Control de los movimientos de un robot cartesiano usando un sistema embebido en un módulo FPGA integrado con dos núcleos ARM Cortex-A9

Motion control of cartesian robot using an embedded system into a FPGA integrated module with dual-core ARM Cortex-A9

Erick Fiestas Sorogastúa[1], *Filiberto Azabache Fernández*[2] y *Sixto Ricardo Prado Gardini*[3]

Recibido: 15 de mayo de 2017
Aceptado: 29 de mayo de 2017

Resumen

En el presente trabajo se describe una metodología para el control de movimiento de un robot cartesiano mediante el uso de la tarjeta De0_Nano_SoC de Intel, que trabaja integrando la potencia de procesamiento de un FPGA y un sistema operativo robusto que se ejecutan en dos núcleos de arquitectura ARM Cortex–A9. Primero se analiza, mediante *Solidworks*, la estructura del robot cartesiano, segundo se obtiene el modelado cinemático del robot y con ello se desarrolla el planificador de trayectorias que usa procesamiento digital de imágenes y un interpolador de tercer orden. Tercero, se diseña un generador de velocidad en tiempo real para motores paso-paso. Cuarto, con los algoritmos validados se pasa a embeberlos, usando lenguaje C, en el sistema de procesamiento (HPS, *Hard Processor System*) de la De0_Nano_SoC, mientras que en el FPGA de la tarjeta se diseñan con VHDL los generadores de pulsos eléctricos para los pasos de los motores paso-paso. Quinto, se desarrolla la interfaz gráfica de usuario para la interacción hombre-máquina. Finalmente, se realizan las validaciones y se muestran los resultados.

Palabras claves: robótica, SoC FPGA, máquina CNC, la cinemática, interpolador cúbico.

Abstract

In this work we describe a methodology to the motion control of a robot making use of an Intel De0_Nano_SoC card which works to exploit the full potential of the FPGA and a solid operational system which is run on a dual-core ARM Cortex-A9 Hard Processor System (HPS). First, we analyze through Solidworks the structure of the cartesian robot. Second, we obtain the kynematic modelling of the robot and this could help to develop the planner of trajectories that uses a digital image processing and an interpolator of third order. Third, we design a generator of real-time speed suitable for stepper motors. Fourth, we embed them with the validated algorithms, using C language, in the Hard Processor System (HPS, *Hard Processor System*) of the De0_Nano_SoC, meanwhile in the FPGA of the card we design with VHDL generators of electric pulses for the process steps of the stepper motors. Fifth, we develop a graphic user interface for the human-machine interaction. Finally, we perform the validations and we show the results.

Keywords: robotics, SoC FPGA, CNC Machine, the kinematics, cubic interpolator.

[1] Ingeniero Electrónico egresado de la Escuela de Ingeniería Electrónica de la UPAO, miembro de apoyo del Área de Emprendimiento e Innovación del Vicerrectorado Académico de Investigación de la UPAO.

[2] Magister en Ingeniería con mención en Gerencia de Operaciones. Director de la Escuela de Ingeniería Electrónica de la UPAO.

[3] Doctor en Robótica y Automatización, investigador del Área de Emprendimiento e Innovación del Vicerrectorado Académico de Investigación de la UPAO.

I. INTRODUCCIÓN

Desde tiempos antiguos la técnica ha permitido al hombre transformar la materia en beneficio propio. La Primera Revolución Industrial (siglo XVIII) permitió visualizar la importancia de perfeccionar las técnicas industriales para aumentar la producción mejorando la calidad y manteniendo la homogeneidad en todos los productos obtenidos. Por lo mismo, la técnica y la ciencia han tenido un papel preponderante en el desarrollo de la humanidad, desarrollándose tecnologías altamente integradoras y complejas como el robot industrial que una vez insertado en un proceso industrial, ha permitido satisfacer la demanda productiva como los estándares de calidad de los países desarrollados.

Es por ello que la venta de los robots industriales va en aumento cada año, teniendo como proyección de ventas para el 2018 de unos 400000 unidades (International Federation of Robotics, 2015). El robot industrial presenta una gran versatilidad y flexibilidad en su trabajo dado que pueden realizar diferentes funciones (manipulación, soldadura, paletizado, etc) mediante el cambio de la herramienta de trabajo o usando un efector-final multipropósito. La flexibilidad con la que cuenta le permite fabricar productos de diversas formas, tamaños y pesos en una misma línea de producción. Todo aquello automáticamente y sin necesidad de cortar la continuidad de los procesos productivos.

Sin embargo, el alto grado de flexibilidad y versatilidad del robot industrial, implica el uso de complejos algoritmos matemáticos, procesamiento de data multisensorica en tiempo real, protocolos de comunicación, etc. Todo ello requiere de soportes electrónicos robustos y estables. En ese sentido, los integrados de Arreglos de Compuertas Programables en Campo (FPGA, por sus siglas en inglés) presentan actualmente un alto grado de prestaciones técnicas (Altera, 2007; National Instruments, 2011):

- Precio: Tiene en un solo chip las funciones realizadas anteriormente por varios chips. Lo que reduce la complejidad del PCB y los costos de logística y producción.
- Paralelismo: las tareas pueden llevarse a cabo en forma paralela y al mismo tiempo utilizar de manera independiente los recursos del sistema, tales: ciclos de reloj, memoria, periféricos de E/S, protocolos, etc.
- Personalización: Permite diseños digitales personalizados.
- Versatilidad: Permite implementar cualquier arquitectura de trabajo.
- Flexibilidad: Permite programar múltiples funciones.
- Lanzamiento: Reduce los tiempos de desarrollo y validación del prototipo dado que el diseño, testeos y correcciones se dan en menor tiempo.
- Emulabilidad: Emulan el trabajo de diferentes tipos de microprocesadores *(softcore).*
- Reciclabilidad: Diseños previos en un FPGA pueden ser usados en proyectos a futuro.
- Confiabilidad: Ejecuta el trabajo de forma estable y sin interferencias entre diferentes procesos en paralelo.

El FPGA trabaja con un sistema operativo estable y seguro, lo que sumado a sus resaltantes características técnicas lo convierte en una computadora en sí. El De0_Nano_SoC de Intel es una tarjeta de desarrollo que integra las prestaciones de un potente FPGA y un HPS *(Hard Processor System),* que es un microprocesador con arquitectura *Advanced RISC Machine* (ARM) de ARM Holdings. El que es usado en este trabajo para implementar de forma embebida el control de movimiento de un robot, el que a su vez puede ser aplicado en otros proyectos.

II. MATERIAL Y MÉTODOS

A. Materiales

En la tabla I se muestra los principales materiales usados en este trabajo.

Tabla I. Lista de materiales

N°	Parte Mecánica	Cantidad
1	CNC Joyo – 2020	1
2	Acople de lapicero al eje Z	1
N°	**Parte de Control**	**Cantidad**
1	De0_Nano_SoC	1
2	Switch final de carrera	5
N°	**Parte de Comunicación**	**Cantidad**
1	Router	1
N°	**Parte de Potencia**	**Cantidad**
1	Controlador DRV8825 de motores paso-paso	1
2	Controlador TB6560AHQ de motores paso-paso	2
3	Fuente switching 24VDC	1
4	Ventiladores 12VDC	6

Fuente: elaboración propia.

B. Análisis estructural

Todo proyecto de desarrollo de un robot debe asegurar la estabilidad y robustez de su estructura mecánica, caso contrario ninguna estrategia de control podrá realizar su trabajo de forma adecuada por más fina que pueda ser la ley de control o la tecnología usada para implementarla. En este trabajo se realiza el análisis estructural mecánico mediante el CAD *solidworks* que permite, entre diversas aplicaciones, el análisis estático lineal que calcula: desplazamientos, deformaciones unitarias, tensiones y fuerzas de reacción bajo el efecto de cargas aplicada (Dassault Systems, 2014). Ver tabla II.

Tabla II. Análisis estático lineal

Estudio	Descripción
Tensión de Von Mises	Llamada también: teoría de la distorsión máxima. Es utilizada para someter un material dúctil a una tensión máxima, o límite elástico, en miras de cuantificar su deformación.
Desplazamiento	Orientada a medir en milímetros el desplazamiento producto de la fuerza aplicada sobre una zona del objeto.
Deformación unitaria equivalente	Es un índice que mide la deformación de los ejes sometidos a esfuerzos de tensión o comprensión axial.
Factor de seguridad	Este valor cuantifica la seguridad, según un criterio de fallos, ante posibles fisuras. Un factor menor a uno, indica que el material ha fallado; o uno, que ha empezado a fallar. Y si es mayor a uno, que es seguro.

Fuente: elaboración propia.

El análisis estructural, basado en los criterios de la tabla I, determina que las áreas críticas del robot cartesiano se encuentran en los soportes laterales (fig. 1). El plano mostrado en la figura 1 es perpendicular al eje Z que tiene la misma dirección que la herramienta de trabajo.

C. Modelado cinemático

A fin de modelar el movimiento de la herramienta de trabajo del robot cartesiano se obtiene la cinemática directa e inversa del mismo mediante matrices de transformación homogénea (4x4) (Craig 2006, Barrientos et. al., 2007).

D. Generación de trayectorias

La herramienta de trabajo del robot cartesiano sigue una trayectoria continua obtenida desde una imagen tipo .jpg o .png. mediante procesamiento digital de imágenes y técnicas de interpolación, que finalmente proporcionan una trayectoria suave para el movimiento del robot (poner referencias estándares). En definitiva los pasos a seguir para obtener y simular las trayectorias de trabajo del robot son:

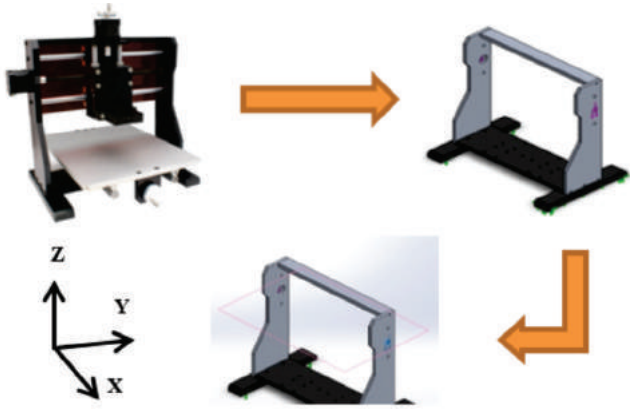


Figura 1. Máquina CNC JOYO-2020 que sirve de base física para el robot cartesiano y modelo CAD del mismo.

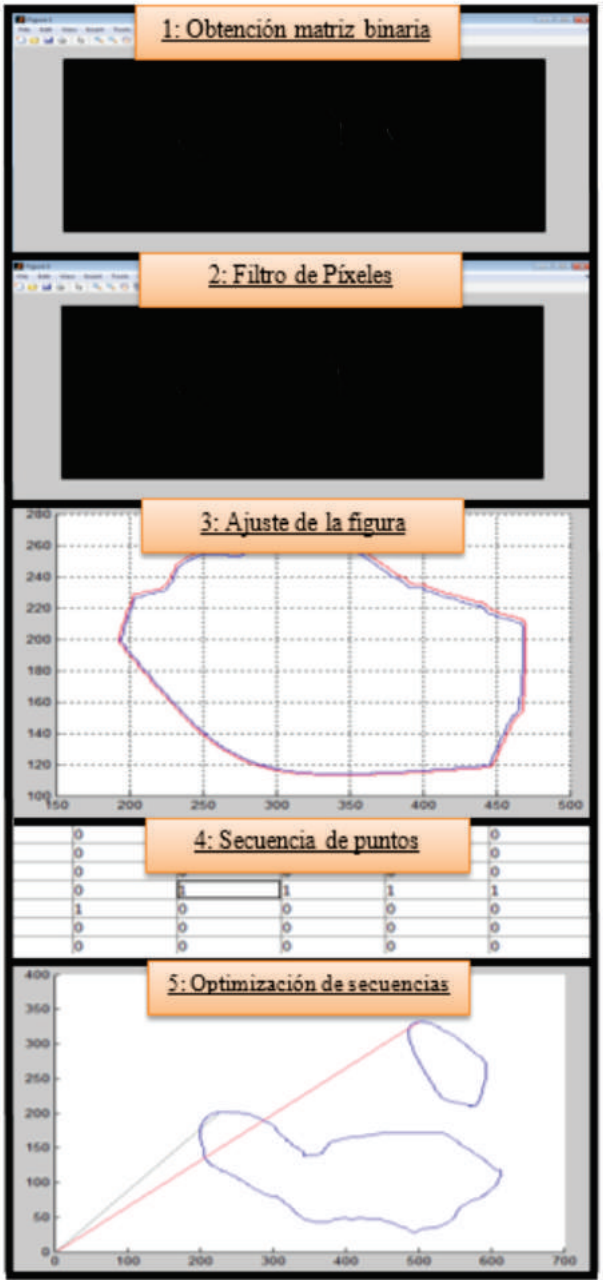


Figura 2. Los 5 pasos para determinar la trayectoria de la herramienta del robot mediante procesamiento digital de imágenes.

- Extraer las coordenadas cartesianas de la imagen .jpg o .png.
- Convertir las coordenadas a valores articulares mediante la solución de la cinemática inversa.
- Interpolar los valores articulares de cada articulación.
- Determinar las velocidades angulares de los motores paso-paso (MPP), por cada valor interpolado.
- Accionar cada articulación.

1. Procesamiento digital de imágenes

En la fig. 2 se muestra el procedimiento para la extracción de las coordenadas cartesianas a partir de una imagen .jpg o .png. El algoritmo se implementa en un *script* de Matlab y hace uso de unas funciones del mismo entorno de programación para la binarización (paso 1) o y para el ajuste de la figura (paso 3).

Es importante conocer las medidas de los píxeles de la computadora que se está usando, para fines de escalado en milímetros de los dibujos .jpg o .png. En DuffuD (2013), se muestra las ecuaciones y características técnicas de la computadora que se deben conocer para hallar las medidas precisas de los píxeles en milímetros.

2. Interpolador cúbico

Se usa el interpolador cúbico para obtener un mejor ponderado entre una adecuada suavidad en las trayectorias y menor complejidad de cálculo (por ejemplo con respecto al interpolador quíntico). Barrientos et. al. (2007) y Fiestas, Castillo, y Briones, W. (2013), indican que el interpolador cúbico cuenta con cuatro condiciones de contorno (en función a valores establecidos por el usuario, a excepción de la última que puede ser hallada por fórmula: la posición, tiempo y velocidad ambos con respecto a cada posición), siendo las dos primeras las que aseguran que se comienza y se termina en los puntos adecuados; y las dos siguientes, que las velocidades de inicio y las finales sean nulas. Se hace uso de la Ec. (1).

$$q(t) = a + b(t - t^i) + c(t - t^i)^2 + d\left(t - t^i\right)^3 \quad (1)$$

$$a = q^i \quad (2)$$

$$b = \dot{q}^i \quad (3)$$

$$c = \frac{3}{T^2}\left(\dot{q}^{i+1} - \dot{q}^i\right) - \frac{1}{T}\left(\dot{q}^{i+1} + 2\dot{q}^i\right) \quad (4)$$

$$d = -\frac{2}{T^3}\left(\dot{q}^{i+1} - \dot{q}^i\right) + \frac{1}{T^2}\left(\dot{q}^{i+1} + \dot{q}^i\right) \quad (5)$$

$$T = t^{i+1} - t^i \quad (6)$$

donde:

t: tiempo.

i: posición en la trayectoria.

$\dot{q}$: velocidad.

3. Algoritmo de David Austin

El efector-final del robot puede seguir trayectorias con velocidad constante según el interpolador cúbico. Sin embargo, a fin de asegurar una mejor performance en los motores paso-paso (MPP), es preferible realizar gradaciones en las velocidades angulares de los MPP de forma que se evita el sobrecalentamiento del motor y los controladores (Kordik, 2015). El algoritmo de Austin (2004), permite generar un vector de conteos mediante ecuaciones que requieren de poca carga computacional. El conteo (llamado

«c», en el presente trabajo) es un valor adimensional usado en el *Timer* de los microcontroladores, que al ser multiplicado por el tiempo (en segundos) del ciclo de instrucción, permite establecer un período igual al tiempo que debe esperar el controlador del MPP para reducir o aumentar la velocidad angular, ec. (7) y (8) respectivamente. El vector de conteos, por lo tanto, es un arreglo con el cual se tiene los períodos del perfil de velocidades del MPP.

$$\delta t = \frac{c}{f} \tag{7}$$

$$\omega = \frac{\alpha f}{c} \tag{8}$$

El vector de conteos, mediante las ec. (9), (10) y (11) respectivamente, presenta comportamientos de tiempo y velocidad angular según las gráficas mostradas en la Fig. 3.

$$c_0 = 0.678 f \sqrt{\frac{2\alpha}{\omega'}} \tag{9}$$

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n+1}\frac{n2-n}{n2-n1}, n1 \le n \le n2 \tag{10}$$

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4(n-m)+1}\frac{n-n3}{m-n3-1}, n3 \le n \le m \tag{11}$$

E. Programación del FPGA

En este trabajo, la Unidad de Control de Robot (RCU) se divide en la Unidad de Procesamiento de Datos (DPU) y la Unidad de Bucle de Control (CLU). Este último se encarga de generar las señales eléctricas, de procesar las señales provenientes de los sensores y activar desactivar cualquier función o herramienta del robot (taladros, ventiladores, etc). El CLU, se implementa en el FPGA de la De0_Nano_SoC.

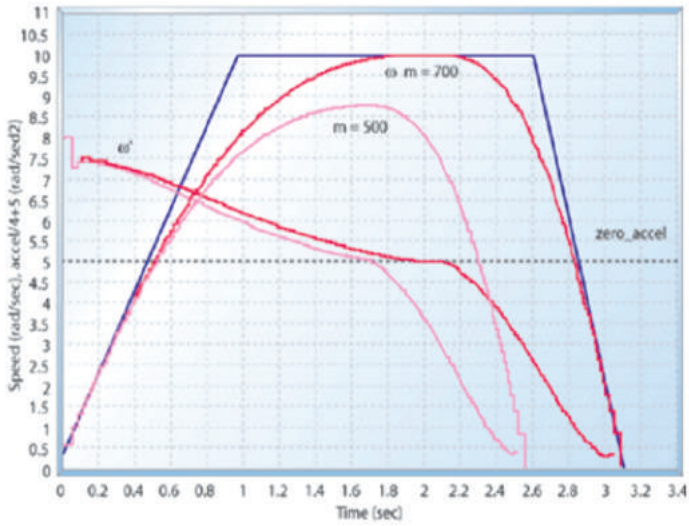


Figura 3. Graficas de velocidad del MPP usando 500 pasos y 700 pasos respectivamente, y comparados con una señal de referencia tipo trapecio. Fuente: Austin (2004).

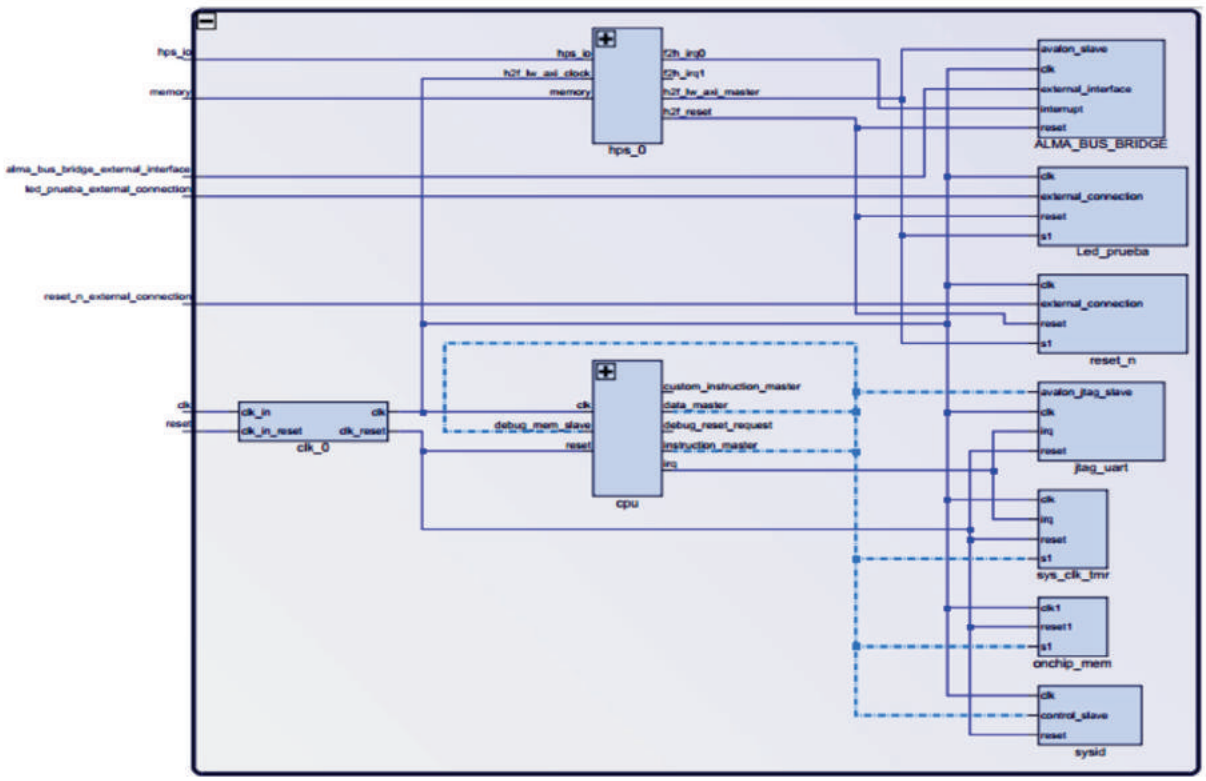


Figura. 4. Sistema Qsys basado en un softcore (bloque: «cpu») y un puente hacia el HPS («hps_0») pag. 8 corregir

La implementación utiliza el software Quartus Prime versión 16.0 Lite Edition y el Qsys 16.0. Este último contiene los IP *Cores* (librerías configurables de núcleo software), que son necesarios para poder implementar un sistema embebido en chip (SoC) en la que interactúan: uno o más *softcores* (hardware de procesador basado en software), un *hardcore* (el HPS, *Hard Processor System,* o ARM Cortex-A9 de dos núcleos) y diversos recursos adicionales.

La implementación de la solución primero usa el Qsys, para desarrollar un *softcore* con sus respectivos periféricos y el puente «h2f_lw_axi_master» de 32 bits para la comunicación entre el HPS y el FPGA.

En la fig. 4 se puede observar el sistema Qsys. El *softcore* (bloques inferiores) cuenta con una comunicación serial JTAG_UART, un temporizador SYS_CLK_TMR, una RAM ONCHIP_MEMORY y un id SYSID. Por otro lado, el HPS_0 utiliza tres bloques de nombres arbitrarios: ALMA_BUS_BRIDGE *(Avalon to external bus bridge),* que activa el «h2f_lw_axi_master» que permite la comunicación entre HPS y el FPGA, lo que permite enviar información como el vector conteo, c, que indica el periodo de trabajo; LED_PRUEBA, permite realizar testeos para lograr un buen desempeño del HPS_0; y RESET_N, que reinicia todos los bloques del FPGA. El reloj de la tarjeta trabaja a 50 MHz.

El sistema Qsys genera códigos VHDL a partir del sistema configurado por el usuario que resulta en generar pulsos eléctricos. Por otro lado, en la fig. 5, se muestra el archivo *.bdf (Block Design Files)* del hardware programado en el FPGA y que se explica a continuación:

Bloque generador de pulsos para el MPP del eje X:

- *clk:* señal del reloj a 50 MHz.
- *Pulsos_com_write_data[31..0]:* recibe el valor del conteo que dividido por la frecuencia del reloj, da el valor del período en segundos.
- *Pulsos_com_reset_n:* resetea el bloque.
- *Pulsos_com_enable:* habilita el bloque.
- *Pulsos_com_query:* bandera de solicitud para saber si ya se ha generado el pulso deseado.
- *Pulsos_com_start:* comenzar con la lectura del nuevo período del pulso a generar.
- *Pulsos_com_direc_rot:* dirección de giro del motor; 1 lógico, en sentido horario.
- *Fin_de_carrera:* cuando se activa el switch de final de carrera llega un 1 lógico a esta entrada, y se detiene el trabajo del bloque.
- *Pulse_ready:* una vez que se ha generado el pulso, un 1 lógico sale por este pin.

- *Pulsos:* salida de los pulsos de paso hacia el controlador del MPP.
- *Direc:* salida del pulso de dirección de giro del motor hacia el controlador del MPP.

El bloque D recibe el valor de 32 bits enviado por el HPS para decodificarlo y distribuirlo a un bloque GP. El bloque D utiliza el bit 30 para determinar si el giro del motor será horario o antihorario. Por consiguiente, el valor de conteo para el período del pulso es de 30 bits. Las entradas y salidas a de este bloque se describen a continuación.

- *clk:* señal del reloj a 50 MHz.
- *reset_n:* resetea el bloque.
- *X_Fin_de_carrera:* recibe la señal del switch de final de carrera del eje X. Cuando se activa, se detienen las operaciones.
- *Y_Fin_de_carrera:* del eje Y.
- *Z_Fin_de_carrera:* del eje Z.
- *address[10..0]:* señal de entrada que direcc iona las órdenes dentro de la lógica del bloque.
- *bus_enable:* señal de entrada que habilita la señal address[10..0] en el código VHDL.
- *byte_enable[3..0]:* habilita solo ciertos sectores de órdenes dentro de la lógica del bloque.
- *rw:* señal de entrada que establece si el HPS va a leer o a escribir.
- *write_data[31..0]:* recibe el valor del conteo para el período y la dirección de giro de un determinado MPP.
- *X_pulse_ready:* recibe el aviso de que el pulso para el MPP X ha sido generado.
- *Y_pulse_ready:* … Y ha sido generado.
- *Z_pulse_ready:* … Z ha sido generado.

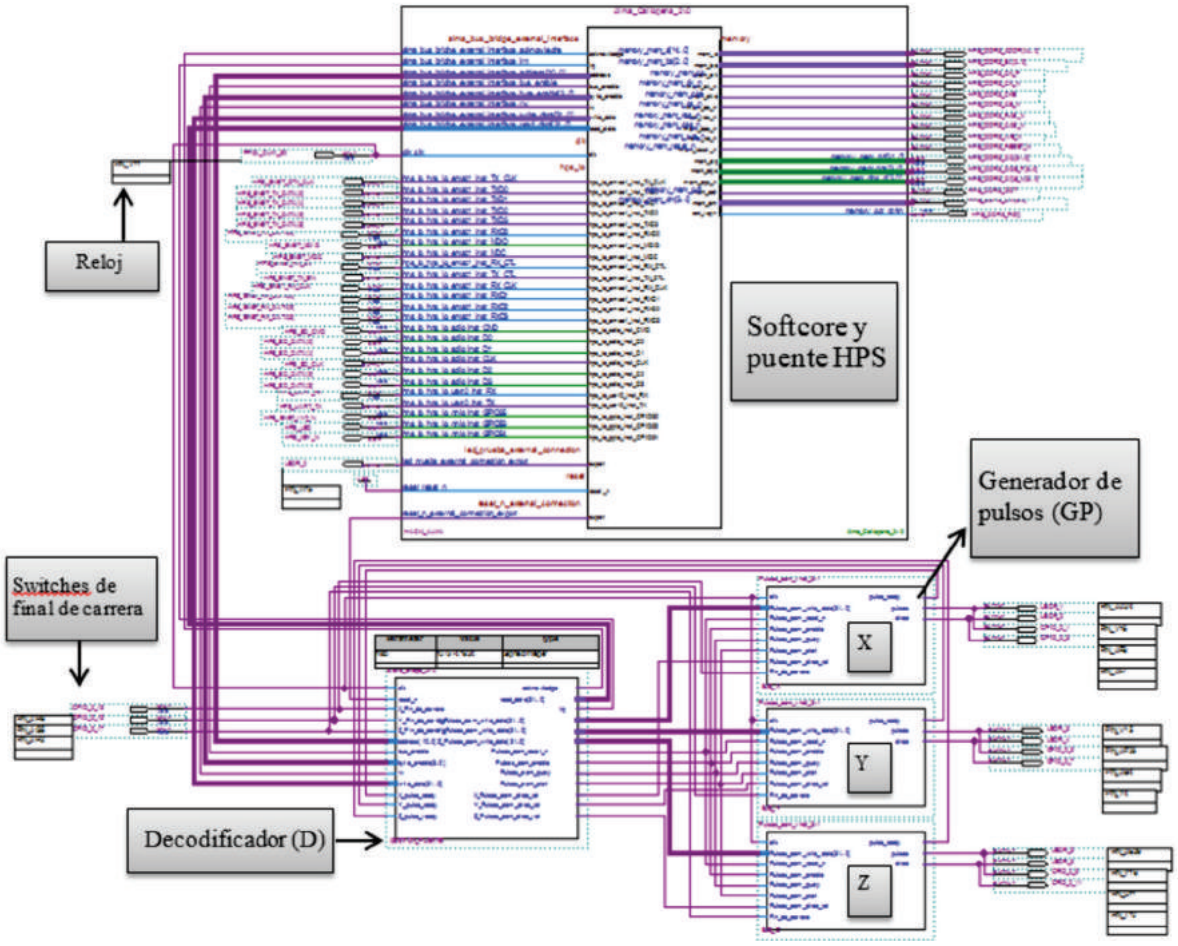


Figura 5. Diagrama de bloques del hardware implementado en el FPGA, mediante el software Quartus Prime 16.0.

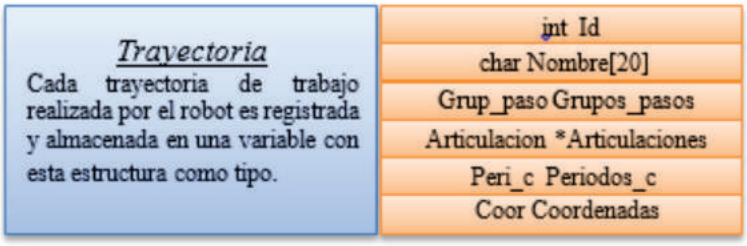


Figura 6. Estructura en lenguaje C «Trayectoria» y sus miembros con sus respectivos tipos de datos.

- *acknoweledge:* pin de salida requerido por las configuraciones del HPS en el Qsys para ser utilizado como bandera de recepción correcta/incorrecta de bits.
- *read_data[31..0]:* envía un dato de 31 bits al HPS desde el FPGA.
- *irq:* pin de interrupción del HPS generado desde el FPGA.
- *X_Pulsos_com_write_data[31..0]:* valor del conteo para el período hacia el bloque GP del MPP X.
- *Y_Pulsos_com_write_data[31..]:* … Y.
- *Z_Pulsos_com_write_data[31..]:* … Z.
- *Pulsos_com_reset_n:* resetea los bloques generadores de pulsos.
- *Pulsos_com_query:* solicita el aviso de culminación de generación de pulso a los bloques GP.
- *Pulsos_com_start:* indica a los bloques GPs que se empiece la lectura del nuevo valor de conteo para el nuevo pulso a generar.
- *X_Pulsos_com_direc_rot:* indica al GP del MPP X la dirección en la que debe girar el motor.
- *Y_Pulsos_com_direc_rot:* para el MPP Y.
- *Z_Pulsos_com_direc_rot:* para el MPP Z.

F. Programación del HPS

El Hard Processor System (HPS), es el hardcore del De0_Nano_SoC, permite programar el DPU de la RCU del robot. Aquí se procesa las tramas que llegan del GUI (interface de usuario). Se extraen las coordenadas, órdenes, solicitudes previstas por el usuario, y las convierte en un formato entendible por el RCU. Aquí también se interpolan las coordenadas y se calculan las velocidades angulares para los MPP mediante el software *Eclipse for* DS-5 v5.23.1 y el lenguaje C. Cabe resaltar que en el presente trabajo se ha tenido la necesidad de instalar el sistema operativo Ubuntu 14.04.4, en el HPS, dado que el trabajo con el procesador de dos núcleos es un tipo de aplicación llamada *Bare-metal,* que requiere tener determinadas licencias. Aunque al usar el sistema operativo mencionado se reduce la velocidad de procesamiento pero se gana estabilidad y seguridad en la aplicación resultante (Kashani y Beuchat, 2015).

La programación del RCU se hace mediante estructuras que permiten un eficiente dinamismo para la relación entre variables. En la Fig. 6 se puede observar la estructura «Trayectoria» que a su vez comprende, otras estructuras y un puntero hacia una estructura mayor. La variable Id es el número de identificación de la trayectoria; *Nombre* es el nombre de la trayectoria; *Grupos_pasos,* la variable del tipo «Grup_paso» donde se almacenan los grupos de paso para los MPP de los ejes X,Y,Z; *Articulaciones, es el puntero del tipo Articulación que modifica a las variables del tipo «Articulación», creadas en la función «main» (ver fig. 7) para almacenar los valores relevantes (nombre de eje, velocidad angular máxima, aceleración inicial del eje, posición máxima, mínima y actual, etc.) de cada articulación de los ejes X, Y, Z; periodos_c, la variable del tipo «Peri_c» donde se almacenan los valores del conteo del período de los pulsos para cada MPP; y *coordenadas,* variable del tipo «Coor» donde se guardan las coordenadas cartesianas en píxeles extraídas de la trama enviada por la PC.

El programa principal (fig. 7) está basado en un procedimiento de cuatro pasos. El primero enlaza los bloques del FPGA y los recursos del HPS con el programa en curso. El segundo estructura toda la información relacionada a las tres articulaciones del robot. El cuarto paso es la operación inversa del primer paso. El tercer paso es donde el servidor inicializa y queda a la espera de «clientes», o sea, usuarios del robot.

Cuando el usuario manda las coordenadas de los movimientos de trabajo del robot a través del GUI, el RCU lo recibe en forma de trama:

[T/C/S][f/p/c/e/i/q/e]-[número_coordenadas:n]-[coordenada1];[coordenada 2];…;[coordenada n][Eje que corresponde a la siguiente trama].

```
Función main()
Inicio
  Inicializar elementos del FPGA y el HPS
  Construir las articulaciones
  Inicializar conexión Cliente-Servidor
  Finalizar elementos del FPGA y el HPS
Fin
```

Figura 7. Pseudocódigo de la Función main( ).

P. ej: «Tf-2-0;3;Y» indica una traslación para la realización de una figura de 2 coordenadas en X (0 y 3), siendo la siguiente trama a enviarse la relacionada al eje Y. «C» es para la configuración de variables, como pueden ser las relacionadas a la cinemática. Y «S» para las relacionadas a solicitudes. En el caso de «p», es solo para la traslación de una coordenada a otra. El resto de etiquetas puede observarse en la Fig. 8.

```
Función Robot(Char RxTrama[])
Inicio
 Switch(RxTrama[0])
  Case «T»:
   Switch(RxTrama[1])
    Case «f»:
     Cambiar estado del robot a 1, ocupado.
     Fragmentar la trama
     Decodificar fragmentos de la trama
     Almacenar coordenadas en un arreglo
     Crear la variable trayectoria
     Generar los pulsos para la trayectoria
     Actualizar valor de las posiciones articulares
     Cambiar estado del robot a 0, desocupado.
    Case «p»:
     Cambiar estado del robot a 1, ocupado.
     Fragmentar la trama
     Decodificar fragmentos de la trama
     Establecer punto intermedio entre coordenada inicial y final
     Almacenar coordenadas en un arreglo
     Crear la variable trayectoria
     Generar los pulsos para la trayectoria
     Actualizar valor de las posiciones articulares
     Cambiar estado del robot a 0, desocupado.
    Case «c»:
     Enviar pulso para el movimiento por paso
  Case «C»:
   Switch(RxTrama[1])
    Case «e»:
     Configurar las variables za y he
  Case «S»:
   Switch(RxTrama[1])
    Case «i»:
     Cambiar estado del robot a 1, ocupado.
     Mover el robot a su posición
     Cambiar estado del robot a 0, desocupado.
    Case «q»:
     Enviar valores articulares actuales a la PC
    Case «e»:
     Enviar estado del robot a la PC
 Retornar Errores
Fin
```

Figura 8. Pseudocódigo de la función Robot (char RxTrama[]).

La generación de los pulsos para el movimiento de los MPP se realiza mediante la función «genera_ pulsos» (fig. 9). Esta función tiene una secuencia operativa parecida a los cinco pasos necesarios para la simulación de las trayectorias, expuestos en el apartado D. La diferencia radica en el primer paso, que en este caso, como las coordenadas (en píxeles) ya han sido extraídas, solo se convierten las unidades de las coordenadas a «pasos»; unidades entendidas por la RCU del robot, pues ésta representa el mínimo movimiento de la articulación, es decir, internamente el espacio cartesiano del robot está basado en «pasos». La otra diferencia es en el cuarto paso. Los grupos de cada articulación (M.Grupos_pasos.q1[], grupos para la articulación del eje X, según el pseudocódigo de la fig. 9) contienen una cantidad de pasos de MPP (1 o 2) a realizarse; entonces dado que no todas las articulaciones harán la misma cantidad de pasos (unas tendrán que moverse más que las otras), se las organiza en una misma cantidad de grupos. P. ej., si la articulación q1 tiene que dar 10 pasos y la q2, 5, se puede crear 5 grupos en el que la q1 realice 2 pasos por grupo, y la q2 solo 1. Esta es la forma por la que se opta para sincronizar los movimientos de las articulaciones. Para la generación de los grupos, el interpolador cúbico permite hacerlo mediante su algoritmia, tomando como referencia la articulación que se mas moverá. Si q1 se moverá 10 pasos, continuando con el ejemplo anterior, puede que lo haga en «n» segundos, que son los que usará el interpolador para interpolar también los valores de la q2.

```
Función Genera_pulsos(Trayectoria *M)
Inicio
 M.Coordenadas.pa_X[tot_coors] ← Píxel_a_pasos(M.Coordenadas.pi_X)
 M.*Articulaciones.q_pasos[tot_coors] ← Cinemática_inversa(M.Coordenadas.pasos_X[tot_coors])
 M.*Articulaciones.q_interpolado[tot_q_interp] ←
Interpolador_cúbico(M.*Articulaciones.q_pasos[tot_coors])
 Tot_pasos ← 0
 Para i←0 hasta tot_q_interp – 1
 inicio
  Crear grupos de pasos:
  M.Grupos_pasos.q1[i] ← abs(M.*Articulaciones.q_interpolado[i] – M.*Articulaciones.q_interpolado[i+1])
  Tot_pasos ← Tot_pasos + M.Grupos_pasos.q1[i]
 fin
 Tot_grupos ← i + 1
 M.Periodos_c.q1[0] ← Calcular_conteo_de_período(Tot_pasos,0)
 Bandera_calc_cont ← 0
 i ← 0
 j ← 0
 Mientras i < Tot_grupos hacer
 inicio
  Reiniciar los bloques del FPGA
  Habilitar los bloques GP del FPGA
  Avisar al bloque D del FPGA que el conteo será enviado
  Enviar_conteo_de_período (M.Periodos_c.q1[j],M.Grupos_pasos.q1[i])
  Reiniciar los bloques GP del FPGA
  Cambiar a modo lectura el bloque D del FPGA
  Mientras ¿el pulso aún sigue generándose? hacer
  inicio
   Si Not Bandera_calc_cont entonces
   inicio
    M.Periodos_c.q1[j+1] ← Calcular_conteo_de_período(Tot_pasos,j)
    j ← j + 1
    Bandera_calc_cont ← 1
   fin
  fin
  Bandera_calc_cont ← 0
  i ← i + 1
 fin
Fin
```

Figura 9. Pseudocódigo de la función genera_pulsos(Trayectoria *M), para la articulación del eje X.

Finalmente el quinto paso, calcula las velocidades angulares de los MPP (ver, Fig. 9).

La comunicación con el FPGA se realiza almacenando valores de 32 bits en una posición específica de memoria del FPGA mediante una librería otorgada por Intel, que luego de almacenar, permite generar una interrupción con el IP Core «hps_0» dirigida al IP Core *Avalon to external bus bridge* (Fig. 4).

G. Desarrollo de la GUI

Mediante la utilización del IDE *(Integrated Development Enviroment)* del software Visual Studio .Net, se implementa la Interfaz Gráfica de Usuario (GUI). En la Fig. 10, se observa el GUI del robot, donde se muestra algunas funciones como la cinemática del robot, TCP/IP, los valores de posición de cada articulación, y botones como «Paso» para generar un solo paso a modo de prueba de las articulaciones del robot, «P-Inicio» para llevar a su posición inicial al robot, entre otros.

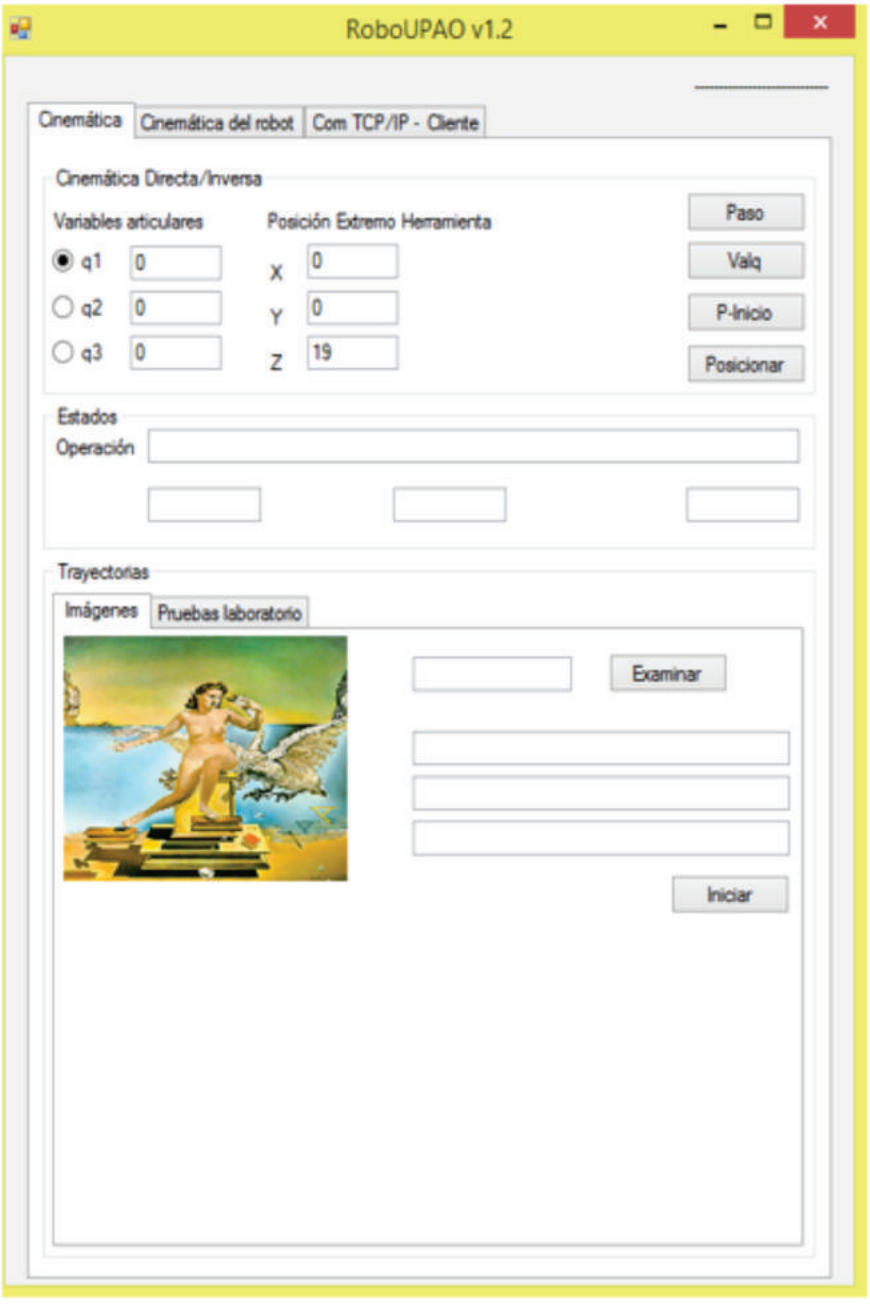


Figura 10. Formulario principal de la interfaz gráfica de usuario del RoboUPAO v1.2

En la GUI se utiliza un ActiveX llamado «Interop.MLApp», que es una librería de tipo servidor para automatizar procedimientos de Matlab. De esta forma, se puede llamar a los scripts relacionados al procesamiento digital de imágenes. Se combina la velocidad de procesamiento de Matlab con la eficiencia del manejo de formularios del Visual Studio .net.

H. Ejecución del robot

Para poner en marcha al robot se debe seguir una serie de pasos que involucran la transferencia del programa a las dos partes de la De0_Nano_SoC. Primero se inicializa el sistema operativo (fig. 11), se ejecuta el software *Eclipse for* DS-5 v5.23.1, y se activa la Terminal. Luego se energiza la tarjeta, y se ingresa el nombre de usuario y la contraseña, el comando «ipconfig eth0|grep config» muestra el IP de la tarjeta que a su vez representa el IP del servidor, que permitirá ejecutar el protocolo SSH *(Secure Shell)* a usarse para la conexión remota con el Ubuntu de la tarjeta que permite ver en la pantalla de la computadora personal, la ejecución del RCU del robot haciendo uso de la red LAN. Por ello, los pasos segundo y tercero son para ingresar la IP en el programa Ssh Shells del Eclipse.

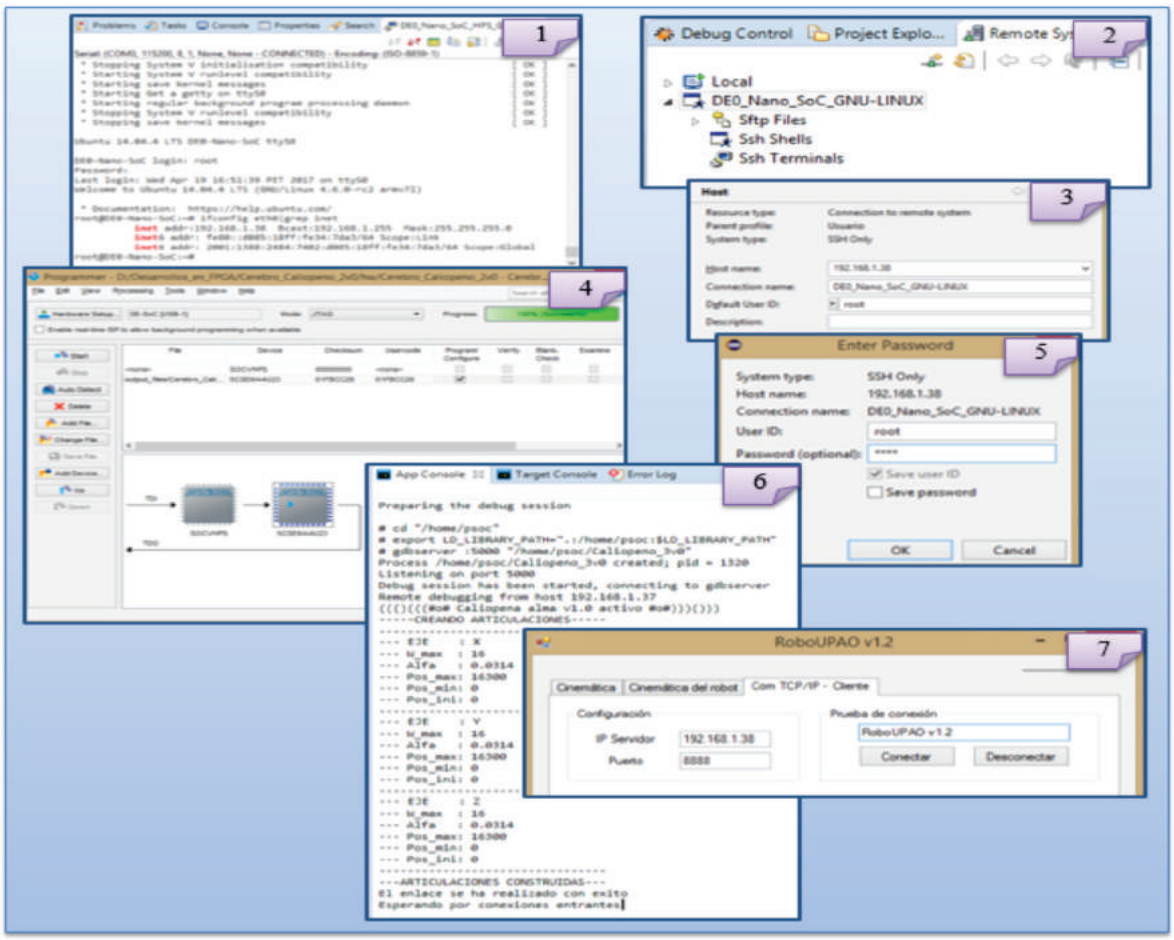


Figura 11. Secuencia de pasos para poner en marcha al Robot UPAO v1.2.

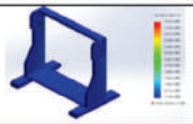
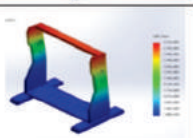
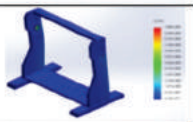
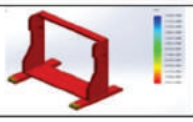
Hasta ahora, solo se ha configurado el sistema no se ha ejecutado ningún programa. El cuarto paso sirve para cargar el *SRAM Object File* (.sof), archivo que contiene la información del sistema programado para el FPGA. Mediante el software Quartus Prime 16.0 Lite Edition, se realiza dicha operación.

El quinto paso es la ejecución del programa del RCU mediante el software Eclipse. Estos recursos no son sino los periféricos. Finalmente, haciendo uso del GUI, en la pestaña de «Com TCP/IP - Cliente», se hace una prueba de conexión presionando el botón «Conectar».

III. RESULTADOS

Los resultados del análisis estructural del robot pueden observarse en la tabla III.

Tabla III. Análisis estructural.

Estudio	Estructura	Valor resultante
Tensión de Von Mises		Soporta como máximo 781327 N/m^2
Desplazamiento		Desplazamiento máximo de 0.00233276 mm
Deformación unitaria equivalente		Deformación máxima de 1.00402e-005
Factor de seguridad		El factor de seguridad mínimo es de 207.704

Fuente: elaboración propia.

Dado que el primer eje (fig. 12a) es independiente de los dos siguientes (fig. 12b), entonces la cinemática se divide en dos subsistemas.

Las Ec. (12) al (17) muestran las matrices homogéneas (4*4) de los dos subsistemas.

$$T_1 = {}^{0}_{\square}S_1 = \begin{bmatrix} 1 & 0 & 0 & q_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$${}^{1}_{\square}S_2 = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & q_2 \\ 0 & 0 & 1 & za \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

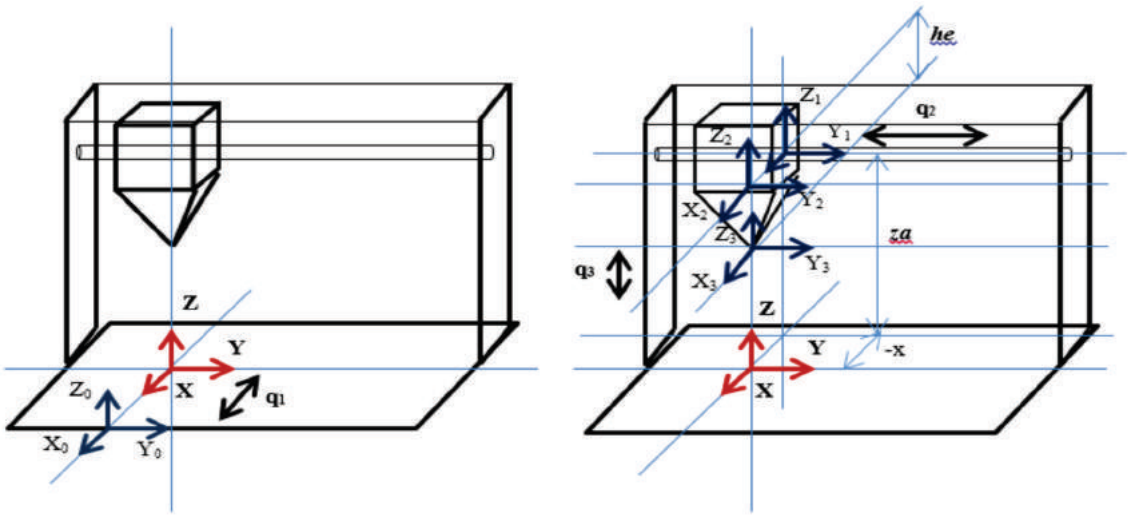


Figura 12. Sistemas de referencias cartesianas de los elementos del robot y variables estructurales básicas: za y he.

$$ {}^{2}S_3 = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (14) $$

$$ {}^{3}S_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -(he + q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (15) $$

$$ T_2 = {}^{1}S_2 \times {}^{2}S_3 \times {}^{3}S_4 \quad (16) $$

$$ T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & q_2 \\ 0 & 0 & 1 & za - (he + q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (17) $$

A partir de las matrices T1 y T2 se obtiene el resultado tanto de la cinemática directa como la inversa. Px, Py y Pz vienen a ser las coordenadas.

$$ P_x = q_1 \quad (18) $$

$$ P_y = q_2 \quad (19) $$

$$ P_z = za - (he + q_3) \quad (20) $$

$$ q_1 = P_x \quad (21) $$

$$ q_2 = P_y \quad (22) $$

$$ q_3 = za - (he + P_z) \quad (23) $$

Con relación al desempeño del robot, primero se obtiene el resultado de una evaluación, haciendo uso de un osciloscopio, de las capacidades de la FPGA para generar pulsos (tabla IV).

Tabla IV. Error porcentural del período de los pulsos generados por el FGPA.

Muestra	Período deseado (s)	Período medido (s)	e%
1	1.00000	0.99800	0.20
2	0.50000	0.49900	0.20
3	0.10000	0.09956	0.44
4	0.01000	0.01012	1.20
5	0.00100	0.00101	1.00
6	0.00050	0.00052	4.00
7	0.00020	0.00024	20.00
8	0.00014	0.00016	14.29
9	0.00006	0.00014	133.3

Fuente: elaboración propia.

En la fig. 13 se muestra la interpolación de dos puntos con unidades en «Pasos», que representa el movimiento mínimo de traslación, que en milímetros es 1.6 mm. «Spasos» significa «Segundos-pasos», y ello permite saber cuántos pasos ha tomado todo el movimiento, pero no da el tiempo que conllevará realizarlos. Ese tiempo se obtiene al calcular las velocidades angulares de paso de los MPP. En la Fig. 14, se observa una gráfica de las velocidades calculadas por el algoritmo Austin (2004).

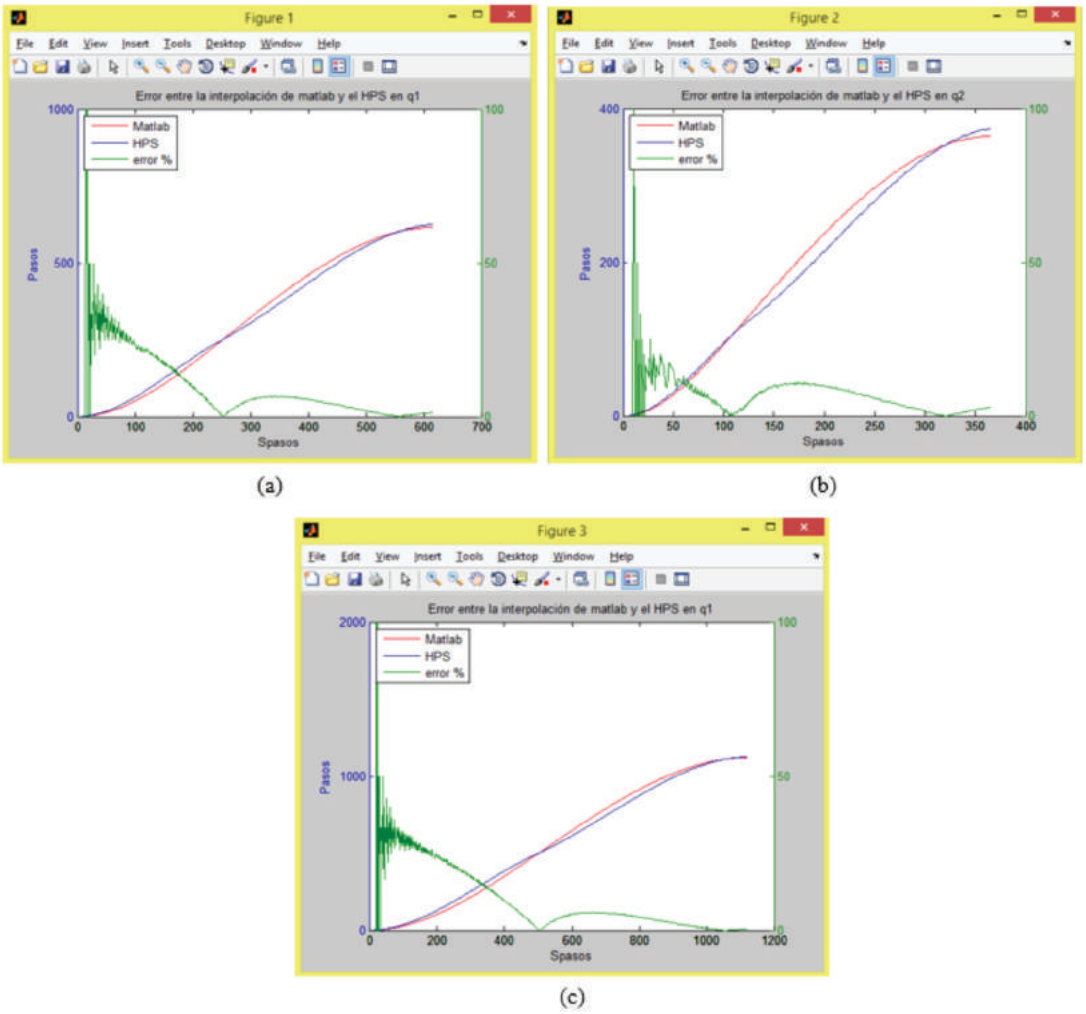


Figura 13. Error porcentual de los valores de la interpolación hecha por Matlab y el HPS para la: (a) articulación q_1, (b) articulación q_2 y (c) de la articulación q_3.

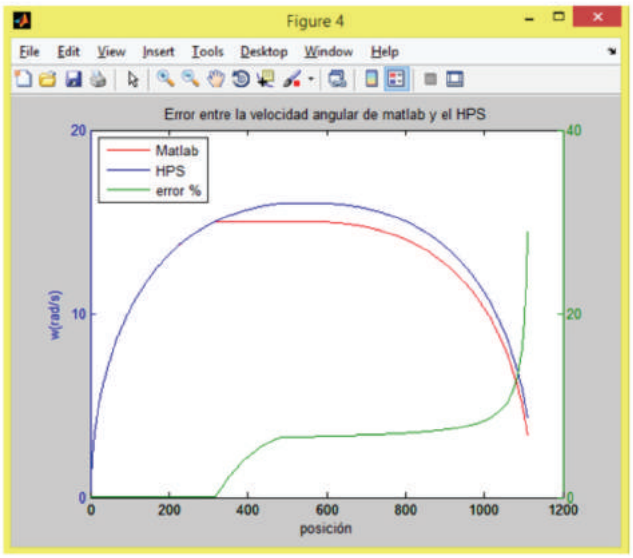


Figura 14. Error porcentual entre la velocidad angular originada por Matlab y la originada por el HPS.

Posteriormente se realiza pruebas con el robot, calculando la exactitud y repetibilidad (Groover, 2007), mediante las siguientes Ec:

$$Exactitud = 0.5RC + 3\sigma \quad (24)$$

$$RC = Máx\left(\frac{p}{n_s}, \frac{L}{2^B - 1}\right) \quad (25)$$

RC: resolución de control dado por el máximo valor entre la resolución mecánica de control y la resolución del computador. Por lo general la máxima es la primera.
σ: desviación estándar de la distribución del error (mm).
p = 1.6: paso del husillo guía (mm/rev).
ns = 200: cantidad de pasos/rev.
L = 120: rango del eje (mm).
B = 32: número de bits en el registro de almacenamiento para el eje.

En la fig. 15 muestra tres conjuntos de trazos realizados con el robot (2,6 y 20 mm respectivamente), obteniéndose los resultados de exactitud y repetibilidad mostrados en la tabla V.

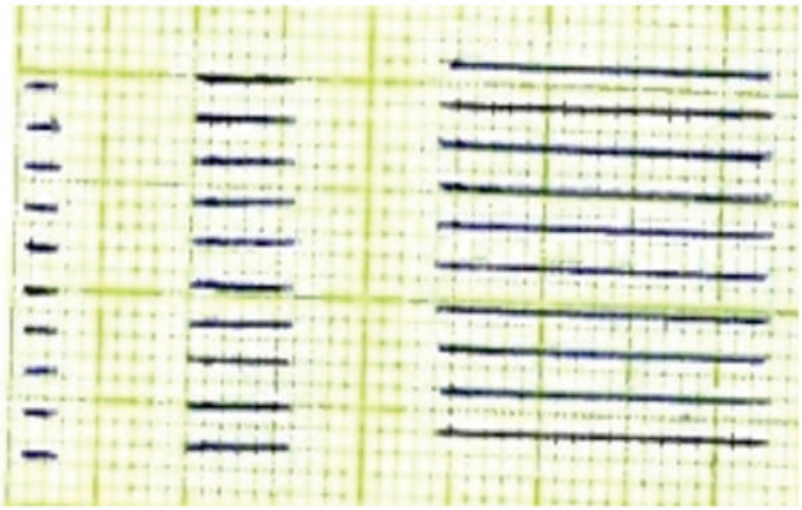


Figura 15. Líneas dibujadas por el Robot Cartesiano.

Tabla V. Repetibilidad y exactitud.

Longitud (mm)	Repetibilidad (mm)	Exactitud (mm)
2	± 0.687	0.691
6	± 0.671	0.675
20	± 0.671	0.674

Fuente: elaboración propia.

Y finalmente, otra forma de medir el error en los trabajos del robot fue mediante el área. Para ello se tienen tres figuras geométricas primarias: Círculo, rectángulo y triángulo. Hay dos tipos de errores relativos aquí. Uno entre el área de la figura de control, o deseada, la cual se toma directamente de la figura dibujada, y el área de la figura luego de haber sido sometida al procesamiento digital de imágenes. Y el otro error es entre el área de esta última y la de la figura real dibujada por el robot (Fig. 16).

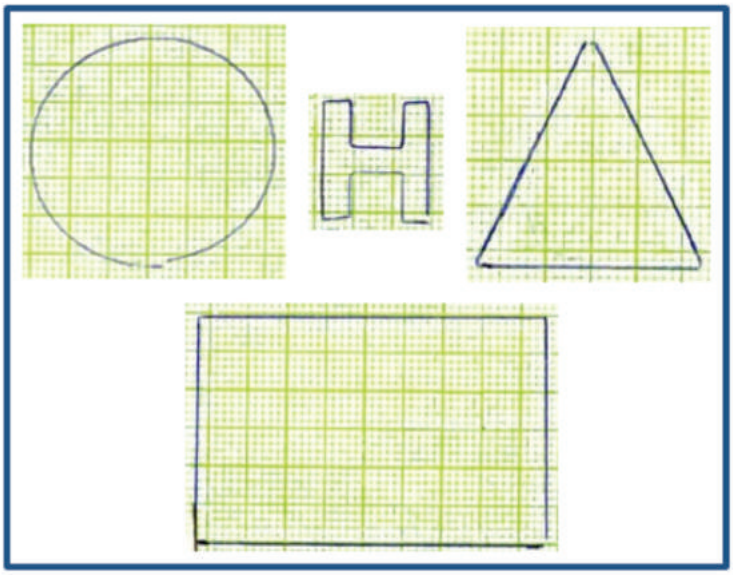


Figura 16. Figuras hechas por el Robot Cartesiano.

IV. DISCUSIÓN

El análisis estructural del robot cartesiano muestra gran estabilidad según los resultados que se muestra en la tabla III. Donde se puede observar un factor de seguridad de 207.7, un valor muy por encima de 1, lo cual quiere decir que se pueden optimizar las partes de la estructura resaltadas de rojo, según se ve en la tabla III.

La cinemática tanto directa como inversa, se obtiene usando MTH, y considerando la poca complejidad de la estructura mecánica del robot, es de fácil cálculo en la RCU. Adicional a las variables cinemáticas se usa dos variables relevantes, *za* y *he.* La primera representa la distancia desde la base de trabajo hasta el eje Y. La segunda es igual a la sustracción entre za y la distancia desde la base de trabajo del robot hasta el extremo de la herramienta, tal y como se muestra en la Fig. 12. Una vez conocidas los valores de *he* y *za,* la cinemática queda rápidamente resuelta para cualquier tipo de estructura basada en este modelo de robot cartesiano, con el eje X independiente de los dos restantes.

Por otro lado, El FPGA usa para la generación de pulsos eléctricos 500 microsegundos, como período mínimo y al aplicar la Ec. (8), representa una velocidad angular máxima de 62.8 rad/s. Sin embargo, habría que realizar una evaluación más rigurosa de la velocidad angular que soportan los MPP, dado que en este trabajo se observa que a partir de los 20 rad/s, ya empezaban a perderse pulsos de forma paulatina.

En el cálculo de la interpolación y generación de velocidades angulares de paso (en el RCU, ver fig. 13 y 14), se observa la presencia de errores oscilantes debido a un problema de redondeo en el lenguaje C. Una propuesta de solución a estos errores sería evaluar mejoras o compensaciones con respecto al redondeo, de tal forma que se consigan una mayor similitud entre lo obtenido por el Matlab y el HPS.

Con respecto al desempeño del robot, primero se calcula la exactitud y la repetibilidad del mismo utilizando la Ec. (24) y el trazado de líneas de 2, 6 y 20 mm. (Fig. 15). La exactitud presenta un valor promedio de 0.68, siendo éste el error más alto de repetibilidad más la mitad de la resolución mecánica de control, que en este caso, teniendo en cuenta el valor de p y ns de la ec. (25), es 0.008 mm/pulso, es decir, un valor bastante reducido que hace referencia al mínimo movimiento ideal que puede hacer el robot al moverse. Sin embargo, más relacionado a errores mecánicos (como la deflexión de la herramienta, que en este caso es un lapicero, o la pérdida de pulsos de los MPP) la repetibilidad con un valor promedio de ±0.68 mm representa el intervalo de error en el que el robot va a posicionar el extremo de su herramienta todas las veces que tenga que volver a un punto específico.

En la Fig. 16 se muestran una serie de dibujos realizados con el robot, teniendo en cuenta los valores de exactitud y repetibilidad con los que cuenta el dispositivo.

V. CONCLUSIONES

Se desarrolló una metodología para controlar los movimientos de un robot industrial usando un módulo FPGA De0_Nano_SoC de la empresa Altera, una PC y un robot industrial cartesiano teniendo como base física la estructura mecánica de una máquina CNC de bajo coste. Después de aplicar la metodología se obtiene un comportamiento del robot cartesiano con unas características técnicas de exactitud igual a 0.68 mm y

repetibilidad igual a ±0.68 mm, que permitieron la realización de diferentes figuras geométricas tal y como se muestra en la Fig. 16.

Por otro lado, el robot cartesiano propuesto en este trabajo se basó en una estructura mecánica de un CNC de bajo coste, modificando enteramente la electrónica y el software de trabajo y puede agregársele más grados de libertad al acoplarle la herramienta de trabajo correspondiente. Cabe destacar que las máquinas de configuración cartesianas han demostrado gran eficiencia en aplicaciones convencionales industriales como corte, seguimiento o trazado de figuras en un plano, coger y dejar, paletizado, manipulación, soldadura.

Se realizó un análisis estructural mecánico, modelamiento matemático, simulación, el diseño del hardware y la programación del software para el RCU y el desarrollo de una interfaz gráfica de usuario, que haga viable la integración hombre-robot.

Finalmente se realizaron las pruebas y publicación de los resultados para su respectivo análisis.

VI. REFERENCIAS BIBLIOGRÁFICAS

Austin, D. (2004). *Generate Stepper-Motor Speed Profiles In Real Time.* Url: http://www.embedded.com/design/mcus-processors-and-socs/4006438/Generate-stepper-motor-speed-profiles-in-real-time

Parnell K., Bryner R. (2004). *Comparing and Contrasting FPGA and Microprocessor System Design and Development.* Url: http://www.xilinx.com/support/documentation/white_papers/wp213.pdf

Altera. (2007). *FPGA vs. DSP Desgin Reliability and Maintenance.* Url: https://www.altera.com/en_US/pdfs/literature/wp/wp-01023.pdf

Altera. (2014). *What is an SoC FPGA?.* Url: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ab/ab1_soc_fpga.pdf

Barrientos, A., Peñín, L. F., Balaguer, C., Aracil, R. (2007). *Fundamentos de Robótica.* (2ª ed.) Madrid, España: McGraw-Hill/Interamericana de España, S.A.U.

Dassault Systemes. (2014). *Análisis Estático Lineal.* Url: http://help.solidworks.com/2014/Spanish/SolidWorks/cworks/c_Linear_Static_Analysis.html

DuffuD. (2013). *Calcular cuánto mide un monitor.* Url: http://www.tuxylinux.com/calcular-cuanto-mide-un-monitor/

Fiestas, E. M., Figueroa, B., Prado, S. R. (2013). *Modelado Cinemático y Dinámico de un Robot Manipulador de 6 Grados de Libertad para Aplicaciones Industriales en la Región La Libertad.* Memorias del Congreso Internacional de Ingeniería Electrónica, Eléctrica y Computación, 98-63.

Fiestas, E. M., Castillo, E. y Briones, W. (2013). *Modelado y Simulación de un Manipulador de Seis Grados de Libertad para Aplicaciones Industriales en la Región La Libertad.* Revista Pueblo Continente, 24(1), 15-27.

Groover, M. (2007). *Robótica.* (3ª ed.) México D.F.: McGraw-Hill/Interamericana Editores S.A de C.V

Kashani, S., Beuchat, R. (2015). SoC-FPGA *Design Guide.* Url: http ://moodle.epfl.ch/mod/resource/view.php?id=924078

Kordik, J. (2015). *Measuring Power Dissiparion in Step Motors and Drivers.* Url: https://www.applied-motion.com/news/2015/10/measuring-power-dissipation-step-motors-and-drives

National Instruments. (2011). *Introducción a la Tecnología FPGA: Los Cinco Beneficios Principales.* Url: http://www.ni.com/white-paper/6984/es/

International Federation of Robotics, (2015). *Industrial Robot Statistics.* Url: http://www.ifr.org/industrial-robots/statistics/