PROYECTO LABOBOT

Juan González Gómez

1/Feb/2002

Resumen

Proyecto realizado como trabajo para el curso de doctorado "Codiseño Hardware-Software de Sistemas Avanzado", curso 2001-2002. El objetivo principal es convertir la plataforma LABOMAT en un sistema de desarrollo para trabajar en robótica, en concreto en el campo de los microrrobots articulados. Las articulaciones de estos robots se implementan con servomecanismos que requieren la generación de una señal PWM para su posicionamiento. Tradicionalmente esta señal se ha generado por software, dedicando muchos recursos del microcontrolador o en su caso, empleando microcontroladores sólo dedicados a esta tarea.

Mediante una plataforma de codiseño, como la tarjeta LABOMAT, es posible delegar esta tarea en un hardware, liberando al microprocesador, que se puede emplear para realizar cálculos de más alto nivel, como la generación de los vectores de posicionamiento y sin consumir CPU en el posicionamiento en sí.

En este proyecto se implementa un módulo hardware encargado de la generación de las señales PWM necesarias para el control de los servos y a través de un interfaz muy sencillo, el software que se ejecuta en el microprocesador puede posicionar estos servos. Se desarrollan para ello unas librerías que permiten que cualquier usuario pueda utilizarlas para el control de robots articulados sin tener que conocer los detalles internos.

Como aplicación práctica, se controlan 4 servos que permiten orientar dos minicámaras y que mediante un programa se mueven según una secuencia de movimiento fija.

Otro de los objetivos es optimizar el diseño hardware lo máximo posible para valorar cuánto de buena es la solución SW/HW frente a la puramente SW. Esta valoración no viene determinada por la velocidad, sino más bien por los recursos consumidos. Con un microcontrolador de 8 bits se pueden controlar 4 servos, quedando todavía CPU para la aplicación en sí. Cuantos más servos se puedan controlar desde el Hardware, mejor será esta solución. Será por tanto objetivo de este trabajo el determinar el número máximo de ellos que se pueden controlar.

Índice

Índice de figuras

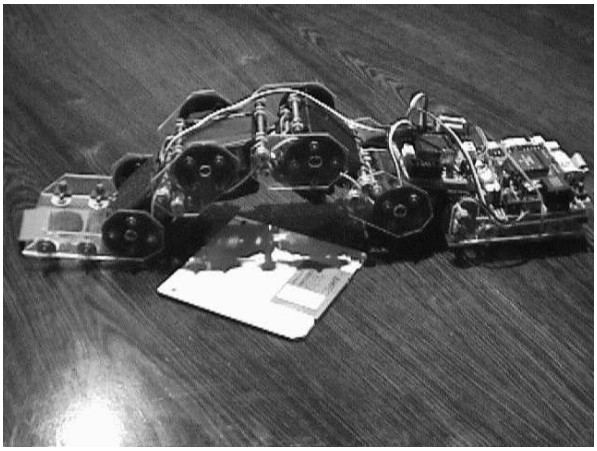


Figura 1: Gusano Robot “CUBE”, construido con 4 servos del tipo Futaba 3003

1. Introducción

El empleo de servomecanismos de bajo precio para la creación de robots prototipos se está volviendo muy popular. Así existen hexápodos[2], brazos articulados[4], robots artículados de diferentes tipos[3][5] y hasta robots humanoides [1], todos ellos realizados con servos muy similares. En la figura 1 se muestra un gusano robot, desarrollado por el autor de este proyecto y que está constituido por 4 servos que conforman las 4 articulaciones. Está controlado por dos microcontroladores 6811 de motorola, uno dedicado exclusivamente a la generación de las señales de control de los servos y otro que almacena y genera las secuencias de movimiento.

Para posicionar este tipo de servos hay que introducirles una señal PWM de unas ciertas características, que dependiendo del ciclo de trabajo hace que el servo se mueva hacia una u otra posición.

En la mayoría de los prototipos, esta señal la genera un microcontrolador, es decir, se obtiene mediante software, aunque algunos microcontroladores disponen de hardware específico que permite su generación.

En la mayoría de los microcontroladores que se emplean para estas aplicaciones de robótica de bajo presupuesto, no está disponible un módulo hardware para posicionar los servos, teniéndose que realizar por software con la consecuente pérdida de recursos que se podrían dedicar a otras tareas. También ocurre que es el software el que se tiene que adaptar al hardware disponible en el microcontrolador: temporizadores, comparadores... no adaptándose al 100 %.

Un enfoque muy interesante es el de diseñar un módulo hardware que se implemente en una FPGA de manera que liberere al microcontrolador de todas las operaciones de posicionamiento del servo, además de hacer abstracción de los mecanismos necesarios para ello, es decir, el poder posicionar un servo sin tener que saber el mecanimos necesario para ello. De esta manera, pequeños microcontroladores de 8 bits podrán generar fácilmente las secuencias de movimiento de varios servos perdiendo muy pocos recursos en ello.

Con este proyecto se pretende utilizar la plataforma LABOMAT para convertirla en un sis-

tema de desarrollo para robótica, comprobando la viabilidad de utilizar una solución basada en SW/HW frente a la puramente SW y determinar cuánto más de buena o mala es. Como aplicación práctica se desarrolla un programa que permite orientar dos minicámaras, controlándose 4 servos. Para ello se contruye un pequeño módulo (la placa PLM-3003) que por un lado se conecta a la LABOMAT y por otro se le conectan los 4 servos que mueven las cámaras.

2. Objetivos

- Estudio de las ecuaciones de los parámetros del PWM para poder optimizar el hardware lo máximo posible, así como determinar la relación entre optimización y precisión en el posicionamiento del servo.
- Desarrollo de un hardware optimizado capaz de generar una señal PWM necesaria para posicionar servos del tipo Futaba 3003, con una precisión determinada. (A nivel de bloques lógicos).
- Implementación en VHDL para ejecutarlo como un proceso hardware en la FPGA 4013 de la tarjeta LABOMAT.
- Desarrollo de unas librerías de acceso al hardware, para poder realizar programas de usuario que hagan abstracción de los detalles internos.
- Determinar el número máximo de servos que se pueden llegar a controlar
- Aplicación: movimiento de dos cámaras con dos grados de libertad cada una.
- Valorar la viabilidad del control software-hardware frente al puramente software.

3. Organización del trabajo

En el apartado 4 se describe el servomecanismo Futaba 3003, que es con el que se trabajará. Casi todos los servos son compatibles con el Futaba 3003, que se está conviertiendo en un estándar "de facto", por lo que todo lo desarrollado con este proyecto será válido para otros modelos de servos. Se presta especial atención a la señal de control, y se definen una serie de parámetros que la caracterizan.

En el apartado 5 es donde se encuentra lo más gordo del trabajo. Se propone un circuito de control genérico y se obtienen las ecuaciones necesarias que permiten su dimensionamiento, de manera que se pueda optimizar lo máximo posible según la precisión requerida para la aplicación. Se aplican las fórmulas para obtener un esquema hardware con una precisión suficientemente buena para la aplicación del movimiento de las minicámaras. El modelo hardware se amplía para el control de varios servos.

Todos los programas en VHDL se encuentran en el apartado 6, mostrándose los correspondientes a dos versiones del proyecto, la 0.6 en la que sólo se mueve un servo y ha servido para

Figura 2: Servomecanismos Futaba 3003

validar el modelo así como realizar las mediciones oportunas y la versión 1.0 que se corresponde con la aplicación práctica del movimiento de las minicámaras. En el apartado 7 se presenta el software desarrollado para ambas versiones, la 0.6 y la 1.0, que permite comunicarse con el hardware para establecer la posición de los servos.

El apartado 8 se dedica a la explicación de la aplicación práctica y finalmente en los apartados 9 y 10 se muestran los resultados obtenidos y las líneas futuras de trabajo.

4. Descripción de los servos Futaba 3003

4.1. Introducción

Los servos futaba modelo 3003 son muy empleados en robótica de bajo presupuesto por sus características de reducido precio, alto par y bastante buena precisión. En la figura 2 se muestra el aspecto de uno de estos servos.

Una de las principales ventajas es su reducido tamaño y la posibilidad de desmontarlo completamente, pudiendo convertirlo en un motor de corriente continua normal (eliminando el circuito de control) o conectando un cable al potenciómetro interno de manera que nuestro sistema de control pueda cerrar el bucle.

El movimiento de estos servos se realiza en “bucle abierto”. Es decir, se le envía una señal de control que le indica la posición en la que debe situarse el eje, pero no tenemos una señal de realimentación que nos indique cuando se ha realizado la operación. El bucle lo cierra interna-

mente el propio Servo. Esta forma de control es muy buena en aplicaciones en las que el servo tiene fuerza de sobra para poderse posicionar, sabiendo que no van a existir obstáculos y que transcurrido un tiempo máximo el servo se posicionará.

4.2. Características

En [7] se pueden encontrar las características de toda la familia de servos Futaba. Las del servo S3003 se resumen a continuación:

Alimentación:	4.5-6v
Velocidad:	0.23 seg/60^o
Par de salida:	3.2 Kg-cm
Tamaño:	40.4 x 19.8 x 36mm
Peso:	37.2g
Recorrido del eje:	180^o

En la figura 3 se ha dibujado el esquema de un servo acotado en milímetros.

4.3. Conexionado

El conexionado del servo con los circuitos de control se realiza a través de un conector plano hembra de 3 vías al que se encuentran conectados los tres cables de acceso al servo. Estos cables son:

- **Rojo**: Alimentación
- **Negro**: Masa (GND)
- **Blanco**: Señal de control

En la figura 4 se muestra cómo es el conector de estos servos y los cables de acceso al servo.

4.4. Señal de control

El control de la posición de los servos se hace aplicando una señal de control como la mostrada en la figura 5. Se trata de una señal periódica, de periodo T=20ms, que se encuentra activa sólo durante un pequeño intervalo de tiempo (T_{on}). Es la anchura de este pulso la que determina la posición del eje del servo. En la figura se han mostrado las dos posiciones extremas. El resto de posiciones se obtienen linealmente cuando la anchura del pulso varía entre 0.3 y 2.3 ms. Así por ejemplo, para situar el servo en su posición central habrá que aplicar un ancho de 1.3ms ($T_{on} = 1{,}3ms$).

Mientras el servo reciba "señal" permanecerá en la posición indicada, ejerciendo "fuerza" (y por tanto consumiendo de manera proporcional a la fuerza que esté realizando para mantenerse

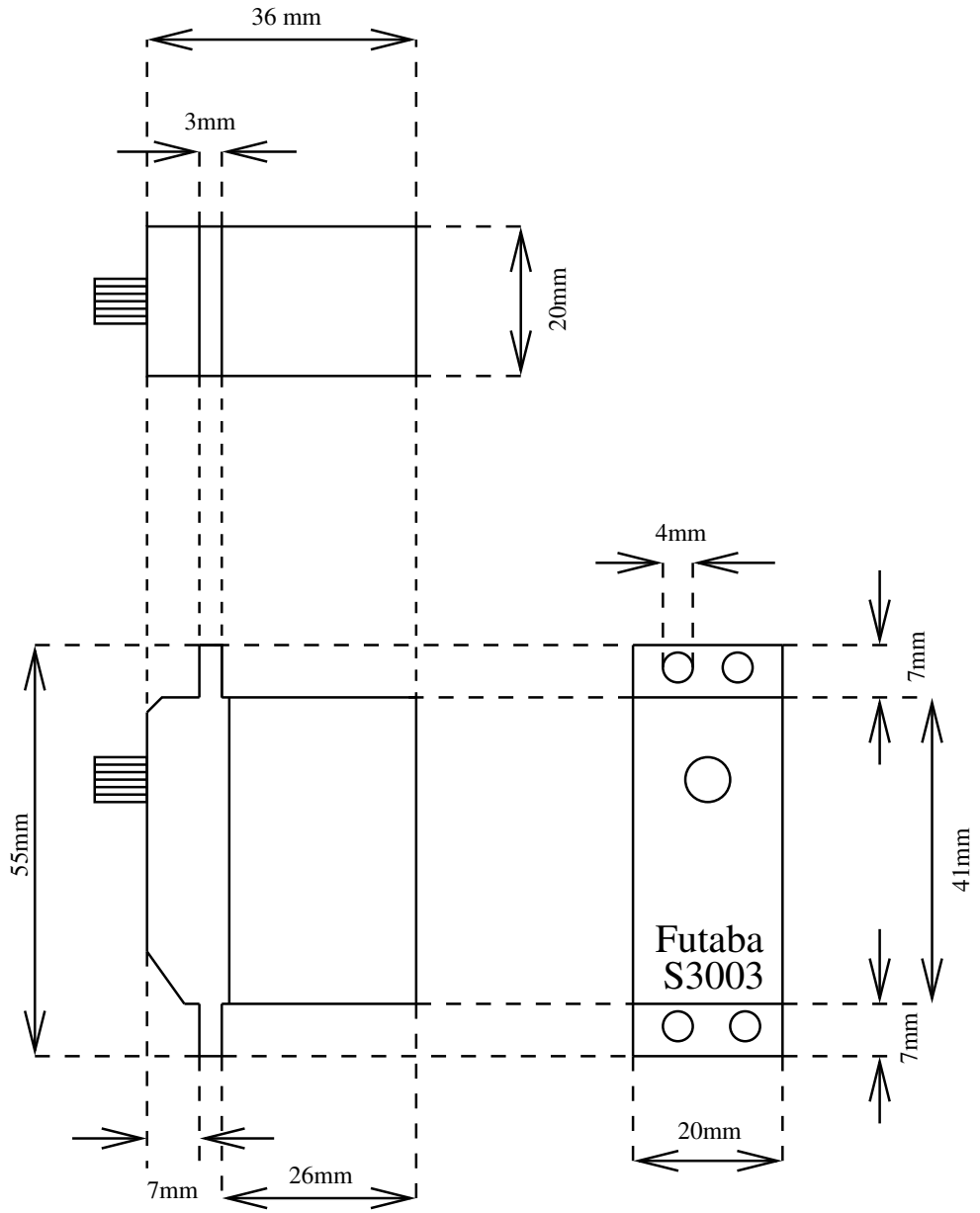


Figura 3: Esquema de las dimensiones del servo F3003

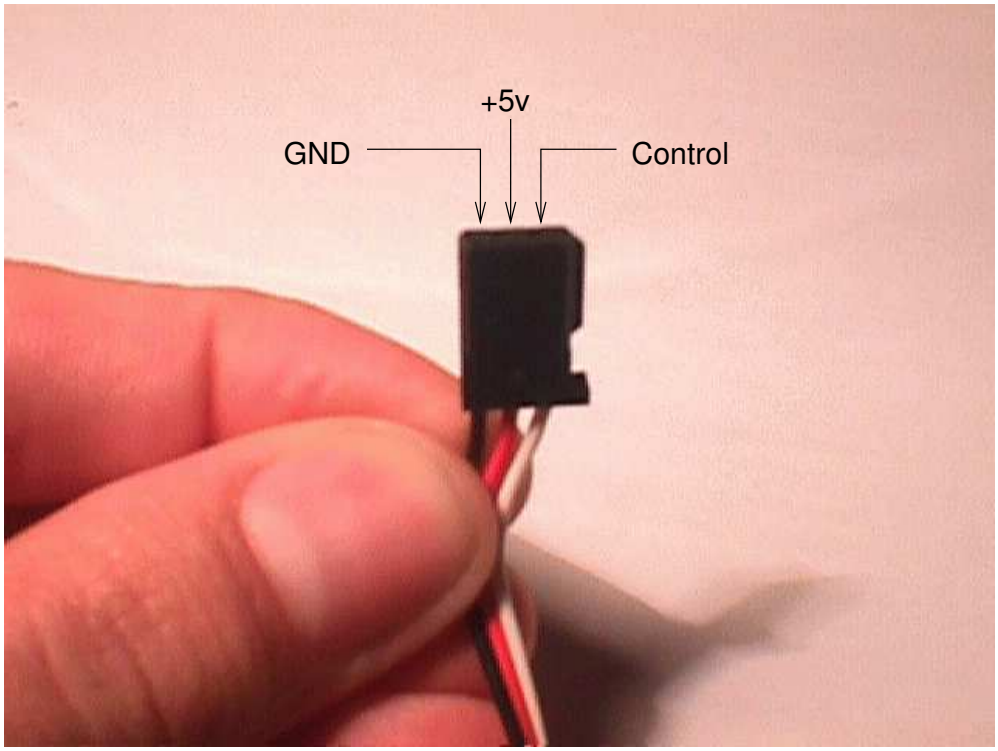


Figura 4: Cables de acceso al servo

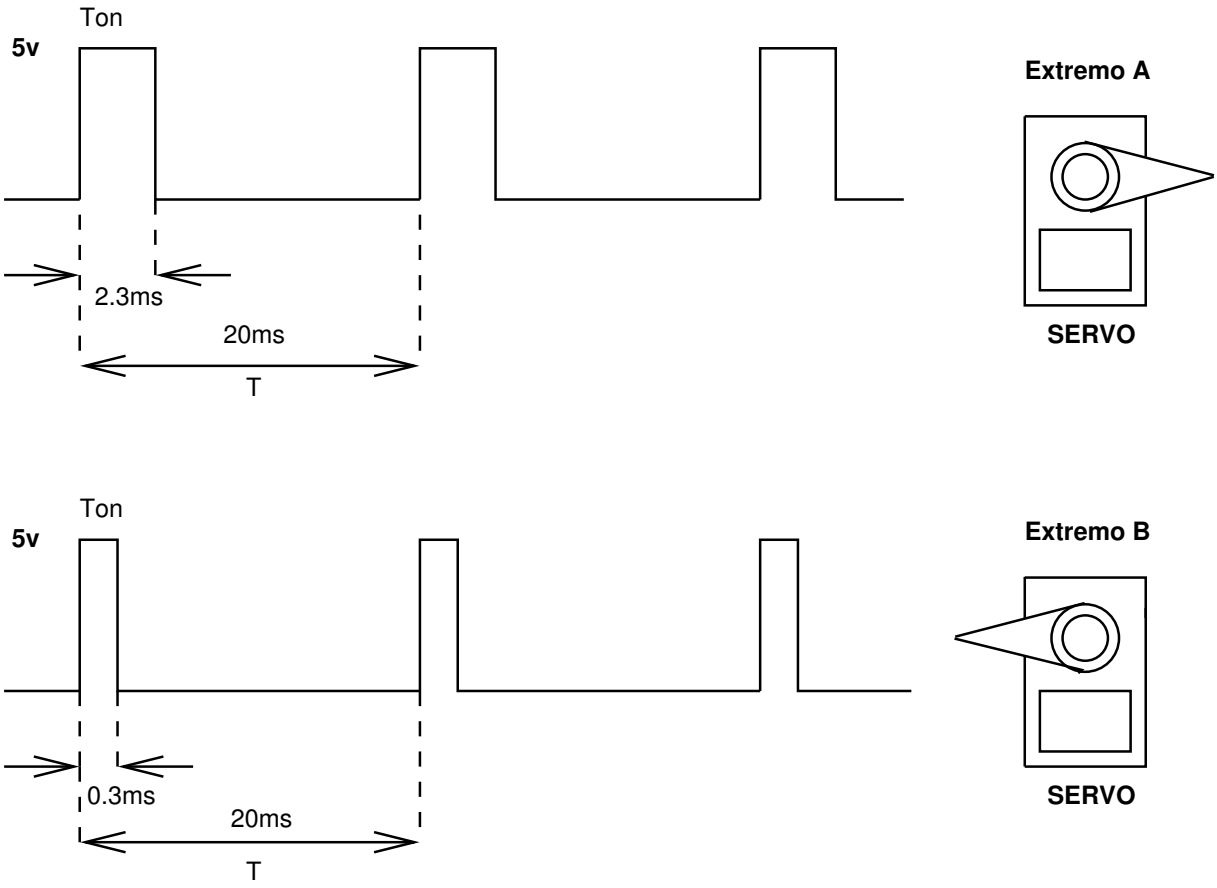


Figura 5: Señal de control del Futaba S3003

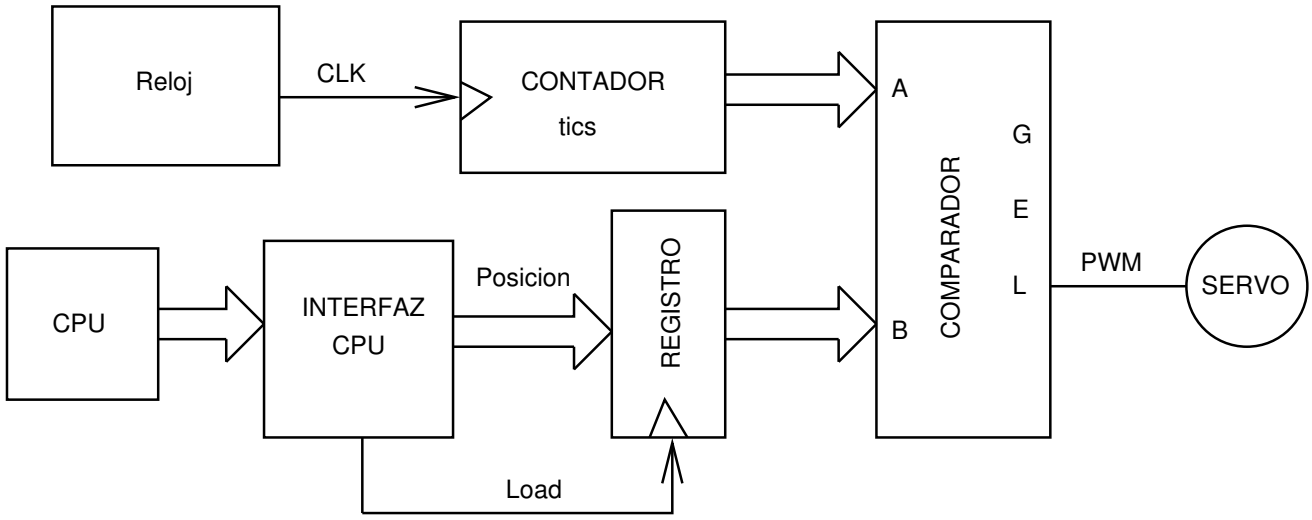


Figura 6: Esquema inicial para la generación de una señal PWM de control de un servo

en ella). Si la señal se deja de aplicar, el servo no ejerce fuerza y se puede mover con la mano (consumiendo sólo lo necesario para la alimentación de su circuito de control).

La nomenclatura empleada para los parámetros de la señal PWM es la siguiente, que será empleada en el apartado 5.1.2 para realizar cálculos:

- T: Periodo de la señal.
- $T_{on}(min)$= Anchura mínima de la señal PWM[1]
- $T_{on}(max)$= Anchura máxima de la señal PWM
- T_{on} : Tiempo durante el cual la señal de PWM está activa. $T_{on} \in [T_{on}(min), T_{on}(max)]$
- W : Margen dinámico en anchura de la señal de PWM: $W = T_{on}(max) - T_{on}(min) = 2ms$

5. Diseño hardware

5.1. Arquitectura para el control de un servo

En la figura 6 se muestra un primer esquema para la generación de la señal PWM de control de un servo. El diseño es muy simple. Mediante el módulo interfaz, el software envía la posición del servo que es almacenada en un registro, que denominaremos **registro de posición**. El reloj

[1]A lo largo del trabajo emplearemos la palabra "anchura" para referirnos a la anchura de la parte de la señal PWM que está activa. Para referirnos a la anchura total de la señal utilizaremos el periodo T.

del sistema incrementa un contador, de manera que se lleva la cuenta de cuántos tics de reloj han transcurrido desde que inicialmente se encontraba a cero.

Mediante un comparador se comprueba si el número de tics recibidos es menor que la posición especificada. Esta posición, que es en realidad el ancho del pulso de la señal PWM, se expresa también en tics de reloj, por lo que mientras que los tics contados sean menores que los tics del ancho del pulso, la señal de salida del comparador, L, permanecerá a nivel alto. En cuanto los tics superen este valor, la señal L se pone a cero.

Tanto el periodo T de la señal PWM como el ancho del pulso (T_{on}) se especifican en tics del reloj principal, de manera que cuanto mayor sea la frecuencia del reloj, mayor número de tics habrá que contar para tener el mismo ancho del pulso (el sistema necesitará un mayor número de bits) y cuando menor frecuencia el error en la determinación del ancho exacto es mayor , el error de discretización es mayor.

5.1.1. Caracterización de parámetros

- **Parámetros de la señal de reloj principal**:
  - T_{clk}: Periodo de la señal de reloj, o lo que es lo mismo, duración de un tic de reloj.
  - $F_{clk} = \frac{1}{T_{clk}}$: Frecuencia de la señal de reloj.
- **Parámetros de la señal PWM**: Se pueden expresar en unidades de tiempo como los definidos en el apartado 4.4, pero también se pueden "discretizar" y expresarse en tics de reloj.
  - **De tiempo continuo**:
    - $T, T_{on}, W, T_{on}(min), T_{on}(max)...$
  - **De tiempo discreto (en tics)**:
    - N_T: Periodo
    - $N_{T_{on}}$: Anchura de la señal PWM
  - Ambos tipos de parámetros están relacionados por el valor del periodo del reloj principal (T_{clk})
    - $T = N_T.T_{clk}$
    - $T_{on} = N_{T_{on}}.T_{clk}$
- **Parámetros de precisión**:
  - P: Precisión del posicionamiento del servo (grados/tic). Lo que se mueve el servo por cada tic de reloj.
- **Parámetros del hardware**:
  - B_c: Número de bits del contador
  - B_r: Número de bits del registro

5.1.2. Dimensionamiento

5.1.2.1. Planteamiento

Tomando como punto de partida la precisión en la posición del servo (P) determinaremos los siguientes parámetros para poder implementar el esquema de la figura 6:

- **Número de bits del contador** (B_c)
- **Número de bits del registro** (B_r)
- **Frecuencia de la señal CLK (**F_{clk}**)** (o su periodo T_{clk})

La generación de la señal de PWM utilizando una arquitectura como la de la figura 6 implica una discretización. Hay que obtener la frecuencia mínima del reloj ($F_{clk}(min)$) o su periodo máximo ($T_{clk}(max)$) que garanticen una precisión de P grados. A partir de estos datos se pueden conocer todos los demás expresados en tics de reloj y a partir de ahí obtener los bits necesarios para el contador y el registro.

Los cálculos se tienen que hacer en dos fases. En la primera se determina N_T a partir de la preción P con lo que calculamos el número mínimo de bits necesarios para el contador y el registro de posición. Sin embargo, el contador debe ser módulo N_T, es decir, que una vez que haya contado todos los tics del periodo de la señal, deberá comenzar desde cero. Por ello hay que tomar $N_T = 2^{B_c}$ y volver a calcular los parámetros anteriores, obteniéndose una nueva precisión P' que será mejor o igual que la precisión tomada inicialmente para los cálculos (P'<=P).

5.1.2.2. Ecuaciones

- **Margen dinámico discreto**: El máximo recorrido del eje del servo es de 180^o y la precisión de P($^o/tic$), lo que implica que el margen dinámico sea:

$$N_w = \frac{180}{P}\, tics \tag{1}$$

- **Valores extremos del reloj principal**: Teniendo en cuenta que el margen dinámico es $W = N_w.T_{clk} = 2ms$, y sustituyendo N_W según la ecuación 1, obtenemos el periodo máximo de la señal de reloj que permite tener una precisión en la posición de P($^o/tic$):

$$T_{clk}(max) = \frac{P}{90}\, ms \tag{2}$$

  y su inversa nos determina la frecuencia mínima de funcionamiento:

$$F_{clk}(min) = \frac{90}{P}\, KHZ \tag{3}$$

- **Periodo discreto de la señal PWM**: El periodo de la señal PWM se divide en N_T tics de reloj. Partiendo de :

$$T = N_T T_{clk} \;\Rightarrow\; N_T = \frac{T}{T_{clk}} = 20.\frac{90}{P} = \frac{1800}{P}$$

obtenemos la ecuación:

$$N_T = \frac{1800}{P} \tag{4}$$

Cuanta más precisión en la posición del servo (menor valor de P), mayor será el número de tics que se deben contar y por tanto mayor será el número de bits B_c del contador, por lo que **cuanto mayor precisión queramos, más recursos emplearemos en la implementación**.

- **Anchura máxima discreta de la señal PWM**: Discretizando la anchura máxima de la señal PWM, obtenemos la anchura máxima en tics de reloj, que nos servirá para determinar el número de bits (B_r) del registro de posición:

$$T_{on}(max) = N_{T_{on}}(max).T_{clk} \;\Rightarrow\; N_{T_{on}}(max) = \frac{T_{on}(max)}{T_{clk}} = \frac{2{,}3}{T_{clk}} = \frac{207}{P} tics$$

$$N_{T_{on}}(max) = \frac{207}{P} tics \tag{5}$$

- **Número de bits del contador**: Se obtienen a partir de la inecuación:

$$2^{B_c} \geq N_T$$

que indica que el contador debe tener los suficientes bits como para poder contar los tics de reloj del periodo de la señal PWM, al menos. Despejando B_c de esta fórmula:

$$B_c = E\left[log_2 N_T + 1\right] \tag{6}$$

- **Número de bits del registro de posición**: Se obtienen de manera similar al número de bits del contador, a partir de la inecuación:

$$2^{B_r} \geq N_{T_{on}}(max)$$

que expresa que necesitaremos un número de bits suficiente como para almacenar la anchura máxima de la señal PWM:

$$B_r = E\left[log_2 N_{T_{on}}(max) + 1\right] \tag{7}$$

5.1.2.3. Proceso de cálculo

Objetivo: Obtener el periodo de la señal de reloj (T_{clk}) o su frecuencia(F_{clk}) así como el mínimo número de bits del contador y del registro de posición, a partir de la precisión en la posición (P). Además el contador tiene que ser módulo N_T , de manera que una vuelta completa de este contador tarde lo mismo que el periodo de la señal PWM.[2]

Teniendo esto en cuenta, la forma de realizar los cálculos es la siguiente:

1. **Fase 1**: Cálculo del número de bits del contador (B_c)
   - *a*) Tomando como partida un valor para la precisión de la posición (P), aplicar la fórmula 4 para obtener un primer valor de N_T.
   - *b*) Aplicando 6 obtenemos el número de bits del contador: B_c.
2. **Fase 2**: Cálculo del resto de parámetros
   - *a*) A partir de B_c obtenemos un segundo valor para N_T, que será el definitivo: $N_T = 2^{B_c}$
   - *b*) Calculamos el valor del periodo del reloj $T_{clk} = \frac{20}{N_T} ms$ (o de su frecuencia). Este valor cumplirá que $T_{clk} < T_{clk}(max)$
   - *c*) A partir de la ecuación 7 obtenemos el número de bits del registro de posición (B_r)
   - *d*) El nuevo valor de N_T nos permite calcular la precisión final P'(Ecuación 4).

5.1.2.4. Aplicación Aplicaremos las fórmulas para calcular los parámetros de dimensionamiento de la unidad de PWM para una precisión inicial de P=1 y P=0.5.

- **Parámetros para una precisión inicial de P=1**

1. **Fase 1**:
   - *a*) $N_T = \frac{1800}{P} = \frac{1800}{1} = 1800\, tics$
   - *b*) Para representar un valor de 1800 necesitamos al menos 11 bits. $B_c = 11\, bits$
   - *c*) Para esta precisión, la frecuencia mínima del reloj es: $F_{clk}(min) = \frac{90}{P} = \frac{90}{1} = 90\, KHZ$
2. **Fase 2**:
   - *a*) Un contador de 11 bits cuenta $2^{11} = 2048$ eventos. Ese será el nuevo valor de N_T. $N_T = 2048\, tics$
   - *b*) El periodo de reloj será: $T_{clk} = \frac{20}{N_T} = \frac{20}{2048} = 0{,}009766\, ms = 9{,}766\, \mu s$. Se necesitará un reloj de frecuencia: $F_{clk} = 102{,}4\, KHZ$, que es mayor que la frecuencia mínima calculada anteriormente.

[2] Si no se cumpliese esta restricción, habría que añadir lógica adicional para que cuando el contador alcanzase el valor N_T, volviese a contar desde cero. Si el contador es módulo N_T esta lógica adicional no es necesaria, con lo que se ahorran recursos en la FPGA y se podrían meter más unidades de PWM.

c) $N_{T_{on}}(max) = \frac{2,3}{T_{clk}} = 236\,tics$. Necesitamos 8 bits para almacenar ese número máximo. $B_r = 8\,bits$

d) $P' = \frac{1800}{N_T} = \frac{1800}{2048} = 0{,}88\,^o$/tic. La precisión final es sensiblemente mejor que la precisión inicial, debido a que hay más tics de reloj de menor anchura.

- **Parámetros para una precisión inicial de P=0.5**

1. **Fase 1**:

   a) $N_T = \frac{1800}{P} = \frac{1800}{0{,}5} = 3600\,tics$

   b) Necisitaremos al menos 12 bits. $B_c = 12\,bits$

   c) Frecuencia mínima del reloj: $F_{clk}(min) = \frac{90}{P} = \frac{90}{0{,}5} = 180\,KHZ$

2. **Fase 2**:

   a) Nuevo valor del periodo discreto: $N_T = 2^{12} = 4096\,tics$

   b) Periodo del reloj: $T_{clk} = \frac{20}{N_T} = \frac{20}{4096} = 0{,}004883\,ms = 4{,}883\,\mu s$. Se necesitará un reloj de frecuencia: $F_{clk} = 204{,}79\,KHZ$

   c) $N_{T_{on}}(max) = \frac{2,3}{T_{clk}} = 471\,tics$. Necesitaremos 9 bits para almanecar la posición. $B_r = 9\,bits$

   d) $P' = \frac{1800}{N_T} = \frac{1800}{4096} = 0{,}44\,^o$/tic.

- **Tabla resumen**:

	Precisión (P)	
	P=1	**P=0.5**
$T_{clk}(max)$	11.11 μs	5.55 μs
$F_{clk}(min)$	90 KHZ	180 KHZ
T_{clk}	9.766 μs	4.883 μs
F_{clk}	102.4 KHZ	204.79 KHZ
B_c	11 bits	12 bits
B_r	8 bits	9 bits
$N_{T_{on}}(max)$	236 tics	471 tics
$N_{T_{on}}(min)$	31 tics	62 tics
P'	0.88 ^{o}/tic	0.44 $^o/tic$

Los parámetros $N_{T_{on}}(max)$ y $N_{T_{on}}(min)$ marcan los valores máximo y mínimo que se pueden almacenar en el registro de posición, que se corresponden con las posiciones extremas del servo.

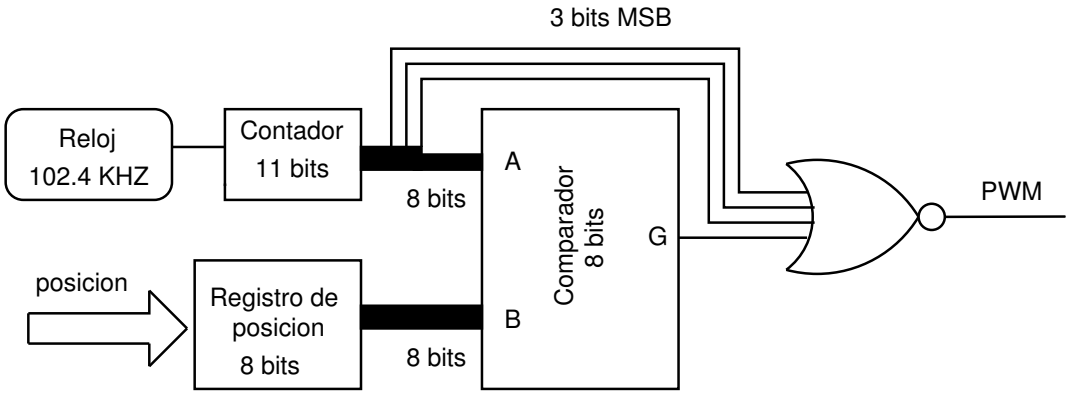


Figura 7: Arquitectura detallada de una unidad PWM de precisión máxima P=1.

5.1.3. Arquitectura detallada para una precisón máxima de P=1

Una precisión de 1^o es suficiente para la mayoría de las aplicaciones con este tipo de servos. Además el que el registro de posición sea de 8 bits es muy adecuado para enviar esta información desde un microcontrolador de 8 bits de una manera sencilla, utilizando sólo un puerto de 8 bits o mediante un bus serie (como el SPI de Motorola).

Con los datos obtenidos en el apartado 5.1.2.4, para una precisión máxima de P=$1^o/tic$, y teniendo en cuenta que hay que minimizar el diseño lo máximo posible para conseguir el mayor número de controladores PWM en la FPGA, se puede dar una arquitectura más detallada del sistema de generación de PWM, que se muestra en la figura 7.

Es necesario un contador de 11 bits y un comparador de 8 bits. Del contador se segregan los tres bits más segnificativos, introduciéndose los 8 restantes en el comparador. Los 8 bits del registro de posición se comparan directamente con los 8 bits de menor peso del contador. Los tres bits de mayor peso del contador, junto con la señal G del comparador (que se activa siempre que A>B, o sea, cuando el contador es mayor que el registro de posición) se introducen en una puerta NOR para obtener la señal PWM de salida.

A continuación veremos la justificación de esta puerta NOR. Llamaremos C_{10}, C_9, C_8 a los 3 bits más significativos del contador. Puesto que el comparador es de 8 bits, la señal L (A<B) se activará siempre que los 8 bits de menor peso del contador sean menores que los 8 bits del registro de posición. Sin embargo esto sólo es válido siempre que los bit de mayor peso del contador sean 0 ($C_{10} = C_9 = C_8 = 0$). En el caso de que alguno de estos sea '1', el contador tendrá un número mayor que la posición y por tanto la señal PWM deberá estar a cero. A la salida del comparador habrá que añadir una lógica adicional que tenga esto en cuenta. Habrá que diseñar un circuito combinacional que tengas como entradas C_{10}, C_9, C_8 y L, y que vendrá descrito por la siguiente tabla:

C_{10}	C_9	C_8	L	**PWM**	**Descripcion**
0	0	0	0	0	El valor del contador es MAYOR o IGUAL que el del registro de posición
0	0	0	0	1	El valor del contador es menor que el del registro de posición
0	0	1	x	0	El valor del contador es MAYOR que el del registro de posición. La señal L hay que ignorarla
0	1	0	x	0	idem
0	1	1	x	0	idem
1	0	0	x	0	idem
1	0	1	x	0	idem
1	1	0	x	0	idem
1	1	1	x	0	idem

Desarrollando la señal PWM se obtiene que:

$$PWM = L.\overline{C_{10}}.\overline{C_9}.\overline{C_8}$$

y si en lugar de la señal L tomamos la señal G del comparador (Ya que podemos considerar que aproximadamente L=$\overline{G}$, descartando la igualdad), la ecuación queda:

$$PWM = \overline{G}.\overline{C_{10}}.\overline{C_9}.\overline{C_8} = \overline{G + C_{10} + C_9 + C_8} = NOR(G, C_{10}, C_9, C_8)$$

Es decir, que con una única puerta NOR de 4 entradas implementamos la función PWM.

5.1.4. Arquitectura final, con precisión máxima de P=1

A la hora de implementar el esquema del apartado 5.1.3 en la tarjeta LABOMAT aparece un problema con el reloj. La señal de reloj que se aplica a la FPGA es programable y se puede variar en un amplio rango. Sin embargo, al configurarlo para trabajar a 102.400 HZ, no aparece ningún mensaje de error y aparentemente se ha configurado correctamente, pero al medir con el osciloscopio se comprueba que esta señal se ha disparado a valores superiores a 50MHZ.

En general, señales menores de 400KHZ producen este efecto. Para solucionarlo **hay que configurar el reloj a una frecuencia de 409.600 HZ** e implementar un divisor de frecuencia que divida esta señal entre 4, de manera que se obtengan los 102.400 KHZ necesarios. Este divisor lo denominaremos "prescaler", quedando la arquitectura definitiva de la unidad de PWM como la mostrada en la figura 8.

Dentro de la línea de puntos se ha encerrado la unidad de pwm, que consta de las siguientes señales de interfaz:

- **Señales de entrada:**
  - **Clk**: Señal de reloj de frecuencia 102.4KHZ
  - **Reset**: Señal de reset para la inicialización del contador del PWM

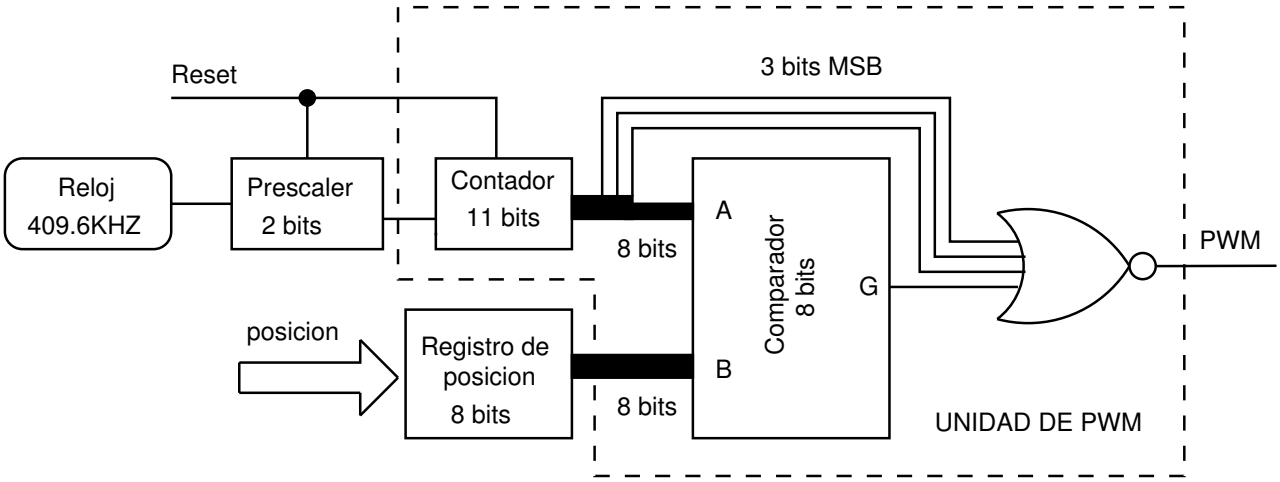


Figura 8: Arquitectura final de la unidad PWM, para una precisión máxima de 1 grado/tic.

- **Posición**: Dato de 8 bits que indica la posición, en tics de reloj, en la que se quiere situar el servo. Este valor debe estar comprendido entre los parámetros $N_{T_{ON}}(min)$ y $N_{T_{ON}}(min)$, que para esta precisión valen 31 y 236 respectivamente y que se corresponden con los extremos opuestos del eje de salida (Separados 180 grados).

- **Señales de salida**:

  - **PWM**: Señal de control que se introduce directamente por el pin de control del servo.

5.2. Arquitectura para el control de varios servos

Aprovechando que en la tarjeta LABOMAT las transferencias de información entre la CPU y la FPGA pueden ser de 32 bits, el esquema propuesto para el control de varios servos se muestra en la figura 9. El sistema tiene un registro de 32 bits por cada 4 servos a controlar. Cada grupo de 8 bits determina la posición de un servo. Para un mayor aprovechamiento de los recursos de la FPGA, el número de servos a controlar debe ser múltiplo de 4. Así por ejemplo, para controlar 8 servos se requerirían 2 registros de 32 bits. Si el número no es múltiplo de 4 se estará desaprovechando parte de la información de uno de los registros de 32 bits.

En la figura 9 se han delimitado por líneas de puntos las partes del hardware que están dentro de la FPGA. Tanto la señal de reset como el reloj están en la tarjeta LABOMAT. Este es el esquema que se emplea para la implementación de la aplicación práctica. (Apartado 8)

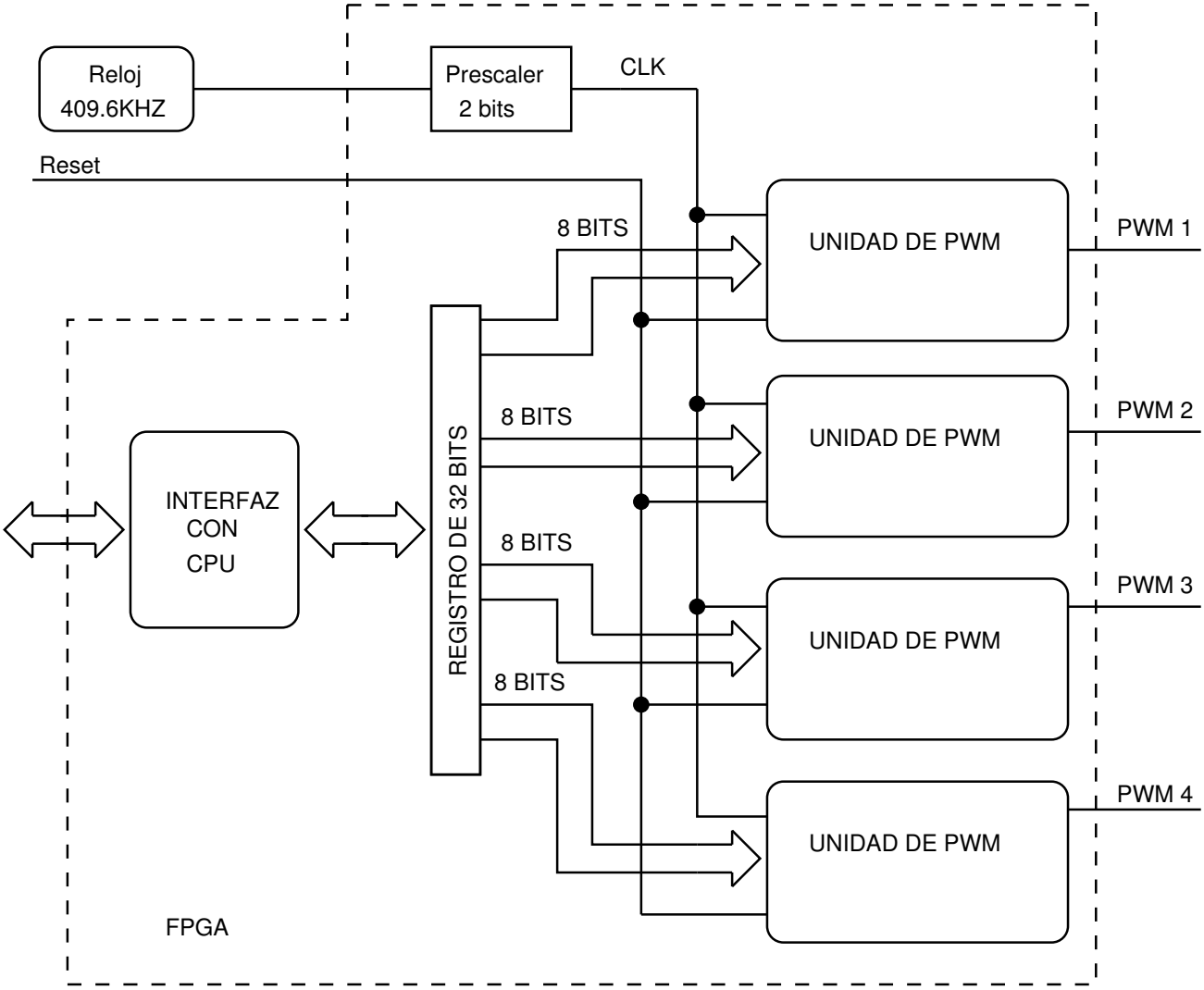


Figura 9: Arquitectura del hardware para el control de 4 servos

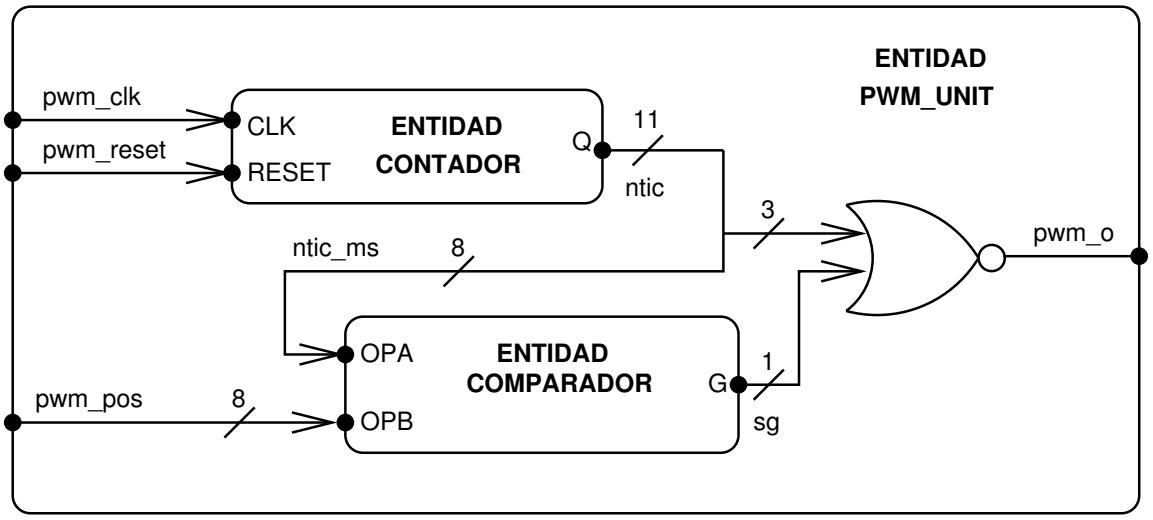


Figura 10: Entidad pwm_unit y entidades que están por debajo

6. PROGRAMAS EN VHDL

6.1. Arquitectura

Vamos a describir dos versiones del proyecto. La versión 0.6 en la que sólo se mueve un servo y que se ha empleado para validar el funcionamiento de la unidad de pwm y la versión 1.0 que se corresponde con la aplicación práctica del movimiento de los 4 servos para la orientación de las minicámaras. Estas dos versiones sólo se diferencian en la entidad de superior (Top-level), siendo las inferiores las mismas para ambas versiones. El software que se ejecuta en la CPU es también diferente.

6.1.1. Version 0.6: Control de un servo

La versión 0.6 permite posicionar sólo un servo y se ha empleado para realizar mediciones con el osciloscopio y comprobar que la parte de interfaz entre la CPU y la FPGA funciona correctamente. El interfaz no se ha optimizado, sino que se ha tomado el desarrollado para el ejemplo del multiplicador[8] y se ha adaptado para el manejo de la unidad PWM en vez del multiplicador. Por ello dispone de dos registros de 32 bits, mapeados en las direcciones 0x16000000 y 0x16000004, que se pueden leer y escribir. Sólo el byte más bajo del primer registro es el que se utiliza para posicionar el servo.

En la figura 10 se ha dibujado la entidad **pwm_unit**, especificándose los puertos de entrada, salida y las señales intermedias usadas, así como las entidades de menor nivel empleadas: entidad **comparador** y entidad **contador**.

El diseño superior de la jerarquía para la versión 0.6 se muestra en la figura 11. Está constituida por una máquina de estados que actúa de interfaz entre la CPU y la FPGA y dos registros de 32 bits que se pueden leer y escribir. Los 8 bits de menos peso del registro 0 son los que

se introducen en la entidad pwm para el posicionamiento del servo. El reloj principal (clk) se introduce en la **entidad prescaler** para dividirlo por 4 y obtener así una frecuencia de 102.4KHZ (clk_4) que es la que necesita la entidad **pwm_unit**. El objetivo de la **entidad perif_access** es el de poder validar el correcto funcionamiento de la unidad de PWM, sabiendo que la máquina de estados de interfaz funciona correctamente, puesto que sólo se ha modificado la salida del registro 0, que en vez de introducirse como entrada del multiplicador, se hace como entrada de la unidad pwm. Es en la versión 1.0 donde se modifica este interfaz para adaptarlo exactamente a las necesidades de la aplicación.

Los ficheros vhdl empleados son los siguientes, que serán los mismo que para la versión 1.0 salvo el de la entidad superior.

- **comparador.vhd**: Entidad comparador. Comparador de 8 bits.
- **contador.vhd**: Entidad contador. Contador de 11 bits.
- **prescaler.vhd**: Entidad prescaler. Divisor de frecuencia entre 4
- **pwm_unid.vhd**. Entidad pwm_unit. Unidad de generacion de pwm
- **perif_access.vhd**. Entidad perif_access. Entidad superior que se sintetiza y se carga en la FPGA.

También se han implementado sus correspondientes bancos de pruebas para comprobar mediante simulación que el funcionamiento es correcto:

- **tb_contador.vhd**: Banco de pruebas para el contador
- **tb_comparador.vhd**: Banco de pruebas para el comparador
- **tb_pwm_unit.vhd**: Banco de pruebas para la unidad PWM

6.1.2. Version 1.0: Aplicación práctica

La versión 1.0 permite controlar 4 servos y es la desarrollada para la aplicación del movimiento de las minicámaras. El interfaz se ha optimizado de modo que sólo se utiliza un registro de 32 bits mapeado en la dirección 0x16000000.

Estas dos versiones sólo se diferencian en el interfaz y en la cantidad de unidades PWM que contienen, por lo que sólo es diferente la entidad superior (perif_acces.vhd), siendo iguales el resto. En la figura 12 se presenta un esquema de la entidad perif_acces.vhd. La señales de interfaz son las mismas, por lo que no se han dibujado.

6.2. Version 0.6: Control de un servo

En este apartado se muestran los programas en vhdl de las diferentes entidades que componen la versión 0.6

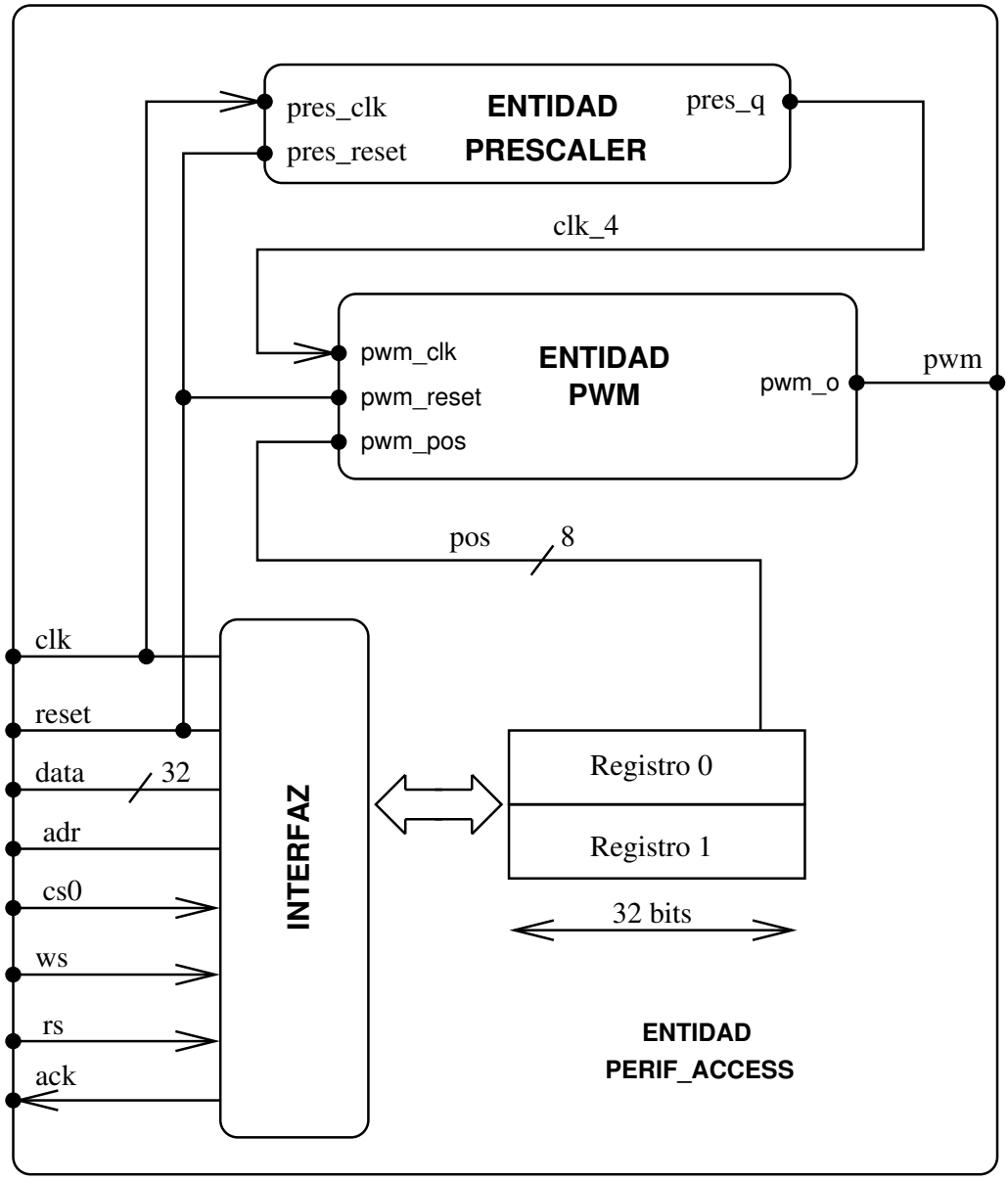


Figura 11: Entidad del nivel superior (top-level): perif_access para la version 0.6

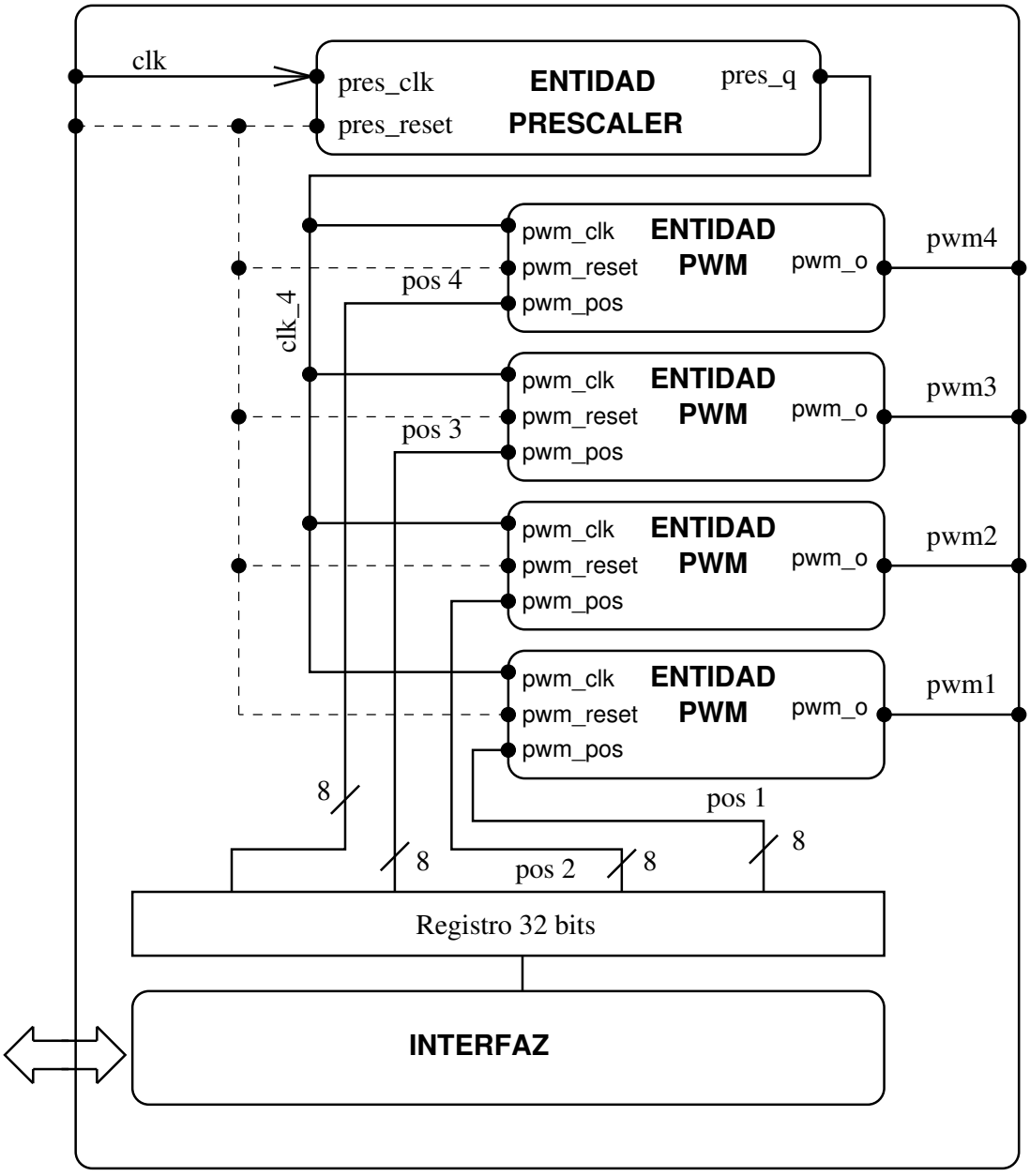


Figura 12: Entidad superior de la version 1.0

6.2.1. Contador

```
------------------------------------------------------------
-- contador.vhd.  Juan Gonzalez. Feb-2002                  --
-- Licencia GPL.                                           --
------------------------------------------------------------
-- PROYECTO LABOBOT                                        --
------------------------------------------------------------
-- Contador de 11 bits                                     --
--                                                         --
-- Entradas: clk   : Reloj                                 --
--           reset : Puesta a cero asíncrona (Nivel Alto)  --
-- Salidas:                                                --
--      -q  : Datos de salida                              --
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity contador is
  port (clk   : in std_logic; -- Reloj
        reset : in std_logic;
        q     : out std_logic_vector (10 downto 0)); --Salida
end contador;
architecture beh of contador is
begin
  output: process(clk,reset)
  variable cuenta : std_logic_vector(10 downto 0);
  begin
    -- Actualizar la cuenta
    if (reset='1') then        -- Reset asíncrono
      cuenta:=(others=>'0');   -- Inicializar contador
      q<=cuenta;
    elsif (clk'event and clk='1') then -- Flanco de subida en reloj
      cuenta:=(cuenta+1);  -- Incrementar contador
      q<=cuenta;
    end if;
  end process;
end beh;
```

6.2.2. Comparador

```
------------------------------------------------------
-- comparador.vhdl.  Juan Gonzalez. Feb-2002        --
-- Licencia GPL.                                    --
------------------------------------------------------
-- PROYECTO LABOBOT                                 --
------------------------------------------------------
-- Comparador de 8 bits                             --
-- Sólo se ha implementado la salida correspondiente--
-- a G (La función mayor que).                      --
--                                                  --
```

```
-- Entradas: opa : Operando A                          --
--           opb : Operando B                          --
-- Salidas:                                            --
--           G  : (A>B)                                --
------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity comparador is
  port (opa : in std_logic_vector(7 downto 0);   -- Operador A
        opb : in std_logic_vector(7 downto 0);   -- Operador B
        G   : out std_logic);                    -- (A>=G)
end comparador;
architecture beh of comparador is
begin
  process (opa, opb)
  begin
    if (opa<opb) then
      G <= '0';
    else
      G <= '1';
    end if;
  end process;
end beh;
```

6.2.3. Prescaler

```
-------------------------------------------------------------
-- prescaler.vhd.  Juan Gonzalez. Feb-2002                --
-- Licencia GPL.                                          --
-------------------------------------------------------------
-- PROYECTO LABOBOT                                       --
-------------------------------------------------------------------
-- Prescaler de 2 bits (Divide por 4)                          --
--                                                             --
-- Entradas: pres_clk   : Reloj                                --
--           pres_reset : Puesta a cero asíncrona (Nivel Alto) --
-- Salidas:                                                    --
--           pres_q : Salida                                   --
-------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity prescaler is
  port (pres_clk  : in std_logic;
        pres_reset: in std_logic;
        pres_q    : out std_logic);
end prescaler;
architecture beh of prescaler is
begin
  output: process(pres_clk,pres_reset)
  variable cuenta : std_logic_vector(1 downto 0);
```

```
  begin
    -- Actualizar la cuenta
    if (pres_reset='1') then
      cuenta:=(others=>'0');
      pres_q<='0';
    elsif (pres_clk'event and pres_clk='1') then
      cuenta:=(cuenta+1);
      pres_q<=cuenta(1);
    end if;
  end process;
end beh;
```

6.2.4. Unidad de PWM

```
-------------------------------------------------------
-- pwm_unit.vhdl  Juan Gonzalez. Febrero-2002      --
-- Licencia GPL.                                   --
-------------------------------------------------------
-- PROYECTO LABOBOT                                --
-------------------------------------------------------
-- UNIDAD DE PWM:                                  --
--                                                 --
-- ENTRADAS:                                       --
--      -pwm_clk  : Señal de reloj                 --
--      -pwm_pos  : Posicion del servo (8 bits)    --
--      -pwm_reset: Reset                          --
-- SALIDAS:                                        --
--      -pwm_o: Señal PWM                          --
-------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity pwm_unit is
  port (pwm_pos : in std_logic_vector (7 downto 0);  -- Posicion
        pwm_clk : in std_logic;                      -- Reloj
        pwm_reset: in std_logic;                     -- Reset.
        pwm_o    : out std_logic);                   -- Señal PWM
end pwm_unit;
architecture beh of pwm_unit is
  component contador is
    port (clk     : in std_logic; -- Reloj
          reset   : in std_logic;
          q       : out std_logic_vector (10 downto 0)); --Salida
  end component;

  component comparador is
    port (opa : in std_logic_vector(7 downto 0);   -- Operador A
          opb : in std_logic_vector(7 downto 0);   -- Operador B
          G   : out std_logic);
  end component;
  signal ntic   : std_logic_vector (10 downto 0); -- Numero de tics de reloj
  signal ntic_ms: std_logic_vector (7 downto 0);  -- 8 bits menos peso de ntic
```

```
  signal sg     : std_logic; -- Señal G
begin
  CONT1: contador   port map (clk=>pwm_clk, q=>ntic, reset=>pwm_reset);
  COMP1: comparador port map (opa=>ntic_ms, opb=>pwm_pos, G=>sg);

  -- Obtener los 8 bits menos significativos del contador
  ntic_ms<=ntic(7 downto 0);

  -- Señal PWM de salida
  pwm_o<= (not ntic(8)) and (not ntic(9)) and (not ntic(10))
           and (not sg);
 end beh;
```

6.2.5. Top-level

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity perif_access is
     port (
       data : inout std_logic_vector(31 downto 0);
       adr : in std_logic;
       cs0 : in std_logic;
       reset : in std_logic;
       ws : in std_logic;
       rs : in std_logic;
       clk : in std_logic;
       ack : out std_logic;
       pwm : out std_logic;
       exit_clk : out std_logic
      );
end perif_access;
architecture Final of perif_access is
        type mem is array (natural range <>) of std_logic_vector(31 downto 0);
        type state is (IDLE, RDOK, WROK);
        signal memoria_rw : mem(0 to 1);
        signal estado : state;
        signal n_estado : state;
        signal load : std_logic;
       ----- ***** -----
       --signal operando1 : std_logic_vector(15 downto 0);
       --SIGNAL operando2 : std_logic_vector(15 downto 0);
       --SIGNAL resultado : std_logic_vector(31 downto 0);
       signal pos         : std_logic_vector(7 downto 0);
       signal clk_4       : std_logic;
       ----- ***** -----
       component pwm_unit is
          port (pwm_pos  : in std_logic_vector (7 downto 0);  -- Posicion
                pwm_clk  : in std_logic;                      -- Reloj
                pwm_reset: in std_logic;                      -- Reset.
```

```
                pwm_o    : out std_logic);                    -- Señal PWM
      end component;
      component prescaler is
        port (pres_clk   : in std_logic;
              pres_reset: in std_logic;
              pres_q    : out std_logic);
      end component;
begin
  exit_clk <= clk;
  process(clk, reset, load, memoria_rw) is
    variable i : integer;
  begin
    if(reset = '1') then
      estado <= IDLE;
      for i in 0 to 1 loop
        memoria_rw(i) <= (others => '0');
      end loop;
    elsif (clk = '1' and clk'event) then
      estado <= n_estado;
        if(load = '1') then
          if(adr = '1') then
            memoria_rw(1) <= data;
          else
            memoria_rw(0) <= data;
          end if;
        end if;
    end if;
  --operando1 <= memoria_rw(0)(31 downto 16);
  --operando2 <= memoria_rw(0)(15 downto 0);
  pos<=memoria_rw(0)(7 downto 0);
  --memoria_rw(1) <= resultado;
end process;
process(estado, cs0, ws, rs)
begin
  case estado is
    when IDLE =>
      ack <= '1';
      load <= '0';
      if(cs0 = '0' and ws = '0' and rs = '1') then
        n_estado <= WROK;
        load <= '1';
      elsif(cs0 = '0' and rs = '0' and ws = '1') then
        n_estado <= RDOK;
      else
        n_estado <= IDLE;
      end if;
    when WROK=>
      ack<='0';
```

```
      load <= '0';
      if(cs0 = '1') then
        n_estado <= IDLE;
      else
        n_estado <= WROK;
      end if;
    when RDOK =>
      ack <= '0';
      load <= '0';
      if(cs0 = '1') then
        n_estado <= IDLE;
      else
        n_estado <= RDOK;
      end if;
  end case;
end process;
process(estado, memoria_rw, adr)
begin
  if(estado = RDOK) then
    if(adr = '1') then
      data <= memoria_rw(1);
    else
      data <= memoria_rw(0);
    end if;
  else data <= (others => 'Z');
  end if;
end process;
PRES1: prescaler port map (pres_clk=>clk,
                           pres_reset=>reset,
                           pres_q=>clk_4);
PWM1: pwm_unit port map (pwm_pos => pos,
                         pwm_reset=>reset,
                         pwm_clk=>clk_4,
                         pwm_o=>pwm);
end Final;
```

6.2.6. Fichero de restricciones

```
##################################################
# PROYECTO LABOBOT. Juan Gonzalez. Febrero 2002  -
#-------------------------------------------------
# Fichero de restricciones                       -
```

```
#-------------------------------------------------
# V0.6                                           #
##################################################
#---- Entidad top_pwm:
NET clk   LOC=P124;   # -- Señal de reloj
NET reset LOC=P63;    # -- Reset
NET pwm   LOC=P127;   # -- PWM: Salida: Pin A0, conector J27
#---- Interfaz con CPU
NET ack LOC=P239;
NET cs0 LOC=P142;
NET ws LOC=P183;
NET rs LOC=P153;
NET adr LOC=P188;
NET data<0> LOC=P206;
NET data<1> LOC=P207;
NET data<2> LOC=P208;
NET data<3> LOC=P209;
NET data<4> LOC=P210;
NET data<5> LOC=P213;
NET data<6> LOC=P214;
NET data<7> LOC=P215;
NET data<8> LOC=P216;
NET data<9> LOC=P217;
NET data<10> LOC=P218;
NET data<11> LOC=P220;
NET data<12> LOC=P221;
NET data<13> LOC=P223;
NET data<14> LOC=P224;
NET data<15> LOC=P225;
NET data<16> LOC=P226;
NET data<17> LOC=P228;
NET data<18> LOC=P229;
NET data<19> LOC=P230;
NET data<20> LOC=P231;
NET data<21> LOC=P232;
NET data<22> LOC=P233;
NET data<23> LOC=P234;
NET data<24> LOC=P177;
NET data<25> LOC=P173;
NET data<26> LOC=P159;
NET data<27> LOC=P152;
NET data<28> LOC=P148;
NET data<29> LOC=P141;
```

```
NET data<30> LOC=P129;
NET data<31> LOC=P123;
```

6.3. Version 1.0: Control de 4 servos

En esta versión se mueven 4 servos, en vez de 1. Para el envío de las posiciones de los servos se utiliza un único registro de 32 bits, que se divide en 4 partes de 8 bits. Cada parte contiene la posicion de uno de los servos.

Obsérvese que se ha suprimido la línea de direcciones (adr). Cómo sólo se ha mapeado un registro, basta con escribir en cualquier dirección en la que se active el chip select 0 (cs0): 16000000-1600FFFF.

6.3.1. Top level

```
----------------------------------------------------------------
-- perif_access.vhd.  Juan Gonzalez. Feb-2002                  --
-- Licencia GPL.                                               --
----------------------------------------------------------------
-- PROYECTO LABOBOT                                            --
----------------------------------------------------------------
-- TOP-level. Se implementan 4 unidades de PWM y la mßqui-     --
-- na de estados para la transferencia de informacion          --
-- entre la CPU y la FPGA.                                     --
--                                                             --
----------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity perif_access is
  port (
    -- Señales usadas para el interfaz entre FPGA y CPU
    data  : inout std_logic_vector(31 downto 0);
    adr   : in std_logic;
    cs0   : in std_logic;
    ws    : in std_logic;
    rs    : in std_logic;
    ack   : out std_logic;

    -- Señales empleadas para las unidades PWM
    reset : in std_logic;
```

```
    clk   : in std_logic;
    pwm1  : out std_logic;
    pwm2  : out std_logic;
    pwm3  : out std_logic;
    pwm4  : out std_logic);
end perif_access;

architecture Final of perif_access is
  type state is (IDLE, RDOK, WROK);
  signal estado     : state;
  signal n_estado   : state;
  signal load       : std_logic;
  -- Registros de posicion de los servos (Interfaz con CPU)
  signal reg_pos    : std_logic_vector(31 downto 0);

  ---------------------------------------------
  -- Señales especificas de las unidades PWM --
  ---------------------------------------------
  -- Posiciones de los servos
  signal pos1  : std_logic_vector(7 downto 0);
  signal pos2  : std_logic_vector(7 downto 0);
  signal pos3  : std_logic_vector(7 downto 0);
  signal pos4  : std_logic_vector(7 downto 0);
  -- Reloj para el PWM. Funciona a una frecuencia 4 veces
  -- inferior que la señal de reloj global (clk)
  signal clk_4 : std_logic;

  ----------------------------------
  -- Declaracion de componentes   --
  ----------------------------------

  -- Unidad de PWM
  component pwm_unit is
    port (pwm_pos  : in std_logic_vector (7 downto 0);  -- Posicion
          pwm_clk  : in std_logic;                      -- Reloj
          pwm_reset: in std_logic;                      -- Reset.
          pwm_o    : out std_logic);                    -- Se±al PWM
    end component;

  -- Divisor de la frecuencia del reloj
  component prescaler is
    port (pres_clk  : in std_logic;
          pres_reset: in std_logic;
```

```
          pres_q    : out std_logic);
  end component;


begin
  ------------------------------------
  -- MAQUINA DE ESTADOS DE INTERFAZ  --
  ------------------------------------

  process(clk, reset, load, reg_pos) is
    variable i : integer;
  begin
    if(reset = '1') then  -- Si hay reset...
      estado <= IDLE;            -- Estado inicial
      reg_pos<=(others => '0');  -- Inicializar registro de posicion
    elsif (clk = '1' and clk'event) then  -- Flanco de subida
      estado <= n_estado;
      if(load = '1') then
        reg_pos <= data;   -- Lectura de la posicion de los servos
      end if;
    end if;

    -- Cada grupo de 8 bits del registro de posicion se envia a
    -- una unidad pwm diferente
    pos1<=reg_pos(7 downto 0);
    pos2<=reg_pos(15 downto 8);
    pos3<=reg_pos(23 downto 16);
    pos4<=reg_pos(31 downto 24);
  end process;

  process(estado, cs0, ws, rs)
  begin
    case estado is
      when IDLE =>
        ack <= '1';
        load <= '0';
        if(cs0 = '0' and ws = '0' and rs = '1') then -- Ciclo de escr
          n_estado <= WROK;
          load <= '1';
        elsif(cs0 = '0' and rs = '0' and ws = '1') then -- Ciclo de l
          n_estado <= RDOK;
        else
          n_estado <= IDLE;
```

```
        end if;
      when WROK =>
        ack <= '0';
        load <= '0';

        if(cs0 = '1') then
          n_estado <= IDLE;
        else
          n_estado <= WROK;
        end if;
      when RDOK =>
        ack <= '0';
        load <= '0';
        if(cs0 = '1') then
          n_estado <= IDLE;
        else
          n_estado <= RDOK;
        end if;
    end case;
  end process;

  process(estado, reg_pos)
  begin
    if(estado = RDOK) then
         data <= reg_pos;
    else data <= (others => 'Z');
    end if;
  end process;

--------------------
--  PRESCALER    --
--------------------
  PRES1: prescaler port map (pres_clk=>clk,
                             pres_reset=>reset,
                             pres_q=>clk_4);
---------------------
-- UNIDADES DE PWM --
---------------------
  PWM_U1: pwm_unit port map (pwm_pos => pos1,
                             pwm_reset=>reset,
                             pwm_clk=>clk_4,
                             pwm_o=>pwm1);
```

```
  PWM_U2: pwm_unit port map (pwm_pos => pos2,
                             pwm_reset=>reset,
                             pwm_clk=>clk_4,
                             pwm_o=>pwm2);

  PWM_U3: pwm_unit port map (pwm_pos => pos3,
                             pwm_reset=>reset,
                             pwm_clk=>clk_4,
                             pwm_o=>pwm3);

  PWM_U4: pwm_unit port map (pwm_pos => pos4,
                             pwm_reset=>reset,
                             pwm_clk=>clk_4,
                             pwm_o=>pwm4);

end Final;
```

6.3.2. Fichero de restricciones

```
##################################################
# PROYECTO LABOBOT. Juan Gonzalez. Febrero 2002  -
#------------------------------------------------
# Fichero de restricciones                      -
#------------------------------------------------
# V0.8                                          #
##################################################

#---- Senales del PWM:
NET clk   LOC=P124;   # -- Señal de reloj
NET reset LOC=P63;    # -- Reset
NET pwm1  LOC=P127;   # -- PWM1: Salida: Pin A0, conector J27
NET pwm2  LOC=P126;   # -- PWM2: Salida: Pin A1, conector J27
NET pwm3  LOC=P125;   # -- PWM3: Salida: Pin A2, conector J27
NET pwm4  LOC=P117;   # -- PWM4: Salida: Pin A3, conector J27

#---- Interfaz con CPU
NET ack LOC=P239;
NET cs0 LOC=P142;
NET ws LOC=P183;
NET rs LOC=P153;
NET adr LOC=P188;
NET data<0> LOC=P206;
NET data<1> LOC=P207;
NET data<2> LOC=P208;
NET data<3> LOC=P209;
NET data<4> LOC=P210;
NET data<5> LOC=P213;
```

```
NET data<6> LOC=P214;
NET data<7> LOC=P215;
NET data<8> LOC=P216;
NET data<9> LOC=P217;
NET data<10> LOC=P218;
NET data<11> LOC=P220;
NET data<12> LOC=P221;
NET data<13> LOC=P223;
NET data<14> LOC=P224;
NET data<15> LOC=P225;
NET data<16> LOC=P226;
NET data<17> LOC=P228;
NET data<18> LOC=P229;
NET data<19> LOC=P230;
NET data<20> LOC=P231;
NET data<21> LOC=P232;
NET data<22> LOC=P233;
NET data<23> LOC=P234;
NET data<24> LOC=P177;
NET data<25> LOC=P173;
NET data<26> LOC=P159;
NET data<27> LOC=P152;
NET data<28> LOC=P148;
NET data<29> LOC=P141;
NET data<30> LOC=P129;
NET data<31> LOC=P123;
```

7. Diseño Software

7.1. Version 0.6: Pruebas de acceso

Este programa se ha realizado para comprobar que la unidad pwm está funcionando. Simplemente se le pide al usuario una posición y se envía al registro 0, que se encuentra mapeado en la dirección 0x16000000.

```
/*****************************************************/
/* PROYECTO LABOBOT.  Juan Gonzalez.  Febrero 2002   */
/* LICENCIA GPL.                                     */
/*****************************************************/

#include <stdio.h>
#include "work.h"

#include <stdio.h>
#include <errno.h>
#include <string.h>
```

```
#include <unistd.h>
#include <bsp.h>
#include <m68360.h>
#include <rtems/error.h>
#include <rtems/rtems_bsdnet.h>
#include <rtems/tftp.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sockio.h>
#include <net/if.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <nc360/md.h>
#include <nc360/cpld.h>
#include <nc360/mubus.h>
#include <nc360/clock.h>
#include <nc360/flash.h>
#include <nc360/interrupts.h>
#include <nc360/identification.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/timeb.h>
#include <time.h>
#include <netdb.h>
#include <math.h>
#include <stdio.h>
#include <sys/time.h>

#include "labomat3.h"
#include "rlib.h"
#include "usuals.h"


#define FPGA 0x16000000
```

```
void main_work(void)
{
  Card32 *punt;
  Card32 dato;
  char buffer[100];

  /* Puntero de acceso a la FPGA */
  punt=(Card32 *)FPGA;

  while (1) {
    printf ("Posicion (hex): ");
    read_line(buffer,sizeof(buffer));
    sscanf(buffer,"%x",&dato);

    (*punt)=dato;
    printf ("Valor:%x (%x)\n",dato,*punt);
  }


  return;
}
```

7.2. Version 1.0: Librerías de control

Para el control de los servos se han realizado unas funciones que permiten al usuario posicionar los servos utilizando grados sexagesimales, en vez de tics de reloj, y además se oculta la forma de acceso al servo (el usuario no tiene que saber nada sobre direcciones de memoria, ni registros mapeados).

Las dos funciones de interfaz son:

- *void servo_posicionar_all(int pos1, int pos2, int pos3, int pos4)*

  Posicionar todos los servos. Los valores de las posiciones están comprendidas en el rango [-90,90], correspondientes a los dos extremos opuestos, separados 180^o

- *void servo_posicionar(int ns, int pos)*

  Posocionar el servo indicado, sin modificar la posición del resto de servos. El parámetro "ns" indica el número de servo, comprendido entre [1-4]

A nivel interno, estas funciones realizan las conversión entre grados sexagesimales y tics de reloj. La fórmula de conversión empleada es la siguiente:

$$pos_{tic} = \frac{(TIC_{MAX} - TIC_{MIN}) * (pos_{grad} + 90)}{180} + TIC_{MIN}$$

de forma que cuando:

- $pos_{grad} = 0 \Rightarrow pos_{tic} = \frac{TIC_{MAX}+TIC_{MIN}}{2}$, el servo se sitúa en la posición central.
- $pos_{grad=-90} \Rightarrow pos_{tic} = TIC_{MIN}$, el servo se sitúa en un extremo.
- $pos_{grad} = 90 \Rightarrow pos_{tic} = TIC_{MAX}$, el servo se sitúa en el otro extremo.

Las constantes TIC_{MAX} y TIC_{MIN} se definen en el programa y determinan las posiciones extremas del servos, en tics de reloj.

Estas librerías de pruebas se encuentran en el mismo programa de pruebas (labobot.c) en el que se ha realizado la aplicación de las minicámaras. El listado completo se puede encontrar en el apartado 8.4.

8. Aplicación práctica: Movimiento de dos minicámaras

8.1. Introducción

Una vez desarrollado el hardware y el software para el control y posicionamiento de los servos, se va a aplicar a un sistema real, compuesto por dos minicámaras con salida de vídeo, colocadas en sendos soportes que disponen de dos servos cada uno, de manera que las minicámaras se pueden orientar hacia cualquier dirección, dentro del campo de movimiento de los servos.

La aplicación pretende demostrar que tanto el software como el hardware funcionan correctamente y que este trabajo no ha quedado sólo en algo teórico. No obstante la aplicación no hace "nada útil". Simplemente permite que las dos minicámaras sigan dos secuencias programadas, una de 4 posiciones y otra de 3.

8.2. Tarjeta PLM-3003

Para la conexión de los servos a la tarjeta LABOMAT se ha construido una placa de interconexión, denominada **PLM-3003** (**PLM** son las siglas de **P**eriférico para **L**abo**M**at), que se muestra en la figura 13 . Permite conectar hasta 4 servos al conector J27 de la LABOMAT. Dispone de una clema para la alimentación de los servos (5v), de un conector acodado de tipo bus de 10x2 pines para conexión a la LABOMAT y de 4 conectores macho para la conexión de los servos. Las señales de control de los servos se conectan a los siguientes pines del conector J27:

Servo	Pin conector J27	Pin FPGA 4013	Señal
1	A0	127	IO32
2	A1	126	IO33
3	A2	125	IO34
4	A3	117	IO35

En la versión 0.6 de Labobot sólo se utiliza el servo 1, puesto que sólo hay una unidad de PWM en la FPGA.

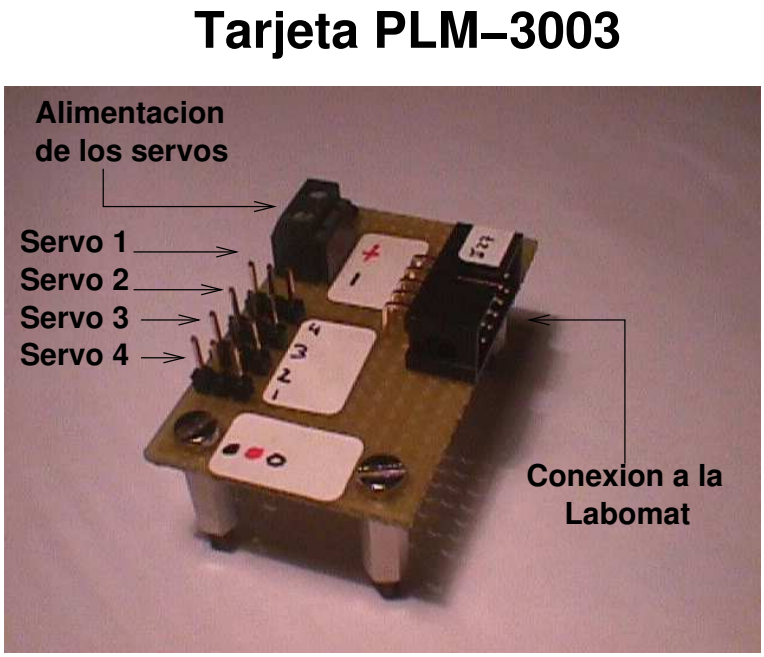


Figura 13: Tarjeta PLM-3003 para la conexión de los servos a la Labomat

8.3. Estructura mecánica

La estructura mecánica se muestra en la figura 14. Está constituida por una base de PVC blanco de 1cm de espesor, a la que se han colocado los servos 1 y 3 utilizando cuatro varillas roscadas de 3mm de diámetro y 6cm de largo. A estos dos servos llevan atornillados a sus ejes una pieza en forma de "L" de alumínio de 1mm, en donde se encuentran los servos 2 y 4 respectivamente. A estos dos servos a su vez van conectadas las minicámaras 1 y 2.

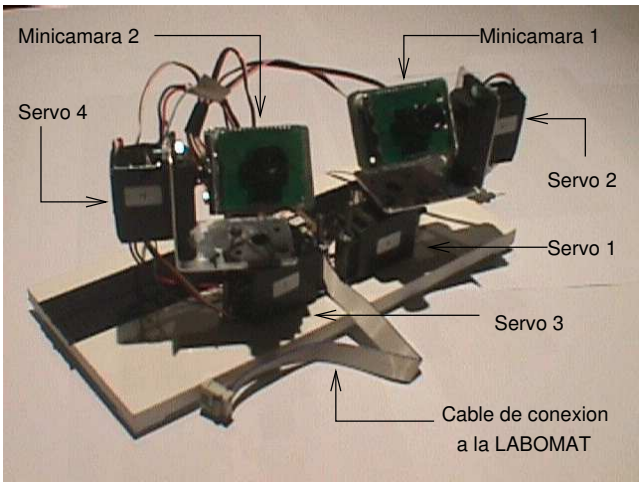
8.4. SW de pruebas

El software que permite reproducir las secuencias de movimiento, así como las librerías de control se muestran a continuación (Fichero labobot.c):

```
/*******************************************************/
/* PROYECTO LABOBOT.  Juan Gonzalez.  Febrero 2002     */
/* LICENCIA GPL.                                       */
/*******************************************************/

#include <stdio.h>
#include "work.h"
```

Vista delantera



Vista trasera

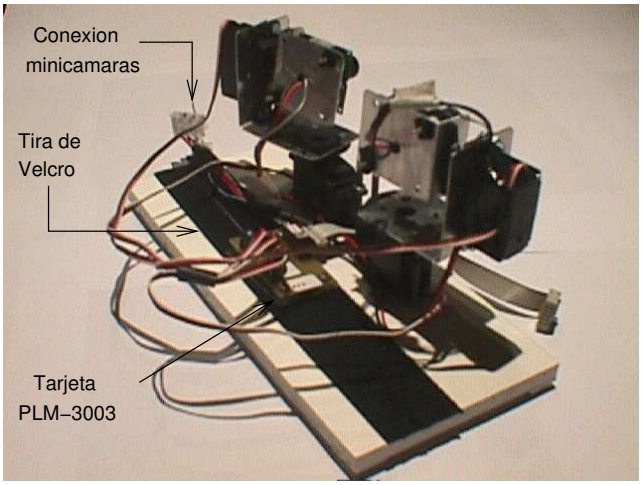


Figura 14: Estructura mecánica: servos y minicámaras

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <bsp.h>
#include <m68360.h>
#include <rtems/error.h>
#include <rtems/rtems_bsdnet.h>
#include <rtems/tftp.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sockio.h>
#include <net/if.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <nc360/md.h>
#include <nc360/cpld.h>
#include <nc360/mubus.h>
#include <nc360/clock.h>
#include <nc360/flash.h>
#include <nc360/interrupts.h>
#include <nc360/identification.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/timeb.h>
#include <time.h>
#include <netdb.h>
#include <math.h>
#include <stdio.h>
#include <sys/time.h>

#include "labomat3.h"
#include "rlib.h"
#include "usuals.h"
```

```
/*****************************************************************
/* MODULO SERVO. Rutinas de posicionamiento de 4 servos conectados  *
/* a la tarjeta PLM-3003 enchufada al conector J27 de la LABOMAT    *
/*****************************************************************

/*------------------*/
/* DEFINICIONES     */
/*------------------*/

/* Direccion de acceso a la FPGA, donde esta mapeado  */
/* el registro de posicion de los servos, de 32 bits  */
#define FPGA 0x16000000

/* Anchura minima, media y maxima de la señal PWM, que se corresponde
/* las posiciones -90, 0 y 90 grados del servo
#define TIC_MIN    0x20
#define TIC_MAX    0xEC

/*---------------------------------*/
/* VARIABLES INTERNAS DEL MODULO  */
/*---------------------------------*/

/* Copia del registro de posicion */
static Card32 reg_pos_copy=0;

/* Puntero al registro de posicion en la FPGA */
static volatile Card32 *reg_pos=(Card32 *)FPGA;

/*-----------------------------*/
/* FUNCIONES DE INTERFAZ      */
/*-----------------------------*/

void servo_posicionar_all(int pos1, int pos2, int pos3, int pos4)
/*****************************************************************
/* Posicionar los 4 servos. Las posiciones de los servos vienen     *
/* en grados sexagesimales [-90,90]                                 *
/*****************************************************************
{
  /* Posiciones en tics de reloj */
  Card8 pos1_tic;
  Card8 pos2_tic;
  Card8 pos3_tic;
  Card8 pos4_tic;
```

```
  /* Realizar la conversion de grados a tics */
  pos1_tic=((TIC_MAX - TIC_MIN)*(pos1+90))/180 + TIC_MIN;
  pos2_tic=((TIC_MAX - TIC_MIN)*(pos2+90))/180 + TIC_MIN;
  pos3_tic=((TIC_MAX - TIC_MIN)*(pos3+90))/180 + TIC_MIN;
  pos4_tic=((TIC_MAX - TIC_MIN)*(pos4+90))/180 + TIC_MIN;

  /* Actualizar copia del registro de posicion */
  reg_pos_copy = (pos4_tic << 24) | (pos3_tic << 16) | (pos2_tic <<

  /* Enviar posiciones al registro de posicion "verdadero" */
  *reg_pos=reg_pos_copy;
}

void servo_posicionar(int ns, int pos)
/***********************************************/
/* Posicionar solo un servo y dejar los demas  */
/* en la misma posicion en la que estaban      */
/* ENTRADAS:                                   */
/*   ns : Numero del servo a posicionar [1-4]  */
/*   pos: Posicion en grados [-90,90]          */
/***********************************************/
{
  Card8 pos_tic;

  /* Numero de servo incorrecto. No se posiciona */
  if (ns<1 || ns>4) return;
  ns--; /* Internamente se usa la numeracion 0-3 para los servos */

  /* Obtener la nueva posicion */
  pos_tic=((TIC_MAX - TIC_MIN)*(pos+90))/180 + TIC_MIN;
  printf ("Posicion%d: \n",pos);
  printf ("Posicion en tics:%x\n",pos_tic);

  /* Actualizar la copia del registro de posicion */
  switch(ns) {
    case 0: reg_pos_copy=(reg_pos_copy & ~0x000000FF);
            reg_pos_copy=(reg_pos_copy | pos_tic);
            break;
    case 1: reg_pos_copy=(reg_pos_copy & ~0x0000FF00);
            reg_pos_copy=(reg_pos_copy | (pos_tic<<8));
            break;
    case 2: reg_pos_copy=(reg_pos_copy & ~0x00FF0000);
```

```
            reg_pos_copy=(reg_pos_copy | (pos_tic<<16));
            break;
    case 3: reg_pos_copy=(reg_pos_copy & ~0xFF000000);
            reg_pos_copy=(reg_pos_copy | (pos_tic<<24));
            break;
  }

  /* Enviar posiciones al registro de posicion "verdadero" */
  *reg_pos=reg_pos_copy;
}

/*-----------------------------*/
/* FUNCIONES PRIVADAS          */
/*-----------------------------*/



/*********************************************/
/* PROGRAMA DE PRUEBA PARA EL MODULO SERVO    */
/*********************************************/
void play_sec1()
/*******************************************/
/* Reproducir una secuencia de movimiento  */
/*******************************************/
{
  int nv=5;           /* Numero de vectores de posicion       */
  int sec1[][4]= {    /* Vectores de posicion de la secuencia */
    {-45,45,-45,-45},
    {-45,-45,-45,45},
    {45,-45,45,45},
    {45,45,45,-45},
    {0,0,0,0}
  };
  int i;

  for (i=0; i<nv; i++) {
    /* Posicionar */
    servo_posicionar_all(sec1[i][0],sec1[i][1],sec1[i][2],sec1[i][3])

    /* Realizar una pausa */
    sleep(1);

  }
```

```
}

void play_sec2()
/*****************************************/
/* Reproducir una secuencia de movimiento  */
/*****************************************/
{
  int nv=3;           /* Numero de vectores de posicion       */
  int sec[][4]= {    /* Vectores de posicion de la secuencia */
    {-90,-90,-90,-90},
    {90,90,90,90},
    {0,0,0,0}
  };
  int i;

  for (i=0; i<nv; i++) {
    /* Posicionar */
    servo_posicionar_all(sec[i][0],sec[i][1],sec[i][2],sec[i][3]);

    /* Realizar una pausa */
    sleep(1);

  }

}

void prueba1()
{
  char   buffer[100];
  int    posicion;

  while (1) {
    printf ("Servo1[-90,90]: ");
    read_line(buffer,sizeof(buffer));
    sscanf(buffer,"%d",&posicion);
    servo_posicionar_all(posicion,posicion,posicion,posicion);

    printf ("  Copia Reg. Posicion:%x \n",reg_pos_copy);
  }
}

void main_work(void)
```

```
{
  char   buffer[100];
  int    opcion;

  /* Inicializar las posiciones de los servos */
  servo_posicionar_all(0,0,0,0);

  while (1) {
    printf ("introduzca secuencia (1-2):\n");
    read_line(buffer,sizeof(buffer));
    sscanf(buffer,"%d",&opcion);
    switch(opcion) {
      case 1 : play_sec1();
               break;
      case 2 : play_sec2();
    }
  }

  return;
}
```

8.5. Probando la aplicación

Esta prueba está pensada para **realizarse localmente**. Se puede probar remotamente, sin embargo no tiene sentido porque no se puede comprobar que el sistema está funcionando. El programa de pruebas l**abobot.c** realiza las impresiones en modo local. No obstante, sólo hay que cambiar las instrucciones printf por rprintf para poder ejecutarlo remotamente. Los pasos a seguir son los siguientes:

1. **Configurar la FPGA XC4013**: Enviar el programa labobot.hex (V1.0)
2. **Configurar el reloj de la FPGA**. Esto se puede realizar tanto local como remotamente.
   - *a*) **Establecer frecuencia PCG** (opcion FRECUENCIA PCG):
     1) Usar una frecuencia de 409600 Hz
     2) Seleccionar PCG1
   - *b*) **Seleccionar el reloj**:
     1) Marcar PCG1
     2) Seleccionar dispositivo XC4013
3. **Realizar la conexión de las minicámaras** al conector J27 de la LABOMAT. Si se realiza la conexión antes de haber configura el reloj, a los servos les entrará una señal PWM de

muy alta frecuencia, lo que provoca que emitan un "pitido" muy molesto, además de poder ser perjudicial para los servos.

No olvidar alimentar la placa PLM-3003 con 5v (a través de una fuente de alimentación capaz de dar al menos 1.2A).

4. **Compilar el programa labobot.c (V1.0)**

5. **Ejecutarlo en modo local**

   Este programa nos permite seleccionar entre dos posibles secuencias. Cada vez que pulsemos la tecla 1 ó 2, seguidas de Return, las minicámaras se posicionarán en las diferentes posiciones indicadas en el software y se volverá al menú principal. A partir de este ejemplo se pueden realizar aplicaciones mucho más complejas de control.

9. Resultados

Una de los objetivos de este proyecto era el determinar cuántos servos se podrían llegar a controlar. Para probarlo se ha desarrollado una nueva versión, la 1.2, en la que se han ido añadiendo unidades de PWM hasta que se han agotado los recursos de la FPGA.

El número máximo de servos que se pueden llegar a controlar, es de 28, necesitándose 7 registros de 32 bits para almacenar las posiciones de todos ellos, enviadas por la CPU.

Como conclusiones podemos sacar las siguientes:

1. **El número de servos que se pueden controlar con esta solución es muchísimo mayor que para el caso puramente software**.

   Por poner un ejemplo con números, sin con un sistema puramente software, basado en el microcontrolador 68hc11 de Motorola se podían controlar 4 servos, serán necesarios 7 microcontroladores conectados a uno maestro para poder controlar los 28 servos).

2. **La solución SW/HW es mucho más sencilla que la puramente SW**

   Como se desprende del ejemplo anterior, en el que es necesaria una red de microcontroladores para poder posicionar los servos de cualquier robot articulado con más de 4 articulaciones.

3. **Mayor versatilidad para su uso en robótica**.

   Incluso para el caso de controlar robots articulados con 4 ó menos servos, la solución HW/SW permite una mayor versatilidad, puesto que parte de los recursos de la FPGA se pueden emplear para implementar controladores necesarios para el robot: infrarrojos, ultrasonidos, sensores de contacto... En la solución SW se están desperdiciando recursos al tener que emplear un microcontrolador dedicado exclusivamente al control de los servos.

4. **Mejora en el propio control de los servos**.

   Cuando se emplea una solución software, es el microcontrolador el que genera mendiante interrupciones las señales PWM. Cuando se están controlando 4 servos y las posiciones de los 4 servos son similares, el ancho de la señal de PWM es casi igual, produciéndose las 4 interrupciones simultaneamente. Puesto que se atienden secuencialmente, por orden de llegada o en su caso por una prioridad preestablecida, ocurren pequeños retrasos que hacen que el servo "vibre". Sólo es apreciable en el caso particular antes mencionado: cuando dos o más servos están en posiciones muy cercanas.

   Con la opción SW/HW esto no ocurre, consiguiéndose un movimiento fluido y sin vibraciones en ningún momento.

Conclusión final de esta trabajo:

No sólo es viable el utilizar una solución de codiseño HW/SW para el control de robots articulados sino que proporciona una serie de ventajas que hacen que la solución puramente software quede en un segundo plano. Esta vía de trabajo en robótica, que el autor del proyecto nunca había explorado, abre nuevas posibilidades hacia diseños de robots más compactos, simples, versátiles y que consuman menos.

---

10. Mejoras futuras

- Añadir a la unidad de PWM un nuevo módulo que funcione como una pequeña matriz de conmutación, permitiendo que una misma unidad de PWM se conecte a diferentes salidas, de manera que varios servos se encuentren en las mismas posiciones. En el ejemplo de las minicámras, si se quiere que estén perfectamente sincronizadas, realizándose los mismos movimientos, este módulo permitiría que se moviesen los 4 servos con sólo 2 unidades de PWM.

- Modificar el interfaz SW para permitir que algunos servos estén deshabilidatos (no tengan señal de PWM, o lo que es lo mismo que tenga un PWM de "ancho 0") mientras otros estén activos. Los servos deshabilitados no consumen y se pueden mover manualmente para hacer pruebas mecánicas, mientras que lo habilitados permanecen fijos en una posición, haciendo fuerza.

- Realizar un Software genérico para programar y reproducir secuencias desde cualquier ordenador conectado en red. Esto permitiría utilizar la tarjeta LABOMAT como un laboratório de ROBOTICA, en el que cada alumno puede probar sus aplicaciones de más alto nivel para el control de un robot conectado a la LABOMAT.

APENDICE A: Entorno de desarrollo

Para el desarrollo de este proyecto se ha utilizado lo siguiente:

- **VHDL**:
  - **Edición**: Para editor los fichero en VHDL se ha usado el editor nedit[9], para LINUX, que realiza un resaltado de la sintáxis.
  - **Simulación**: Entorno BLUEHDL[10], versión de estudiantil para plataformas LINUX. Todas las entidades VHDL, salvo la de nivel superior, se han simulado en este entorno, utilizando sus correspondientes bancos de pruebas.
  - **Síntesis**: Herramientas de Xilinx (Foundation) para entorno Windows :-(
- **DOCUMENTACIÓN**:
  - **Texto**: Lyx[11], entorno Linux. Procesador de texto que hace de "front-end" para Latex.
  - **Dibujos**: Xfig[12], entorno Linux. Creación de gráficos vectoriales.

APENDICE B: Agradecimientos

Tengo que agradecer a muchas personas su ayuda prestada para la realización de este trabajo:

- a **Iván González**, por su ayuda en el manejo de la tarjeta Labomat y del entorno Foundation. Yo nunca había utilizado este entorno y nunca había programado en VHDL. Gracias a Iván esto a resultado muchísimo más fácil. ¡Gracias!
- a **Sergio López-Bueno**, porque es "una máquina" y siempre saca tiempo de su apretada agenda para ayudarte en todo lo que pueda. Me ha ayudado mucho en mis primeros pasos con VHDL y Foundation. ¡Gracias!
- a **Gustavo**, por el tiempo que perdió instalándome el entorno Foundation de Xilinx en el portátil, así como otras herramientas muy útiles. ¡Gracias!
- a **Andrés Prieto-Moreno**, que es el autor de la estructura mecánica de las minicámaras y que yo he utilizado para este proyecto. ¡Gracias!

Referencias

[1] Proyecto de un robot humanoide de pequeñas proporciones [en línea]. http://www.symbio.jst.go.jp/PINO/index.html [Ultima consulta Feb-2002]

[2] Un hexápodo construido con tres servos [en línea]. http://www.geocities.com/acicuecalo/descrp_bcho5/descrp_bcho5.htm[Ultima consulta Feb-2002]

[3] Varios robots artículados diseñados por la empresa Microbotica[en línea]. http://www.microbotica.es/artic.htm [Ultima consulta Enero-2002]

[4] Brazos robots comerciales realizados con servos[en línea]. http://www.robotstore.com/catalog/list.asp?cid=2 [Ultima consulta Enero-2002]

[5] Robots con patas comerciales realizados con servos. http://www.robotstore.com/catalog/list.asp?cid=3[Ultima consulta Feb-2002]

[6] Tarjeta de desarrollo LABOMAT. http://www.ii.uam.es/~laboweb/LabWeb/index.php3 [Ultima consulta Enero-2002]

[7] Catálogo de servos de la casa Futaba. http://www.futaba-rc.com/servos/futm0029.html [Ultima consulta Feb-2002]

[8] Ejemplo de un multiplicador para la tarjeta LABOMAT. http://www.ii.uam.es/~laboweb/LabWeb/index.php3?seccion=1&pagina=2

[9] Editor de textos nedit que resalta la sintaxis de los programas en VHDL, entre otros lenguajes[En línea]. http://www.nedit.org/download/current.shtml [Ultima consulta Mar-2002]

[10] BLUEHDL. Entorno de simulación de programas en VHDL[En línea]. http://www.bluepc.com/bluehdl.html [Ultima consulta Mar-2002]

[11] Procesador de textos Lyx [En línea]. http://www.lyx.org/ [Ultima consulta: Mar-2002]

[12] Xfig. Programa de gráficos vectoriales[En línea]. http://www.xfig.org/ [Ultima consulta: Mar-2002]