Terence Henriod

HW 9

CPE 301: Microprocessors System Design

October 29, 2013

1. Assume the 8250 UART (the same UART as used in PCs) is connected to an ATmega2560 single board computer mapped as external data memory with a start address of FCC0 Hex. Write the following Arduino C language functions. See data sheet below for 8250 special function register details.

For those who are interested, the data memory map for the 2560 is shown in the Atmega2560 data manual - Figure 9-1. External Memory with Sector Select. ASSUME THE 8250 UART IS ALREADY INITIALIZED (so you do not need to write UARTinit()).

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Global Variables (Yes, these are global!)

Mostly used for register pointers that can't be declared constant.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

volatile unsigned char\* RBR = (unsigned char\*) 0xFCC0; // data Recieve BuffeR

volatile unsigned char\* THR = (unsigned char\*) 0xFCC0; // Transmit Holding Register

volatile unsigned char\* LSR = (unsigned char\*) 0xFCC5; // the UART status register

1. kbhit(void): – will examine the RDA status bit and return a true (non-zero value) if RDA is true, otherwise it will return a false (0 value).

unsigned char myKbhit()

{

// return 1 if kbhit detected

return ( LSR & 0x01 ); // data ready flag in bit 0

}

1. getchar(void): – will read one character from the serial port and return it.c.

unsigned char myGetChar()

{

// return the char data sitting in the RBR

return \*RBR;

}

1. putchar(char vname): – will write the character vname to the serial port as soon as TBE is true.

void myPutChar( unsigned char vname )

{

// wait for previous data to be transmitted (wait for status flag)

while( (LSR & 0x40) == 0 ) // TBE flag in bit 6

{};

// once THR is known to be empty, write the data to the THR to be output

\*THR = vname;

// no return - void

}

1. Write an Arduino C language function printstr(\*unsigned char str0) which will print out a NULL terminated ASCII string using the serial I/O functions defined in Question 1 above. The input pointer str0 points to the first element of the string to be printed.

void myPrintString( unsigned char\* str0 )

{

// variables

volatile int ndx = 0;

// find the length

while( str0[ndx] != '\0' )

{

// output the character

putChar( str0[ndx] );

// increment the index

ndx++;

}

// no return - void

}

1. Write an ARDUINO C function called strlen() that determines the length of a null-terminated ASCII string. Pass a 16-bit address pointer to the first element of the string to the function. Return the length, excluding the null byte.

int myStrLen( unsigned char\* str0 )

{

// variables

volatile int ndx = 0;

// find the length

while( str0[ndx] != '\0' )

{

// count this character towards the length

ndx++;

}

// return the length

return length;

}

1. Write an ARDUINO C function to create a fixed length destination string by copying a source string and truncating or padding with spaces as needed. Pass three variables to the function: 1) the length of the output string, 2) a 16-bit address pointer to the first element of the source string, and 3) a 16-bit address pointer to the first element of the destination string. The function does not return any value.

void myStrCpy( int destLen, unsigned char\* source, unsigned char\* destination )

{

// variables

volatile int ndx = 0;

// copy the characters over one at a time

// this operation may store a truncated version of source

while( ( source[ndx] != '\0' ) && ( ndx < (destLen - 1) ) )

{

destination[ndx] = sorce[ndx];

}

// if necessary, pad with spaces

while( ndx < (destLen - 1) )

{

// store spaces

destination[ndx] = ' ';

}

// terminate the destination string with '\0'

destination[destLen - 1] = '\0';

// no return - void

}

1. Write an ARDUINO C function to compare two character strings and determine whether one is less than, greater than, or equal to the other. Assume the strings are null-terminated. Pass a 16-bit address pointer to the first element of the first string, and a 16-bit address pointer to the first element of the second string. Return -1 if the first is less than the second, return 0 if the strings are equivalent, and return + 1 if the first is greater than the second.

int myStrCmp( unsigned char\* first, unsigned char\* second )

{

// variables

volatile int result = 0;

volatile int ndx = 0;

// find the first unequal character

while( ( first[ndx] == second[ndx] ) &&

( ( first[ndx] != '\0' ) &&

( second[ndx] != '\0' ) ) )

{

// increment the index

ndx++

}

// test the first differing character

// case: the first string is greater than the second

if( first[ndx] > second[ndx] )

{

// result is 1

result = 1;

}

// case: the first string is less than the second

else if( first[ndx] > second[ndx] )

{

// result is -1

result = -1;

}

// case: the strings are the same

// loop broke at NULL terminators, if first two tests fail, both

// are equivalent; result is already zero

// return the result

return result;

}