Taller de Buses

Organización del Computador 1

Primer Cuatrimestre 2018 - Turno Mañana

Introducción

Un bus está compuesto por cables o líneas que se pueden dividir según su propósito: para datos, para direcciones o de control.

En este taller simularemos el comportamiento de un bus. Cada línea del bus estará representada por un archivo. Estos archivos contendrán un solo número entero: 0, 1 ó -1, identificando el estado de dicha línea. El valor -1 representa el estado de alta impedancia, en el cual el dispositivo se desconecta de la línea.

Los dispositivos del sistema que interactúan por medio del bus estarán simulados por programas que leen y escriben estos archivos según corresponda al protocolo.

Para no complicar en exceso la simulación a realizar, se simularán protocolos sincrónicos. Este tipo de buses utiliza una línea especial llamada "reloj" (`clk`) que varía entre 0 y 1 regularmente.

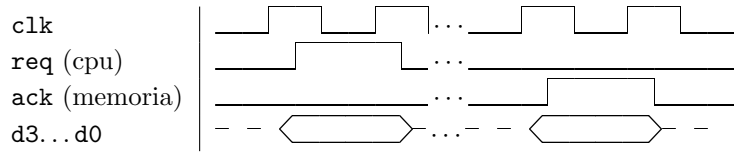
El comportamiento de los dispositivos se divide en dos etapas identificadas por el estado del reloj. Durante el estado 1 (⎍), los dispositivos pueden escribir en las señales que les correspondan pero no pueden leerlas, ya que las líneas no están estables. Cuando el reloj pasa a valer 0 (estado ⎍) los dispositivos pueden leer las líneas, pero no pueden escribirlas.

Ejemplo

En este ejemplo se muestra un ciclo de lectura entre un *cpu* y la memoria de sólo lectura. El bus esta compuesto por siete señales: `req`, `ack`, `d0`, `d1`, `d2`, `d3` y la señal de sincronía `clk` (los dispositivos sólo pueden leerla).

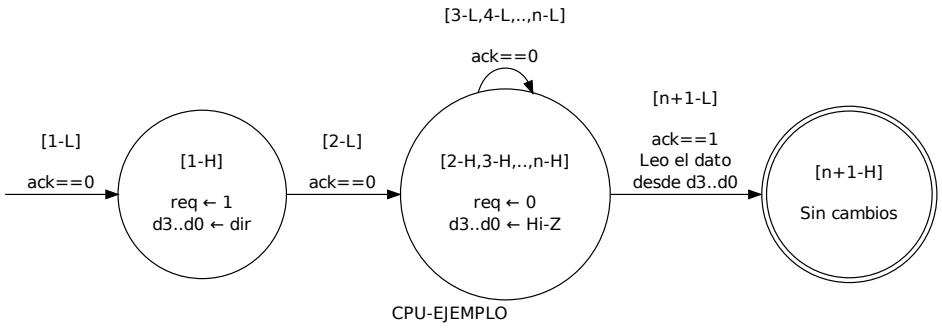
La señal `req` es comandada por el cpu, mientras que la señal `ack` es comandada por la memoria. El resto de las señales se comparten entre los dos dispositivos.

Anotando entre paréntesis la entidad con autoridad sobre cada línea, el protocolo del lectura de este bus consiste en:



- El cpu coloca en las señales `d3...d0` la dirección del dato y levanta la señal `req`.
- La memoria detecta que la señal `req` está levantada y lee `d3...d0`.
- El cpu libera las señales `d3...d0` y baja la señal `req`.
- La memoria coloca el dato de la dirección pedida y levanta la señal `ack`.
- El cpu lee el dato.
- La memoria finaliza el ciclo liberando las señales `d3...d0` y bajando su señal `ack`.

La máquina de estados que describe el protocolo implementado por el CPU es:



Este protocolo está implentado en los siguientes archivos:

- `reloj.cpp`: Programa que escribe `clk` regularmente.
- `memoria.cpp`: Programa que implementa el comportamiento de la *memoria*.
- `cpu.cpp`: Programa que implementa el comportamiento del *cpu*.
- `print.cpp`: Programa que permite imprimir el bus para los semiciclos bajos (semiciclo de lectura).
- `bus.h`: Definición de funciones útiles.
- `bus.cpp`: Implementación de funciones útiles.
- `Makefile`: Archivo para la generación de los ejecutables.

Las funciones implementadas en `bus.cpp` son:

- `int read(string linea)`: Dado el nombre de una línea (el nombre del archivo), lee su contenido.
- `void write(string linea, int x)`: Escribe el valor `x` en la línea de nombre `linea` (para Hi-Z se escribe -1).

Para poder probar el ejemplo se deben compilar los programas (comando `make`). Luego deben ser ejecutados siguiendo el siguiente orden: `reloj`, `print`, `memoria` y por último `cpu`. El programa `cpu` solicitará los 4 *bits* de la dirección a ser leída y procesará ese pedido.

Para visualizar el bus, ejecutar: `python visual.py` (sin parámetros). En forma alternativa, se pueden lanzar todos los programas automáticamente con el visualizador gráfico, ejecutando únicamente: `python visual.py a a a a`, donde `a` corresponde a un *bit* de la dirección que el CPU solicita a la memoria. Ejemplo: `python visual.py 1 0 0 1`

Ejercicio 1

Ingresar a la carpeta `ej1` y compilar el código. Luego, abrir tres consolas y en cada una de ellas ejecutar en orden los programas `reloj`, `print` y `KITT`. El visualizador `visual.py` en este caso se ejecuta sin parámetros. Si se desea que los programas se corran automáticamente se debe ejecutar únicamente: `python visual.py 0`.

Se pide:

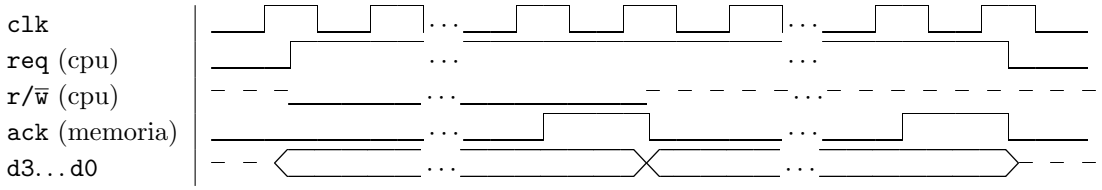
a) Realizar el diagrama de tiempos para todas las señales

b) Modificar el programa `KITT` para que realice la siguiente secuencia:

Paso 1	0	0	0
Paso 2	0	1	0
Paso 3	1	0	1
Paso 4	0	1	0

Ejercicio 2

Completar el esqueleto de *cpu* provisto para simular el comportamiento de un *ciclo de escritura* para un bus sincrónico con las señales `req`, `ack`, `r/w`, `d0`, `d1`, `d2` y `d3`:



- El cpu baja la señal `r/w`, coloca en las señales `d3...d0` la dirección en donde desea escribir y levanta la señal `req`.
- La memoria lee la señal `req` activa y la dirección en `d3...d0`.
- El cpu mantiene sus señales a la espera de una señal de la memoria. La cantidad de ciclos que tarda en aparecer no es conocida por el cpu.
- La memoria levanta la señal `ack` durante un ciclo, indicándole al cpu que ya puede colocar el dato que desea guardar en `d3...d0`.
- El cpu libera la señal `r/w` y coloca en `d3...d0` el dato que desea almacenar.
- La memoria lee el dato colocado por el cpu y procede a almacenarlo en la dirección leída previamente.
- El cpu mantiene sus señales a la espera de una señal de la memoria. La cantidad de ciclos que tarda en aparecer no es conocida por el cpu.
- La memoria levanta nuevamente la señal `ack` durante un ciclo de reloj indicándole al cpu que ya almacenó el dato y que puede finalizar el ciclo de escritura.
- El cpu baja todas sus señales y libera las señales `d3...d0`. La memoria finaliza el ciclo de escritura bajando su señal `ack`.

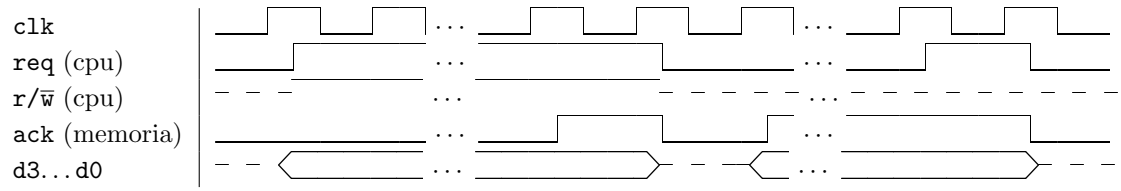
Se pide:

- Realizar la máquina de estados del CPU que describa el protocolo.
- Agregar el código necesario dentro del archivo `cpu.cpp` en la carpeta `ej2` para simular el comportamiento de una CPU compatible con el protocolo descripto.

Si se utiliza el visualizador `visual.py` se debe ejecutar de la siguiente forma: `python visual.py a a a a v v v v`, donde `a` indica un *bit* de dirección y `v` un *bit* del valor en esa dirección. Si se ejecuta sin parámetros se asume que los procesos se lanzarán manualmente.

Ejercicio 3

Utilizando de base lo desarrollado hasta ahora, construir una simulación del comportamiento de un *ciclo de lectura* para un bus sincrónico con las señales `req`, `ack`, `r/w`, `d0`, `d1`, `d2` y `d3`. El comportamiento se describe por el siguiente diagrama de tiempos:



- El cpu levanta la señal `r/w`, coloca en las señales `d3...d0` la dirección que desea leer y levanta la señal `req`.
- La memoria lee la señal `req` activa, la dirección en `d3...d0` y comienza la búsqueda del dato requerido por el cpu.
- El cpu mantiene sus señales a la espera de una señal de la memoria. La cantidad de ciclos que tarda en aparecer no es conocida por el cpu.

- La memoria levanta la señal `ack` durante un ciclo de reloj indicándole al cpu que ya puede liberar las líneas `d3...d0` para que la memoria escriba los datos.
- El cpu libera la señal `r/w`, las líneas `d3...d0` y baja la línea de `req`.
- La memoria espera un ciclo de reloj a que el cpu libere las líneas `d3...d0` donde escribirá el dato.
- La memoria escribe en las líneas `d3...d0` el dato requerido por el cpu y levanta su señal `ack`.
- La memoria mantiene sus señales a la espera de una señal del cpu. La cantidad de ciclos que tarda en aparecer no es conocida por la memoria.
- El cpu levanta la señal `req` durante un ciclo de reloj indicándole a la memoria que ya leyó el dato y que puede finalizar el ciclo de lectura.
- La memoria baja todas sus señales y libera las señales `d3...d0`.

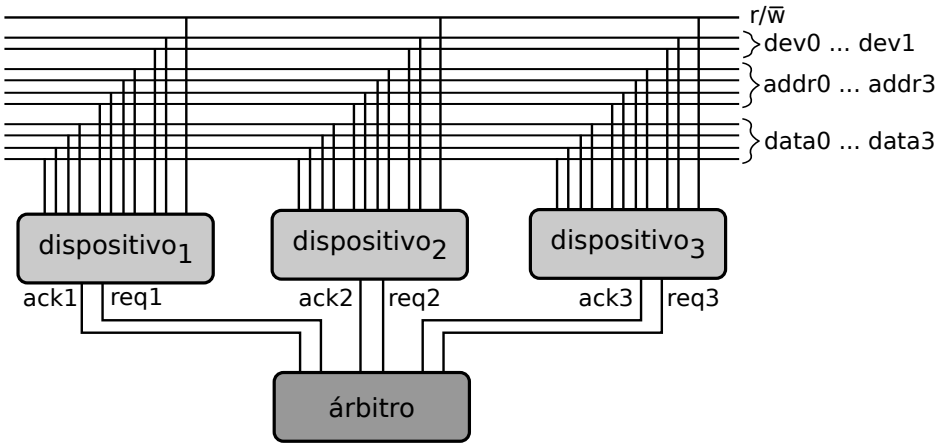
Se pide:

- Realizar la máquina de estados del CPU que describa el protocolo.
- Completar el programa dado de forma que permita tomar una dirección y procesar un ciclo de lectura. Para esto agregue el código necesario dentro del archivo `cpu.cpp` en la carpeta `ej3`.

El visualizador en este ejercicio debe invocarse como: `python visual.py a a a a`.

Ejercicio 4 (optativo)

Se desea simular un nuevo sistema diferente al de los dos ejercicios anteriores. En este caso se poseen tres dispositivos, donde cada uno puede leer o escribir en cualquiera de los otros dos. El dispositivo que realice la lectura o escritura se llamará *maestro* y el que responda al pedido se llamará *esclavo*. El sistema requiere simular dos protocolos independientes, uno para adquirir el bus y otro para utilizarlo.

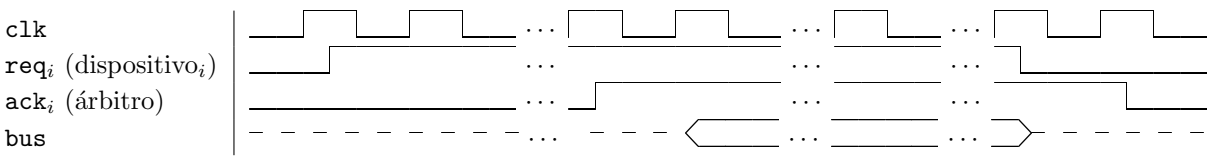


Adquisición del bus

Además de los tres dispositivos, el sistema posee un "árbitro" que permite organizar las solicitudes del bus. Cada dispositivo posee dos señales que lo conectan con el bus, `req` y `ack`. El dispositivo i puede escribir en $\texttt{req}_i$ y leer de $\texttt{ack}_i$. Mientras que el árbitro puede leer $\texttt{req}_i$ y escribir $\texttt{ack}_i$.

Para adquirir el bus, el dispositivo debe encender su señal $\texttt{req}_i$ sincrónicamente y esperar una respuesta del árbitro. El árbitro en algún ciclo de reloj posterior, levantará la señal $\texttt{ack}_i$ correspondiente al dispositivo que solicitó el bus.

Al ciclo siguiente el dispositivo puede hacer uso del bus. Cuando el dispositivo desea liberar el bus, debe bajar sincrónicamente su señal $\texttt{req}_i$ y al ciclo siguiente el árbitro bajará la señal $\texttt{ack}_i$ indicando la liberación del bus.



1 – El dispositivo i levanta la señal $\texttt{req}_i$.

2 – El dispositivo i espera que el árbitro levante la señal $\texttt{ack}_i$ para utilizar el bus, lo cual puede tardar una cantidad variable de ciclos en ocurrir.

3 – El árbitro levanta la señal $\texttt{ack}_i$ otorgando el bus al dispositivo i.

4 – El dispositivo i puede utilizar el bus libremente durante los ciclos que desee.

5 – Uso del bus por parte del dispositivo i.

6 – El dispositivo i libera las señales y baja su señal $\texttt{req}_i$.

7 – El árbitro del bus baja la señal $\texttt{ack}_i$ liberando el bus para un nuevo pedido.

Protocolo de bus

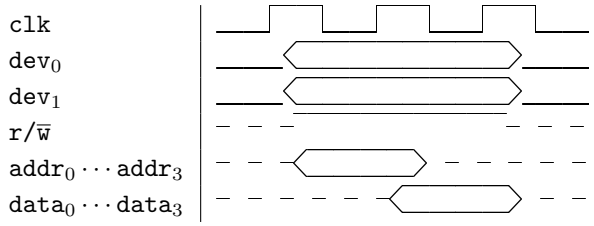
El bus cuenta con las siguientes señales, sin enumerar el reloj (`clk`):

- $\texttt{dev}_0$, $\texttt{dev}_1$: Indica el dispositivo con el cual interactuar.
- $\texttt{addr}_0 \cdots \texttt{addr}_3$: Indica la dirección donde leer o escribir.
- $\texttt{data}_0 \cdots \texttt{data}_3$: Indica el dato enviado o recibido por los dispositivos.
- $\texttt{r/}\overline{\texttt{w}}$: Indica si se trata de un ciclo de lectura o escritura.

Las señales $\texttt{dev}_0$ y $\texttt{dev}_1$ indican el dispositivo con el cual interactuar dependiendo de la siguiente tabla:

$\texttt{dev}_1$	$\texttt{dev}_0$	
0	0	ninguno
0	1	dispositivo 1
1	0	dispositivo 2
1	1	dispositivo 3

Ciclo de lectura:

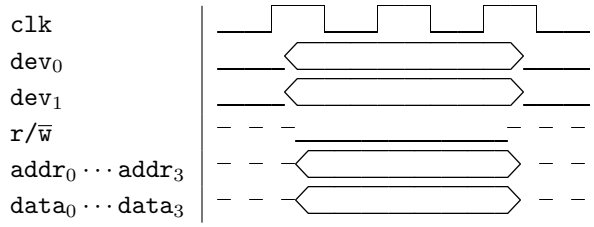


1 – El dispositivo maestro setea las señales $\texttt{dev}_0$ y $\texttt{dev}_1$ indicando el dispositivo donde leer, levanta la señal $\texttt{r/}\overline{\texttt{w}}$ y setea las señales de $\texttt{addr}_0 \cdots \texttt{addr}_3$ indicando la dirección a leer durante un ciclo de reloj. El dispositivo esclavo reconoce que debe atender el pedido y lee la dirección del bus para comenzar el ciclo de lectura.

2 – El dispositivo esclavo setea las señales $\texttt{data}_0 \cdots \texttt{data}_3$ con el dato requerido durante un ciclo de reloj.

3 – El dispositivo maestro baja sus líneas y setea $\texttt{dev}_0$ y $\texttt{dev}_1$ a dispositivo `ninguno`.

Ciclo de escritura:



1-2 – El dispositivo maestro setea las señales $\texttt{dev}_0$ y $\texttt{dev}_1$ indicando el dispositivo en el cual escribir, baja la señal $\texttt{r/}\overline{\texttt{w}}$, setea las señales de $\texttt{addr}_0 \cdots \texttt{addr}_3$ indicando la dirección a donde escribir y coloca el dato en las señales $\texttt{data}_0 \cdots \texttt{data}_3$. Todo esto durante dos ciclos de reloj. El dispositivo esclavo reconoce que debe atender el pedido y comienza la escritura del dato en la dirección indicada.

3 – El dispositivo maestro baja sus líneas y setea $\texttt{dev}_0$ y $\texttt{dev}_1$ a dispositivo `ninguno`.

Se pide:

- Completar el programa dado de forma que permita:
  a) Responder a un ciclo de lectura
  b) Responder a un ciclo de escritura
  c) Generar un ciclo de lectura
  d) Generar un ciclo de escritura

  Para esto agregue el código necesario dentro del archivo `dispositivo.cpp` en la carpeta `ej4`.

A. Comandos útiles para usar linux

A continuación se listan comandos necesarios para la ejecución del taller.

`cd nombreDirectorio` Cambia del directorio actual al subdirectorio llamado nombreDirectorio

`cd ..` cambia del directorio actual al directorio inmediatamente superior

`pwd` imprime en pantalla el directorio actual

`ls` lista el contenido del directorio actual

`make` ejecuta las instrucciones especificadas adentro del archivo Makefile

`python visual.py` ejecuta el visualizador unicamente en modo visualización, se deben lanzar los procesos de manera independiente

`./ejecutable` lanza el programa compilado en el archivo ejecutable. Por ejemplo, para el programa del reloj hacemos `./reloj`

Control + c para terminar la ejecución de un programa.