Zachery Takkesh
Jason Chang
May 18, 2018

Final Project: Serial Port Calculator

Introduction

The serial port calculator was designed for students to understand the interaction between the PC and the 8051 Microcontroller through a terminal emulator such as the Hyperterminal program. The purpose of this lab was for students to apply their knowledge regarding serial port communication and programming methodologies to transfer and receive data back and forth between the PC peripheral (keyboard) and the 8051 Microcontroller using C code development. Hyperterminal acts as a PC so the user can input different integers and arithmetic operations and the accurate result will be returned. The purpose of our project was to create a serial port calculator that accurately receives two, three-digit integers inputs along with a minimum of four arithmetic operations and outputs results via HyperTerminal.

Operation

The program behaves similarly to that of a calculator and allows users to input two three-digit integers along with five different operations; add, sub, multiply, divide, and exponential. Hyperterminal takes in the inputted characters and transfers the data to the 8051 Microcontroller. The result will be calculated by the program built-in 8051 and returned to be displayed on the Hyperterminal.

In detail, there are a total of seven input slots for the calculator. The first three inputs represent the first integer of the operations; if the number is one digit, the user must input a “0” on all places that have no numerical value, hundreds, tens or ones respectively**.** The fourth slot of the input is for the four basic arithmetic operators such as “ **+, -, *, /** ”. However, our team decided to implement a fifth operator, power, being represented by the carat key, “**^**” . The last three inputs are for the second operand of the function with the same rules applied**.** Once the function is inputed, the program will immediately output the result of the operation to HyperTerminal and instantly return to a new line to receive the next set of inputs. The visuals being displayed on the Hyperterminal is a blinking carriage line that awaits user input, and everything the user inputs on the keyboard will be displayed immediately on the HyperTerminal interface, however the command will only execute correctly and instantly if the direction for utilizing the calculator is followed, three bits for operand, followed by the fourth bit for the operator and the last three bits also for the second operand.

Serial Communication Theory and Explanation

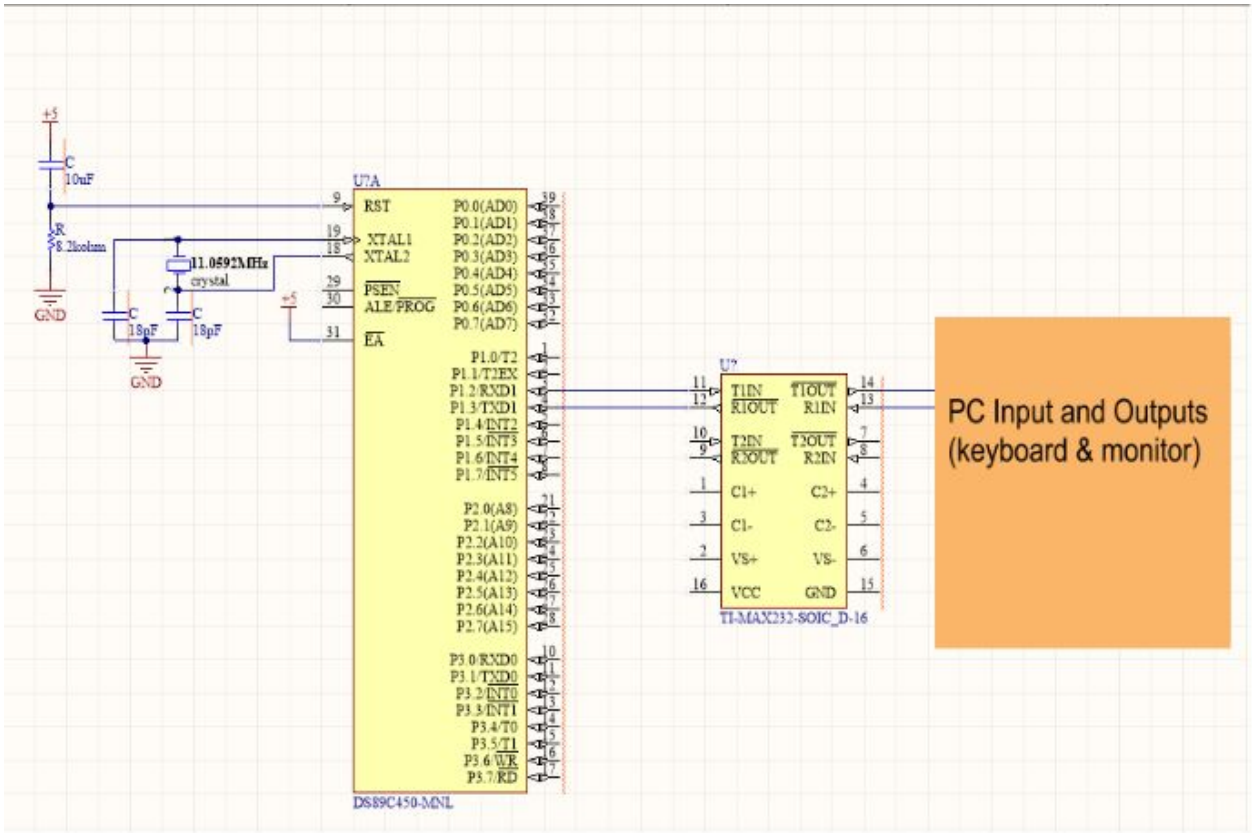
The 8-bit auto reload counter is used to receive the input operands of the calculation. The value inside the TH(timer high) register is held by the 8-bit auto reload and allows the counter to overflow and reset to the value inside the TH register without the flag having to be reset. There are two main forms of data communication, parallel, and serial, with parallel, data is transferred multiple bits in one tick, while serial is one bit at a time. However, serial communication is much faster than parallel communication due to the fact that in parallel communication, a lot of electronic noise is generated, thus slowing down the transfer speed causing parallel communication to be actually slower than serial communication when doing simple processing.

Serial communication that utilizes the 8-bit auto reload register is an asynchronous transfer technique which requires a start indicator bit followed by the 8-bit data and in addition, a stop indicator bit. The transfer feed of this technique will vary depending on the baud rate (bits per second) in which the rate in this case is defined by how many bits can be transferred per second.

The RS232 is required to translate the binary value of 0 and 1 to voltage high and low due to the fact that the 8051 only receives data in forms of voltage in order to transfer data into the 8051. The half duplex bus is used by the usb (universal serial bus) as a means of transferring and receiving data, thus allowing the RS232 to convert logic “1” to +25v and logic “0” to -25v.

Schematic

Made Via **CircuitMaker**



Results and Conclusion

This program involved the usage of a terminal connected to an 8051 through a computer, the purpose being the development of a serial port calculator that translates user inputs and performs arithmetic calculations by sending ASCII values from the computer via keyboard to the 8051 and converting it to numerical values after receiving it through the terminal and displaying it on the terminal in numerical value rather than the ASCII value. The arithmetic operation is decided upon receiving the fourth bit and the operator is decided based on conditional statements, such as, multiplication, subtraction, addition, and division as the user is required to enter seven bits to accurately utilize the program. The first and the last three bits are the numerical values or operands in hundreds, tens, and ones, respectively with the fourth bit being the operator. All these instructions were developed through C code and compiled into a hex type object in order to be programmed into the 8051 via Hyperterminal.

Through this project, numerous problems aroused, however the most prominent issues revolved around; negative values, decimal values, and variable size types which were not able to hold extremely large values resulting from exponential operations as was implemented in our project. For the extremely large operations, this was the case because the different data types in C code were only able to hold a limited value, therefore, by using the power function, the larger the value of the result, the more limited we were on the data types we were able to use. In the end, the only data type that we were able to use was the **long** type, which was able to store our exponential calculations without an overflow error.

Although sufficient time was given to complete this project in its simplest form in a timely manner, we were not able to address all the issues that arose during the time constraint. Some of these included negative values, decimal values, and order of operations in which we were not able to address adequately. If we were given more time, we would want to address first and foremost, the order of operations because as of now, the program executes instructions in the order that it is given, meaning that it would add before multiply if the add was inputted before the multiply which is not arithmetically correct. Furthermore, we would attempt to implement decimal and negative values which would require a decent amount of time as decimal values require an entire new set of variables to hold each bit for each operand and the negative also.

Software Description

Implementing a function to both transfer and receive bits decluttered the main function and continuous while loop, while reducing the length of the source code to more than half of the original length. (over six-hundred lines).

```c
//Jason Chang    015610151
//Zachery Takkesh 015656509
#include <ds89c4xx.h>
#include <math.h>

// Receive function
// This function receives the ASCII code via HyperTerminal
// for any character on keyboard
// returns character to main
unsigned char receive(void){

    while (RI_0 == 0);
    //clear receive interrupt flag
    RI_0 = 0;
    return SBUF0;
}

// Transfer function
// This function takes in a variable from
// main and outputs via HyperTerminal
void transfer(unsigned char send){

  SBUF0 = send;
  while(TI_0 == 0);
  // clear transfer interrupt flag
  TI_0 = 0;

}

// Main function
void main (void){

// Initialize Variables for each bit input
unsigned char op1, op2, op3, operator, op4, op5, op6;

// Variable for left and right hand sign of operation
unsigned char operand1, operand2;

// Variable for each bit place holder
unsigned int  bit_9, bit_8, bit_7, bit_6, bit_5;
unsigned int  bit_4, bit_3, bit_2, bit_1 ;

// long data type required for power operation
long  result, temp;

  // Initialize timer mode
  TMOD  = 0x20; // 8-bit auto reload mode
  TH1   = 0xFD; // baud rate for 9600
  SCON0 = 0x50; // 8-bit data, 1 stop bit, 1 start bit
  TR1   = 1;    // start timer

  // Continuous  loop
  while (1)
  {

  // Clearing all of the bits for a new  set of an arithmetic operation
   bit_9    = 0;
   bit_8    = 0;
   bit_7    = 0;
   bit_6    = 0;
   bit_5    = 0;
   bit_4    = 0;
   bit_3    = 0;
   bit_2    = 0;
   bit_1    = 0;
   result   = 0;
   temp     = 0;

    // 1st bit receive and transfer
    op1 = receive();
    transfer(op1);

```

```
73        // convert from ASCII to number
74        op1 = op1 - 48;
75
76        // 2nd bit receive and transfer
77        op2 = receive();
78        transfer(op2);
79
80        // convert from ASCII to number
81        op2 = op2 - 48;
82
83        // 3rd bit receive and transfer
84        op3 = receive();
85        transfer(op3);
86
87        // convert from ASCII to number
88        op3 = op3 - 48;
89
90        // combine the first three inputs
91        operand1 =(op1*100)+ (op2*10) + op3;
92
93
94        // operator receive and transfer (4th bit)
95        operator = receive();
96        transfer(operator);
97
98        //4th bit receive and transfer
99        op4 = receive();
100       transfer(op4);
101
102       //convert from ASCII to number
103       op4 = op4 - 48;
104
105       //5th bit receive and transfer
106       op5 = receive();
107       transfer(op5);
108
109       //convert from ASCII to number
110       op5 = op5 - 48;
111
112       //6th bit receive and transfer
113       op6 = receive();
114       transfer(op6);
115
116       //convert from ASCII to number
117       op6 = op6 - 48;
118
119       //combine last three inputs
120       operand2 = (op4*100) + (op5*10) + op6;
121
122
123
124     // Conditional evalution statements
125     // for arithmetic operations
126
127     // add operation
128     if(operator == '+')
129        result = operand1 + operand2;
130
131     // subtract operation
132     else if(operator == '-')
133        result = operand1 - operand2;
134
135     // multiply operation
136     else if(operator == '*')
137        result = operand1 * operand2;
138
139     // divide operation
140     else if(operator == '/' )
141        result = operand1 / operand2;
142
143     // power operation
144     else if(operator == '^')
```

```
145         result =  pow(operand1,operand2);
146
147     // default condition
148      else
149        result = 'E';
150
151
152     // Set result into temp to get the
153     // accurate bit in each number location
154     temp = result;
155
156       // Value at hundred-millionth spot
157       while (temp >= 100000000)
158       {
159         temp = temp - 100000000;
160         bit_9 = bit_9 + 1;
161       }
162       // Value at ten-millionth spot
163       while (temp >= 10000000)
164       {
165         temp = temp  - 10000000;
166         bit_8 = bit_8 + 1;
167       }
168
169       // Value at millionth spot
170       while (temp >= 1000000)
171       {
172         temp = temp  - 1000000;
173         bit_7 = bit_7 + 1;
174       }
175
176       // Value at hundred-thousandth spot
177       while (temp >= 100000)
178       {
179         temp = temp  - 100000;
180         bit_6 = bit_6 + 1;
181       }
182
183       // Value at ten-thousandth spot
184       while (temp >= 10000)
185       {
186         temp = temp  - 10000;
187         bit_5 = bit_5 + 1;
188       }
189
190       // Value at thousandth spot
191       while (temp >= 1000)
192       {
193         temp = temp  - 1000;
194         bit_4 = bit_4 + 1;
195       }
196
197       // Value at hundredth spot
198       while (temp >= 100)
199       {
200         temp = temp - 100;
201         bit_3 = bit_3 + 1;
202       }
203
204       // Value at tenth spot
205       while (temp >= 10)
206       {
207         temp = temp - 10;
208         bit_2 = bit_2 + 1;
209       }
210       // Value at ones spot
211       while (temp >= 1)
212       {
213         temp = temp - 1 ;
214         bit_1 = bit_1 + 1;
215       }
216
```

```
217       // Convert each value location back to ASCII
218       bit_9  = bit_9 + 48;
219       bit_8  = bit_8 + 48;
220       bit_7  = bit_7 + 48;
221       bit_6  = bit_6 + 48;
222       bit_5  = bit_5 + 48;
223       bit_4  = bit_4 + 48;
224       bit_3  = bit_3 + 48;
225       bit_2  = bit_2 + 48;
226       bit_1  = bit_1 + 48;
227
228      // for visual purposes
229      // being sent back to terminal
230      // new line
231      transfer('\n');
232      // carriage return
233      transfer('\r');
234      // equal sign
235      transfer(61);
236      // space sign
237      transfer(32);
238
239      // transfer each bit back to terminal
240      // bit_9 being the most significant bit(hundred-millionth)
241      // bit_1 being the least signicant bit(ones)
242      transfer(bit_9);
243      transfer(bit_8);
244      transfer(bit_7);
245      transfer(bit_6);
246      transfer(bit_5);
247      transfer(bit_4);
248      transfer(bit_3);
249      transfer(bit_2);
250      transfer(bit_1);
251
252      // new line transfer for visual purposes
253      transfer('\n');
254      // carriage return to start at beginning of line
255      transfer('\r');
256      // visual pointer for terminal
257      transfer('>');
258
259    } // end of continous loop
260
261  } // end of main function
```