

Memory Buses*

Charles B. Cameron

12 November 2008

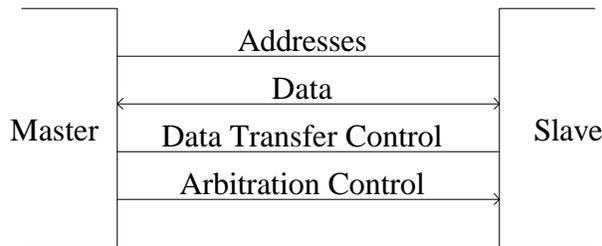


Figure 1: Memory Buses

1 Components of a Memory-Bus

The purpose of a memory bus is to convey data from a master device (a processor) to a slave device (a memory) and vice versa. There are several aspects to a bus. Figure 1 shows the principal components of a memory bus.

Addresses are provided by the master to the slave. Since addresses have n bits, there will be 2^n possible addresses in the system. It is quite common, though, to have a system in which not all locations are implemented. For example, we might have a system with 10-bit addresses, capable of handling $2^{10} = 1024$ distinct addresses, yet containing only a single 8-bit address memory device with 256 locations within it. The remaining 768 locations would be unimplemented.

Once the master has provided a valid address (one corresponding to an implemented memory location), a data transfer will ensue. Whether this is a write

(a transfer from the master to the slave) or a read (a transfer in the opposite direction), the data will flow across the data lines of the memory bus. The decision about whether to perform a read or a write is made by the master and signaled via the control bus. The control bus typically includes additional controls such as a chip select to designate a single slave device. Doing this permits multiple slaves to be connected to the same data lines without the risk of more than one of them trying to place data on the data lines.

It is also possible to have multiple masters share the same address, data, and control lines in a memory bus. This requires additional control lines, shown in Figure 1 as Arbitration Control lines. The arbitration control selects a single master to place data on the address lines and the control lines. The master does not take over the arbitration lines: these are always handled by the same arbitration control unit. However, this might be a task permanently assigned to one of the master devices rather than to a distinct device.

There are two ways for multiple masters to signify a desire to get control of the memory bus. One of these relies on an open-collector line, the other relies on a tristate line. It is permissible for multiple masters simultaneously to request control of the bus if the request line is an open-collector line. If this happens on a tristate request line, a short circuit may result since one contending master might be trying to put a logical 1 on the line and another might be trying to put a 0 on the same line. There are two commonly used approaches to making the tristate request line work:

- The arbitration controller selects one master at

*Course notes for EE361 Microprocessor-based Digital Design

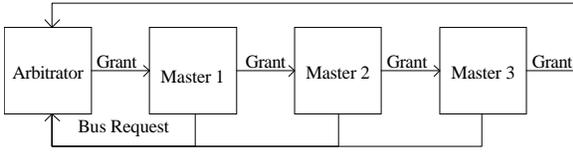


Figure 2: Arbitration via a Daisy Chain

a time and polls it to see if it wants control of the memory bus.

- There are separate request lines for each of the competing masters. The arbitration controller might use a priority encoder for choosing the winning contender.

When an open-collector request line is used, there need only be one such line. However, the arbitration controller still needs a way of discovering which master requested control of the memory bus.

Some systems use a daisy-chain arrangement, as shown in Figure 2. In this scheme, a master requests access to the memory bus by asserting the Bus Request signal, which is typically an open-collector line. The arbitrator cannot know which master requested the access but issues a Grant signal to the first master in the daisy chain. This device therefore has the first crack at accessing the bus. If that master did not request access to the bus, it simply passes the grant on to the next device in the daisy chain. On the other hand, if it did request access, it refrains from passing on the grant until it has finished using the bus.

The last master in the daisy chain behaves in the same manner. However, its output grant is returned to the arbitrator. On seeing the grant come back, the arbitrator knows that all devices have had a chance to use the bus and are now finished with it. Once a master has passed the grant on to the next master in the daisy chain, it must also refrain from using the memory bus itself and will have to wait for another arbitration cycle. For now, it has missed its chance.

As shown in Figure 2, the daisy-chain gives access to the bus to every master which needs it, albeit in a particular order of priority. In the version of the scheme shown in Figure 3, the arbitrator intervenes

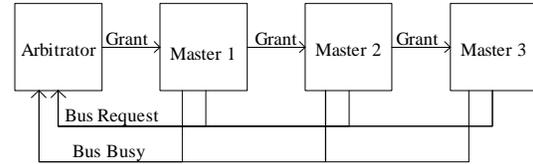


Figure 3: Daisy-Chain Arbitration of a Single Master

after every bus access. Once a master has received the grant signal, it asserts the Bus Busy signal and withdraws (de-asserts) the Bus Request. At this point the arbitrator withdraws the Grant signal. This does not mean that the master must relinquish control of the bus, however. Once the master has finished with the bus it does not relay the grant signal to the next master in the daisy chain, as is the case in the scheme shown in Figure 2. Instead, it signals the arbitrator that it is finished by withdrawing the Bus Busy signal.

In the meantime, other masters may or may not have asserted the Bus Request line. If none have then the the arbitrator need do nothing until the next time the Bus Request line is asserted by some master. If another device has already requested the bus while the first master has been using it, then the arbitrator can issue a new grant right away.

An advantage to the scheme in Figure 3 over that in Figure 2 is that high-priority devices will never have to wait for more than one lower-priority master to finish with the bus before they get service again. A disadvantage is that low-priority masters can be completely blocked from ever getting access to the memory bus if higher priority masters are always using it. The designer must carefully assess these factors in deciding which scheme to use.

Under the daisy-chain arrangement, the amount of time a master waits to get access to the bus depends on several significant factors:

- Where is it in the chain? The masters closest to the arbitrator have the highest priority. For the scheme in Figure 2 this means they will be serviced first within a cycle. For the scheme in Figure 3 it may allow them to totally monopolize

the bus. When only one master requests access to the bus, the position of the master is of less significance. Only the time for the grant signal to propagate through the daisy chain then affects the access delay.

- How many masters are in the chain? Generally speaking, the more masters there are competing for a bus, the lower will be the access speed, at least for low-priority masters.
- How often does a given master require access to the bus? If the bus is in particularly high demand, with every master needing frequent access, masters may have to wait frequently for their turn to use the data bus.
- For how long does a master use the bus? Some masters, such as Direct Memory Access (DMA) controllers may need it for lengthy bursts of data. All other uses are completely blocked while this is going on.

2 Bus Handshake Protocols

We shall consider four kinds of bus handshake protocols:

1. Synchronous
2. Asynchronous
3. Semisynchronous
4. Split cycle

2.1 Synchronous Bus Handshakes

In many ways Synchronous Bus Handshakes are the easiest to implement. All transfers are performed according to a clock transmitted by the master and the clock is the only control signal needed. A single data transfer occurs on every clock cycle. A slave is required to keep up with the master, a requirement which constrains either the clock rate or the choice of slaves in a system. This is usually the fastest kind of transfer.

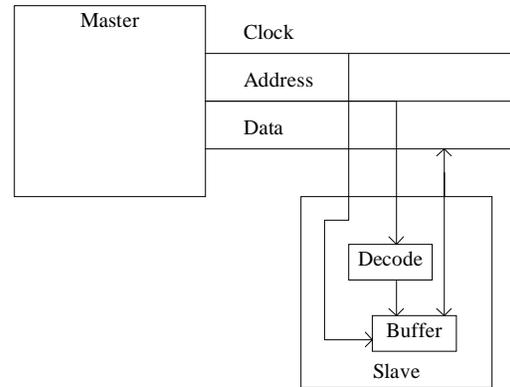


Figure 4: Synchronous Handshaking

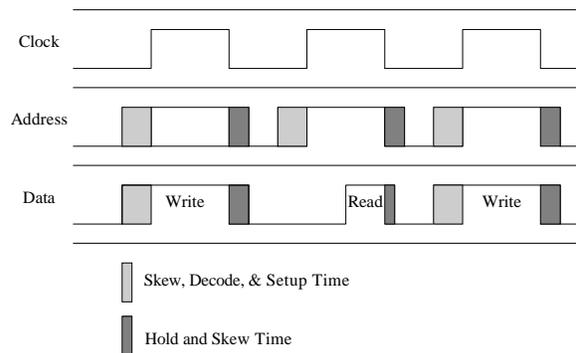


Figure 5: Timing for Synchronous Bus Handshaking

Figure 4 shows the general arrangement. For a write operation, the master places an address and data on the bus. A data transfer takes place when the clock cycle begins. The address may be partially decoded by circuitry outside the memory device, circuitry which activates the designated slave by issuing a chip-select signal to it. Additional decoding of the address is usually necessary within the slave device because more than one of the addressable locations supported by the address length is likely stored within the device.

There are several sources of delay in the transfer, as indicated in Figure 5.

- Propagation delay from the master to the slave

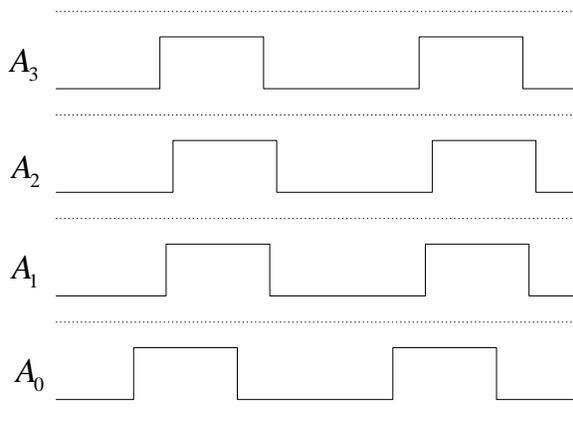


Figure 6: Signal Skew on the Address Bus

and from the slave to the master. The propagation delay often varies from one address or data line to the next, with the result that the signal becomes *skewed*, as shown in Figure 6. Skew may also be introduced by variations in the propagation time through different gates. In fact, typically these are more significant sources of skew. Differences in input capacitance will also induce skew since they will require different amounts of time to cause an input to reach the same voltage.

- Decode time. Although the address decoding circuits are asynchronous (combinational) circuits and so are very fast, they nonetheless take some time to process the address.
- Setup time. A signal may need to be present for some non-zero amount of time before being used.
- Hold time. A signal may need to be present for some non-zero amount of time after being used.

The figure shows two examples of a write operation and one of a read.

For both write and read operations, the master must ensure the fully decoded address is available at the address inputs to the slave when the clock cycle begins. It does this by taking into account the maximum amounts of propagation delay, skew, decode delay, and setup time required and sending the

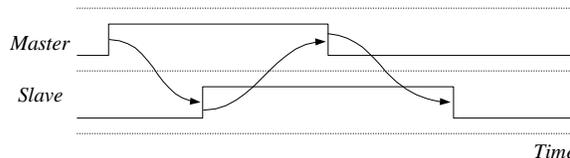


Figure 7: Master-Slave Protocol

address at least that far in advance of the beginning of the clock cycle. The master must also observe the hold time required by the slave. The same requirements apply to data during a write cycle.

The figure suggests that the slave needs half a clock cycle to accept the data from the master but this is not the most common situation. Usually, the leading edge of the clock cycle is used to trigger the transfer. After the slave places data on the bus, that data will not reach the master until another skew delay has elapsed. The master may impose a setup and a hold time on the slave, too. In the figure, the transfer from slave to master is depicted as occurring halfway through the clock cycle and so the master's setup time is not shown, although the skew and hold times are.

If the hold and setup times differ between master and slave or between address and data, the longest requirements must be honored.

The most serious drawback to synchronous bus handshaking is that the clock rate cannot exceed that which can be accommodated by the slowest slave on the bus. If the slowest device is seldom needed, the slow pace it imposes on all other devices can be intolerable.

2.2 Asynchronous Bus Handshakes

There is no clock when Asynchronous Bus Handshaking is used. A pair of control lines is dedicated to coordinating the transfer, as shown in Figure 7. As a result, the principal drawback to the synchronous handshaking scheme is overcome: fast devices respond quickly, even though slow devices may take longer.

The master device operates the **Master** line and

the slave device operates the **Slave** line. A typical handshaking sequence using these two control lines is :

1. Master device asserts the **Master** signal, effectively telling the slave to accept or provide data.
2. Slave device responds by asserting the **Slave** signal, telling the master “OK”.
3. Master responds by withdrawing (de-asserting) the **Master** signal, telling the slave to finish the data transfer.
4. Slave withdraws the **Slave** signal, telling the master it has finished.

This is a completely interlocked asynchronous bus handshaking sequence. It is very widely used because it is so reliable and efficient. However, synchronous handshaking will produce faster transmission in a system where the slaves do not have widely divergent access times.

2.3 Semisynchronous Bus Handshakes

The semisynchronous bus uses the clock signal of the synchronous handshake but adds an additional signal, Wait. If a slave cannot complete its data transfer within a single clock cycle it asserts the Wait signal, telling the master to insert extra clock cycles before completing the operation. Thus the semisynchronous bus has attractive features of both the synchronous and the asynchronous bus: slow slaves can be serviced, as they can on the asynchronous bus, without interfering with the rapid operation of fast slaves, as they do on the synchronous bus.

There is a limit on the physical length of the semisynchronous bus, imposed because the Wait signal must reach the master before the clock cycle ends. This length is the distance traversed in one half the round-trip signal time. Asynchronous buses do not suffer from this restriction: the master will wait as long as necessary for the slave to respond.

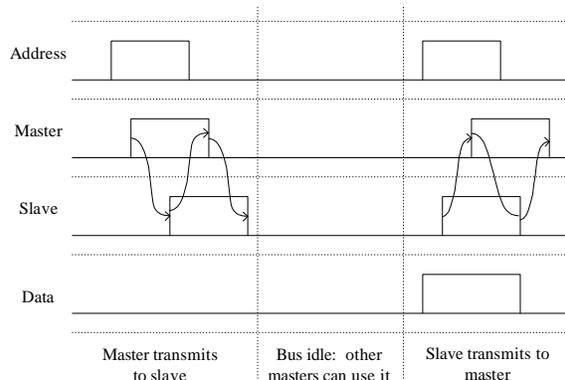


Figure 8: Split-Cycle Bus Timing

2.4 Split-Cycle Bus Handshakes

Another scheme for supporting a mixture of fast and slow devices is the split-cycle bus. It uses an asynchronous bus, with a twist, as depicted in Figure 8. In the first phase of communication, the master sends a command to the slave. However, it does not wait for a reply, instead terminating the exchange. Some time later, when the slave has completed its task, it sends data back to the master by taking on the role of the master and treating the master as a slave. The hardware required to do this is more complex, though, because both the master and the slave must be capable of controlling the bus and a bus arbitrator is absolutely required.

The read command is a little different in split-cycle operation. The master must pass the slave its address when it contacts the slave. The slave uses this address later to call the master up again. So the two halves of a split read cycle really both are writes: one by the master, one by the slave.

References

- [1] Stone, Harold S., *Microcomputing Interfacing*, Reading, Massachusetts, Addison-Wesley Publishing Company, 1982.