Científica, vol. 20, núm. 2, pp. 63-69, julio-diciembre 2016.
ISSN 1665-0654, ESIME Instituto Politécnico Nacional MÉXICO

Árbitros semi-rotatorios con VHDL

Salvador **Saucedo-Flores**
Roberto **Galicia-Galicia**

Instituto Politécnico Nacional
Escuela Superior de Ingeniería Mecánica y Eléctrica, Unidad Zacatenco, Unidad Profesional Adolfo López Mateos, Col. Zacatenco, Delegación Gustavo A. Madero, Ciudad de México, CP 07340.
MÉXICO

correo electrónico (email): ingesauz@msn.com

Recibido 04-11-2015, aceptado 31-05-2016.

Resumen

En los sistemas de buses en un chip (*System on-Chip*, SoC), las propiedades intelectuales (*Intellectual Property*, IP) necesitan comunicarse entre cada una para acceder a la funcionalidad requerida. Cuando el bus del SoC es conectado con más IP, la contención ocurre mientras múltiples IP requieren el uso del bus al mismo tiempo. Ello causa que la comunicación entre arquitecturas basadas en un chip bus sea un reto mayor para el diseñador del sistema con las actuales tecnologías SoC. Las arquitecturas de comunicación deben poder adaptarse por sí mismas de acuerdo con los requisitos de tiempo real de las IP. Por ello, los árbitros de buses son propuestos. El árbitro juega un papel importante en el bus de comunicación del SoC. Los máster en un bus de SoC pueden requerir simultáneamente y, por ello, un árbitro es requerido para decidir cuál máster es electo para el acceso del bus. Un árbitro de buses juega un rol vital en el manejo de peticiones desde el maestro y las respuestas del esclavo (como señal ACK, reintento, etc.). El principal objetivo del algoritmo de arbitración es asegurar que solo un maestro tenga acceso al bus en cualquier tiempo dado, los demás maestros son forzados a permanecer ociosos hasta que reciben la concesión de usar el bus.

Palabras clave: arbitración rotatoria de turno, transferencia a nivel de registro, diseño para sistemas en un integrado, multiprocesadores, codificador programable de prioridad, ancho de banda, división multiplexada de tiempo.

Abstract
(Arbitration Mechanism with VHDL)

Performance of multicore shared bus embedded controller depends on how effectively the sharing resources can be utilized. Common bus in System on Chip (SoC) is one of the sharing resources, shared by the multiple master cores and also acting as a channel between masternucleus and slave core (peripherals) or memories. Arbiter is especialist to use the shared resource (shared bus) successfully, so operation also depends on arbitration techniques.

Arbitration mechanism is used to ensure that only one master has access to the bus at any one time. The arbiter performs this function by observing a number of different demands to use the bus.

Index terms: Round Robin Arbitration (RRA), Register Transfer Level (RTL), System-on-chip Design (SoCD), Multiprocessors, Programmable Priority Encoder (PPE), Bandwith (BW), Time Division Multiplexed (TDM).

1. Introducción

El problema más significante en el diseño del árbitro reside en su eficiencia y bajo costo. Se deben hallar arquitecturas de hardware factibles que puedan satisfacer los siguientes cinco objetivos: *i*) elegir una implementación paralela capaz de arbitrar las peticiones rápidamente [1], [2], [3]; *ii*) mantener el costo de su área a un mínimo valor; *iii*) mantener la ruta crítica a un mínimo valor; *iv*) permitir el procesamiento descentralizado de las entradas: esto es cada petición de entrada debe procesarse en una unidad individual para procesar unidades independientemente; y *v*) asegurar simplicidad en el diseño VLSI por eficiencia, rápidas y automáticas generaciones.

2. Desarrollo

2.1. Árbitro *round-robin* y lapsos iguales

Se obtuvo de la web el módulo VHDL para implementar un árbitro de prioridad rotatoria con espacios iguales de tiempo con cuatro peticionarios y cuatro concesiones, que lo

adecuamos para seis peticionarios y las mismas seis concesiones [4].

```
library ieee;
use ieee.std_logic_1164.all;

entity round_robin is
 port (
  req : in  std_logic_vector(5 downto 0);  -- Request signal
  gnt : buffer std_logic_vector(5 downto 0);  -- grant signal
  clk : in  std_logic;                 -- system clock
  rst : in  std_logic);                -- async reset

end round_robin;

architecture arch of round_robin is

type state is (s_idle,s0,s1,s2,s3,s4,s5);  -- State declaration
signal present_state,next_state : state;  -- State container
begin  -- arch

 -- purpose: fixing the priority and grant signal assignment
 -- type   : combinational
 -- inputs : req
 -- outputs: gnt
 priority: process (present_state,req)
 begin  -- process priority

   case present_state is

   when s_idle =>   if(req(0)='1')then
      gnt<="000001";  next_state<=s0;
    elsif req(1)='1' then  gnt<="000010";
      next_state<=s1;  elsif req(2)='1' then
      gnt<="000100";   next_state<=s2;
    elsif req(3)='1' then  gnt<="001000";
    next_state<=s3;  elsif req(4)='1' then
      gnt<="010000";     next_state<=s4;
   elsif req(5)='1' then    gnt<="100000";
      next_state<=s5;     else
       gnt<="000000";   next_state<=s_idle;
    end if;

 when s0 =>  if(req(1)='1')then
   gnt<="000010";  next_state<=s1;
  elsif req(2)='1' then  gnt<="000100";
   next_state<=s2;  elsif req(3)='1' then
   gnt<="001000";    next_state<=s3;
  elsif req(4)='1' then  gnt<="010000";
   next_state<=s0;  elsif req(5)='1' then
   gnt<="100000";   next_state<=s0;
  else     gnt<="000000";
    next_state<=s_idle;
   end if;

  when s1 =>   if(req(2)='1')then
    gnt<="000100";     next_state<=s2;
  elsif req(3)='1' then    gnt<="001000";
    next_state<=s3;  elsif req(4)='1' then
    gnt<="010000";    next_state<=s4;
  elsif req(5)='1' then   gnt<="100000";
    next_state<=s5;  elsif req(0)='1' then
    gnt<="000001";    next_state<=s0;
  elsif req(1)='1' then    gnt<="000010";
    next_state<=s1;     else
   gnt<="000000";   next_state<=s_idle;
end if;

--por brevedad se omiten los casos para s2 a s4

when s5 =>
     if(req(0)='1')then
       gnt<="000001";
        next_state<=s0;
     elsif(req(1)='1')then
       gnt<="000010";
        next_state<=s1;
     elsif req(2)='1' then
       gnt<="000100";
        next_state<=s2;
     elsif req(3)='1' then
       gnt<="001000";
        next_state<=s3;
     elsif req(4)='1' then
       gnt<="010000";
        next_state<=s3;
     elsif req(5)='1' then
       gnt<="100000";
        next_state<=s4;
      else
        gnt<="000000";
         next_state<=s_idle;
      end if;
  end case;
 end process priority;
 -- purpose: State Assignment
 -- type   : sequential
 -- inputs : clk, rst, present_state
 -- outputs: next_state

 State_assignment: process (clk, rst)
 begin  -- process State_assignment
  if rst = '1' then               -- asynchronous reset (active high)
    present_state<=s_idle;
```

```
   elsif clk'event and clk = '1' then  -- rising clock edge
      present_state<=next_state;
   end if;
  end process State_assignment;

-- purpose: Assertion_assignment
 -- type   : combinational
 -- inputs : gnt
 -- outputs:

  Assertion_assignment: process (gnt)
  begin  -- process Assertion_assignment
   if(gnt="000001") then
      assert false report "Request 0 is granted" severity note;

   elsif gnt="000010" then
      assert false report "Request 1 is granted" severity note;

   elsif gnt="000100" then
      assert false report "Request 2 is granted" severity note;

   elsif gnt="001000" then
      assert false report "Request 3 is granted" severity note;
  --  end if;

   elsif gnt="010000" then
      assert false report "Request 4 is granted" severity note;
   elsif gnt="100000" then
      assert false report "Request 5 is granted" severity note;
    end if;
  end process Assertion_assignment;
end arch;
```

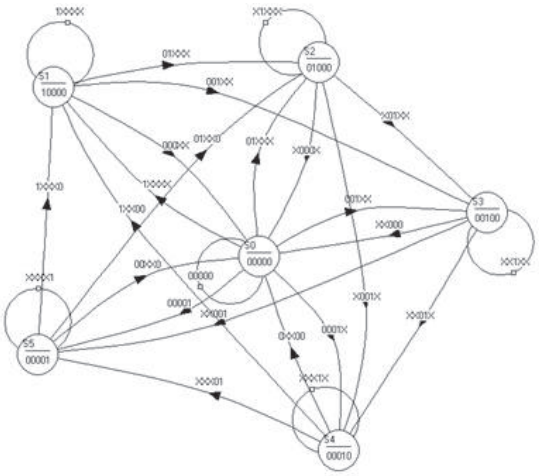

Fig. 1. Diagrama de estado del árbitro semi-rotatorio.

2.2. Árbitro semi-rotatorio con boole.deusto

Se empleó software didáctico gratuito de la web que se puede definir el diagrama de estados y es capaz de generar el módulo VHDL para implementar el diseño. Se implementó con éxito el módulo VHDL generado por el software vasco en una GAL22V10 para cinco peticionarios. La Fig. 1 cómo definir el diagrama de estados. Por brevedad, no se adjunta el módulo VHDL [5].

2.3. Árbitro con captura esquemática

Se usó el software profesional ispLEVER classic para crear un árbitro con prioridad rotatoria usando la biblioteca Vantis y el estilo jerárquico. La Fig. 2 exhibe el símbolo más importante [6], [7].

El diseño se probó sobre la tarjeta de desarrollo Breakout de Lattice, que usa el CPLD ispMACH4256ZE. Emplea un contador de anillo y cinco decoders 74138, también de la biblioteca Vantis (véase Fig. 3 y Fig. 4).

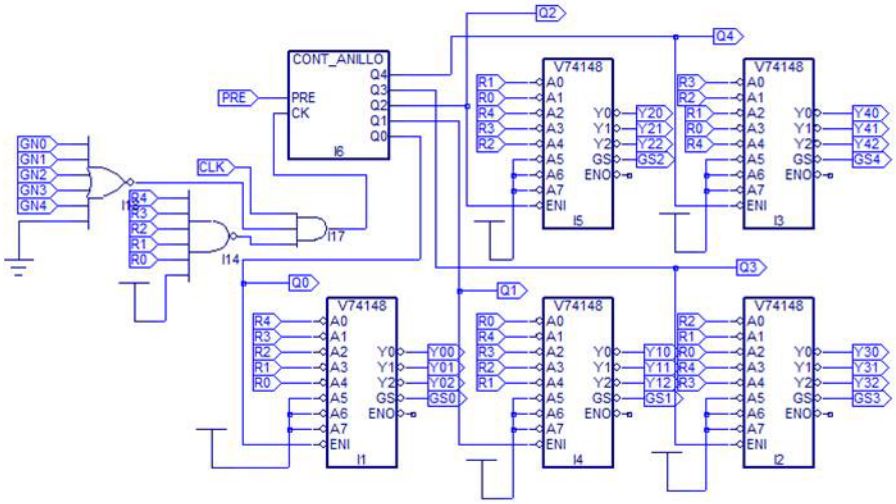

Fig. 2. Símbolo con codificadores de prioridad.

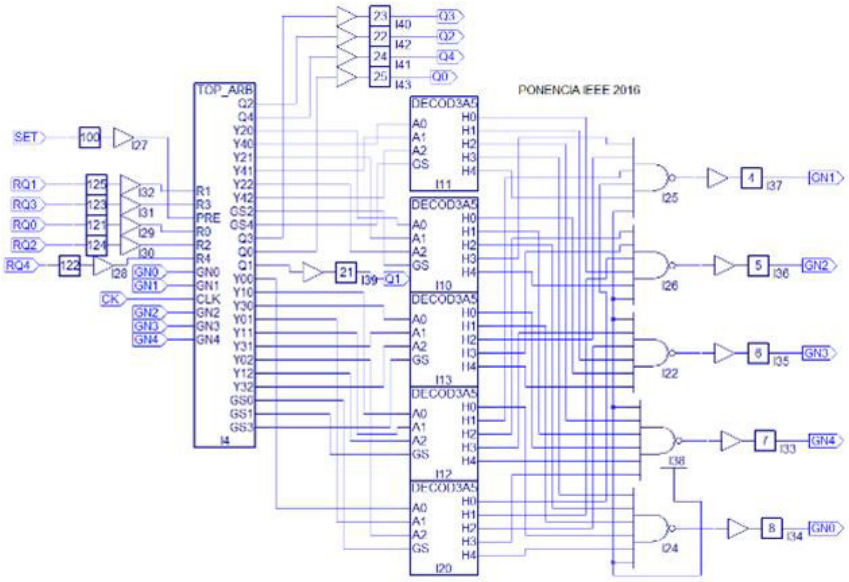

Fig. 3. Esquema de jerarquía máxima.

2.4. TDM esquema de división de tiempo

En el esquema de división multiplexada de tiempo (TDM), el tiempo de ejecución es dividido en ranuras de tiempo (mini-ranuras) y coloca cada ranura para la petición del maestro, Ello es un algoritmo de arbitración de dos niveles [8]. En primer nivel, se tiene una rueda de tiempo, donde las ranuras han sido reservadas para cada maestro particular, múltiples ranuras serían necesarias para realizar todas las transferencias, si hay algún maestro solicitando en el momento actual y tiene pendiente una petición para acceder el bus, el árbitro concesiona el acceso al que pide el bus y la rueda se moverá a otra ranura próxima, esto es muy simple y fácil de implementar pero su desventaja asociada con ella, la pobre latencia y el tiempo perdido en las ranuras. Para superar tal desventaja, el segundo nivel de arbitración permite que el bus para otro maestro solicitando. Por ejemplo, si la presente ranura está reservada para M1 para comunicarse y sus datos no están el segundo nivel de arbitración cuenta con un puntero round-robin rr2 que incrementa el puntero desde su posición actual M2 hacia otra petición próxima M4, según se muestra en la Fig. 5. Otro árbitro similar se presenta en el siguiente listado.

Se muestra el árbitro de B. Krill [8] para 24 peticionarios con lapsos de tiempo iguales (véase Fig. 6).

```
-- Copyright (c) 2009 Benjamin Krill <benjamin@krll.de>
--
-- Permission is hereby granted, free of charge, to --- any person obtaining a copy
-- of this software and associated documentation ---- files (the "Software"), to deal
-- in the Software without restriction, including ------ without limitation the rights
-- to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

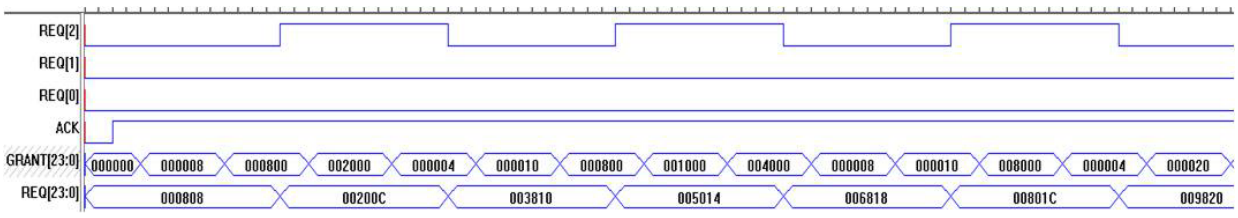

Fig. 4. Simulación de los vectores de prueba.

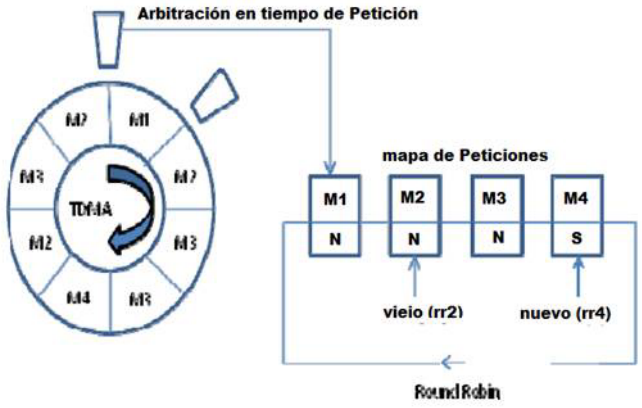

Fig. 5. Arquitectura de comunicación [5].

```
-- copies of the Software, and to permit persons to whom the
Software is
-- furnished to do so, subject to the following conditions:
--
-- The above copyright notice and this permission notice
shall be included in
-- all copies or substantial portions of the Software.
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rrarbiter24 is
  generic ( CNT : integer := 24 );
  port (
    clk   : in   std_logic;
    rst_n : in   std_logic;
    req   : in   std_logic_vector(CNT-1 downto 0);
    ack   : in   std_logic;
    grant : out  std_logic_vector(CNT-1 downto 0)
    );
end;
architecture rrarbiter of rrarbiter24 is
signal grant_q  : std_logic_vector(CNT-1 downto
0);
signal pre_req  : std_logic_vector(CNT-1 downto
0);
  signal sel_gnt  : std_logic_vector(CNT-1 downto
0);
  signal isol_lsb : std_logic_vector(CNT-1 downto
0);
  signal mask_pre : std_logic_vector(CNT-1 downto 0);
  signal win     : std_logic_vector(CNT-1 downto 0);
begin
          grant    <= grant_q;
mask_pre <=     req and not
(std_logic_vector(unsigned(pre_req) - 1) or pre_req); --
Descarta a ganadores previos
sel_gnt  <= mask_pre and
std_logic_vector(unsigned(not(mask_pre)) + 1);      --
Selecciona nuevo ganador
isol_lsb <=     req and
```

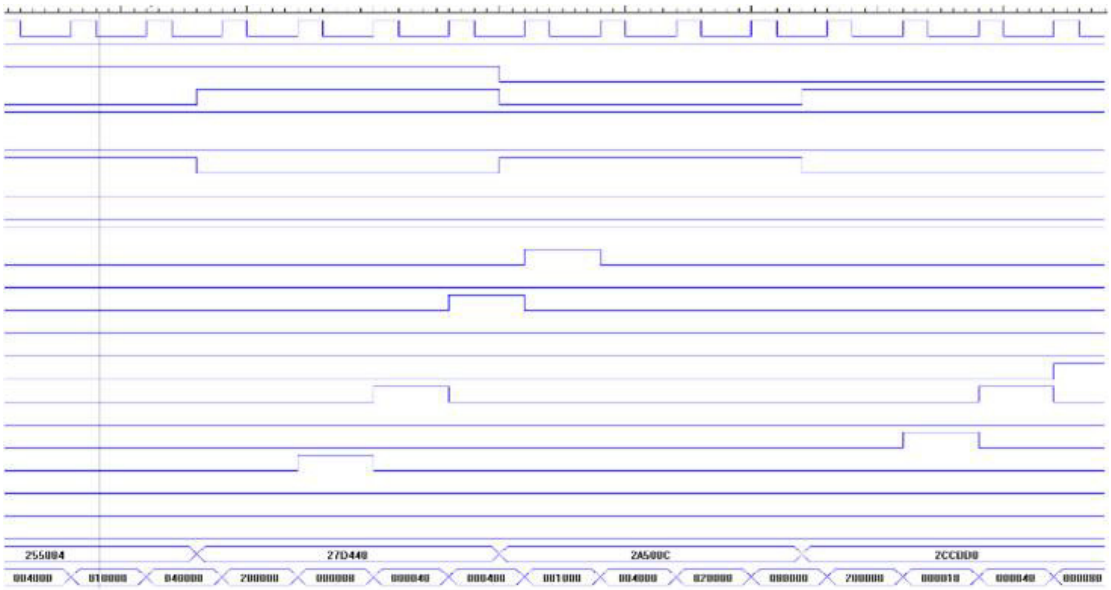

Fig. 6. Simulación con 24 masters.

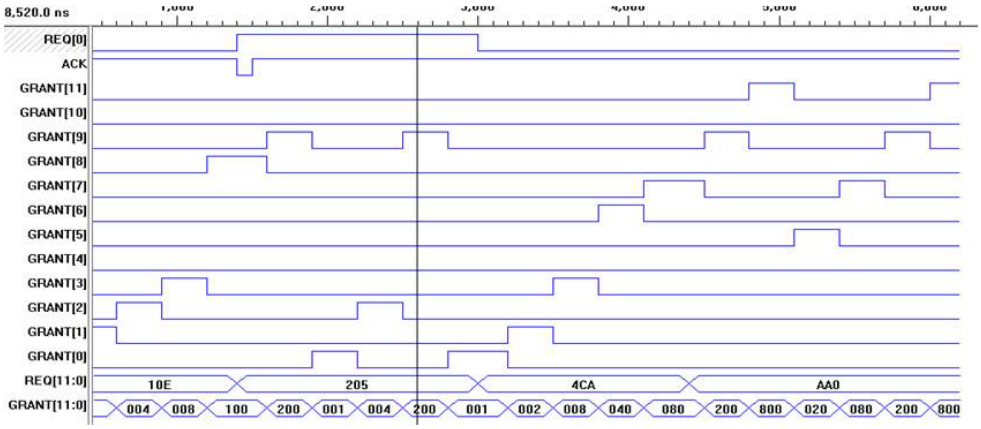

Fig. 7. Simulación con 12 masters.

```
std_logic_vector(unsigned(not(req)) + 1);          -- Isolate
least significant set bit.
win     <= sel_gnt when mask_pre /= (CNT-1 downto 0 =>
'0') else isol_lsb;

            process (clk, rst_n)
            begin
            if rst_n = '0' then
                        pre_req <= (others => '0');
                        grant_q <= (others => '0');
            elsif rising_edge(clk) then
                        grant_q <= grant_q;
                        pre_req <= pre_req;
if grant_q = (CNT-1 downto 0 => '0') or ack =
'1'

then
if win /= (CNT-1 downto 0 => '0')
then
                                    pre_req <= win;
                                    end if;
                                    grant_q <= win;
                        end if;
            end if;
            end process;
end rrarbiter;
```

Para facilitar el análisis de los vectores de prueba el listado anterior se redujo a doce peticionarios, generándose el diagrama de pruebas, según la Fig. 7.

2.5. Árbitro semi-rotatorio con biestables

Se diseñó mediante el software didáctico LogicAid un árbitro con cuatro peticiones que se pasa a su logigrama, mismo que se muestra en la Figura 8. Dicho logigrama se puede pasar a un CPLD o armarlo con flip-flops comerciales.

3. Conclusiones

a) Es imperativo el introducir el software de diseño de circuitos digitales y evitar el uso de compuertas muy obsoletas.
b) Es conveniente ver temas algo más avanzados como registros de corrimiento LFSR y diseño de árbitros.

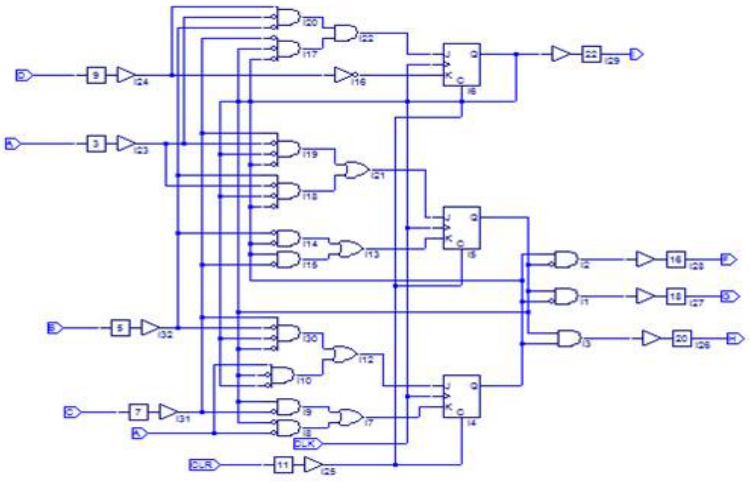

Fig. 8. Arbitro diseñado con LogicAid

c) Otra sugerencia es el empleo del campus virtual de la escuela, que permite una interacción más ágil y más completa con el grupo.
d) Se sugiere el uso de herramientas de diseño en la web, como AGT (*Arbiter Generator Tool*) para el diseño rápido de árbitros.

Referencias

[1] J. M. Jou, Y. L. Lee and S. S. Wu, "Efficient design and generation of a multi-facet arbiter," en *Application Specific Processors (SASP), 2010 IEEE 8th Symposium*, Anaheim, CA, USA, julio, 2010, pp. 111-114 [en línea]. Disponible en doi:10.1109/SASP.2010.5521137

[2] R. Shashidhar R., S. N. Sujay and G. S. Pavan G.S3, "Implementation of Bus Arbiter Using Round Robin Scheme," *International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)*, vol. 3, no. 7, pp. 14937-14944, julio, 2014.

[3] S. S. Zalivaka, A. V. Puchkov, V. P. Klybik, A. A. Ivaniuk and C. H. Chang, "Multi-valued Arbiters for Quality Enhancement of PUF Responses on FPGA Implementation," *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macao, China, 25-28 enero, 2016.

[4] VLSICoding, *VHDL Code for Round Robin Arbiter with Fixed Time Slices*, octubre, 2013 [en línea]. Disponible en: https://vlsicoding.blogspot.mx/2013/10/design-round-robin-arbiter-with-fixed.html

[5] R. Khanam, H. Sharma and S. Gaur, "Design a low latency Arbiter for on chip Communication Architecture," en *2015 International Conference on Computing, Communication & Automation (ICCCA)*, Noida, India, 15-16 mayo, 2015, pp. 1421-1426.

[6] E. S. Shin, V. J. Mooney and G. F. Riley, "Round-robin Arbiter Design and Generation," en *Proceedings of the 15th international symposium on System Synthesis ISSS'O2*, 2-4 octubre, 2002, Kyoto, Japón, pp. 243-248.

[7] A. Paul, M. Haque Khan, M. M. Rahman, T. Z. Khan. P. Podder and Y. A., "Reconfigurable Parallel Architecture of High Speed Round Robin Arbiter," en *Electrical, Electronics, Signals, Communication and Optimization (EESCO)*, Visakhapatnam, India, 24-25 enero, 2015.

[8] B. Krill, " VHDL Round-Robin Arbiter," octubre, 2015. Disponible en www.krll.de/portfolio/round-robin-arbiter