CÁLCULO DEL CÓDIGO DE VERIFICACIÓN CRC EN PROTOCOLO DE COMUNICACIÓN DYNAMIXEL 2.0 HACIENDO USO DE FPGA PARA EL ACCIONAMIENTO PARALELO DE MOTORES DE LA SERIE XL

Octavio Sánchez García

Instituto Politécnico Nacional/Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada

osanchez0112@gmail.com

Moisés Vicente Márquez Olivera

Instituto Politécnico Nacional/Centro de Investigación e Innovación Tecnológica

mvmarquez@ipn.mx

Viridiana G. Hernández Herrera

Instituto Politécnico Nacional/Centro de Investigación e Innovación Tecnológica

vhernandezhe@ipn.mx

Antonio Gustavo Juárez Gracia

Instituto Politécnico Nacional/Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada

cicata81@yahoo.es

Resumen

Este artículo describe el diseño e implementación en lógica de compuertas reprogramables *FPGA* (Field Programmable Gate Arrays), la comunicación entre la interfaz de usuario y los motores de la serie Dynamixel modelo XL-320, con la finalidad de accionar en arquitectura paralela a tantos motores como se requiera. Con el fin de comprobar que la comunicación se realizó con éxito se utiliza el código de verificación de redundancia cíclica CRC (Cyclic Redundancy Check), el cual de acuerdo con el protocolo de comunicación Dynamixel 2.0 es dinámico y requiere ser calculado para cada uno de los paquetes de instrucción a enviar al

motor, para este cálculo se diseñó e implementó una máquina de estados embebida en FPGA, capaz de determinar el CRC paralelamente para dos o más motores en un mismo tiempo de ejecución.

Palabra(s) Clave: CRC, Dynamixel, FPGA, ROBOTIS.

Abstract

This article describes the design and implementation on FPGA (Field Programmable Gate Arrays), the communication between the user interface and the Dynamixel series XL-320 series motors, to operate in parallel architecture to many motors as required. In order to verify that the communication was successful, the Cyclic Redundancy Check (CRC) is used, which according to Dynamixel 2.0 communication protocol is dynamic and requires calculation for each of the instruction packets to be sent to the engine, for this calculation a state machine embedded in FPGA was designed and implemented, able of determining the CRC in parallel for two or more motors at the same execution time.

Keywords: *CRC, Dynamixel, FPGA, ROBOTIS*

1. Introducción

En la actualidad, la comunidad científica ha enfocado su interés en el desarrollo de robots móviles apoyados en la versatilidad de los sistemas mecatrónicos. En comparación con robots industriales, los robots móviles de servicio se desempeñan en ambientes no controlados, lo cual implica que en algunos casos deban desplazarse en entornos desconocidos, por tal motivo es necesario el diseño y desarrollo de algoritmos de control que tengan como objetivo imitar la locomoción biológica [1, 2, 3] de insectos, brazos, humanoides, entre otros. No obstante, debido a la complejidad de imitar este tipo de locomoción, es preciso un controlador que sea capaz de accionar eficientemente un conjunto de motores, cuyo número está relacionado con los grados de libertad del robot. Así mismo, el controlador debe sincronizar el accionamiento de los motores considerando la cinemática y trayectorias de desplazamiento necesarias para su movimiento en los diferentes entornos de navegación.

El uso de servomotores de alto desempeño como son los de la serie Dynamixel [4], incrementa la eficiencia de los robots móviles de servicio debido a que ofrecen un diseño compacto y modular, considerando las prestaciones de alto torque con respecto al tamaño del actuador, además, se tiene una caja de engranes, un motor de CD (corriente directa) de precisión, un controlador de velocidad, un controlador de posición y una conexión en red capaz de comunicar hasta 256 motores con una sola línea de comunicación que opera con comandos de acceso secuencial.

Entre los desarrollos realizados empleando este tipo de motores es posible encontrar el trabajo de Bautista et al. [5] que exponen la simplificación de sincronía de los motores en un robot móvil con cuatro extremidades al utilizar actuadores conectados en red de comunicación, al comparar el desempeño de servomotores de radio control contra los de la serie Dynamixel, estos últimos presentaron una mayor capacidad de torque durante la realización de los movimientos, asimismo la resolución del movimiento angular es altamente superior y la red de comunicación es confiable. Sánchez et al. [6] aprovechan la resolución del movimiento angular de estos motores, debido a que el desplazamiento angular mínimo permisible en el Dynamixel es de 0.088°; al usarlos como actuador de la resolución espacial de un brazo robótico, cuyos movimientos son determinados por medio de una red neuronal artificial.

Una de las desventajas que tienen los Dynamixel es que trabajan con base en la estructura de programación secuencial, debido a que la comunicación entre la interfaz gráfica (maestro) y los motores (esclavos) es realizada por medio del controlador proporcionado por el fabricante que se basa en microcontroladores ARM y ATMega, los cuales realizan sus operaciones secuencialmente, este inconveniente lo experimentan en su trabajo Melo et al. [7] que analizan los movimientos requeridos en un robot con forma de serpiente, haciendo notar que es inviable mantener procesos de locomoción continuos, observando la funcionalidad limitada de los motores debido al protocolo de comunicación secuencial, cabe mencionar que la desventaja secuencial puede solucionarse parcialmente con el método de interrupciones con prioridad paralela de los

microcontroladores. Buscando no depender del controlador del fabricante AL-Busaidi [8] emplea la plataforma de desarrollo Arduino y MATLAB, para crear una interfaz capaz de realizar un puente entre la PC y lo motores para aplicaciones educacionales, no obstante, el uso de este tipo de herramientas sigue siendo una limitante para accionar a los motores de forma paralela, por ende al activar más de dos motores en un mismo instante, siempre existirá un retraso acumulativo entre la activación de uno con respecto a los demás.

En este artículo se propone diseñar e implementar en lógica de compuertas reprogramables *FPGA* (Field Programmable Gate Arrays), la comunicación entre la interfaz de usuario y los motores de la serie Dynamixel modelo XL-320, con la finalidad de controlar paralelamente tantos motores como el robot requiera, para ello se diseñó e implementó una máquina de estados capaz de realizar el cálculo del código de verificación de redundancia cíclica CRC (Cyclic Redundancy Check) para cada uno de los paquetes de instrucción incluidos en el protocolo de comunicación Dynamixel 2.0.

2. Métodos

Para este trabajo se han seleccionado seis servomotores Dynamixel XL-320, con las características mostradas en la tabla 1.

Tabla 1 Resumen de características del motor Dynamixel XL – 320.

Peso	16.7 g
Resolución	0.29°
Relación de la caja de engranes	238:1
Torque	0.39 Nm
Velocidad máxima sin carga	114 rpm
Voltaje	6 – 8.4 V (7.4 V recomendado)
Corriente	0.6 A
Baudrate	7843 bps – 1 Mbps
Protocolo de comunicación	Serial asíncrono medio dúplex
Conexión en red	253 máximo

Cabe mencionar que los Dynamixel poseen una tabla de control contenida en una memoria RAM, a la cual es indispensable acceder para modificar los parámetros de configuración y operación que permiten accionar los motores, mediante

paquetes de instrucción enviados en serie siguiendo el estándar del protocolo de comunicación universal asíncrono de transmisión/recepción *UART* (Universal Asynchronous Receiver-Transmitter), con una sola línea de conexión. El modelo del motor estudiado en el presente trabajo, utiliza la estructura de paquetes de instrucción mostrada en la tabla 2 que pertenece al protocolo Dynamixel 2.0 del fabricante de estos motores, ROBOTIS.

Tabla 2 Paquete de instrucción del protocolo Dynamixel 2.0.

Encabezado			Reservado	ID motor	Tamaño de paquete		Instrucción	Dirección		Dato / Tamaño		CRC 16 bits	
0xFF	0xFF	0xFD	0x00	ID	Tamaño bajo	Tamaño alto	Número de Instrucción	Dir. bajo	Dir. alto	Dato bajo	Dato alto	CRC_L	CRC_H

Los datos a enviar en el paquete de instrucciones son de un byte de longitud, y son susceptibles a tener diferentes significados dependiendo de la posición en la que se encuentren los catorce datos dentro de la trama:

- Encabezado: Este campo, compuesto por tres bytes de valor estático, indica el inicio de un paquete de instrucción.
- Reservado: Valor para diferenciar el encabezado del identificador del motor.
- ID motor: Determina que motor recibirá y ejecutará la instrucción, el rango permisible de identificadores con este protocolo es de 253 motores.
- Tamaño de paquete: Compuesto por dos bytes, donde el primero representa a los bits menos significativos y el segundo a los más significativos, he indica la cantidad de datos dentro de la trama después del identificador de motor, sin considerar el CRC.
- Instrucción: Define el propósito del paquete, con doce posibles operaciones.
- Dirección: Estos dos bytes bajo y alto, definen el lugar de memoria sobre la cual se ejecutará la instrucción deseada.
- Dato o tamaño: Este campo en caso que la instrucción sea de escritura, indica el valor a modificar dentro de la tabla de control, no obstante, si la

instrucción es de lectura, señala la cantidad de bytes que se esperan como respuesta del motor.

- CRC 16 bits: El protocolo contempla dos bytes para el uso del código *CRC*, el cual es un método general para la detección de errores de comunicación, el cálculo de CRC se realiza con base en los datos a enviar dentro del paquete por lo que mantiene una dependencia con los mismos.

Algoritmo de cálculo para el CRC

Como se puede observar, el paquete de instrucción se compone principalmente de valores que serán otorgados por el usuario para el accionamiento de los Dynamixel, exceptuando el código CRC que debe ser calculado. El CRC es un código de verificación, fundamentado en operaciones de polinomios de coeficientes enteros base dos, teniendo como objetivo añadir *r* bits al final del mensaje de transmisión *T(x)*, de forma tal que *T(x)* sea divisible de forma exacta entre el polinomio generador predeterminado *G(x)* de grado *r*. Cabe mencionar que el polinomio G(x) es el mismo para el transmisor y el receptor. La validación del protocolo de comunicación es realizada por el receptor, el cual divide la trama recibida entre el polinomio *G(x)*, obligando a que el residuo sea nulo, de lo contrario es posible decir que no se recibieron los datos satisfactoriamente, además de que no es factible determinar cuál o cuáles de los datos del paquete llegaron erróneos.

En este caso de estudio en el que se emplea un motor Dynamixel del modelo XL-320, el protocolo de comunicación del receptor, requiere por especificación del fabricante que el cálculo del CRC sea a 16 bits con el estándar (IBM/ANSI) considerando que $G(x) = X^{16} + X^{15} + X^2 + 1$. Lo cual implica que es un polinomio con $r = 16$, en consecuencia, el CRC a enviar en el paquete mostrado en la tabla 2 es de 16 bits. Se debe destacar que el CRC debe ser calculado por cada byte del paquete de instrucciones, por tal motivo, en esta aplicación el código de verificación debe calcularse y acumularse byte a byte para finalmente obtener el *T(x)* + CRC de todo el paquete. La metodología para realizar el cálculo se muestra en la figura 1.

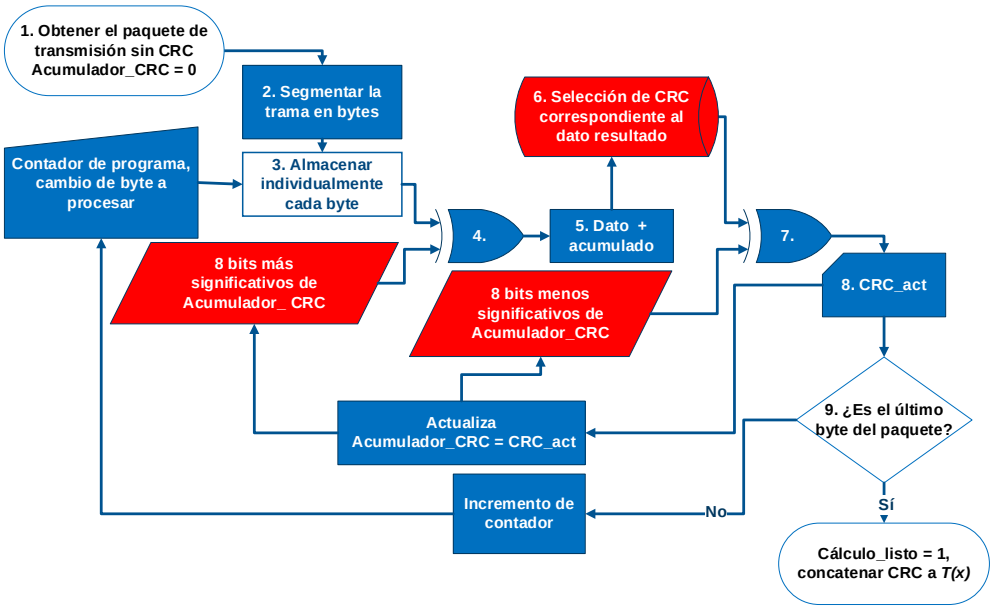

Figura 1 Metodología para el cálculo del CRC en protocolo 2.0 Dynamixel.

En el primer paso de esta metodología se debe garantizar que la trama *T(x)* de la instrucción a realizar este completa, excluyendo al CRC, además se implementa un espacio de memoria con el nombre como “Acumulador_CRC”, para registrar los cálculos acumulativos del código de verificación en cada dato de la trama, cabe mencionar que este registro debe ser inicializado con valor cero cada vez que se va a enviar una nueva instrucción. Validado lo anterior en el paso dos se debe segmentar el paquete de instrucción en bytes para en el paso tres almacenar independientemente a cada dato en un arreglo llamado *dat[i]* donde i = 12, y así aplicar el algoritmo de cálculo de CRC por cada dato como lo marca el algoritmo de cálculo, cabe destacar que este arreglo modifica su apuntador con base en un contador de programa que aumenta su valor en uno cada que se calcula un nuevo CRC y el *dat[i]* no ha llegado a su desborde. En el paso cuatro el acumulador de CRC debe ser añadido al dato en proceso de cálculo *dat[i]*, para que se pueda considerar que el código de verificación es de toda la trama y no de un dato en específico, esto es realizado por medio de una división binaria entre *dat[i]* y el byte menos significativo del “Acumulador_CRC”, resultando en “*dat_accum(x)*”, este dato es al que se le debe dividir entre el polinomio generador *G(x)* para determinar el nuevo código de verificación, lo cual obliga a realizar en ocho ocasiones la

operación *dat_accum(x)/G(x)*. Con el fin de evitar esta iteración de operaciones, en el paso seis se ha almacenado preliminarmente en una *tab[j]*, mostrada en el anexo 1, donde j = 256, la división de los polinomios *dat_accum(x)/G(x)* para todos los valores posibles de *dat_accum(x)*, que al ser de un byte de longitud se podrán encontrar valores de 0 hasta 255 únicamente, luego entonces para obtener el cálculo de CRC en este paso basta con observar el valor de *dat_accum(x)* y usarlo como apuntador en la *tab[j]* otorgando el resultado concurrentemente de *tab[dat_accum(x)]*.

En el paso siete como ya se tiene el CRC para el dato en proceso *dat_accum(x)* debe ahora actualizarse "Acumulador_CRC" añadiendo por medio de división polinómica tab*[dat_accum(x)]* al "Acumulador_CRC", esto es posible realizando un corrimiento de ocho bits al actual acumulador y dividiéndolo entre *tab[dat_accum(x)]*, obteniendo de esta manera el nuevo CRC acumulado. Por último, el paso nueve puede repetir todo el proceso o terminarlo al validar si el dato *dat[i]* es el último de la trama, en el caso de que sea el dato concluyente del paquete, se iguala a uno la señal de "Cálculo_listo" y el CRC acumulado es concatenado al final del paquete de instrucciones, caso contrario se incrementa al contador de programa y se vuelve a repetir la metodología.

Implementación del algoritmo de cálculo para el CRC en FPGA

Para implementar esta metodología en FPGA se hace uso de la tarjeta de desarrollo Atlys del fabricante XILINX, con las especificaciones mostradas en la tabla 3.

Tabla 3 Resumen de especificaciones de la tarjeta de desarrollo Atlys.

FPGA	Spartan – 6 LX45
Reloj	100 MHz
Entradas y salidas	22 básicas y 40 de alta velocidad
Memoria	128 Mb DDR2
Comunicación	UBS-UART

Se diseñó una máquina de estados, mostrada en la figura 2, para implementar el algoritmo de cálculo del CRC por medio de cuatro estados; en el estado S0, la

máquina de estados comienza a funcionar al recibir un evento del ciclo de reloj y en la entrada “Nuevo cálculo” el valor lógico uno, inicializando las variables “Cálculo_listo”, “contador_progr” y al “Acumulador_CRC”, además el valor del acumulador de CRC se asigna a dos variables auxiliares, “aux” y “aux1” con el fin de realizar las operaciones binarias sin riesgo a modificarlo. Cabe mencionar que en este estado se selecciona el dato *dat[i]*, de acuerdo con el contador de programa que se encarga de multiplexar en la secuencia correcta a los datos que están previamente registrados, obteniendo como resultado de este estado las variables auxiliares definidas y el *dat[i]* al cual se le calculará él CRC.

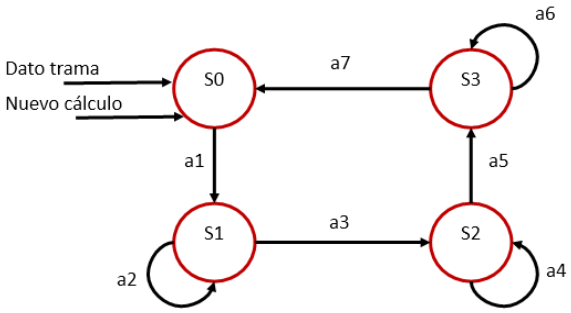

Figura 2 Diagrama de estados para el cálculo de CRC.

Haciendo uso de concatenación en cada evento de reloj, en el estado S1 se realiza corrimiento de 8 bits a la derecha a la variable aux, y a la izquierda a aux2, para obtener el byte más significativo y el menos significativo del acumulador de CRC, respectivamente, que serán utilizados en los siguientes estados. Entonces hasta este instante se tienen establecidos los valores con corrimiento del acumulador de CRC y el dato para proceso de cálculo, luego entonces se ingresa al estado S2, en el cual se divide entre el dato en proceso y el byte menos significativo del acumulador de CRC almacenado en aux para obtener como resultado de este estado *dat+crc(x)*.

En el siguiente evento de reloj se debe determinar el valor de CRC que le corresponde al valor obtenido *dat+crc(x),* por medio de la tabla *tab[]* almacenada en una secuencia de casos de VHDL, y se realiza la división entre el byte más significativo del acumulador, registrado en aux y *tab[dat+crc(x)]*, por último en el

siguiente pulso de reloj se registra el resultado de esta división en el acumulador de CRC y se incrementa el valor del contador de programa, para iniciar de nuevo el ciclo, hay que destacar que este ciclo es realizado siempre y cuando este contador sea menor a 12, con lo cual se garantiza solo se calculara para el tamaño exacto del paquete de instrucciones. En el caso de que el dato en proceso sea el último de la trama se activa la bandera de cálculo listo con el valor lógico uno.

3. Resultados

Como resultado se obtiene un esquema modular para el cálculo de CRC, mostrado en la figura 3, como se puede observar a la entrada están todos los datos de la trama previamente registrados, el reloj de 100MHZ y la señal de nuevo dato operando lógicamente con la señal de reinicio del sistema, ya que cualquiera de las dos condiciones puede iniciar el cálculo del CRC. Como resultado se obtiene el CRC acumulado dividido en bytes para ingresarlos directamente en el paquete de instrucción.

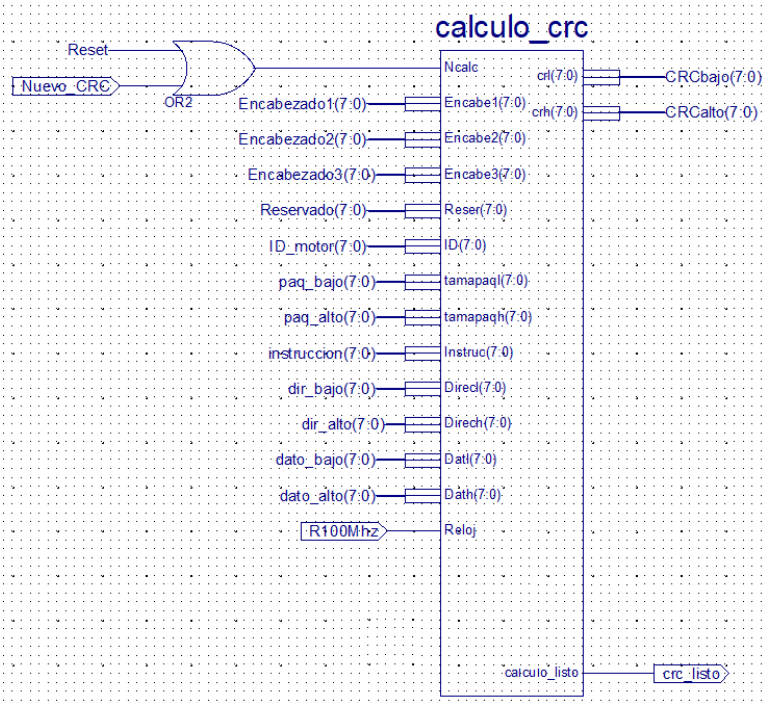

Figura 3 Módulo para el cálculo de CRC implementado en este trabajo.

Para demostrar que el módulo generado para el cálculo funciona, se realiza un experimento en la tarjeta de desarrollo Atlys, teniendo como datos del paquete de instrucción los mostrados en la tabla 4. Este paquete de instrucción modificará el valor del registro 0x19 al valor 0x07 que, con base en la hoja de especificaciones del fabricante, el registro 0x19 es el encargado de modificar el color del led de notificaciones del Dynamixel, como resultado se obtendrá el led de notificaciones encendido en color blanco.

Tabla 4 Paquete de instrucción para experimentación.

Encabezado			Reservado	ID motor	Tamaño de paquete		Instrucción	Dirección		Dato		CRC 16 bits	
0xFF	0xFF	0xFD	0x00	0x01	0x07	0x00	0x03	0x19	0x00	0x07	0x00	CRC_L	CRC_H

Una vez que estos valores fueron predefinidos se solicita al FPGA generar el cálculo del CRC para completar el paquete de instrucción, por medio de la señal “Nuevo_CRC”, obteniendo el paquete de instrucción mostrado en la figura 4, cabe mencionar que se realizó manualmente el cálculo del CRC para esta trama, con el fin de validar que el cálculo que realice el FPGA sea correcto, teniendo como resultado esperado CRC = 0x3B5C.

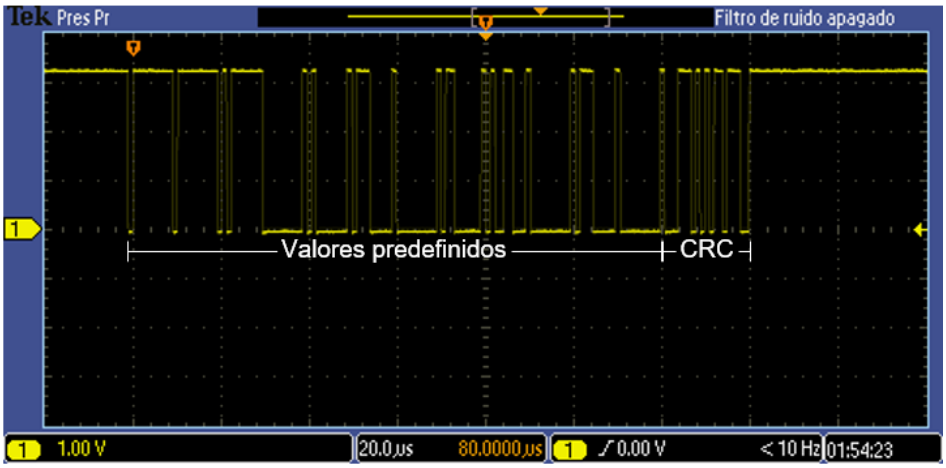

Figura 4 Paquete de instrucción obtenido después del cálculo del FPGA.

Al analizar el paquete de instrucción obtenido se observa que el cálculo fue correcto, obteniendo el valor de CRC = 0x3B5C, y el Dynamixel realizó la

operación solicitada cambiando el led de notificación a color blanco. Se realizó la medición del tiempo que tarda la máquina de estados implementada en FPGA en realizar el cálculo del CRC de la trama completa, demorando 1.69 µs.

Cabe mencionar que se realizó la experimentación con todo el set de operaciones del Dynamixel XL-320 obteniendo el CRC correspondiente para cada una y logrando que el motor realizará las operaciones solicitadas.

Como última prueba experimental y para validar que se pueden accionar paralelamente a los seis Dynamixel, se repitió el módulo de la figura 3 en seis ocasiones y se asignaron salidas individuales para cada motor, obteniendo que el cálculo de CRC fue realizado correctamente para los paquetes de instrucción de cada motor, sin alterar el tiempo de cálculo y logrando que él envió de la trama de instrucción sea paralelamente.

4. Discusión

Con este trabajo se demuestra que es posible accionar paralelamente tantos motores Dynamixel como sean requeridos por la aplicación donde se estén utilizando, descartando problemas de funcionalidad limitada en tiempo de ejecución debido al protocolo de comunicación, permitiendo de esta manera que al utilizarlos en aplicaciones de locomoción biológica, los movimientos sean más continuos en comparación con los trabajos que utilizan el controlador del fabricante para accionarlos.

El tiempo de cálculo de CRC en FPGA además de permitir comunicarse en estructura de programación paralela, permite realizar el cálculo mientras se envían los datos predeterminados del paquete de instrucción ya que en enviar los doce datos por el protocolo de comunicación UART el sistema tarda 192 µs, dando suficiente tiempo para que el cálculo de CRC que toma solo 1.69 µs sea realizado y concatenado al paquete de instrucción.

5. Bibliografía y Referencias

[1] Thai C.N. Exploring Robotics with ROBOTIS Systems, S.I. Publishing, Ed. Springer International Publishing, 2015.

[2] Kang X., Shen W., Chen W y Wang J. The control of Dynamixel RX-28 based on VC++ for the locomotion of cockroach robot, 4th IEEE Conference on Industrial Electronics, 2009.

[3] Rivas D., Alvarez M., Velasco J., Mamarandi J., Carrillo J.L., Bautista V., Galarza O. Reyes P., Erazo M., Pérez M. y Huerta M. BRACON: Control system for a robotic arm with 6 degrees of freedom for education systems, 6th International Conference on Automation, Robotics and Applications (ICARA), 2015.

[4] Ortega L.A., Salazar A., Tapia M.E., Velasco J., Santoyo M., Camarillo K.A., Hernández L.A., Hernández M., Pérez G.I. Open architecture controller for a 22-DOF humanoid robot, XVIII Congreso Mexicano de Robótica, 2016.

[5] Bautista R., Aguirre A., Ramos A., López R. Diseño, Construcción y Control de un Robot Cuadrúpedo con Actuadores de Alto Desempeño Conectados en Red, 10° Congreso Nacional de Mecatrónica, 2011.

[6] Sánchez A., Terán A., Ibarra A., Abatta L., Alulema D., Morocho D. y Encalada F. Desing and construction o fan anthropomorphic robotic arm of seven degrees of freedom with kinematic and Dynamic analysis basde on genetic algorithms, IEEE International Conference on Automatica (ICA-ACCA), 2016.

[7] Melo K., Leon J., Monsalve J., Fernandez V. y Gonzalez D. Simulation and control integrated framework for modular snake robots locomotion research, IEEE/SICE International Symposium on System Integration (SII), 2012

[8] Al-Busaidi A.M. Development o fan educational environment for online control of a biped robot using MATLAB and Arduino, 9th France-Japan 7th Europe-Asia Congress on and Research and Education in Mechatonics (REM), 2012

Anexo 1

Tabla de CRC posible para cada valor de dato

Con longitud de 8 bits

```
unsigned short crc_table[256] = {
    0x0000, 0x8005, 0x800F, 0x000A, 0x801B, 0x001E, 0x0014, 0x8011,
    0x8033, 0x0036, 0x003C, 0x8039, 0x0028, 0x802D, 0x8027, 0x0022,
    0x8063, 0x0066, 0x006C, 0x8069, 0x0078, 0x807D, 0x8077, 0x0072,
    0x0050, 0x8055, 0x805F, 0x005A, 0x804B, 0x004E, 0x0044, 0x8041,
    0x80C3, 0x00C6, 0x00CC, 0x80C9, 0x00D8, 0x80DD, 0x80D7, 0x00D2,
    0x00F0, 0x80F5, 0x80FF, 0x00FA, 0x80EB, 0x00EE, 0x00E4, 0x80E1,
    0x00A0, 0x80A5, 0x80AF, 0x00AA, 0x80BB, 0x00BE, 0x00B4, 0x80B1,
    0x8093, 0x0096, 0x009C, 0x8099, 0x0088, 0x808D, 0x8087, 0x0082,
    0x8183, 0x0186, 0x018C, 0x8189, 0x0198, 0x819D, 0x8197, 0x0192,
    0x01B0, 0x81B5, 0x81BF, 0x01BA, 0x81AB, 0x01AE, 0x01A4, 0x81A1,
    0x01E0, 0x81E5, 0x81EF, 0x01EA, 0x81FB, 0x01FE, 0x01F4, 0x81F1,
    0x81D3, 0x01D6, 0x01DC, 0x81D9, 0x01C8, 0x81CD, 0x81C7, 0x01C2,
    0x0140, 0x8145, 0x814F, 0x014A, 0x815B, 0x015E, 0x0154, 0x8151,
    0x8173, 0x0176, 0x017C, 0x8179, 0x0168, 0x816D, 0x8167, 0x0162,
    0x8123, 0x0126, 0x012C, 0x8129, 0x0138, 0x813D, 0x8137, 0x0132,
    0x0110, 0x8115, 0x811F, 0x011A, 0x810B, 0x010E, 0x0104, 0x8101,
    0x8303, 0x0306, 0x030C, 0x8309, 0x0318, 0x831D, 0x8317, 0x0312,
    0x0330, 0x8335, 0x833F, 0x033A, 0x832B, 0x032E, 0x0324, 0x8321,
    0x0360, 0x8365, 0x836F, 0x036A, 0x837B, 0x037E, 0x0374, 0x8371,
    0x8353, 0x0356, 0x035C, 0x8359, 0x0348, 0x834D, 0x8347, 0x0342,
    0x03C0, 0x83C5, 0x83CF, 0x03CA, 0x83DB, 0x03DE, 0x03D4, 0x83D1,
    0x83F3, 0x03F6, 0x03FC, 0x83F9, 0x03E8, 0x83ED, 0x83E7, 0x03E2,
    0x83A3, 0x03A6, 0x03AC, 0x83A9, 0x03B8, 0x83BD, 0x83B7, 0x03B2,
    0x0390, 0x8395, 0x839F, 0x039A, 0x838B, 0x038E, 0x0384, 0x8381,
    0x0280, 0x8285, 0x828F, 0x028A, 0x829B, 0x029E, 0x0294, 0x8291,
    0x82B3, 0x02B6, 0x02BC, 0x82B9, 0x02A8, 0x82AD, 0x82A7, 0x02A2,
    0x82E3, 0x02E6, 0x02EC, 0x82E9, 0x02F8, 0x82FD, 0x82F7, 0x02F2,
    0x02D0, 0x82D5, 0x82DF, 0x02DA, 0x82CB, 0x02CE, 0x02C4, 0x82C1,
    0x8243, 0x0246, 0x024C, 0x8249, 0x0258, 0x825D, 0x8257, 0x0252,
    0x0270, 0x8275, 0x827F, 0x027A, 0x826B, 0x026E, 0x0264, 0x8261,
    0x0220, 0x8225, 0x822F, 0x022A, 0x823B, 0x023E, 0x0234, 0x8231,
    0x8213, 0x0216, 0x021C, 0x8219, 0x0208, 0x820D, 0x8207, 0x0202
};
```