Analisis de desempeño: Communication on Chip

David Ricardo Martínez Hernández **Código:** 261931
Sergio Ándres Zapata Palomino **Código:** 261261

***Resumen*— En este informe se analizan las principales características de un bus simple, y como este es descrito mediante código en System C. Mediante este código se diseñan nuevos sistemas de buses simples observando así las relaciones de precedencia entre los master del bus. También se observan criterios en las métricas de desempeño, como la latencia, el throughput y el nivel de utilización. Finalmente se describen diferentes políticas de arbitraje para así establecer mediante métricas de desempeño cual es mejor. A partir de los resultados obtenidos se plantean las coclusiones.**

***Palabras clave*— Bus, Master, Slave, SystemC.**

I. Introducción

Los sistemas constituidos sobre un solo chip deben poseer redes que sean capaces de comunicar los generadores de información maestros con los receptores o esclavos.
Estas redes deben tener capacidad de interactuar dentro el sistema minimizando las pérdidas en energía y velocidad en la transacción de los paquetes que viajan a través de ellos, existen muchas implementaciones como por mencionar **Mbus**, **Simple Bus**, **NoC**, cada una de ellas con diferentes formas de abordar el problema.
En este laboratorio se modificara un sistema Simple Bus, teniendo la posibilidad de implementar el modelo de comunicación, y definir la política de arbitraje a **TMDA**, luego comparar el desempeño con una política de arbitraje de prioridad fija.
Luego cuantificar el desempeño por las métricas de latencia, Thoughtput y nivel de utilización.

- **Nivel de Utilización:** que debe mostrar cuanto de la estructura de comunicación se está utilizando.
- **Latencia:** es una medida del tiempo que el sistema tarda en producir resultados.
- **Throughput:** es la cantidad de trabajos o tareas completadas por el sistema por unidad de tiempo.

II. Marco Teórico

A. Bus

Es un sistema digital que transfiere datos entre los componentes de una computadora o entre computadoras. Está formado por cables o pistas en un circuito impreso, dispositivos como resistores y condensadores además de circuitos integrados [2]. Existen diferentes tipos de buses, uno de ellos es el bus simple el cual se caracteriza por no tener un sistema de *pipelining* , ni de transacción dividida ni tampoco posee estados de espera por parte de los Master. Un bus permite la comunicación entre un maestro y un esclavo. La forma en la que se realiza esta comunicación se puede dividir en interfaces:

- **Blocking:** La información se transmite en ráfagas.

Fig. 1: Diagrama Round Robin

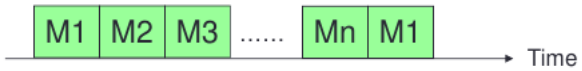

Fig. 2: Diagrama TDMA

- **Non-blocking**: La información se transmite un dato a la vez de acuerdo al ciclo de reloj.
- **Direct:** Su principal característica es que tiene un acceso directo al bus y al esclavo.

B. Políticas de Arbitraje.

1) Round Robin: Es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.
Una forma sencilla de entender el round robin es imaginar una secuencia para ”tomar turnos” (ver Fig 1).

2) TDMA (Time Division Multiple Access): Es un método que permite a varios elementos compartir el mismo canal dividiendo la prioridad en intervalos de tiempo (Ver Fig 2)

3) Fixed priority: A cada master se le asigna un número de prioridad, entre menor sea el número mayor sera la prioridad del master. Este número siempre sera respetado durante todo el proceso (ver Fig. 3):

III. Análisis de Resultados

Para poder realizar esta práctica es necesario ejecutar el ejemplo “simple_bus”, el cual se encuentra en la carpeta ***systemc-2.3.0/examples/sysc/simple_bus***, se utilizó el Makefile desarrollado durante el semestre para la correcta ejecución de SystemC.

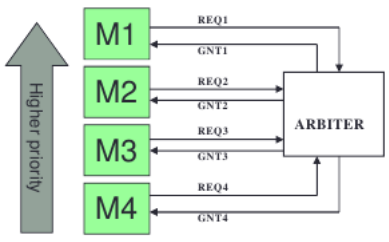

Fig. 3: Diagrama Fixed priority

A. Compilación y ejecución del ejemplo simple_bus de SystemC-2.3.0.

Al ejecutar el ejemplo sin ninguna modificación, ejecutando en la consola *make all*, se obtuvo el siguiente resultado (Fig. 4)

```
File  Edit  View  Search  Terminal  Help
6.5e+06 top.master_d : mem[78:87] = (146, 160, 17a, 1a7)
6.6e+06 top.master_d : mem[78:87] = (151, 16c, 187, 1b5)
6.7e+06 top.master_d : mem[78:87] = (151, 16c, 187, 1b5)
6.8e+06 top.master_d : mem[78:87] = (151, 16c, 187, 1b5)
6.9e+06 top.master_d : mem[78:87] = (15c, 178, 194, 1c3)
7e+06 top.master_d : mem[78:87] = (15c, 178, 194, 1c3)
7.1e+06 top.master_d : mem[78:87] = (15c, 178, 194, 1c3)
7.2e+06 top.master_d : mem[78:87] = (16c, 189, 194, 1c3)
7.3e+06 top.master_d : mem[78:87] = (177, 195, 1a1, 1e4)
7.4e+06 top.master_d : mem[78:87] = (177, 195, 1a1, 1e4)
7.5e+06 top.master_d : mem[78:87] = (177, 195, 1a1, 1e4)
7.6e+06 top.master_d : mem[78:87] = (182, 1a1, 1ae, 1f2)
7.7e+06 top.master_d : mem[78:87] = (182, 1a1, 1ae, 1f2)
7.8e+06 top.master_d : mem[78:87] = (182, 1a1, 1ae, 1f2)
7.9e+06 top.master_d : mem[78:87] = (182, 1a1, 1ae, 1f2)
8e+06 top.master_d : mem[78:87] = (18d, 1ad, 1cd, 213)
8.1e+06 top.master_d : mem[78:87] = (18d, 1ad, 1cd, 213)
8.2e+06 top.master_d : mem[78:87] = (18d, 1ad, 1cd, 213)
8.3e+06 top.master_d : mem[78:87] = (18d, 1ad, 1cd, 213)
8.4e+06 top.master_d : mem[78:87] = (198, 1b9, 1da, 221)
8.5e+06 top.master_d : mem[78:87] = (198, 1b9, 1da, 221)
8.6e+06 top.master_d : mem[78:87] = (198, 1b9, 1da, 221)
8.7e+06 top.master_d : mem[78:87] = (1b3, 1c5, 1e7, 22f)
8.8e+06 top.master_d : mem[78:87] = (1b3, 1d6, 1f9, 242)
8.9e+06 top.master_d : mem[78:87] = (1b3, 1d6, 1f9, 242)
9e+06 top.master_d : mem[78:87] = (1b3, 1d6, 1f9, 242)
9.1e+06 top.master_d : mem[78:87] = (1be, 1e2, 206, 250)
9.2e+06 top.master_d : mem[78:87] = (1be, 1e2, 206, 250)
9.3e+06 top.master_d : mem[78:87] = (1be, 1e2, 206, 250)
9.4e+06 top.master_d : mem[78:87] = (1c9, 1ee, 213, 25e)
9.5e+06 top.master_d : mem[78:87] = (1d9, 1ff, 225, 25e)
9.6e+06 top.master_d : mem[78:87] = (1d9, 1ff, 225, 271)
9.7e+06 top.master_d : mem[78:87] = (1d9, 1ff, 225, 271)
9.8e+06 top.master_d : mem[78:87] = (1e4, 20b, 232, 27f)
9.9e+06 top.master_d : mem[78:87] = (1e4, 20b, 232, 27f)
davito@davito:~/Documents/Verificacion/systemc-2.3.0/exam
ples/sysc/simple_bus$
```

Fig. 4: Salida de la simulación del ejemplo sin ninguna modificación.

Como se puede observar en la Fig. 4 el master direct tiene siempre ocupado el Bus y no se lo sede a ningún otro master por la prioridad que tiene, su prioridad es siempre respondida sin importar quien tenga ocupado el Bus.
Pero para poder observar quien tenia ocupado el bus y como se encontraban las políticas de arbitraje fue necesario modificar las las siguientes lineas con el siguiente código:

Listing 1: Código modificado.

```
78 bus = new simple_bus("bus",true);
79 arbiter = new simple_bus_arbiter
  ("arbiter", false);
```

Se simulo nuevamente con esas instrucciones habilitadas y se obtuvo o siguiente:

```
File  Edit  View  Search  Terminal  Help
9.9065e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9065e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.927e+06 top.bus : read(3) @ 48
9.9275e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9275e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.928e+06 top.bus : write(3) @ 48
9.9285e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9285e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.949e+06 top.bus : read(3) @ 4c
9.9495e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9495e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.95e+06 top.bus : write(3) @ 4c
9.9505e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9505e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.971e+06 top.bus : read(3) @ 50
9.9715e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9715e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.972e+06 top.bus : write(3) @ 50
9.9725e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9725e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.993e+06 top.bus : read(3) @ 54
9.9935e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9935e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
9.994e+06 top.bus : write(3) @ 54
9.9945e+06 top.bus : request (3) [SIMPLE_BUS_REQUEST]
9.9945e+06 top.bus Handle Slave(3)
  --> status=(SIMPLE_BUS_OK)
davito@davito:~/Documents/Verificacion/systemc-2.3.0/exam
ples/sysc/simple_bus$
```

Fig. 5: Salida de la simulación del ejemplo con al modificación del Listing 1.

B. Sistema de bus simple con dos Master Blocking, dos Master not Blocking y un Slave slow memory.

Para generar este sistema es necesario modificar el código de la libreria ***simple_bus_text.h***, y debe quedar de la siguiente manera:

Listing 2: Código modificado en la definición de las librerías.

```
41#include "simple_bus_master_blocking.h"
42#include
  "simple_bus_master_non_blocking.h"
43#include "simple_bus_slow_mem.h"
44#include "simple_bus.h"
45#include "simple_bus_arbiter.h"
46#include "simple_bus_test.h"
```

Listing 3: Código modificado para la definición de las instancias.

```
50SC_MODULE(simple_bus_test)
51{
52  // channels
53  sc_clock C1;
54
55  // module instances
56  simple_bus_master_blocking
  *master_b;
57  simple_bus_master_non_blocking
  *master_nb;
```

```
58 simple_bus_master_blocking
  *master_b_2;
59 simple_bus_master_non_blocking
  *master_nb_2;
60 simple_bus_slow_mem 6*mem_slow;
61 simple_bus *bus;
62 simple_bus_arbiter *arbiter;
```

En el Listing 3 se puede observar que ya se han declarado los masters solicitados en la entre las líneas 58 y 59.

Listing 4: Código modificado para la declaración del constructor.

```
66 // constructor
67 SC_CTOR(simple_bus_test)
68  : C1("C1")
69 {
70  // create instances
71  master_b = new
  simple_bus_master_blocking
  ("master_b", 4, 0x4c, false, 10000);
72  master_nb = new
  simple_bus_master_non_blocking
  ("master_nb", 3, 0x38, false, 10000);
73  master_b_2 = new
  simple_bus_master_blocking
  ("master_b_2", 1, 0x24, false, 10000);
74  master_nb_2 = new
  simple_bus_master_non_blocking
  ("master_nb_2", 2, 0x10, false, 10000);
75  mem_slow = new
  simple_bus_slow_mem
  ("mem_slow", 0x00, 0xff, 1);
76  bus = new simple_bus("bus",true);
  // verbose output
77  arbiter = new
  simple_bus_arbiter("arbiter", false);
```

Listing 5: Código modificado para la declaración del clock en el constructor.

```
83 // connect instances
84 bus->clock(C1);
85 master_b->clock(C1);
86 master_nb->clock(C1);
87 mem_slow->clock(C1);
88 master_b_2->clock(C1);
89 master_nb_2->clock(C1);
90 master_b->bus_port(*bus);
91 master_nb->bus_port(*bus);
92 master_b_2->bus_port(*bus);
93 master_nb_2->bus_port(*bus);
94 bus->arbiter_port(*arbiter);
95 bus->slave_port(*mem_slow);
```

Listing 6: Código modificado para la declaración del destructor.

```
101 // destructor
102 ~simple_bus_test()
103 {
104  if (master_b)
  {delete master_b; master_b = 0;}
105  if (master_nb)
  {delete master_nb; master_nb = 0;}
106  if (master_b_2)
  {delete master_b; master_b_2 = 0;}
107  if (master_nb_2)
  {delete master_nb; master_nb_2 = 0;}
108  if (mem_slow) {delete mem_slow; mem_slow = 0;}
109  if (bus) {delete bus; bus = 0;}
110  if (arbiter) {delete arbiter; arbiter = 0;}
```

Al realizar los cambios anteriores se simulo nuevamente y se obtuvo el siguiente resultado (Fig. 6)

```
File  Edit  View  Search  Terminal  Help
140500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_WAIT)
141500 SLV [116]
141500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_OK)
142500 top.bus : request (4) [SIMPLE_BUS_WAIT]
142500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_WAIT)
143500 SLV [120]
143500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_OK)
144500 top.bus : request (4) [SIMPLE_BUS_WAIT]
144500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_WAIT)
145500 SLV [124]
145500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_OK)
146500 top.bus : request (4) [SIMPLE_BUS_WAIT]
146500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_WAIT)
147500 SLV [128]
147500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_OK)
148500 top.bus : request (4) [SIMPLE_BUS_WAIT]
148500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_WAIT)
149500 SLV [132]
149500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_OK)
150500 top.bus : request (4) [SIMPLE_BUS_WAIT]
150500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_WAIT)
151500 SLV [136]
151500 top.bus Handle Slave(4)
  --> status=(SIMPLE_BUS_OK)
davito@davito:~/Documents/Verificacion/Laboratorios/Labor
atorio_4/Prueba$
```

Fig. 6: Salida de la simulación al modificar el Listing 6.

IV. Pruebas extras

En Listing 4 se observar creación de los cuatro masters, dos blocking y dos non-blocking, también se puede ver el grado de prioridad que tiene cada uno de estos masters la cual corresponde al primer número que aparece después de nombrarlo, entre menor sea este número, mayor será su prioridad.
El último número, que corresponde a 10000 indica cada cuanto se va a hacer una request, este número se compara con el tiempo total de la simulación el cual se define en el archivo *simple_bus_main.cpp* el cual se puede observar en el Listing 7

Listing 7: Contenido del archivo simple_bus_main.cpp.

```
36#include "systemc.h"
37#include "simple_bus_test.h"
38
39int sc_main(int, char **)
40{
41  simple_bus_test top("top");
42
43  sc_start(10000, SC_NS);
44
45   return 0;
46}
```

En la línea 43 en el comando ***sc_start()*** se define el tiempo de simulación y las unidades de tiempo usadas, en este caso serán nanosegundos.
Al utilizar en la consola el comando *./simple_bus | grep read* se obtuvo lo siguiente (ver Fig. 7):

```
0 top.bus : burst_read(4) @ 4c
0 top.bus : read(3) @ 38
0 top.bus : burst_read(1) @ 24
0 top.bus : read(2) @ 10
```

Fig. 7: Resultado al ejecutar el comando *./simple_bus | grep read*.

El orden en el que aparecen estas instrucciones corresponde al orden en el que se ejecutaron las instrucciones read, se puede ver que se tomaron en el orden en que estaban inicialmente escritas en el programa sin tener en cuenta su precedencia.
Después de esto se ejecutó la siguiente instrucción *./simple_bus | grep write* y se obtuvo lo siguiente en la consola (ver Fig 8).

```
34000 top.bus : write(2) @ 10
38000 top.bus : write(3) @ 38
48000 top.bus : burst_write(1) @ 24
120000 top.bus : burst_write(4) @ 4c
```

Fig. 8: Resultado al ejecutar el comando *./simple_bus | grep write*.

En la imagen anterior se puede ver que el orden en el que se ejecutan las instrucciones ha cambiado, y no solo de acuerdo a su precedencia sino al tipo de master con el que se trabaja. Los primeros dos que se ejecutan son los non-blocking masters, ésto a pesar de que hay otro master con una prioridad mayor, después de estos se ejecutan los blocking masters de acuerdo a su orden de prioridad. Se puede concluir entonces que los non-blocking masters tienen prioridad sobre los blocking, incluso si su número de prioridad es mayor.
Posteriormente se intentó usar la misma prioridad para cada uno de los masters pero al compilarlo se obtuvo un error, por lo que se puede concluir que cada master debe tener un número de prioridad única.
Después de esto se volvió a habilitar el arbiter y a deshabilitar el bus para ver que resultado se obtenía. En la consola se obtuvo lo siguiente (ver Fig. 9):

```
20500 top.arbiter :
    R[4](-SIMPLE_BUS_REQUEST@4c)
    R[3](-SIMPLE_BUS_REQUEST@38)
    R[1](-SIMPLE_BUS_WAIT@4c)
    R[2](-SIMPLE_BUS_REQUEST@10) -> R[1] (rule 3)
22500 top.arbiter :
    R[4](-SIMPLE_BUS_REQUEST@4c)
    R[3](-SIMPLE_BUS_REQUEST@38)
    R[1](-SIMPLE_BUS_WAIT@50)
    R[2](-SIMPLE_BUS_REQUEST@10) -> R[1] (rule 3)
24500 top.arbiter :
    R[4](-SIMPLE_BUS_REQUEST@4c)
    R[3](-SIMPLE_BUS_REQUEST@38)
    R[1](-SIMPLE_BUS_WAIT@54)
    R[2](-SIMPLE_BUS_REQUEST@10) -> R[1] (rule 3)
26500 top.arbiter :
    R[4](-SIMPLE_BUS_REQUEST@4c)
    R[3](-SIMPLE_BUS_REQUEST@38)
    R[1](-SIMPLE_BUS_WAIT@58)
    R[2](-SIMPLE_BUS_REQUEST@10) -> R[1] (rule 3)
```

Fig. 9: Habilitación del arbiter y deshabilitación del bus.

En la Fig. 9 se puede ver una parte del resultado que se obtuvo. Aquí se puede ver como cada uno de los masters está pidiendo permiso para usar el bus. En la parte derecha se puede ver a cual de los masters se le da prioridad y de acuerdo a cual norma de arbitraje. Existen 3 normas de arbitraje, las cuales son:

1) Si la solicitud corresponde a una "*locked burst request*" entonces a esta siempre se le dará prioridad.
2) Si la última solicitud tiene su bandera "*lock*" activa, y tiene una nueva solicitud entonces se le dará prioridad a esta.
3) La solicitud con la mayor prioridad (menor número) es seleccionada de la cola y tomada como prioridad.

Ya que no se está haciendo uso del comando "*lock*" entonces la prioridad será regida por la norma 3.

V. Conclusiones

- Cuando un master non-blocking y un blocking están solicitando el bus al mismo tiempo, el master non-blocking va a tener prioridad sobre el otro, no importa el número de prioridad que tenga el master blocking.
- El método de arbitraje usado por el bus simple original es el de prioridad fija, cada solicitud por parte de los master es leída en el orden de llegada, sin embargo se le da prioridad a aquel master con un número de prioridad menor. Esto siempre y cuando otro master no tenga la bandera "lock" activada o no sea un direct master, el cual tiene una prioridad directa y no tiene que esperar.
- En casos donde se requiera que una solicitud por parte del master para que sea atendida lo más rápido posible se podría usar la interface de bus directa, esta interface siempre va a tener la mayor prioridad, evitando así cualquier protocolo de arbitraje y la latencia que esto produce.
- Al umentar el número de maestros se ven afectadas las métricas porque todos los maestros comparten los mismos recursos del bus y dependiendo de las politicas de arbitraje se vera afectada la latencia.

REFERENCIAS

[1] Patterson, David & Hennessy John *"'Computer Organization And Design - The Hardware-Software Interface"'*. Kindle Edition, Fourth Edition, 2006.

[2] Wikipedia. Sitio Web:http://en.wikipedia.org/wiki/Bus_%28computing%29 . Visitado el 24 de enero de 2014.