Universidad de Concepción
Facultad de Ciencias Físicas y Matemáticas
Departamento de Física

Diseño de prototipos II:

Programación y diseño avanzado de sistemas digitales, para uso en radío astronomía

Autor:
Gonzalo Burgos F

Profesor a cargo :
Dr. Rodrigo Reeves

Índice general

Índice de figuras

Capítulo 1

Introducción

El avance científico está siempre acompañado de la necesidad de generar nuevos equipos capaces de detectar y medir los parámetros de un fenómeno, independientemente del área de estudio. Esto ha hecho que cada vez la precisión en las mediciones y el tratamiento de los datos deba ser más riguroso y eficiente. Esta necesidad ha sido suplida gracias a la aparición de las nuevas tecnología de procesamiento (calculadoras, procesadores, etc.), dicho de otro modo, en el desarrollo digital del procesamiento de información.

Actualmente los sistemas digitales son capaces de solucionar problemas complejos a través de la implementación de algoritmos, tanto en hardware como software, donde la mayor diferencia se centra en el desempeño del sistema considerando su velocidad, potencia, costo y el tamaño de los dispositivos. Hoy en día los sistemas enbebidos conforman la mayor parte los sistemas digitales (un ejemplo cercano es la RAM de una computadora) los cuales permiten una alta adquisición, procesamiento y control de datos.

Cuando se habla de programación en hardware se hace referencia los componentes físicos del sistema digital, ya sean compuertas lógicas, micro-procesadores, u otros. Los cuales pueden ser programados de manera directa (HDL). El Software por otra parte corresponde a códigos pre-programados capaces de realizar tareas a partir de los componentes del hardware. Si bien ambas formas de programación trabajan sobre un mismo sistema físico estas presentan distintas ventajas y desventajas; El hardware por una parte nos entrega un mayor potencia de procesamiento(desempeño), pero presenta menor flexibilidad en la programación; El software por su parte presenta un flexibilidad enorme a la hora del desarrollo en programación, pero genera una gran pérdida en la potencia y velocidad.

En la actualidad se buscan sistemas de programación en hardware de fácil desarrollo, con el objetivo de lograr una gran potencia, eficiencia y flexibilidad de programación. Uno de los avances en este ámbito es el uso de placas de desarrollo con FPGA (Field Programmable Gate Array).

Esta tecnología digital es útil ya que es capaz de generar una estructura sobre el sistema de procesamiento de información de forma directa, sin la necesidad de hacer uso de un software y de manera instantánea. Además, proveer un entorno de desarrollo capaz de interactuar con múltiples señales y/o estímulos de forma simultánea. Obteniendo de esta forma una herramienta de procesamiento muy poderosa y de fácil desarrollo.

Capítulo 2

Hardware

El fin de este estudio es poder programar sistemas que consten de una FPGA en su arquitectura como lo es la Artix-7 35t Arty FPGA, para ello es necesario conocer de antemano las componente que forman una FPGA y su funcionalidad dentro de ella. Por ello en esta sección se abordará la FPGA desde un punto de vista estructural y como ésta genera una programación jerárquica desde el funcionamiento de sus principales componentes. Obviando su composición a base de circuitos analógicos (como amplificadores y filtros), transistores y diodos, y la física detrás de ellos.

2.1. FPGA

La sigla FPGA viene del inglés Field Programmable Gate Array que representa literalmente su funcionamiento, el cual consiste en conjuntos dispositivos semiconductores distribuidos en una matriz lógica de bloques reconfigurable, a través de conexiones reconfigurables. En la Fig.(2.2) se presenta un esquema ilustrativo .

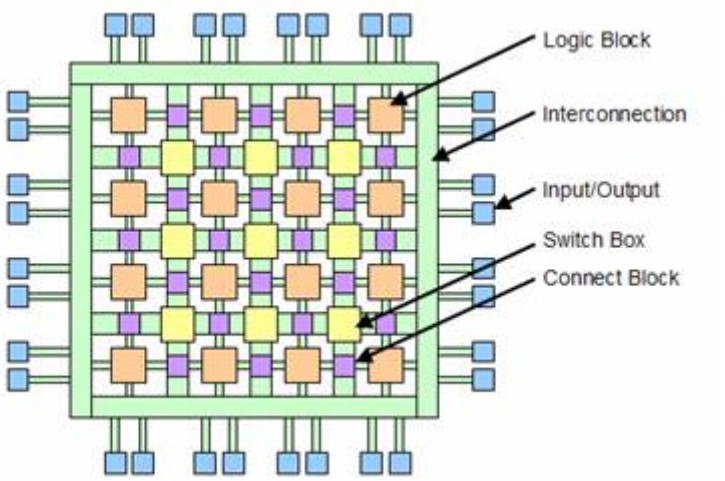

Figura 2.1: Esquema FPGA

Dentro de esta estructura se incluyen además dispositivos especializados como; multiplicadores/acumuladores, bloques de RAM, manejo de reloj, comunicación a alta velocidad, interfaces de memoria y E/S, conversión A/D.

2.1.1. Estructura y Funcionamiento

El funcionamiento de la FPGA sigue un orden jerárquico de programación, en sus principios más elementales cuenta con un gran número de pequeños bloques lógicos configurables basados en LUT (lookup table) encargado de la implementación de funciones lógicas, flip-flop capaces de almacenar información desde el sistema o externa a este y multiplexores que permiten la entrada y salida de información del flip-flop o del bloque, juntos forman la primera estructura programable la cual recibe el nombre de celda lógica (LC), ver Fig.(2.3)

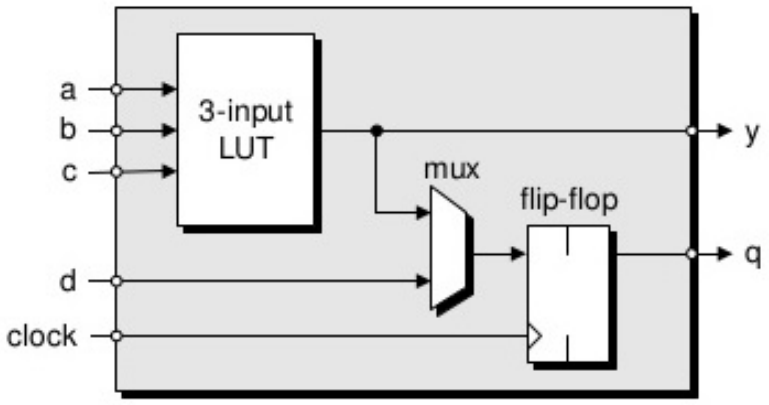

Figura 2.2: Celda lógica (simplificada)

Los LUT de una LC son reconfigurables, generando distintas aplicaciones para cada LC. Uno de ellos es el uso de lut como RAM y el otro es el uso de LUT en cadena para generar registros de desplazamiento, así un LUT tiene 3 posibles aplicaciones.

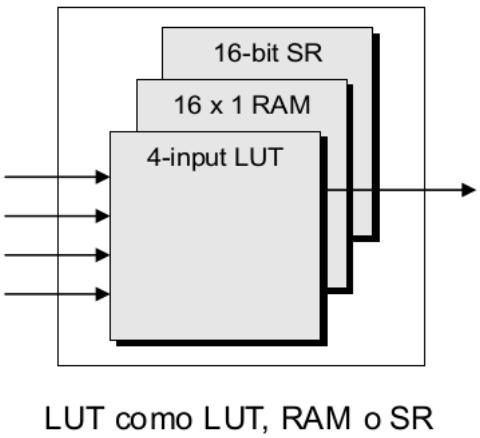

Figura 2.3: Configuraciones de LUT

La siguiente jerarquía en la programación es la generada a partir de un conjunto LC, recibe el nombre de Slice, ver Fig.(2.3). Dado que un Slice esta compuestos por un conjunto de LC estos presentan 2 configuración, SLICEL donde los LUT son usados en su forma basica y SLICEM donde los LUT son usados como RAM o SR.

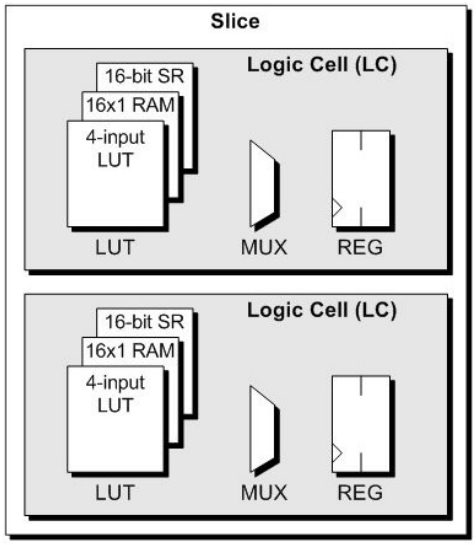

Figura 2.4: Slice logico (simplificado)

Un nivel más arriba en la estructura de programación se encuentra los bloques de lógica programable (CLB), que corresponden a la agrupación de Slice, ver Fig.(2.5).

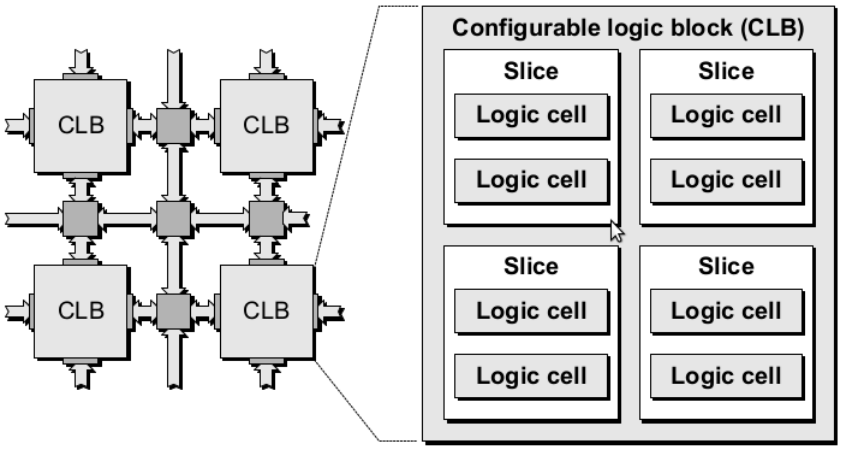

Figura 2.5: Bloque de lógica programable (CLB)

Este elemento es la base de la red de interconexión de la FPGA, la arquitectura de CLB permite una rápida conexión de transferencia entre los diferentes Slice de cada columna. En la siguiente Fig.(2.6) se aprecia la interfaz de transferencia entre los diferentes CLB y los respectivos Slice.

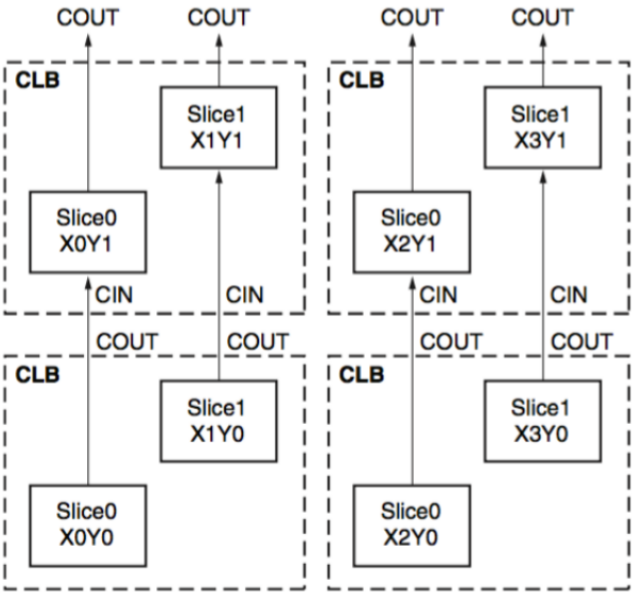

Figura 2.6: Conexiones CLB

Si bien los CLB son el corazón del procesamiento de las FPGA, estas cuentan con una estructura mas amplia de trabajo donde se añaden 2 nuevos componentes bloques RAM y Multiplicadores/acumuladores (M/A). En la Fig.(2.6) se muestra un esquema de distribución de estos componentes. Los

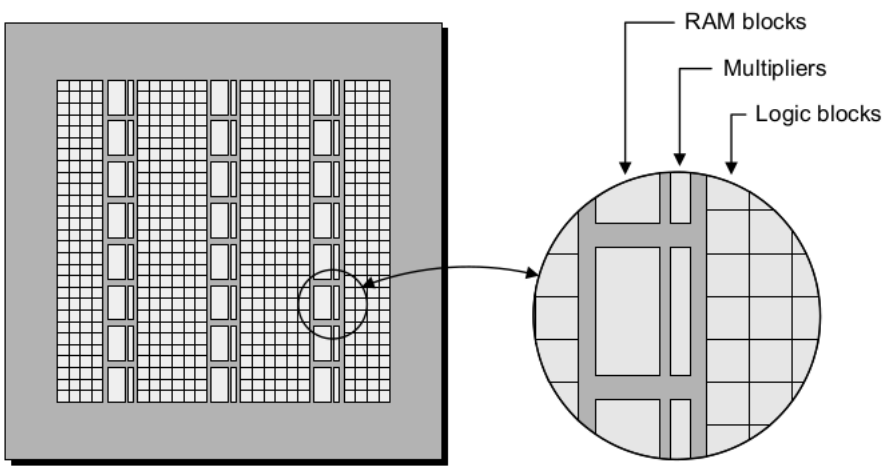

Figura 2.7: Bloques RAM y M/A

bloques RAM son memorias de mayor tamaño y más eficientes que las memorias LUT usadas como memorias distribuidas, además es posible configurarlas poder ser usadas como memorias de doble puerta, FIFO (First in, first out), entre otras aplicaciones.
Los Multiplicadores/acumuladores por su parte generan una rápida cadena de transporte de información capas de implementar sumadores entre los CLB, y además habilitan la programación en hardware de aplicaciones DSP (digital signal processor) de alto desempeño.
Otro componente de importancia presente en la arquitectura de la FPGA corresponde al digital clock manager, el cual permite generar señales de control (relojes) a partir de una señal externa, posee además líneas dedicadas a la distribución de las señales de reloj tanto global como regionalmente, reduciendo el jitter.

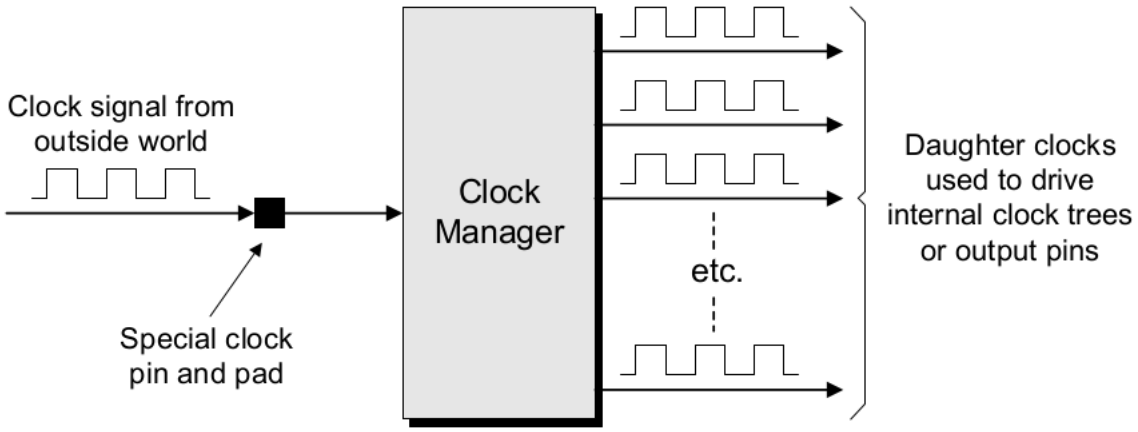

Figura 2.8: Caption

Finalmente la FPGA en sus parte mas externa presenta una serie componentes de interacción como los son las entradas y salidas (E/S) programables de cada banco y bloques de transferencia de datos con transmisor (Tx) y receptor (Rx) combinados.

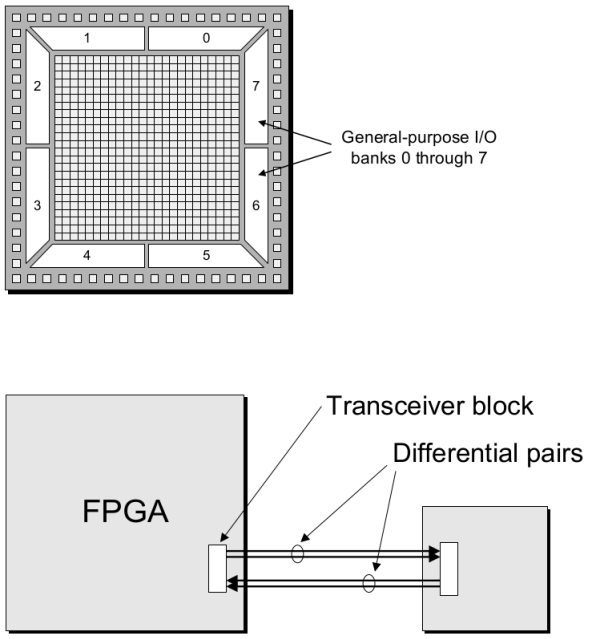

Figura 2.9: E/S y bloques de transferencia

Antes de seguir, es importante notar que toda la arquitectura antes mencionada es básica para todas las FPGA, pero cada uno de los modelos de FPGA dependiendo del fabricante y la serie presenta diferentes distribuciones, componentes o funcionalidades dentro de ellas.

2.2. Placas de desarrollo

En esta sección se abordan las características de la placa de desarrollo seleccionada para el prototipo, se contempla la descripción de las características principales de la tarjeta y las especificaciones del fabricante, haciendo énfasis en las características de su FPGA.

2.2.1. Artix-7 35T Arty FPGA

Es una placa desarrollo producida y distribuida por Xilinx, su programación es compatible con Vivado, herramienta de diseño de alto desempeño también desarrollada por Xilin, el entorno y herramientas de Vivado serán discutidas más adelante.

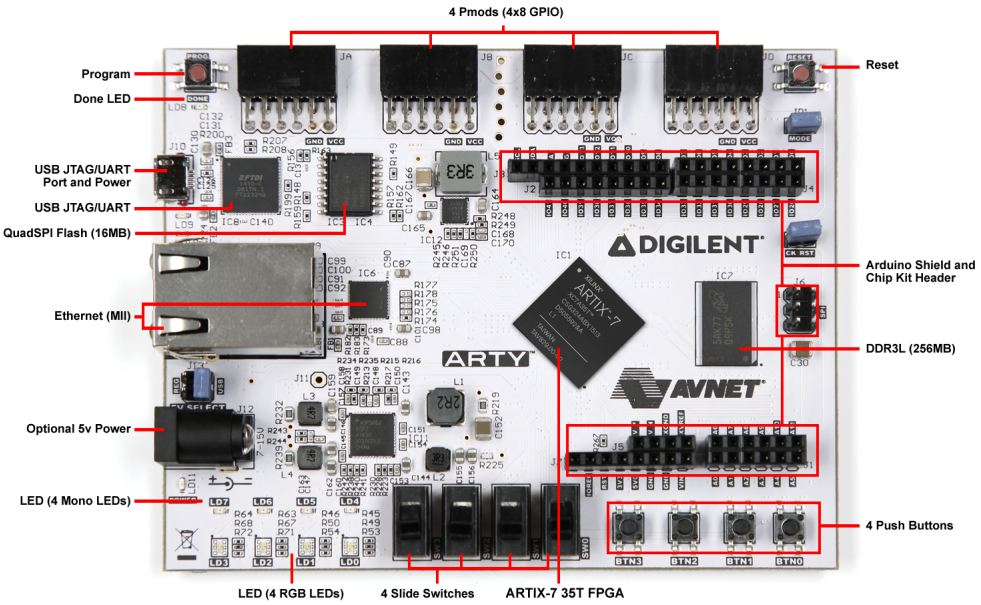

Figura 2.10: Artyx-7 35T Arty FPGA

En la Fig.(2.10) se muestra la arquitectura y distribución de los componentes de la placa, de manera más específica, estos son:

- Xilinx Artix-35T FPGA (xc7a35ticsg324-1L)
  - 5,200 slices (cada slice contiene cuatro LUTs de 6-input y 8 flip-flops);
  - 1,800 Kbits de velocidad en bloques de RAM;
  - 5 etapas de control de reloj, cada una con un phase-locked loop (PLL);
  - 90 DSP slices (digital signal processor);
  - Un reloj interno con velocidad sobre los 450 Mhz;
  - n analog-to-digital converter (XADC);
  - Programable por JTAG y Quad-SPI Flash
- Características y herramientas del sistema
  - RAM de 256 MB DDR3L con una capacidad de 16-bit bus @ 667MHz;
  - 16MB Quad-SPI Flash
  - Circuito de programación USB-JTAG.

    - Alimentación de poder a través de USB o cualquier fuente entre 7V-15V
- Conectividad del sistema
    - 10/100 Mbps Ethernet
    - USB-UART Bridge
- Dispositivos de interacción
    - 4 Switches
    - 4 Buttons
    - 1 Reset Button
    - 4 LEDs
    - 4 RGB LEDs
- Extinciones de conexión
    - 4 conectores Pmod
    - Arduino/ChipKit Shield connector

Capítulo 3

Lenguaje de programación y software

En esta sección se abordaran las principales herramientas entregadas por el software Vivavdo y el lenguaje de programación System Verilog. El lenguaje completo esta disponible en el siguiente link y la interfaz Vivado esta disponible aquí

3.1. SystemVerilog

Los lenguajes de programación como éste, permiten responder a la necesidad de diseñar y verificar sistemas digitales de gran complejidad a través de modelos de computo específico. La modelación de estos sistemas puede llevarse acabo a diferentes niveles; estructural, combinacional, FSM y transferencia de registros (RTL) y haciendo uso de herramientas de modelación como esquemas jerárquicos en compuertas lógicas, diagramas de estado o lenguajes de descripción de hardware HDL.

Para realizar esta tarea SystemVerilog se hace uso de módulos de integración (DUT) basados en la interconexión de módulos de menor nivel, o de compuertas lógicas, o la combinación de ambos. Los DUT de un sistema especifican como es generado un estado en el sistema a partir de entradas y estados pasados, y como se generan salidas a partir de los estados y entradas, dicho de otra modo system Verilog es un lenguaje de modelación que integra los ya nombrados niveles de programación junto al uso de su propio HDL basado en Verilog.

3.1.1. Herramientas del lenguaje

- **Datos**
  - Nets; Representan las conexiones estructurales entre componentes, no posee capacidad de almacenamiento .
  - Registers; Representan variables con capacidad de almacenar información.
- **Variables**
  - bit; puede tomar valores 0,1,X y Z cuando se habla de un bit de 4 estados o 0 y 1 cuando se habla de bit de 2 estados
  - byte; formado por 8 bits de 2 estados.
  - shorint, formado por 16 bits de 2 estados.
  - int; formado por 32 bits de 2 estados.
  - time; formado por 64 bit tamaño de 4 estados.
  - longint; formado por 64 bits de 2 estados.

- **Operadores**

  Aritméticos

  - + ; a=+8'h01 =¿a=+8'h01.
  - + ; a=-8'h01 =¿a=-8'h01.
  - + ; a=b+c.
  - - ; a=b-c.
  - * ; a=b*c.
  - / ; a=b/c =¿Parte entera de la división.
  - %; a=b %c =¿Módulo de la división.

  Relacionales

  - < ; $a < b$, a es menor que b.
  - > ; $a < b$, a es mayor que b.
  - <=; $a <= b$, a es menor o igual que b.
  - >=; $a >= b$, a es mayor o igual que b.

  Igualdad

  - == ; a==b, a es igual a b .
  - != ; a!=b, a es diferente de b.
  - === ; a===b, a es igual a b, incluyendo estados X y Z.
  - !== ; a!==b, a esdiferente de b, incluyendo los estados X y Z.

  Logicos

  - ! ; Negación.
  - && ; AND.
  - || ; OR

  Bit a bit y de reducción
  Los operadores de reducción operan sobre un conjunto de bits y entregan una sola salida, mientras que los bit a bit solo operan sobre un par de bits.

  - $\sim$; Negación.
  - & ;AND.
  - | ; OR.
  - $\wedge$; OR exclusiva.

  Desplazamiento, concatenación y condicional

  - $\ll$; Desplazamiento a la izquierda.
  - $\gg$; Desplazamiento a la Derecha.
  - ; Concatenación de elementos (bits o arreglos de bits).
  - ? ; Condicional, selecciona un valor arbitrario dependiendo del estado (0 ó 1).

- **Estructuras**
  - **If -else**
    Se encarga de controlar la ejecución de otras sentencias y/o asignaciones.
  - **Case, caseZ y caseX**
    Sentencias que evalúa una expresión y en función de su valor ejecuta la sentencia o grupo de sentencias agrupadas en el primer caso coincidente. Mientras que case evalúa considerando 4 posibles valores para cada bit, caseZ considera el Z como valores indiferentes y caseX considera tanto X como Z como valores indiferentes.
  - **Bucle**
    Existen 4 tipos de sentencias bucles que deben ser contenidas en bloques procedurales.
    - *Forever*: se ejecuta de forma continua, sin condición de finalización.
    - *Repeat*: se ejecuta un determinado número de veces, este número condiciona su finalización.
    - *While*: se ejecuta mientras la condición que se evalúa es cierta.
    - *For*: se ejecuta iterativamente de forma creciente desde un valor inicial hasta uno final.

3.1.2. Niveles de modelación y sistemas

- **Estructural**
  A este nivel SystemVerilog describen el comportamiento de un módulo como la interconexión de otros módulos o compuertas lógicas. Esto permite generar nuevos módulos a partir de la raíz de la programación en la Fig.(3.1) se muestra un ejemplo de este nivel de modelación.

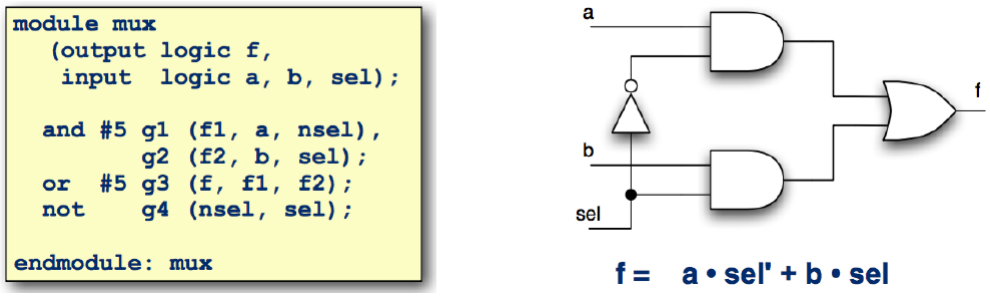

Figura 3.1: Modelación estructural

- **Descripción RTL**
  Es un nivel de abstracción diseñado para modelar circuitos secuenciales sincrónicos en función del flujo de señales digitales (arreglos continuos de datos) entre los registros de hardware, y las operaciones lógicas realizadas sobre las señales. La operaciones se expresan a un nivel de abstracción procedural o de operaciones aritméticas/lógicas, en vez de sobre compuertas lógicas interconectas. Para trabajar a este nivel SystemVerilog entrega un entorno llamado bloques procedurales los cuales se ejecutan cada vez que alguna señal utilizada en lectura dentro del bloque cambia de valor. La Fig(3.2) se muestra un ejemplo de este nivel de modelación.

- **Sistemas secuenciales**
  Los sistemas digitales secuencias no responden solamente al valor actual de sus entradas, sino que también son capaces de usar y almacenar memoria de los estados anteriores. Estos sistemas pueden ser; asincrónico donde los estados pueden cambiar en cualquier momento, suelen ser rápidos y eficiente pero complejos de analizar, verificar e implementar de manera confiable;

```
module fullAdder (output logic s, cOut,
                  input logic a, b, cIn);
  logic t;

  always_comb begin
    t = #5 a ^ b; // no usar t antes de esto
    s = #10 t ^ cIn;
  end

  always_comb begin
    cOut = t & cIn;
    cOut = # 6 cOut | (a & b); // cOut no genera memoria
  end

endmodule : fullAdder
```

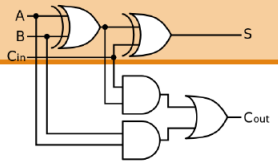

Figura 3.2: Modelación procedural

o bien sincrónicos donde los cambios de estado sólo son permitidos en instantes discretos de tiempo y todos ocurren de manera simultánea. En la Fig.(3.3) se muestra un ejemplo del uso de estos sistemas, donde es utilizado para generar un señal periódica de menor frecuencia a partir de una de mayor frecuencia.

```
module clockDivider (input logic clk, output divClk);
  parameter TC = 1000;
  localparam nBits = $clog2(TC) + 1;  // $clog2 calcula el techo del log2 del argumento
                                      // El " + 1" es para permitir dividir por 2

  logic [nBits - 1: 0] counter;       // " + 1" garantiza un registro de mínimo 2 bits

  assign divClk = (counter == TC - 1);

  always_ff @(posedge clk)
    if (divClk)
      counter <= 0;
    else
      counter <= counter + 1;

endmodule : clockDivider
```

Figura 3.3: Sistema secuencial

3.2. Vivado

En esta sección se abordarán las características del entorno de desarrollo y las distintas etapas que permite la configuración de las tarjetas. En la Fig.(3.4) se muestra el entorno de Vivado.
La enumeración presente en Fig(3.4) corresponden a:

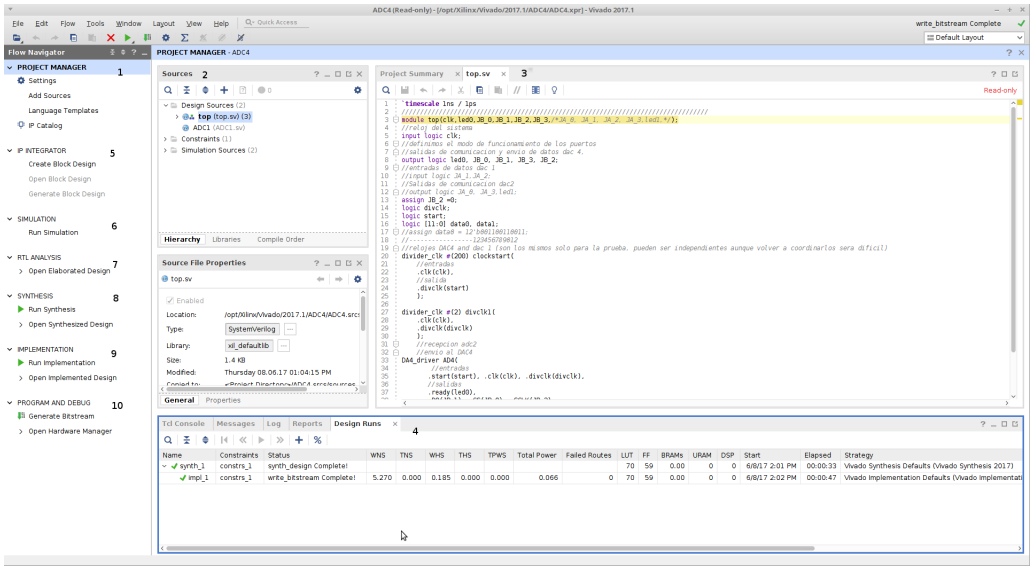

Figura 3.4: Escritorio Vivado

1. Sección en cargada de la creación y diseño de bloques de trabajo.
2. Sección que ordena jerárquicamente y da acceso a los bloques de trabajo, constrains y fuentes de simulación.
3. Sección que muestra el contenido de los bloques de trabajo, además permite la edición de dichos bloques.
4. Sección que muestra una serie de herramientas de programación como lo es la consola TCL, mesajes de alerta y error de programación, el estado actual del proyecto, etc.
5. Sección que permite transformar los bloques de código programados en bloques gráficos.
6. Sección que permite realizar un simulación del código y verificar su funcionamiento.
7. Sección que realiza análisis RTL del código y genera un esquema visual de bloques Fig.(3.5).
8. Sección que genera y sintetiza el código, Fig.(3.6).
9. Sección que genera el archivo final de implementación, el código de la sección anterior es sintetizado considerando ahora el constrains y el modelo de la placa, presenta además una representación gráfica del programación final.
10. Sección final, aquí se implementa el código en la tarjeta.

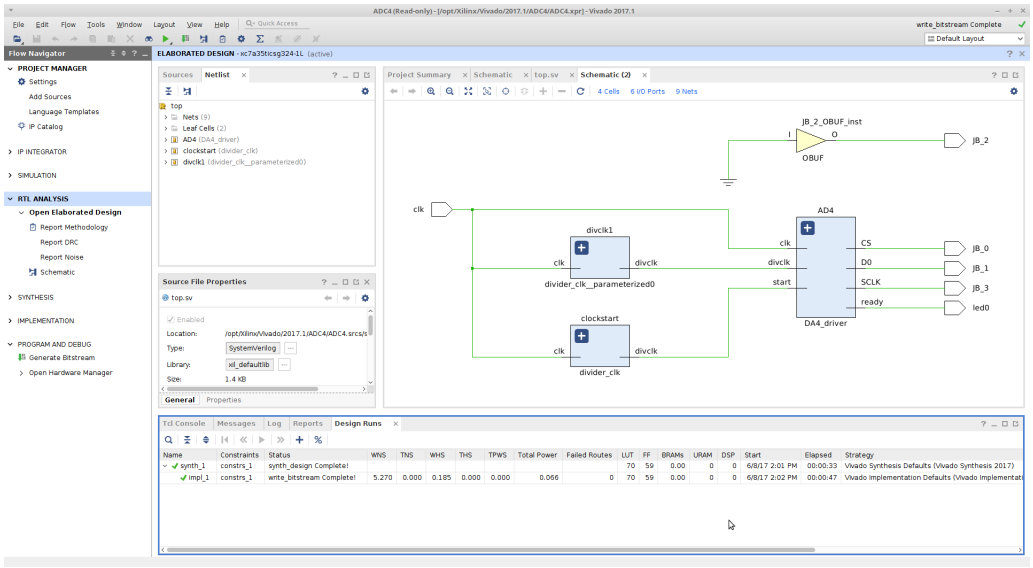

Figura 3.5: Análisis RTL

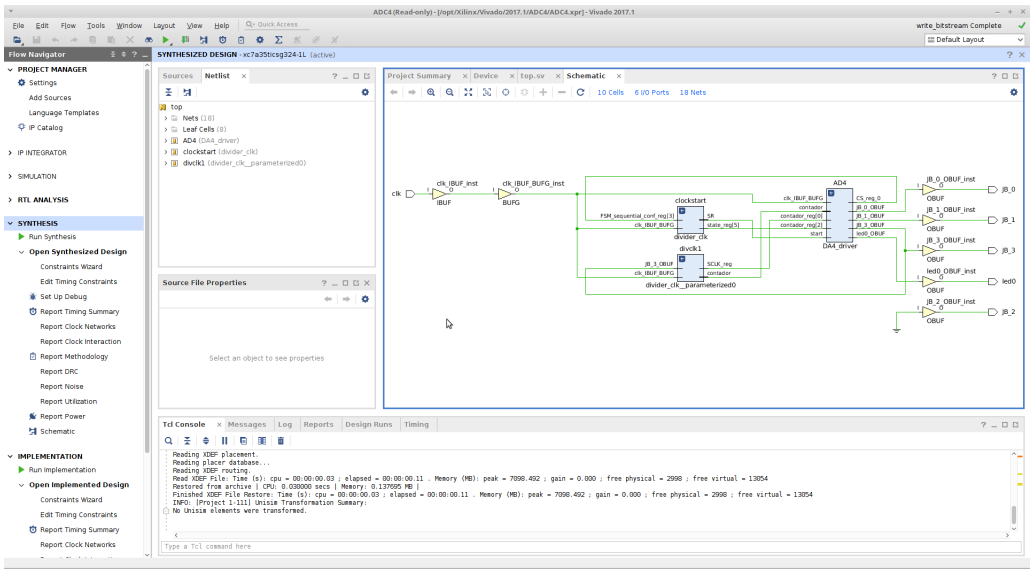

Figura 3.6: Síntesis de código

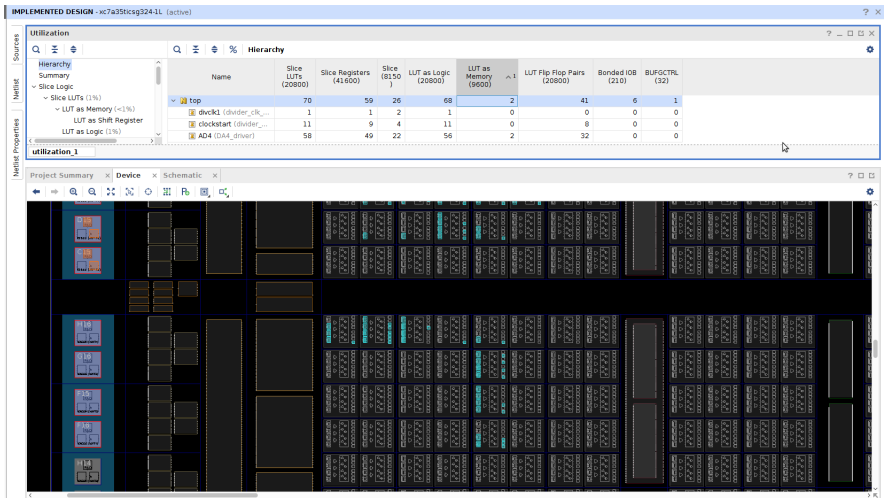

Figura 3.7: Implementación de código

Capítulo 4

Códigos

En este última sección se presentará el funcionamiento de los bloques programados a lo largo del semestre, estos bloques son implementados en conjuntos para generar programas más complejos.

4.1. Divisor de reloj

Este bloque de trabajo es sumamente útil, ya que es necesario para la coordinación de tareas a diferentes tiempos o tareas que ejecuten sub-rutinas a tiempos discretos. Su funcionamiento se basa en el conteo de ciclos o pulsos del reloj interno de la tarjeta para generar nuevos pulsos de reloj de mayor tiempo. En la Fig.(4.1) se muestra el esquema de funcionamiento del bloque para un divisor de 1 ciclo y bajo ella el código funcionamiento.

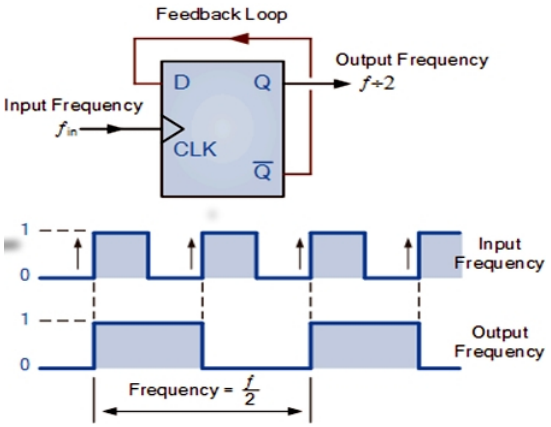

Figura 4.1: Divisor de reloj

```
1  `timescale 1ns / 1ps
2  module divider_clk(
3          //Entradas
4          clk,
5          //Salida
6          divclk
7           );
8  //---------- PARAMETERS ----------
9  parameter divisor = 10; // Parametro que divide la frecuencia del reloj,
       por defecto 10
10 // ---------- LOCAL PARAMETERS ----------
11 localparam nBits_div = $clog2(divisor) + 1;   //$clog2(N) indica el numero
       de bit necesarias para almacenar N
```

```
12 // ---------- INPUTS AND OUTPUTS ----------
13 input logic  clk;                          //define el reloj de entrad,
      reloj del sistema
14 output logic divclk;                       //se disena el valor de reloj
      de salida
15 // ---------- MODULE ----------
16 logic [nBits_div - 1: 0] contador=0; // " + 1" garantiza un registro de
      minimo 2 bits
17 assign divclk = (contador == (divisor - 1));    //asigna un el valor de 1
      o 0 al reloj de salida
18 always_ff @(posedge clk) begin       //secuencia contadora en cada paso
      del reloj
19     if (divclk)                                  // si el divCLK es 1 has
         lo siguiente
20         contador <= 0;                           // devuelve el contador a
            cero
21     else                                         //si divCLK es 0
22         contador <= contador + 1;                // suma uno al contador
23 end
24 endmodule
```

4.2. Conversor análogo digital

Este código es el encargado de la comunicación entre la tarjeta y un ADC, para la configuración de este código es necesario conocer al interfaz pre-programada en AD1, en este caso SPI. Esta interfaz precisa de 4 canales de comunicación, los que se muestran el Fig.(4.2), donde CS es el pulso proveniente desde el master que inicia la comunicación con Slave o le indica que se active, MOSI (Master Output Slave Input) corresponde a la salida de datos desde la tarjeta (Master) que son recibidos por el AD1 (Slave), MISO (Master Input Slave Output) es la salida de datos desde es slave que son resididos por Master y finalmente SCK (Clock) es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit. Luego de tener en mente este protocolo de comunicación, es necesario un sumo cuidado en los tiempo de sincronización utilizados, un error en estos se traduce en error de lectura de datos o simplemente la desconexión del sistema.
Ya establecida la comunicación entre el AD1 y la tarjeta es necesario concatenar los bits enviados por AD1 para generar el valor medido por él.
La configuración más óptima para la Arty 35t es usar dos divisores de reloj un de parámetro 100 para el CS y otro de parámetro 3 para SCK.

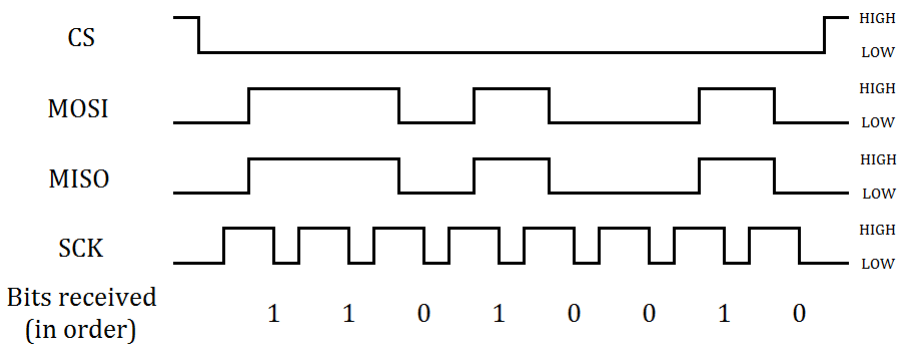

Figura 4.2: Sincronizacion de datos

```
1  `timescale 1ns / 1ps
2  module AD_driver(start, clk, divClk,ready, data0, data1,D0, D1, CS, SCLK);
3  // ---------- INPUTS AND OUTPUTS ----------
4  input logic start, clk, divClk;
5  output logic ready;
6  output logic [11:0] data0, data1;           // valores de salida
7  input logic D0, D1;                         // bits de entrada
8  output logic CS, SCLK;                      //  Comunicaci n , cuando leer
       y ciclos de lectura
9  // ---------- MODULE ----------
10 logic [3:0] state; // registro de estado
11 logic [10:0] data0_shift, data1_shift;      // memoria de paso
12 logic clkEn;                                //un reloj interno
13 always_comb clkEn = divClk & ~SCLK;
14 always_ff @(posedge clk) begin
15     if (start)
16         SCLK <= 1;
17     else if (divClk)
18         SCLK <= ~SCLK;
19 end
20 always_ff @(posedge clk) begin
21     if (start) begin
22         ready <= 0;
23         CS <= 1;
24         state <= 0;
25     end
26     else if (clkEn) begin       //  inicia la cuenta de los datos cada vez
           que clken este activo
27         if (state == 0) begin
28             CS <= 0;
29             state <= state + 1;
30         end
31         else if (state < 15) begin
32             data0_shift <= {data0_shift[9:0], D0};  //guarda los bits
33             data1_shift <= {data1_shift[9:0], D1};  //de entrada
34             state <= state + 1;
35         end
36         else begin                      //a qui llega el dato con 11 de
               sus 12 bits
37             data0 <= {data0_shift, D0}; //carga los ultimos 11 valores
                   leidos y agrega un ultimo valor
38             data1 <= {data1_shift, D1};
39             CS <= 1;                    //indica al adc que deje de enviar
                    senal de datos
40             ready <= 1;                 //indica que la lectura del datos
                   termino
41         end
42     end
43 end
44 endmodule: AD_driver
```

4.3. Conversor digital análogo de 8 salidas

Este equipo es capaz de generar señales a partir del envio de cadenas de bits, en estas se debe enviar primero una cadena de 4 bits que especifican el canal de salida de la señal y posterior a ello una cadena de 12 bits especificando el valor de salida de la señal. El código funciona de manera análoga al anterior, es decir, utiliza el protocolo SPI de comunicación, solo que esta vez los datos son enviados desde la tarjeta al dispositivo.
Los valores utilizados en los divisores de reloj implementados en la sincronización del sistema tomaron parámetros de 200 para CS y 2 para SCLK

```
1  `timescale 1ns / 1ps
2  module DA4_driver(              //se definen los par metros que entran al
       m dulo
3          //entradas
4          start, clk, divclk,
5          //salidas
6          ready,D0,
7          CS, SCLK
8  );
9  // Interfaz con FPGA:,
10 input logic         start, clk, divclk;      // Frecuencia m xima divClk:
     40MHz-----relojes
11 output logic        ready;                   //indicador de que esta listo
12 // Interfaz con PMOD AD1:
13 output logic CS, SCLK;                       //confecciones con el DAC,
     SYNC debe ser equivalente a CS
14 output logic D0;                             //se ales de datos
15 // ---------- MODULE ----------
16 logic [5:0] state;                           // registro de estado
17                                              // registros de desplazamiento
                                                    interno
18 logic [11:0] data1,data2,data3,data4,data5,data6,data7,data8;
19 logic [3:0] conf=0;                          //contador de se al de envi
20 logic [4:0] progra=0;                        //contador para la
     configuraci n
21 logic [31:0] data_shift;                     //registro de envi
22 logic clkEn;                                 //clock de envi
23 //----relojes de convercion de datos-----
24 assign clkEn = divclk & ~SCLK;               //definiendo clockEn
25 //dato enviado
26 assign D0    = data_shift[31];
27 // valores de las salidas
28 assign data1 = 12'b000001100010; // numero 98 ---> 0.06V en el pin data3
     problemas al usar overload
29 assign data2 = 12'b000000010000; // numero 16 ---> 0.01V en el pin data6
     problemas no usar
30 assign data3 = 12'b011100001010; // numero 1802 ---> 1.1V
31 assign data4 = 12'b000111101100; // numero 492 ---> 0.3V
32 assign data5 = 12'b001111010111; // numero 983 ---> 0.6V
33 assign data6 = 12'b010001111011; // numero 1147 ---> 0.7V
34 assign data7 = 12'b010100011111; // numero 1311 ---> 0.8V
35 assign data8 = 12'b010111000011; // numero 1475 ---> 0.9V
36 always_ff @(posedge clk) begin
```

```
37      if( start )
38          SCLK <= 1;
39      else if (divclk)
40          SCLK <= ~SCLK;
41  end
42  //Seleccion de envio de datos
43  //------registro de datos-----
44  always_ff @(posedge clk) begin
45  //configuracion de voltaje de referencia dac4
46  if (conf==0) begin
47              if (start) begin                          //si clock star es
                    1, pa asigna los valores que se enviaran
48                  data_shift[7:0]   <= 8'b0000001;      //registro de
                       datos no relevantes
49                  data_shift[19:8]  <= 12'b000000000000 ;//informaci n
                       enviada
50                  data_shift[23:20] <= 4'b0000;         //bit de
                       direcci n
51                  data_shift[27:24] <= 4'b1000;         //bit de orden
                       0000 es escribir
52                  data_shift[31:28] <= 4'b0000;         //bits no
                       relevantes
53                  ready <= 0;                           //estatus de la
                       conversi n
54                  CS  <= 1;                             //se al de
                       sincronizaci n si es 1 no convierte
55                  state <= 0;                           //contador de la
                       conversi n
56              end
57       //-----envio de datos--------
58              else if (clkEn) begin                     //clokEn activo
59                  if (state == 0) begin                 //inicio del
                       contador
60                      CS <=  0;                         //pulso de
                           sincronizaci n 0 inicia la conversi n
61                      state <= state + 1;               //suma de estado
62                  end
63       //entre un ciclo existe el tiempo suficiente para no caer en estados
            de meta-estabilidad
64                  else if (state < 33) begin            //conversi n de
                       vector de bits a bit usando desplazamiento a la
                       izquierda
65                      data_shift <= data_shift << 1;    //conversi n para
                            el puerto 1
66                      state <= state + 1;
67                  end
68                  else begin
69                      CS <= 1;                          //termina la
                           transferencia de valores
70                      ready <= 1;                       //indica que la
                           transformaci n termin
71                      conf <= conf+1;
72                  end
```

```
73              end
74       end
75      //envio a cada puerto
76  else if (conf==1) begin
77       if (start) begin                          //si clock star es 1,
            asigna los valores que se enviar n
78              data_shift[7:0]   <= 8'b00000000;  //registro de datos no
                relevantes
79              data_shift[19:8]  <= data1 ;       //informaci n a enviar
                enviada
80              data_shift[23:20] <= 4'b0000  ;    //bit de direcci n
81              data_shift[27:24] <= 4'b0011    ;  //bit de orden 0000 es
                escribir
82              data_shift[31:28] <= 4'b0000    ;  //bits no relevantes
83              ready <= 0;                        //estatus de la
                conversi n
84              CS  <= 1;                          //se al de
                sincronizaci n si es 1 no convierte
85              state <= 0;                        //contador de la
                conversi n
86          end
87  //-----envio de datos--------
88          else if (clkEn) begin                  //clokEn activo
89              if (state == 0) begin              //inicio del contador
90                  CS <=  0;                      //pulso de sincronizaci n
                        0 inicia la conversi n
91                  state <= state + 1;            //suma de estado
92              end
93  //entre un ciclo existe el tiempo suficiente para no caer en estados de
       meta-estabilidad
94              else if (state < 33) begin         //conversi n de vector de
                    bits a bit usando desplazamiento a la izquierda
95                  data_shift <= data_shift << 1;  //conversi n para el
                       puerto 1
96                  state <= state + 1;
97              end
98              else begin
99                  CS <= 1;                       //termina la transferencia
                        de valores
100                 ready <= 1;                    //indica que la
                       transformaci n termin
101                 conf <= conf+1;
102             end
103         end
104      end
105     .  //el c digo es demasiado extenso para ser escrito de forma
           completa
106     .  //los puntos suspensivos representan el envi  de consecutivo de
           datos a cada una de las salidas de forma crecientes
107     .
108     .
109 else if (conf==8) begin
110          if (start) begin
```

```
111             data_shift[7:0]   <= 8'b00000000;
112             data_shift[19:8]  <= data8      ;
113             data_shift[23:20] <= 4'b0111 ;
114             data_shift[27:24] <= 4'b0011    ;
115             data_shift[31:28] <= 4'b0000    ;
116             ready <= 0;
117             CS  <= 1;
118             state <= 0;
119          end
120          //-----envio de datos--------
121          else if (clkEn) begin
122              if (state == 0) begin
123                 CS <=  0;
124                 state <= state + 1;
125              end
126
127          else if (state < 33) begin
128              data_shift <= data_shift << 1;
129              state <= state + 1;
130          end
131          else begin
132             CS <= 1;
133             ready <= 1;
134             conf<=conf+1;
135          end
136        end
137    end
138 else if (conf==9) begin
139      if(progra<30)begin
140            conf <=0;
141            progra<=progra+1;
142        end
143        else begin
144            conf<=1;
145        end
146 end
147 end
148 endmodule
```

Capítulo 5

Conclusión

El desarrollo de tecnologías de procesamiento como la FPGA, arduino u otros, basados en la configuración o reconfiguración de sistemas digitales han permitido generar un sin fin de nuevas aplicaciones y facilitado su desarrollo. Si bien en el informe sólo se presentan los avances logrados en la configuración de periféricos, estos son la puerta de entrada al procesamiento digital, ya que todas las señales analógicas digitalizadas pueden ahora ser procesadas de esta forma, con lo que ahora sólo es necesario programas nuestro sistema para llevar acabo el análisis de los datos.
Como se logra apreciar la programación de este tipo de sistema presenta algunas complicaciones adicionales a la hora de la implementado, ya que además de manejar el lenguaje de programación (capitulo 3), resulta necesario conocer de antemano el funcionamiento y limitaciones de los componentes digitales que lo conforman (capítulo 2), ya que finalmente estos elementos son los programados. Una vez logrado el entendimiento de estos dos, la programación de este tipo de sistemas comienza a ser más familiar e intuitiva, lo cual representa un gran logro para el desarrollo de futuras aplicaciones.