Overview

° Parallel Interfacing
° Serial Interfacing
  • UART
  • RS232

Anatomy: 5 components of any Computer

- Processor (active)
  ° Control ("brain")
  ° Datapath ("brawn")
- Memory (passive)
- Devices
  ° Input
  ° Output
- Computer
  "Computer" (where programs, data live when running)
  "Control" ("brain")
  "Datapath" ("brawn")

Memory

- PCI: Internal (Backplane) I/O bus
- SCSI: External I/O bus

Review: Buses in a PC: Connect a few devices

- CPU Memory bus
- PCI: Internal (Backplane) I/O bus
- SCSI: External I/O bus

Data rates

- Memory: 133 MHz, 8 bytes
  \[ 1064 \text{ MB/s (peak)} \]
- PCI: 33 MHz, 8 bytes wide
  \[ 264 \text{ MB/s (peak)} \]
- SCSI: “Ultra3” (80 MHz), “Wide” (2 bytes)
  \[ 160 \text{ MB/s (peak)} \]
- Ethernet: 12.5 MB/s (peak)
Review: I/O Device Examples and Speeds

- **I/O Speed:** bytes transferred per second (from mouse to display: million-to-1)

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate (Kbytes/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Input</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Input</td>
<td>Human</td>
<td>0.02</td>
</tr>
<tr>
<td>Line Printer</td>
<td>Output</td>
<td>Human</td>
<td>1.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Network-LAN</td>
<td>I or O</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
</tbody>
</table>

Review: DSLMU I/O Addressing

<table>
<thead>
<tr>
<th>Offset</th>
<th>Mode</th>
<th>Port Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>R/W</td>
<td>Port A</td>
<td>Bidirectional data port to LEDs, LCD, etc.</td>
</tr>
<tr>
<td>0x04</td>
<td>R/W</td>
<td>Port B</td>
<td>Control port (some bits are read only)</td>
</tr>
<tr>
<td>0x08</td>
<td>R/W</td>
<td>Timer</td>
<td>8-bit free-running 1 kHz timer</td>
</tr>
<tr>
<td>0x0C</td>
<td>R/W</td>
<td>Timer Compare</td>
<td>Allows timer interrupts to be generated</td>
</tr>
<tr>
<td>0x10</td>
<td>RO</td>
<td>Serial RxD</td>
<td>Read a byte from the serial port</td>
</tr>
<tr>
<td>0x10</td>
<td>WO</td>
<td>Serial TxD</td>
<td>Write a byte to the serial port</td>
</tr>
<tr>
<td>0x14</td>
<td>WO</td>
<td>Serial Status</td>
<td>Serial port status port</td>
</tr>
<tr>
<td>0x18</td>
<td>R/W</td>
<td>IRQ Status</td>
<td>Bitmap of currently-active interrupts</td>
</tr>
<tr>
<td>0x1C</td>
<td>R/W</td>
<td>IRQ Enable</td>
<td>Controls which interrupts are enabled</td>
</tr>
<tr>
<td>0x20</td>
<td>WO</td>
<td>Debug Stop</td>
<td>Stops program execution when written to</td>
</tr>
</tbody>
</table>

Parallel Interfacing

- **In Parallel** multiple bytes are transferred between the processor and external devices.
  - Mem ↔ Processor 1, 2 or 4 bytes
  - LCD ↔ Processor 1 byte
  - The advantage: speed all data bits are transferred simultaneously via the system bus (or an extension of this bus).

Parallel Interfacing Problems

- **More cost:** one wire for each bit + 1 bit for clock (strobe).
- **May suffer from skew problem due to unequal time delay for each signal.**

Used for high data rates over short distances <few cm
Serial Interfacing

- In serial I/O, the data bits are sent one at a time across a single line.
  - The advantage of serial I/O is lower cost (in terms of the number of wires connecting the microcomputer to peripheral device)
  - The disadvantage is slower speed.

Parallel <-> Serial Interfacing

- Since communication within a microprocessor takes place over the system bus in parallel form, there is obviously a need for parallel-to serial (and serial-to parallel) conversion when interfacing to serial devices.

Asynchronous Serial Communication

- Used in character oriented data transmission between a microprocessor and an external device
  - Transmitter and Receiver each has its own clock running at the same frequency
  - How to synchronize two clocks so to sample in the middle of the data?

Making Asyn. Transmission Work

- Receiver Synchronisation:
  - The transmission of first bit should starts with a transition on the data line (1->0)
  - send an extra ‘start’ bit (= 0) before sending the 8-bit data,
  - data line is always set back to 1 at the end.
  - 1 -> 0 transition always occurs at the start of each transmission.
  - the receive clock now samples 9 bits (start + 8 data bits),
  - the gap (idle time) between successive groups of 9 bits can change
  - Character wide synchronisation (Asynchronous)
Receiver Clock Synchronisation Issues

- The receiver clock can be made equal to the baud rate.
- Clock must be very accurate in order to sample the incoming bit stream in the centre of its cycle.
- The sample point needs to be very close to the centre of the bit cell for reliable data recovery.
- The actual variation from the centre on the bit cell is referred to as 'ratchet error'.

Improving Receiver Clock Synchronisation

- If the clock is made 16 times the baud rate, then the ratchet error can be relaxed from ±1% to ±5%.
- Ratchet relaxes to ±25% for 64 times the baud rate.

Parallel ↔ Serial Conversion

- Asynchronous data transmission uses a special device called Universal Asynchronous Receiver Transmitter (UART).
- UART is used to simultaneously transmit and receive serial data.
- Performs the appropriate parallel/serial conversions and inserting or checking the extra bits used to keep the serial data synchronised.
- UART typically configured as 2-4 I/O addresses: input/output status port(s), and output/input data port(s).
- Bytes sent as 8-bit parallel data to the output data address by the computer are converted into a standard-format serial bit stream for transmission by a transmitter inside the UART.
- Similarly, an incoming serial bit stream is detected by a receiver inside the UART and converted into parallel data that can be read by the computer from the UART's input data address.

Full Duplex VS Half Duplex Data Transmission

- Simultaneous conversion of an incoming and an outgoing serial data stream is called full duplex.
  - It requires two data carriers (TxD, and RxD).
  - Implemented with three wires: one for the outgoing stream (TxD), one for the incoming stream (RxD), and the third for a common ground line.
  - The UART does provide for standard full duplex handshaking conventions.
- Half duplex allows two-way communications, hence the name duplex, but only one direction is active at a time.
Synchronous Serial Data Transmission

- In Asynchronous data transmission TX and RX clocks are unsynchronised
  - Inefficient (for each 7 bits we send 3 – 4 extra bits)
  - Synchonisation across characters

- In Synchronous Data Transmission TX and RX clocks are synchronised
  - A common shared clock, (I2C), or clocking information embedded in the data stream (USB, Ethernet)
  - Fast (many bytes send before a re-synchronisation)
  - Synchonisation across frames vs characters

Serial Data Channels on AT91 on DSLMU Board

- Two Universal Synchronous Asynchronous Receiver Transmitter (USART)
  - Programmable Baud rate
  - Can generate interrupts

DSLMU/KOMODO Serial I/Os

- DSLMU Serial Port 1: memory-mapped terminal (Connected to the PC for program download and debugging)
  - Read from PC Keyboard (receiver); 2 device regs
  - Writes to PC terminal (transmitter); 2 device regs

<table>
<thead>
<tr>
<th>Receiver Status</th>
<th>Unused (00...00)</th>
<th>Ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000014</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Receiver Data</th>
<th>Unused (00...00)</th>
<th>Received Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmitter Status</th>
<th>Unused (00...00)</th>
<th>Ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000014</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmitter Data</th>
<th>Unused</th>
<th>Transmitted Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DSLMU/Komodo Serial I/Os

- Status register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register not yet been read (or ready to be read);
    1 ⇒ 0 when data is read from Data Reg
  - Transmitter: Ready==1 means transmitter is ready to accept a new character;
    0 ⇒ Transmitter still busy writing last char

- Data register rightmost byte has data
  - Receiver: last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display
DSLMU/KOMODO Serial I/Os Interrupts

- IRQ Enable: Enables individual interrupts
- IRQ Status: Indicates raising interrupt.
- When a character is received or sent, an interrupt is raised.

Asyn. Serial Communication Standard (RS232C)

- Standard for communication of ASCII-coded character data between devices such as data computers and modems
  - Low speed and cheap
- Standard definition:
  - The voltages used to represent 0 and 1 (Electrical)
  - The rate at which data is sent.
  - The format of the data sent.
  - The connectors to be used (physical and mechanical)
  - Extra control signals that may be used.
- Typical data rate (baud rate) are: 75, 300, 1200, 2400, 9600, 19200 and 115,000 bits/sec
- Typical frame:

    Data | Start | 1 1 0 0 1 0 | Parity Stop Bits | Idle

    ASCII Character “S”

RS232C Definitions

- The parity bit:
  - is used as an error check.
  - The total number of ‘1’s in the character+parity is made either odd (odd parity) or even (even parity).
  - Any single-bit error makes the parity bit appear wrong.
- The stop bit(s):
  - exist to allow for the case where one frame is transmitted immediately after another.
  - The stop bits, which are always 1, ensure the next start bit’s 1 → 0 transition. (1, 1/2 or 2 bits)
- Voltage values:
  - >±5 should be used (Normally >±13 used)
  - +5 represents logic low (space) and –5 logic high (mark)
- Physical characteristic:
  - 25 way connector, (9 way is more popular now)

From UART to RS232-C

- The UART is responsible for certain parts in RS232-C standard specifications:
  - framing and transmitting TX data
  - receiving and extracting the RX data
  - baud rate generation
- The electrical signaling is handled by a driver
  - logic inversion and voltage translation

R232 Interface in DSLMU
### Non Standard RS-232 Standard

- RS-232 has earned the distinction of being the most non-standard standard in electronics!
  - In general, two RS-232 devices, when connected together, won't work.
- RS-232 was designed for connecting DTEs ("data terminal equipment") (like PC) to DCEs ("data communication equipment") (like modem).
- A DTE has a male and a DCE a female connector
  - Corresponding pins in DTE connector connect to corresponding pins in DCE connector.
- The IBM PC looks like a DTE with a male connector
- The DSLUM board also looks like a DTE with a male connector
- How to connect PC to DSLMU?
  - Use "null modem"; cable that crosses TxD and RxD wires.

### Reading Material

- Reading Material:
  - [http://www.beyondlogic.org/serial/serial.htm](http://www.beyondlogic.org/serial/serial.htm)
  - [http://www.sangoma.com/signal.htm](http://www.sangoma.com/signal.htm)
  - Hardware Reference Manual on CD-ROM

### "And In Conclusion"

- **Parallel Interfacing**
  - Fast but expensive
- **Serial Interfacing**
  - Slow but inexpensive
- **Synchronous Serial Interfacing**
  - Fast and more efficient but requires clock synchronisation
- **Asynchronous Serial Interfacing**
  - Slower and less efficient but does not require clock synchronisation
- **RS232 Standard**
  - The most widely used serial communication standard for communication between DTE and DCE devices