Práctica RPD. Reconfiguración parcial dinámica

Christopher Carmona Vargas

SBM - Sistemas SoPC Basados en Módulos,

Máster en Sistemas Electrónicos Avanzados

Índice

1. Introducción

A la hora de configurar una FPGA es posible que haya módulos o componentes que no se usen tan frecuentemente como algunos otros. Como los recursos de la FPGA son limitados se decidió buscar una nueva política de funcionamiento reconfigurable. Para enfrentar este problema los diseñadores hardware concluyeron que la mejor opción era hacer que la FPGA tuviera zonas reconfigurables, de manera que no hubiera que reconfigurar la FPGA por completo.

Son muchos los casos en los que ciertos módulos de la FPGA son cruciales para la comunicación con otros dispositivos y es crucial mantener una comunicación interrumpida con estos. Es por esto, que cuando se hace una reconfiguración asíncrona es posible que se generen *glich-es* en la comunicación de los dispositivos, comprometiendo así su correcto funcionamiento. Para enfrentar esta problemática los diseñadores hardware desarrollaron un método de reconfiguración síncrona. Esto es posible desde el lanzamiento de la series 7 de Xilinx ya que se añadió la tecnología *glich-less* cuyo objetivo era sincronizar la reconfiguración con el reloj del sistema, lo cual, garantiza que las señales se actualicen de forma controlada y no se comprometa las zonas que dependen de la integridad de los datos.

2. Desarrollo

La metodología que siguen los diseñadores para hacer este tipo de diseños reconfigurables cumple ciertos criterios. Para comenzar debe comprobarse la integridad de los diseños individuales de forma que se sepa que todos los diseños son sintetizables e implementables. Posteriormente, debe seguirse un proceso de compilación paso a paso el cual se indica en este documento.

2.1. Entity del WBS

Como se ha indicado en el párrafo anterior, debe comprobarse que los diseños sean sintetizables e implementables. No obstante, existe otro requisito añadido el cual permite la reconfigurabilidad. Como es de esperar, la parte estática del diseño no puede adaptarse a entidades diferentes, lo que implica que el diseño de los esclavo debe tener las mismas entradas y salidas. De esta manera, se hace transparente para la parte estática el cambio de esclavo.

```
entity wbs is
    Port (

            RST_I:  in  std_logic;          -- WB : Global RESET signal
            ADR_I:  in std_logic_vector(15 downto 0 );          -- WB : Register selection
            CLK_I:  in  std_logic;          -- WB : Global bus clock
            STB_I:  in  std_logic;          -- WB : Access qualify from the master
            CYC_I_SHARED:  in  std_logic;   -- WB : Access qualify from the arbitrer
            WE_I:   in  std_logic;          -- WB : Read/write request
            DAT_I:  in std_logic_vector(15 downto 0 ); -- WB : 16 bits data bus
                                                       -- input
            DAT_O:  out std_logic_vector(15 downto 0 ); -- WB : 16 bits data bus
                                                       -- ouput
            ACK_O:  out std_logic;          -- WB : Ack from to the master
            ENABLE0:  out  std_logic;
            ENABLE1:  out  std_logic;
            DISPLAY0_1:  out  std_logic_vector(7 downto 0)
            );
end wbs;
```

Figura 1: *Entity del WBS*

Como se observa en la Figura 1 la entidad del bloque PWM implementado coincide con el de los otros dos diseños entregados por el instructor.

2.2. Síntesis del diseño

```
# Primer paso para la reconfiguraci n parcial din mica:


# run_pr.tcl lanza la s ntetsis y la implementaci n de las
    configuraciones que se le indican
# En nuestro caso solo se ha realizado la s ntetsis de las
    configuraciones wbs_add, wbs_mult y wbs_pwm
# Estas s ntesis son las que se van a usar m s adelante
source run_pr.tcl -notrace
```

Código 1: *Síntesis que se van a realizar en el proyecto.*

En el bloque de instrucciones 1 se ha lanzado el primer *script* donde se especifican que partes de la reconfiguración se van a automatizar y cuales van ser los fuentes de las partes reconfigurables y la parte estática.

2.3. Implentación del diseño

```
# Selecciona la parte que se va a usar en esta pr ctica, y se le indica
    a Vivado
set part xc7z020-clg484-1


# Selecciona la placa que se va a usar en esta pr ctica, y se le indica
    a Vivado
set board zedboard


# Se crea un dise o en memoria
create_project -in_memory -part $part


#Se carga el dise o est tico
add_files .Synth/Static/top_synth.dcp


```

```
17 # Se cargan las constrains del dise o de alto nivel
18 add_files .Sources/xdc/top_io_$board.xdc
19
20 set_property USED_IN {implementation} [get_files ./Sources/xdc/
       top_io_$board.xdc]
21
22
23 # Carga los dos primeros checkpoints de la s ntesis de wbs_add
24 add_files ./Synth/wbs_add/wbs_synth.dcp
25
26 set_property SCOPED_TO_CELLS {inst_wbs} [get_files ./Synth/wbs_add/
       wbs_synth.dcp]
27
28
29 # Une el dise o entero
30 link_design -mode default -reconfig_partitions {inst_wbs} -part $part
       -top top
31
32
33 # Guarda el estado del dise o ensamblado para esta primera
       configuraci n
34 write_checkpoint -force ./Checkpoint/top_link_wbs_add.dcp
```

Código 2: *Selección del hardware y sintetización de la configura por defecto. En este caso la configuración con el esclavo de suma.*

En el bloque 2 se indica cual sera el *target* y la tarjeta de desarrollo. Posteriormente, se ha añadido el diseño predeterminado. Antes de finalizar el bloque, se ha hecho un punto de control tras enlazar la parte reconfigurable del esclavo de suma con la parte estática. Esta será la configuración predeterminada.

2.4. Creación y configuración del PBlock

```
1
2 # Se dibuja el PBlock y se configura
3
4
5 # Guarda el PBlock y sus propiedades
6 write_xdc ./Sources/xdc/top_all.xdc
7
8
9 # Obtiene el proceso que se encuentra en uno de los archivos Tcl
10 source ./Tcl_HD/hd_utils.tcl
11
12
13 # Escribe la informaci n de los constrains
14 export_pblocks -file ./Sources/xdc/pblocks.xdc
```

Código 3: *Bloque de codigo para indicar el la zona de FPGA donde se hubicara la parte reconfigurable del diseño.*

En este paso se ha indicado la ubicación del esclavo reconfigurable mediante la interfaz gráfica de Vivado. Se ha especificado una zona del chip X0Y1 de la FPGA como se indica en la Figura 2. En este caso se ha tenido que habilitar las opciones de rutado automático y el *snapping mode*. Estas opciones permiten el ajuste del PBlock seleccionado para no cortar módulos internos de la selección y el ajuste de la altura del PBlock para mantener

las restricciones de propagación de acuerdo con el reloj. La instrucción de la última linea exporta el bloque dibujado al documento de restricciones.

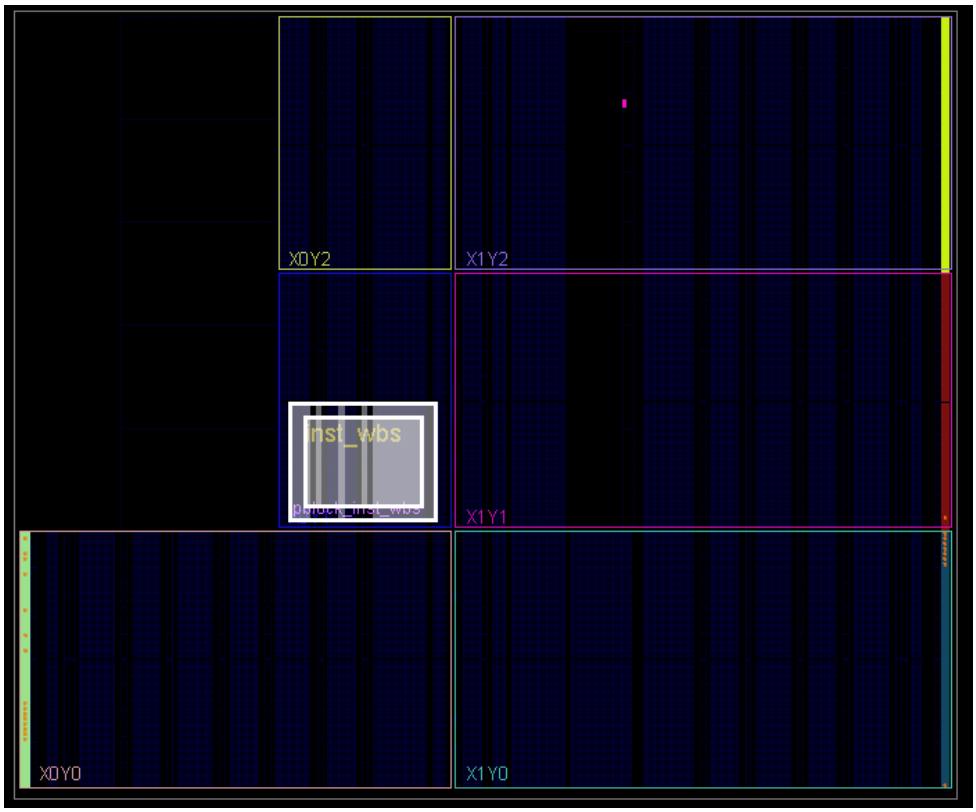


Figura 2: *Interfaz de Vivado mostrando el PBlock seleccionado.*

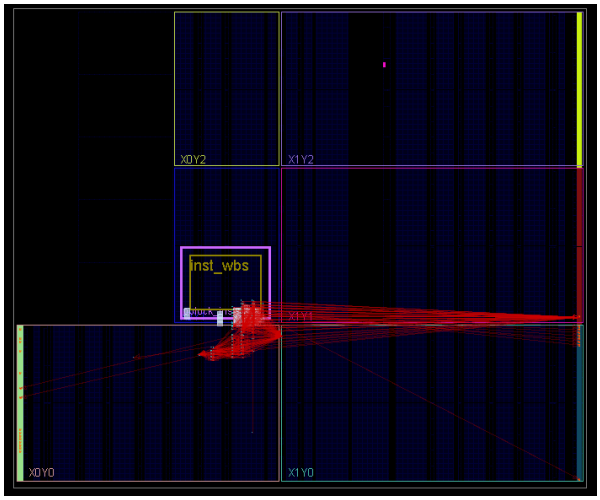
2.5. Implentación de la primera configuración

```
1
2  # Optimiza el dise o
3  opt_design
4
5
6  # Coloca el dise o
7  place_design
8
9  # Se plantean las conexiones del dise o
10 route_design
11
12
13 # Guarda el checkpoint del dise o completo
14 write_checkpoint -force Implement/Config_wbs_add_implement/
       top_route_design.dcp
15
16
17 # Crea archivos de informe sobre el uso
18 report_utilization -file Implement/Config_wbs_add_implement/
       top_utilization.rpt
19
20
21 # Crea archvios de informe sobre el resumen de los tiempos
```

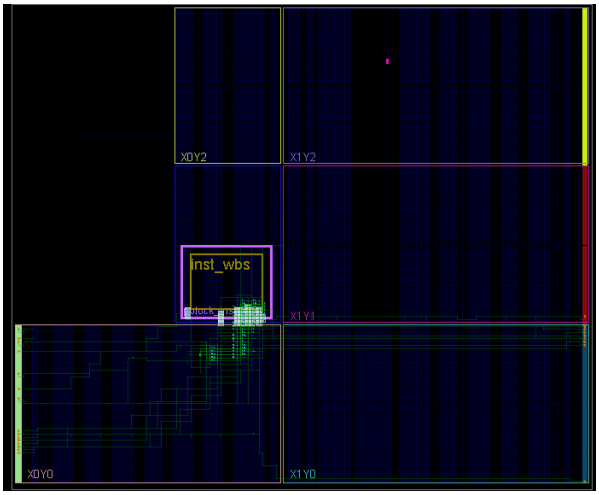
```
report_timing_summary -file Implement/Config_wbs_add_implement/
    top_timing_summary.rpt


# Borra la l gica del m dulo reconfigurable
update_design -cell inst_wbs -black_box


# Asegura el emplazamiento y el rutado
lock_design -level routing


Escribe el checkpoint restante del dise o est tico
write_checkpoint -force Checkpoint/static_route_design.dcp


# Cierra el dise o
close_project
```

Código 4: *Código para implementación de la primera configuración.*

En este paso se ha implementado el diseño de la configuración predeterminada y se ha guardado un punto de control una vez realizado el emplazamiento y el rutado. El resultado se puede observar en las Figuras 3a y 3b. Tras esto, se ha limpiado el contenido de la PBlock y se ha guardado un punto de control con un PBlock sin esclavo asignado para las siguientes reconfiguraciones.



a *Emplazamiento del diseño del esclavo de suma.*



b *Rutado del diseño del esclavo de suma.*

2.6. Implentación de la segunda configuración

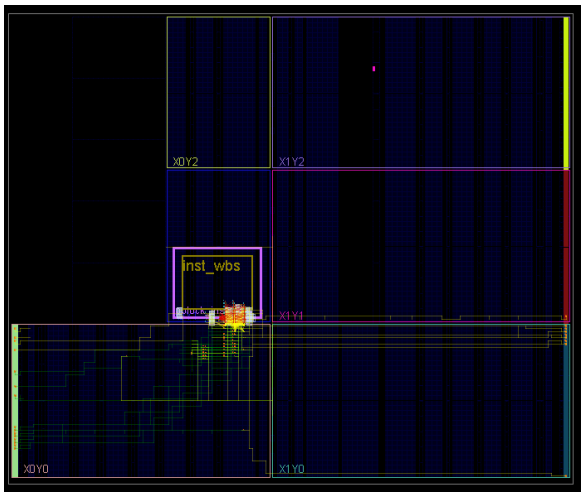
```
# Crea un nuevo dise o
create_project -in_memory -part $part


# Carga el dise o est tico
add_files ./Checkpoint/static_route_design.dcp


# Carga el checkpoints de la s ntesis de wbs_mult
add_files ./Synth/wbs_mult/wbs_synth.dcp
```

```
11
12 set_property SCOPED_TO_CELLS {inst_wbs} [get_files
       .Synthwbs_multwbs_synth.dcp]
13
14
15 # Une el dise o entero
16 link_design -mode default -reconfig_partitions {inst_wbs} -part $part
       -top top
17
18
19 # Optimiza el dise o
20 opt_design
21
22
23 # Coloca el dise o
24 place_design
25
26
27 # Se plantean las conexiones del dise o
28 route_design
29
30
31 # Se guarda el checkpoint del dise o completo
32 write_checkpoint -force
       ImplementConfig_wbs_mult_importtop_route_design.dcp
33
34
35 # Crea archivos de informe sobre el uso
36 report_utilization -file
       ImplementConfig_wbs_mult_importtop_utilization.rpt
37
38
39 # Crea archvios de informe sobre el resumen de los tiempos
40 report_timing_summary -file
       ImplementConfig_wbs_mult_importtop_timing_summary.rpt
41
42
43 # Verifica la configuraci n
44 pr_verify ImplementConfig_wbs_add_implementtop_route_design.dcp
       ImplementConfig_wbs_mult_importtop_route_design.dcp
45
46
47 # Cierra el proyecyo
48 close_project
```

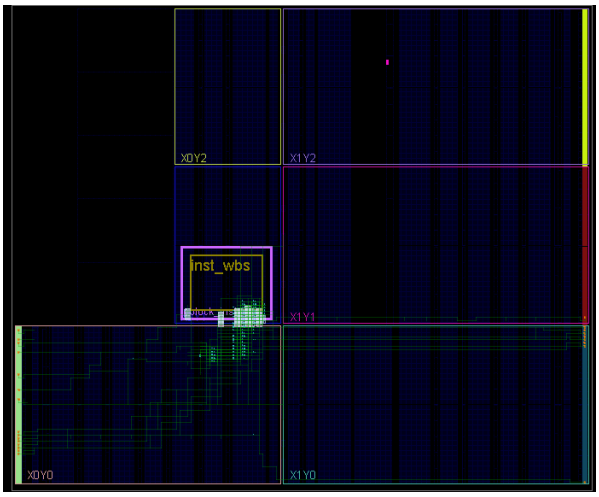
Código 5: *Código para implementación de la segunda configuración.*

En este bloque de código 5 se ha partido desde el punto de control del apartado anterior donde tras guardar el punto de control de la configuración predeterminada se ha guardado un punto de control con un Pblock "vaciado". Partiendo de este punto, se ha indicado la ubicación del siguiente esclavo a implementar y se han añadido los archivos relacionados con esta configuración. Acto seguido se ha enlazado con el top.

Ya se dispone de la configuración completa para iniciar la implementación así que se optimiza, emplaza y ruta el diseño. El resultado se puede observar en las Figuras 4a y 4b.



a *Emplazamiento del diseño del esclavo de multiplicación.*



b *Rutado del diseño del esclavo de multiplicación.*

Para comprobar la compatibilidad de los diseños se ha lanzado el comando `pr_verify` donde en la Figura 5, en la última linea se observa que la compatibilidad de los dos diseños es correcta.

```
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-479] Netlist was created with Vivado 2018.3
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Timing 38-478] Restoring timing data from binary archive.
INFO: [Timing 38-479] Binary timing data restore complete.
INFO: [Project 1-856] Restoring constraints from binary archive.
INFO: [Project 1-853] Binary constraint restore complete.
Reading XDEF placement.
Reading placer database...
Reading XDEF routing.
Read XDEF File: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.122 . Memory (MB): peak = 2411.590 ; gain = 0.000
Restored from archive | CPU: 0.000000 secs | Memory: 0.000000 MB |
Finished XDEF File Restore: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.123 . Memory (MB): peak = 2411.590 ; gain = 0.000
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 2411.590 ; gain = 0.000
INFO: [Project 1-604] Checkpoint was created with Vivado v2018.3 (64-bit) build 2405991
INFO: [Constraints 18-1020] PR Verify Summary:
DCP1: Implement/Config_wbs_add_implement/top_route_design.dcp
  Number of reconfigurable modules compared = 1
  Number of partition pins compared         = 0
  Number of static tiles compared           = 1428
  Number of static sites compared           = 110
  Number of static cells compared           = 558
  Number of static routed nodes compared    = 7019
  Number of static routed pips compared     = 6504

DCP2: Implement/Config_wbs_mult_import/top_route_design.dcp
  Number of reconfigurable modules compared = 1
  Number of partition pins compared         = 0
  Number of static tiles compared           = 1428
  Number of static sites compared           = 110
  Number of static cells compared           = 558
  Number of static routed nodes compared    = 7019
  Number of static routed pips compared     = 6504
INFO: [Vivado 12-3253] PR_VERIFY: check points Implement/Config_wbs_add_implement/top_route_design.dcp and Implement/Config_wbs_mult_import/top_route_design.dcp are compatible
```

Figura 5: *Verificación tras añadir la segunda configuración*

2.7. Implentación de la tercera configuración

```
# Crea un nuevo dise o
create_project -in_memory -part $part


# Carga el dise o est tico
add_files ./Checkpoint/static_route_design.dcp


# Carga el checkpoints de la s ntesis de wbs_pwm
add_files ./Synth/wbs_pwm/twbs_synth.dcp
```

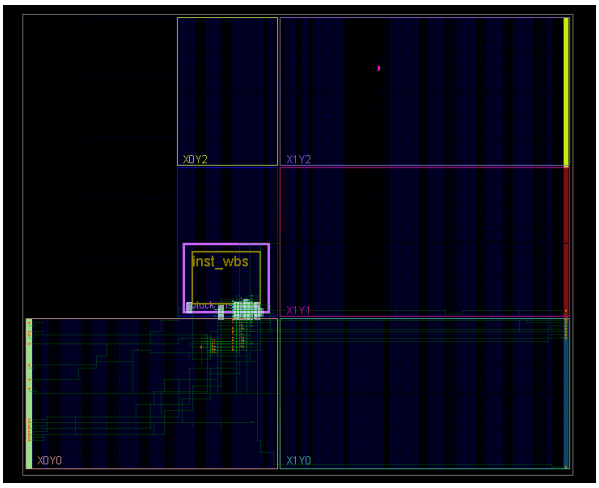
```
11
12 set_property SCOPED_TO_CELLS {inst_wbs} [get_files
       .Synthwbs_pwmwbs_synth.dcp]
13
14
15 # Une el dise o entero
16 link_design -mode default -reconfig_partitions {inst_wbs} -part $part
       -top top
17
18
19 # Optimiza el dise o
20 opt_design
21
22
23 # Coloca el dise o
24 place_design
25
26
27 # Se plantean las conexiones del dise o
28 route_design
29
30
31 # Se guarda el checkpoint del dise o completo
32 write_checkpoint -force Implement/Config_wbs_pwm_import/
       top_route_design.dcp
33
34
35 # Crea archivos de informe sobre el uso
36 report_utilization -file Implement/Config_wbs_pwm_import/
       top_utilization.rpt
37
38
39 # Crea archvios de informe sobre el resumen de los tiempos
40 report_timing_summary -file Implement/Config_wbs_pwm_import/
       top_timing_summary.rpt
41
42
43 # Verifica la configuraci n
44 pr_verify Implement/Config_wbs_add_implement/top_route_design.dcp
       Implement/Config_wbs_pwm_import/top_route_design.dcp
45
46
47 # Cierra el proyecto
48 close_project
```

Código 6: *Declaraciones de funciones con varios tipos de argumntos de entrada y salida.*

Este proceso es idéntico al del paso anterior pero esta vez implementado el modulo PWM. Aun que no se indique en el bloque de instrucciones se ha comprobado la compatibilidad del diseño PWM con los otros dos diseños de forma satisfactoria.

El resultado de la optimización, emplazamiento y rutado se puede observar en las Figuras 6a y 6b.

a *Emplazamiento del diseño del esclavo de PWM.*



b *Rutado del diseño del esclavo de PWM.*

2.8. Generar los Bitstreams

```
# Cuarto paso generar los bitstream general y parciales.


#################### Primera configuraci n ####################


# Se lee la primera configuraci n en la memor a
open_checkpoint Implement/Config_wbs_add_implement/top_route_design.dcp


# Se genera el bitstream parcial y general para este dise o
write_bitstream -force -file Bitstreams/Config_wbs_add.bit


# Se cierra el proyecto
close_project


#################### Segunda configuraci n ####################


# Se lee la segunda configuraci n en la memor a
open_checkpoint Implement/Config_wbs_mult_import/top_route_design.dcp


# Se genera el bitstream parcial y general para esta segunda
   configuraci n (wbs_mult)
write_bitstream -force -file Bitstreams/Config_wbs_mult.bit


# Se cierra el proyecto
close_project


#################### Tercera configuraci n ####################


# Se lee la tercera configuraci n en la memor a
```

```
38 open_checkpoint Implement/Config_wbs_pwm_import/top_route_design.dcp
39
40
41 # Se genera el bitstream parcial y general para esta tercera
       configuraci n (wbs_pwm)
42 write_bitstream -force -file Bitstreams/Config_wbs_pwm.bit
43
44
45 # Se cierra el proyecto
46 close_project
47
48
49 ##################### Grey Box ####################
50
51
52 # Se lee el rutado del dise o est tico
53 open_checkpoint Checkpoint/static_route_design.dcp
54
55
56
57 update_design -cell inst_wbs -buffer_ports
58
59
60 # Coloca el dise o
61 place_design
62
63
64 # Plantea las conexiones del dise o
65 route_design
66
67
68 # Esto asegura que el dise o este completamente implementado
69
70
71 # Escribe el checkpoint del dise o completo
72 write_checkpoint -force Checkpoint/Config_greybox.dcp
73
74
75 # Genera el bitstream parcial y general con "grey boxes" ademas de
       borrar el bitstream del m dulo reconfigurable
76 write_bitstream -force -file Bitstreams/config_greybox.bit
77
78 close_project
```

Código 7: *Declaraciones de funciones con varios tipos de argumentos de entrada y salida.*

En este último paso se han generado todos los *bitstream* estático y parciales para poder implementar toda la configuración.

Analizando el código del blobe 7, podemos observar que los *bitstreams* generados son los siguientes:

- Implementación de esclavo de suma.
- Implementación de esclavo de multiplicación.
- Implementación de esclavo de PWM.
- Implementación de esclavo de *Grey box*

Los tres primeros son los que se refiere a la configuración de cada uno de los esclavos mientras que el último, la opción *grey box*, se refiere al *bitstream* con la configuración "virgen" o sin esclavo. Estos *bitstreams* se pueden usar para minimizar el consumo de la FPGA si ninguno de estos módulos fuera necesario.

2.9. Uso de recursos

Respecto al uso de recursos de la FPGA se ha analizado el contenido de los *report* generados a los largo del proceso implementación y escritura de *bitstreams* anteriores. En el bloque de código 8 se observa el uso de recurso de cada uno de los esclavos. los cuales cumplen correctamente con la cantidad de recursos otorgados en apartados anteriores a la hora de dibujar el PBlock.

Es de destacar que en la primera configuración, es decir, la de la suma, el numero de recursos fijado se mantiene a cero mientras que en los otros dos la columna coincide complementamente. Esto indica, de manera implícita, el uso de recursos de la parte estática del diseño y de aqui podemos conocer el uso de recursos de todas las configuraciones.

Además de la información obtenida en la tabla, se observa que tanto la configuración PWM como en la configuración multiplicación se hace uso de un módulo DSP mientras que con el de suma simplemente se hace la suma mediante lógica LUT.

```
wbs_add
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs                 |  332 |     0 |     53200 |  0.62 |
|   LUT as Logic             |  300 |     0 |     53200 |  0.56 |
|   LUT as Memory            |   32 |     0 |     17400 |  0.18 |
|     LUT as Distributed RAM |   24 |     0 |           |       |
|     LUT as Shift Register  |    8 |     0 |           |       |
| Slice Registers            |  448 |     0 |    106400 |  0.42 |
|   Register as Flip Flop    |  448 |     0 |    106400 |  0.42 |
|   Register as Latch        |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                   |   32 |     0 |     26600 |  0.12 |
| F8 Muxes                   |    0 |     0 |     13300 |  0.00 |
+----------------------------+------+-------+-----------+-------+

wbs_mul
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs                 |  314 |   204 |     53200 |  0.59 |
|   LUT as Logic             |  282 |   172 |     53200 |  0.53 |
|   LUT as Memory            |   32 |    32 |     17400 |  0.18 |
|     LUT as Distributed RAM |   24 |    24 |           |       |
|     LUT as Shift Register  |    8 |     8 |           |       |
| Slice Registers            |  432 |   206 |    106400 |  0.41 |
|   Register as Flip Flop    |  432 |   206 |    106400 |  0.41 |
|   Register as Latch        |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                   |   32 |     0 |     26600 |  0.12 |
| F8 Muxes                   |    0 |     0 |     13300 |  0.00 |
+----------------------------+------+-------+-----------+-------+

Wbs_pwm
+----------------------------+------+-------+-----------+-------+
```

```
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Slice LUTs                  |  330 |   204 |     53200 |  0.62 |
|   LUT as Logic              |  298 |   172 |     53200 |  0.56 |
|   LUT as Memory             |   32 |    32 |     17400 |  0.18 |
|     LUT as Distributed RAM  |   24 |    24 |           |       |
|     LUT as Shift Register   |    8 |     8 |           |       |
| Slice Registers             |  362 |   206 |    106400 |  0.34 |
|   Register as Flip Flop     |  362 |   206 |    106400 |  0.34 |
|   Register as Latch         |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                    |    0 |     0 |     26600 |  0.00 |
| F8 Muxes                    |    0 |     0 |     13300 |  0.00 |
+-----------------------------+------+-------+-----------+-------+
```

Código 8: *Uso de recursos de la FPGA*

3. Resultados obtenidos

Tras cargar los bitstreams en la placa se ha podido comprobar el correcto funcionamiento de todos los bistreams menos del PWM. Para solucionar este problema se ha adaptadp el diseño Top de manera que dejará abierta desde el modulo *máster* la salida de los pines Led. Una vez corregido esto se ha podido comprobar el correcto funcionamiento del modulo PWM y se ha podido conmutar entre los diferentes esclavos si necesidad de reiniciar la FPGA.

4. Conclusiones

La reconfiguración parcial dinámica es una herramienta que permite ahorra tiempo de diseño, recursos de la FPGA y recursos energéticos. Al permitir hacer una reconfiguración parcial, se obtiene la ventaja de reconfigurar solo aquellos módulos que sean necesarios cambiar. Así, en el caso más común en el que la FPGA es el nexo de comunicación entre dispositivos externos y μP o μC se confirma la integridad de dicha comunican. De la misma manera permite implementar configuraciones de redundancia que mantengan esa comunicación y, en caso de que alguno de aquellos módulos fallará se podría reconfigurar el modulo caído mientras que el otro redundante mantiene viva la conexión. De estos ejemplos tambien se concluye la importancia de la tecnología glich-less para este tipo de reprogramaciones pues es necesario no comprometer las señales anteriormente mencionadas.

Por otro lado es necesario hacer una uso correcto de los PBlock que se asigna a cada una de las zonas reconfigurables siempre teniendo en cuenta los recursos que se usarán para todos los diseños. En este caso se ha observado que el PBlock asignado cumple más que de sobra las necesidades de los esclavos y que por ende se podría reducir el tamaño del bloque reconfigurable sin necesidad de comprometer los diseños.