Reporte de Proyecto de Laboratorio 1

Mauricio Caamaño, Marco Espinoza, Tomás González

***Abstract*—El presente reporte describe el desarrollo del primer proyecto de laboratorio del curso de Prototipado de FPGAs de la Maestría en Electrónica con énfasis en Sistemas Empotrados del Tecnológico de Costa Rica. Se utilizó el lenguaje de descripción de hardware Verilog para desarrollar elementos lógicos básicos: latch SR, un módulo aritmético y un módulo contador. Además, se implementó una máquina de estados con una aplicación de ejemplo, utilizando la herramienta de generación de máquinas de estado del software para desarrollo Quartus de Altera.**

***Index Terms*—FPGA, latch SR, máquinas de estado finito.**

I. Parte 1 - Elementos lógicos básicos

A. Descripción del diseño

Para la parte 1 se diseñaron elementos básicos, los cuales se describiran en detalle a continuación.

1) Latch SR: Para el diseño del Latch SR, se elaboró el diseño proveído en el enunciado del proyecto. La imagen 1, muestra el circuito diseñado.

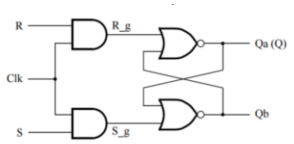

Figure 1. Latch SR implementado

Tal y como se aprecia en la figura, el diseño del latch consta de 2 compuertas lógicas AND y dos compuertas lógicas tipo NOR. Además, recibe como entradas un reloj llamado clk, y dos entradas llamadas Set (S) y Reset (R).
Debido a que se está implementando un latch, el cambio debe ocurrir durante el flanco, y no durante el disparo del reloj. La tabla de verdad del circuito es la que se muestra en la tabla 1.

Table I
Tabla de verdad para el latch SR

Entradas			Salidas		
S	**R**	**CLK**	**Qa**	**Qb**	
0	0	X	Qa	Qb	No hay cambio
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	?	?	No válido

2) Módulo aritmético: El módulo aritmético diseñado permite la suma y resta de dos variables de 16 bits, para ello se utiliza un bit de operación que indicará si se va a realizar una suma o una resta. Se tiene una salida de 16 bits que posee el resultado de la operación, y además se tiene de un bit de over-flow en caso de que el resultado de la operación realizada supere los 16 bits.
La imagen 2 muestra el diseño del circuito a diseñar, donde se puede apreciar que el bit de operación es el selector. Además en la tabla 2 observa la tabla de verdad del circuito implementado, donde se indica que para este diseño, si el bit de operación es 0, significa que la operación a realizar será una resta, por el contrario si el bit de operación es 1, indica que la operación a realizar será una suma. La variable resultado tendrá el valor obtenido de la operación, mientras que el bit de over-flow será siempre 0 a no ser que haya habido un overflow en la suma, o el resultado de la resta es negativo. Cabe recalcar que este circuito es combinacional, lo que quiere decir que no depende de ningún reloj para operar.

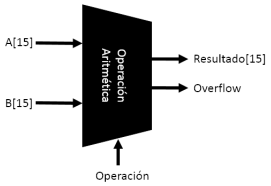

Figure 2. Diagrama del circuito de operación aritmética

Table II
Tabla de verdad para el circuito de operación aritmética

Entradas			Salidas	
A[15]	**B[15]**	**Operación**	**Resultado**	**Overflow**
A	B	0	A[15] - B[15]	0 / 1
A	B	1	A[15] + B[15]	0 / 1

3) Contador: El contador diseñado, tal y como se explica en el enunciado del proyecto debe de cumplir con las siguientes condiciones:

- Funcionar con el flanco positivo del reloj y poseer un reset síncrono, lo que quiere decir que el reset debe darse durante el flanco positivo del reloj en caso de que se diera la condición.
- Longitud paramétrica con un valor por defecto de 8 bits, lo que implica que el tamaño de la salida del contador diseñado, será de 8 bits.
- Selección para conteo hacia adelante o hacia atrás, es decir, se tendrá un bit que determinará si el conteo es ascendente o descendente.
- Opción de habilitar/deshabilitar el conteo, osea, que se tendrá también un bit de habilitación, que en caso de que no esté seteado, significará que el contador se mantendrá

en la posición actual hasta que este bit se habilite nuevamente.

Basado en estas condiciones, se diseñó el contador utilizando la tabla de verdad que se muestra en la tabla 3. Como se muestra en la tabla, se tienen 3 entradas además del reloj, si el reset es 1, entonces el contador es reseteado, si la variable enable está en 0, el contador va a mantener el valor actual y no va a cambiar hasta que esta variable tome el valor de 1, una vez que la variable de habilitación esta en 1, si la variable up_down está en 1, el contador va a contar de manera ascendente, por el contrario si la variable up_down tiene el valor de 0, el contador va a contar de manera descendente.

Table III
TABLA DE VERDAD PARA EL CONTADOR DISEÑADO

Entradas			Salidas
Reset	**Up_Down**	**Enable**	**Contador**
0	0	0	Contador
0	0	1	Contador-1
0	1	0	Contador
0	1	1	Contador + 1
1	X	X	0

B. Consumo de recursos FPGA

Para el reporte de consumo de recursos, se utilizó como referencia el sumario de utilización del Quartus, implementando el diseño en el dispositivo *Cyclone V: 5CSEMA5F31C6*.

Table IV
CONSUMO DE RECURSOS DE LA IMPLEMENTACIÓN DE LA MÁQUINA DE ESTADOS FINITA

Recurso	Utilización	Porcentaje
ALMs	15 / 32.070	1 %
LABs total	4 / 3.207	1 %
ALUTs combinacionales	30	
Registros dedicados	8 / 64.140	1 %
Pinos E/S	66 / 457	14 %
Relojes globales	1 / 16	6 %

C. Resultados obtenidos

1) Latch SR: A continuación, en la figura siguiente se muestra el resultado del test bench para el latch SR.

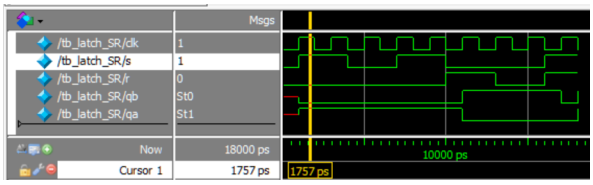

Figure 3. Simulacion del circuito para el Latch SR

Los resultados mostrados en la figura, validan la tabla de verdad mostrada en la tabla 1, la cual debe ser cumplida por el latch SR. Se puede observar que inicialmente el valor de la entrada S es 1 y la entrada R es 0, por lo que la salida qa debe de tener el valor de 1. Cuando el valor de ambas entradas es 0, el valor tanto de qa como de qb se mantiene en el valor anterior, tal y como se aprecia en la figura. Una vez que la entrada R cambia a 1 y la entrada S cambia a 0, se puede ver como la salida qa cambia al valor de 0. Y por último, cuando ambas entradas es 1, el valor de salida se indefine, ya que este es el estado prohibido para el latch SR. Por lo tanto, se puede confirmar que el circuito diseñado para el latch SR funciona correctamente.

2) Módulo aritmético: El resultado de la simulación del circuito aritmético se muestra en la figura siguiente.

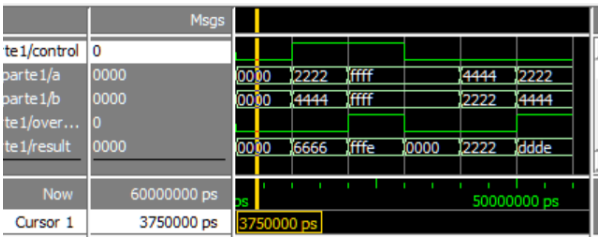

Figure 4. Simulación del circuito aritmético

Como se observa en a figura, cuando la señal de control pasa al valor de 1, quiere decir que la operación a realizar es una suma. Por lo que cuando la entrada A tiene el valor de 0x2222 y la entrada B el valor 0x4444, el resultado obtenido es 0x6666 y la variable overflow se encuentra en 0x0. Además, cuando ambas entradas tienen el valor de 0xFFFF se puede observar como la salida de overflow pasa a 1, y el resultado obtenido es 0x1FFFE tomando en cuenta el overflow, que demuestra que el circuito está trabajando de manera adecuada. Una vez que la variable de control cambia a 0, se puede observar como cuando ambas entradas tienen el valor de 0xFFFF, el resultado obtenido es 0x0, y además cuando el valor de B es mayor que el valor de A, el resultado obtenido es negativo y la variable de overflow tiene el valor de 1. Por lo que con estas pruebas se puede demostrar que el circuito aritmético está funcionando como se espera.

3) Contador: La siguiente figura tiene el resultado del testbench utilizado para validar el contador diseñado para esta sección. En la figura, se puede apreciar que cuando el valor

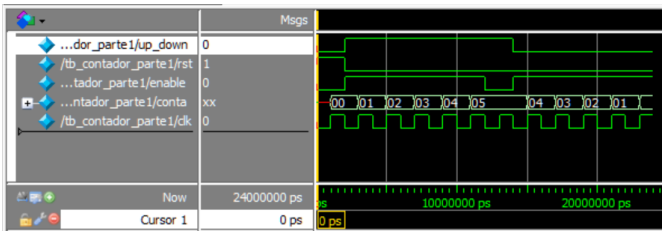

Figure 5. Simulación del contador diseñado

del reset es 0x1, la salida contador se reinicia al valor de 0x0, una vez que el valor de reset retorna al valor de 0x0 y el valor de enable es 0x1, el contador empieza a contar de

manea ascendente debido a que la variable up_down tiene el valor de 0x1. Una vez que la variable enable es 0x0, se puede observar como el contador mantiene el valor, y cuando la variable enable cambia a 0x1 nuevamente y la variable up_down cambia a 0x0, se puede observar como el contador empieza a decrementar. Con estas condiciones se comprueba que el contador cumple con lo establecido en el diseño solicitado.

II. Parte 2 - Máquina de estados

A. Descripción del diseño

Para esta parte se creó una máquina de estados que controla la entrada y salida de autos de un estacionamiento. Dos sensores detectan cuando un auto pasa a través de ellos. El criterio para que un auto cuente como que entró o salió es que primero un sensor se active, luego ambos se activen, luego el primer sensor que se activó se desactive y finalmente que el segundo sensor de desactive. Dependiendo de cuál sensor se active primero se sigue un flujo en la máquina de estados que lleva a decir si el auto entró o salió.

Para esta máquina de estados se crearon los siguientes estados:

- `idle`
- `sensor1_activated`
- `sensor2_activated`
- `both_sensors_active1`
- `both_sensors_active2`
- `sensor1_deactivated`
- `sensor2_deactivated`
- `car_entered`
- `car_exited`
- `error`

Se empieza en el estado `idle`. Si el sensor 1 se activa se pasa al estado `sensor1_activated`. Luego, si el sensor 1 se desactiva se devuelve a idle pero si el sensor 2 se activa se pasa a `both_sensors_active1`. Acá, si el sensor 2 se desactiva se devuelve a `sensor1_activated` y si el sensor 1 se desactiva se pasa a `sensor1_deactivated`. En esta caso si el sensor 1 se reactiva se devuelve a `both_sensors_active1` y si el sensor 2 se desactiva se pasa a `car_entered` y se finaliza el ciclo. El caso en el que el carro sale es parecido, simplemente se intercambian el sensor 1 y el sensor 2. El estado `error` es para cuando hay alguna operación inesperada, como por ejemplo que estando en un estado con ambos sensores activos, de pronto ambos sensores se desactiven al mismo tiempo.

B. Consumo de recursos FPGA

Para el reporte de consumo de recursos, se utilizó como referencia el sumario de utilización del Quartus, implementando el diseño en el dispositivo *Cyclone V: 5CSEMA5F31C6*.

C. Resultados obtenidos

Con el fin de comprobar la lógica desarrollada para la máquina de estados, se creó un archivo *testbench* que describe los estímulos necesarios para simular los diferentes eventos a los cuales el sistema debe responder. Esto se realizó asignando valores a las entradas del sistema, en este caso los sensores, llamados `s1` y `s2`, y observando el estado de las salidas para controlar las puertas de acceso `enter` y `exit`. Además, el registro de conteo de cantidad de autos en el parqueo también se observó como salida mediante el registro `cnt`. Tal como las entradas de los sensores, en el *testbench* también se definieron las señales de control de reloj y reset. Para simular el reloj, se definió la entrada `clk` con la primitiva `always` que permite asignar un valor periódico.

Table V
Consumo de recursos de la implementación de la máquina de estados finita

Recurso	Utilización	Porcentaje
ALMs	42 / 32.070	1 %
LABs total	6 / 3.207	1 %
ALUTs combinacionales	81	
Registros dedicados	10 / 64.140	1 %
Pinos E/S	32 / 457	8 %
Relojes globales	1 / 16	6 %

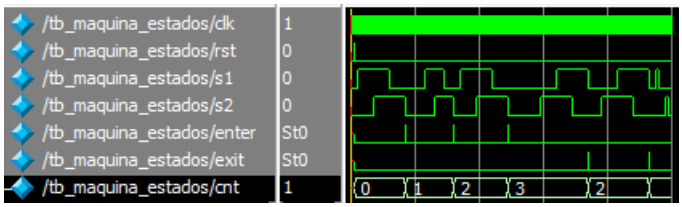

Figure 6. Simulación de la máquina de estados

Tal como se muestra en la figura 6, se estimuló el circuito para simular la entrada de tres automóviles, encendiendo los sensores 1 y 2 consecutivamente y luego apagándolos, por cada automóvil. Cada evento también genera un pulso de la salida `enter`. Seguidamente se simula la secuencia contraria, indicando la salida de un automóvil, lo cual activa un pulso de la salida `exit`. La variable `cnt` muestra el número de automóviles. Finalmente, se observa cómo por la activación individual de cada sensor, simulando el paso de un peatón, no se genera ningún pulso de entrada o salida, ni se modifica el contador de automóviles.

III. Conclusiones

- Se diseñaron circuitos tanto combinacionales como secuenciales y los mismos fueron validados diseñando su respectivo *testbench*
- Para la elaboración del latch SR, se utilizaron funcionas lógicas y se demostró, a través de su *testbench*, la zona prohibida cuando ambas entradas S y R tienen el valor de 1.
- Se diseñó y se comprobó el funcionamiento de la máquina de estados realizada con ayuda de la herramienta de generación para máquinas de estado finitas.