

# System Timing

Paolo Bernardi

# System timing

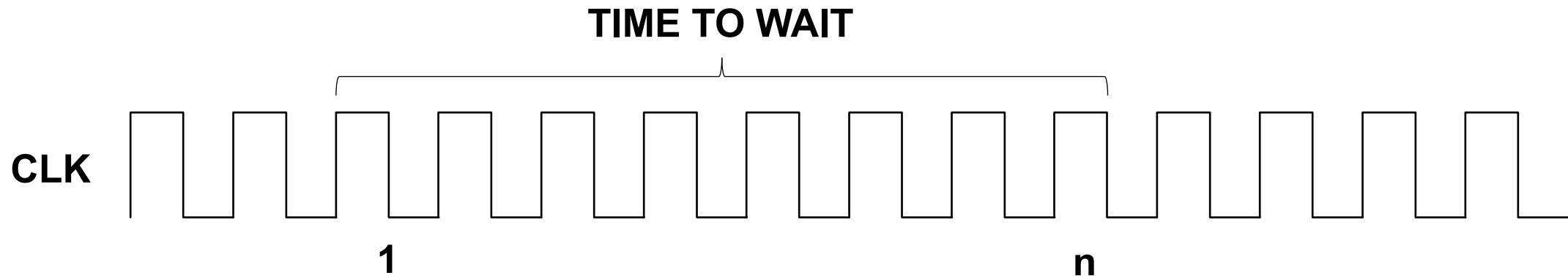
- A system can require
  - To wait for a delay time
  - To perform operations at regular time
- These functionalities are supported by peripheral cores called timers
- Timers main function is therefore to give the programmer opportunities to synchronize the system, based on counting.

# User view of timers

- They are configurable modules that implement a kind of counting mode
- Usually, when a count procedure reaches its end, the system needs to react
  - Typically an interrupt handler is entered at this point
  - Along timer count, the CPU can enter a reduced power mode.

# Working principle

- Timers are supplied with a (dedicated) clock signal
- Timers ability is to count based on the clock
- Timers include registers to be programmed with a number of clock cycles to count



**COUNT = number of clock cycles**

# Counting modes and timing computation

- A timer may be designed following different philosophies
  - Decreasing count – interrupt when count reaches 0
  - Increasing count – interrupt when a match value is reached
- Whatever the counting mode is the following formula can be used to compute the number of clock cycles to count

$$\text{time [s]} = \text{count} * \text{Clock\_Period [s]}$$

$$\text{count} = \text{time [s]} / \text{Clock\_Period [s]}$$

$$\text{count} = \text{time [s]} * \text{frequency [1/s]}$$

# Timing computation examples

- To obtain a time setup of
  - 10 seconds
  - with a 25MHz frequency

$$\rightarrow \text{count} = 10 \text{ [s]} * 25 * 10^6 \text{ [1/s]}$$

$$\rightarrow \text{count} = 25 * 10^7$$

$$\rightarrow \text{count} = 0x0EE6B280$$

- To obtain a time setup of
  - 10 milliseconds
  - With a frequency of 100MHz

$$\rightarrow \text{count} = 10 * 10^{-3} \text{ [s]} * 100 * 10^6 \text{ [1/s]}$$

$$\rightarrow \text{count} = 10^6$$

$$\rightarrow \text{count} = 0x000F4240$$

# Timer count limits

- If a timing request is large, the count value could not fit in the timer register
- Hardware and software features can be used to address this issue
  - HW – Cascade of counters
  - HW – Prescalers (explained later...)
  - SW – Handler software count of HW events.

# Timers in the LPC1768

- It is quite normal that a System-on-Chip includes several timers
- Standard Timers – to be programmed by the user to implement delays and regular intervals
- Operating System Timers – to be used by system management software
- Extra Timers – providing the system with specific functionalities
  - Repetitive Interrupt Timer
  - PWM.



# Standard Timers

- The Timer/Counter (TC) is designed to count cycles of the peripheral clock (PCLK) or an externally-supplied clock, and can optionally generate interrupts or perform other actions at specified timer values, based on four match registers.
- It also includes four capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

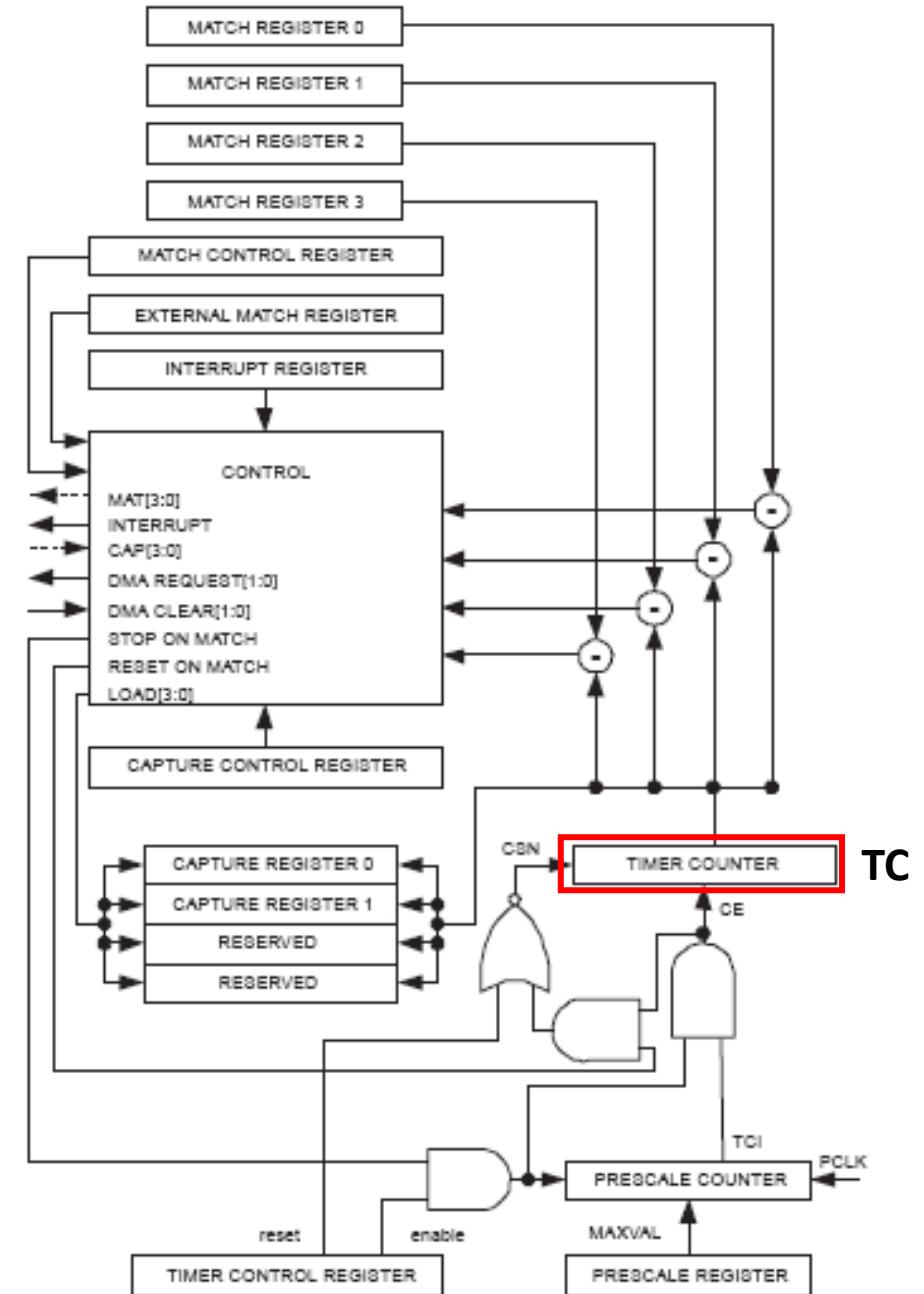
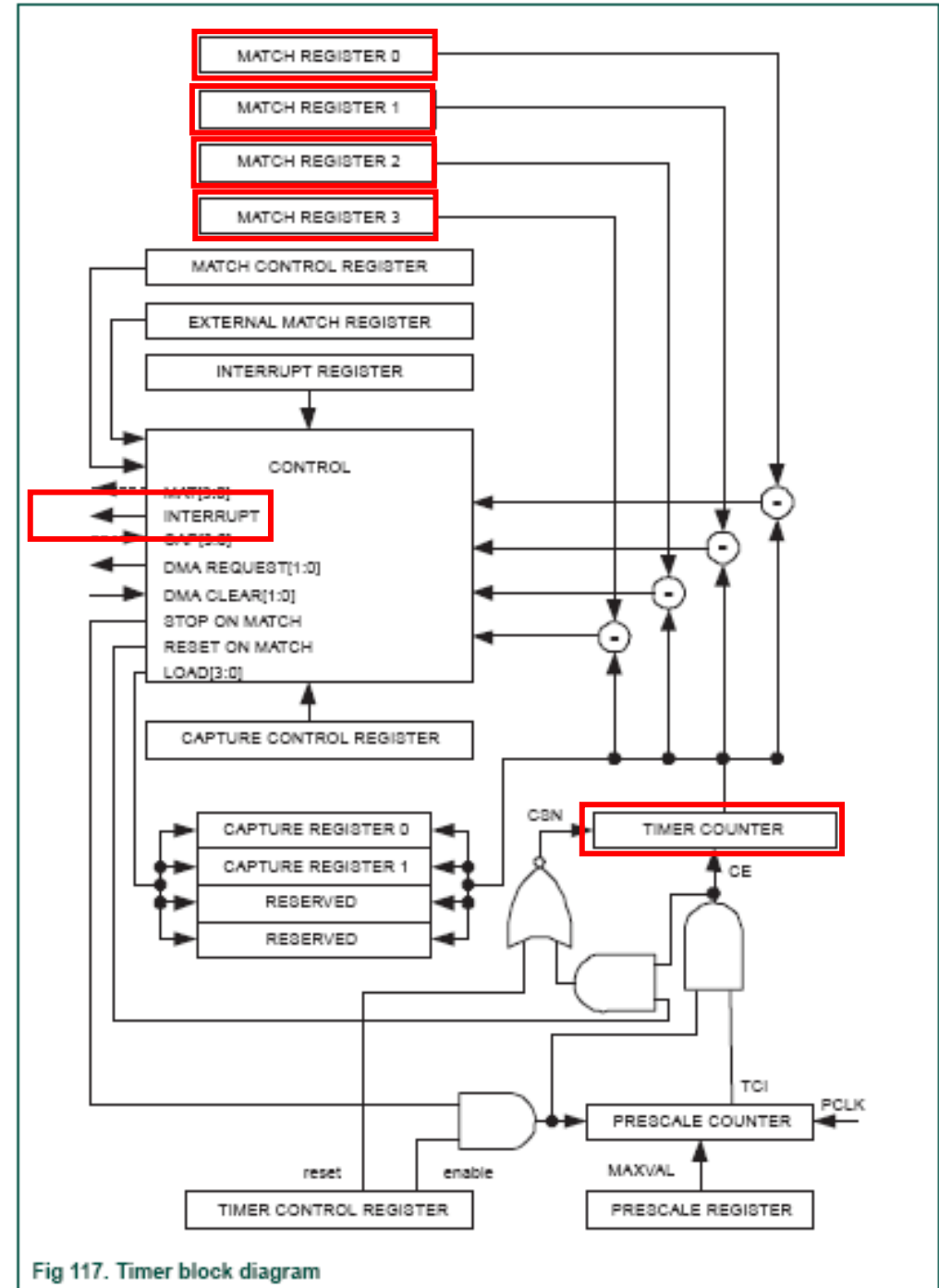


Fig 117. Timer block diagram

# Match registers

- Four 32-bit match registers allows:
  - Continuous operation with optional interrupt generation on match
  - Stop timer on match with optional interrupt generation
  - Reset timer on match with optional interrupt generation
- A unique interrupt is generated, the ISR must understand which of the 4 match registers fired the interrupt (by reading the IR)



# Capture Signals

- A transition on a capture pin can be configured to load one of the Capture Registers
- with the value in the Timer Counter and optionally generate an interrupt.

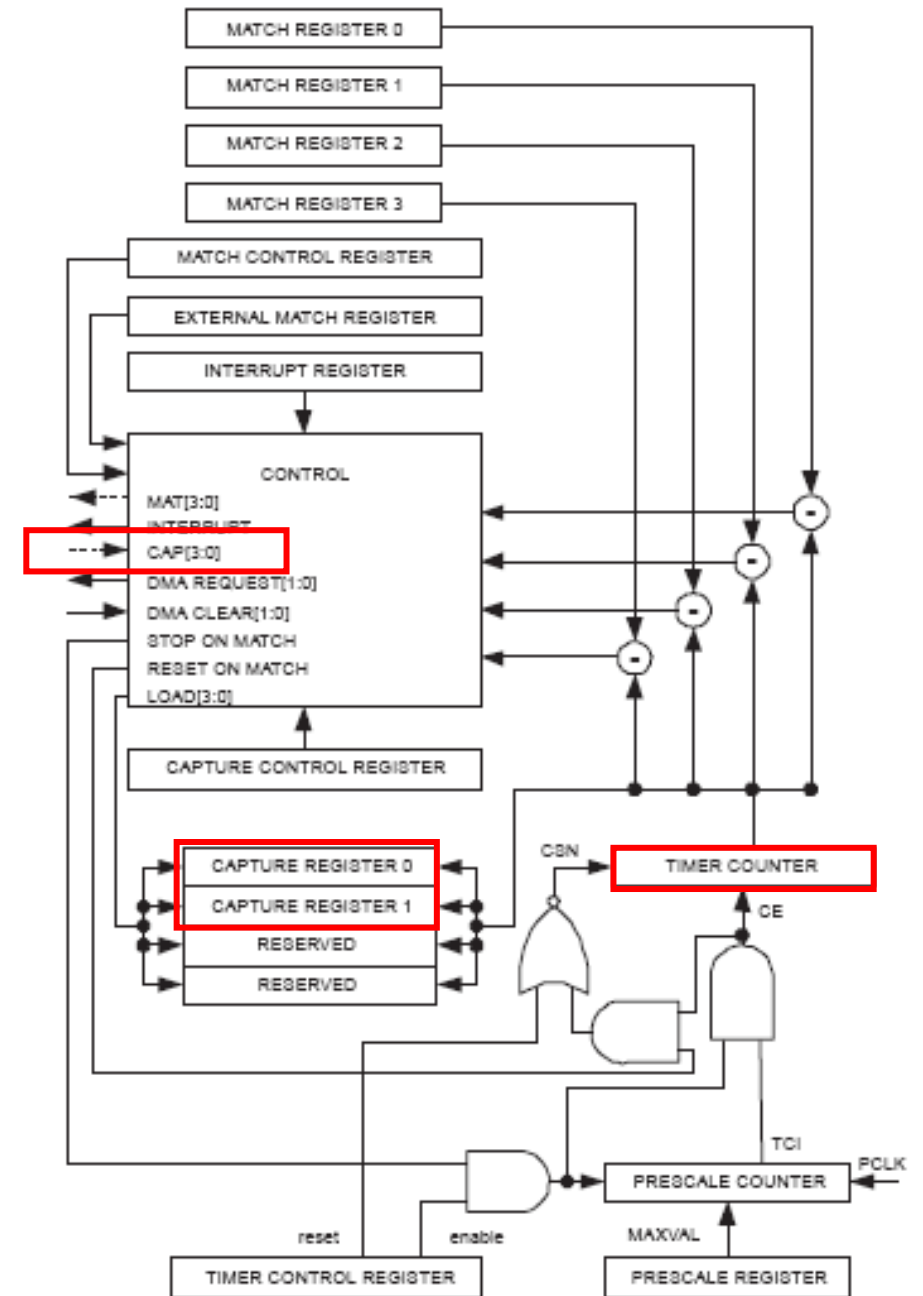


Fig 117. Timer block diagram

# External Match Output

- When a match register (MR3:0) equals the timer counter (TC) this output can either toggle, go low, go high, or do nothing.

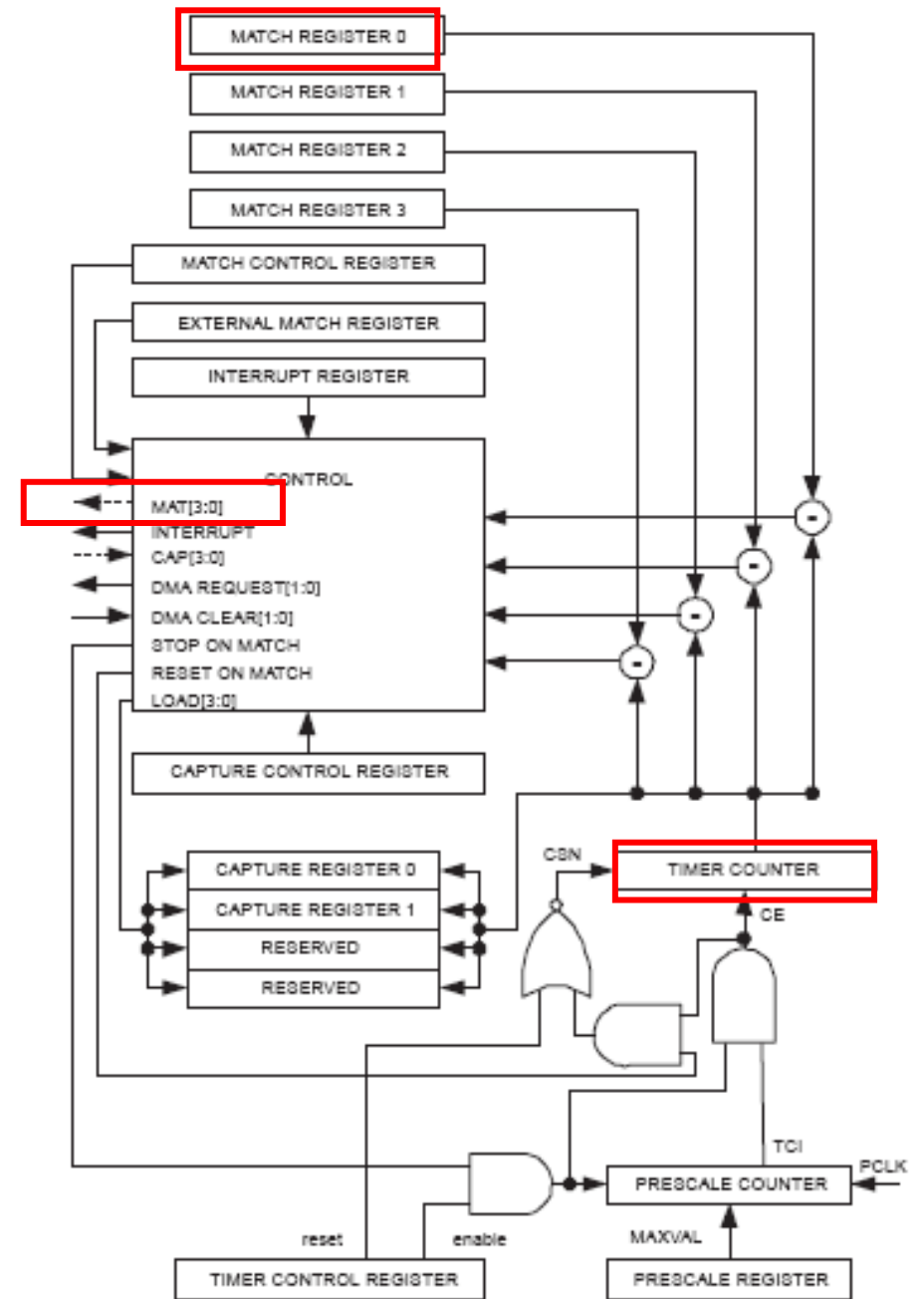


Fig 117. Timer block diagram

# Prescale Register

- TC is incremented every Prescale Register + 1 clock cycles ( $PR + 1$ )
- By default PR is 0, thus TC incremented every clock cycle
- It acts as frequency divider for achieving large timing requests.

$$\text{count} = \text{time[s]} * \frac{\text{frequency [1/s]}}{(PR+1)}$$

$$\text{count} * (PR + 1) = \text{time[s]} * \text{frequency [1/s]}$$

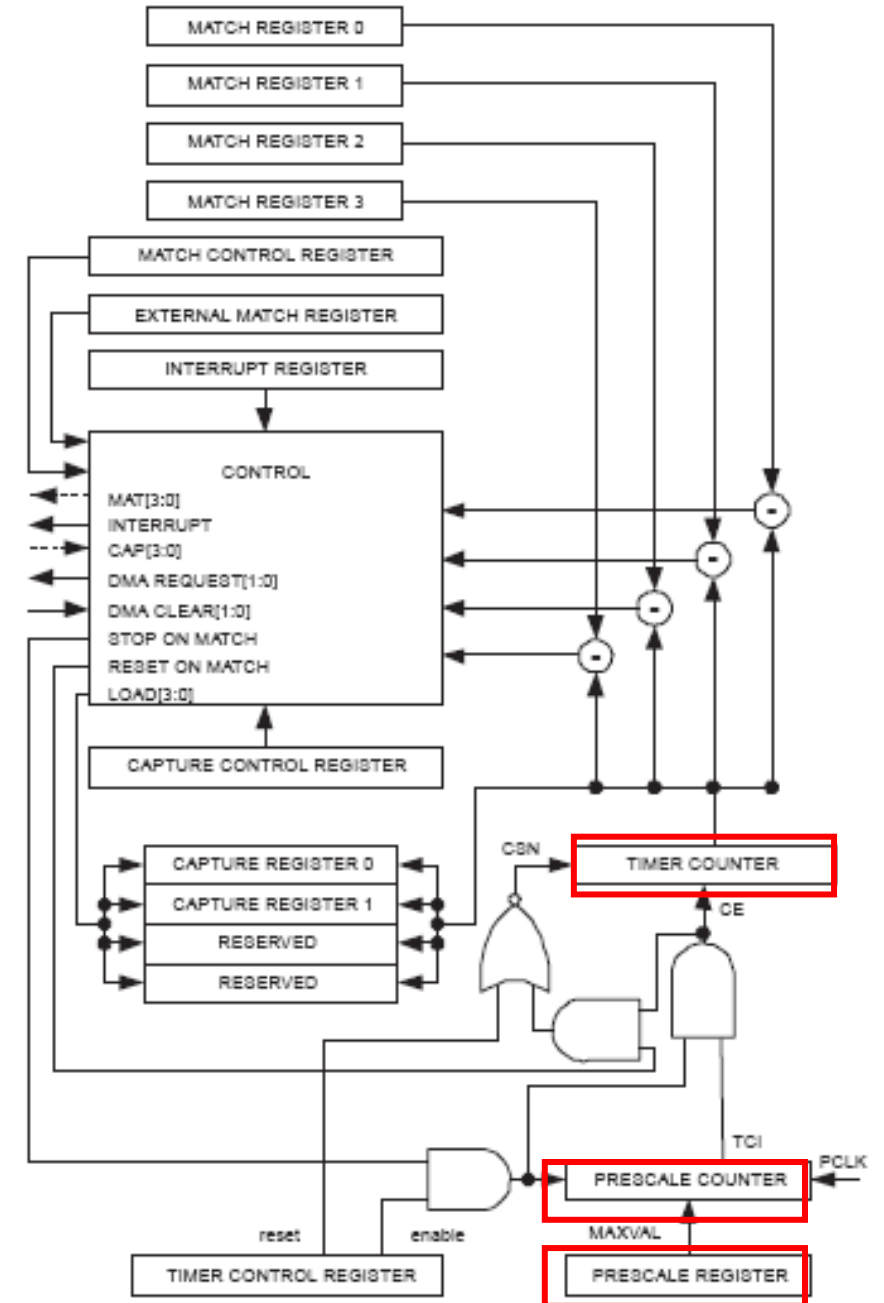
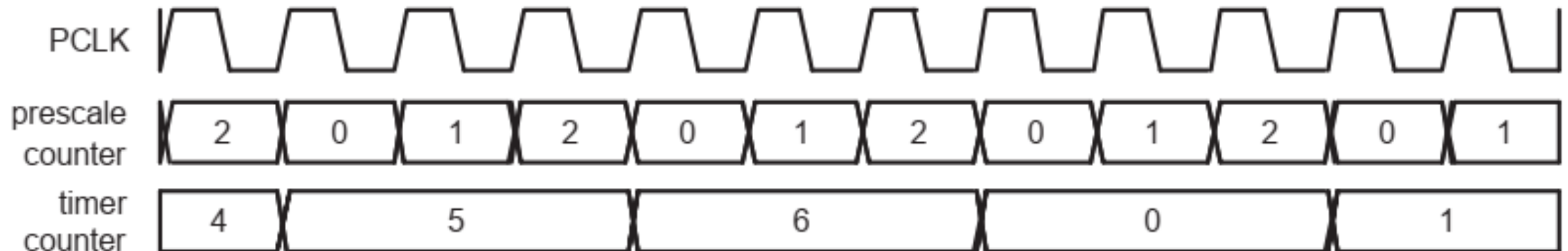


Fig 117. Timer block diagram

# Prescaler in action

- $PR = 2$  and  $MR$  (Match Register) = 6
- TC incremented every  $PR + 1 = 3$  clock cycle



# Main registers (basic timer usage)

Table 426. TIMER/COUNTER0-3 register map

Generic Name	Description	Access	Reset Value <sup>1</sup>
IR	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	R/W	0
TCR	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	R/W	0
TC	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	R/W	0
PR	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	R/W	0
PC	Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.	R/W	0
MCR	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	R/W	0
MR0	Match Register 0. MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.	R/W	0

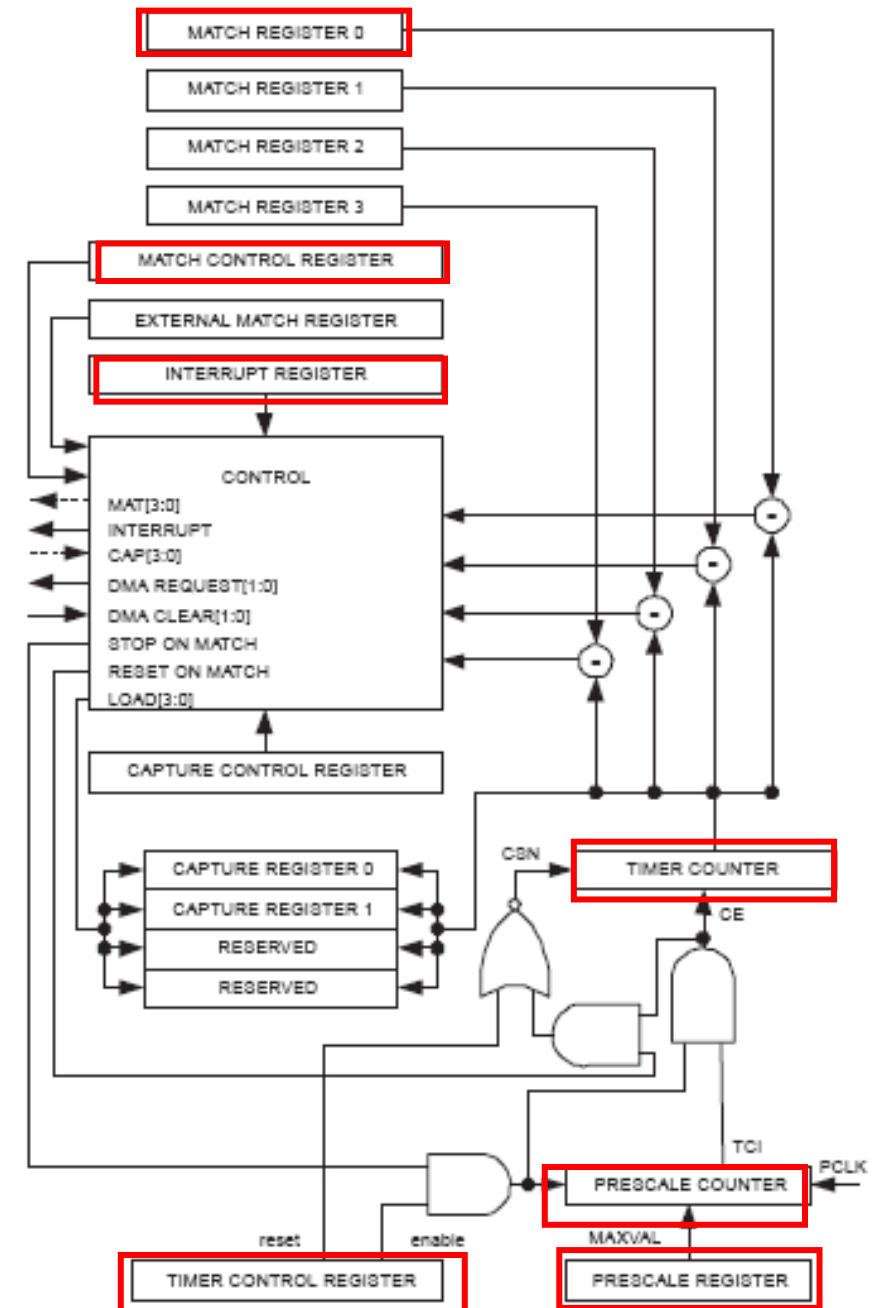


Fig 117. Timer block diagram

# Match Registers

- The Match register values are continuously compared to the Timer Counter value
- When the two values are equal, actions can be triggered automatically
- The action possibilities are
  - to generate an interrupt
  - reset the Timer Counter
  - stop the timer
- Actions are controlled by the settings in the Match Control Register.



# Match Control Register

- The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter
- 12 bits accessible – each of the four MR is controlled with 3 bits

**Table 430. Match Control Register (T[0/1/2/3]MCR - addresses 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014)**  
bit description

Bit	Symbol	Value	Description	Reset Value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		0	Feature disabled.	
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		0	This interrupt is disabled	

# Timer Control Register

- The Timer Control Register (TCR) is used to control the operation of the Timer/Counter
  - Counter Enable = 1 → Timer/Counter activated
  - Counter Reset = 1 → Timer/Counter fixed to reset value

Table 428. Timer Control Register (TCR, TIMERN: TnTCR - addresses 0x4000 4004, 0x4000 8004, 0x4009 0004, 0x4009 4004) bit description

Bit	Symbol	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero.	0
31:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

# Interrupt Register

- The Interrupt Register consists of 4 bits for the match interrupts and 2 bits for the capture interrupts.
  - If an interrupt is generated then the corresponding bit in the IR will be high.
  - Otherwise, the bit will be low.
- Writing a logic one to the corresponding IR bit will reset the interrupt
- Writing a zero has no effect.

**Table 427. Interrupt Register (T[0/1/2/3]IR - addresses 0x4000 4000, 0x4000 8000, 0x4009 0000, 0x4009 4000) bit description**

Bit	Symbol	Description	Reset Value
0	MR0 Interrupt	Interrupt flag for match channel 0.	0
1	MR1 Interrupt	Interrupt flag for match channel 1.	0
2	MR2 Interrupt	Interrupt flag for match channel 2.	0
3	MR3 Interrupt	Interrupt flag for match channel 3.	0
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.	0
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.	0
31:6	-	Reserved	-

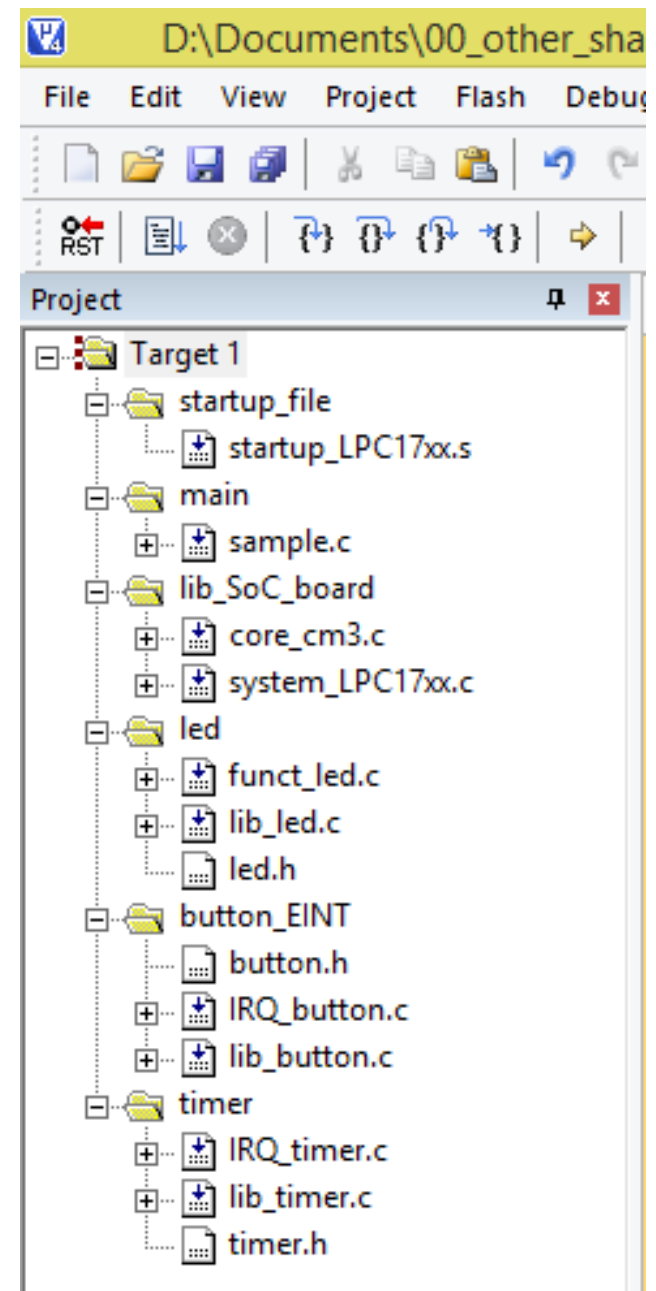
# Prescale Register

- The 32-bit Prescale register specifies the maximum value for the Prescale Counter
- The 32-bit Prescale Counter controls division of PCLK by some constant value before it is applied to the Timer Counter
- The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale register, the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK. This causes the Timer Counter to increment on every PCLK when PR = 0, every 2 pclk's when PR = 1

PR	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	R/W	0	T0PR - 0x4000 400C T1PR - 0x4000 800C T2PR - 0x4009 000C T3PR - 0x4009 400C
PC	Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.	R/W	0	T0PC - 0x4000 4010 T1PC - 0x4000 8010 T2PC - 0x4009 0010 T3PC - 0x4009 4010

# Timer library

- lib\_timer.c
  - init\_timer(timer\_num, timerInterval )
  - enable\_timer(timer\_num );
  - disable\_timer(timer\_num );
  - reset\_timer(timer\_num );
- IRQ\_timer.c
  - TIMER0\_IRQHandler();
  - TIMER1\_IRQHandler();



# Delay setup example

- Setup a 10 seconds delay with 25MHz supply frequency
- Raise and interrupt, reset and stop TC

```
23
24 int main (void) {
25
26     SystemInit();           /* System
27     LED_init();             /* LED In:
28     BUTTON_init();          /* BUTTON
29     init_timer(0,0x0EE6B280); /* TIMERO
30     enable_timer(0);
31
32     LPC_SC->PCON |= 0x1;     /* power-
33     LPC_SC->PCON &= 0xFFFFFFF0;
34
35     while (1) {             /* Loop f
36         __ASM("wfi");
37     }
```

See slide 6 about  
0x0EE6B280

# init\_timer function

- MCR setup occurs in init\_timer through a configuration wizard.

```
23
24 int main (void) {
25
26     SystemInit();
27     LED_init();
28     BUTTON_init();
29     init_timer(0, 0x0EE6B280);
30     enable_timer(0);
31
32     LPC_SC->PCON |= 0x1;
33     LPC_SC->PCON &= 0xFFFFFFF;
34
35     while (1) {
36         __ASM("wfi");
37     }
38
39 }
40
```

The screenshot shows the 'Timer 0' configuration window. The 'Match Channels' section is expanded, showing the following settings:

- MCR: 0x00000007 (highlighted with a red box)
- MR0: 0x0EE6B280 (highlighted with a red box)
- MR1: 0x00000000
- MR2: 0x00000000
- MR3: 0x00000000

Below the match channels, the 'Interrupt on MR0' section is checked (highlighted with a red box):

- ☒ Interrupt on MR0
- ☒ Reset on MR0
- ☒ Stop on MR0

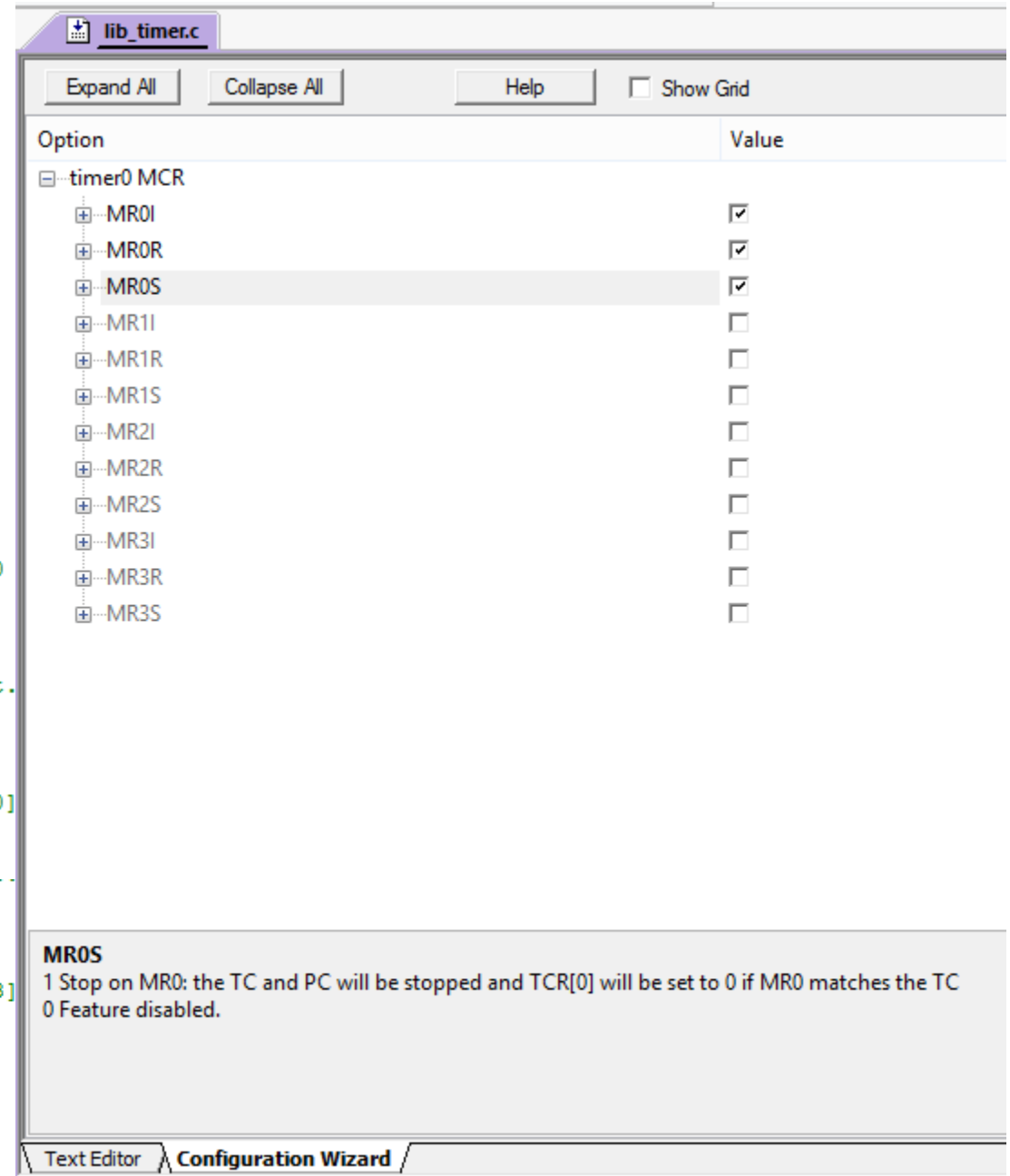
The 'Capture Channels' section is also visible, showing CCR: 0x00000000, CR0: 0x00000000, and CR1: 0x00000000.

The 'Count Control' section at the bottom shows CTCR: 0x00000000, Mode: Timer, and Counter Input: CAP0.0.

# lib\_timer.c

- Text editor and configuration wizard

```
91  /*** <<< Use Configuration Wizard in Context Menu >>> ***
92  // <h> timer0 MCR
93  //   <e.0> MR0I
94  //   <i> 1 Interrupt on MR0: an interrupt is generated when MR0
95  //   <i> 0 This interrupt is disabled
96  //   </e>
97  //   <e.1> MR0R
98  //   <i> 1 Reset on MR0: the TC will be reset if MR0 matches it.
99  //   <i> 0 Feature disabled.
100 //   </e>
101 //   <e.2> MR0S
102 //   <i> 1 Stop on MR0: the TC and PC will be stopped and TCR[0]
103 //   <i> 0 Feature disabled.
...
135 //   <i> 0 Feature disabled.
136 //   </e>
137 //   <e.11> MR3S
138 //   <i> 1 Stop on MR3: the TC and PC will be stopped and TCR[3]
139 //   <i> 0 Feature disabled.
140 //   </e>
141     LPC_TIM0->MCR = 7;
142 // </h>
143 /*** <<< end of configuration section >>>   ***
```





# Enable\_timer function

- The timer is made running.

```
23
24 int main (void) {
25
26     SystemInit();
27     LED_init();
28     BUTTON_init();
29     init_timer(0, 0x0EE6B280);
30     enable_timer(0);
31
32     LPC_SC->PCON |= 0x1;
33     LPC_SC->PCON &= 0xFFFFFFF;
34
35     while (1) {
36         __ASM("wfi");
37     }
38 }
```

Timer 0

Prescaler  
PR: 0x00000000  
PC: 0x00000000

Timer  
TCR: 0x00000001  
TC: 0x00000132  
☒ Enable  
☐ Reset

Interrupt Register  
IR: 0x00000000

Match Channels  
MCR: 0x00000007  
EMR: 0x00000000  
MR0: 0x0EE6B280  
MR1: 0x00000000  
MR2: 0x00000000  
MR3: 0x00000000  
☒ Interrupt on MR0  
☐ Interrupt on MR1  
☐ Interrupt on MR2  
☐ Interrupt on MR3  
☒ Reset on MR0  
☐ Reset on MR1  
☐ Reset on MR2  
☐ Reset on MR3  
☒ Stop on MR0  
☐ Stop on MR1  
☐ Stop on MR2  
☐ Stop on MR3  
EMC0: Nothing  
EMC1: Nothing  
EMC2: Nothing  
EMC3: Nothing  
☐ External Match 0  
☐ External Match 1  
☐ External Match 2  
☐ External Match 3  
☐ MR0 Interrupt  
☐ MR1 Interrupt  
☐ MR2 Interrupt  
☐ MR3 Interrupt

Capture Channels  
CCR: 0x00000000  
CR0: 0x00000000  
CR1: 0x00000000  
☐ Rising Edge 0  
☐ Rising Edge 1  
☐ Falling Edge 0  
☐ Falling Edge 1  
☐ Interrupt on Event 0  
☐ Interrupt on Event 1  
☐ CAP0.0  
☐ CAP0.1  
☐ CR0 Interrupt  
☐ CR1 Interrupt

Count Control  
CTCR: 0x00000000  
Mode: Timer  
Counter Input: CAP0.0

# Timing check

- Measure time between breakpoints.

```
20
21 /*-----Main Program-----*/
22
23
24 int main (void) {
25
26     SystemInit();           /* System Init
27     LED_init();             /* LED Initial
28     BUTTON_init();          /* BUTTON Init
29     init_timer(0,0x0EE6B280); /* TIMER0 Init
30     enable_timer(0);
31
32     LPC_SC->PCON |= 0x1;     /* power-down
33     LPC_SC->PCON &= 0xFFFFFFF0;
```

```
13 /*-----
14 ** Function name:   Timer0_IRQHandler
15 **
16 ** Descriptions:   Timer/Counter 0 interrupt handler
17 **
18 ** parameters:     None
19 ** Returned value:  None
20 **
21 ****
22
23 void TIMER0_IRQHandler (void)
24 {
25
26     LPC_TIM0->IR = 1;        /* clear interrupt flag */
27     return;
28 }
```

IRQ\_timer.clib\_timer.csample.cstartup\_LPC17xx.s

```
4  * Note(s):
5  *-----
6  *
7  * This software is supplied "AS IS" without warranties of any kind.
8  *
9  * Copyright (c) 2017 Politecnico di Torino. All rights reserved.
10 *-----
11
12 #include <stdio.h>
13 #include "LPC17xx.H"           /* LPC17xx definitions
14 #include "led/led.h"
15 #include "button_EXINT/button.h"
16 #include "timer/timer.h"
17
18 /* Led external variables from funct_led */
19 extern unsigned char led_value; /* defined in funct_led.c
20
21 /*-----
22 Main Program
23 *-----
24 int main (void) {
25
26     SystemInit();               /* System Initialization
27     LED_init();                 /* LED Initialization
28     BUTTON_init();              /* BUTTON Initialization
29     init_timer(0,0x0EE6B280);    /* TIMER0 Initialization
30     enable_timer(0);
31
32     LPC_SC->PCON |= 0x1;         /* power-down mode
33     LPC_SC->PCON &= 0xFFFFFFF0;
34
35     while (1) {                 /* Loop forever
36         __ASM("wfi");
37     }
38
39 }
40
```

Timer 0

Prescaler

PR: 0x00000000

PC: 0x00000000

Timer

TCR: 0x00000000

TC: 0x00000000

☐ Enable

☐ Reset

Interrupt Register

IR: 0x00000000

Match Channels

MCR: 0x00000007

EMR: 0x00000000

MR0: 0x0EE6B280

MR1: 0x00000000

MR2: 0x00000000

MR3: 0x00000000

☒ Interrupt on MR0

☐ Interrupt on MR1

☐ Interrupt on MR2

☐ Interrupt on MR3

☒ Reset on MR0

☐ Reset on MR1

☐ Reset on MR2

☐ Reset on MR3

☒ Stop on MR0

☐ Stop on MR1

☐ Stop on MR2

☐ Stop on MR3

EMC0: Nothing

EMC1: Nothing

EMC2: Nothing

EMC3: Nothing

☐ External Match 0

☐ External Match 1

☐ External Match 2

☐ External Match 3

☐ MR0 Interrupt

☐ MR1 Interrupt

☐ MR2 Interrupt

☐ MR3 Interrupt

Capture Channels

CCR: 0x00000000

CR0: 0x00000000

CR1: 0x00000000

☐ Rising Edge 0

☐ Rising Edge 1

☐ Falling Edge 0

☐ Falling Edge 1

☐ Interrupt on Event 0

☐ Interrupt on Event 1

☐ CAP0.0

☐ CAP0.1

☐ CR0 Interrupt

☐ CR1 Interrupt

Count Control

CTCR: 0x00000000

Mode: Timer

Counter Input: CAP0.0

Simulation

t1: 0.00003513 sec

L:30 C:1

CAP NUM SCRL OVR R/W

IRQ\_timer.c lib\_timer.c sample.c startup\_LPC17xx.s

```
1 /*****
2 **-----File Info-----
3 ** File name:          IRQ_timer.c
4 ** Last modified Date: 2014-09-25
5 ** Last Version:      V1.00
6 ** Descriptions:      functions to manage T0 and T1 interrupt
7 ** Correlated files:  timer.h
8 **
9 *****/
10 #include "lpc17xx.h"
11 #include "timer.h"
12
13 /*****
14 ** Function name:      Timer0_IRQHandler
15 **
16 ** Descriptions:      Timer/Counter 0 interrupt handler
17 **
18 ** parameters:        None
19 ** Returned value:     None
20 **
21 *****/
22
23 void TIMER0_IRQHandler (void)
24 {
25     LPC_TIM0->IR = 1; /* clear interrupt flag */
26     return;
27 }
28
29
30
31 /*****
32 ** Function name:      Timer1_IRQHandler
33 **
34 ** Descriptions:      Timer/Counter 1 interrupt handler
35 **
36 ** parameters:        None
37 ** Returned value:     None
38 *****/
```

Timer 0

Prescaler: PR: 0x00000000 PC: 0x00000000

Timer: TCR: 0x00000000 TC: 0x00000000

Interrupt Register: IR: 0x00000001

Match Channels: MCR: 0x00000007 MR0: 0x0EE6B280

EMR: 0x00000000 MR1: 0x00000000

Interrupt on MR0 ☒ Reset on MR0 ☒ Stop on MR0 ☒ Interrupt on MR1 ☐ Reset on MR1 ☐ Stop on MR1 ☐ Stop on MR2 ☐ Stop on MR3 ☐

EMC0: Nothing EMC1: Nothing EMC2: Nothing EMC3: Nothing

External Match 0 ☐ External Match 1 ☐ External Match 2 ☐ External Match 3 ☐

MR0 Interrupt ☒ MR1 Interrupt ☐ MR2 Interrupt ☐ MR3 Interrupt ☐

Capture Channels: CCR: 0x00000000 CR0: 0x00000000 CR1: 0x00000000

Rising Edge 0 ☐ Falling Edge 0 ☐ Interrupt on Event 0 ☐ CAP0.0 ☐ CR0 Interrupt ☐

Rising Edge 1 ☐ Falling Edge 1 ☐ Interrupt on Event 1 ☐ CAP0.1 ☐ CR1 Interrupt ☐

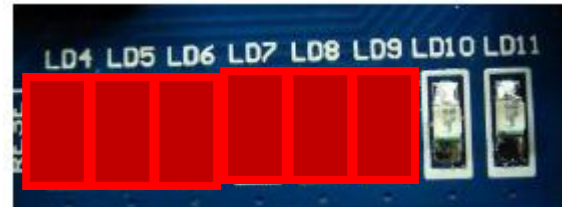
Count Control: CTCR: 0x00000000 Mode: Timer Counter Input: CAP0.0

Clear Interrupt flag for MR0

Simulation t1: 10.00003533 sec L:26 C:1 CAP NUM SCRL OVR R/W

# Exercise 1

- Setup regular interval time interruption (Tutorial)
  - INT0 -> turn ON LED0
  - KEY1 -> turn ON LED1
  - KEY2 -> turn OFF LED0 and LED1



# Exercise 2

- Improve the existing library to use:
  - All the timers (0-3)
  - All 4 Match Registers
  - The Prescaler
- Assume the system is running at 12 MHz, compute the maximum achievable delay
- Try to overcome this limitation using
  - HW resources available in the peripheral
  - SW methods.