Manejo del Interval Timer Core en el procesador NIOS II

Por: Kalun José Lau Gan

2021

Aspectos generales del módulo Interval Timer Core

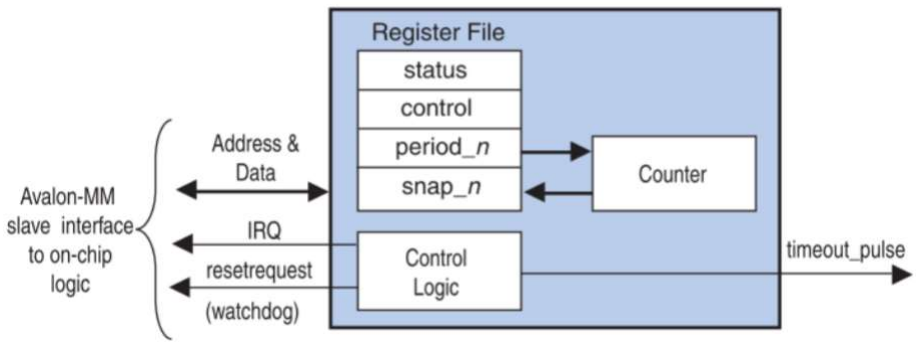
- Temporizador con resolución configurable entre 32 bit y 64 bit.
- De cuenta descendente.
- Cuando la cuenta llega a cero puede generar una interrupción al NIOS II.
- El periodo de temporizado (timeout) se define en Qsys (configuración del módulo) o en programa de usuario.
- Se puede cargar una cuenta inicial.
- Se puede leer la cuenta actual.

Referencias:

- Intel (ex. Altera), “Embedded Peripherals IP User Guide”, ug_embedded_ip.pdf
  - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_embedded_ip.pdf
- Intel (ex. Altera), “Nios II Software Developer’s Handbook”, n2sw_nii5v2.pdf
  - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf

Diagrama de bloques del Interval Timer Core

- Cada uno de los registros es de 16 bits de ancho
- El módulo posee un set de 6 registros (modo 32 bits) o un set de 10 registros (modo 64 bits)
  - Un registro de estado (status)
  - Un registro de control
  - Registros para el ingreso del periodo (2 para modo 32 bits y 4 para modo 64 bits)
  - Registros para la captura de la cuenta actual (2 para modo 32 bits y 4 para modo 64 bits)



Detalle de los registro del timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	*(1)*					RUN	TO
1	control	RW	*(1)*			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

5

Detalle de los registro del timer

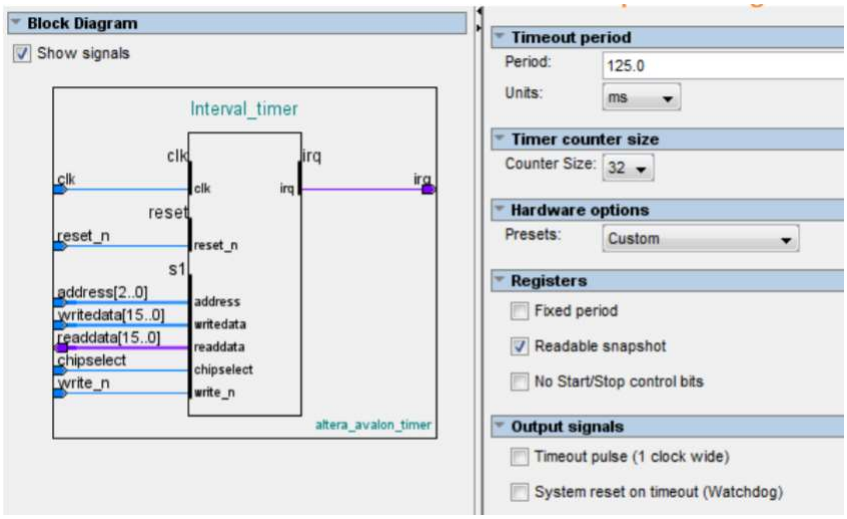
Bit	Name	R/W/C	Description
0	TO	RC	The TO (timeout) bit is set to 1 when the internal counter reaches zero. Once set by a timeout event, the TO bit stays set until explicitly cleared by a master peripheral. Write zero to the status register to clear the TO bit.
1	RUN	R	The RUN bit reads as 1 when the internal counter is running; otherwise this bit reads as 0. The RUN bit is not changed by a write operation to the status register.
0	ITO	RW	If the ITO bit is 1, the interval timer core generates an IRQ when the status register's TO bit is 1. When the ITO bit is 0, the timer does not generate IRQs.
1	CONT	RW	The CONT (continuous) bit determines how the internal counter behaves when it reaches zero. If the CONT bit is 1, the counter runs continuously until it is stopped by the STOP bit. If CONT is 0, the counter stops after it reaches zero. When the counter reaches zero, it reloads with the value stored in the period registers, regardless of the CONT bit.
2	START *(1)*	W	Writing a 1 to the START bit starts the internal counter running (counting down). The START bit is an event bit that enables the counter when a write operation is performed. If the timer is stopped, writing a 1 to the START bit causes the timer to restart counting from the number currently stored in its counter. If the timer is already running, writing a 1 to START has no effect. Writing 0 to the START bit has no effect.
3	STOP *(1)*	W	Writing a 1 to the STOP bit stops the internal counter. The STOP bit is an event bit that causes the counter to stop when a write operation is performed. If the timer is already stopped, writing a 1 to STOP has no effect. Writing a 0 to the stop bit has no effect. If the timer hardware is configured with **Start/Stop control bits** off, writing the STOP bit has no effect.

6

Detalle de los registros del timer

- Registros **period_n**
  - Se especifican en ellos el periodo de temporizado
  - Dependiendo de la resolución especificada, se tendrán dos registros en 32 bits y cuatro registros en 64 bits
  - El valor a introducirse en los registros period_n:
    - $period\_n = periodo\_a\_temporizar * freq\_osc - 1$
    - freq_osc viene a ser el valor del cristal que alimenta al procesador NIOSII (línea clk)
- Registros **snap_n**
  - Para obtener el valor de la cuenta en dicho instante
  - Se hace un proceso de escritura a uno de los registros snap_n, dicha acción hará que la cuenta se copie en los registros snap_n

Ventana de instanciación del Interval Timer Core en Qsys:



Ejemplo: Generar una onda cuadrada de 10% de duty cycle a través de PIO.0 y empleando el módulo Interval Timer Core

Connections	Name	Description	Export	Clock	Base	End	IRQ
	clk_0	Clock Source					
	clk_in	Clock Input	clk				
	clk_in_reset	Reset Input	Double-click to export				
	clk	Clock Output	Double-click to export	clk_0			
	clk_reset	Reset Output	Double-click to export				
	nios2_qsys_0	Nios II Processor					
	clk	Clock Input	Double-click to export	clk_0			
	reset_n	Reset Input	Double-click to export	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			IRQ 31
	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	jtag_debug_module_re...	Reset Output	Double-click to export	[clk]			
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0800	0x0fff	
	custom_instruction_m...	Custom Instruction Master	Double-click to export				
	jtag_uart_0	JTAG UART					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000	0x0007	
	onchip_memory2_0	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to export	clk_0			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x1000	0x1fff	
	reset1	Reset Input	Double-click to export	[clk1]			
	pio_0	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x4000	0x400f	
	external_connection	Conduit	pio_0_external_connection				
	timer_0	Interval Timer					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x6000	0x601f	
	pio_1	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x5000	0x500f	
	external_connection	Conduit	pio_1_external_connection				

- PIO de 8 bits en modo entrada en la dirección 0x5000
- PIO de 8 bits en modo salida en la dirección 0x4000
- Timer de 32 bits con 125ms de timeout, carrera contínua y en la dirección 0x6000

Ejemplo: Generar una onda cuadrada de 10% de duty cycle a través de PIO.0 y empleando el módulo Interval Timer Core

- Se esta preguntando constantemente (polling) a TO si es que es uno (si cuenta llegó a cero).
- La bandera TO debe de ser colocada a cero cuando ocurra el evento de cuenta=0.
- Para este ejemplo el timer se ha configurado en el Qsys para que no haya control de inicio/parada (carrera contínua).

```
#include "sys/alt_stdio.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "system.h"

int main(void)
{
  alt_putstr("Hola del NIOSII ahi tiunamani 4!\n");
  IOWR_ALTERA_AVALON_TIMER_STATUS(0x6000, 0x00);                  //Bajar bandera TO
  IOWR_ALTERA_AVALON_TIMER_PERIODH(0x6000, 0x0000);
  IOWR_ALTERA_AVALON_TIMER_PERIODL(0x6000, 0xFFFF);               //Carga de periodo inicial
  IOWR_ALTERA_AVALON_TIMER_CONTROL(0x6000, 0x06);                 //Timer iniciado y modo continuo
  while(1){
      if ((IORD_ALTERA_AVALON_TIMER_STATUS(0x6000) & 0x01) == 1){   //Pregunto si llego a cero el timer (TO=1)
          if ((IORD_ALTERA_AVALON_PIO_DATA(0x4000) & 0x01) == 1){   //Pregunto si PIO.0 = 1
              IOWR_ALTERA_AVALON_PIO_DATA(0x4000, 0x00);            //Mando PIO.0 a 0
              IOWR_ALTERA_AVALON_TIMER_PERIODH(0x6000, 0x0007);
              IOWR_ALTERA_AVALON_TIMER_PERIODL(0x6000, 0xFFFF);     //Carga de periodo largo
          }
          else{
              IOWR_ALTERA_AVALON_PIO_DATA(0x4000, 0x01);            //Mando PIO.0 a 1
              IOWR_ALTERA_AVALON_TIMER_PERIODH(0x6000, 0x0000);
              IOWR_ALTERA_AVALON_TIMER_PERIODL(0x6000, 0xFFFF);     //Carga de periodo corto
          }
          IOWR_ALTERA_AVALON_TIMER_STATUS(0x6000, 0x00);            //Bajo la bandera TO
      }
  }
  return 0;
}
```

Configuración de la interrupción del Timer

- El módulo Timer puede generar interrupción cuando haga timeout.
- En el QSys se deberá asignar un IRQ al módulo Timer
- Si llega a timeout se levanta la bandera TO y si se habilita previamente el ITO, el Timer enviará una señal de interrupción al procesador NIOSII
- Si activamos la interrupción ya no será necesario estar preguntando constantemente (polling) si TO = 1, se esperaría el evento de interrupción.
- Con esto el CPU puede dedicarse a hacer otra tarea cuando exista la eventualidad (el timeout del Timer en este caso) atenderla.

11

Ejemplo: Generar una señal de PWM a 100Hz y de 5% de Duty Cycle empleando el módulo Timer y su interrupción por timeout

```c
#include "system.h"
#include "sys/alt_irq.h" // interrupt
#include "sys/alt_stdio.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"

unsigned int timer_isr_context; // define global variable to stor

void Timer_Initial(void);
void Timer_ISR(void* isr_context);

int main(void)
{
  alt_putstr("Hola del NIOSII ahi tryout 8!\n");
  Timer_Initial();
  while(1){
      //Rutina principal, sin uso
       }
}

// Inicialización de la interrupción del Timer
void Timer_Initial(void)
{
  void* isr_context_ptr = (void*) &timer_isr_context;
  IOWR_ALTERA_AVALON_TIMER_PERIODH(0x6000, 0x0003);
  IOWR_ALTERA_AVALON_TIMER_PERIODL(0x6000, 0xD08F);          //periodo inicial de temporizacion
  IOWR_ALTERA_AVALON_TIMER_CONTROL(0x6000,
    ALTERA_AVALON_TIMER_CONTROL_START_MSK | // START = 1
    ALTERA_AVALON_TIMER_CONTROL_CONT_MSK  | // CONT  = 1
    ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);   // ITO   = 1    //configuracion del Timer

  // Registro de la interrupcion del Timer
  alt_ic_isr_register(
             0, // Etiqueta del controlador de interrupcion, copiado de system.h
             1, // Numero de interrupcion asignada (IRQ), copiado de system.h
             Timer_ISR, // Sub-funcion de la rutina de interrupcion
             isr_context_ptr, // points to the data structure related to the device driver instance
             0x0); // flags, reserved unused
```

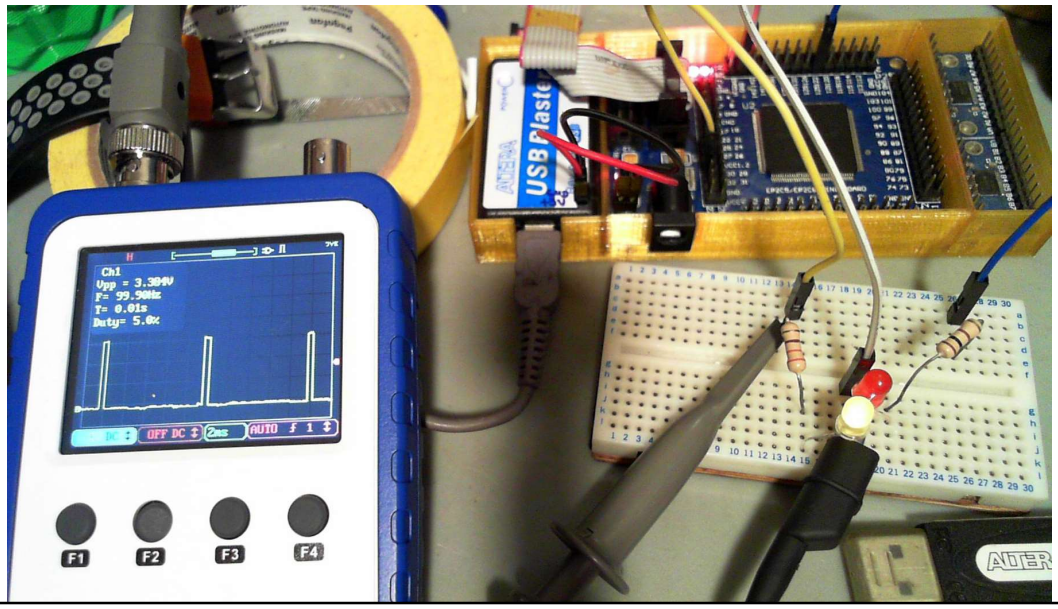
```c
//Sub-funcion de la rutina de interrupcion
void Timer_ISR(void* timer_isr_context)
{
     // Repuesta a la interrupción, bajando bandera TO
  IOWR_ALTERA_AVALON_TIMER_STATUS(0x6000,
      ~ ALTERA_AVALON_TIMER_STATUS_TO_MSK);   // TO = 0

     // Código para la rutina de interrupción
      if ((IORD_ALTERA_AVALON_PIO_DATA(0x4000) & 0x01) == 1){   //Pregunto si PIO.0 = 1
          IOWR_ALTERA_AVALON_PIO_DATA(0x4000, 0x00);            //Mando PIO.0 a 0
          IOWR_ALTERA_AVALON_TIMER_PERIODH(0x6000, 0x0007);
          IOWR_ALTERA_AVALON_TIMER_PERIODL(0x6000, 0x3F78);     //Periodo 0.5ms

      }
      else{
          IOWR_ALTERA_AVALON_PIO_DATA(0x4000, 0x01);            //Mando PIO.0 a 1
          IOWR_ALTERA_AVALON_TIMER_PERIODH(0x6000, 0x0000);
          IOWR_ALTERA_AVALON_TIMER_PERIODL(0x6000, 0x61A7);     //Periodo 9.5ms
      }                                                         //Freq 100Hz 5%DC
}
```

12

Ejemplo: Generar una señal de PWM a 100Hz y de 5% de Duty Cycle empleando el módulo Timer y su interrupción por timeout



13

14