Integración PS – PL

PEDRO IGNACIO MARTOS

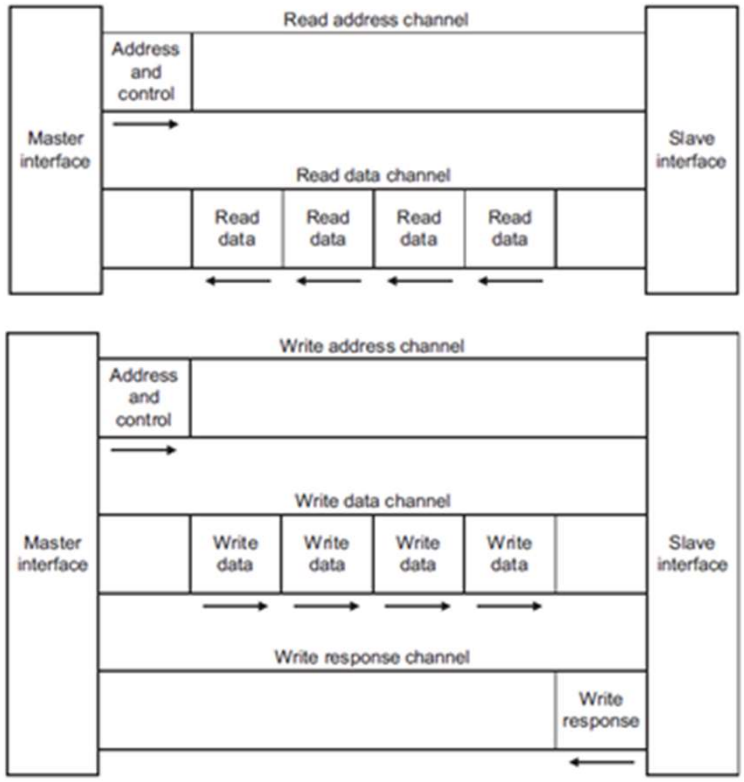
PMARTOS@FI.UBA.AR

Contenido

- **Bus AXI4**
  - AXI4 – Lite modo Esclavo
  - AXI4 – Lite modo Maestro
  - AXI4 – Full
- IP Propia compatible con AXI4

Bus AXI4

- Las señales de datos y control se dividen en 5 “Canales”
  - Read Address Channel
  - Read Data Channel
  - Write Address Channel
  - Write Data Channel
  - Write Response Channel

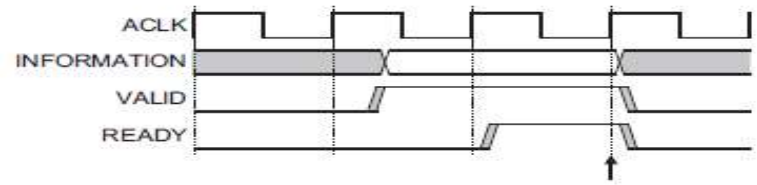


Protocolo

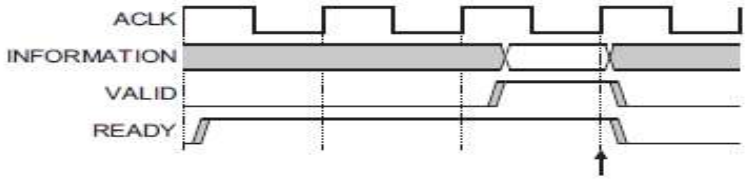
- Para la transferencia de datos se utiliza un protocolo VALID/READY
  - El ORIGEN fija y mantiene la señal VALID mientras haya DATOS disponibles para transferir
  - EL DESTINO fija y mantiene la señal READY mientras pueda aceptar datos
  - Los DATOS se transfieren mientras estén activas simultáneamente las señales VALID y READY
  - El ORIGEN envia DATOS o desactiva la señal VALID para finalizar la transferencia
  - El DESTINO desactiva la señal READY cuando no puede recibir mas DATOS

Protocolo

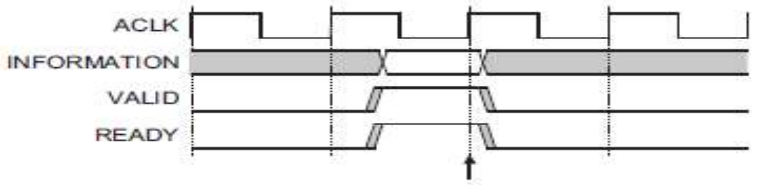
- Cada canal tiene su propio conjunto de señales VALID/READY
  - Address (read/write)
  - Data (read/write)
  - Response (write)
- Estas señales permiten gestionar distintas situaciones
  - Wait states: ciclos de espera dentro de la transferencia
  - Always Ready: la disponibilidad de datos regula la transferencia
  - Same cycle acknowledge: la transferencia se realiza dentro del mismo ciclo de reloj del bus



Inserting Wait States



Always Ready

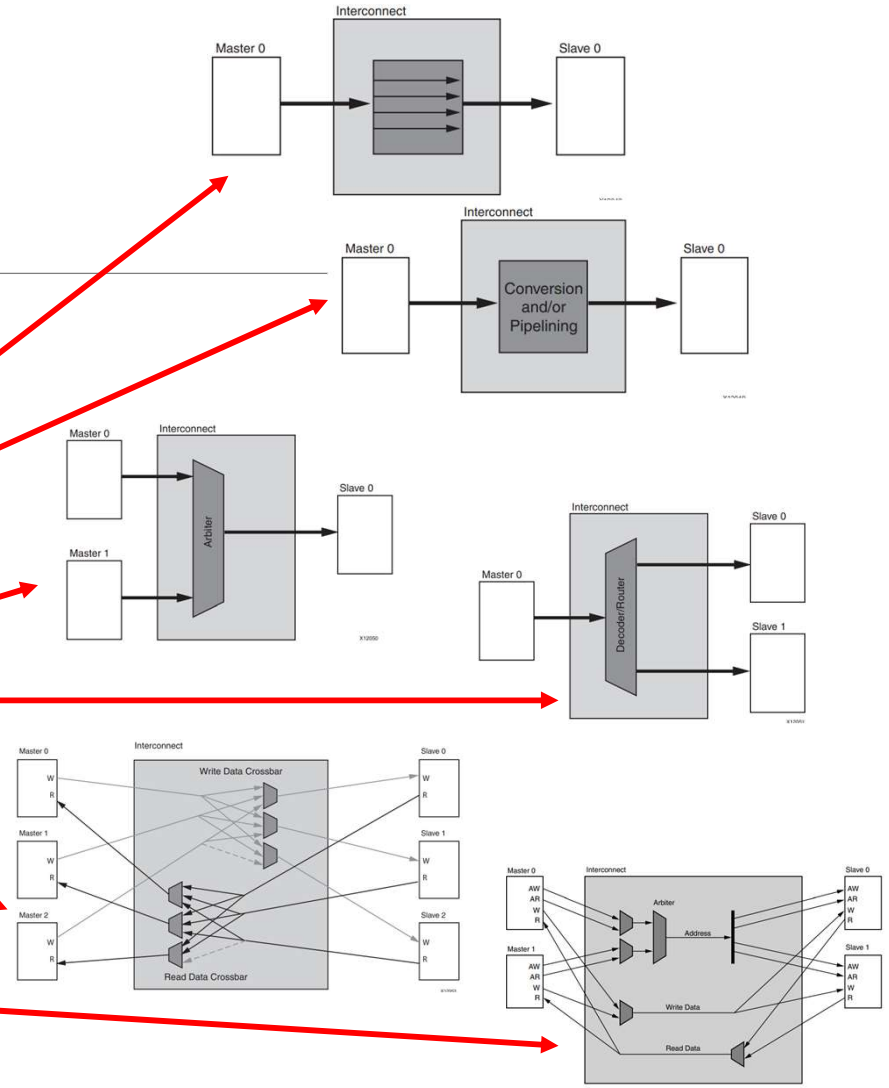


Same Cycle Acknowledge

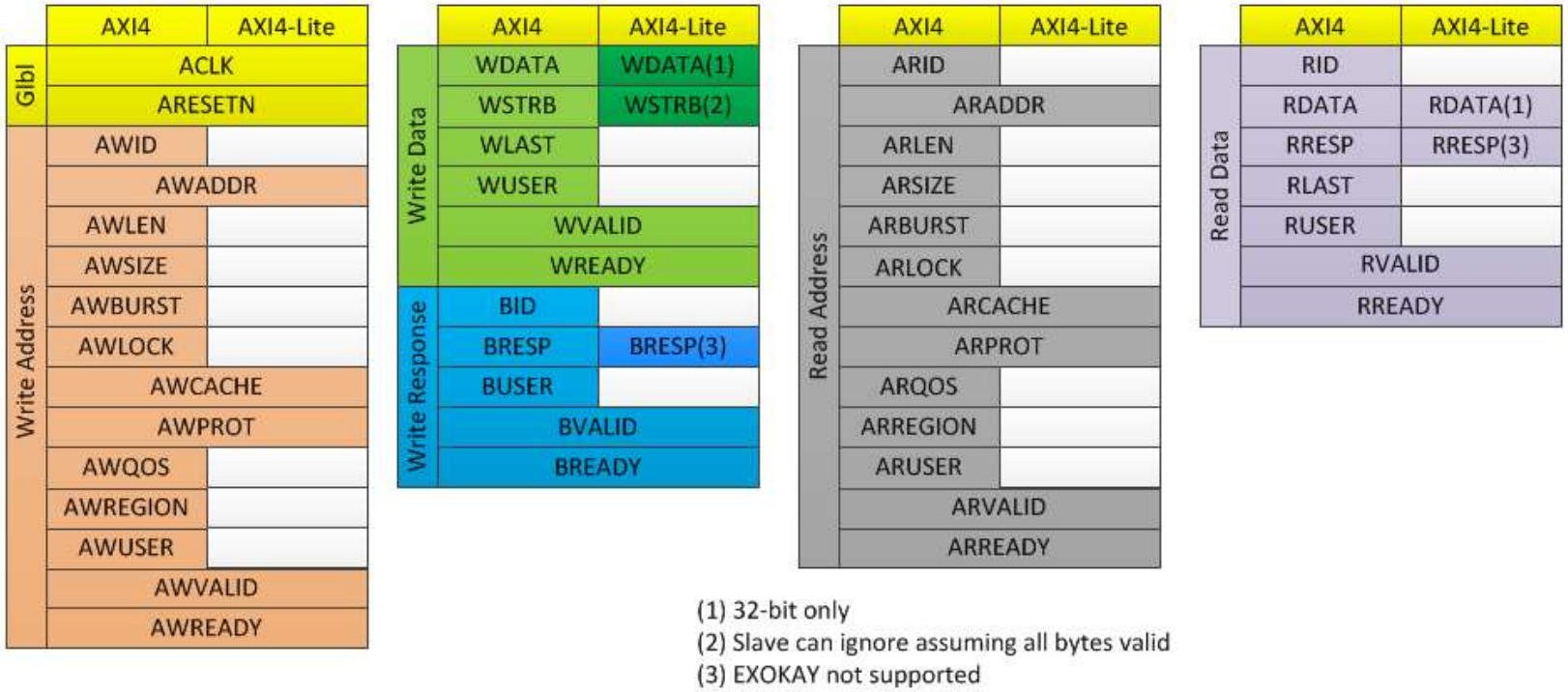
Bloque AXI Interconnect

- Permite distintos tipos de conexiones
  - Pass Through: Un solo dispositivo Maestro y un solo dispositivo Esclavo. El bloque es transparente
  - Solo conversión: similar a passthrough pero con conversión de tamaño de datos o temporización.
  - N a 1: Varios dispositivos Maestro acceden a un único dispositivo Esclavo (p.ej. un controlador de memoria)
  - 1 a N: Un dispositivo Maestro accede a varios dispositivos Esclavos (p.ej. un procesador y sus periféricos)
  - N a M Crossbar: los caminos y señales de lectura y escritura son independientes, puede realizarse una lectura y una escritura simultaneas
  - N a M Acceso Compartido: El bus es compartido, solo se puede hacer una operación (lectura o escritura) a la vez

La información detallada de este bloque esta en la hoja de datos DS768



Señales de cada canal



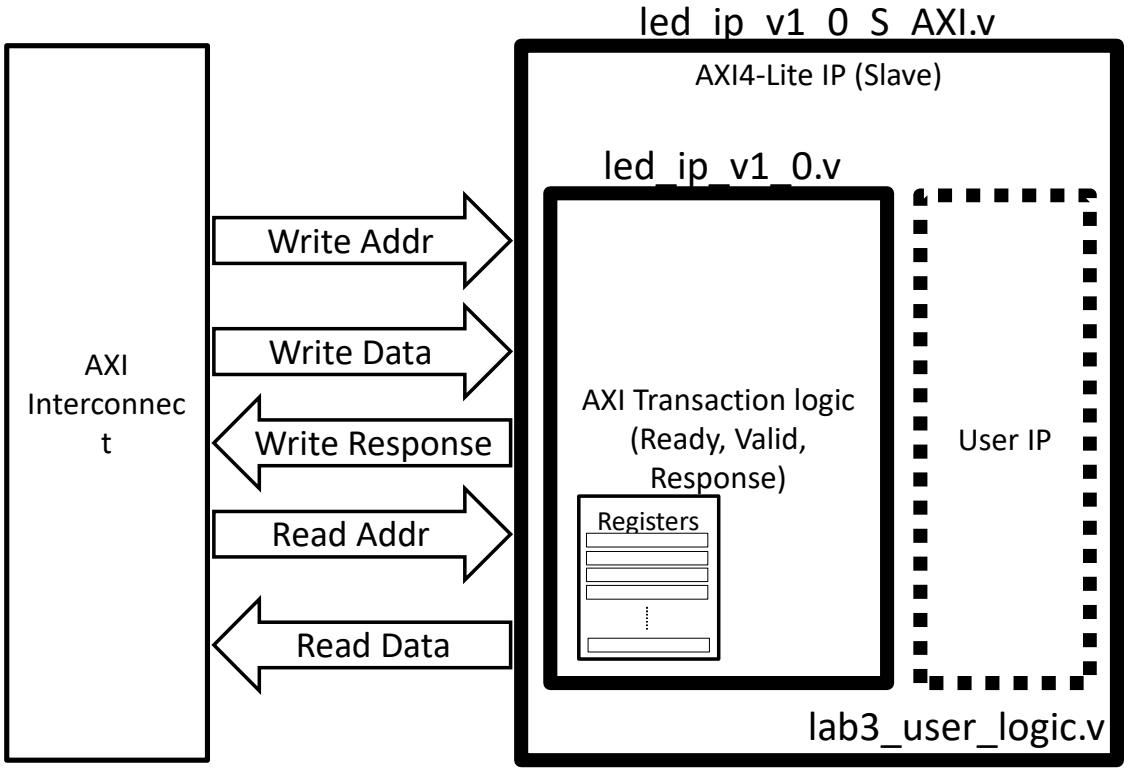
- La explicación detallada de las señales se puede ver en “AMBA AXI and ACE Protocol Specification Issue E” (ARM IHI 0022E)

Contenido

- Bus AXI4
  - **AXI4 – Lite modo Esclavo**
  - AXI4 – Lite modo Maestro
  - AXI4 – Full
- IP Propia compatible con AXI4

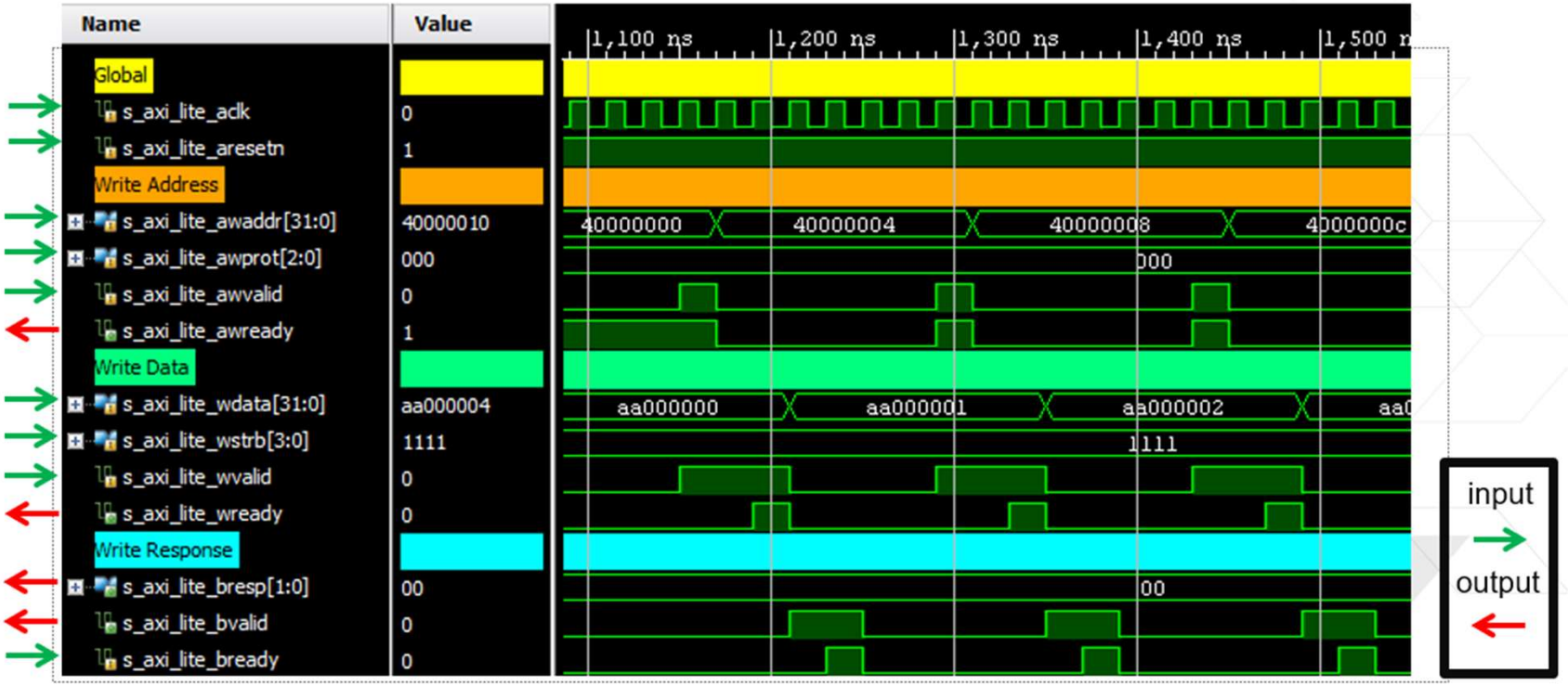
AXI4 – Lite modo Esclavo

- Es un subconjunto de AXI4 – Full
  - Tiene los mismos canáles pero menos señales.
- El dispositivo esclavo siempre responde a transacciones de lectura o escrituras iniciadas por el dispositivo maestro.
- Se transfiere un solo dato por transacción. No soporta ráfagas.
- Esta pensado para periféricos con bajas tasas de transferencia



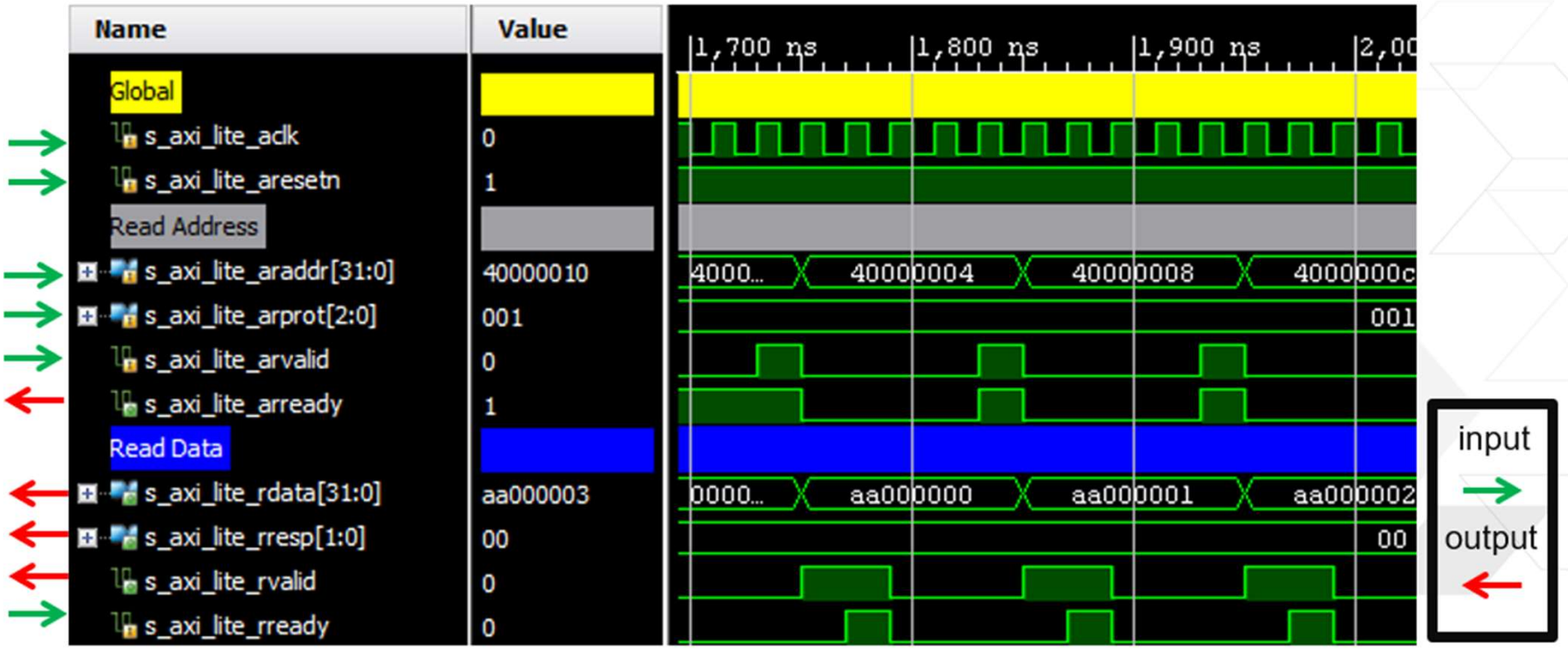
AXI4 – Lite modo Esclavo

- Ciclo de Escritura



AXI4 – Lite modo Esclavo

- Ciclo de Lectura

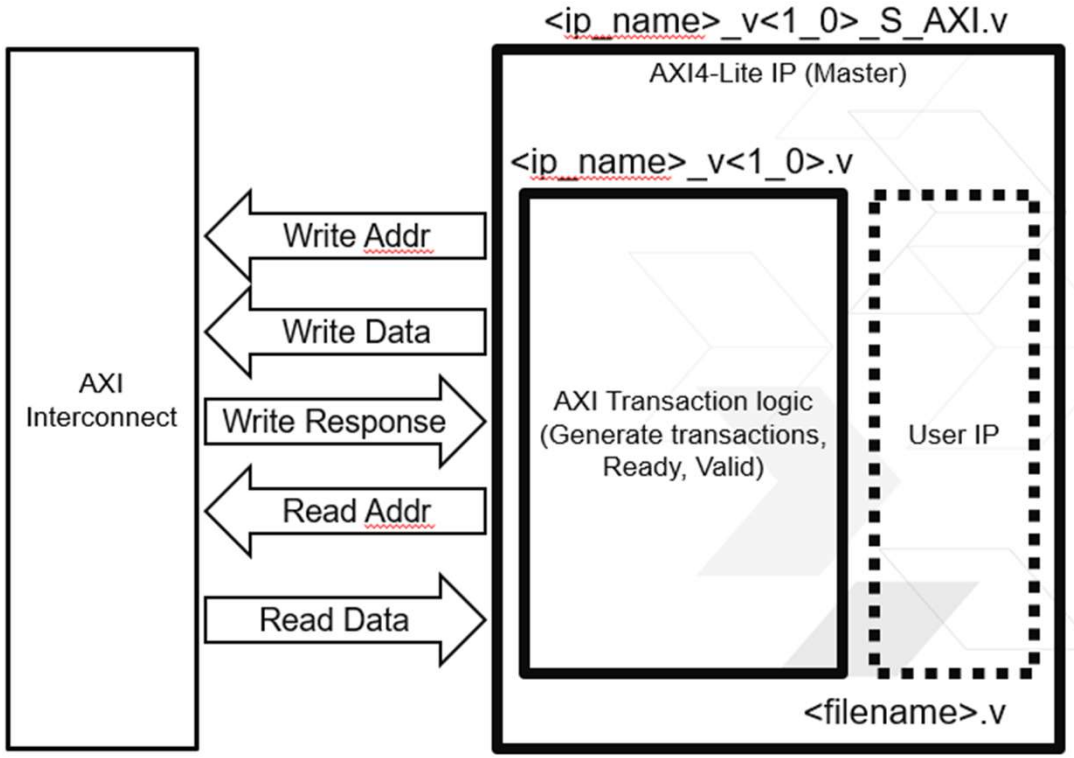


Contenido

- Bus AXI4
  - AXI4 – Lite modo Esclavo
  - **AXI4 – Lite modo Maestro**
  - AXI4 – Full
- IP Propia compatible con AXI4

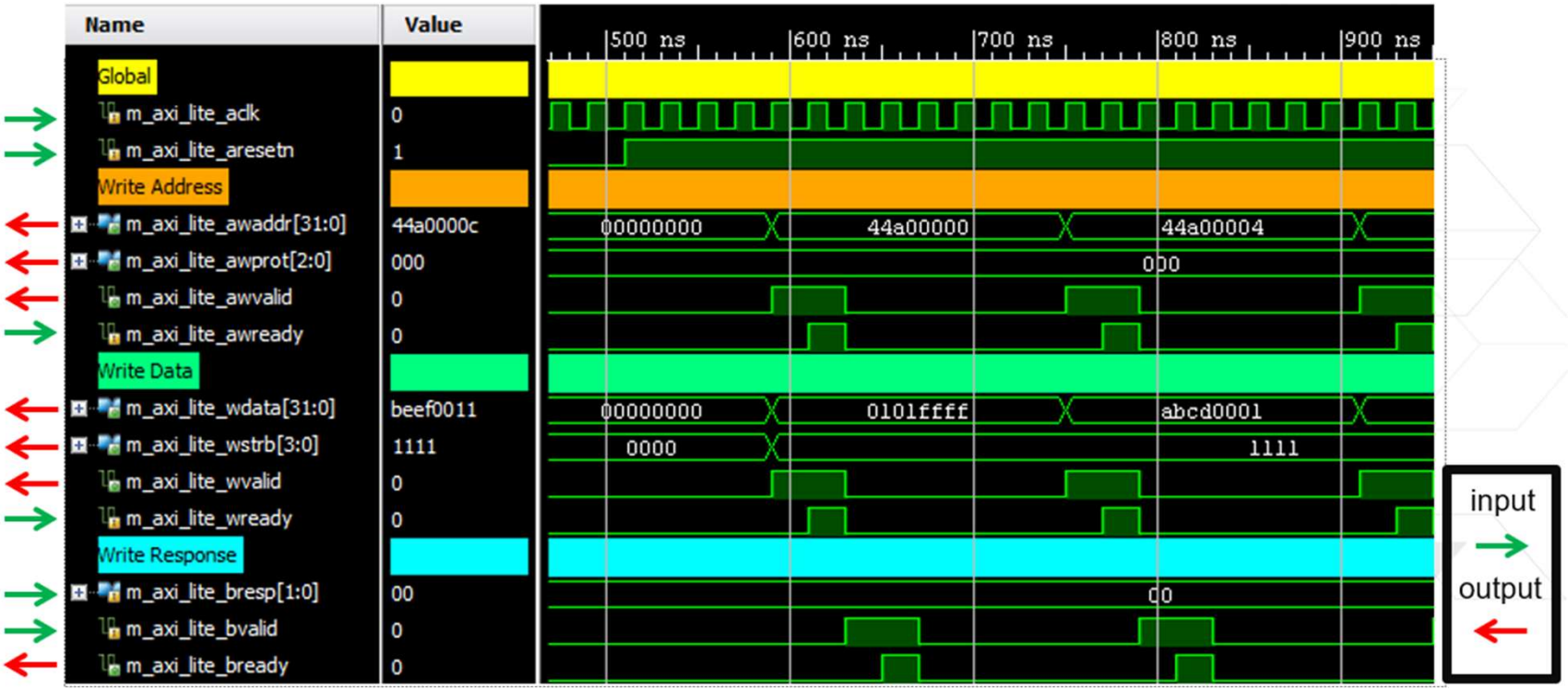
AXI4 – Lite modo Maestro

- Es un subconjunto de AXI4 – Full
  - Tiene los mismos canáles pero menos señales.
- El dispositivo maestro siempre inicia las transacciones de lectura o escritura.
- Se transfiere un solo dato por transacción. No soporta ráfagas.



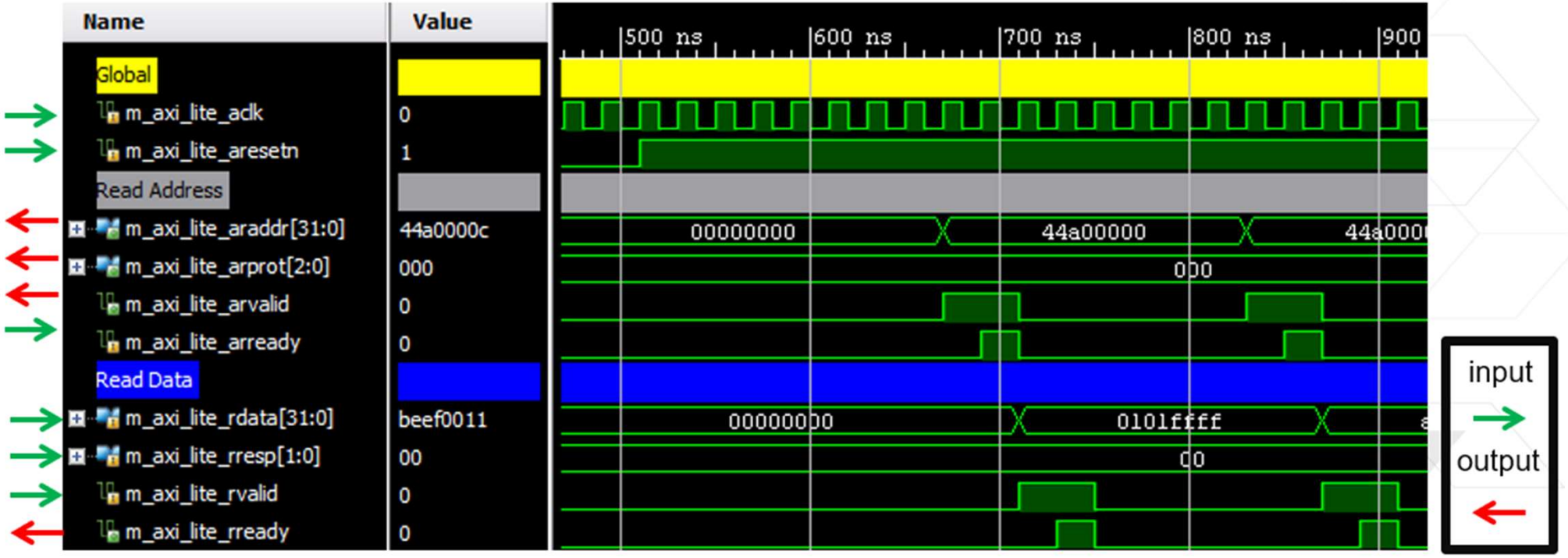
AXI4 – Lite modo Maestro

- Ciclo de Escritura



AXI4 – Lite modo Maestro

- Ciclo de Lectura



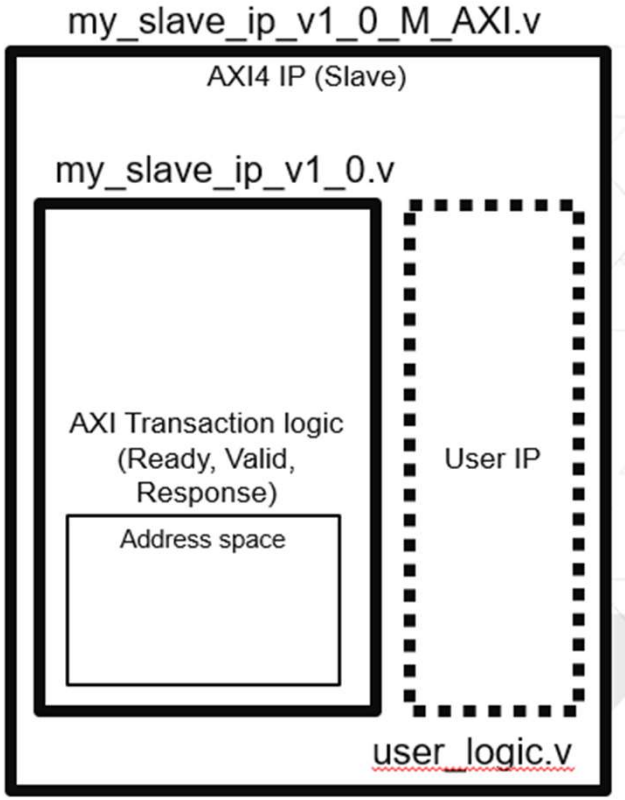
Contenido

- Bus AXI4
  - AXI4 – Lite modo Esclavo
  - AXI4 – Lite modo Maestro
  - **AXI4 – Full**

- IP Propia compatible con AXI4

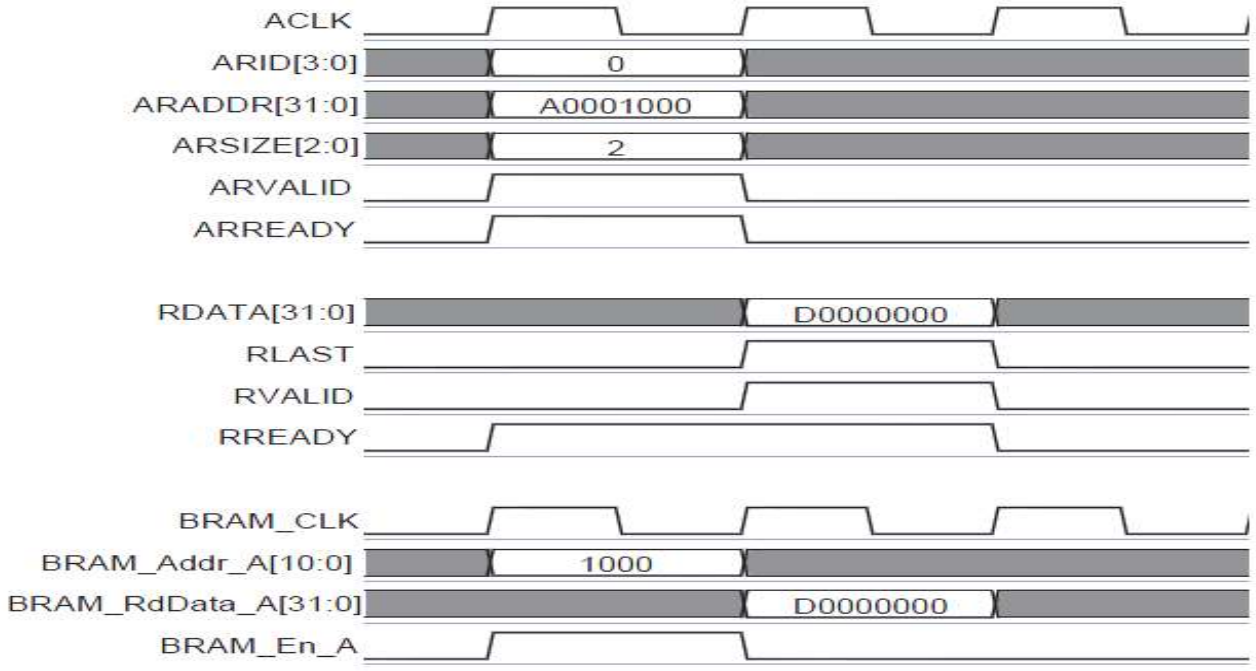
AXI4 – Full

- Mismas señales que en AXI4 – Lite
- Agregado de señales para poder hacer transferencias en ráfaga y ordenar la secuencia de transferencias.
- Las operaciones de lectura/escritura de un solo dato son similares a AXI4 – Lite
- Dispone de un espacio de memoria propio



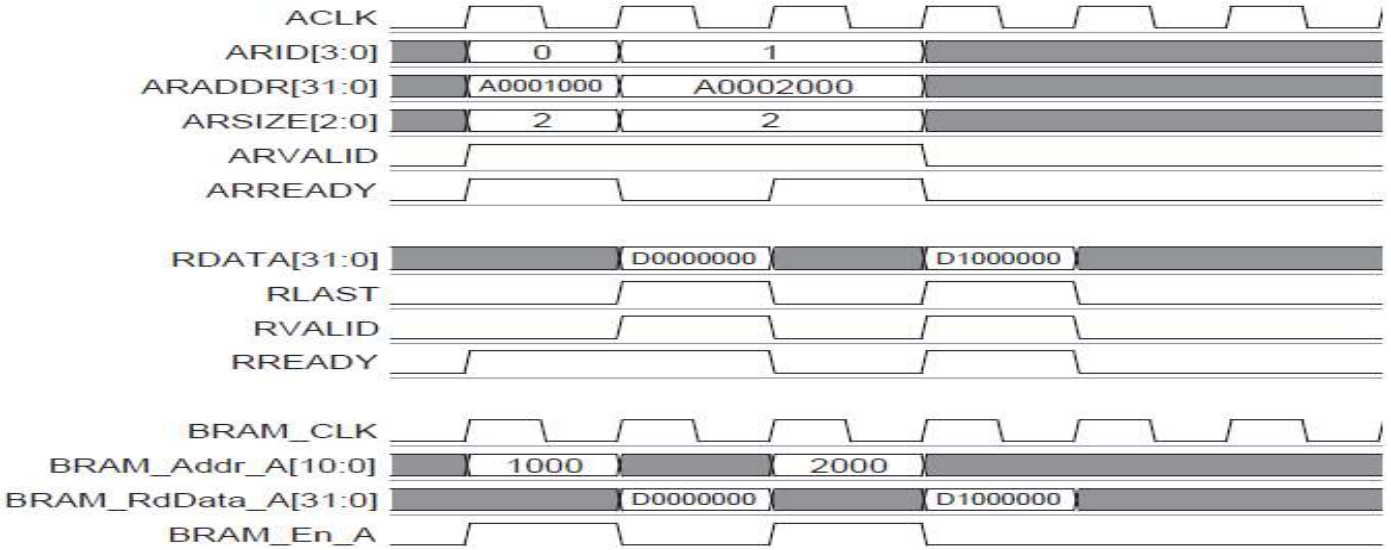
AXI4 – Full

- Ejemplo: Lectura de un solo dato



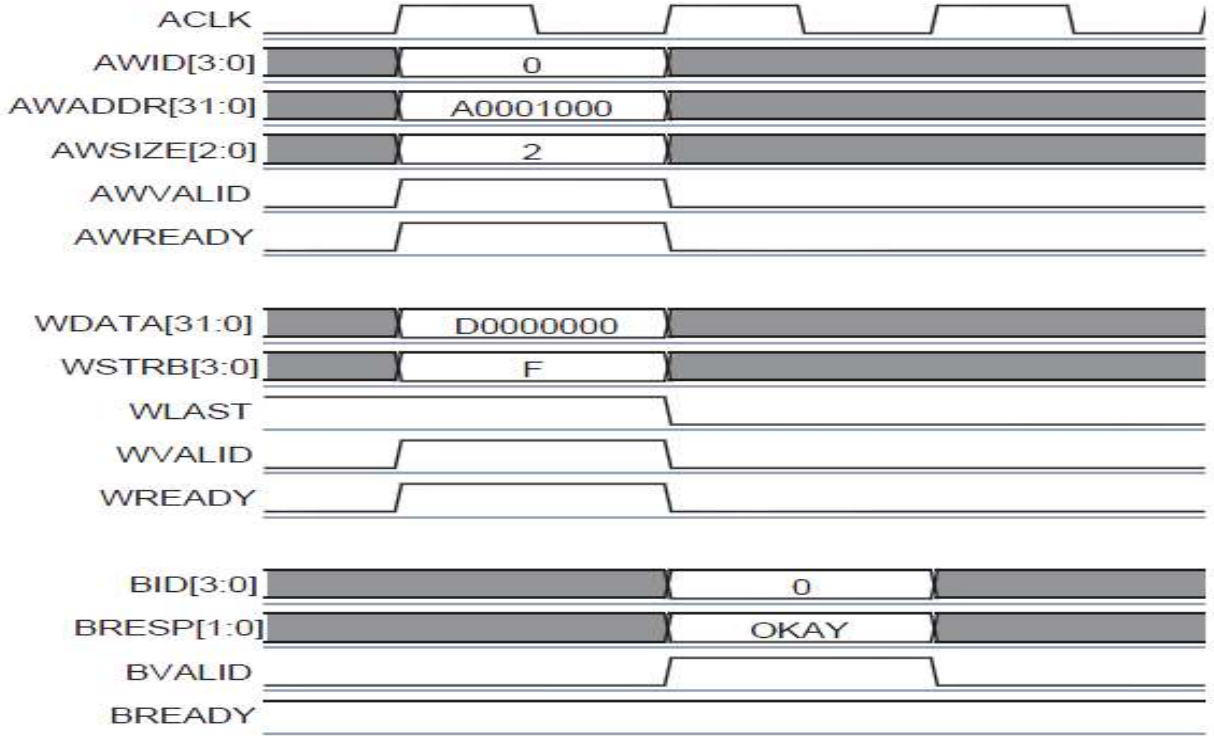
AXI - Full

- Ejemplo: Lectura de varios datos, no en ráfaga:
  - Cada lectura tiene su propio ReadID (ARID[3:0])
  - Cada lectura tiene su propio RLAST
  - ARSIZE = 2: cada lectura son 4 bytes



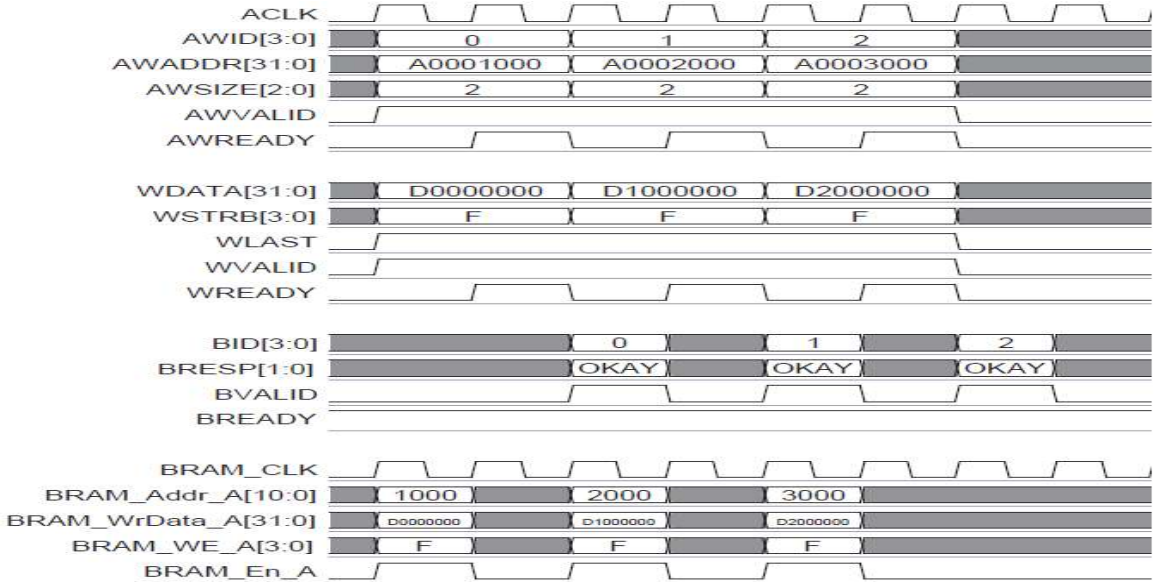
AXI4 – Full

- Ejemplo: Escritura de un solo dato



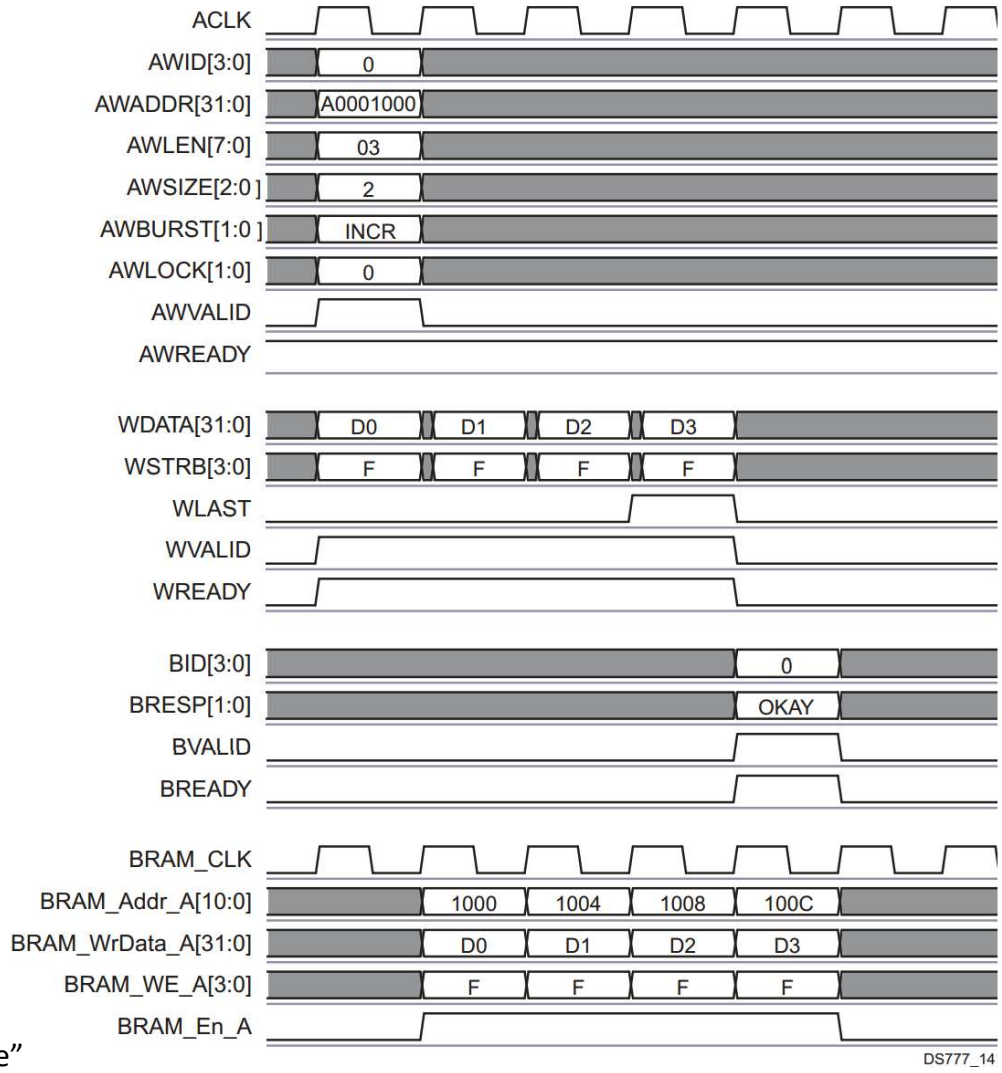
AXI - Full

- Ejemplo: Escritura de varios datos, no en ráfaga:
  - Cada escritura tiene su propio WriteID (AWID[3:0])
  - Cada escritura tiene su propio OKAY
  - WSTRB = 0xF: cada escritura es de una palabra completa



AXI4 – Full

- Ejemplo: 4 escrituras en ráfaga
  - Un solo AWID
  - AWLEN = 3 (tamaño de la ráfaga)
  - AWBURST = INCR: ráfaga incremental desde AWADDR
  - Cantidad de escrituras: AWLEN + 1
  - WSTRB = 0xF: se escriben palabras completas
  - LAST indica el fin de la rafaga
  - Un solo RESP
  - Una ráfaga no puede cruzar bloques de 4KB



Información mas detallada se puede encontrar en UG761: “AXI Reference Guide”

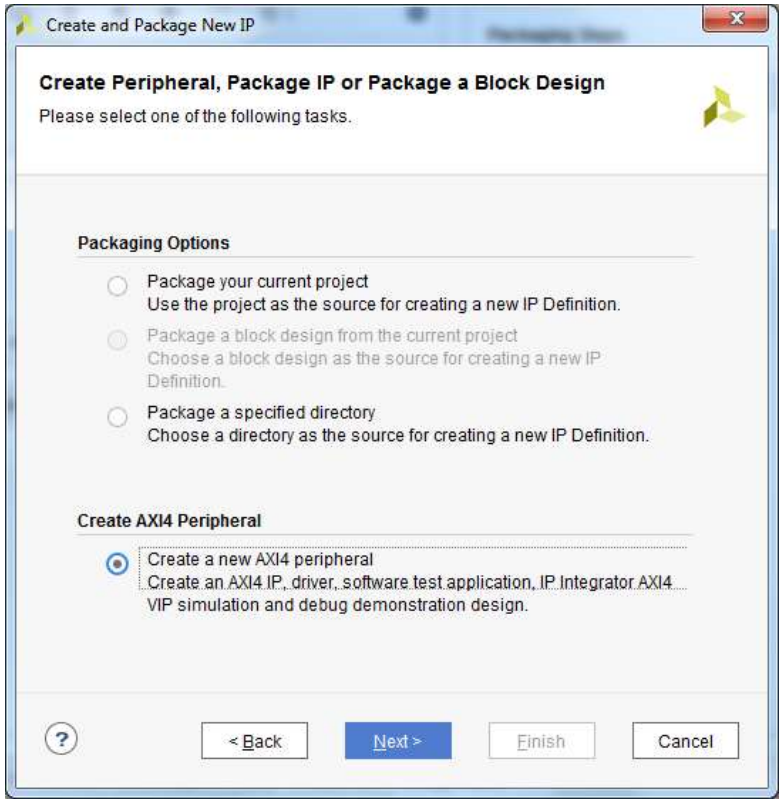
Contenido

- Bus AXI4
  - AXI4 – Lite modo Esclavo
  - AXI4 – Lite modo Maestro
  - AXI4 – Full

- **IP Propia compatible con AXI4**

Creación de periféricos AXI4

- Se utiliza el asistente “Create and Package IP”
- Permite generar plantillas para conectar un perférico al bus AXI
  - AXI Lite / Full / Stream
  - Maestro / Esclavo
- Tambien genera:
  - El software controlador
  - Una aplicación de prueba
  - Un modelo de bus (BFM – Bus Functional Model) para simulación
- Funciona de manera similar a la gestión de IP propia del bloque PL



Plantilla AXI4 – Lite

- Plantilla HDL con la implementación de la interfaz AXI4 – Lite con datos de 32bit
- Se especifica la cantidad de registros del periférico (mínimo 4)
- La plantilla permite realizar lectura/escritura de los registros
- La IP propia se conecta a los registros y puede tener su propio diseño jerarquico

```
case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
  2'h0:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_inc
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 0
        slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8],
      end
  2'h1:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_inc
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 1
        slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8],
      end
  2'h2:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_inc
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 2
        slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8],
      end
  2'h3:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_inc
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 3
        slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8],
      end
```

Plantilla AXI4 – Full

- Plantilla HDL con la implementación de la interfaz AXI4 – Full con datos de 32bit
- Soporta transferencias en ráfaga
- Se puede especificar el tamaño del espacio de memoria (hasta 1024 Bytes)
- La plantilla genera código de ejemplo implementando BlockMemory, la IP propia puede utilizarla o conectarse directamente

```
case (S_AXI_AWBURST)
  2'b00: // fixed burst
  // The write address for all the beats in the transaction are fixed
    begin
      axi_awaddr <= axi_awaddr;
      //for awsize = 4 bytes (010)
    end
  2'b01: //incremental burst
  // The write address for all the beats in the transaction are increments by
    begin
      axi_awaddr[C_S_AXI_ADDR_WIDTH - 1:ADDR_LSB] <= axi_awaddr[C_S_AXI_ADDR_W
      //awaddr aligned to 4 byte boundary
      axi_awaddr[ADDR_LSB-1:0]  <= {ADDR_LSB{1'b0}};
      //for awsize = 4 bytes (010)
    end
  2'b10: //Wrapping burst
  // The write address wraps when the address reaches wrap boundary
    if (aw_wrap_en)
      begin
        axi_awaddr <= (axi_awaddr - aw_wrap_size);
      end
    else
      begin
        axi_awaddr[C_S_AXI_ADDR_WIDTH - 1:ADDR_LSB] <= axi_awaddr[C_S_AXI_ADDR_
        axi_awaddr[ADDR_LSB-1:0]  <= {ADDR_LSB{1'b0}};
      end
  default: //reserved (incremental burst for example)
    begin
      axi_awaddr <= axi_awaddr[C_S_AXI_ADDR_WIDTH - 1:ADDR_LSB] + 1;
      //for awsize = 4 bytes (010)
```

Archivos creados

- Component.xml: Descripcion en formato XACT
- BD: script TCL con el Block Diagram
- Drivers:
  - Codigo C con el controlador y la aplicación de ejemplo
  - Funciones de lectura y escritura de los registros o zona de memoria
- HDL: plantilla en Verilog o VHDL

```
XStatus LED_IP_Reg_SelfTest(void * baseaddr_p)
{
    ------------------------------------------------------------

    xil_printf("******************************\n\r");
    xil_printf("* User Peripheral Self Test\n\r");
    xil_printf("******************************\n\n\r");

    /*
     * Write to user logic slave module register(s) and read back
     */
    xil_printf("User logic slave module test...\n\r");

    for (write_loop_index = 0 ; write_loop_index < 4; write_loop_index++)
      LED_IP_mWriteReg (baseaddr, write_loop_index*4, (write_loop_index+1
      READ_WRITE_MUL_FACTOR);
    for (read_loop_index = 0 ; read_loop_index < 4; read_loop_index++)
      if ( LED_IP_mReadReg (baseaddr, read_loop_index*4) != (read_loop_in
      +1)*READ_WRITE_MUL_FACTOR){
        xil_printf ("Error reading register value at address %x\n", (int)
        baseaddr + read_loop_index*4);
        return XST_FAILURE;
      }
```

Practica 3

- En esta Práctica se ampliará el sistema agregando IP propia y conectándola a través de buses AXI4.
- Se agregará un controlador de memoria AXI BRAM y su memoria asociada
- Se agregará un periférico de salida a través de una plantilla