Proyecto III: Multiplicador secuencial

Prof. Kaleb Alfaro Badilla
Dr.-Ing Pablo Mendoza Ponce

1. Introducción

El diseño de sistemas digitales requiere habilidad de implementación de algoritmos por medio de circuitos lógicos. Muchos algoritmos en la práctica usan iteraciones o búcles que a la hora de traducirlos a implementaciones de lógica booleana, surge la necesidad de un control lógico que habilite el correcto flujo de datos en circuito. Asimismo, las interfaces de bloque a bloque se diseñan con protocolos de bus para ayudar a estandarizar la comunicación entre unidades. Estos protocolos de bus facilitan las pruebas unitarias sobre bloques porque toda unidad se puede controlar de una manera similar.

En este proyecto busca introducir la implementación de algoritmos al estudiante, por medio del diseño de una unidad de cálculo de multiplicación. Y de la misma forma, esta unidad deberá respetar un protocolo de bus para su correcto funcionamiento.

2. Objetivo general

Introducir al estudiante a la implementación de algoritmos por medio de máquinas de estados complejas

3. Objetivos específicos

1. Elaborar una implementación de un diseño digital en una FPGA.
2. Construir un *testbench* básico para validar las especificaciones del diseño.
3. Implementar el algoritmo de Booth con una Máquina de estados con técnicas avanzadas.
4. Coordinación de trabajo en equipo mediante el uso de herramientas de control de versiones.
5. Practicar planificación de tareas para trabajo de grupo.

4. Especificación del diseño

Se solicita el desarrollo de una **unidad de multiplicación secuencial**. El mismo deberá construirse según las pautas fundamentales de diseño digital sincrónico. El circuito constará de tres bloques de constructivos o subsistemas:

1. Subsistema de lectura de datos.
2. Subsistema de cálculo de multiplicación.
3. Subsistema de despliegue de resultado.

4.1. Subsistema de lectura

1. Este bloque adquiere los operandos **A** y **B** de 8 bits cada uno para realizar la operación de multiplicación.
2. Los operandos **A** y **B** se interpretarán como enteros sin signo.
3. La entrada del código deberá ser capturada y sincronizada con el sistema principal por medio de un circuito antirebote de al menos 4 etapas (4 FF en cascada) por switch.
4. El circuito esperará ante el accionar de un *push button*.
5. El circuito **no iniciará otra operación** hasta que el **push button** vuelva a su estado inicial (cero).

4.2. Subsistema de cálculo de multiplicación

1. Este sistema recibe los operandos **A** y **B** del subsistema de lectura.
2. La operación de multiplicación se iniciará cuando el subsistema de lectura le indique a este subsistema que los operandos son válidos por medio de una señal de **inicio**.
3. El cálculo de multiplicación se realizará de manera iterativa por medio del Algoritmo de que se describe en la sección 5.
4. El bloque indicará que el resultado es valido por medio de una señal de listo (**ready**)
5. Debe de implementar este sistema siguiente la metodología de separar la ruta de control de la ruta de datos. De esta manera, el subsistema de cáculo correspondería al diagrama de la Figura 1.

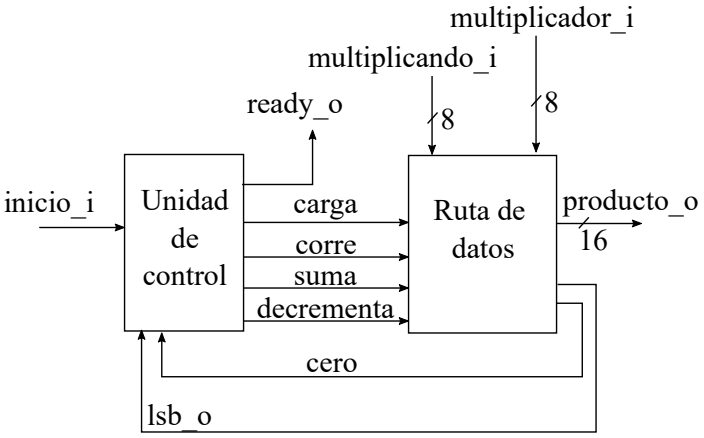
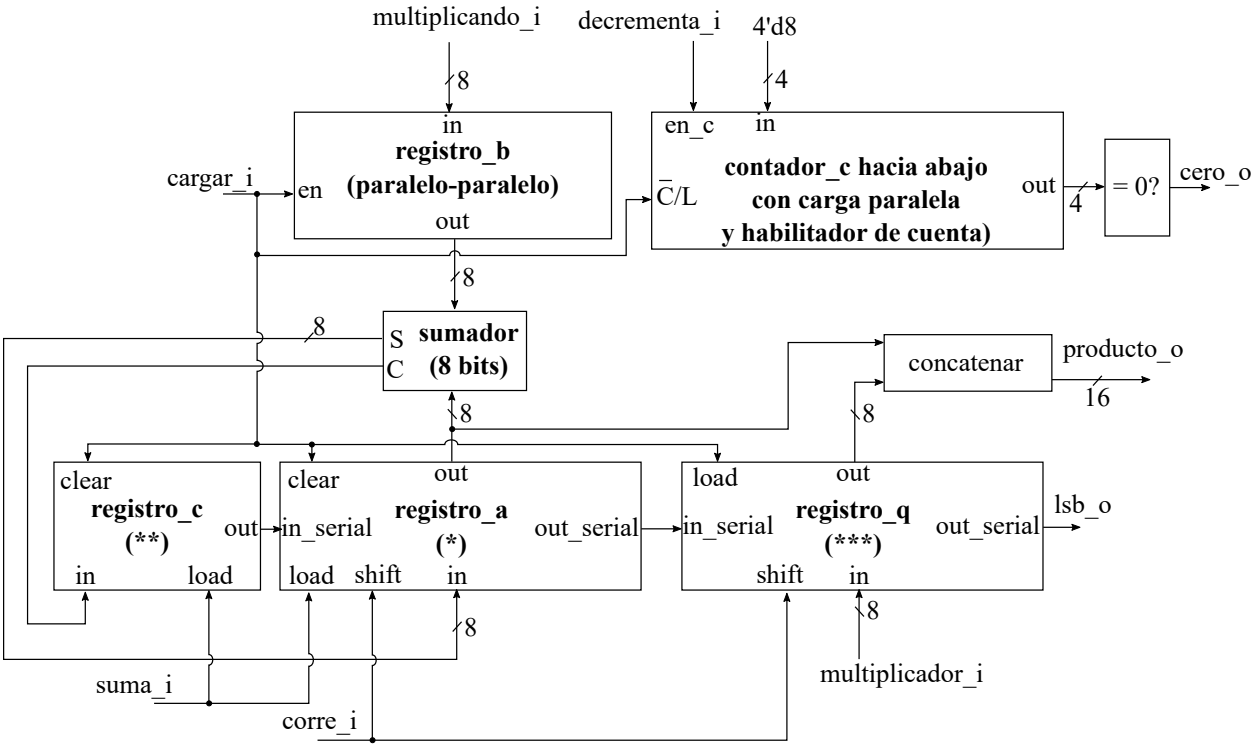


Figura 1: Diagrama de bloques del subsistema de cáculo de multiplicación.

4.3. Subsistema de despliegue

1. Al finalizar la operación, el sistema mostrará los resultados en los LEDs disponibles (16).
2. Lo que se muestra en los LEDs solamente se actualiza al finalizar una operación. No se debe mostrar ningún valor intermedio.



* carga y lectura paralela,con corrimiento a la derecha y entrada de clear (a parte de reset)

** carga y lectura paralela, con entrada de clear

*** igual que *, pero sin clear

Figura 2: Diagrama de bloques del circuito que implementa la ruta de datos para el algoritmo de Booth.

5. Algoritmo de Booth

El algoritmo de Booth es un método de cálculo iterativo de la operación aritmética de multiplicación. A diferencia de la implementación directa combinacional, este método requiere menos componentes combinacional (un solo sumador en comparación de N sumadores). Pero por ser un método iterativo su rendimiento es menor (llega a la respuesta después de varios ciclos de reloj).

El algoritmo lo puede leer explicado de este `http://vlabs.iitkgp.ernet.in/coa/exp7/index.html`. De esta manera, el diagrama se implementa siguiendo el diagrama presentado en Fig. 1. En el caso de la ruta de datos, la implementación debe seguir el diagrama de la Fig. 2.

En el caso de la unidad de control, esta debe ser implementada de dos maneras diferentes:

- Máquina de estados finita.
- Máquina microprogramada.

6. Guía de diseño

Los estudiantes deberán generar un diagrama de bloques para cada subsistema, con su funcionalidad descrita y su esquema de interconexión. Deberá presentarse adecuadamente la **ruta de datos** desde la

salida hasta la entrada, con una descripción comportamental de cada sub-bloque dentro de los subsistemas (i.e., muxes, decos, registros, etc.) similar a la Fig. 2. **No será necesario llevar la descripción a álgebra booleana**. Además, se deben plantear de forma adecuada los diagramas de flujo y estados para las máquinas de estados que implementan la ruta de control.

Cada subsistema deberá estar adecuadamente registrado a la entrada y salida. El flujo de datos debe ser indicado de manera explícita. No es necesario mostrar en detalle los bloques en que se implementen las máquinas de estados finitos (FSM) que se diseñen, pero sí sus diagramas de estado y las señales de control que manejan cada bloque en la ruta de datos.

Se debe elaborar la ruta de datos para el algoritmo de Booth como un bloque estructural que contenga las entradas y salidas (incluyendo reset y reloj) descritas en las Figuras 1 y 2. Este bloque debe ser verificado con un testbench para comprobar su funcionalidad (independientemente de la máquina de estados de control usada).

En una segunda etapa, se debe desarrollar la unidad de control usando una máquina de estados finita y conectarla a la ruta de datos anteriormente desarrollada. Se debe demostrar por medio de las simulaciones que el sistema completo cumple con la funcionalidad propuesta.

En una tercer etapa se sustituye la unidad de control (construida como una FSM), por una máquina microprograma. Junto con este instructivo se incluye un modelo de máquina microprogramada que puede ser utilizada. Desarrolle el microcódigo que ejecute la funcionalidad requerida. Se debe demostrar por medio de las simulaciones que el sistema completo cumple con el algoritmo de multiplicación.

6.1. Sobre la codificación del HDL

Se utilizarán **SystemVerilog** y el suite de herramientas de Vivado para desarrollar el sistema completo. Se seguirá el formato visto en clase para el estilo de codificación, cuyo apego por parte de los estudiantes **formará parte de la evaluación final.**

6.2. Sobre la verificación

Para cada subsistema, será obligatorio presentar simulaciones tanto a nivel de RTL (pre-síntesis) como con información de temporizado (post-síntesis y post-colocación-y-ruteo), las que serán evaluadas por el profesor de manera presencial en las fechas que se indicarán en el cronograma de la sección 7. Para ello, el grupo deberá generar las pruebas de banco o *testbenches* necesarios

6.3. Sobre el informe

Los estudiantes deberán presentar un informe en formato *Markdown* donde se las siguientes características del diseño final (el archivo se cargará como archivo `README.md` del repositorio):

1. Una descripción general del funcionamiento del circuito completo y de cada subsistema.
2. Diagramas de bloques de cada subsistema y su funcionamiento fundamental, según descritos en la sección 6.
3. Diagramas de estado de todas las FSM diseñadas (si existen), según descritos en la sección 6.
4. Ejemplo y análisis de una simulación funcional del sistema completo, desde el estímulo de entrada hasta el manejo de los 7 segmentos.
5. Análisis de consumo de recursos en la FPGA (LUTs, FFs, etc.) y del consumo de potencia que reporta la herramienta Vivado.
6. Análisis de principales problemas hallados durante el trabajo y de las soluciones aplicadas.

7. Evaluación (100 %)

Rubro	**Fecha de Entrega Límite**	**Valor** (%)	**Detalles**
Propuesta de plan de trabajo	18 de mayo	15	Se revisará un esquema de plan de trabajo con cada grupo personalmente sobre la plataforma del repositorio en GitHub (ver). Debe indicar quienes serán los integrantes responsables para cada tarea y cuales son sus especificaciones de forma clara (entradas, salidas, casos de prueba).
Avance	25 de mayo	35	Se revisará el cumplimiento del plan de trabajo realizado hasta el momento y se verificará al menos el funcionamiento en simulación RTL y post-síntesis. Problemas encontrados durante el transcurso del proyecto deben ser reportados en el plan de trabajo.
Defensa del proyecto	1 de junio	50	Cada grupo realizará una demostración del funcionamiento del diseño implementado en una placa de FPGA (Nexys 3, Nexys 4, Basys 3 o Nexys A7). Se revisará el cumplimiento de la especificación solicitada y del plan de trabajo. La mitad de este 50 % dependerá del informe adjunto que se monte en el repositorio (en el archivo `README.md` del proyecto, usando formato *Markdown*).

8. Apéndice A

A continuación se provee un código HDL en *Systemverilog* **como referencia** para la arquitectura del multiplicador **sin control**. Lo puede utilizar o editar a conveniencia.

```
typedef struct {
  logic load_A;
  logic load_B;
  logic load_add;
  logic shift_HQ_LQ_Q_1;
  logic add_sub;
} mult_control_t;

module mult_with_no_sm#(
  parameter N = 8
)(
  input logic clk,
  input logic rst,

  input logic [N-1:0] A,
```

```
  input logic [N-1:0] B,

  input mult_control_t mult_control,

  output logic [2] Q_LSB,
  output logic [2*N-1:0] Y
);

  logic [N-1:0] M;
  logic [N-1:0] adder_sub_out;
  logic [2*N:0] shift;
  logic [N-1:0] HQ;
  logic [N-1:0] LQ;
  logic Q_1;

  //reg_M

  always_ff@(posedge clk, rst) begin
    if(rst)
      M<='b0;
    else
      M <= (mult_control.load_A)? A : M;
  end

  // adder/sub

  always_comb begin
    if(mult_control.add_sub)
      adder_sub_out = M + HQ;
    else
      adder_sub_out = M - HQ;
  end

  //shift registers

  always_comb begin
    Y = {HQ,LQ};
    HQ = shift[2*N:N+1];
    LQ = shift[N:1];
    Q_1 = shift[0];
    Q_LSB = {LQ[0],Q_1};
  end

  always_ff@(posedge clk, rst) begin
    if(rst)
      shift <= 'b0;
    else if(mult_control.shift_HQ_LQ_Q_1)
      //arithmetic shift
      shift <= $signed(shift)>>>1;
    else begin
      if(mult_control.load_B)
```

```
            shift[N:1] <= B;
        if(mult_control.load_add)
            shift[2*N:N+1] <= adder_sub_out;
        end
    end

endmodule
```