Bus Basics

A *bus* is a group of signals (wires) that communicates among several devices. The bus that directly connects the CPU with memory is called the *system bus* (or *memory bus*). I/O devices may connect directly to the system bus as well. Load and store instructions cause the CPU to issue read and write operations to the memory and I/O devices on the system bus.

A typical system bus is composed of three smaller busses:

1. address
2. data
3. control

The address & data busses each carry a single value, i.e., an *n*-wire bus carries values in the range 0 ... 2^n-1. For now, let's imagine a simple bus with 32 address lines and 32 data lines.

The control bus is more a collection of independent signals. Our very simple bus has only three control signals:

- RD/$\overline{\text{WR}}$
- $\overline{\text{ENABLE}}$
- $\overline{\text{ACK}}$ (acknowledge)

Bus Transactions

A complete sequence of bus actions required to perform a read or write is called a *bus transaction*.

Most bus transactions involve two devices. The device that initiates the operation is called the *master.* The other device (which responds to the master) is called the *slave*. A particular device may be a master in some transactions and a slave in others.

A simplified read transaction on our simple bus:

1. The bus master (e.g. the CPU) simultaneously:
   1. drives the address it wants to read on the address bus
   2. drives the RD/$\overline{\text{WR}}$ control signal high
   3. asserts the $\overline{\text{ENABLE}}$ control signal
2. A particular memory location or I/O device register (the slave) recognizes its address on the address bus with $\overline{\text{ENABLE}}$ asserted and RD/$\overline{\text{WR}}$ high, so it:
   1. drives its data value on the data bus
   2. asserts the $\overline{\text{ACK}}$ signal
3. The master sees the $\overline{\text{ACK}}$ signal, so it reads the data from the data bus and stops driving the address and control signals

Bus Transactions (cont'd)

A simplified write transaction on our simple bus:

1. The bus master (e.g. the CPU) simultaneously:
   1. drives the address it wants to write on the address bus
   2. drives the data value it wants to write on the data bus
   3. drives the RD/$\overline{\text{WR}}$ control signal low
   4. asserts the $\overline{\text{ENABLE}}$ control signal
2. A particular memory location or I/O device register (the slave) recognizes its address on the address bus with $\overline{\text{ENABLE}}$ asserted and RD/$\overline{\text{WR}}$ low, so it:
   1. takes the new value from the data bus and stores it
   2. asserts the $\overline{\text{ACK}}$ signal
3. The master sees the $\overline{\text{ACK}}$ signal, so it stops driving the address, data, and control signals

A Very Simple Memory

Let's say we want to build a one-word (32-bit) memory location at address 0xFFFF0000. We can do this in two easy steps:

1. Build a circuit that stores the value sent when the CPU writes to the address 0xFFFF0000.
2. Add logic to send that value back to the CPU when it reads from the address 0xFFFF0000.

What instruction sequence would test this memory location?

A Very Simple Output Device

An I/O register is not very different from a memory location. For example, we can interface an LED to the bus in two easy steps:

1. Build a one-bit, write-only “memory location”, say at the least significant bit (bit 31) of address 0xFFFF0008.

2. Add a circuit that turns the LED on or off based on the value stored in the memory.

What instruction sequence would turn this LED on?

What instruction sequence would turn this LED off?

How would you modify the circuit to let the CPU determine whether the LED is on or off?

A Very Simple Input Device

Input is even easier. Let’s interface a pushbutton switch such that the least significant bit of an I/O register at address 0xFFFF0010 reflects its state (pushed or not pushed).

What are the basic requirements of this interface?

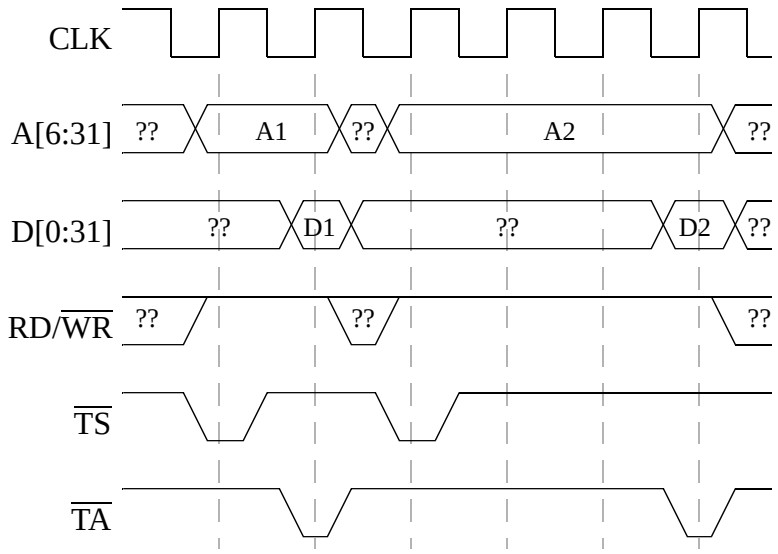
What does the circuit look like?

MPC823 Bus

The basic function of the actual MPC823 bus you'll be using is similar, but slightly more complicated. See Chapter 13 of the data book.

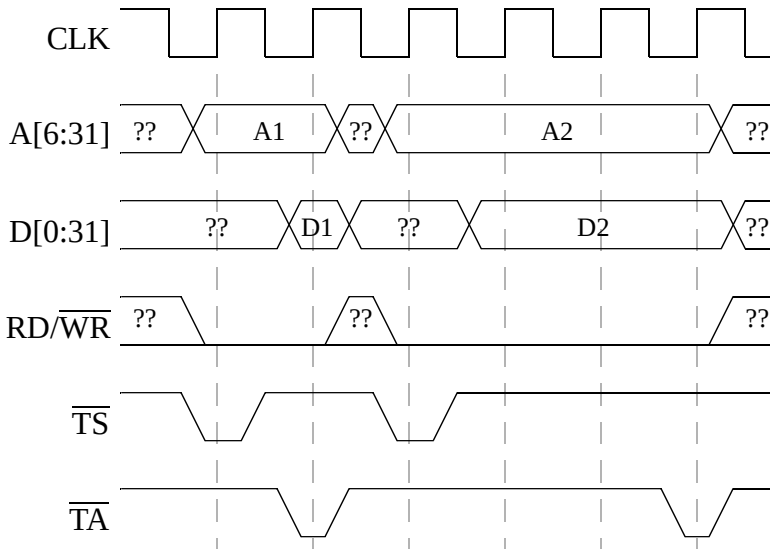
- Timing is controlled by a global clock; all signals are defined in reference to the *rising edge* of this clock.
- 32-bit data bus D[0:31]
- 26-bit address bus A[6:31]
  - A[0:5] not sent off chip
  - on-chip peripherals still see 32 address bits
- basic control lines:
  - RD/$\overline{\text{WR}}$
  - $\overline{\text{TS}}$ (transfer start) — like $\overline{\text{ENABLE}}$, but only asserted on first clock cycle of transaction
  - $\overline{\text{TA}}$ (transfer acknowledge) — like $\overline{\text{ACK}}$

MPC823 Bus Example: Read



- Master drives address, RD/$\overline{\text{WR}}$, and $\overline{\text{TS}}$ to initiate a read transaction. Address and RD/$\overline{\text{WR}}$ guaranteed valid at same rising clock edge where $\overline{\text{TS}}$ is asserted. Master deasserts $\overline{\text{TS}}$ after one cycle, but keeps driving address and RD/$\overline{\text{WR}}$ until it sees $\overline{\text{TA}}$.
- Slaves look at address and RD/$\overline{\text{WR}}$ when $\overline{\text{TS}}$ asserted. One will drive data and assert $\overline{\text{TA}}$. Master samples data on same rising clock edge where $\overline{\text{TA}}$ is asserted.
- Minimum transaction takes two clock cycles.
- Transactions can take longer; slow slaves add wait states by not asserting $\overline{\text{TA}}$.

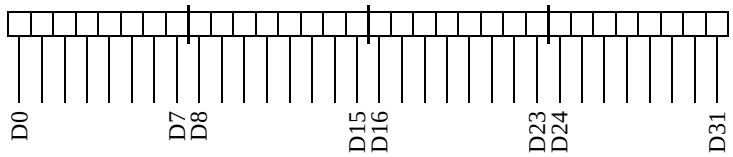
MPC823 Bus Example: Write



- Starts similar to read, except for sense of RD/$\overline{WR}$.
- Master drives data by *second* cycle.
- Again, slaves look at address and RD/$\overline{WR}$ when $\overline{TS}$ asserted. One will read data and assert $\overline{TA}$.
- Again, minimum transaction takes two clock cycles, but slaves can take longer by not asserting $\overline{TA}$. Master will keep driving address, RD/$\overline{WR}$, and data until it sees $\overline{TA}$.

Dealing with Smaller Accesses

- Consider a 32-bit memory location at address 0x1000:



- Assume the word value stored at 0x1000 is 0x12345678, and r4 contains 0x1000. What happens on:
  - lwz r3, 0(r4)
  - lbz r3, 0(r4)
  - lbz r3, 1(r4)
  - lbz r3, 2(r4)
  - lbz r3, 3(r4)
  - lhz r3, 0(r4)
  - lhz r3, 2(r4)
- What must the 32-bit memory location do to handle byte and halfword reads?
- What does this imply about where the CPU expects data on data bus?

Smaller Accesses: Writes

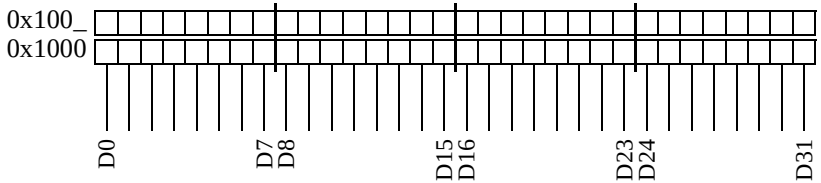
- What about stb r2, 1(r4)?

- On a write access, what two factors determine which bits in a 32-bit word are updated?

- On most wider busses, the master drives *byte enable* lines instead of less significant address bits:
  - Moto 68000 (16-bit bus): $\overline{\text{LDS}}$, $\overline{\text{UDS}}$ (no address LSB)
  - 32-bit busses: replace low 2 addr bits w/4 byte enables
  - MPC823 does not:
    - full byte address provided (incl. A[30] & A[31])
    - size (byte, halfword, word) encoded on two control lines (TSIZ[0-1], see Table 13-4)
    - equivalent, but messier: basically you have to generate your own byte enables

Unaligned Accesses

- Consider two adjacent 32-bit memory locations:



- What is the address of the second location?

- Assume r4 = 0x1000. What happens if the CPU executes a 'lwz r3, 1(r4)'?

- The fundamental problem with unaligned accesses is that they may require multiple bus transactions. The MPC823 will handle this in hardware, but some will not... instead, your program will crash.