Universidad de Costa Rica
Escuela de Ingeniería Eléctrica

Circuitos Digitales II - IE523
Profesor Jorge Soto

Tarea #2

Emilio Javier Rojas Álvarez
B15680

19 de agosto de 2017

Resumen

Un registro de desplazamiento es un dispositivo que almacena una serie de bits, tiene la capacidad de desplazar los bits hacia la izquierda o derecha. Al dispositivo se le pueden agregar funcionalidades adicionales como cargarle un valor deseado o insertarle valores desde afuera. Estas tres funcionalidades son consideradas en este trabajo. En total se presentan dos registros de desplazamiento, uno de 4 bits y el otro de 32 bits, implementado con módulos de registros de 4 bits. Al primer registro se le aplican pruebas para probar cargas en serie(se le insertan bits desde algún lado), rotaciones circulares, y carga en paralelo(se insertan todos los bits al registro).

La comprobación del caso de 4 bits permite dar una certeza del funcionamiento de estos inclusive cuando se combinen entre sí. Resultó de utilidad separar las funciones del módulo de 4 bits(que es realmente el registro de desplazamiento en ambos casos) y el de 32 bits(encargarse de interconectar correctamente los módulos de 4bits y actualizar su estado interno cuando sea pertinente).

Descripción Arquitectónica

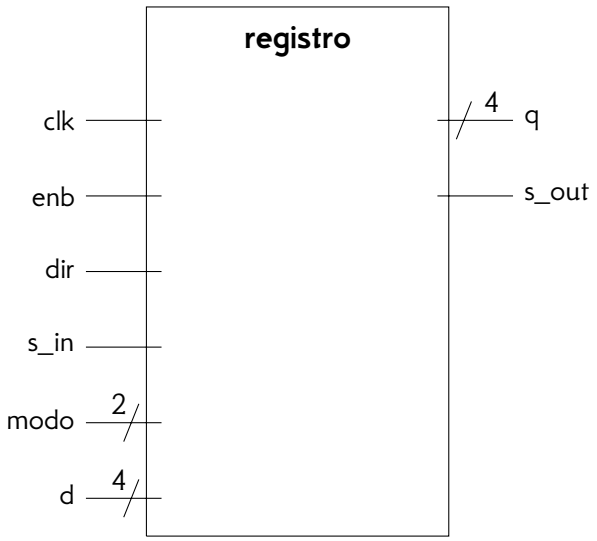

Figura 1: Módulo registro.

Inicialmente se implementa el módulo `registro`, en la figura 1 se muestra el bloque correspondiente a este módulo, indicando a la izquierda sus entradas, y a la derecha sus salidas.

Posteriormente se unen 8 de estos módulos para formar un registro de 32 bits. Las conexiones que unen los registros de 4 bits son seleccionadas mediante sentencias condicionales(pueden verse como MUXES).

Plan de Pruebas

Se han realizado un total de 5 pruebas para el registro de 4 bits y una prueba para el registro de 32 bits:

- `registro/test1`: Esta prueba se encarga de verificar el funcionamiento en modo de carga en serie hacia la izquierda.
- `registro/test2`: Esta prueba se encarga de verificar el funcionamiento en modo de carga en serie hacia la derecha.
- `registro/test3`: Esta prueba se encarga de verificar el funcionamiento en modo de rotación circular hacia la izquierda.
- `registro/test4`: Esta prueba se encarga de verificar el funcionamiento en modo de rotación circular hacia la derecha.
- `registro/test5`: Esta prueba se encarga de verificar el funcionamiento en modo de carga en paralelo.
- `registro32bits/test1`: Esta prueba se encarga de verificar el funcionamiento del registro de 32 bits con distintas pruebas, pasando por todos los modos de funcionamiento.

Instrucciones Utilizadas en la Simulación

Este proyecto ha sido desarrollado en Ubuntu 16.04, el entorno de trabajo ha sido pensado para este sistema operativo, sin embargo debería funcionar en distribuciones similares.

Para la generación de los archivos requeridos se agrega la utilidad `build.sh`. Al ser llamado de la manera[1]:

```
./build.sh
```

realiza la compilación(`iverilog`) y los tests(`vvp`) para cada test de los dos módulos existentes, `registro` y `registro32bits`. Debido a que la herramienta GTKWave permite recargar un archivo, no hace falta abrir esta herramienta cada vez que se generan los archivos `.vcd`, por lo que esta manera de compilación resulta cómoda durante el proceso de desarrollo.

La utilidad mencionada recibe los siguientes parámetros:

- `-c`: bandera de compilación, indica que se compilan las pruebas(`iverilog`).
- `-t`: bandera de testeo, indica que se testean las pruebas(`vvp`).
- `-v`: bandera de vista, indica que se muestran una a una las señales de onda generadas(`gtkwave`).

Como se puede observar de lo anterior, `.build.sh` es lo mismo que `.build.sh -c -t`. Para un chequeo de todas las pruebas resulta útil el comando:

[1] Requiere ser ejecutable, en caso de no serlo: `chmod +x build.sh`

```
./build.sh -c -t -v
```

que compila, y muestra todas las pruebas una tras otra.

En caso que se quiera ejecutar una única prueba, se ejecuta lo siguiente:

```
mkdir -p build
mkdir -p tests
iverilog -o build/<modulo>/<prueba> pruebas/<modulo>/<prueba>.v <dependencias>
vvp build/<modulo>/<prueba>
gtkwave tests/<modulo>/<prueba>.gtkw
```

Por ejemplo para el módulo `registro32bits`, la prueba `test1`:

```
mkdir -p build
mkdir -p tests
iverilog -o build/registro32bits/test1 pruebas/registro32bits/test1.v modulos/
    registro.v modulos/registro32bits.v
vvp build/registro32bits/test1
gtkwave tests/registro32bits/test1.gtkw
```

Los dos primeros comandos crean directorios para los archivos de salida en caso que no existan. Los últimos tres comandos pueden ser utilizados de manera separada según sean las necesidades, notando que `vvp` requiere que se haya corrido `iverilog`, y `gtkwave` que se haya corrido `vvp`.

Ejemplos de los Resultados

A continuación se muestra lo que el usuario puede esperar al correr los tests(`vvp`) en la terminal, o bien al abrir la heramienta GTKWave(`gtkwave`). Para todos los casos de ondas resulta sencillo observar los patrones descritos en las pruebas.

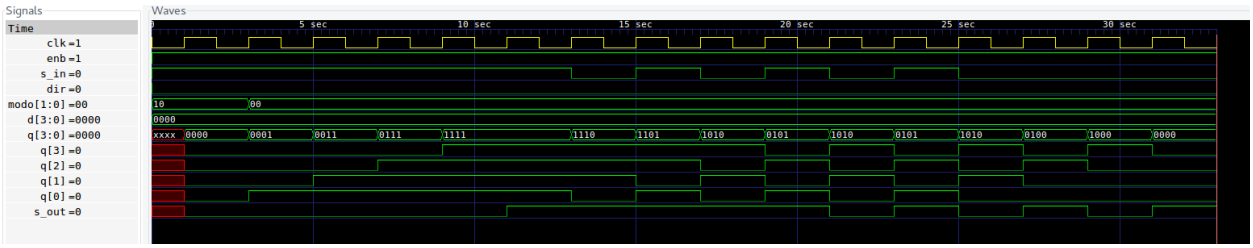

Figura 2: Visualización del `test1` para `registro` en GTKWave.

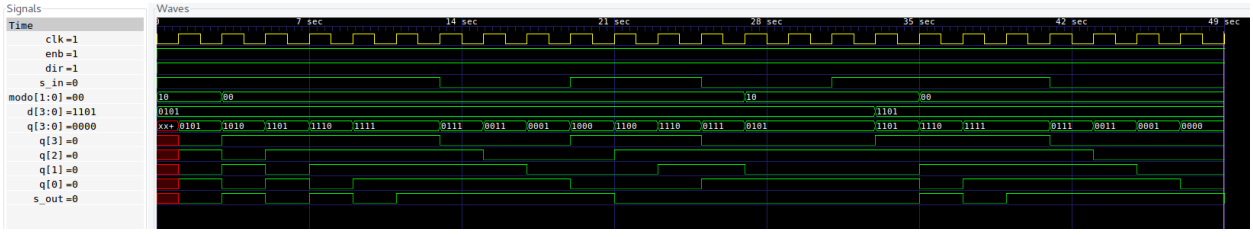

Figura 3: Visualización del `test2` para `registro` en GTKWave.

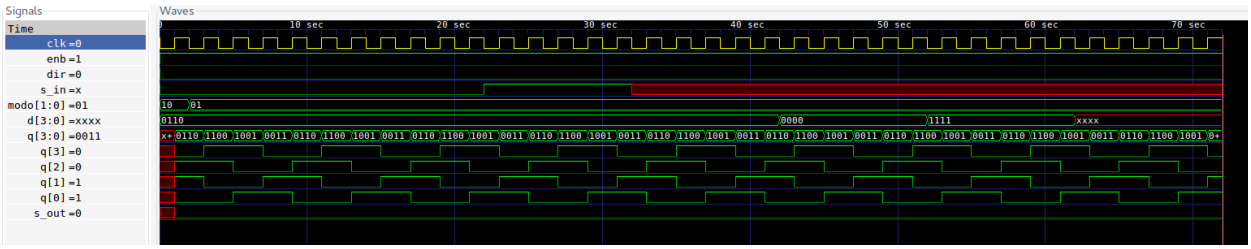

Figura 4: Visualización del `test3` para `registro` en GTKWave.

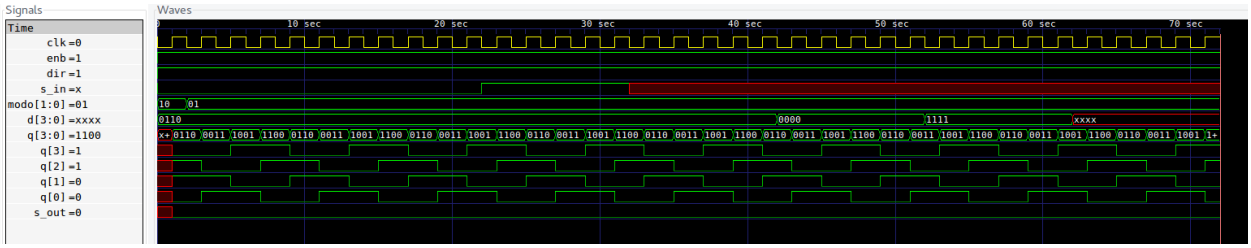

Figura 5: Visualización del `test4` para `registro` en GTKWave.

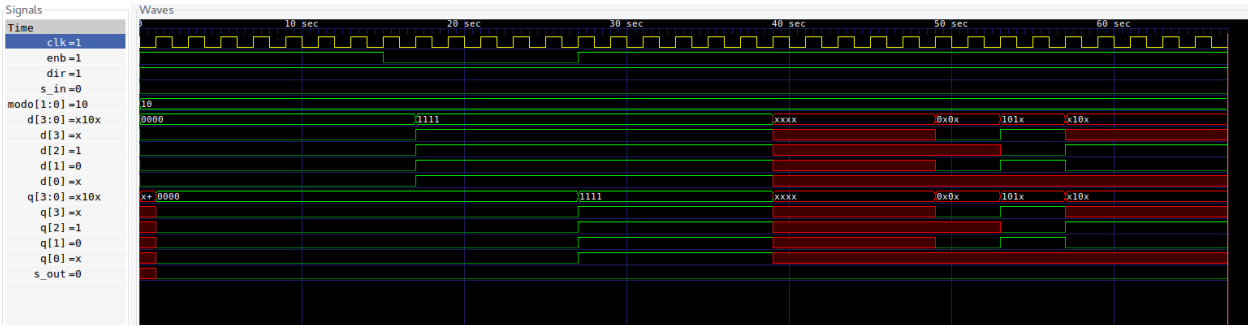

Figura 6: Visualización del `test5` para `registro` en GTKWave.

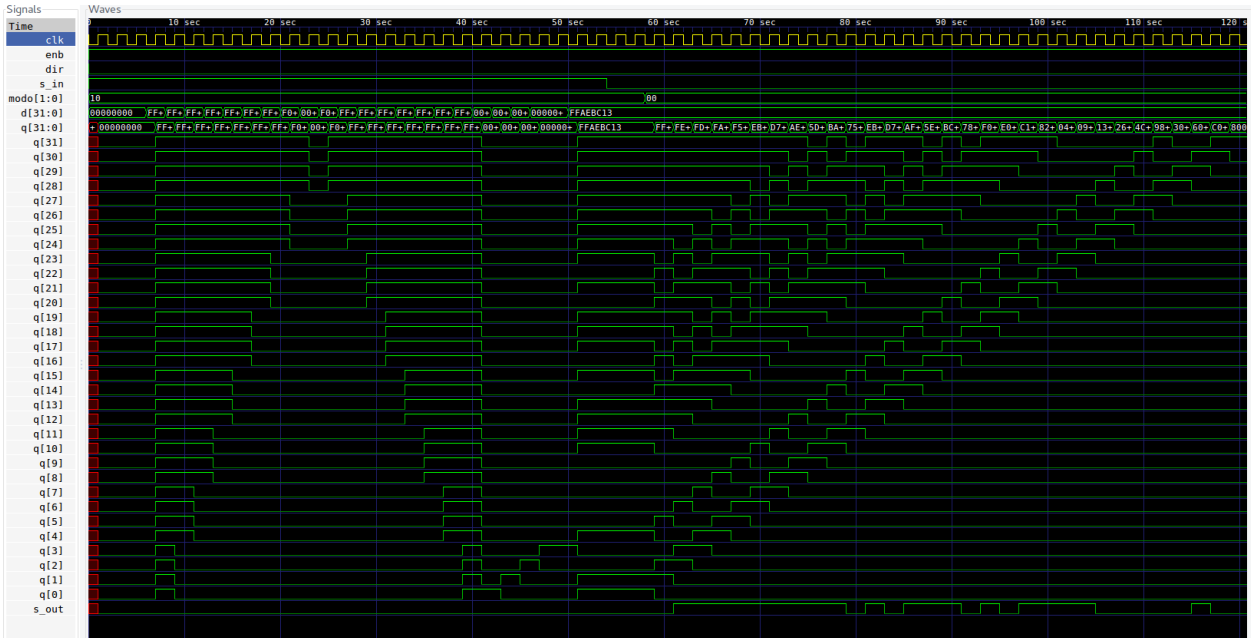

Figura 7: Visualización del `test1` para `registro32bits` en GTKWave.

```
$ ./build.sh -c -t -v
** Modulo modulos/registro.v
**********************
Testbench test1
**********************
iverilog -o build/registro/test1 pruebas/registro/test1.v modulos/registro.v modulos/registro32bits.v

vvp -M ~/.local/install/ivl/lib/ivl build/registro/test1

Test Bench 1 registro
Prueba la funcionalidad de carga en serie a la izquierda
********************************************************
VCD info: dumpfile tests/registro/test1.vcd opened for output.
               tiempo  dir     s_in    modo    d       q       s_out

                  0    0       1       10      0000    xxxx    x
                  1    0       1       10      0000    0000    0
                  3    0       1       00      0000    0001    0
                  5    0       1       00      0000    0011    0
                  7    0       1       00      0000    0111    0
                  9    0       1       00      0000    1111    0
                 11    0       1       00      0000    1111    1
                 13    0       0       00      0000    1110    1
                 15    0       1       00      0000    1101    1
                 17    0       0       00      0000    1010    1
                 19    0       1       00      0000    0101    1
                 21    0       0       00      0000    1010    0
                 23    0       1       00      0000    0101    1
                 25    0       0       00      0000    1010    0
                 27    0       0       00      0000    0100    1
                 29    0       0       00      0000    1000    0
                 31    0       0       00      0000    0000    1
                 33    0       0       00      0000    0000    0
gtkwave tests/registro/test1.gtkw


GTKWave Analyzer v3.3.83 (w)1999-2017 BSI

[0] start time.
[33] end time.
WM Destroy
Testbench test2
**********************
iverilog -o build/registro/test2 pruebas/registro/test2.v modulos/registro.v modulos/registro32bits.v

vvp -M ~/.local/install/ivl/lib/ivl build/registro/test2
```

Figura 8: Salida en la terminal al ejecutar `./build.sh -c -t -v`.

Conclusiones

El trabajo ha sido en gran medida investigativo, así como de prueba y error, esto ha sido provechoso puesto que conforme se avanzó en el proyecto, resultó cada vez más sencillo identificar errores recurrentes y encontrar soluciones a nuevas dificultades. Se ha dedicado gran parte del trabajo a la lectura de documentación tanto oficial como externa de las distintas herramientas utilizadas, se espera que en futuros trabajos encontrar la información se pueda realizar de manera más rápida dado que ya se sabe donde y como encontrarla.

Modelar y resolver el problema mediante máquinas de estado es posible, y quizás no represente gran dificultad, sin embargo es un proyecto adecuado para comprender conceptos clave a la hora de describir un dispositivo. La jerarquía de módulos, en la mínima manera que se aplicó, muestra lo escalable que puede ser verilog para el diseño de dispositivos de alta complejidad, y el testeo de unidades individuales facilita la interacción de módulos previamente creados, permitiendo, si fuera el caso, que varios diseñadores trabajen en un mismo dispositivo de manera simultanea.

Recomendaciones

El proyecto no se estructuró teniendo en mente utilizar la herramienta `make`, al inicio se utilizó un script para la compilación, a medida que crecía el proyecto se le agregaban nuevas funcionalidades. A la hora de intentar migrar el script a `make` se encontraron dificultades, y se dedicó tiempo para investigación sobre esta herramienta. Si bien la herramienta propuesta funcione muy bien para el proyecto, se pretende en próximas entregas utilizar `make` para adherirse a los estándares de compilación.

Para actualizar la información interna del registro de 32 bits(esto es, la interconexión de los submódulos) se utiliza el flanco negativo del reloj, esto con el fin de tener en el flanco positivo el estado más reciente del registro. No se ha investigado sobre el impacto que pueda tener esto en el sistema, o a la hora de utilizar el módulo `registro32bits` como un submódulo. Para la presente entrega funciona de manera muy satisfactoria.

Contabilización de tiempos

Descripción	**Tiempo**
Lectura y comprensión de la especificación	20 minutos
Modelado inicial del proyecto en papel	40 minutos
Investigación inicial	90 minutos
Descripción de módulo `registro`	60 minutos
Descripción de módulo `registro32bits`	100 minutos
Creación de pruebas	140 minutos
Creación de script para compilación	100 minutos
Pruebas y corrección de descripciones (Esto incluye la búsqueda de soluciones a problemas.)	300 minutos
Confección del reporte y presentación (Incluyendo generación de reultados e imagenes)	160 minutos
Total	1010 minutos (16 horas y 50 minutos)

Cuadro 1: Contabilización de tiempos