Adaptación de Arquitecturas Software de Control de Robots en Tiempo Real a Plataformas Hardware Genéricas, Parametrizables y Reconfigurables

Requena F, Ortiz F.J, Suardíaz J, Iborra A

Universidad Politécnica de Cartagena UPCT, División de Sistemas e Ingeniería Electrónica DSIE
Campus Muralla del Mar,s/n. Cartagena, E-30202, Spain
francisco_requena@hotmail.com, francisco.ortiz@upct.es, juan.suardiaz@upct.es,
andres.iborra@upct.es
http://www.dte.upct.es

Resumen. En este artículo se presenta la implementación en una plataforma hardware reconfigurable de una arquitectura jerárquica y parametrizable para controlar ejes de robots de propósito general. Previamente se diseñó una arquitectura software de referencia para unidades de control de robots de servicios teleoperados (ACROSET), descrita con una notación semiformal, UML (*Unified Modelling Language)*. Esta arquitectura fue implementada inicialmente con el lenguaje Ada95, utilizando como plataforma un PC industrial. En el presente trabajo, utilizando el lenguaje VHDL y las herramientas Xilinx ISE 4.2, se presenta una implementación puramente hardware de dicha arquitectura en una FPGA. Todo este desarrollo se enmarca dentro de la investigación que realiza la División de Sistemas e Ingeniería Electrónica (DSIE) de la Universidad Politécnica de Cartagena (UPCT), a través del proyecto europeo *"Environmental Friendly and cost-effective Technology for Coating Removal"* (EFTCoR).

1. Antecedentes. Proyecto *GOYA*.

GOYA [6] es un sistema teleoperado para chorreado aplicado a la limpieza de cascos de buques como uno de los pasos de mantenimiento de barcos. El principal objetivo del proyecto *GOYA* fue el desarrollo de una tecnología fiable y de bajo coste respetuosa con el medio ambiente para el chorreado con granalla de cascos, capaz de obtener una superficie preparada de alta calidad junto con una reducción de residuos y emisión al medio ambiente. Esta tecnología es integrada en un sistema de chorreado completamente automatizado y de bajo coste. En los sistemas teleoperados como *GOYA*, un operador se encarga de mover el robot en función a la información proporcionada por la plataforma de teleoperación. El robot se mueve gracias a su unidad de control, que utiliza la información proveniente de los sensores y los comandos de movimiento provenientes del sistema de teleoperación para generar las señales convenientes hacia los actuadores (motores eléctricos, dispositivos hidráulicos, relés, etc.). Los principales componentes del sistema *GOYA* [2], [3] son la plataforma de teleoperación y la unidad de control, unidos por una conexión *Ethernet*, y finalmente el sistema mecánico (ver Fig.1).

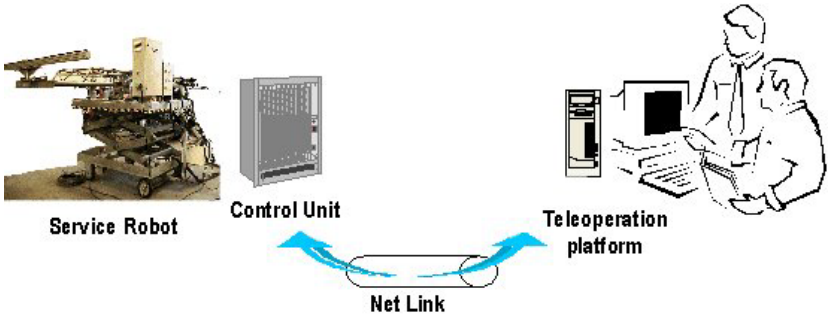

Fig. 1. Esquema general del sistema de teleoperación

El subsistema mecánico (Fig. 2) consta de los siguientes módulos funcionales:
- Plataforma de elevación (eje z): Este elemento mecánico está constituido por un sistema de elevación hidráulico que asciende o desciende gobernado por un actuador hidráulico.
- Brazo de posicionamiento (eje y): con el que se pretende acercar o alejar la herramienta de la superficie del barco sobre el eje Y. Está constituido por dos raíles móviles cada uno soportado por un par de patines. En su extremo soporta el cilindro neumático que lleva la herramienta de chorreo.
- Carreta de posicionamiento de la herramienta (eje x): La herramienta se monta sobre un carrete deslizante sobre el que actúa un cilindro neumático sin rodamiento. Este es el movimiento según el eje X.

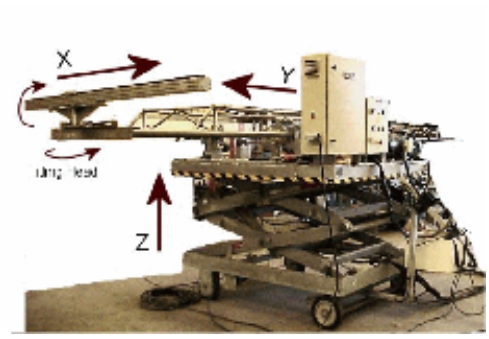

Fig. 2. Robot de limpieza de cascos de buques: GOYA **Fig. 3.** Robot 3D de *Fischertechnik*

- Herramienta: El material abrasivo es proyectado contra el casco del buque a través de una manguera. Su apertura y cierre están controlados por un sistema neumático.

En cuanto a la unidad de control del robot, se propuso una arquitectura de referencia software con posibilidad de ser implementada también en hardware. Con objeto de realizar pruebas de laboratorio con un prototipo más manejable, se utilizó una maqueta de "*robot 3D*" (Fig. 3) de entrenamiento de la marca *fischertechnik* [5]. Los resultados de dichas pruebas son fácilmente extrapolables al GOYA, así como a los robots que están siendo desarrollados paralelamente en el proyecto europeo *EFTCoR*, como mejora del prototipo *GOYA*.

2. La Unidad de Control

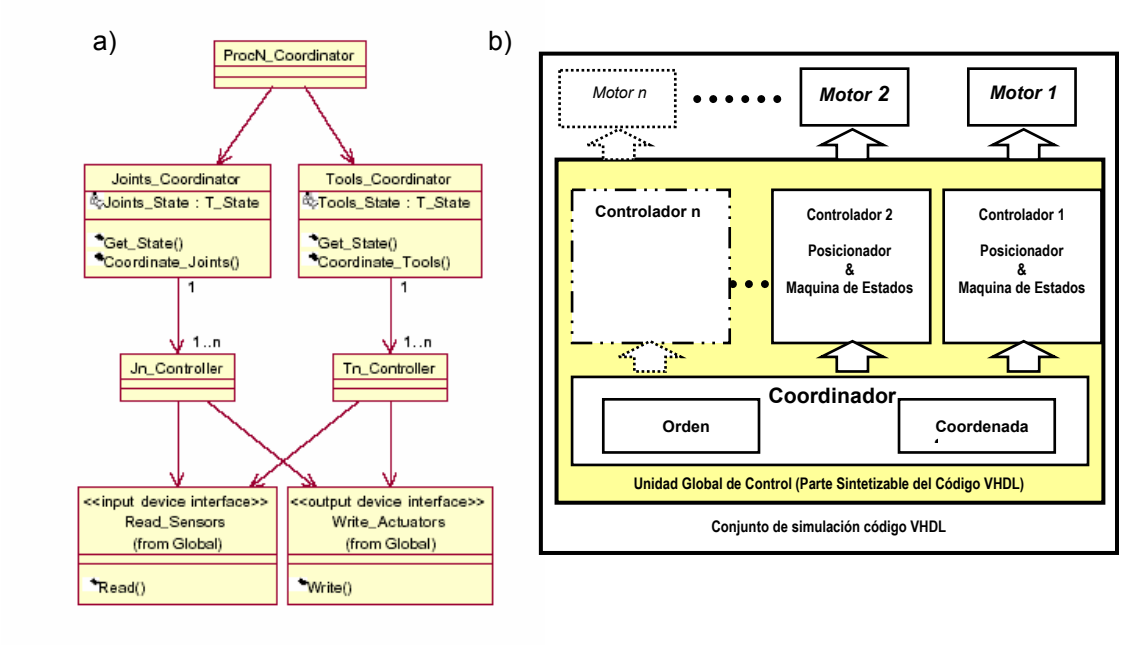

Fig. 4. a) Diagrama de clases UML de la arquitectura. b) Diagrama de programación VHDL

Para controlar un robot de servicios teleoperado como es GOYA, se propuso una arquitectura de referencia para la Unidad de Control (ACROSET) [4], [7], que pudiera ser reutilizada en otros robots de servicios con restricciones de tiempo real, y que se pueda implementar tanto en *software* como en *hardware* o una combinación de ambos. En el robot GOYA fue implementada en un PC Industrial usando el lenguaje Ada95 [1]. Para su especificación y diseño se utiliza una notación semiformal como es UML *(Unified Modelling Language)* [8]. En este artículo se presenta la implementación *hardware* de dicha arquitectura en una FPGA. En la figura 4.a se muestra un diagrama de clases UML donde se puede percibir que la arquitectura tiene una distribución jerárquica. De arriba abajo, se utiliza un coordinador de proceso, unos coordinadores de las distintas articulaciones del robot y de las herramientas, y finalmente, una serie de controladores *Jn_Controller* y multiplicidad 1..n, dependiendo del número de ejes que haya que controlar, así como un controlador de herramienta, *Tn_Controller*. El nivel más bajo de la jerarquía actúa como interfaz con los sensores y actuadores.

El número de componentes utilizados en la implementación de ACROSET, dependerá del tipo de robot utilizado. Por ejemplo, para el GOYA, resultarían los siguientes componentes: *J1_Controller* para la plataforma de elevación (eje z), *J2_Controller* para el brazo de posicionamiento (eje y) y *J3_Controller* para el posicionador (eje x). Solo se tiene una herramienta por lo que la multiplicidad del *Tn_Controller* es 1: *T1_Controller* para la herramienta de chorreado. Por encima de los controladores se encuentra el objeto coordinador (*Joints_Coordinator*) que es necesario para sincronizar los movimientos.

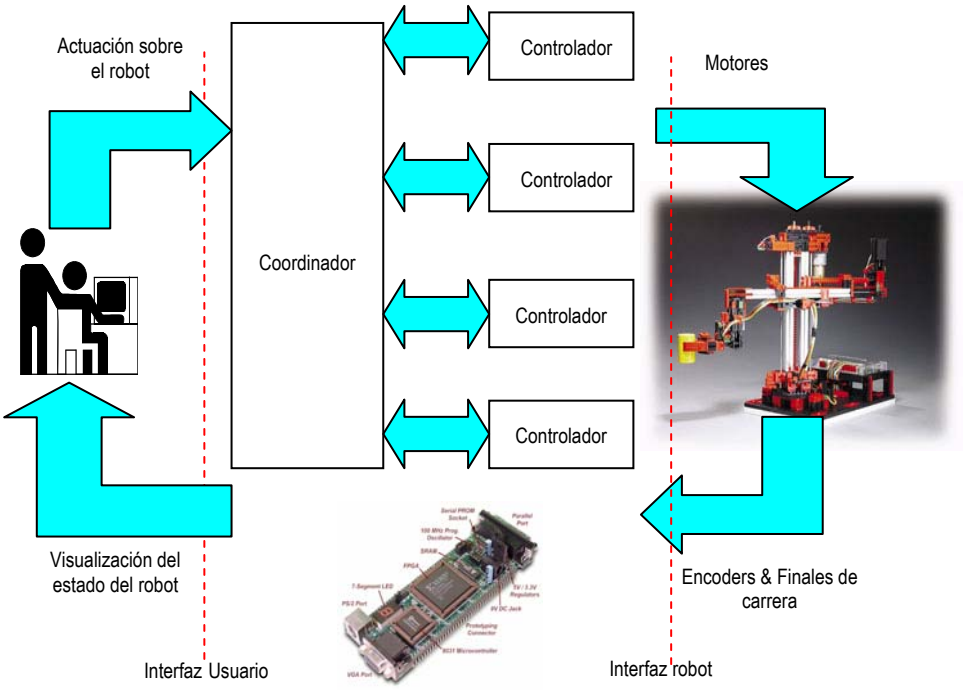

Fig. 5. Esquema del sistema de control del robot para su implementación hardware

Partiendo de la arquitectura propuesta, se procede en este trabajo a realizar una implementación totalmente hardware de la misma, originando una implementación *VHDL [10]* jerárquica y parametrizada, de tal manera que pueda servir para diferentes configuraciones de robots (ver Fig. 5). Como herramienta de desarrollo se utiliza *Xilinx ISE 4.2* [11]. Al igual que se definió en ACROSET, se dispone de distintos controladores de un solo eje, coordinados a su vez por otro elemento que recibe el nombre de *coordinador*. En la arquitectura desarrollada, se pretende que las órdenes de control de todos los ejes, accedan al *coordinador* en un único vector, y que sea éste el encargado de distribuir las diferentes órdenes y coordenadas a los distintos *controladores* de cada motor. A su vez cada *controlador* se encarga de accionar cada motor y recibir la información de posición mediante el empleo de *encoders* y finales de carrera.

En la implementación *hardware*, la instanciación de uno o más coordinadores, no supone ningún problema a la hora de generar código, pues la conectividad y la interoperabilidad están totalmente garantizados. Como se vio anteriormente en la presentación del robot objeto de las pruebas, su simplicidad nos permite integrar el control de la herramienta como el de cualquier otro eje. Respecto a la unidad de control, se ha implementado en lenguaje *VHDL,* para su posterior síntesis en *FPGA,* la estructura de un *coordinador* y un *controlador* genéricos y parametrizables, de tal manera que se les pueda usar en cualquier diseño, independientemente del ancho de buses y número de ejes (ver Figs. 7 y 8). A su vez, y con el objeto de poder simular el comportamiento del sistema mediante la herramienta *Modelsim [13]*, se ha diseñado en *VHDL* un "*motor*" para poder

hacer la simulación completa, que no será posteriormente sintetizado. La forma en la que se ha llevado a cabo la implementación del sistema puede verse en la figura 4.b.

2.1 El controlador.

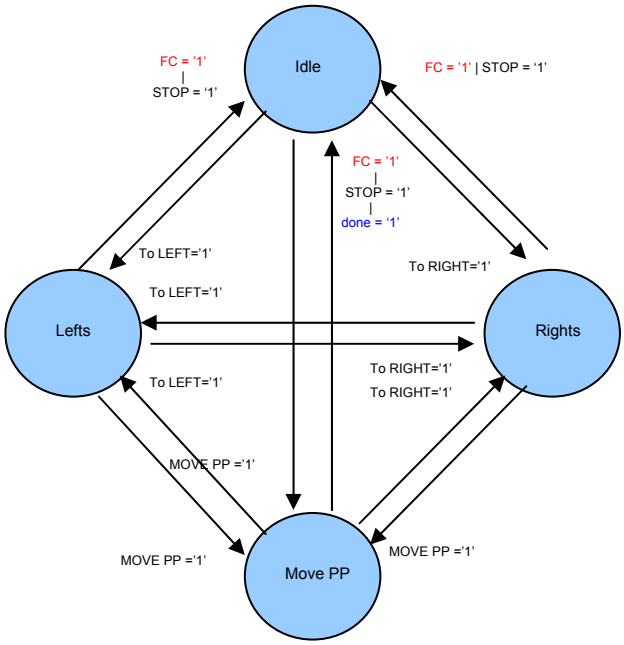

Fig. 6. Máquina de estados del controlador

Para la implementación de la unidad de control se ha separado el diseño en dos partes, la *máquina de estados* y el *contador de posiciones* que está dentro del posicionador. La *máquina de estados* define cuatro estados posibles de funcionamiento del *controlador*:

Estado	Entradas	Variables de Estado
Idle	To left	FC='1'
Rights	To right	Done='1'
Lefts	Move PP & Coord	
Move PP	Stop	

Tabla 1. Representación de los estados del sistema

El funcionamiento de la *máquina de estados* se puede ver reflejado en la figura 6. Partiendo del estado inicial *Idle* se puede alcanzar cualquiera de los otros tres estados, dando la entrada correspondiente. Desde cada uno de los estados, se puede en cualquier momento alcanzar cualquiera de los otros tres, con la orden pertinente. En el caso de estar activado un final de carrera, el sistema pasará inmediatamente a *Idle*, solo pudiendo salir de él, utilizando la orden pertinente para desactivar físicamente el final de carrera. En el estado de movimiento punto a punto cuando se activa la señal de éxito (done) se ha alcanzado la coordenada seleccionada y se pasa a estado *Idle*. Una de las grandes ventajas que nos ofrece esta arquitectura es la fácil adaptación de la estructura global, al propio sistema de control. En este ejemplo se ha diseñado una *máquina de estados*, pero se podría optar por emplear otro tipo de sistema de control, como un *PID* o un *control Fuzzy* implementado en la misma *FPGA*, sin tener que variar apenas la estructura. Creando librerías de controladores, se podrían seleccionar diferentes tipos de control para cada eje, pudiéndose alcanzar un gran número de configuraciones y pudiendo por tanto diseñar a medida las unidades de control, dependiendo del *robot* y la aplicación de éste. Esto hace que la estructura nunca deje de mejorarse y perfeccionarse, de ahí el interés de nuestro trabajo.

Entradas	Función	Salidas	
clk	Reloj	Done	Posición alcanzada
A_rst	Reset Asíncrono	Error	Señal de error
reset	Reset Síncrono	Output	Orden al motor
encoder	Encoder		00 Parado
FC_L	Final de carrera derecha		01 A derechas
move	Orden de movimiento		10 A izquierdas
			11 Parado (Forzado por programa)

Tabla 2. Tabla de entradas/salidas

El final de carrera de la derecha no está físicamente en el robot, pero se implementa lógicamente en la FPGA mediante lógica combinacional, a partir de la salida del posicionador *axis_position*. Ver figura 8.

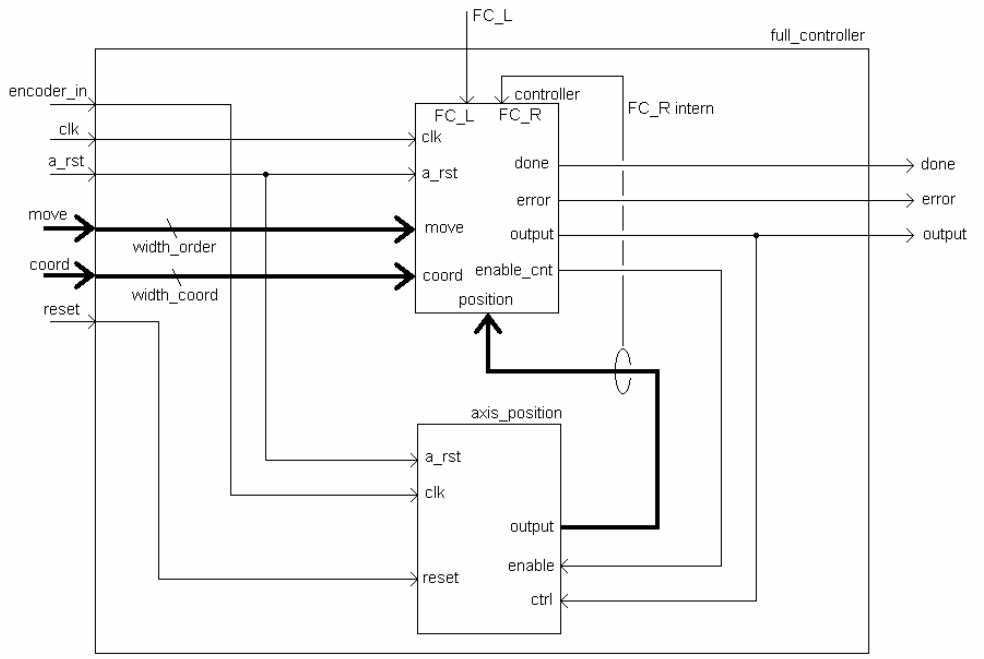

Fig. 7. Módulo controlador compuesto de controlador y posicionador

2.2 El coordinador

El *coordinador* se encarga de separar las ordenes de movimiento de las coordenadas en cada uno de los comandos y para cada motor. El diseño consta de '*n'* registros para almacenar el código de movimiento y *'n'* para almacenar las coordenadas, siendo *'n'* el número de motores. Los comandos entran al *coordinador* agrupados en un único vector y son separados en dos *arrays* de vectores de órdenes y de coordenadas.

El vector de reset síncrono no se ha modificado y se deja directamente conectado a la salida por si en futuras mejoras se pretendiera realizar alguna modificación dentro del coordinador.

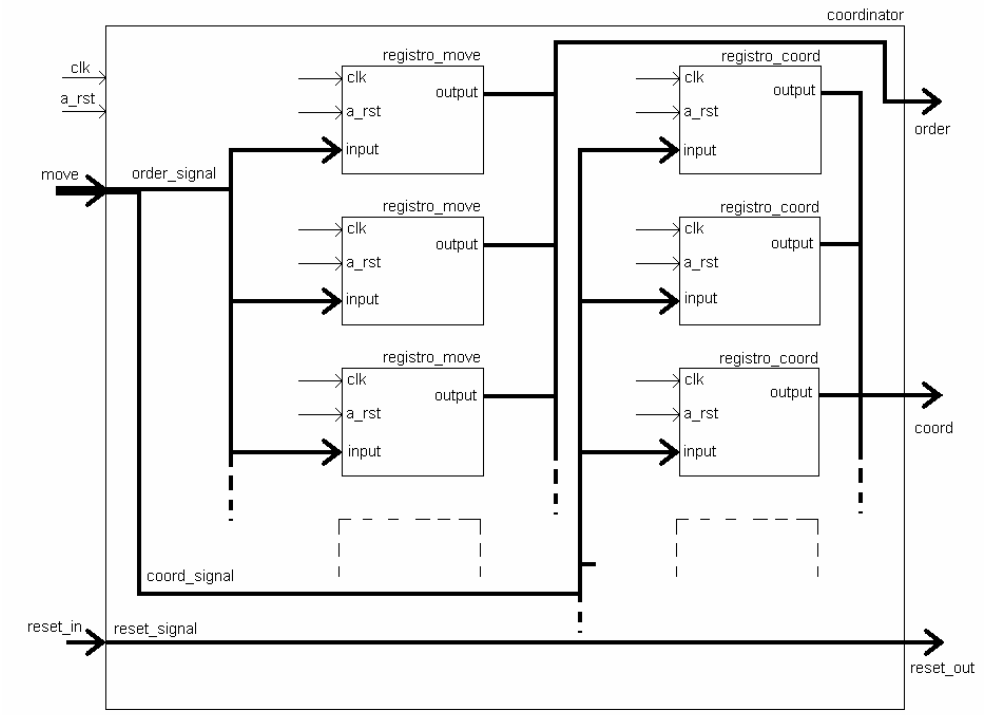

Fig. 8. Modulo coordinador compuesto de registros de orden y coordenada

3. Simulación para un caso concreto.

Para demostrar la funcionalidad de la arquitectura propuesta, se ha realizado una simulación del comportamiento del robot 3D. Se dispone de cuatro motores, tres para los ejes y uno para la herramienta y un coordinador, por lo tanto la variable *NUM_JOINTS* del código *VHDL* se fija en 4. Por otro lado se debe escoger el ancho del bus de direcciones y del bus de orden de movimiento (*WIDTH_COORD* = 8 y *WIDTH_ORDER* = 2 respectivamente). El periodo de la señal de reloj es de 100 ps.

La simulación comienza con el reseteo asíncrono de todos los registros del sistema. El reseteo síncrono queda reservado para funciones de calibración, esta simulación parte del supuesto de que la calibración ya ha sido realizada y se parte de una posición del robot en la que todos los finales de carrera están activados, es decir, todos los contadores de posición se encuentran a cero. El vector de movimiento global incluye las órdenes de movimiento y las coordenadas.

Movimiento				Coordenada			
01	01	11	10	00000000	00000000	00001111	00000000

Los ocho primeros bits representan las ordenes de movimiento a cada uno de los motores y los 32 restantes, los 4 vectores de coordenadas. El ejemplo consiste en mover los dos primeros motores a la derecha, el tercero a la coordenada 15 y el cuarto a la izquierda. Solo el movimiento punto a punto requiere de coordenada, por lo que los otros movimientos tienen la coordenada a 0. La simulación nos muestra que los motores 1 y 2 comienzan a moverse y no paran hasta que se le ordena la parada con ‘00’. El motor 3 realiza su movimiento y se detiene en la posición 15. El motor 4 no se pone en funcionamiento por estar el final de carrera a izquierdas activado, por eso se activa la señal de error. De los datos de salida obtenidos en la simulación se observa el correcto funcionamiento en condiciones normales y de error. En Tabla 3 se puede ver un resumen de los resultados de la implementación. Al ser una plataforma hardware de control, la concurrencia nos asegura una rápida respuesta a las órdenes del operador y al entorno. El proceso de calibración no se ha simulado, pero podría perfectamente implementarse como un estado más dentro del controlador y poder hacer que todos los motores se muevan hacia la posición de inicio (finales de carrera a cero) y reseteando a continuación los contadores de posición.

Virtex II Device Usage	
Device Usage:	
Registers	32
Counters	4
Multiplexers	4
Adders/Substractors	12
Comparators	20
Flip-Flops/Latches	140
IO Buffres	61
Frecuency:	
Maximum value	145.794 MHz

Tabla 3. Tabla resumen de implementación

4. Conclusiones

Se ha adaptado una estructura de control orientada a *software en tiempo real*, a una plataforma *hardware*, consiguiendo garantizar la *concurrencia* de los procesos de control, integrando el sistema en un solo *chip* y dejando abierta la estructura a futuras mejoras y modificaciones. Este trabajo sirve como base a futuras investigaciones del *DSIE* relacionadas con el diseño de arquitecturas de control de robots y más concretamente abre una nueva línea de investigación en estructuras hardware de control. A raíz de él se comienza a crear una librería de componentes *VHDL*, que están siendo usados actualmente en la elaboración de sistemas de control más complejos que el aquí expuesto, cuya funcionalidad y prestaciones están siendo probadas en nuestros proyectos de investigación, con notables resultados.

Reconocimientos

Este trabajo ha sido financiado parcialmente por el proyecto EFTCoR de la *Unión Europea* dentro del V Programa Marco (GROWTH G3RD-CT-00794). Se han obtenido fondos adicionales de *IZAR Construcciones Navales, S.A*., el *Ministerio Español de Ciencia y Tecnología* así como la *Fundación Séneca* del *Gobierno Regional de Murcia*.

Referencias

1. Ada 95 Reference manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995. Springer-Verlag, LNCS no. 1246.
2. Alonso, A., Álvarez, B., Pastor, J.A., de la Puente, J.A., and Iborra, A.: Software Architecture for a Robot Teleoperation System. 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control, (1997).
3. Álvarez B, Iborra A, Alonso A, De la Puente, J.A, and Pastor,J.A.: Developing multi-application remote systems. Nuclear Engineering International. Vol.45, No. 548, (2000). ISSN: 0029-5507.
4. Álvarez B, Ortiz F., Martínez A , Sánchez, P, Pastor JA, Iborra A - Towards a Generic Software Architecture for a Service Robot Controller. 15th IFAC World Congress, Barcelona, (2002).
5. Fischertechnik www.fischertechnik.de.
6. Ortiz, F., Iborra, A., Marin, F., Álvarez, B., and Fernandez, J.M.: GOYA - A teleoperated system for blasting applied to ships maintenance. 3rd International Conference on Climbing and Walking Robots, Spain, (2000).
7. Ortiz, F., Martínez, AS., Álvarez, B., Iborra, A., Fernández, JM.: Development of a Control System for Teleoperated Robots Using UML and Ada95. In: Blieberger, J., Strohmeier, A. (eds.): Lecture Notes in Computer Science, Vol. 2361. Springer-Verlag, Berlin Heidelberg New York (2002) ISBN 3-540-43784-3
8. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, Boston, (1998).
9. Scott, N.R.: Computer Number System and Arithmetic – The University of Michigan Prentice Hall (1985).
10. IEEE Standard VHDL Language Reference Manual. Std 1076-1993, 1993.
11. Vanden Bout, D., Xilinx inc. XS40 Manual v 1.4.
12. Yeap, G.K.: Practical Low Power Digital VLSI Design Kluwer Academic Publishers (1998).
13. Mentor Graphics ModelSim. http://www.model.com.