Comp3211 Solution Week5

Usama Malik

August 25, 2003

1 Question 1

QUESTION: **[Y5.2] Consider a VHDL type that can take on the values (0, 1, X, U). The values X and U correspond to the values unknown and uninitialised respectively. Define a resolved type that takes on these values and write and test a resolution function for this resolved type.**

SOLUTION

Recall: A resolved type is a signal that may have multiple drivers. The algorithm for resolving the issue of the signal value at any time is captured in the *resolution function*. The signal must be declared as a *resolved type* meaning that there is a resolution function associated with it. This function should accurately reflect the bahavior of the physical system being modelled (Y:pp–131..132). The following is an example declaration of resolved signals.

```
type std_complogic is('0', –Forcing 0
                      '1' –Forcing 1
                      'X' –Forcing Unknown
                      'U' –Forcing Uninitialised
                      );
function resolved (s: std_complogic_vector) return std_complogic;
subtype std_logic is resolved std_complogic;
```

The following is an example resolution function.

	0	1	X	U
0	0	X	X	U
1	X	1	X	U
X	X	X	X	U
U	U	U	U	U

In order to test the above function, we can declare a signal of type *std_complogic* in multiple processes and assert them concurrently. The above resolution function can be implemented using the *function* statement in VHDL.

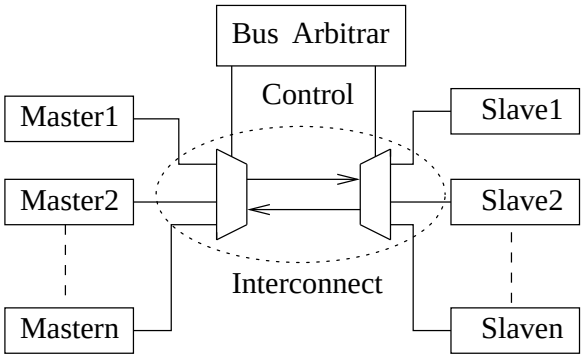

Figure 1: A simplified view of a bus system.

2 Question 2

QUESTION: **[Y5.3] Using a concurrent procedure looks very much like using a component in a hierarchically structured design. What is the difference between using a concurrent procedure and constructing a structural design?**
SOLUTION
A procedure cannot have a state.

3 Question 3

QUESTION: **Describe using a state diagram an asynchronous handshake protocol as initiated by the bus master. Describe a suitable VHDL architecture for this side of the protocol.**
SOLUTION
Recall: Asynchronous logic is logic synchronised without a global clock. A simplified view of a bus system is shown in Figure 1. The arbitrar grants a *Bus grant* signal to a winner (assume that there is only one winner). Now, the procedure for data transfer is simple. In the first step, the Master asserts a MSYNC signal. When the slave sees this signal, it performs the work as fast as possible and at the end asserts a SSYNCH signal. When the Master sees this, it knows that the data is ready. It latches the data and then asserts a negation of MSYNC. When the slave sees this signal, it asserts a negation of SSYNCH and the cycle is complete. DISCUSSION What else do we need if the bus lines are tristated not multiplexed?

An FSM description of this is shown in Figure 2.

A possible VHDL implementation is provided below (modified from [Y: page74]).

```
library IEEE;
use IEEE.std_logic_1164.all;

entity handshake is
port(data: inout std_logic_vector(31 downto 0);
```

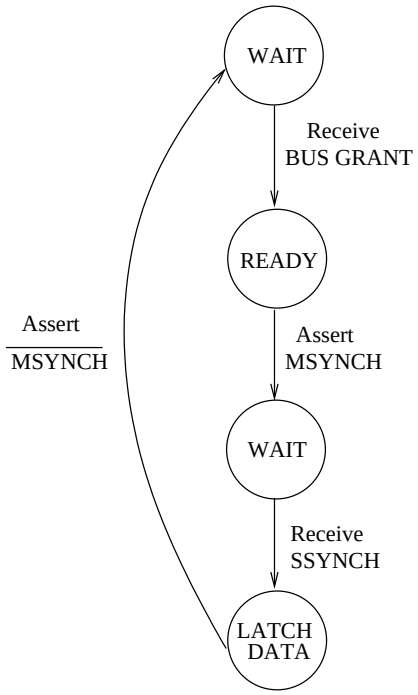

Figure 2: An FSM for an asynchronous handshake protocol.

```
  bus_grant:in std_logic;
  MSYNCH:out std_logic;
  SYNCH:in std_logic );
end handshake;
architecture behavioral of handshake is
signal transmit_data: std_logic_vector(31 downto 0);
begin
send:process
begin
wait until (bus_grant='1'); –wait for bus grant
MSYNCH <= '1';
wait until SYNCH <= '1';
data <= transmit_data; –send data
MSYNCH <= '0';
wait until SYNCH <= '0';
end process send;

recv:process
variable rec_data: std_logic_vector(31 downto 0);
begin
wait until (data'event and bus_grant='1') ; –wait for the interrupt
MSYNCH <= '1';
wait until SYNCH <= '1';
rec_data := data
MSYNCH <= '0';
wait until SYNCH <= '0';
end process recv;
end behavioral
```

4 Question 4

QUESTION: **[SR3.5] What is the benefit of using a multiple-bus architecture compared to a single bus architecture.**
SOLUTION

- Match the communication requirements of the devices. Slow devices use slow buses (Camera → USB) and vice versa. Thus, system load can be balanced (Compare: Memory hierarchy: Cache → RAM → Disk. How about a processor hierarchy? Main processor, peripheral processors such as *modem*, *VGA card*, *sound card* and so on: discuss!).
- A single bus *can* become the system bottleneck.
- Multiple buses can be used to induce *fault tolerance*.

- Manage the design complexity.

5 Question 5

QUESTION: **[S3.8] The VAX SB1 bus uses a distributed, synchronous arbitration scheme. Each SB1 device (i.e., processor, memory, I/O module) has a unique priority and is assigned a unique transfer request (TR) line. The SB1 has 16 such lines (TR0, TR1, ..., TR15), with TR0 having the highest priority. When a device wants to use the bus, it places a reservation for a future time slot by asserting its TR line during the current time slot. At the end of the current time slot, each device with a pending reservation examines the TR lines; the highest priority device with a reservation uses the next slot. A maximum of 17 devices can be attached to the bus. The device with priority 16 has no TR line. Why not?**

SOLUTION

As the last device has the lowest priority, it can only access the bus when no other device has a pending request. This means that the devices 0-16 donot need to know about the 17^{th} device and hence there is no TR line for it.

6 Question 6

QUESTION: **[S3.9] Paradoxically, the lowest priority device usually has the lowest average wait time. For this reason the processor is usually given the lowest priority on the SB1. Why does the priority 16 device usually have the lowest average wait time? Under what circumstances would this not be true?**

SOLUTION

If the devices 0-16 have made no request, they will not examine the TR lines (i.e. less delay) and the 17^{th} device will get the access (notice that the protocol is synchronous). There is a possibility of starvation for the last device. This can happen if the system is overloaded.