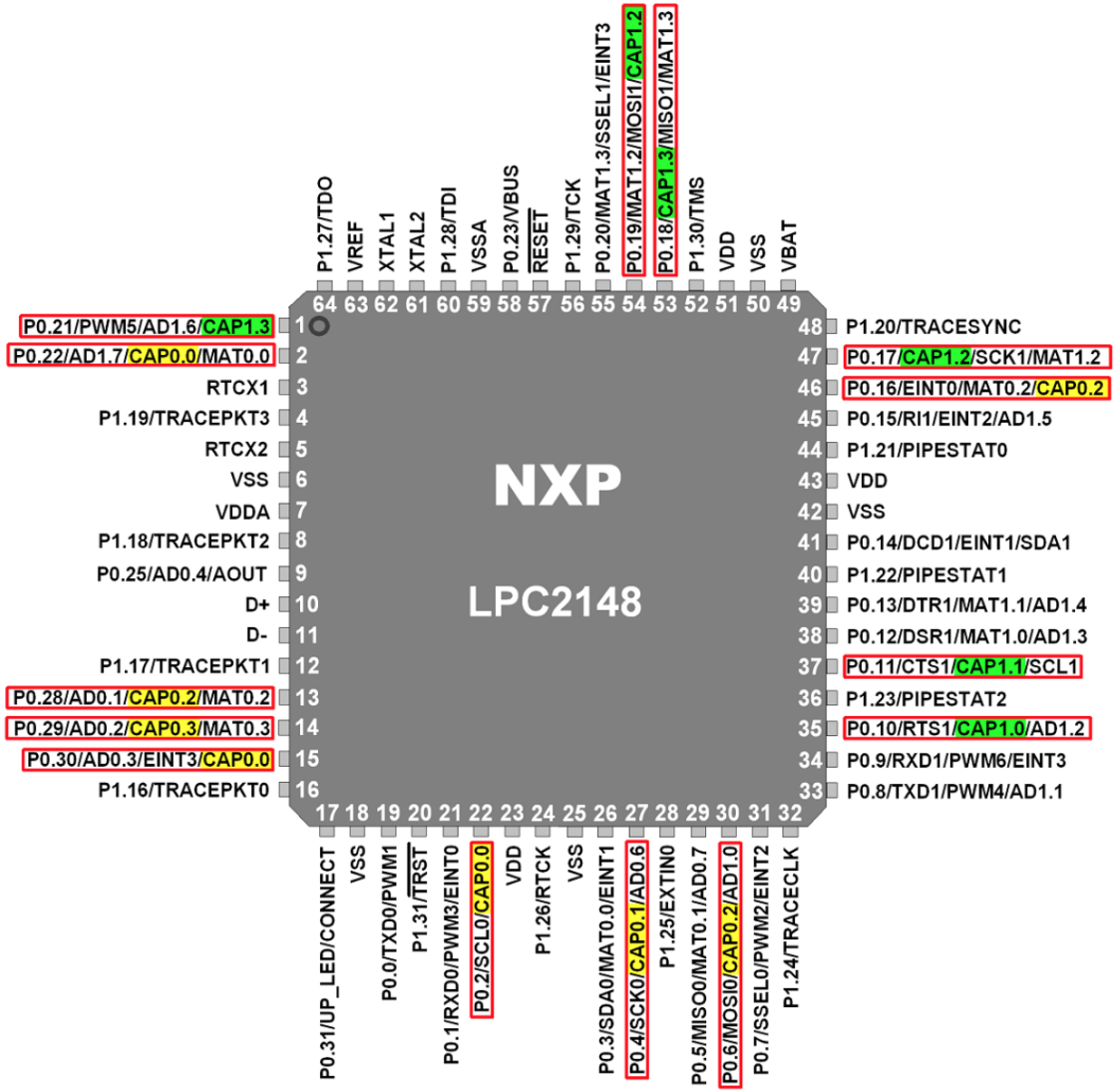
LPC2148 Timer/Counter

- LPC2148 Timer/Counter | ARM7-LPC2148 (electronicwings.com)

- **Timer**
  - specific type of clock which is used to measure the time intervals.
  - It provides/measures the time interval by counting the input clocks
- **Counter**
  - similar to Timers but works in a reverse manner to the timers.
  - It counts the external events or we can say external clock ticks.
  - It is mostly used to measure frequency from the counts of clock ticks.
- Timer and Counter functionally same, except timer uses the peripheral clock (PCLK) for its timing, while counter uses an external source
- A counter is called a capture timer – used to count external event via capture input

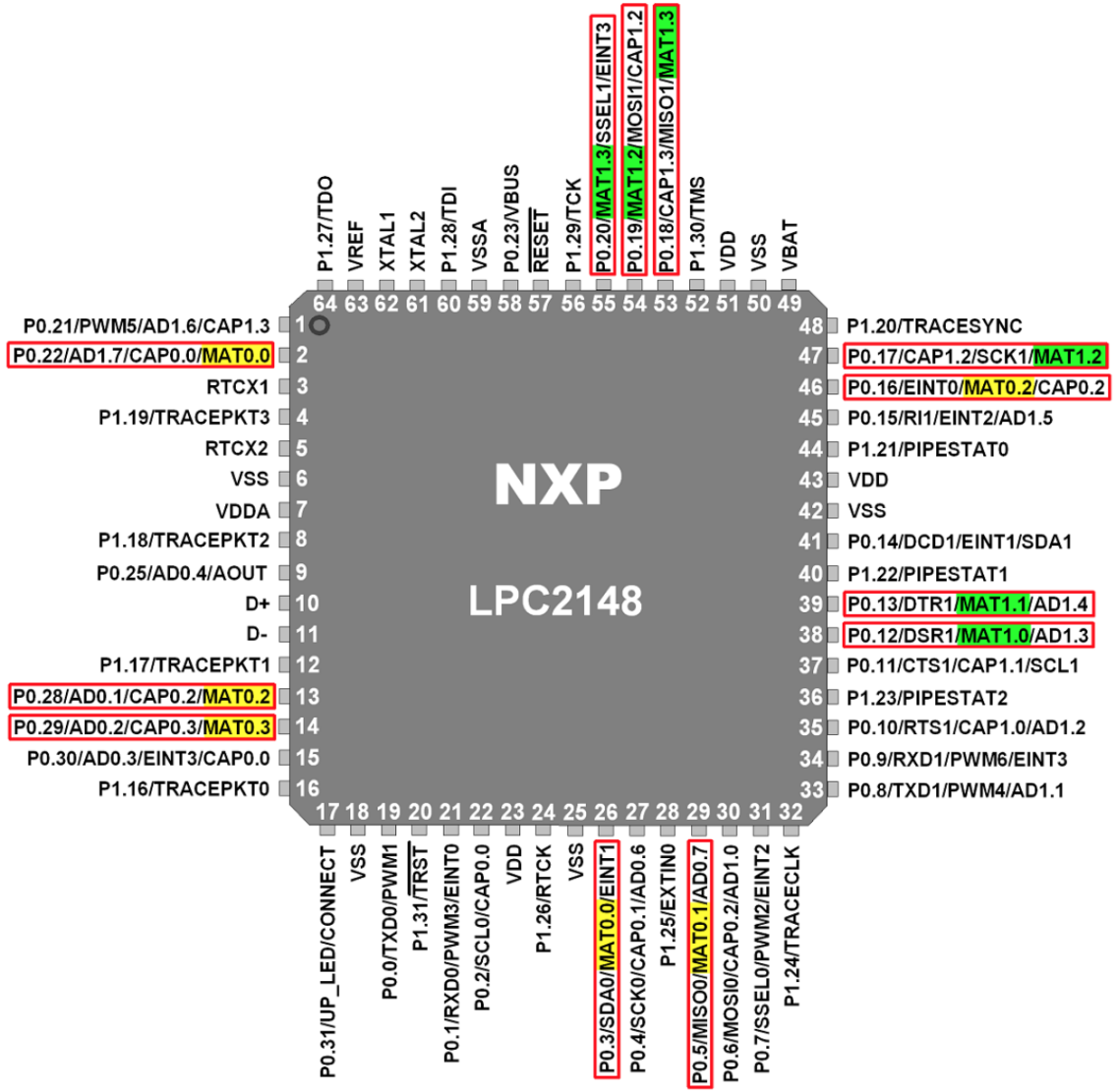
- LPC2148 Timer has input of peripheral clock (PCLK) or an external clock. It counts the clock from either of these clock sources for its operation.
- LPC2148 Timer/Counter can generate an interrupt signal at specified time value.
- LPC2148 has match registers that contain count value which is continuously compared with the value of the Timer register. When the value in the Timer register matches the value in the match register, specific action (timer reset, or timer stop, or generate an interrupt) is taken.
- LPC2148 has capture registers which can be used to capture the timer value on a specific external event on capture pins.

Timer Capture Mode



- LPC2148 Timer includes input capture feature, using which we can capture/trap the timer counter value on events like rising edge (positive going) and falling edge (negative going). With this function, we can measure frequency, pulse width, duty cycle of the input signal
- The Capture Control Register decides which event is to be recorded (e.g. rising/falling edge or both) and whether an interrupt is to be generated.
- On each capture event on a capture pin, corresponding capture register gets copied with the TC (Timer Counter) value

Control Timer and output - Match



- This 32-bit value is written in Match Register (MR) and continuously compared with 32-bit Timer Counter (TC) value.
- When this match register value and Timer Counter (TC) value are equal, we can reset/stop Timer Counter (TC) and can generate an interrupt.
- Timer counter (TC) counts the input PCLK (Peripheral Clock) i.e. one count of Timer Counter (TC) is equal to the one cycle of PCLK
- We can also extend one count of Timer Counter by several PCLK clocks using the Prescale Counter (PC). Prescale Register (PR) is used to set the maximum value for PC.
- We can make specific pins (also called as match pins) of LPC2148 High/Low/Toggle on the event match occurrence.

- 2 - 32-bit Timers – Timer0/Counter0 & Timer1/Counter1
- Special Function Registers for Timer 0
  - Timer Count Register- T0TC
    - 32-bit registers
    - Range of counting from 0 to 0xFFFFFFFF and then wraps back to the 0x00000000
    - This register is incremented on every tick of the clock (i.e. PCLK), if the prescale counter is made 0
  - Timer Control Register- T0TCR
    - 8-bit register
    - Only lowest 2 bits are used
    - Bit 0- Enable bit
      - 1 : counter is enabled and starts. T0TC register is incremented for every clock cycle of PCLK
    - Bit 1- Reset bit
      - 1: the counter reset on the next positive edge of PCLK

- Match Registers (MR0 to MR3)
  - 4 -32-bit match registers are available : MR0 to MR3
  - One match register may be sufficient for one timer operation
  - Loading number into the match register
- Match Control Register – T0MCR
  - 16-bit register, used to specify an event to occur when match occurs
  - Bit 0 – I : When 1 – an interrupt is activated when match occurs

    When 0 – the interrupt is disabled
  - Bit 1 – R : When 1 – timer count register (T0TC) is reset when match occurs

    When 0 – this feature is disabled
  - Bit 2 – S : When 1 – time count (T0TC) and the pre-scale counter will be stopped when match occurs, also Enable bit of the T0TCR is made 0

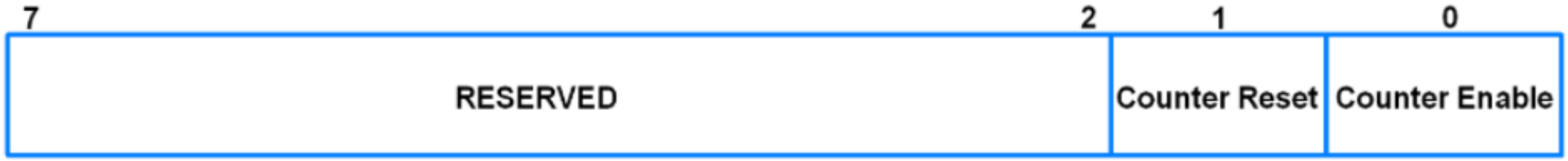
1. **T0IR (Timer0 Interrupt Register)**

   - It is an 8-bit read-write register.
   - Consists of 4 bits for match register interrupts and 4 bits for compare register interrupts.
   - If interrupt is generated, then the corresponding bit in this register will be high, otherwise it will be low.
   - Writing a 1 to any bit of this register will reset that interrupt. Writing a 0 has no effect.

T0IR (Timer0 Interrupt Register)

2. **T0TCR (Timer0 Timer Control Register)**

- It is an 8-bit read-write register.
- It is used to control the operation of the timer counter.



T0TCR (Timer0 Timer Control Register)

- **Bit 0 – Counter Enable**

  0 = Counters are disabled

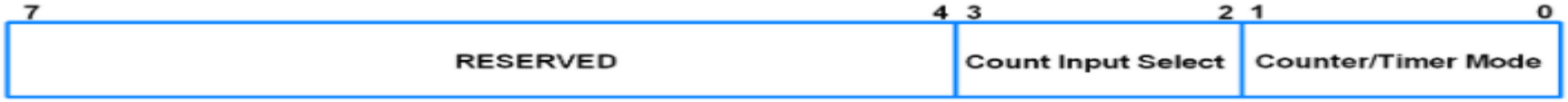
  1 = Timer counter and Prescale counter are enabled for counting
- **Bit 1 – Counter Reset**

  0 = Counter not reset

  1 = Timer counter and Prescale counter are synchronously reset on next positive edge of PCLK

3. **T0CTCR (Timer0 Counter Control Register)**

- It is an 8-bit read-write register.



T0CTCR (Timer0 Counter Control Register)

- Used to select between timer counter mode.
- When in counter mode, it is used to select the pin and edges for counting.
- **Bits 1:0 – Counter/Timer Mode**
  This field selects which rising edges of PCLK can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC).
  00 = Timer Mode: Every rising edge of PCLK
  01 = Counter Mode: TC is incremented on rising edge on the capture input selected by Bits 3:2.
  10 = Counter Mode: TC is incremented on falling edge on the capture input selected by Bits 3:2
  01 = Counter Mode: TC is incremented on both edges on the capture input selected by Bits 3:2
- **Bits 3:2 – Count Input Select**
  When bits 1:0 in this register are not 00, these bits select which capture pin is sampled for clocking.
  00 = CAP0.0
  01 = CAP0.1
  10 = CAP0.2
  11 = CAP0.3

1. **T0TC (Timer0 Timer Counter)**

· It is a 32-bit timer counter.

· It is incremented when the Prescale Counter (PC) reaches its maximum value held by Prescaler Register (PR).

Note : When TC overflow occurs, it does not generate any overflow interrupt. Alternatively, we can use match register to detect overflow event if needed.

4. **T0PR (Timer0 Prescale Register)**

   - It is a 32-bit register.
   - It holds the maximum value of the Prescale Counter.

5. **T0PC (Timer0 Prescale Counter Register)**

   - It is a 32-bit register.
   - It controls the division of PCLK by some constant value before it is applied to the Timer Counter.
   - It is incremented on every PCLK.
   - When it reaches the value in Prescale Register, the Timer Counter is incremented and Prescale Counter is reset on next PCLK.

6. **T0MR0-T0MR3 (Timer0 Match Registers)**

   - These are 32-bit registers.
   - The values stored in these registers are continuously compared with the Timer Counter value.
   - When the two values are equal, the timer can be reset or stop or an interrupt may be generated. The T0MCR controls what action should be taken on a match.

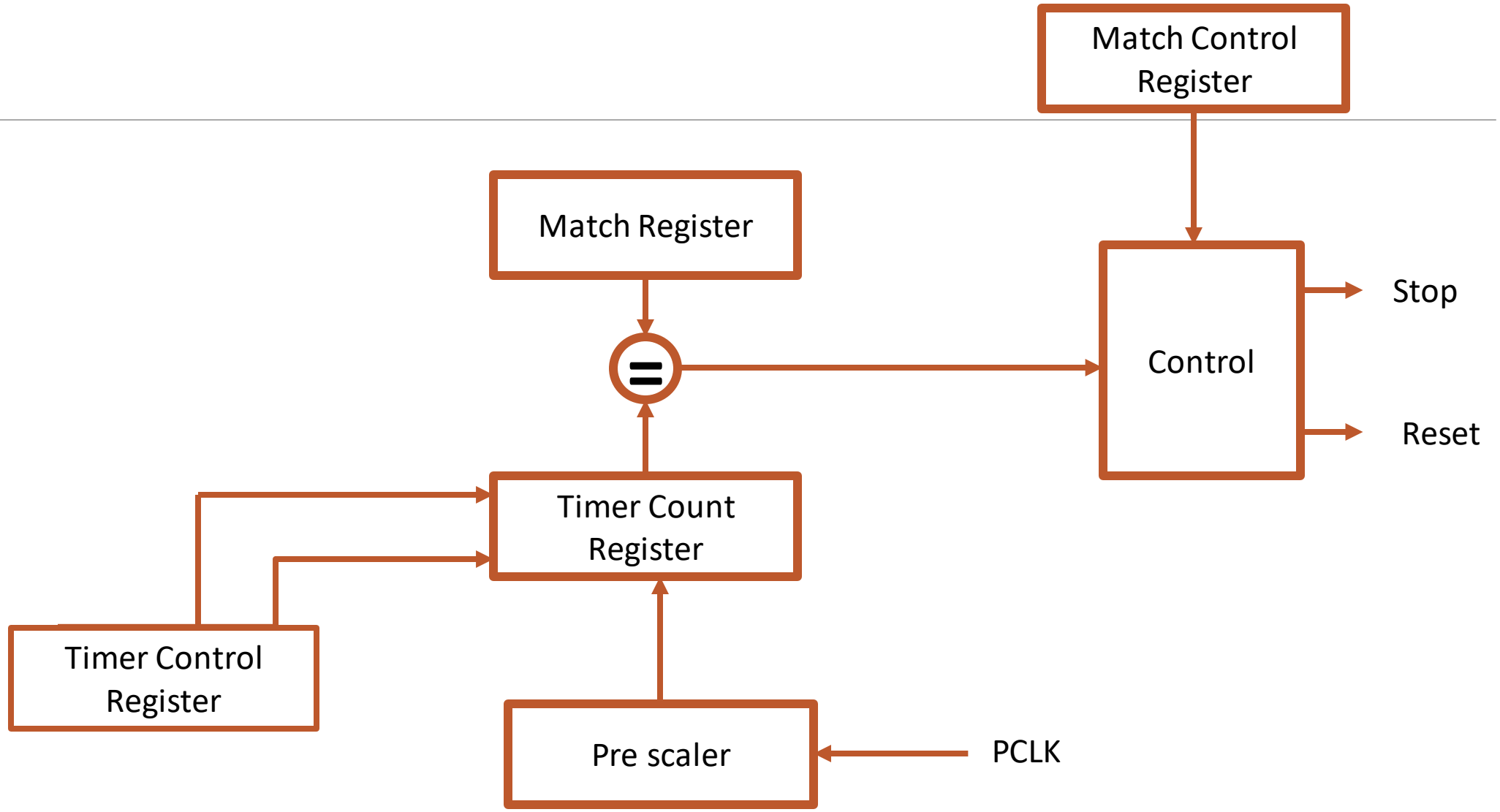
7. **T0MCR (Timer0 Match Control Register)**

- It is a 16-bit register.
- It controls what action is to be taken on a match between the Match Registers and Timer Counter.

T0MCR (Timer0 Match Control Register)

- **Bit 0 – MR0I (Match register 0 interrupt)**
  0 = This interrupt is disabled
  1 = Interrupt on MR0. An interrupt is generated when MR0 matches the value in TC (Timer Counter)
- **Bit 1 – MR0R (Match register 0 reset)**
  0 = This feature is disabled
  1 = Reset on MR0. The TC (Timer Counter) will be reset if MR0 matches it
- **Bit 2 – MR0S (Match register 0 stop)**
  0 = This feature is disabled
  1 = Stop on MR0. The TC (Timer Counter) and PC (Prescale Counter) is stopped and Counter Enable bit in T0TCR is set to 0 if MR0 matches TC
- MR1, MR2 and MR3 bits function in the same manner as MR0 bits.

Simplified Block Diagram of Timer Unit



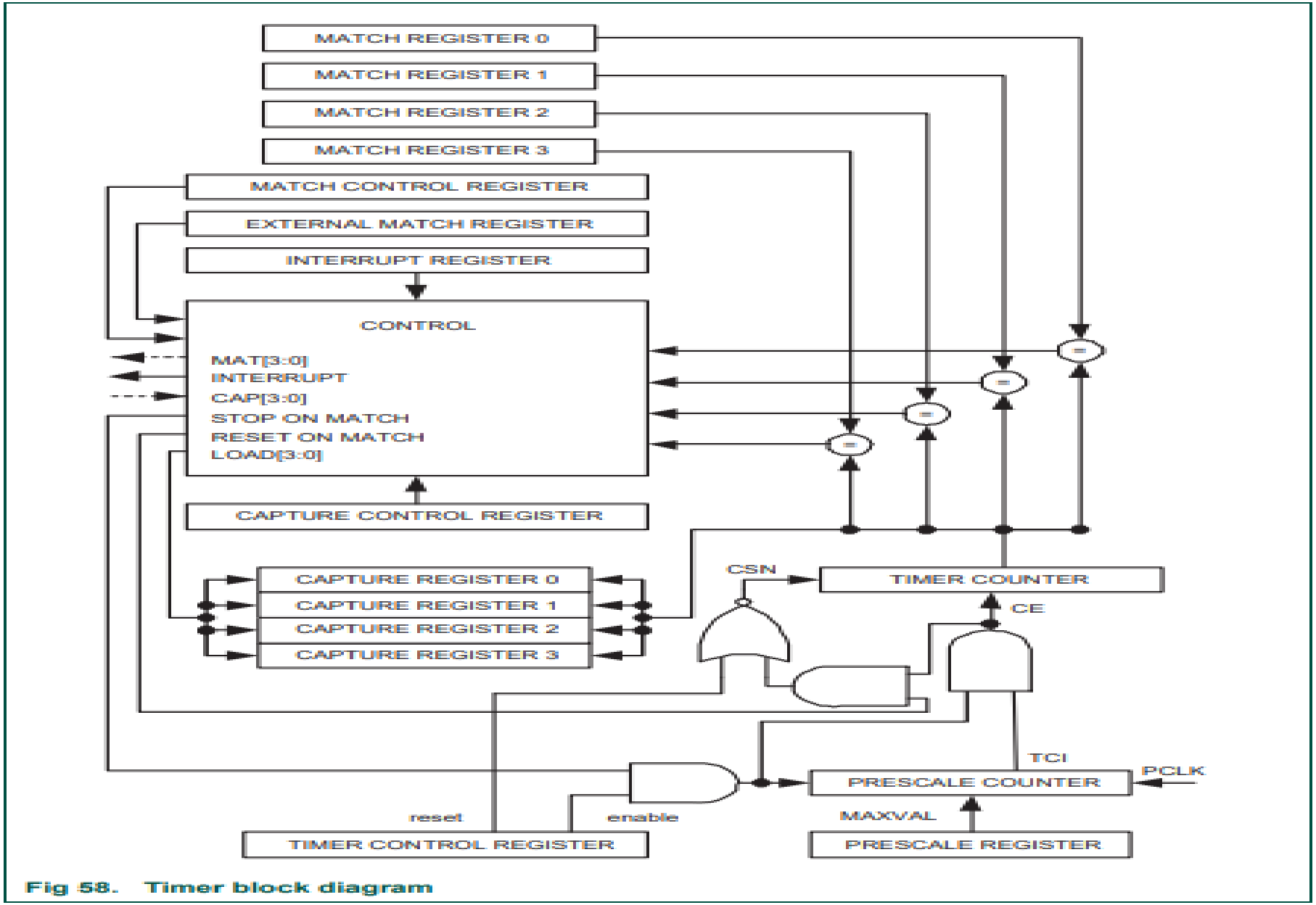


Fig 58. Timer block diagram

Timer Operation

1. Load a number in a match register
2. Start the timer by enabling the ‘E’ bit of T0TCR
3. The timer count register (T0TC) starts incrementing for every tick of the peripheral clock (PCLK) (no pre-scaling is done)
4. When the content of the T0TC equals the value in the match register, timing is said to have occurred
5. One of many possibilities can be made to occur when this happens
6. The possibilities are to reset the timer count register, stop the timer, or generate an interrupts. This setting is done in the T0MCR register

Calculation of Timer Output Frequency

- Crystal Frequency = 60Mhz
- PCLK = 60 MHz/4 = 15MHz
- T=0.067us
- If the match value =0xFF (256 periods of PCLK)
- Timer creates delay of 256x 0.067us = 17.075us, this delay is half the period of the square wave.
- The period of the signal is 2x17.075us =34us

Using Prescaler

- To get lower frequency output – facility of using prescale counter
- 2 register with prescaling – Prescale counter, prescale register
- Prescale counter increments for every PCLK, when it counts up to the value of the prescale counter (T0PR), it allows the timer counter (T0TC) to increment its value by 1
- This causes T0TC to increment on every PCLK when PR=0, increment on every 2 PCLK when PR=1, increment on every 3 PCLK when PR=2, so on
- In effect load a number to T0PR, which will cause the output frequency to get divided

Program

```
#include <lpc214x.h>

__irq void T0_ISR (void)
{
        IO0PIN = ( IO0PIN ^ (0x00000100) );     /* Toggle P0.8 pin */
        T0IR = ( T0IR | (0x01) );
        VICVectAddr = 0x00;
}
```

```
int main (void)
{
VPBDIV = 0x00000002; /* For Pclk = 30MHz */ /* We have configured Cclk=60MHz. Above instruction makes
                                Pclk = Cclk/2 = 30MHz */
PINSEL0 = PINSEL0 | 0x00000020; /* Configure P0.2 as Capture 0.0 */
IO0DIR = ( IO0DIR | (0x00000100) ); /* 8 P0.8-P0.15 as output pins for LED port */
IO0PIN = IO0PIN | 0x00000100; /* Writing 1 to LED pin P0.8 */
VICVectAddr0 = (unsigned) T0_ISR; /* T0 ISR Address */
VICVectCntl0 = 0x00000024; /* Enable T0 IRQ slot */
VICIntEnable = 0x00000010; /* Enable T0 interrupt */
VICIntSelect = 0x00000000; /* T0 configured as IRQ */
T0TCR = 0x02; /* Reset TC and PR */
T0CTCR = 0x00; /* Timer mode, increment on every rising edge */
T0PR = 0x1D; /* Load Pre-Scalar counter with 29 (0 to 29 = 30), so that timer counts every 1usec */
T0MR0 = 100000; /* Load timer counter for 100msec delay, 1usec*1000*100 */
T0MCR = 0x0003; /* Interrupt generate on match and reset timer */
T0TCR = 0x01; /* Enable timer */
while (1); }
```

Basic Step for Timer

- Set appropriate value in TxCTCR
- Define the Prescale value in TxPR
- Set Value(s) in Match Register(s) if required
- Set appropriate value in TxMCR if using Match registers / Interrupts
- Reset Timer – Which resets PR and TC
- Set TxTCR to 0x01 to Enable the Timer when required
- Reset TxTCR to 0x00 to Disable the Timer when required

Note: where x denote which timer we want to use.

Function for initializing timer0

```
void initTimer0(void)
{
/*Assuming that PLL0 has been setup with CCLK = 60Mhz and PCLK
    also = 60Mhz.*/
T0CTCR = 0x0; // Select timer
 T0PR = 60000-1; //(Value in Decimal!) - Increment T0TC at every 60000
    clock cycle
//count begins from zero hence subtracting 1
//60000 clock cycles @60Mhz = 1 mS
T0TCR = 0x02; //Reset Timer
 }
```

Function for generating delay

```
void delay(unsigned int milliseconds)  // Using Timer0
{
T0TCR = 0x02; //Reset Timer
T0TCR = 0x01; //Enable timer
while(T0TC < milliseconds); //wait until timer counter reaches the
    desired delay
 T0TCR = 0x00; //Disable timer
}
```

Program for PLL

```
void initClocks(void)
{
PLL0CON = 0x01;   //Enable PLL
PLL0CFG = 0x24;   //Multiplier and divider setup
PLL0FEED = 0xAA;  //Feed sequence
PLL0FEED = 0x55;
while(!(PLL0STAT & 0x00000400)); //is locked?
PLL0CON = 0x03;   //Connect PLL after PLL is locked
PLL0FEED = 0xAA;  //Feed sequence
PLL0FEED = 0x55;
VPBDIV = 0x01;    // PCLK is same as CCLK i.e.60 MHz
}
```

Using interrupt method generating delay

```
#include <lpc214x.h>
#define MR0I (1<<0) //Interrupt When TC matches MR0
#define MR0R (1<<1) //Reset TC when TC matches MR0
#define DELAY_MS 500 //0.5 Seconds Delay
#define PRESCALE 60000 //60000 PCLK clock cycles to increment TC by 1
void delayMS(unsigned int milliseconds);
void initClocks(void); // PLL Function
void initTimer0(void);
__irq void T0ISR(void); //Interrupt function ISR
```

```
int main(void)
{
initClocks(); //Initialize CPU and Peripheral Clocks @ 60Mhz
initTimer0(); //Initialize Timer0
IO0DIR = 0xFFFFFFFF; //Configure all pins on Port 0 as Output
IO0PIN = 0xF;
T0TCR = 0x01; //Enable timer
while(1); //Infinite Idle Loop //return 0; //normally this wont execute ever :P
}
```

```
void initTimer0(void)
 {
//----------Configure Timer0------------
T0CTCR = 0x0;
T0PR =60000; //Fosc= 60 Mhz
T0MR0 = DELAY_MS-1;
 //(Value in Decimal!) Zero Indexed Count - hence subtracting 1
T0MCR = MR0I | MR0R; //Set bit0 & bit1 to High which is to : Interrupt &
    Reset TC on MR0
//----------Setup Timer0 Interrupt------------
VICVectAddr4 = (unsigned )T0ISR; //Pointer Interrupt Function (ISR)
 VICVectCntl4 = 0x20 | 4; //0x20 (i.e bit5 = 1) -> to enable Vectored IRQ slot
    //0x4 (bit[4:0]) -> this the source number - here its timer0 which has VIC
    channel mask # as
 VICIntEnable = 0x10; //Enable timer0 int
T0TCR = 0x02; //Reset Timer
}
```

```
__irq void T0ISR(void)
 {
long int regVal;
 regVal = T0IR; //Read current IR value
IO0PIN = ~IO0PIN; //Toggle the state of the Pins
T0IR = regVal; //Write back to IR to clear Interrupt Flag
VICVectAddr = 0x0; //This is to signal end of interrupt execution
}
```

/* TOGGLE LEDS FROM P0.0 TO P0.15 WITH EXACT DELAY OF 3SEC */

```
#include<lpc21xx.h>
	void delay(void);
	int main()
	{
	VPBDIV=0X02; //30MHZ//
	PINSEL0=0X00000000;
	IODIR0=0X000000FF;
	IOCLR0=0X000000FF;
	while(1)
	{
	IOSET0=0X000000FF;
	delay();
	IOCLR0=0X000000FF;
	delay();
	}
	}
void delay(void)
	{
	T0PR=30000;
	T0MR0=3000;
	T0TC=0x00000000;
	T0TCR=0X01; //START TIMER//
	while(T0TC !=T0MR0); //1 SEC//
	T0TCR=0X02; //STOP TIMER/
	}
```

```
/* TOGGLE LEDS FROM P0.0 TO P0.7 WITH EXACT DELAY OF 2 SEC BY USING
   T0MR3.TOGGLE LEDS FROM P0.15 TO P0.23 WITH EXACT DELAY OF 5 SEC BY USING
   T1MR2. */

#include<lpc21xx.h>
void delay1(void);
void delay2(void);
int main()
{
VPBDIV=0X02; //30MHZ//
PINSEL0=0X00000000;
IODIR0=0X00FF00FF;
IOCLR0=0X00FF00FF;
while(1)
{
IOSET0=0X00FF00FF;
delay1();
IOCLR0=0X00FF00FF;
delay2();
}
}
```

```
void delay1(void)
    {
    T0PR=30000;
    T0MR0=1000;
    T0TC=0x00000000;
    T0TCR=0X01; //START TIMER//
    while(T0TC !=T0MR0); //1 SEC//
    T0TCR=0X02; //STOP TIMER/
    }
    void delay2(void)
    {
    T1PR=30000;
    T1MR2=2000;
    T1TC=0x00000000;
    T1TCR=0X01; //START TIMER//
    while(T1TC !=T1MR2); //2 SEC//
    T1TCR=0X02; //STOP TIMER//
    }
```

Thank You