Homework

• Reading
  – 7400 TTL Logic Data Sheets

• Machine Projects
  – MP3 due at start of class today
  – Start looking at MP4 now

• Labs
  – Continue labs with your assigned section
MP3 Discussion

• What did you learn in MP3?
Some Notes on Machine Problem 4

• Two parts:
  – Chip tester
  – Digital oscilloscope

• Both use the parallel port to acquire digital data
  – Chip tester uses parallel port as the source of bits
  – Digital oscilloscope uses the serial port as the source of bits

• Most of the code is written!
  – The number of new lines of code is small 😊
  – But effort to understand other code is substantial 😞
Chip Tester Hardware

- Special device attached to parallel port
- Example shown is one fourth of 74LS00
Chip Tester Software

- Chip data structures central to the testing code
Chip Tester Software

- Chip data structures central to the testing code

```c
typedef struct {
    char *chipname;               /* "LS00" or whatever */
    char *chipdesc;               /* description string */
    int n_highpins;              /* no. of logic-1 pins */
    int *highpins;               /* array of logic-1 pin nos. */
    int n_lowpins;               /* no. of logic-0 pins */
    int *lowpins;                /* array of logic-0 pin nos. */
    int n_inpins;                /* no. of input pins */
    int *inpins;                 /* array of input pin nos. */
    int n_outpins;               /* no. of output pins */
    int *outpins;                /* array of output pin nos. */
    PFI_I_IPTR softchip;         /* does chip logic in software */
} Chip;
```
Chip Tester Configuration

• SAPC boards 5 & 6 have the parallel port wired to test an LS00 (quad NAND gate chip)
• SAPC boards 7 & 8 have the parallel port wired to test an LS138 (3-to-8 decoder chip)
• The given code makes a correspondence between:
  – Data register bits for the parallel port (0:7)
  – The DB25 connector (pins 2-9 and 10-13, 15)
  – The chip pins (1-14 or 1-16)
  – The direction of the chip pins (input or output)
    (Input or output is defined from the chip point of view)
• Note that the LS00 is symmetric in that the pins on the left side of the chip are functionally equivalent to the pins on the right side
ls00_chip.c

- Builds external (global) Chip struct for ls00

static int high_pindata[] = {14}; /* VCC and active high inputs */
static int low_pindata[] = {7}; /* GND and active low inputs */
static int input_pindata[] = {1, 2, 4, 5, 9, 10, 12, 13};
static int output_pindata[] = {3, 6, 8, 11};

Chip ls00 = {"LS00","quad NAND gate",
    sizeof(high_pindata)/sizeof(high_pindata[0]),high_pindata,
    sizeof(low_pindata)/sizeof(low_pindata[0]),low_pindata,
    sizeof(input_pindata)/sizeof(input_pindata[0]),input_pindata,
    sizeof(output_pindata)/sizeof(output_pindata[0]),output_pindata,
    ls00_softchip}
The truth table holds the testing information:

```c
/* truth table:   in pin  5 4 2 1  out pin  6 3 */
static int TT[] = {
    0x03,     /* 0 0 0 0           1 1 */
    0x03,     /* 0 0 0 1           1 1 */
    0x03,     /* 0 0 1 0           1 1 */
    0x02,     /* 0 0 1 1           1 0 */
    0x03,     /* 0 1 0 0           1 1 */
    0x03,     /* 0 1 0 1           1 1 */
    0x03,     /* 0 1 1 0           1 1 */
    0x02,     /* 0 1 1 1           1 0 */
    0x03,     /* 1 0 0 0           1 1 */
    0x01,     /* 1 0 0 1           0 1 */
    0x03,     /* 1 0 1 0           1 1 */
    0x03,     /* 1 0 1 1           1 0 */
    0x01,     /* 1 1 0 0           0 1 */
    0x01,     /* 1 1 0 1           0 1 */
    0x01,     /* 1 1 1 0           0 1 */
    0x00,     /* 1 1 1 1           0 0 */
};
```
ls138_softchip

- Have three bits of input data (A₀, A₁, and A₂)
- Have three bits of enable (E₁#, E₂#, and E₃)
  - Where # means an inverted signal or active low
- Can support only five bits of output data (O₀#, O₁#, …O₇#)
- Since softchip function has two arguments, the input data and the output results one solution would be a 32 entry table – but there are other ways of doing this
Changes to chiptest.c

1. Finish writing code in chiptest function
   Generate the possible input bit patterns to chip
   Read back the output pins from the chip
   Compare output bit patterns to softchip results to see if chip is functioning correctly

2. Test with provided ls00 tables – good and bad

3. Write new tables for LS138

4. Test with your tables – good and bad

5. Write discussion.txt
Digital Oscilloscope

• Hook back the COM1 port to LPT1 but use a level converter (called a line receiver) to make sure voltage conversion is done (We don’t want a “Was” Gate)
• As project description explains, sampling rate is about 100 times faster than bit times (at 9600 baud)
• Using | for high and _ for low, plus ||nn|| or ___nn___ for a sequence of highs and lows, the displayed results might be: ||88||___67____|____||___||82||
  – or “88 highs”, “67 lows”, “1 high”, “2 lows”, “2 highs”, “3 lows”, and “82 highs”
Digital Oscilloscope Hardware

- Connection between serial and parallel port
- Through voltage converter (+/- 12 → 0 – 5V)
UART Receiver Sampling

• Characters are sent/received asynchronously
  – Clocks of receiver and transmitter are independent and only nominally at the same rate (+/- 0.01%)
  – Furthermore, the phases of the clocks relative to each other are completely arbitrary

• Receiver strategy:
  – Synch on initial edge then “center sample” bits
  – Sample 16 times the baud rate, starting with the eighth clock period after leading edge of start bit
UART Receiver Sampling

• “Ideal” Serial Data Waveform

Mark Idle
Start Bit
LSB Data
LSB Data

. . .

• What the Receiver “sees”

Late
Early
Who knows?
Wow!

• Therefore receiver “center samples” data bits to get accurate indication of one or zero state
UART Receiver Sampling

- Receiver runs its clock to check for one or zero state of input RXD signal at 16 times bit rate:

  - Count 8 clock times to get to center of start bit
  - Count 16 clock times to sample at center of each data bit interval

  Avoids “seeing” any glitches between the bit intervals
Setting the Baud Rate

- UART over samples incoming bit
  - Sample rate is 16X baudrate
  - UART has an input clock at 1.843200 MHz
  - Must calculate baudrate divisor:
    \[
    \text{Divisor} = \frac{\text{UART}_{-}\text{BAUD}_{-}\text{CLOCKHZ}}{(\text{baudrate} \times 16)}
    \]
  - Look at serial.h where UART baud rate is set through the Line Control Register (LCR) and DLM/DLL registers
    - Turn on Divisor latch access bit (UART_LCR_DLAB)
    - Load MSB of divide down counter value into UART_DLM
    - Load LSB of divide down counter value into UART_DLL
    - Turn off Divisor latch access bit (UART_LCR_DLAB)
Baudrate Generator

• Set DLAB bit in “Line Control” UART register to use UART registers 0 and 1 for 16-bit divisor value

\[
\text{outpt}(\text{baseport} + \text{UART}_\text{LCR}, (\text{inpt}(\text{baseport} + \text{UART}_\text{LCR})) | \text{UART}_\text{LCR}_\text{DLAB});
\]

• Then put out two bytes containing divisor
  – 9600 bps => divisor of 12
  – 19,000 bps => divisor of 6

![Diagram showing the relationship between DLAB Bit, Address 0, Address 1, DLM/DLL Registers, Receive/Transmit Holding Register(s), Interrupt Enable Register, and Baud Rate Generator (16 times baud rate).]
Remaining Steps in scope.c

• After setting up baud rate
  – Output a character to the serial port (COM1)
  – Collect the data
    • collect.c inputs from the LP_STATUS bit and saves into an array of data points up to a MAXDATA times
    • You are asked to re-implement collect.c as an assembly language routine ascollect.s
      – Code is similar to collect.c in that it loops MAXDATA times moving the LP_STATUS bit into the data array
  – Display the results
    • This code is written for you