Actividad Previa - Diseño de Procesadores

Multiplicador de Booth

---

1. Índice

---

2. Diseño:

El circuito representa un multiplicador de Booth de 3 bits donde los registros A y Q han sido extendidos en 1 bit para evitar errores de overflow.

- uc.v - unidad de control
- datapath.v - camino de datos
- mult.v - circuito multiplicador de Booth

La unidad de control en su diseño final hace uso de 8 estados **[Figura 4]** incluyendo el inicial y final. El diseño de estos permite que se reduzca el número de estados visitados saltando directamente a los estados de desplazamiento cuando la suma o resta no sean necesarias.

2.1. Diseño base

El diseño base propuesto se conforma de 8 estados los cuales se ejecutan de forma secuencial por lo tanto realizar la operación de multiplicación sobre todas las combinaciones posibles de números en complemento 2 de 3 bits tardaría ***512*** **ciclos**. Estos se podrían reducir si evitáramos realizar saltos a estados de suma o resta cuando no sean necesarios.

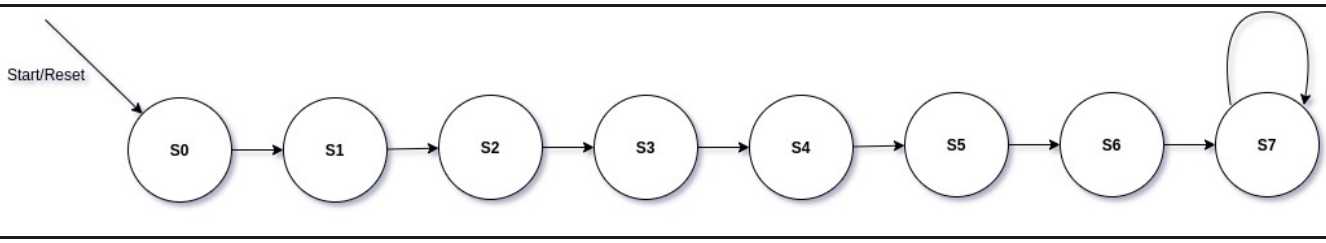


Figura 1: *Estados de la unidad de control base*

2.2. Primera iteración

Partiendo de la suposición de que evitar las sumas y restas produce una mejora se creó una nueva máquina de estados. Como se puede observar ya no es secuencial sino que podemos evitar las sumas/restas. Sin embargo la modificación requiere añadir estados donde se selecciona el siguiente salto.

Debido a esto el nuevo número de estados es 11 y niegan la posible mejora al tener que ejecutarse los nuevos estados en cada iteración dando como resultado un total de ***544*** **ciclos** y una **aceleración de** ***0.94*** respecto al diseño base.

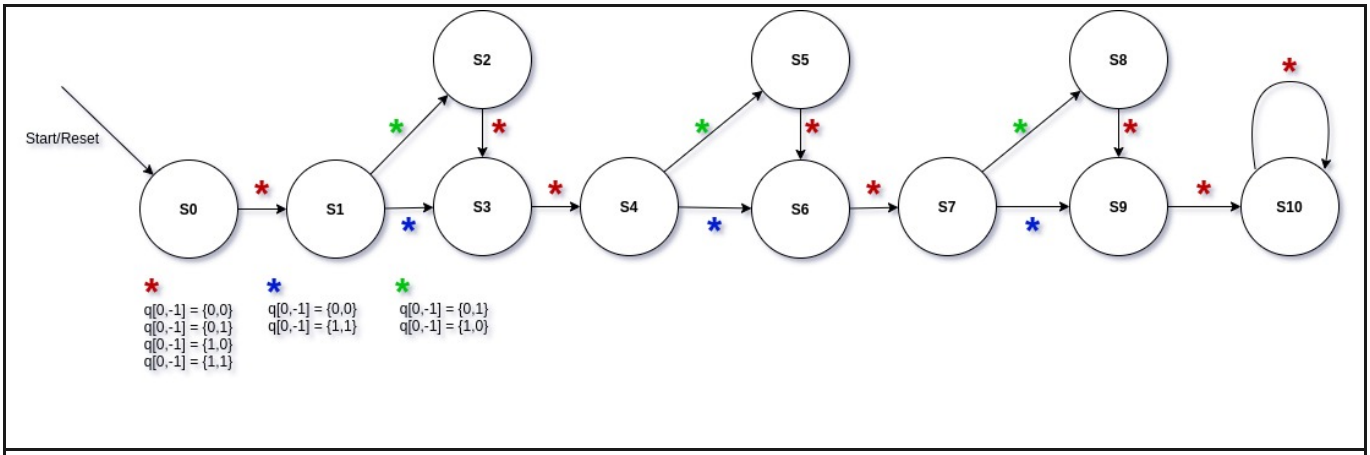


Figura 2: *Estados de la unidad de control tras la primera iteración*

2.3. Segunda iteración

La única solución a la regresión de la primera iteración del diseño es realizar el desplazamiento y la selección del siguiente estado en el propio estado de desplazamiento.

A pesar de ello esto resulta imposible ya que los registros son de carga síncrona y no estarán actualizados hasta la siguiente señal de reloj. La solución es predecir el siguiente estado antes de que los registros se actualicen. Para ello la unidad de control recibe los bits ***q[1:0]*** y ***q[-1]*** y usa ***q[1:0]***.

q[-1] sigue siendo necesario para la elección de estado en ***S1***.

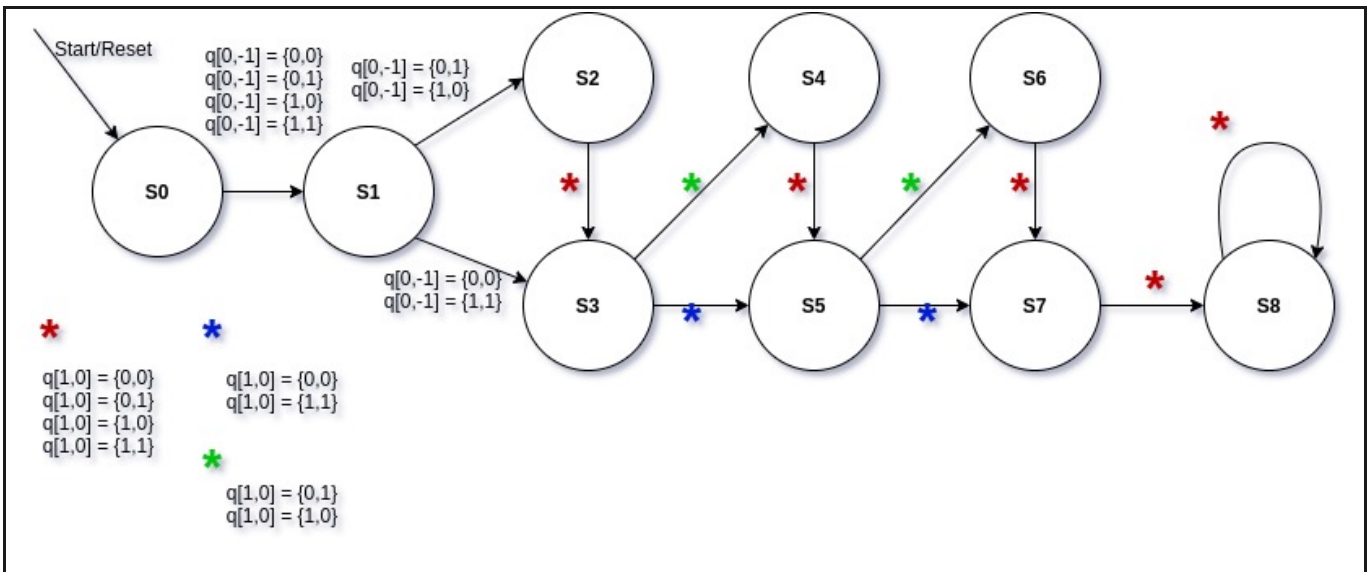


Figura 3: *Estados de la unidad de control tras la segunda iteración*

Con estos cambios se elimina el overhead de los estados extra y obtenemos la ventaja que intentábamos obtener. Los nuevos ciclos para terminar las pruebas son ***416*** lo que supone una aceleración de ***1.231*** respecto al diseño base y ***1.307*** respecto a la primera iteración.

2.4. Tercera iteración

Si bien ya se ha conseguido una mejora mayor a 1 aún se puede mejorar el diseño eliminando el estado ***S1*** de la iteración previa. Para ello sería necesario obtener ***q[0]*** desde el testbench (***q[-1]*** en este caso es insignificante ya que siempre será 0) cuando el estado actual sea ***S0***. Podemos reutilizar la señal ***init*** de la *UC* para realizar esta tarea.

```
//fichero mult.v
//Q_mult: numero recibido desde el testbench
//q_uc: señal que recibe la UC
//q: q[1:-1] Salida del registro Q
assign q_uc = init ? {Q_mult[1:0], 1'b0} : q;
```

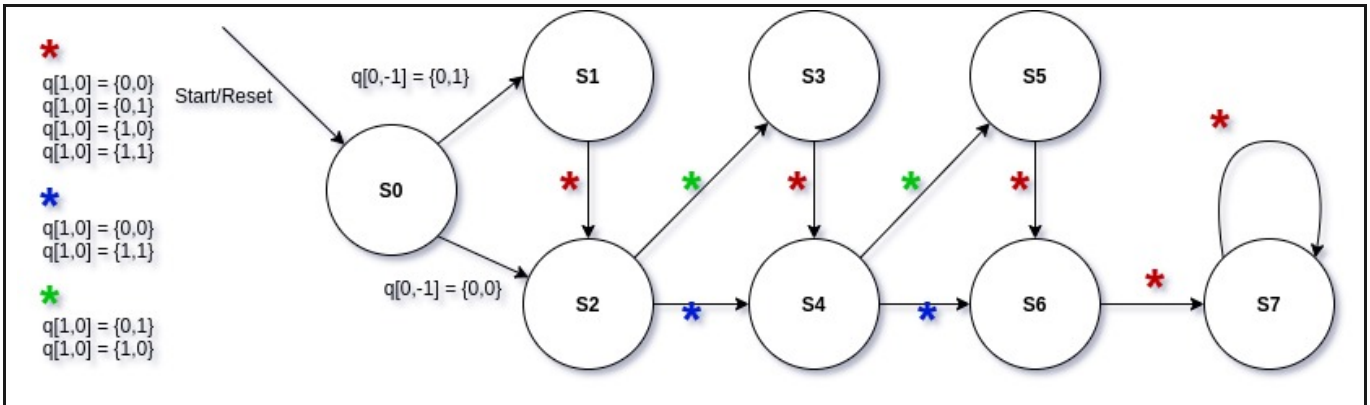


Figura 4: *Estados de la unidad de control tras la tercera iteración*

Tras estos cambios se tarda ***352*** **ciclos** en completar las pruebas y la aceleración es ***1.454***.

2.5. Diseño final

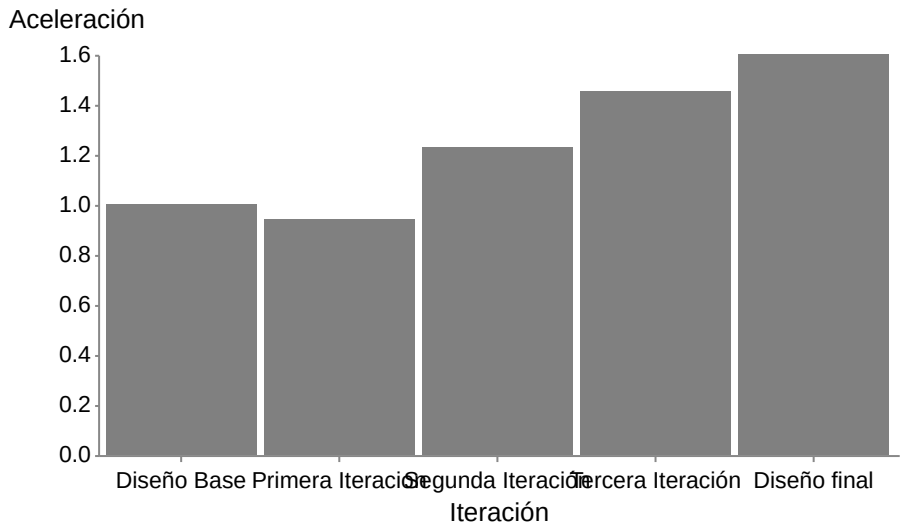
El diseño final es similar a la tercera iteración con la salvedad de realizar las operaciones aritméticas y desplazamiento en el mismo estado. Esto permite reducir el número de ciclos en 1 cada vez que se realice una suma o resta. Para ello se ha modificado la estructura de forma que el resultado de las operaciones

entre desplazado al registro A. Esta mejora a demás de reducir los ciclos a ***320***, con la consecuente mejora en la aceleración, hace que cualquier operación se ejecute exactamente ***5 ciclos***.

3. Aceleración

Los cálculos de la aceleración se han realizado con los ciclos de reloj al no tener una medida de tiempo disponible.

$$Aceleración = (CiclosBase/CiclosMejora)$$



4. Señales GTKWave

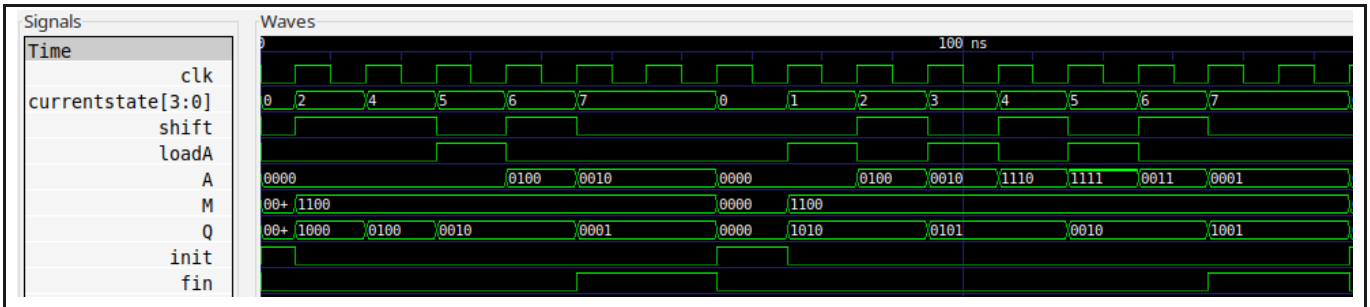


Figura 5: *Señales del diseño en su tercera iteración*

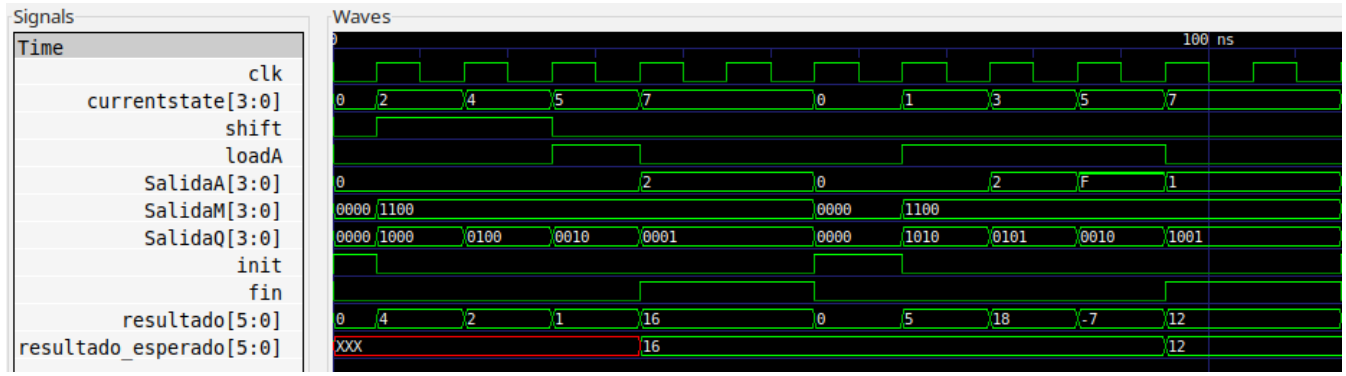


Figura 6: *Señales del diseño final*

5. Compilación:

Los ficheros contienen los includes necesarios (Aquellos ficheros de los que dependen, ie. componentes.v) para compilarlos sin tener que especificar todos los ficheros.

Compilar testbench:

```
iverilog <-o booth> multiplicador_tb.v
```