Computer Structure Lab - Fall 2000
Dr. Guy Even

Quiz on Nov. 9th on Resa Bus

1 Handout #1: The RESA's Backplane Bus

In this assignment you will learn about the RESA's backplane bus. We start with a general description of buses, and continue with a description of the bus protocol used in the RESA's backplane bus.

1.1 What is a bus?

In this section we describe basic notions and terminology in busses.

Suppose we have multiple devices that want to communicate with each other (e.g. a CPU board and a memory board). The simplest way to enable communication is to connect all the devices to the same wires so that each device can read and write signals to the wires. A wire which connects devices is called a *bus*. Often, several wires are connected in parallel to the same devices to enable the transmission of more than a single bit simultaneously.

Connecting the devices together by wires does not solve the problem of communication. There are a few issues which need to be addressed, such as: How do the devices call each other (i.e. naming)? How do they actually communicate (i.e. language)? How do the devices share the common resource (i.e. controlling contention)? How is the common resource shared in a fair way (i.e. fairness)? How do we make sure that a device does not hold the common resource indefinitely (i.e. deadlock prevention)?

We consider a setting in which communication takes place in chunks called *transactions*, which means that transactions are the basic "unit of communication". The analogy of a transaction is a single telephone conversation: one party calls another party; some communication takes place, and the conversation is over. A transaction is initiated by one party (i.e. a device) which either wants to send data to another party or wishes to retrieve data from another party. Note that the data transmitted during a transaction can be a bit, byte, or even several bytes (i.e. bursts). The party which initiates the transaction is called the *master* and the party which is asked to respond is called the *slave*. This means that devices are divided into two types: masters - devices that initiate transactions, and slaves - devices that respond to transactions. For example, a CPU board is a master and a memory board is a slave in bus transactions that implement load/store instructions.

We differentiate between two types of transactions: (a) a *write transaction* is a transaction in which the master wants to send a value to the slave (for example, in an execution of a store instruction, the CPU writes a value to the main memory); and (b) a *read transaction* is a transaction in which the master wants to receive a value from the slave (for example, in an execution of a load instruction, the CPU reads a value from the main memory).

The communication over the bus is determined by a *protocol*. A protocol is an algorithm, however, it differs from the algorithms you have seen so far in a few ways: (a) The protocol is executed by multiple parties as opposed to the algorithms you have seen which are executed by a single central "entity". (b) A protocol does not perform a "pure computation" like an algorithm that computes, for example, primes numbers. The purpose of a

protocol is to coordinate the usage of the common resource, namely, the bus. One can view the bus protocol as the "rules of the road".

The transactions which we consider are *address based*. This means that the bit signals sent along the bus are divided into fields, which include: an address field, a data field, and some additional control signals. The master in a transaction transmits the address of the data item to be transmitted. In a write transaction, the master also writes the data in the data field. In a read transaction, the slave writes the data in the data field. It helps to view the address field as if it is divided into two parts: (a) the slave's name so that the slave can detect that it is being accessed in the current transaction; and (b) the name of the item within the slave that is being accessed. This means that each slave has an *address space* associated with it; different slaves have different address spaces.

When a transaction begins, the master device does not know how long it will take the slave device to complete the transaction. There are two reasons for this uncertainty: (a) the delay of the slave device might not be constant; and (b) this helps simplify the design. The master doesn't need to know the delays of all the slave devices, and the master's control is not based on the delays of the slave devices. Instead, the slave device reports the completion of the transaction to the master device. The number of cycles it takes the slave device to respond is referred to as the number of *wait states*. A slave device with k wait states responds $k + 1$ clock cycles after the master requests the transfer of data.

The *bus controller* coordinates the usage of the bus. The bus controller can be regarded as the "traffic light" of the bus.

1.2 The backplane bus protocol

In this section we describe the bus protocol used by the RESA's backplane bus.

Properties

Before we describe the RESA bus signals and the protocol, we mention two properties of the RESA bus protocol:

1. The backplane bus protocol of the RESA computer is a *synchronous* protocol. This means that there is a global clock signal which is present in all the devices that are connected to the bus. Every transition of a signal in the bus is synchronized with an edge of the clock. There are transitions that are synchronized with the falling edges, and there are transitions that are synchronized with the rising edges. There is an exception to this rule, as you will notice later.

2. In each transaction, one word of data is transmitted. Other protocols exist that can support bursts of data to save the setup time spent in establishing a transaction.

Signals

A bus protocol is an algorithm, and the data structures in a bus protocol are signals. The following signals are transmitted along the bus:

Clock (`clk`): This is a single bit signal which serves as the common clock. The bus controller is the only device that may write this value. The other devices only read

it. The clock signal is usually a periodic signal which has a pulse width equal to half the period.

Bus Request ($\texttt{br}_i$): Master device i (there are up to 4 master devices) is connected to the bus controller with a bit signal called the bus request. Whenever master device i wishes to initiate a transaction, it asserts[1] the $\texttt{br}_i$ signal to signal to the bus controller that it would like to start a transaction. Only master device i may write to the $\texttt{br}_i$ signal. This signal can be viewed as a private link between master device i and the bus controller.

Bus Grant ($\texttt{bg}_i$): A second private link between master device i and the bus controller is the $\texttt{bg}_i$ signal. The $\texttt{bg}_i$ signal is asserted by the bus controller to signal that master device i may start a transaction, that is, the bus is free and master device i may use it. This is like a green light in a traffic light.

Address Strobe (as): The beginning of a transaction is signaled by a master device by asserting the as signal. This tells the slave device that participates in this transaction that, in the next clock cycle, information will be sent on the bus. This information is sent by the master and the slave should "listen" to it. Note that when the Address Strobe signal is asserted, the address has not been asserted yet, and therefore, it is not determined yet which device is the slave device in the current transaction.

Address ($\texttt{A}[31:0]$): The address signals are used to transmit the address of the data item which is transmitted during the transaction. Note that the address of a data item holds the slave's device name as well as the item's name within the slave. For this purpose, some of the address bits are used to address the slave device, the the rest are used to address the data item within the device. The address signals are always asserted by the master.

Data ($\texttt{D}[31:0]$): The data signals are used to transmit the data item which is actually being transmitted in a transaction. Recall that this is the purpose of the whole transaction! In a write transaction, the master asserts the Data signals, and in a read transaction the slave asserts the Data signals.

Write (wr): The wr signal is asserted by the master to indicate whether the transaction is a read transaction or a write transaction.

Acknowledge (ack): The slave acknowledges the transmission of the data item by asserting the ack signal. In a write transaction, the slave signals that the data has been read and processed by it. In a read transaction, the slave signals that the requested data item will be transmitted in the next clock cycle (Why not wait until the requested data is transmitted? Why does the slave warn the master that the data will be transmitted?).

Protocol

It is common to describe bus protocols using *timing diagrams*. A timing diagram depicts the behavior of the signals during a transaction. Figure 1 depicts the timing diagrams of

[1] The term "to assert" means that it make a signal active. As you will see, some signals are active when they are low and some are active when they are high.

read and write transactions. We make the following remarks about the notation used in the timing diagrams:

1. Some of the bus signals are negated for reasons that are explained in Section 1.3. The negation of signal `x` is denoted by `/x`.

2. The time it takes for a transition of a signal from one logical level to another is denoted by a box, the vertical sides of which determine the earliest and latest possible times at which it is stable. For example, the `as` signal is asserted and disabled after a rising edge of the clock. This notation is used for transitions of signals that can only hold logical levels (i.e. low or high).

3. The assertion of a signal the value of which is not uniquely determined before it is asserted is denoted a diagonal transition. This notation is used for signals that can hold logical levels as well as non-logical levels (i.e. undefined values).

4. Signals that can be either low or high during a transaction are depicted using two parallel horizontal lines, one for the low value and the other for the high value. For example, the address and data signals can be either high or low, and their value before they are asserted may be non-logical. Therefore, the transitions are depicted using diagonal signals, and the asserted cycles are depicted using both high level and low level signals. Note that the address and data buses each have 32 bits. However, in the timing diagram we depict only a single bit, since all the bits have the same behavior.

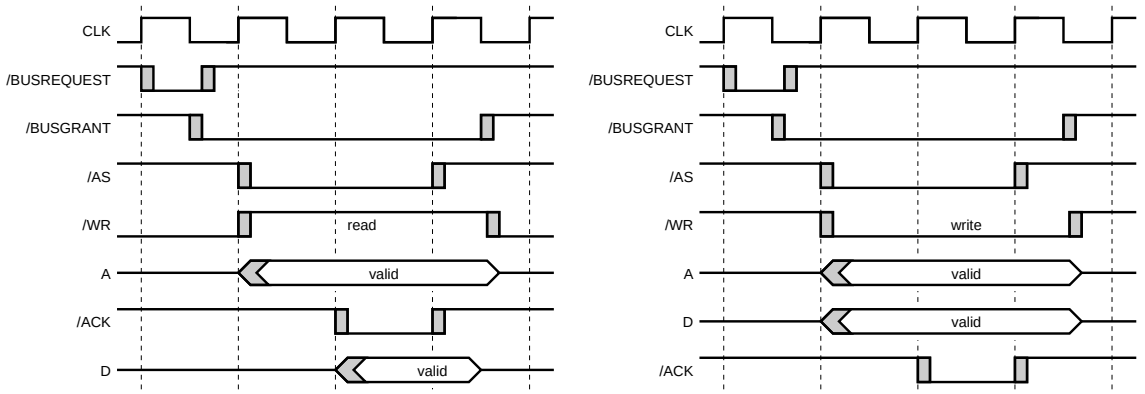

Figure 1: The backplane bus protocol (shaded areas denote periods of uncertainty)

1.3 Wired-Or

In a bus there are wires that multiple devices can write to. For example, the data signals can be written by every device (master or slave). We now describe the electrical implementation which enables multiple devices to write to the same wire.

There are two situations depending on the desired signal value when no device writes to a shared wire.

1. The signal value is undefined when no device writes to the shared wire, namely, we do not care what its value is (it may be logical or even non-logical). In this case, the devices with a write permission are connected to the shared wire via drivers, and each device generates a local driver enable signal. The bus protocol guarantees that, at every given point of time, at most one driver enable signal is active. If all the driver enable signals are inactive, the signal value on the wire need not be even logical. For example, the values of the address and data signals are relevant only if there is a device that is writing them, otherwise, their value is undefined.

2. If the signal value is important even when no device writes to the shared wire, then a special mechanism is used for enforcing a default signal value. The mechanism used for this purpose is called *open collector* or *wired OR*. Figure 2 depicts a wired NOR mechanism (we will return to the issue of NOR vs. OR shortly). The devices are connected to the shared wire via drivers, and the wire is connected via a resistor to VCC (the power source). The resistance of the resistor is much higher than that of an enabled driver, so when `/OE`=1, the voltage of the shared wire is low. We consider 3 cases: (a) no device wants to write a value. In this case, all the driver enable signals are disabled, and the wire is high due to current flowing through the resistor. (b) at least one device wants to write a 0 to the wire. In this case, the driver enable signal is activated, and the driver pulls the wire down since the driver input is GND (ground). (Ground wins.) (c) one device wants to write a 1 to the wire. This case is reduced to case (a) by disabling all the driver enable signals.

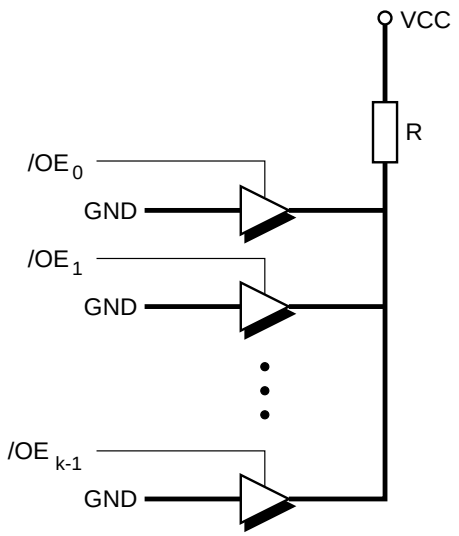

Figure 2: A wired NOR (i.e. $NOR(/\texttt{OE}_0, \ldots, /\texttt{OE}_{k-1})$)

This implementation of wired-NOR has the property that if there is a device that wants to write a 0 and another device that wants to write a 1, then the 0 wins. Such a conflict does not create any electrical problems, but the RESA's bus protocol does not allow such situations.

The bus signals that require a wired OR mechanism are the address strobe (`as`), the write (`wr`), bus request (`br`), bus grant (`bg`), and the acknowledge (`ack`). The pull-

up resistor of the wired OR signals is placed on the bus controller card. (Note that the bus request and bus grant signals are only written by a single device, and therefore, do not require a wired OR mechanism. The RESA designers chose to use wired-OR to deal with a situation where two master devices are connected by mistake to the same `br` signal. A conflict in a wired-OR mechanism does not cause electrical problems, whereas two drivers that "fight" each other can cause damages.)

Note that the mechanism proposed in Figure 2 computes the NOR of the driver enable signals $/\mathtt{OE}_i$. This is equivalent to computing $AND(\mathtt{OE}_0, \ldots, \mathtt{OE}_{k-1})$. This is why *active low* signals are used rather than *active high*. Namely, instead of using the signals `as`,`wr`,`br`,`bg`,`ack`, we use the negated signals `/as`,`/wr`,`/br`,`/bg`,`/ack`.

Supplementary Bibliography

W. Stallings, *Computer Organization and Architecture*, Prentice Hall International Inc., Chapter 3.4.

J. M. Feldman and C. T. Retter, *Computer Architecture*, McGraw Hill International, Chapter 8.3.

2 Homework (submit in the beginning of 3rd lab meeting)

Consider a CPU that wants to communicate over the RESA bus as a master device. The CPU is connected to the RESA bus via a simple bus interface. The simple bus interface is placed on the CPU board between the CPU and to the RESA bus. Communication between the CPU and the bus interface is implemented by 3 registers and 3 control signals. The functionality of the registers is as follows:

$R0$**:** a data-in register through which data is fed to the CPU.

$R1$**:** a data-out register through which data is sent from the CPU.

$R2$**:** an address register that is written by the CPU.

The control signals are as follows:

`rd_req`**:** a signal sent by the CPU to the bus interface. This signal indicates that the CPU wishes to initiate a read transaction.

`wr_req`**:** a signal sent by the CPU to the bus interface. This signal indicates that the CPU wishes to initiate a write transaction.

`done`**:** a signal sent by the bus interface to the CPU. This signal indicates the completion of a transaction.

For example, a read transaction is implemented as follows: When the CPU wishes to read data from a slave, it writes the address (combined addresses of the slave and the data item) to the address register $R2$ and sets the `rd_req` signal to "1". The bus interface, handles the request, and initiates a read transaction over the RESA bus. When the data is fetched, it is stored by the bus interface in the data-in register $R0$, and the `done` signal is set to 1 for one clock cycle. The CPU will then immediately change the `rd_req` signal back to "0".

1. Describe how a write transaction takes place.

2. Draw the datapath, registers, and drivers of this bus interface.

3. Design the control logic of the bus interface. Write the equations for: the register clock enable signals, the output enable signals of drivers, and all the RESA bus signals. Use active low signals for all the signals that require a wired-OR mechanism.

4. Draw the timing diagram of all the signals described above in the bus interface for a read transaction and a write transaction.

5. Project "ex1" in the lab contains 4 macros: a CPU, a Bus Interface, a Bus Controller, and a Slave. The CPU alternates between read and write transactions. Your task is to design the Bus Interface macro.

   Remarks: (a) The Bus Controller is very simple. It returns a bus grant for every bus request. (b) The Slave only returns an `ack` signal. This means that the Data signals on the RESA bus are completely ignored both by the Slave and by the CPU.

   (a) Design the Bus Interface macro (design entry). Submit a printout of your schematics and VHDL programs.

   (b) Simulate the project using the following input waveforms: a 100ns period `clk` and a `reset` pulse of duration 200ns. Submit a printout of your simulation for a read transaction and a write transaction.