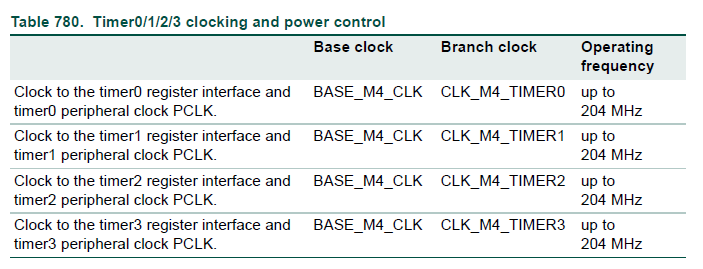
TIMERS

**Caracterisiticas**

Cada Timer/Counter está diseñado para contar ciclos del clock periférico (PCLK) o de un clock externo. Pueden opcionalmente generar interrupciones y otras acciones en valores del timer específicos, basados en los match registers. También incluye 4 entradas de capturas para “capturar” el valor del timer cuando ocurre una transición de la señal de entrada, y opcionalmente generar una interrupción.

**Caracterisiticas**

* 4 Timers/Counters de 32-bits con un Prescaler programable de 32-bits cada uno.
* 4 32-bits Capture channel por cada Timer que pueden tomar un snapshot del valor del timer con una transición de la señal de entrada (input signal). Opcionalmente puede generar una interrupción.
* 4 32-bits Match Registers que pueden permitir:
  + Generar interrupción
  + Detener el Timer y opcionalmente generar interrupción
  + Resetear el Timer y opcionalmente generar interrupción
* 4 Salidas externas (External outputs) correspondientes a cada Match Register con la posibilidad con setear en 0, 1 y togglear la salida externa en caso de suceder el match.

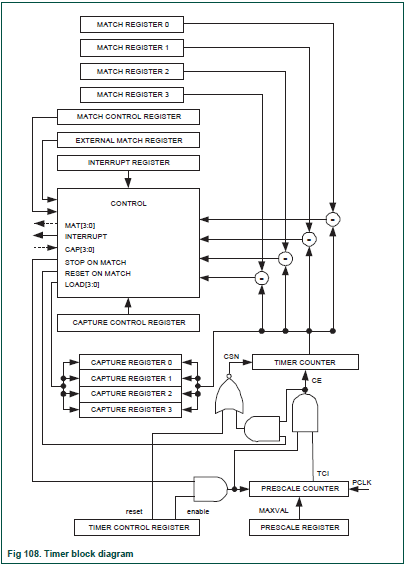


**Registros del microcontrolador**

|  |  |
| --- | --- |
| TIMER | REGISTRO BASE |
| TIMER0 | 0x 4008 4000 |
| TIMER1 | 0x 4008 5000 |
| TIMER2 | 0x 400C 3000 |
| TIMER3 | 0x 400C 4000 |

|  |  |  |  |
| --- | --- | --- | --- |
| SIGLAS | NOMBRE | OFFSET | DESCRIPCIÓN |
| IR | Interrupt Register | 0x000 | Interrupt flags para los match y captura de los timers. 1 = interrupción generada. |
| TCR | Timer Control Register | 0x004 | Activa y desactiva las operaciones de los Timer/Counter. |
| TC | Timer Counter | 0x008 | Registro de 32-bits. Incrementa cuando el prescale counter llega a su máximo valor. Este evento no genera una interrupción. Genera un overflow. |
| PR | Prescale Register | 0x00C | Registro de 32-bits. Especifica el máximo valor del Prescale Counter. |
| PC | Prescale Counter | 0x010 | Registro de 32-bits. Controla la división del PCLK por una constante antes de ser aplicado al Timer Counter. |
| MCR | Match Control Register | 0x014 | Controla qué operaciones son realizadas cuando uno de los Match Registers metchea el Timer Counter (estas son: Generación de interrupción MR y Reset del TC). |
| MR0 | Match Register 0 | 0x018 | Registro de 32-bits. Especifica el valor para matchear el Timer Counter |
| MR1 | Match Register 1 | 0x01C | Idem |
| MR2 | Match Register 2 | 0x020 | Idem |
| MR3 | Match Register 3 | 0x024 | Idem |
| CCR | Capture Control Register | 0x028 | Controla si uno de las 4 capturas es cargado con el valor del Timer Counter cuando el capture event ocurre y si una interrupción es generada por el capture evento. |
| CR0 | Capture Register 0 | 0x02C | Valor del timer counter capture |
| CR1 | Capture Register 1 | 0x030 | Idem |
| CR2 | Capture Register 2 | 0x034 | Idem |
| CR3 | Capture Register 3 | 0x038 | Idem |
| EMR | External Match Register | 0x03C | Controla y muestra el estado de los external match pins. |
| CTCR | Count Control Register | 0x070 | Usado para seleccionar entre modo Timer o modo Counter (en este modo se selecciona el pin y el flanco para contar).  Timer Mode (Cuenta cada flanco asc. Del PCLK)  Counter Mode rising/falling/both Edge |

**Diagrama**



**Primitivas y structs – Nivel Sapi**

**Ubicación**: firmware\_v3/libs/sapi/sapi\_v0.6.2/soc/peripherals/src/**sapi\_timer.c**

**typedef** **struct** {

callBackFuncPtr\_t timerCompareMatchFunctionPointer[4];

} timerDinamicData\_t;

**void** **Timer\_Init**( uint8\_t timerNumber , uint32\_t ticks, callBackFuncPtr\_t voidFunctionPointer );

* **Definición**: Inicializa el Timer periférico.
* **timerNumber**: corresponde al Timer (0, 1, 2 o 3).
* **ticks**: números de ticks necesarios para finalizar el ciclo.
* **voidFunctionPointer**: ptr a función a ejecutar al final del ciclo.
* Funciones utilizadas del nivel Chip:

/\* Enable timer clock and reset it \*/

Chip\_TIMER\_Init();

Chip\_RGU\_TriggerReset();

/\* Update the defalut function pointer name of the Compare match 0\*/

timer\_dd[timerNumber].timerCompareMatchFunctionPointer

/\* Initialize compare match with the specified ticks (number of counts needed to clear the match counter) \*/

Chip\_TIMER\_MatchEnableInt( *TIMERCOMPAREMATCH0* );

Chip\_TIMER\_SetMatch(*TIMERCOMPAREMATCH0*, ticks );

/\* Makes Timer Match 0 period the timer period\*/

Chip\_TIMER\_ResetOnMatchEnable(*TIMERCOMPAREMATCH0* );

/\*Enable timer\*/

Chip\_TIMER\_Enable();

/\* Enable timer interrupt \*/

NVIC\_SetPriority();

NVIC\_EnableIRQ();

NVIC\_ClearPendingIRQ(

}

**void** **Timer\_DeInit**( uint8\_t timerNumber );

* **Definición**: Desactiva el timer (número) pasado por argumento.
* **timerNumber**: corresponde al Timer (0, 1, 2 o 3).
* Funciones utilizadas del nivel Chip:

NVIC\_DisableIRQ(timer\_sd[timerNumber].IRQn);

Chip\_TIMER\_Disable(timer\_sd[timerNumber].name);

Chip\_TIMER\_DeInit(timer\_sd[timerNumber].name);

uint32\_t **Timer\_microsecondsToTicks**( uint32\_t uS );

* **Definición**: Convierte el argumento en microsegundos a número de ticks.
* **Us**: microsegundos a convertir en ticks.
* Puede ser usado en el segundo parámetro de la función Timer\_Init().
* Funciones utilizadas del nivel Chip:

uS\*(LPC4337\_MAX\_FREC/1000000)

**void** **Timer\_EnableCompareMatch**( uint8\_t timerNumber, uint8\_t compareMatchNumber, uint32\_t ticks, callBackFuncPtr\_t voidFunctionPointer );

* **Definición**: Habilita un compare match in un timer.
* **compareMatchNumber**: Compare match (number: 1 2 o 3).
* **ticks**: valor de comparación en número de ticks.
* Funciones utilizadas del nivel Chip

Chip\_TIMER\_MatchEnableInt();

Chip\_TIMER\_SetMatch(ticks);

**void** **Timer\_DisableCompareMatch**( uint8\_t timerNumber, uint8\_t compareMatchNumber );

* **Definición**: Desabilita el compare match (number).
* **compareMatchNumber**: Compare match (number: 1 2 o 3).

Funciones utilizadas del nivel Chip:

Chip\_TIMER\_ClearMatch();

Chip\_TIMER\_MatchDisableInt();

**void** **Timer\_SetCompareMatch**( uint8\_t timerNumber, uint8\_t compareMatchNumber, uint32\_t ticks );

Permite cambiar el valor de comparación del compareMatchNumber del

timerNumber.

Funciones utilizadas del nivel Chip:

Chip\_TIMER\_SetMatch(ticks);

**Primitivas y structs – Timer – Nivel Chip**

**Ubicación**: firmware\_v3/libs/lpc\_open/lpc\_chip\_43xx/src/**timer\_18xx\_43xx.c**

**typedef** **struct** { /\*!< TIMERn Structure \*/

\_\_IO uint32\_t IR;

\_\_IO uint32\_t TCR;

\_\_IO uint32\_t TC;

\_\_IO uint32\_t PR;

\_\_IO uint32\_t 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. \*/

\_\_IO uint32\_t 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. \*/

\_\_IO uint32\_t MR[4];

\_\_IO uint32\_t CCR;

\_\_IO uint32\_t CR[4];

\_\_IO uint32\_t EMR;

\_\_I uint32\_t RESERVED0[12];

\_\_IO uint32\_t CTCR;

} LPC\_TIMER\_T;

**void** **Chip\_TIMER\_Init**(LPC\_TIMER\_T \*pTMR)

{

Chip\_Clock\_Enable(Chip\_TIMER\_GetClockIndex(pTMR));

}

**void** **Chip\_TIMER\_DeInit**(LPC\_TIMER\_T \*pTMR)

{

Chip\_Clock\_Disable(Chip\_TIMER\_GetClockIndex(pTMR));

}

**void** **Chip\_TIMER\_Reset**(LPC\_TIMER\_T \*pTMR)

{

uint32\_t reg;

/\* Disable timer, set terminal count to non-0 \*/

reg = pTMR->TCR;

pTMR->TCR = 0;

pTMR->TC = 1;

/\* Reset timer counter \*/

pTMR->TCR = TIMER\_RESET;

/\* Wait for terminal count to clear \*/

**while** (pTMR->TC != 0) {}

/\* Restore timer state \*/

pTMR->TCR = reg;

}

/\* Returns clock index for the peripheral block \*/

STATIC CHIP\_CCU\_CLK\_T **Chip\_TIMER\_GetClockIndex**(LPC\_TIMER\_T \*pTMR)

{

CHIP\_CCU\_CLK\_T clkTMR;

**if** (pTMR == LPC\_TIMER3) {

clkTMR = CLK\_MX\_TIMER3;

}

**else** **if** (pTMR == LPC\_TIMER2) {

clkTMR = CLK\_MX\_TIMER2;

}

**else** **if** (pTMR == LPC\_TIMER1) {

clkTMR = CLK\_MX\_TIMER1;

}

**else** {

clkTMR = CLK\_MX\_TIMER0;

}

**return** clkTMR;

}

**Primitivas y structs – Clock – Nivel Chip**

**Ubicación**: firmware\_v3/libs/lpc\_open/lpc\_chip\_43xx/src/**clock\_18xx\_43xx.c**

/\* Enables a peripheral clock \*/

**void** **Chip\_Clock\_Enable**(CHIP\_CCU\_CLK\_T clk)

{

/\* Start peripheral clock running \*/

**if** (clk >= CLK\_CCU2\_START) {

LPC\_CCU2->CLKCCU[clk - CLK\_CCU2\_START].CFG |= 1;

}

**else** {

LPC\_CCU1->CLKCCU[clk].CFG |= 1;

}

}

/\* Disables a peripheral clock \*/

**void** **Chip\_Clock\_Disable**(CHIP\_CCU\_CLK\_T clk)

{

/\* Stop peripheral clock \*/

**if** (clk >= CLK\_CCU2\_START) {

LPC\_CCU2->CLKCCU[clk - CLK\_CCU2\_START].CFG &= ~1;

}

**else** {

LPC\_CCU1->CLKCCU[clk].CFG &= ~1;

}

}