

Reporte Interno tiempos de acceso a Memoria en plataforma Zynq.

Electratraining. Agosto-sept 2023. V0.1 (basado en pruebas RCXX de marzo 2020 en vivado 2018.3)

Contents

Introducción.....	3
Circuito 1. Path-Through Video DMA.....	4
1.1. Crear Proyecto HLS	4
1.2. Crear proyecto Vivado	4
1.3. Crear proyecto en Vitis.....	5
1.4. Notas y conclusiones:.....	6
Circuito 2. Escritura a Memoria AXI-Master (cpyData)	8
2.1. Crear Proyecto HLS	8
2.2. Crear proyecto Vivado	8
2.3. Crear proyecto en Vitis.....	9
2.4. Notas y conclusiones:.....	11
Circuito 3. Escritura a Memoria AXI-Master usando Burst (movData)	12
3.1 Notas y conclusiones:.....	12
Circuito 4. Escritura directa (Wr_data_dir)	14
4.1. Proyecto HLS	14
4.2. Proyecto Vivado	14
4.3. Proyecto Vitis	15
4.4 Notas y conclusiones:.....	17
Circuito 5. Escritura directa con FIFOs (Wr_data_dir).....	18
5.1. Proyecto HLS (mismo que 4.1)	18
5.2. Proyecto Vivado	18
5.3. Proyecto Vitis	18
5.4 Notas y conclusiones:.....	19
Circuito 6. Múltiples lecturas y escrituras. Acceso a 4 HPs y 2 GPs (proj_4HP_2GP)	21
6.1. Proyectos HLS.....	21
6.2. Proyecto Vivado	21
6.3. Proyectos Vitis.....	23
6.4 Notas y conclusiones:.....	24

6.4.1 Comparación con el sistema ITCL.....	25
--	----

Introducción

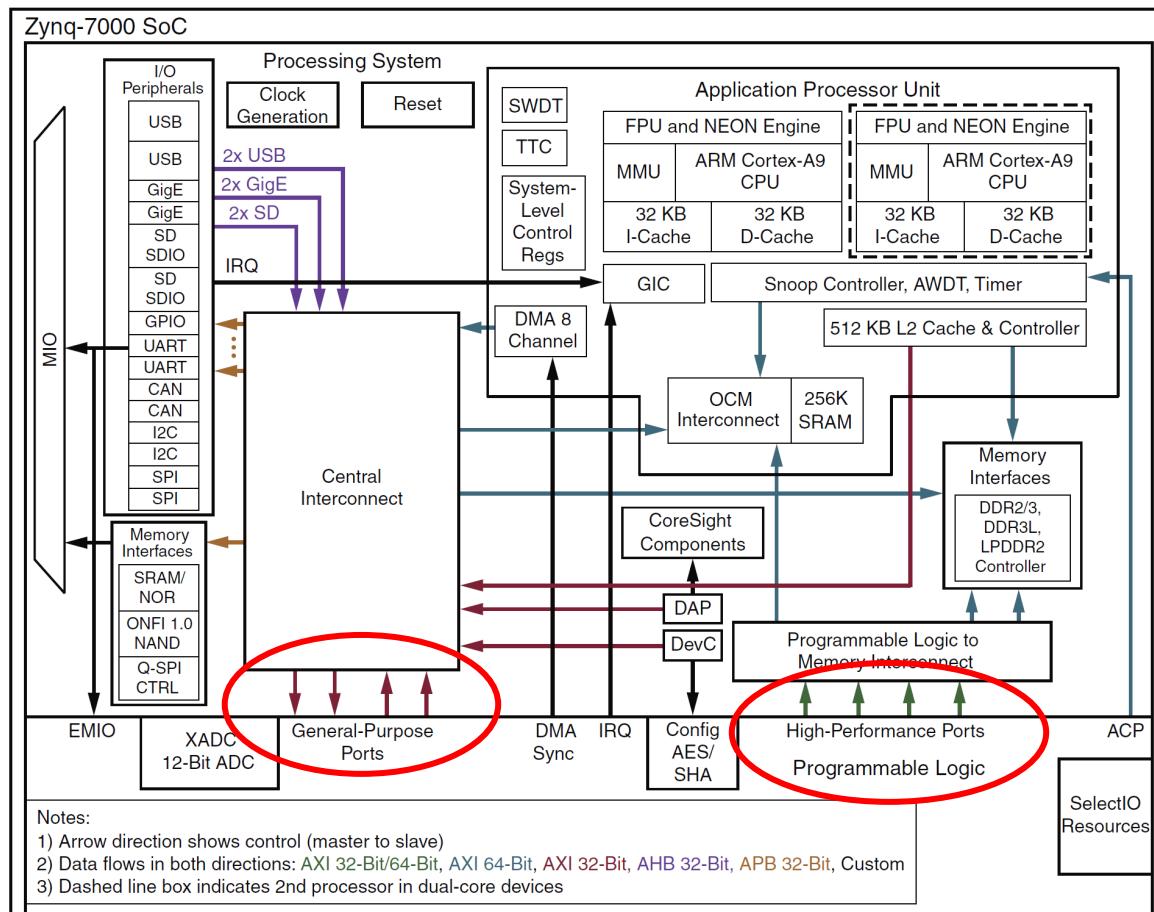
Objetivo: Analizar a través de sistemas simples el comportamiento en escritura y lectura a memoria de la plataforma Zynq 7000 utilizando el controlador de memoria provisto en por el PS (Processing System) a una única memoria DDR externa. Si bien Xilinx provee notas de aplicación con información de anchos de banda y latencia es necesario poseer la comprensión en detalle de la interacción de accesos concurrentes.

Setup: Se utiliza una tarjeta Zybo Z-20. Dispositivo XC7Z020-1CLG400C y memoria DDR externa 1 GB DDR3L con bus 32-bit @ 533 MHz (1066 Mtransf) (dos Micron MT41K256M16HA-125 DDR3L). Los ejemplos son portables a cualquier sistema basado en Zynq 7000 conectado a una DDR externa.

Los ejemplos son desarrollados en Vivado 2023.1 y Vivado HLS 2023.1 (basado en pruebas previas en 2018.3. en marzo 2020)

Generalidades: Por simplicidad y claridad, los diseños Vivado estarán orientados al uso del IP-Integrator y el diseño de IP-cores en Vitis-HLS.

Pre-requisitos para entender este reporte: Conocimiento de Vivado, nociones de Vitis-HLS y arquitectura Zynq 7K. Es relevante recordar los canales de comunicación desde la PL de la Zynq hacia la DDR:



Circuito 1. Path-Through Video DMA.

El circuito captura imagen (matriz bidimensional desde memoria) de 64 bits de ancho y escribe matriz del mismo tamaño de 32 bits. El hecho de usar matrices de 64 bits de ancho es previendo un “worst case”, equivalente a dos matrices de 32 de ancho (una matriz de mascara y otra de translación).

Para el movimiento de datos se configuran AXI Video DMAs en ambas direcciones. Se lee desde HP0 y escribe en HP1. Un core *Path Through* suma parte alta y baja de 32 bits. Saca un stream de datos (pixels) que utiliza un otro AXI Video DMA que lo escribe en memoria.

Este esquema es cercano a la cota máxima de rendimiento esperable para una lectura escritura desde DDR y a DDR, asumiendo que el movimiento que genera el componente AXI Video DMA es altamente optimizado.

1.1. Crear Proyecto HLS

Desde el directorio “movDataZynq\prj_hls” ejecutar el script ..\scripts\path_trg_hls_script.tcl

```
movDataZynq\prj_hls>vitis_hls -f ..\scripts\path_trg_hls_script.tcl
```

Genera la carpeta con la exportación del core IP “\prjs_hls\hls_path_Thr\solution1\impl\ip”

El core HLS únicamente cuenta la cantidad de pixeles que pasan por él y suma la parte baja y alta de los 64 bits generando una salida de 32 bits.

1.1.a (opcional) Para abrir el proyecto HLS:

```
Vitis_hls -p hls_path_Thr
```

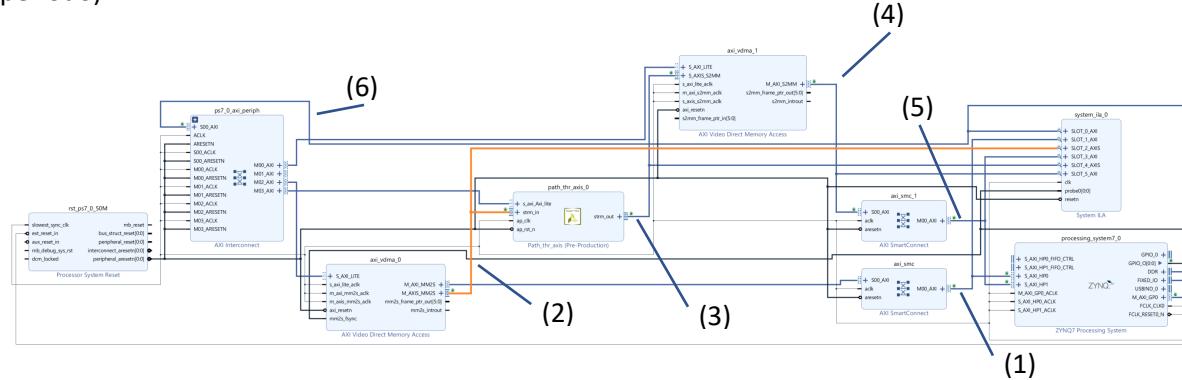
O bien en el GUI, open Project. Este proyecto no tiene testbech.

1.2. Crear proyecto Vivado

Crear proyecto para la tarjeta objetivo (Zybo en este caso).

Agregar el repositorio IP del core HLS (Project settings -> IP -> Repository)

Crear el Block Design, agregar el wrapper HDL. El Zynq está configurado con un reloj de 100 MHz (10 ns periodo).



Existen dos scripts uno que crea el proyecto y llama al segundo (path_thr_db.tcl) que crea el diagrama de bloques. Desde la consola tcl y desde el directorio /movDataZynq/prj_vivado de Vivado se puede ejecutar:

```
source ../scripts/path_thr_vivado.tcl
```

Que creará el proyecto.

Comentarios del proyecto:

- Axi_video_dma0. Configurado para leer memoria desde HP0 y escribe un stream de 64 bits que entra al core path_thr.
- Axi_video_dma1. Configurado para leer un stream de 32 bits y escribir la DDR via el puerto HP1.
- El inicio de los videos DMA se lleva a cabo por GPIOs del PS
- El sistema ILA monitoriza el camino de datos. (1) Lectura de DDR; (2) salida stream 64 bits, (3) stream de 32 bits; (4) escritura a DDR en 32 bits; (5) conversión y escritura a 64 bits; (6) puerto GP usado para configurar y leer el core IP.

1.3. Crear proyecto en Vitis

Implementar y generar bitstream. Exportar HW incluyendo bitstream.

Lanzar SDK (Vitis) y crear un proyecto C++ en baremetal (file -> new Application Project)

Agregar el fichero arm_c_code/pathThrough.cc (importante es un proyecto c++ en baremetal)

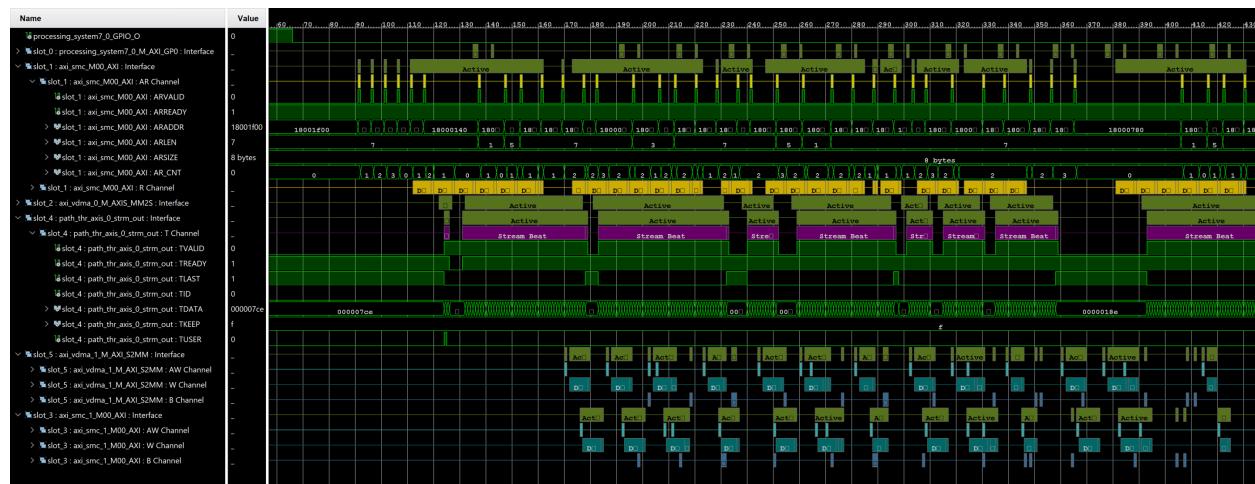


Fig: Una primera ejecución de una matriz de 20 x 50 @ 100Mhz

PL ejecutando a 100 MHz

Total Writes: 1000 (20 x 50). RunTime: 11.7 us, 85822.18 FPS.

Total Writes: 1008000 (720 x 1400). RunTime: 10.401 ms, 96.15 FPS.

Total Writes: 2073600 (1080 x 1920). RunTime: 21.388 ms, 46.76 FPS.

Una palabra cada 10.46 ns. Aprox. un 4,6% de ciclos sin escribir

Aumentando la PL a 200 MHz

Total Writes: 1000 (20 x 50). RunTime: 7.5 us, 133483.53 FPS.

Total Writes: 1008000 (720 x 1400). RunTime: 6.098 ms, 163 FPS.

Total Writes: 2073600 (1080 x 1920). RunTime: 12.552 ms, 79.67 FPS.

Una palabra cada 6.55 ns. Aprox. un 31% ciclos sin escribir

```

*****
* DATA MOVEMENT Path Through      *
Num. Writes: 2073600

Freq PS: 666666687, Freq PL: 100000000. Ratio: 6.67. TimerFreq: 333333343
** Calibrating the PS timer:
init_time: 3219625 cycles.
curr_time: 3219650 cycles.
calibrationPS: 25 PS clock cycles (0.075 us).
Programming VDMA_MATRIX_0 registers...
Programming VDMA_1 registers...

RunTime: 7128115 PS clock cycles (21405.8 us - 21.384 ms). FPS: 46.76 .
Size X: 1920, Size X: 1080. TotalSize (X*Y): 2073600
Total Writes: 2073600

*****
* DATA MOVEMENT Path Through      *
Num. Writes: 2073600

Freq PS: 666666687, Freq PL: 200000000. Ratio: 3.33. TimerFreq: 333333343
** Calibrating the PS timer:
init_time: 3217148 cycles.
curr_time: 3217173 cycles.
calibrationPS: 25 PS clock cycles (0.075 us).
Programming VDMA_MATRIX_0 registers...
Programming VDMA_1 registers...

RunTime: 4183936 PS clock cycles (12564.4 us - 12.552 ms). FPS: 79.67.
Size X: 1920, Size X: 1080. TotalSize (X*Y): 2073600
Total Writes: 2073600

```

1.4. Notas y conclusiones:

La forma de medir el tiempo de ejecución es muy grosera, se trata de leer la cantidad de pixels que pasan por el core HLS. Si bien la precisión es del orden de 20-30 ciclos de timer (más de 50 ciclos de la PS, en el orden de 0.1 us) permite llevar a cabo conclusiones.

Con un reloj de 100 MHz en la PL, mover 1400x720 (1M de puntos) de 64 bits y volver a escribir esa cantidad de puntos, pero de 32 bits está en el orden de 11 ms. En condiciones ideales cerca de 100 FPS.

Escribir FullHD 1920x1080 (2M) de puntos de 64 bits y volver a escribir esa cantidad de puntos, pero de 32 bits está en el orden de 21 ms. En el orden de 45 FPS.

Se puede leer y escribir a la vez 100 Millones de muestras por segundo sin influencia entre ellos. Son unos $(8+4)*100$ MB/sec. 1.2 GBytes/sec (800 MB read + 400MB write)

DDR 3 MAX teórico con interfaz 32 bits		
MHZ	DDR	MB/s
533	1066	4264

Al subir a 200 MHz (poco realista para Zynq 7K). Se transfiere efectivamente cada 6.55 ns. Esto sería llegar hasta cerca de 150 Millones de muestras por segundo. Cerca de 1.8 GB/sec.

Period (ns)	Tiempo (ns)	numTransf	Media/transf	Overhead
10	10568771	1008000	10.48	1.04
10	21694906	2073600	10.46	1.04
5	6609282	1008000	6.55	1.31
5	13573735	2073600	6.54	1.31

Media/transf = Tiempo medio de cada transferencia (en ns).

Overhead = Cuantos ciclos extras a los teóricos necesita para enviar los datos.

Name	Constraints	Status	Progress	WNS	TNS	WHS	THS
✓ synth_1 (active)	constrs_1	synth_design Complete!	100%				
✓ impl_1	constrs_1	write_bitstream Complete!	100%	-2.172	-2029.7	0.050	0.000
Out-of-Context Module Runs							
> ✓ design_1		Submodule Runs Complete	100%				

Fig: A 200 MHZ hay violaciones de timing en la implementación (la mayoría en los ILA)

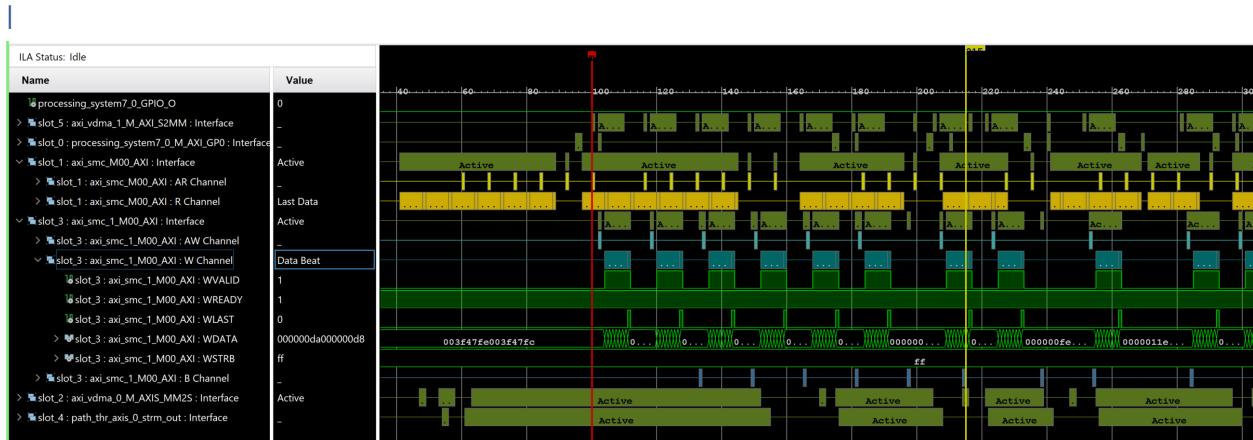


Fig: Captura a 200 MHZ. Las escrituras a 64 bits de ancho, aunque el stream de entrada al DMA es de 32.

Circuito 2. Escritura a Memoria AXI-Master (cpyData)

El circuito captura imagen (matriz bidimensional desde memoria) de 64 bits de ancho y escribe matriz del mismo tamaño de 32 bits usando una interfaz AXI-Master. Escribe datos uno a uno.

Para el movimiento de datos se configura un AXI VIDEO DMA para leer datos de memoria usando el puerto HP0 del Zynq (64 bits). El IP-core en HLS lee el stream y lo escribe como maestro AXI usando el puerto HP1, hace escritura elemento a elemento. El IP core almacena la suma de parte alta y baja de 32 bits.

2.1. Crear Proyecto HLS

Desde el directorio “movDataZynq\prjs_hls” ejecutar el script ..\scripts\cpyData_script.tcl

```
movDataZynq\prj_hls> vivado_hls -f ..\scripts\cpyData_hls_script.tcl
```

Genera la carpeta con la exportación del core IP “\prj_hls\mov_data\solution1\impl\ip”

2.1.a (opcional) Para abrir el proyecto HLS:

```
vitis_hls -p cpyData
```

O bien en el GUI, open Project. Analizar los resultados de implementación. Importante mantener un initiation interval de 1 (ii=1) en el diseño. Existen contadores de líneas y filas a modo de demo.

2.1.b (opcional) Simular y co-simular

El testbench envia un stream de 64 bits y revisa lo escrito por el IP-Core en la memoria.

Se puede simular con el testbench provisto. La co-simulación requiere cambiar la interfaz de ap_ctrl_none a axi_lite o dejar por defecto ap_ctrl_hs

2.2. Crear proyecto Vivado

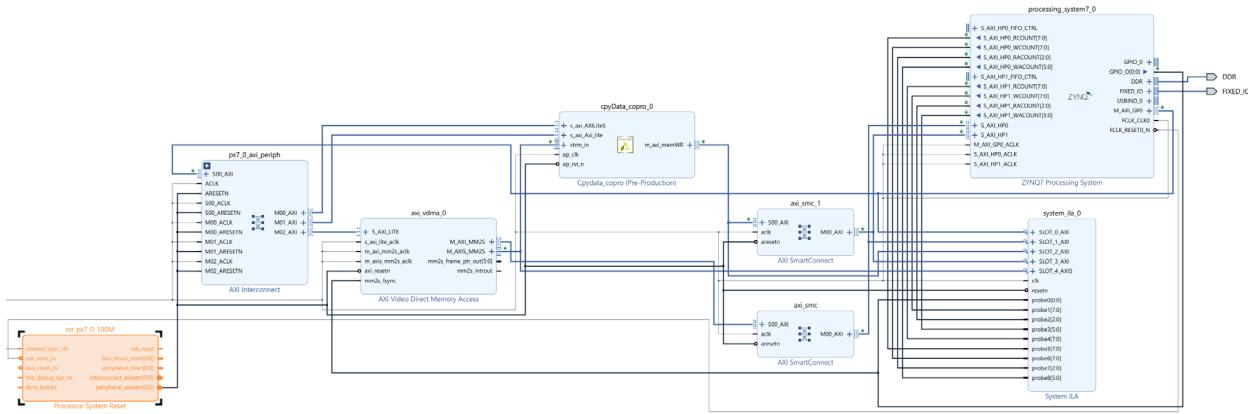
Crear proyecto para la tarjeta objetivo (Zybo Z-20).

Agregar el repositorio de IP-Cores HLS (Project settings -> IP -> Repository)

Crear el Block Design, agregar el wrapper HDL. El Zynq está configurado con un reloj de 100 MHz (10 ns periodo).

Existen dos scripts uno que crea el proyecto y llama al segundo (cpyData_vivado_db.tcl) que crea el diagrama de bloque. Desde la consola tcl y desde el directorio /movDataZynq/prjs_vivado de Vivado se puede ejecutar:

```
source ../scripts/cpyData_vivado.tcl
```



Comentarios del proyecto:

- Axi_video_dma0. Configurado para leer memoria desde HP0 y escribe un stream de 64 bits que entra al core copyData_copro.
- El inicio del video DMA se lleva a cabo por un GPIOs del PS.
- El IP-Core Escribe como maestro DMA datos consecutivos de 32 bits lo escribe a la DDR. El AXI SmartConnect traduce a escrituras de 64 bits y se conecta al puerto HP1 del Zynq.
- El sistema ILA monitoriza el camino de datos. (1) Lectura de DDR; (2) salida stream 64 bits, (3) Escrituras de 32 bits del core; (4) conversión y escritura a 64 bits; (5) puerto GP usado para configurar y leer el core IP. (6) Se monitorizan las contadores de lectura y escritura de HP0 y HP1.

2.3. Crear proyecto en Vitis

Implementar y generar bitstream. Exportar HW incluyendo bitstream.

Lanzar Vitis (SDK) y crear un proyecto C++ en baremetal (file -> new Application Project)

Agregar el fichero arm_c_code/copyData_axi_master.cc

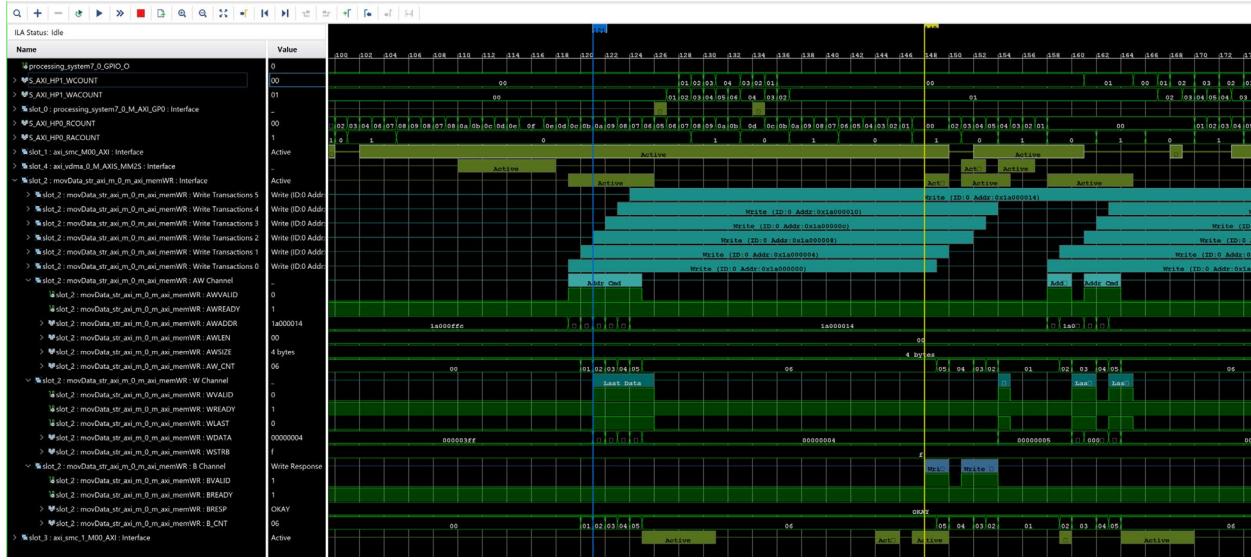
El programa genera una matriz de 64 bits donde contiene índices de posición en la parte baja y un número de secuencia en la parte alta. El hardware se supone que lo escribe en la posición indicada por los índices respecto de una dirección base.

32 bits	16 bits	16 bits
Nro secuencia	Índice col	Índice row

Existen 4 secuencias de inicialización. La matriz 1 es el direccionamiento secuencial. La matriz 2 dirección en diagonal, la matriz 3 lo hace por columnas, la matriz 4 lo en orden inverso. Como en estos ejemplos de matrices de 6 col x 4 row (24 elementos)

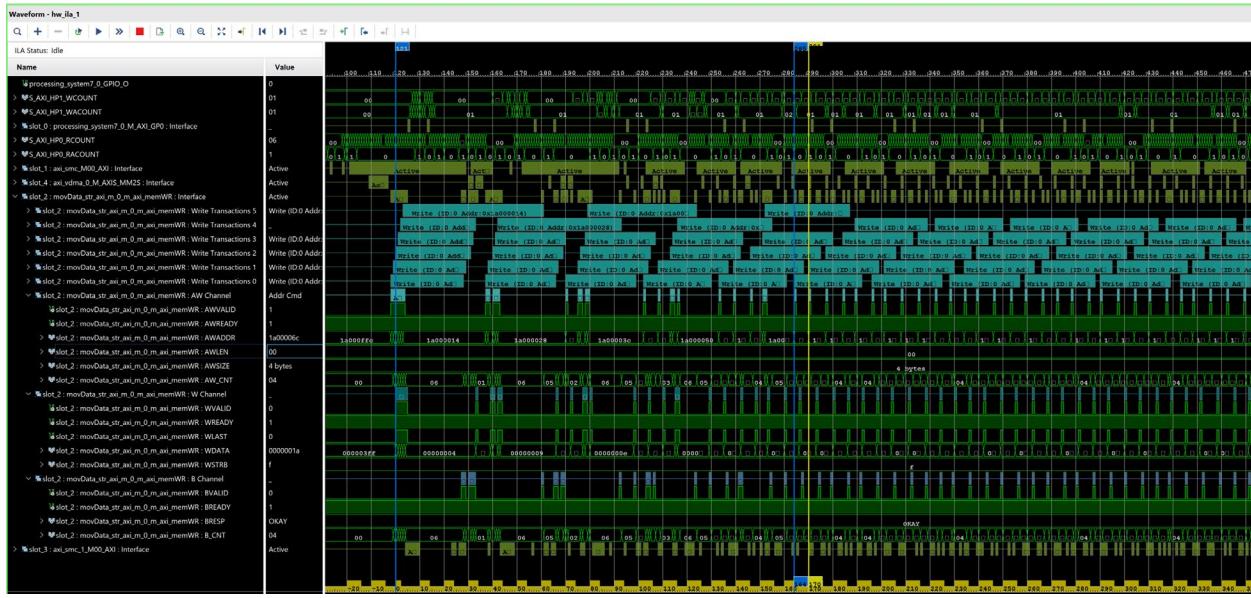
Matrix_1 (secuencial)						Matrix_2 (diagonal)						Matrix_3 (vertical)						Matrix_4 (vertical)						
0	1	2	4	4	5	0	4	8	12	16	20	0	4	8	12	16	20	23	22	21	20	19	18	
6	7	8	9	10	11	21	1	5	9	13	17	1	5	9	13	17	21	17	16	15	14	13	12	
12	13	14	15	16	17	18	22	2	6	10	14	18	22	2	6	10	14	18	11	10	9	8	7	6
18	19	20	21	22	23	15	19	23	3	7	11	3	7	11	15	19	23	5	4	3	2	1	0	

Una primera ejecución de una matriz de 32 x 32 @ 100Mhz con la matrix_1



Las escrituras se encolan hasta un máximo de 6 y esperan la respuesta del canal de respuesta (B channel) que son en el orden de 25 ciclos a 100 Mhz las primeras respuestas y luego en régimen permanente sobre los 4-5 ciclos de la PL a 100 Mhz (esto para la secuencia 1).

A 166 MHz, 7-8 ciclos (~50 ns)



PL a 100MHz

Total Writes: 1008000 (720 x 1400). RunTime: 38.408 ms, 26.04 FPS.

Total Writes: 2073600 (1080 x 1920). RunTime: 78.813 ms, 12.69 FPS.

Aumentando la PL a 166 MHz

Total Writes: 1008000 (720 x 1400). RunTime: 37.357 ms, 26.77 FPS.

Total Writes: 2073600 (1080 x 1920). RunTime: 76.847 ms, 13.01 FPS.

El pipeline de escritura hace que en la práctica se esté escribiendo cada 3.8 ciclos de la PL a 100 Mhz, o cada 5.5 ciclos a 166 Mhz.

Period(ns)	Tiempo (ns)	numTransf	Media/transf	Overhead
10	38408566	1008000	38.104	3.810
10	78813375	2073600	38.008	3.801
6	37394400	1008000	37.098	6.183
6	76923600	2073600	37.097	6.183

Media/transf = Tiempo medio de cada transferencia (en ns).

Overhead = Cuantos ciclos extras a los ideales necesita para enviar los datos.

* Los tiempos de ejecución para las diferentes secuencias (4 tipos de matrices) son lo mismo dado que el overhead de acceso al controlador, oculta las latencias de memoria.

2.4. Notas y conclusiones:

Mover 1400x720 (1M) está en el orden de 40ms. Apenas 25 FPS...

Mover 1920x1080 (2M) está en el orden de 77ms. No más de 13 FPS...

El controlador de memoria de HLS solo encola 6 direcciones hasta una respuesta.

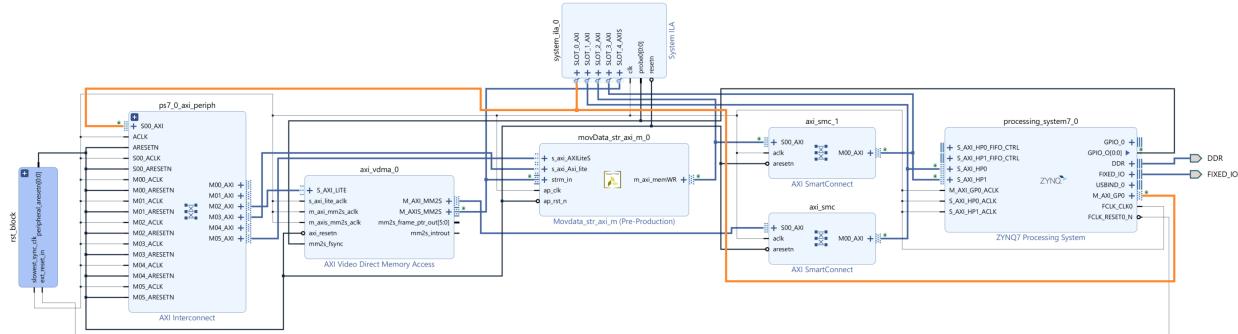
Evidentemente las escrituras simples que esperan las respuestas del canal B, son demasiado lentas. Para aumentar el ancho de Banda es necesario usar ráfagas de escritura y/o lazo abierto. Las ráfagas solo se pueden realizar si son direcciones consecutivas.

Circuito 3. Escritura a Memoria AXI-Master usando Burst (movData)

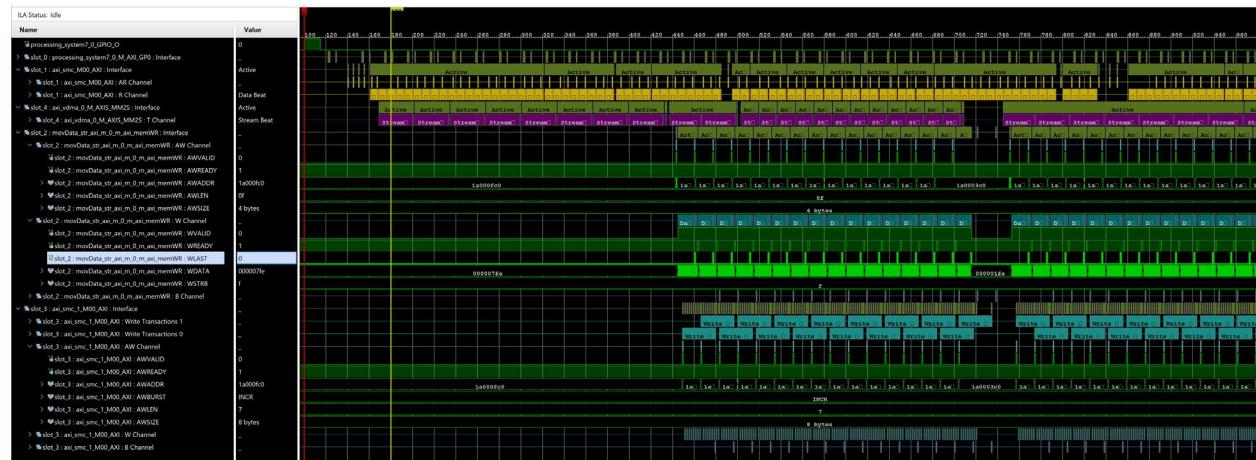
El circuito captura imagen (matriz bidimensional desde memoria) de 64 bits de ancho y escribe matriz del mismo tamaño de 32 bits usando una interfaz AXI-Master con ráfagas (burst).

Para el movimiento de datos se configura un AXI VIDEO DMAs para leer datos de memoria usando el puerto HP0 del Zynq (64 bits). Un core en HLS lee el stream y lo escribe como maestro AXI usando el puerto HP1. El core almacena la suma parte alta y baja de 32 bits. Se puede configurar el tamaño de la ráfaga de escritura (Burst).

**** Esto no aporta nada, no siendo que un core HLS escribe como máster con similar rendimiento a un Axi VideoDMA. SOLO es válido para escrituras “en orden”.**



Ejemplo transfiriendo 32x32 (1024). El core genera ráfagas de 16 escrituras (awlen 0xF) de 32bits (AWSIZE 4bytes). El AXI_smartConnect lo convierte en ráfagas de 8 (awlen 0x7) pero de 64 bits de ancho.



3.1 Notas y conclusiones:

Los tiempos de escritura son similares a la opción `path_through` de dos AXI Video DMA (como es esperable).

Se puede aumentar el clock y se observan similares resultados a `path through`. Con 166 Mhz (6 ns periodo) cerca del límite de la memoria. **SOLO es válido para escrituras “en orden”.**

Círculo 4. Escritura directa (Wr_data_dir)

El círculo captura imagen (matriz bidimensional desde memoria) de 64 bits de ancho y escribe matriz del mismo tamaño de 32 bits usando una escritura directa en el puerto HP1. Usa el canal WADRRES y WDATA en lazo abierto. Utiliza las señales **valid** y **ready** en cada canal.

Para el movimiento de datos se configura un AXI VIDEO DMAs para leer datos de memoria usando el puerto HPO del Zynq (64 bits). El IP-core en HLS lee el stream y escribe dos AXI-stream de 32 bits. Uno con la dirección y otro con el dato.

4.1. Proyecto HLS

Desde el directorio “movDataZynq\prj_hls” ejecutar el script ..\scripts\wr_data_dir_hls_script.tcl

```
movDataZynq\prj_hls>vitis_hls -f ..\scripts\wr_data_dir_hls_script.tcl
```

Genera la carpeta con la exportación del core IP “\prj_hls\ wr_data_dir \solution1\impl\ip”

* Las salidas son AXI-Stream. Es decir, si la señal de TREADY de algunas de las salidas no está a uno frena el pipeline. Si se cablea a uno el TREADY actual sin handshaking

4.1.a (opcional) Para abrir el proyecto HLS:

```
vitis_hls -p wr_data_dir
```

O bien en el GUI, open Project. Analizar los resultados de implementación. Importante mantener un initiation interval de 1 (ii=1) del bucle de lectura y escritura.

4.1.b (opcional) Simular y co-simular

El testbench envia un stream de 64 bits y revisa lo escrito por el IP-Core.

Se puede simular con el testbench provisto. La co-simulación requiere cambiar la interfaz de ap_ctrl_none a axi_lite o dejar por defecto ap_ctrl_hs

4.2. Proyecto Vivado

Crear proyecto para la tarjeta objetivo (Zybo Z-20). Usar Verilog como código por defecto.

Agregar el repositorio IP-Cores HLS (Project settings -> IP -> Repository)

Crear el Block Design, agregar el wrapper HDL. El Zynq está configurado con un reloj de 100 MHz (10 ns periodo)

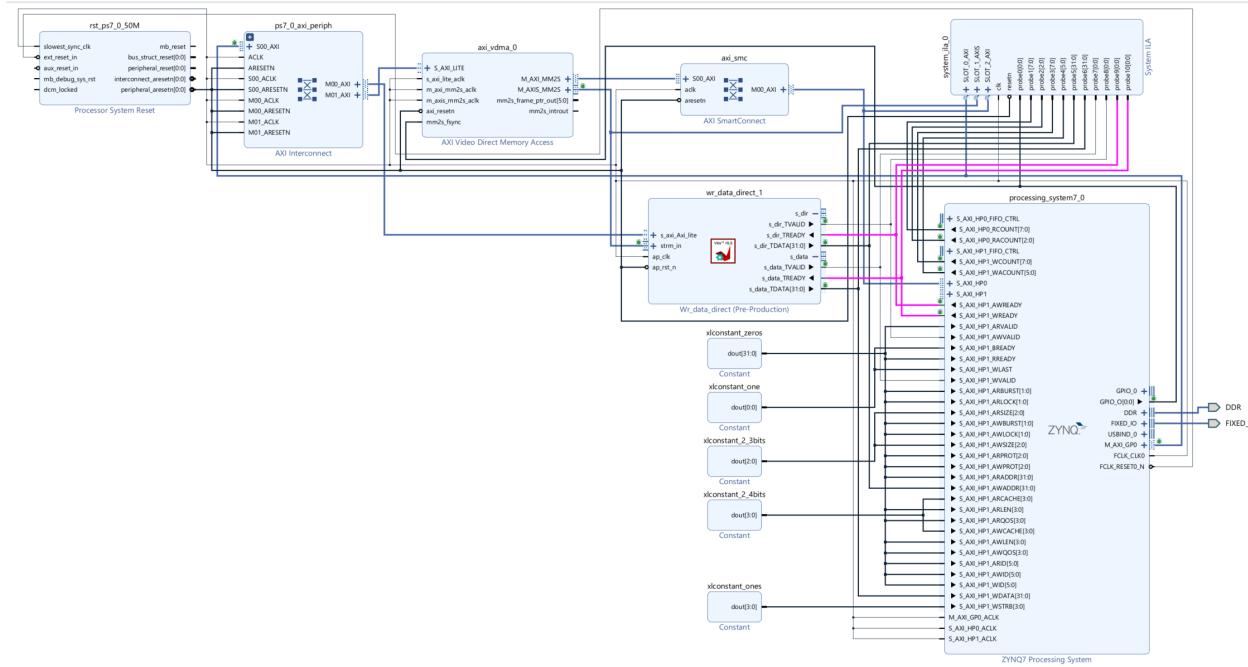
Existen dos scripts uno que crea el proyecto y llama al segundo (wrData_direct_db.tcl) que crea el diagrama de bloques. Desde la consola tcl y desde el directorio /movDataZynq/prj_vivado de Vivado se puede ejecutar:

```
source ../scripts/wrData_direct_vivado.tcl
```

Comentarios del proyecto:

- Axi_video_dma0. Configurado para leer memoria desde HPO y escribir un stream de 64 bits que entra al IP-Core wr_data_direct.
- El inicio del video DMA se lleva a cabo por un GPIOs del PS

- El IP-Core Escribe datos y dirección en los canales WADDRESS y WDATA directamente en el puerto HP1. Utiliza los handshakings TVALID – TREADY (resaltados los TVALID).
- El sistema ILA monitoriza el camino de datos. (1) Lectura de DDR; (2) salida stream 64 bits, (3) Escrituras de direcciones y datos de 32 bits del core. Incluye las señales de VALID y READY de cada canal; (4) puerto GP usado para configurar y leer el core IP.



4.3. Proyecto Vitis

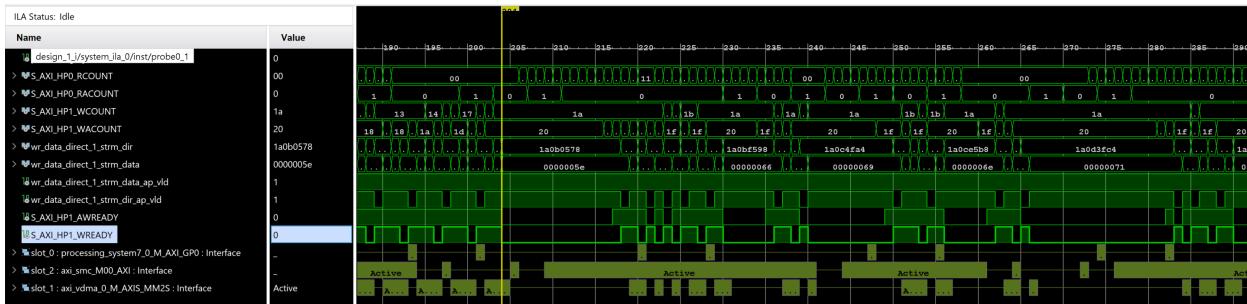
Implementar y generar bitstream. Exportar HW incluyendo bitstream.

Lanzar SDK y crear un proyecto C++ en baremetal (file -> new Application Project)

Agregar el fichero arm_c_code/cpyData_direct.cc

El programa genera una matriz de 64 bits donde contiene índices de posición en la parte baja y un número de secuencia en la parte alta. El hardware se supone que lo escribe en la posición indicada por los índices respecto de una dirección base. Existen 4 secuencias. Las 4 descriptas en 3.2 (secuencial, diagonal, vertical, invertida), más la otra opción diagonal.

Se muestra la captura de una transmisión de una matriz de 2M de la secuencia matrix_2. Se puede ver que la señal WREADY se baja regularmente. Como existe handshaking existe un “backpressure” al canal WADDRESS y luego hacia el *stream* de entrada.



Si se quita el handshaking (se quita las conexiones de AWREADY y WREADY) esto no genera detenciones, pero... AWREADY en el caso de la figura se baja y no es escrito ese valor. ¡La consecuencia es que se desincroniza desde allí en adelante!

Tiempos de ejecución para $720 \times 1400 = 1008000$ y $1080 \times 1920 = 2073600$. Reloj a 100 MHz y 166 MHz

Secuencia	Periodo(ns)	Tiempo (ms)	numTransf	Media/transf	Overhead
matriz 1	10	12.615	1008000	12.51	1.251
matriz 3	10	17.266	1008000	17.12	1.712
matriz 1	10	25.948	2073600	12.51	1.251
matriz 3	10	35.637	2073600	17.18	1.718

*

Secuencia	Period(ns)	Tiempo (ms)	numTransf	Media/transf	Overhead
Matrix 1	6	8.508	1008000	8.44	1.407
Matrix 3	6	17.401	1008000	17.26	2.877
Matrix 1	6	17.503	2073600	8.44	1.407
Matrix 3	6	35.853	2073600	17.29	2.882

Media/transf = Tiempo medio de cada transferencia (en ns).

Overhead = Cuantos ciclos extras a los ideales necesita para enviar los datos.

```
*****
* DATA MOVEMENT Direct Write core *
*****
Freq PS: 666666687, Freq PL: 166666666. Ratio: 4.00. TimerFreq: 33333333
** Calibrating the PS timer:
init_time: 2644134 cycles.
curr_time: 2644158 cycles.
calibrationPS: 24 PS clock cycles (0.072 us).
Programming VDMA_MATRIX_0 registers...
Initialize Matrix with content 1 (Sequential Data)
RunTime: 2836114 PS clock cycles (8516.9 us - 8.508 ms). FPS: 117.53.
Total Writes: 1008000. Success Transmitted Data
Initialize Matrix with content 2 (Diagonal Data)
RunTime: 5800429 PS clock cycles (17418.7 us - 17.401 ms). FPS: 57.47.
Total Writes: 1008000. Success Transmitted Data
Initialize Matrix with content 3 (vertical Data)
RunTime: 5807589 PS clock cycles (17440.2 us - 17.423 ms). FPS: 57.40.
Total Writes: 1008000. Success Transmitted Data
Initialize Matrix with content 4 (Inverted Data)
RunTime: 2835978 PS clock cycles (8516.5 us - 8.508 ms). FPS: 117.54.
Total Writes: 1008000. Success Transmitted Data
Initialize Matrix with content 5 (Diagonal x Data)
RunTime: 5694495 PS clock cycles (17100.6 us - 17.084 ms). FPS: 58.54.
Total Writes: 1008000. Success Transmitted Data
*****
* DATA MOVEMENT Direct Write core *
*****
Freq PS: 666666687, Freq PL: 166666666. Ratio: 4.00. TimerFreq: 333333343
** Calibrating the PS timer:
init_time: 2644833 cycles.
curr_time: 2644857 cycles.
calibrationPS: 24 PS clock cycles (0.072 us).
Programming VDMA_MATRIX_0 registers...
Initialize Matrix with content 1 (Sequential Data)
RunTime: 5833644 PS clock cycles (17518.5 us - 17.501 ms). FPS: 57.14.
Total Writes: 2073600. Success Transmitted Data
Initialize Matrix with content 2 (Diagonal Data)
RunTime: 11945028 PS clock cycles (35871.0 us - 35.835 ms). FPS: 27.91.
Total Writes: 2073600. Success Transmitted Data
Initialize Matrix with content 3 (vertical Data)
RunTime: 11960075 PS clock cycles (35916.1 us - 35.880 ms). FPS: 27.87.
Total Writes: 2073600. Success Transmitted Data
Initialize Matrix with content 4 (Inverted Data)
RunTime: 5833605 PS clock cycles (17518.3 us - 17.501 ms). FPS: 57.14.
Total Writes: 2073600. Success Transmitted Data
Initialize Matrix with content 5 (Diagonal x Data)
RunTime: 11848821 PS clock cycles (35582.0 us - 35.546 ms). FPS: 28.13.
Total Writes: 2073600. Success Transmitted Data
```

4.4 Notas y conclusiones:

La escritura a lazo abierto funciona razonablemente bien. Pero es muy importante hacerlo mirando las señales de TREADY de los canales de dirección y datos.

Dar por hecho que se escriben solo con TVALID lleva a desincronizaciones dir/dato. Peor aun quedan en las FIFOs internas de AW y W un valor sin utilizar y futuros usos comenzarán desincronizados

El funcionamiento de este IP-Core de este apartado, si no puede escribir un canal ($TREADY = 0$), bloquea la escritura de ambos canales. Esto genera detenciones que penaliza el rendimiento en torno a un 25%. Esto se puede optimizar con el uso de FIFOs independientes para cada canal (AW y W). Ver la siguiente sección donde se analiza este escenario.

Circuito 5. Escritura directa con FIFOs (Wr_data_dir)

El circuito es similar al anterior. En efecto usa el mismo IP-core (wr_data_direct) (1) que el circuito anterior, solo que agrega FIFOs antes de la escritura (2 y 3). El módulo adapter_axi (4) son conexiones que simplifican la conexión AXI-master. Adicionalmente utiliza un reloj de lectura (166 MHz) para HP0 y 200 MHZ del lado de la escritura en HP1. Luego se usa 125Mhz-142MHz para cncliur que el limite es el controlado de memoria.

5.1. Proyecto HLS (mismo que 4.1)

Desde el directorio “movDataZynq\prj_hls” ejecutar el script ..\scripts\wr_data_dir_hls_script.tcl

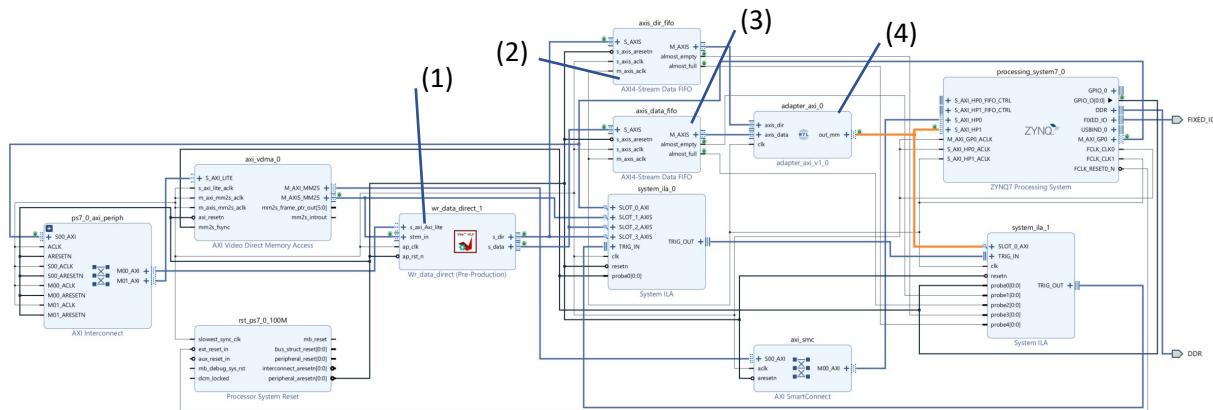
```
movDataZynq\prj_hls>vivado_hls -f ..\scripts\wr_data_dir_hls_script.tcl
```

Genera la carpeta con la exportación del core IP “\prj_hls\ wr_data_dir \solution1\impl\ip”

5.2. Proyecto Vivado

Existen dos scripts uno que crea el proyecto y llama al segundo (wrData_dir_FIFO_db.tcl) que crea el diagrama de bloque. Desde la consola tcl y desde el directorio /movDataZynq/prj_vivado de Vivado se puede ejecutar:

```
source ../scripts/wrData_dir_FIFO_vivado.tcl
```

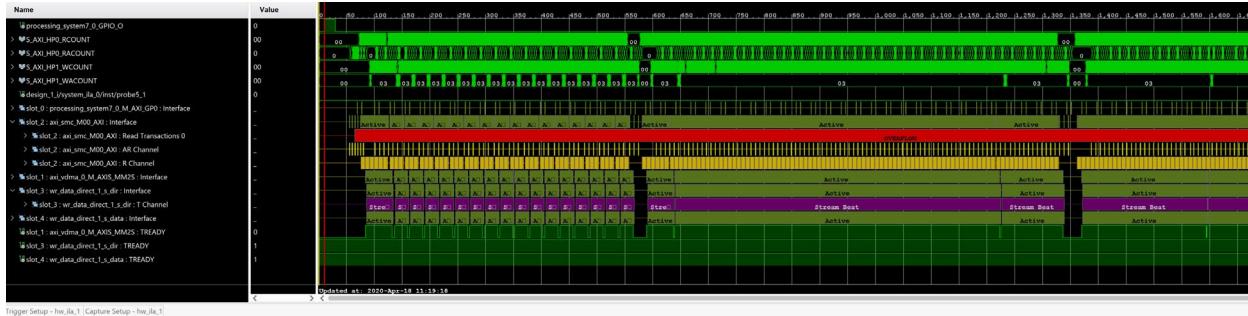


5.3. Proyecto Vitis

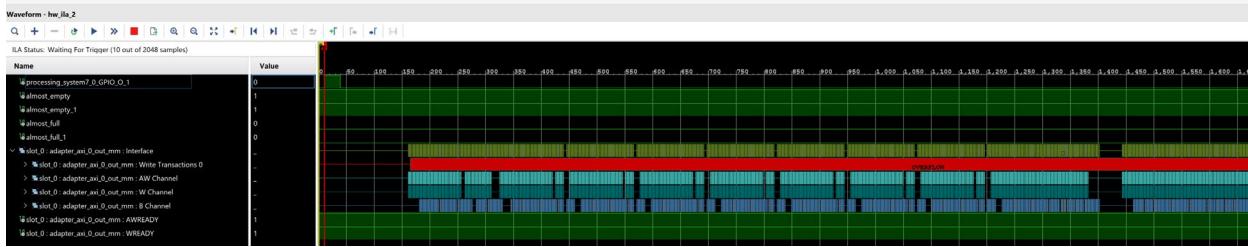
Implementar y generar bitstream. Exportar HW incluyendo bitstream. Lanzar Vitis y crear un proyecto C++ en baremetal (file -> new Application Project). Agregar el fichero arm_c_code/cpyData_direct.cc (El mismo que en 5.4).

Component	Cl...	Req...	Actu...	Range...
> Processor/Memory Clocks				
> IO Peripheral Clocks				
< PL Fabric Clocks				
<input checked="" type="checkbox"/> FCLK_CLK0	I	150	142.857	0.100000 :
<input checked="" type="checkbox"/> FCLK_CLK1	I	166	166.666	0.100000 :
<input type="checkbox"/> FCLK_CLK2	IO PLL	50	10.0000	0.100000 :
<input type="checkbox"/> FCLK_CLK3	IO PLL	50	10.0000	0.100000 :

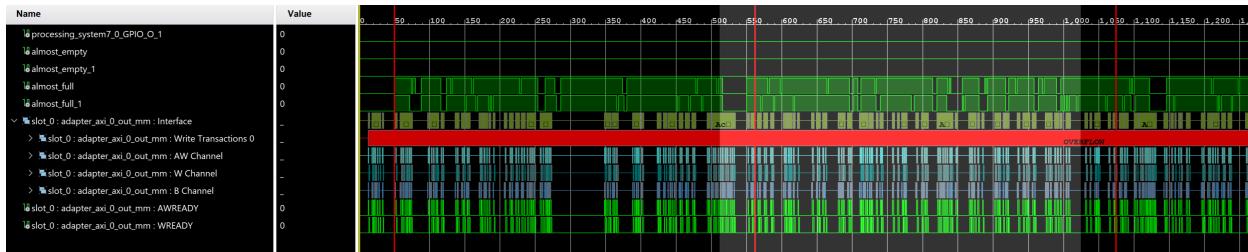
Para la matriz 1 y matriz 4, escribiendo a 200 MHZ. Nunca baje el **awready** o **wready** del HP1. Por tanto, nunca se llenan las FIFOs y no hace “backpressure” al IP-core.



Trigger Setup - hweila_1, Capture Setup - hweila_1



Sin embargo, con las secuencias 2 y 3 hay muchos momentos de detenciones.



Reloj 166-200

	Period (ns)	Tiempo (ms)	(ns)	numTransf	Media/transf	Overhead	FPS
Matrix 1	5	8.503	8503000	1008000	8.44	1.687	
Matrix 2	5	17.278	17278000	1008000	17.14	3.428	
Matrix 1	5	17.496	17496000	2073600	8.44	1.688	57.15
Matrix 2	5	35.483	35483000	2073600	17.11	3.422	28.18

Reloj 125-142

	Period (ns)	Tiempo (ms)	(ns)	numTransf	Media/transf	Overhead	FPS
Matrix 1	7	8.504	8504000	1008000	8.44	1.205	
Matrix 2	7	17.239	17239000	1008000	17.10	2.443	
Matrix 1	7	17.497	17497000	2073600	8.44	1.205	57.15
Matrix 2	7	35.483	35483000	2073600	17.11	2.445	28.18

5.4 Notas y conclusiones:

La escritura a lazo abierto funciona razonablemente bien. Pero es muy importante hacerlo mirando las señales de TREADY.

Con el uso de FIFOs en la escritura se mejora las escrituras solo a baja velocidad de escritura. El efecto de la detención de ambos caminos es despreciable

Usar un dominio de reloj más rápido en el lado de la escritura apenas tiene efecto.

Círculo 6. Múltiples lecturas y escrituras. Acceso a 4 HPs y 2 GPs (proj_4HP_2GP)

El circuito lee simultáneamente 2 matrices por HP0 y HP2 (de 64 bits de ancho) y escribe dos matrices de la misma dimensión, pero 32 bits de ancho en HP1 y HP3. La escritura usa buffers ping-pong en DDR, para que otros dos AXI Video DMA lean las matrices del mismo tamaño, pero con lecturas de 64 bits (**hace la mitad de lecturas**).

El objetivo es tener los 6 canales máster de la PL generando tráfico hacia y desde la DDR. Este circuito permitirá cuantificar las consecuencias y retardos de los accesos concurrentes a memoria.

Se han realizado dos IP-cores en HLS. Un primer IP-Core que realiza las escrituras en memoria en función de un *stream* de 64 bits de entrada (similar a circuito 4 y 5 - wr_data_direct), solo que permite configurar múltiples direcciones base para implementar un buffer ping-pong.

El segundo core “consume” el *stream* de lectura para realizar verificación del sistema.

6.1. Proyectos HLS

Se generan dos IP cores wr_data_dir_adv y collector_display.

Desde el directorio “movDataZynq\prjs_hls” ejecutar el script ..\scripts\wr_data_dir_adv_hls_script.tcl
movDataZynq\prjs_hls>vitis_hls -f ..\scripts\wr_data_dir_adv_hls_script.tcl

Genera la carpeta con la exportación del core IP “\prj_hls\ wr_data_dir_adv \solution1\impl\ip”

El IP-core recibe un *stream* de 64 bits y saca dos *streams* de 32 bits. Uno con dirección efectiva y otro con los 24 bits del número de pixel (cada pixel de la imagen tiene un numero) y pone en los 8 bits superiores un número de secuencia de frame (para reconocer diferentes imágenes). La dirección final son 8 bits del superiores de la parte alta apuntada por el *frame* pointer y los 24 de la dirección row*width_image + col.

Stream Entrada			Stream salida data		Stream salida dirección	
32 bits	16 bits	16 bits	8 bits	24 bits	8 bits	24 bits
Nro pixel	Índice col	Índice row	SecFrame	Nro pix(23:0)	FramePtr	Índice direcc

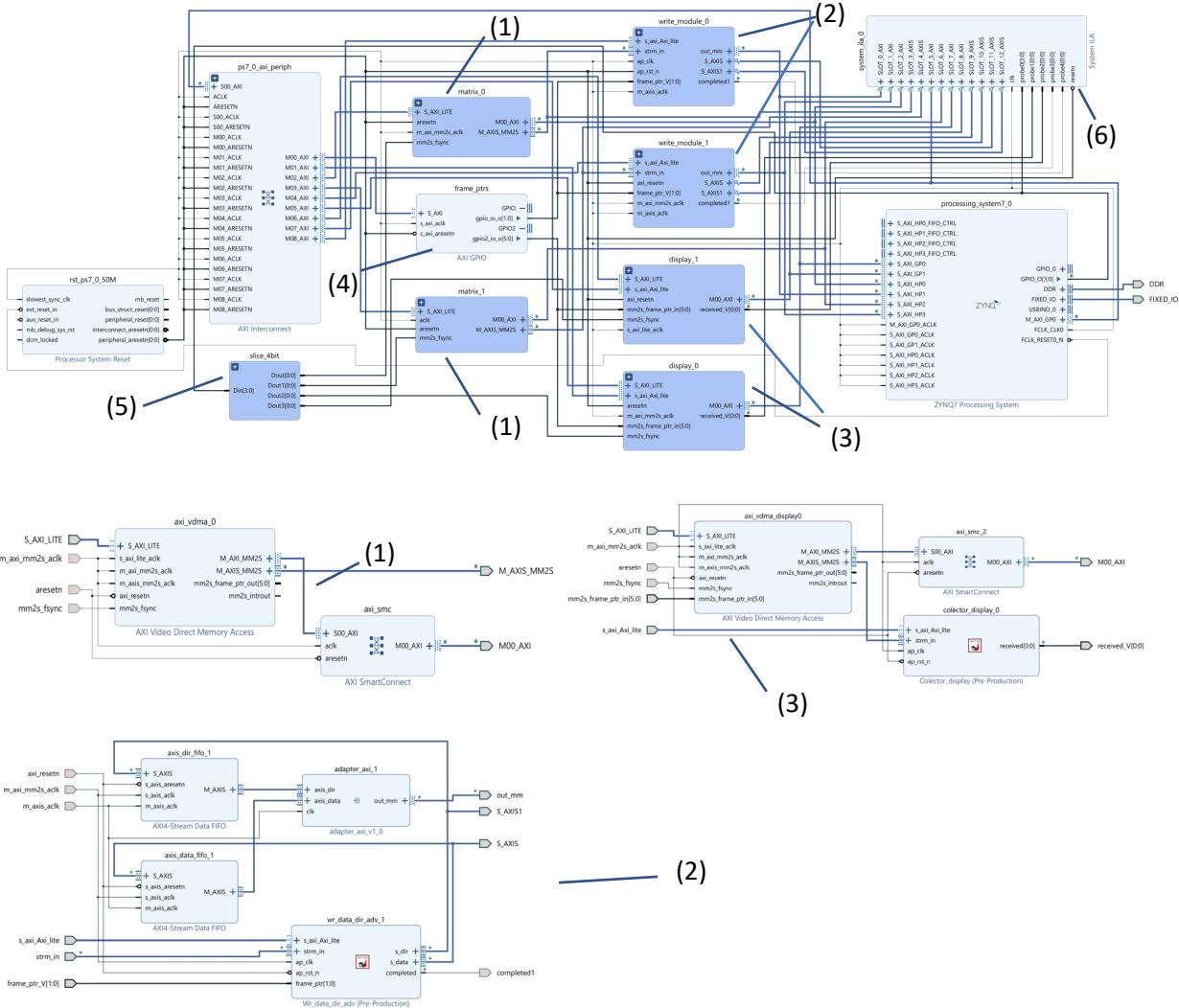
Desde el directorio “movDataZynq\prjs_hls” ejecutar el script ..\scripts\wr_data_dir_adv_hls_script.tcl
vitis_hls -f ..\scripts\collector_display_hls_script.tcl

Genera la carpeta con la exportación del core IP “\prjs_hls\ collector_display \solution1\impl\ip”

6.2. Proyecto Vivado

Existen dos scripts uno que crea el proyecto y llama al segundo (proj_4HP_2GP_db.tcl) que crea el diagrama de bloque. Desde la consola tcl y desde el directorio /movDataZynq/prj_vivado de Vivado se puede ejecutar:

```
source ../scripts/proj_4HP_2GP_vivado.tcl
```



Detalles del circuito:

- El módulo jerárquico **matrix_x** (1), lee una imagen de 64 bits de ancho de memoria y saca un *stream* de datos
- El bloque jerárquico **write_module_x** (2) tiene el bloque HLS **wr_data_dir_adv**, con dos FIFOs a la salida y el modulo **adapter_axi** para poder conectarlo directamente a los puerto HP.
- El bloque jerárquico **display_x** (3) tiene un **video_dma** que lee una imagen de 64 bits de la memoria, la salida en streaming es consumida por el bloque HLS **colector_display**.
- El GPIO (4) genera el frame activo tanto para las escrituras en memoria de los bloques **wr_data_dir_adv** (2) como de los bloques **diasplay** (3)
- La señal de disparo de los 4 AXI Video DMA se hacen desde los GPIO del PS. El bloque (5) conecta cada una de las entradas de disparo.
- Módulo ILA para monitorizar el diseño (6).

6.3. Proyectos Vitis

Existen dos proyectos diferentes. Uno de inicialización y otro de pruebas. Para ello:

Implementar y generar bitstream. Exportar HW incluyendo bitstream. Lanzar Vitis y crear un proyecto C++ en baremetal (file -> new Application Project).

6.3.1 Proyecto Inic sistemas

El primer proyecto inicializa el sistema y lanza las lecturas y escrituras durante una única iteración. Para el proyecto de “inicialización”: Agregar el fichero arm_c_code/cpy_direct_multipl_inic.cc

Resultados para 1M y 2 M puntos en los moviemientos

```
*****
* DATA MOVEMENT: two Data Read in HP. *
* two WR in HP, two RD in GP (64 bits) *

Freq PS: 666666687, Freq PL: 100000000. Ratio: 6.666667

** Calibrating the PS timer:
init_time: 3742066 cycles.
curr_time: 3742090 cycles.
calibrationPS: 24 PS clock cycles (0.072 us).
Programming VDMA MATRIX_0 registers...
Programming VDMA MATRIX_1 registers...
Programming VDMA DISPLAY_0 registers...
Programming VDMA DISPLAY_1 registers...
Initialize Mat 0, with content 1 (consecutive data)
Initialize Mat 1, with content 1 (consecutive data)
RunTimeRead HP: 4016081 PS clock cycles (12060.3 us - 12.048 ms). FPS: 83.00.
Total Writes: 1008000
RunTimeRead GP: 3422790 PS clock cycles (10278.6 us - 10.268 ms). FPS: 97.39.
Total Reads: 504000, 64 bit Wide

Success moved Data
Flag statistics (transmited for copro0): 1008000
Flag statistics (transmited for copro1): 1008000
display 0 - proc.elem: 504000 received_img: 4
display 1 - proc.elem: 504000 received_img: 4

*****
* DATA MOVEMENT: two Data Read in HP. *
* two WR in HP, two RD in GP (64 bits) *

Freq PS: 666666687, Freq PL: 100000000. Ratio: 6.666667

** Calibrating the PS timer:
init_time: 3742246 cycles.
curr_time: 3742250 cycles.
calibrationPS: 24 PS clock cycles (0.072 us).
Programming VDMA MATRIX_0 registers...
Programming VDMA MATRIX_1 registers...
Programming VDMA DISPLAY_0 registers...
Programming VDMA DISPLAY_1 registers...
Initialize Mat 0, with content 3 (vertical Data)
Initialize Mat 1, with content 3 (vertical Data)
RunTimeRead HP: 13347998 PS clock cycles (40084.1 us - 40.044 ms). FPS: 24.97.
Total Writes: 1008000
RunTimeRead GP: 3415463 PS clock cycles (10256.6 us - 10.246 ms). FPS: 97.60.
Total Reads: 504000, 64 bit Wide

Success moved Data
Flag statistics (transmited for copro0): 1008000
Flag statistics (transmited for copro1): 1008000
display 0 - proc.elem: 504000 received_img: 5
display 1 - proc.elem: 504000 received_img: 5

*****
* DATA MOVEMENT: two Data Read in HP. *
* two WR in HP, two RD in GP (64 bits) *

Freq PS: 666666687, Freq PL: 100000000. Ratio: 6.666667

** Calibrating the PS timer:
init_time: 3743724 cycles.
curr_time: 3743748 cycles.
calibrationPS: 24 PS clock cycles (0.072 us).
Programming VDMA MATRIX_0 registers...
Programming VDMA MATRIX_1 registers...
Programming VDMA DISPLAY_0 registers...
Programming VDMA DISPLAY_1 registers...
Initialize Mat 0, with content 1 (consecutive data)
Initialize Mat 1, with content 1 (consecutive data)
RunTimeRead HP: 8272837 PS clock cycles (24843.4 us - 24.819 ms). FPS: 40.29.
Total Writes: 2073600
RunTimeRead GP: 7043185 PS clock cycles (21150.7 us - 21.130 ms). FPS: 47.33.
Total Reads: 1036800, 64 bit Wide

Success moved Data
Flag statistics (transmited for copro0): 2073600
Flag statistics (transmited for copro1): 2073600
display 0 - proc.elem: 1036800 received_img: 7
display 1 - proc.elem: 1036800 received_img: 7

*****
* DATA MOVEMENT: two Data Read in HP. *
* two WR in HP, two RD in GP (64 bits) *

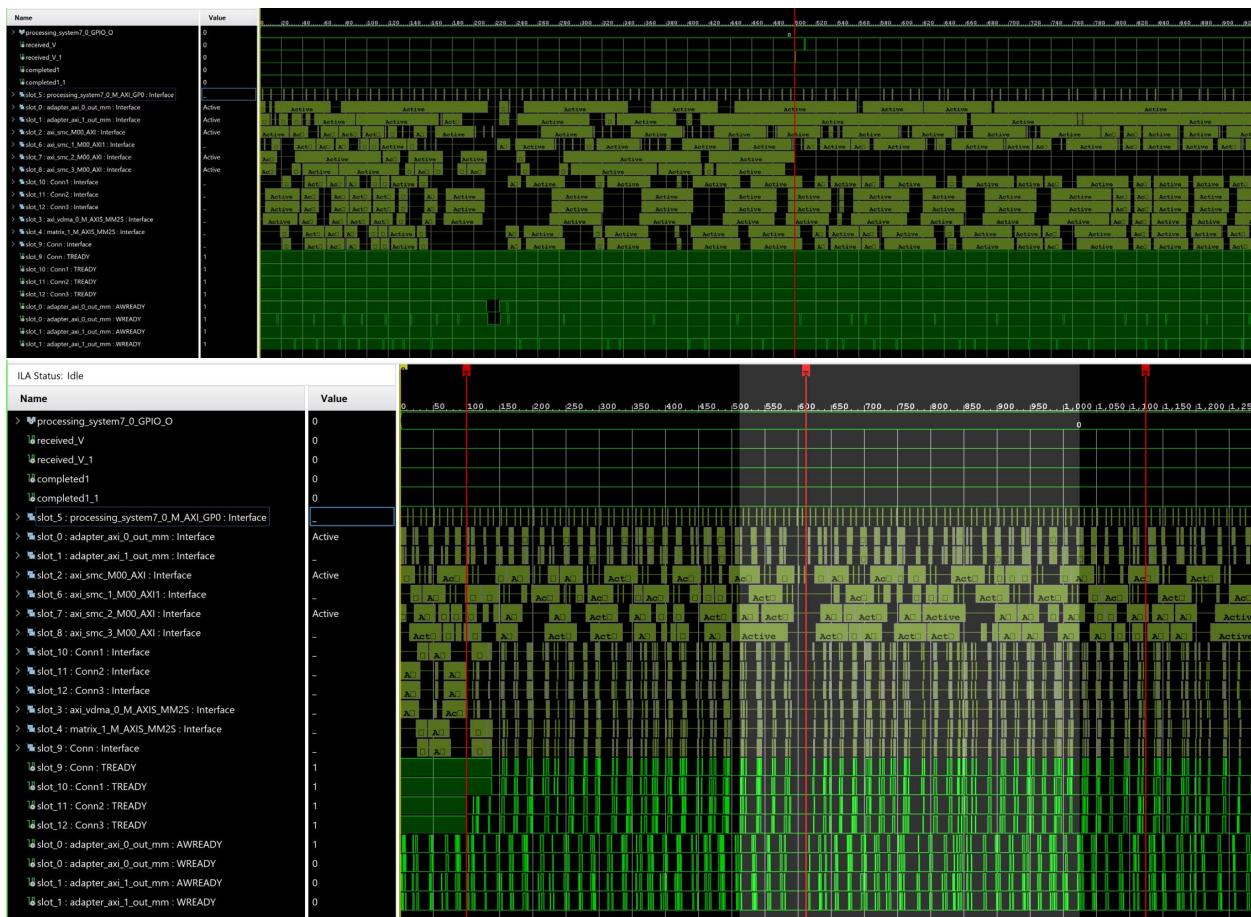
Freq PS: 666666687, Freq PL: 100000000. Ratio: 6.666667

** Calibrating the PS timer:
init_time: 37442252 cycles.
curr_time: 37442276 cycles.
calibrationPS: 24 PS clock cycles (0.072 us).
Programming VDMA MATRIX_0 registers...
Programming VDMA MATRIX_1 registers...
Programming VDMA DISPLAY_0 registers...
Programming VDMA DISPLAY_1 registers...
Initialize Mat 0, with content 3 (vertical Data)
Initialize Mat 1, with content 3 (vertical Data)
RunTimeRead HP: 27666246 PS clock cycles (83081.8 us - 82.999 ms). FPS: 12.05.
Total Writes: 2073600
RunTimeRead GP: 7310904 PS clock cycles (21954.7 us - 21.933 ms). FPS: 45.59.
Total Reads: 1036800, 64 bit Wide

Success moved Data
Flag statistics (transmited for copro0): 2073600
Flag statistics (transmited for copro1): 2073600
display 0 - proc.elem: 1036800 received_img: 6
display 1 - proc.elem: 1036800 received_img: 6
```

6.3.2 Proyecto funcionamiento continuo

El segundo proyecto permite lanzar el sistema con diferentes configuraciones. Agregar el fichero arm_c_code/cpy_direct_mult_contMuv.cc



Movimiento de imágenes 1080x1920 (full HD) secuenciales

Resumen tiempos totales en milisegundos.

TamPix 2073600	Una matriz	2 matrices	2Mat+Lect	1Mat/2Mat	Compl/2 mat	Compl/1Mat
Matrix_1	21.897	24.820	26.266	1.133	1.058	1.200
Matrix_3	35.728	74.719	82.503	2.091	1.104	2.309
Matrix_1y3	21.4/35.6	46.687	51.762			

Una Matriz: Lee Matriz de TamPix de 64 bits y escribe una matriz del mismo tamaño de 32 bits.

2 matrices: Lee y escribe 2 matrices a la vez. Usando HP0-HP1 y HP2-HP3

2Mat+Lect: Las dos lecturas escrituras anteriores y en paralelo leer dos matrices de TamPix/2 pero de 64 bits de ancho. Es decir, lee una de las matrices de TamPix de 32 bits escritas anteriormente.

6.4 Notas y conclusiones:

La sobrecarga de tiempo de ejecución por accesos concurrentes es relativamente baja para el caso de accesos secuenciales a 100 MHz desde la PL.

Es obvio que el acceso concurrente a memoria también penaliza la lectura de las matrices y no solo las escrituras.

En las mediciones para matriz R-COL, no hay detenciones excesivas. Apenas 7% más que un acceso secuencial.

La matriz 3 (acceso por columnas) que tiene mala localidad de acceso a la DDR hace que sature el acceso al controlador.

6.4.1 Comparación con el sistema ITCL

El caso real de ITCL es más laxo que este escenario. La matriz 1 de lectura es 1980 x 1080 y las escrituras tienen bastante localidad.

