Lecture Five: Interfacing and Communication

6. Introduction to Universal Serial Bus (USB)

6.1. Basic Principles

The Universal Serial Bus (USB) is a host-controlled, token-based high-speed serial network that allows communication between many of devices operating at different speeds. There are a number of USB products that enable combining USB into an embedded system. USB devices usually exist within the same room, and are typically less than 4 meters from each other. USB 2.0 supports three speeds.

High Speed - 480Mbits/s

Full Speed - 12Mbits/s

Low Speed - 1.5Mbits/s

The original USB version 1.1 supported just full speed mode and a low speed mode. The Universal Serial Bus is host-controlled, which means the host regulates communication on the bus, and there can only be one host per bus. On the other hand, version 2.0, includes a Host Cooperation Protocol that allows two devices cooperate for the role of host. Data can be sent by various operation methods using a token-based protocol. USB uses a star topology, using a hub to connect additional devices. A hub is at the center of each star. Each wire segment is a point-to point connection between the host and a hub or function, or a hub connected to another hub or function, as shown in Figure 5.13. Because the hub provides power, it can monitor power to each device switching off a device drawing too much current without disrupting other devices. The hub can filter out high speed and full speed transactions so lower speed devices do not receive them. Because USB uses a 7-bit address, up to 127 devices can be connected.

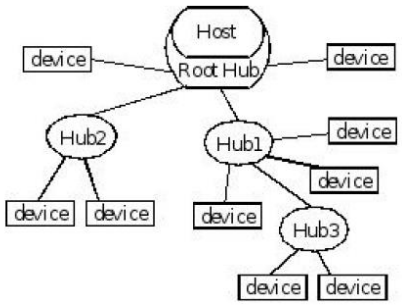

Figure 5.13. USB network topology.

The electrical specification for USB used four shielded wires (+5V power, D+, D- and ground). The D+ and D- are twisted pair differential data signals. It uses **Non Return to Zero Invert** (**NRZI**) encoding to send data with a sync field to synchronize the host and receiver clocks.

USB drivers will dynamically load and unload. When a device plugged into the bus, the host will detect this addition, probe the device and load the appropriate driver. Similarly, when the device is unplugged, the host will detect its absence and automatically unload the driver. The USB architecture understands four basic types of data transfers:

• Control Transfers: Used to configure a device at attach time.

• Bulk Data Transfers: Generated large quantities and have wide dynamic latitude in transmission constraints.

• Interrupt Data Transfers: Used for timely but reliable delivery of data.

• Isochronous Data Transfers (Also called streaming real time transfers). Isochronous transfer is appropriate for real-time applications like in audio or video applications.

A USB device indicates its speed by pulling either the D+ or D- line to 3.3 V, as shown in Figure 5.14. A pull-up resistor attached to D+ specifies full speed, and a pull-up resistor attached to D- means low speed. These device-side resistors are also used by the host or hub to detect the presence of a device connected to its port. Without a pull-up resistor, the host or hub assumes there is nothing connected.

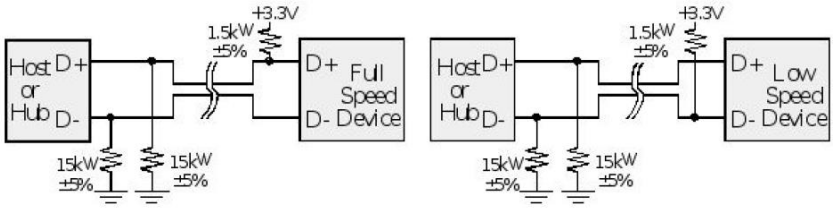

Figure 5.14. Pull-up resistors on USB devices signal specify the speed.

Each USB operation consists of three packets

Token Packet (header),

Optional Data Packet, (information) and

Status Packet (acknowledge)

The host initiates all communication, beginning with the Token Packet, which describes the type of transaction, the direction, the device address and designated endpoint. The next packet is generally a data packet carrying the information and is followed by a handshaking packet, reporting if the data or token was received successfully, or if the endpoint is not available to accept data. Data is transmitted least significant bit first. Some USB packets are shown in Figure 5.15. All packets must start with a sync field. The **sync** field is 8 bits long at low and full speed or 32 bits long for high speed and is used to synchronize the clock of the receiver with that of the transmitter. **PID** (Packet ID) is used to identify the type of packet that is being sent, as shown in Table 5.3. The **address** field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported. The **endpoint** field is made up of 4 bits, allowing 16 possible endpoints. **Cyclic Redundancy Checks** are performed on the data within the packet payload. All token packets have a 5- bit CRC while data packets have a 16-bit CRC. EOP stands for **End of packet**. **Start of Frame** Packets (SOF) consist of an 11-bit frame number is sent by the host every 1ms ± 500ns on a full speed bus or every 125 μs ± 0.0625 μs on a high speed bus.

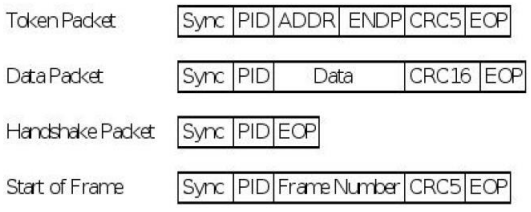

Figure 5.15. USB packet types.

Group	PID Value	Packet Identifier
Token	0001	OUT Token, Address + endpoint
	1001	IN Token, Address + endpoint
	0101	SOF Token, Start-of-Frame marker and frame number
	1101	SETUP Token, Address + endpoint
Data	0011	DATA0
	1011	DATA1
	0111	DATA2 (high speed)
	1111	MDATA (high speed)
Handshake	0010	ACK Handshake, Receiver accepts error-free data packet
	1010	NAK Handshake, device cannot accept data or cannot send data
	1110	STALL Handshake, Endpoint is halted or pipe request not supported
	0110	NYET (No Response Yet from receiver)
Special	1100	PREamble, Enables downstream bus traffic to low-speed devices.
	1100	ERR, Split Transaction Error Handshake
	1000	Split, High-speed Split Transaction Token
	0100	Ping, High-speed flow control probe for a bulk/control endpoint

Table 5.3. USB PID numbers.

USB **functions** are USB devices that provide a capability or function such as a Printer, Scanner, Modem or other peripheral. Most functions will have a series of buffers, typically 8 bytes long. **End point s**can be described as sources or sinks of data, shown as **EP0In** , **EP0Out** etc. in Figure 5.16. As the bus is host centric, endpoints occur at the end of the communications channel at the USB function. The host software may send a packet to an endpoint buffer in a peripheral device. If the device wishes to send data to the host, the device cannot simply write to the bus as the bus is controlled by the host. Therefore, it writes data to endpoint buffer specified for input, and the data sits in the buffer until such time when the host sends a IN packet to that endpoint requesting the data. Endpoints can also be seen as the interface between the hardware of the function device and the firmware running on the function device.

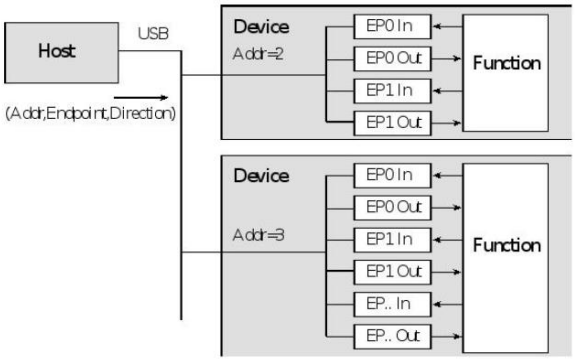

Figure 5.16. USB data flow model.

7. General Approaches to High-Speed Interfaces

7.1. Hardware FIFO

The **hardware FIFO** can be placed between the I/O device and the computer. Assume in this situation, the average serial bandwidth is low enough for the software to read the data from the serial port and write it to memory. Without the hardware FIFO, the latency requirement of a serial input port is the time it takes to transmit one data frame. To reduce this latency requirement (without changing the average bandwidth requirement) we can add a hardware FIFO between the receive shift register and the receive data register, as illustrated in Figure 5.19. Many of the I/O devices on the Texas Instruments microcontrollers employ hardware FIFOs.

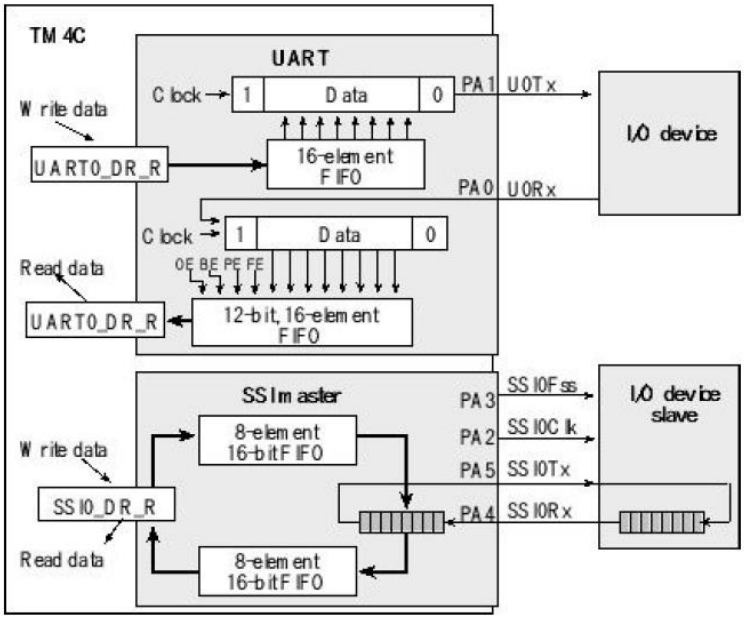

Figure 5.19. High-speed I/O devices employ hardware FIFOs to reduce the latency requirement of the interface.

A hardware FIFO, placed between the output data register and the transmit shift register, allows the software to write multiple bytes of data to the interface and then perform other tasks while the frames are being sent.

7.2. Dual Port Memory

One approach that allows a large amount of data to be transmitted from the software to the hardware is the dual port memory, Figure 5.20. A dual port memory allows shared access to the same memory between the software and hardware. For example, the software can create a graphics image in the dual port memory using standard memory write operations. At the same time the video graphics hardware can fetch information out of the same memory and display it on the computer monitor. In this way, the data need not be explicitly transmitted from the computer to the graphics display hardware. To implement a dual port memory, there must be a way to decide the condition when both the software and hardware wish to access the device simultaneously. One mechanism to decide simultaneous requests is to stop the processor using a MRDY signal so that the software temporarily waits while the video hardware fetches what it needs. Once the video hardware is done, the MRDY signal is released and the software resumes. Most microcontroller memory interfaces do not support this sort of hardware initiated cycle

stretching. If both processors wish to access the memory at the same time, one of the processors is delayed.

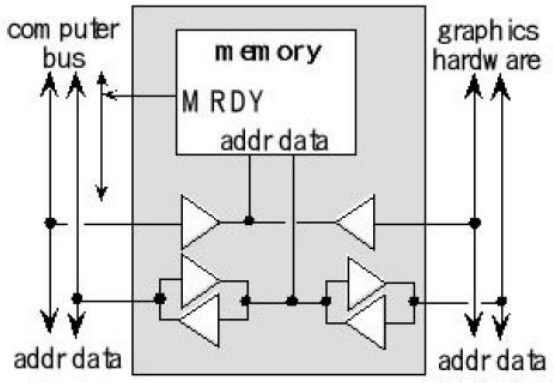

Figure 5.20. A dual port memory can be accessed by two different modules.

7.3. Bank-Switched Memory

Another approach similar to the dual port memory is the bank-switched memory, see Figure 5.21. A bank-switched memory also allows shared access to the same memory between the software and hardware. The difference between bank-switched and dual port is the bank-switched memory has two modes. In one mode (M=1), the computer has access to memory bank A, and the I/O hardware has access to memory bank B. In the other mode (M=0), the computer has access to memory bank B, and the I/O hardware has access to memory bank A. Because access is restricted in this way, there are no conflicts to resolve.

Graphics controllers use bank switching. One processor transfers data from the front buffer and displays it on the screen. A second processor builds the next image in the back buffer. To create the video output, the buffers are switched at a regular rate. Many high speed data acquisition systems all employ bank switching. The ADC hardware can write into one bank while the computer software processes previously collected data in the other. When the ADC sampling hardware fills a bank, the memory mode is switched, and the software and hardware swap access rights to the memory banks. In a similar way, a real-time waveform generator or sound playback system can use the bank-switched approach. The software creates the data and stores it into one bank, while the hardware reads data from the other bank that was previously filled. Again, when the hardware is finished, then the memory bank mode is switched.

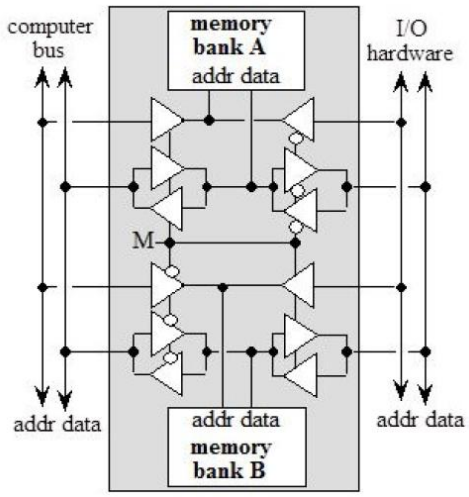

Figure 5.21. A bank-switched memory can be accessed by two different modules, one at a time.

7.4. Fundamental Approach to DMA

With a software-controlled interface (busy-wait or interrupts) if we wish to transfer data from an input device into RAM, you must first transfer it from input to the processor, then from the processor into RAM. In addition, this transfer is explicitly controlled by executing software. In order to improve performance, we will transfer data directly from input to RAM or RAM to output using **D**irect **M**emory **A**ccess, DMA. Because DMA bandwidth can be as high as the bus bandwidth, we will use this method to interface high bandwidth devices like disks, digital scopes, cameras, and networks. Similarly, because the latency of this type of interface depends only on hardware and is usually just a couple of bus cycles, we will use DMA for situations that require a very fast response. On the other hand, software-controlled interfaces have the potential to perform more complex operations than simply transferring the data to/from memory. For example, the software could perform error checking, convert from one format to another, implement compression/decompression, and detect events. These more complex I/O operations may preclude the usage of DMA.

7.4.1. DMA Cycles

During a DMA read cycle, the processor can still access flash memory and ROM, while hardware automatically transfers data from RAM to the output device (Figure 5.22). The address on the bus specifies the RAM location from which to read the data. The µDMA controller on the TM4C has many different configuration options to burst transfer data to and from arbitrary locations. For example, it may automatically increment the RAM source address to stream an array to an output device. The TM4C series does not support DMA transfers with flash memory or ROM because they are on a separate internal bus.

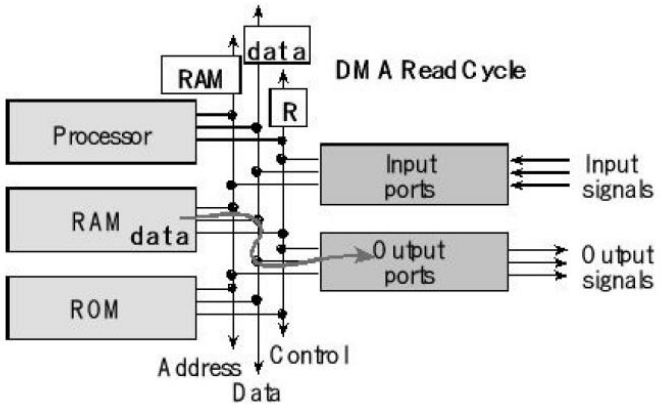

Figure 5.22. A DMA read cycle copies data from RAM to an output device.

During a DMA write cycle, the processor can still access flash memory and ROM, while hardware automatically transfers data from the input device to RAM (Figure 5.23). The address on the bus specifies the RAM location to which to write the data. A useful configuration mode could be to have the μDMA controller automatically increment the RAM destination address to stream data from an input device. In some DMA interfaces, two DMA cycles are required to transfer the data. The first DMA cycle brings data from the source into the DMA module, and the second DMA cycle sends the data to its destination.

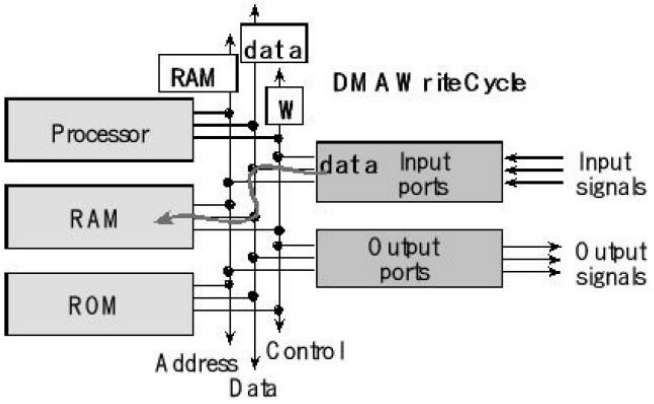

Figure 5.23. A DMA write cycle copies data from the input device into RAM.