Establishing communication between Qt program and DE2-115 FPGA

Howard Edwards, Michael Micros, Jonathon Rigney, Megan Rowland

Abstract(Jonathon Rigney) —
This report is only part of a series of papers aimed at designing and implementing a system to perform real time video processing using a Field Programmable Gate Array(FPGA) controlled by a Graphical User Interface(GUI). This paper deals specifically with establishing communication between Qt and the DE2-115 FPGA. The slider value from a Qt GUI program was succesfully transferred to the FPGA via RS232 serial communication. The design makes use of the QSerialPort Class on the Qt side, in order to be able to send the slider value one character at a time over the RS232 connection. On the FPGA side, a NIOS II (Basic Computer) processor is programmed onto the board making additional use of a C program. The C program receives the characters sent from the Qt program, and displays the values on 2 7-segment displays on the DE2-115 FPGA.

Index Terms — **FPGA, 8-bit counter, Seven Segment Display**

I. Introduction(Howard Edwards)

This report documents the goal to establish communication between a graphical user-interface, GUI, and a field-programmable gate array, FPGA, board. This goal was a fraction of an integrated system that performs the processing of real-time video through an user-interface on a computer and the use of FPGAs. The GUI was developed with Qt cross-platform application, and the FPGA board was the Altera DE2- 115. Both the Qt GUI creation and the Altera DE2-115 familiarization was discussed in another report by Edwards, Micros et al.

Communication was established between the GUI and FPGA through RS232. The GUI sends data to the computers RS232 port which is then receive by the DE2-115 boards via its RS232 interface. In order for the board to receive this data, a basic computer called the DE2 Basic Computer was programmed onto the FPGA. One of its main component is the Altera Nios II processor. In order to assemble and compile Nios II programs, written in C programming language or assembly, the Altera Monitor program was utilized.

The program was written in C programming language to program Nios II. The program allows for the value of the brightness slider in the GUI to be received from the RS232 interface and displayed onto the boards 7-segment display. Before this was accomplished, the RS232 library was implemented onto the Qt application to send the sliders value to the computer's port to be received. With both Qt and Nios II written programs running in unison, communication between GUI and the FPGA board operated as desired.

II. Design (Michael Micros)

A. Summary of Design– The overall design aims at achieving communication between a Qt program and a DE2-115 Altera FPGA. More specifically, we attempt to send a slider value used in a GUI over an RS232 port, in order to display the value on 2 7-segment LEDs of the FPGA. To achieve this minor modification needed to be made to pre-existing code from a previous paper by Edwards et al that implements a GUI that allows the user to adjust the brightness and contrast of an image. Additionally, code was written to read from the RS232 port of the FPGA and display the received digits.

B. Detail Description

Qt Design: In order to establish communication through the QT GUI and the RS232 serial port, the QSerialPort Class, provided by QT, is utilized. The RS232 serial port on the computer utilized for this project is named com1 and is configured as com1 in the QT program. For the RS232 serial port, the correct baud rate, data bits, parity, and stop bits need to be configured in the mainwindow.cpp file of the QT program to transmit data to the Altera board. The RS232 serial port is configured for a baud rate of 115,200, 8-bit data, odd parity, and one stop bit. These settings need to be mirrored in the QT program for correct transmission of data. If any of the values are not configured correctly, the data will be garbled on the receiving end. The settings are added to the initialization of the MainWindow function in mainwindow.cpp file of the QT program to open and configure and connection to the RS232 serial port. The corresponding code can be seen in Fig. 1.

After the connection to the RS232 serial port is opened and configured, the value obtained by the change in the brightness slider is transmitted to the Altera board. In order to do this, code is added to the function called

when the brightness slider is moved to the left or right, on_brightnessSlider_valueChanged(int value). The value transmitted needs to be in the form of a const char. Also, the value transmitted needs to be two digits, so a zero is added to all of the single digit values before converting. To add the 0 digit we use the Qstring.insert() function that takes as parameters the position to which to add the QChar, and the QChar to be inserted. This will allow the receiving end to display the value on the 7-segment displays correctly. The value from the brightness slider is converted from an integer to a const char and transmitted through the RS232 serial port. The corresponding code can be seen in Fig. 2.

```
QSerialPort serial;

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    serial.open(QIODevice::ReadWrite);
    serial.setPortName("com1");
    serial.setBaudRate(QSerialPort::Baud115200);
    serial.setDataBits(QSerialPort::Data8);
    serial.setParity(QSerialPort::OddParity);
    serial.setStopBits(QSerialPort::OneStop);
    serial.setFlowControl(QSerialPort::NoFlowControl);
    if(serial.open(QIODevice::ReadWrite))
        serial.write("ok*");
    else
        cout << "Error" << endl;

}
```

Figure 1: C++ code to open and configure the RS232 serial port

```
QString tempString = 0;

if (value < 10){
    tempString = QString("%1").arg(value);
    tempString.insert(0, '0');
}
else
    tempString  = QString("%1").arg(value);

serial.write(tempString.toStdString().c_str());
```

Figure 2: C++ code to translate and send a value through the RS232 serial port

Nios II (C) program:
The Nios II design consists of a project created in the Altera Monitor Program that imports a pre written C program and loads it onto the FPGA board. The C code runs in an infinite while loop and performs 3 basic tasks. 1) It listens on the RS232 port for incoming data. 2) When data is received, it outputs it to the console (for testing and confirmation). 3)Sets the appropriate values to the LEDs on the board.

The RS232 port is referred to in the Nios II by the address of the port which is 0x10001010. In the while loop, we read bit 15 of the value located in the address. This bit is called RVALID and represents whether data is ready to be accessed. The incoming data(characters) are stored in a FIFO stack and accessed one at a time by reading the 0-7 bits of the aforementioned address.

Once a digit is read, it is displayed on the Altera Monitor Program console by writing to the 0-7 byte of the JTAG UART port located in address 0x10001000. The final task required was displaying the received value (2 characters) on the 7-segment LEDs. In order to be able to distinguish which bit was the first bit and which was the second, an int variable was used to keep track of this. The variable would toggle between 1 and 2. In each cycle through the while loop, the value of the variable would be checked in an if statement, and the flow of the program would shift to the appropriate part of code. The first (most significant digit) character is displayed on LED HEX1, which is done by assigning the digit to the value of address 0x10000021. The digit is copied to an int variable dig1 for reasons explained later.

When we receive the second digit, the code enters the appropriate part of the program where the LED HEX0 is given the appropriate value, by assigning the proper 7 bit vector to address 0x10000020. Because LED HEX0 and HEX1 are adjacent byte values, during testing we noticed that when writing to HEX0, HEX1 would be cleared (we assume it was filled with 0s appended to HEX0 because we are not writing an 8-bit value but a 32 bit value probably). That is the purpose on variable dig1 introduced earlier. After writing to HEX0 we rewrite HEX1.

The C code that performs the previous tasks is included in the Appendix.

III. Evaluation (Megan Rowland)

The implemented design performed as designed. When running the GUI from QT, a user will select an image. When adjusting the brightness of that image, the value from the brightness slider will be transmitted through the RS232 serial port to the Altera board via the DE2-115 basic computer. When the DE2-115 basic computer receives the value it will then write that value to the correct 7-segment display. The most significant digit will be written to the left 7-segment display and the least significant digit will be written to the right 7-segment display. If the value from the brightness slider is a single digit, the value will be transmitted as two digits with a leading zero and the value viewed on the 7-segment display will be a zero written to the left 7-segment display and the single digit written to the right 7-segment display. Every time the brightness slider is moved, the value on the 7-segment display updates:

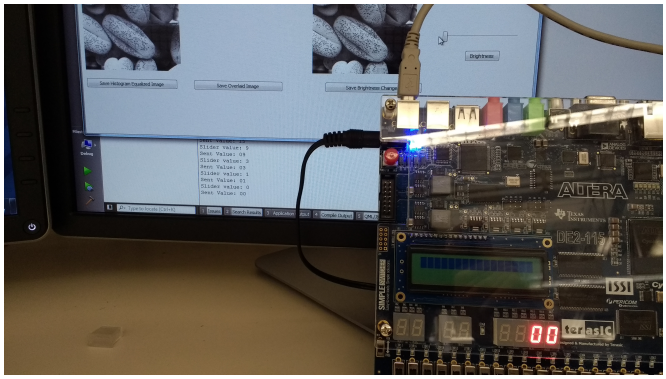

Figure 3: Brightness slider moved down to 0

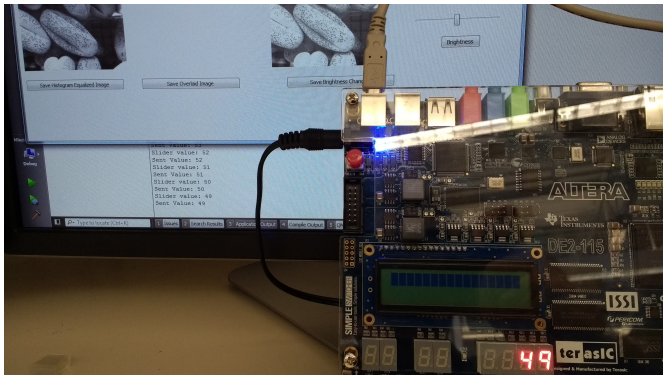

Figure 4: Brightness slider moved to 49

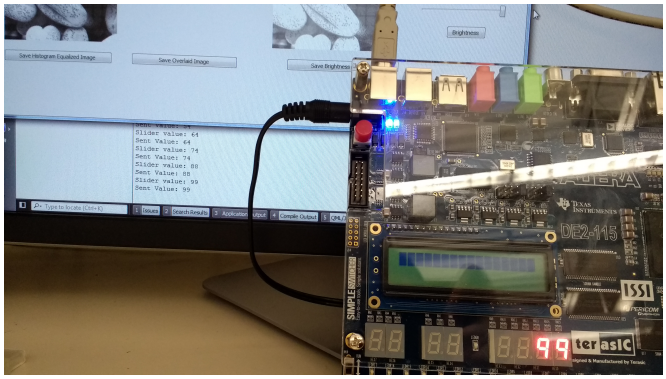

Figure 5: Brightness slider moved up to 99

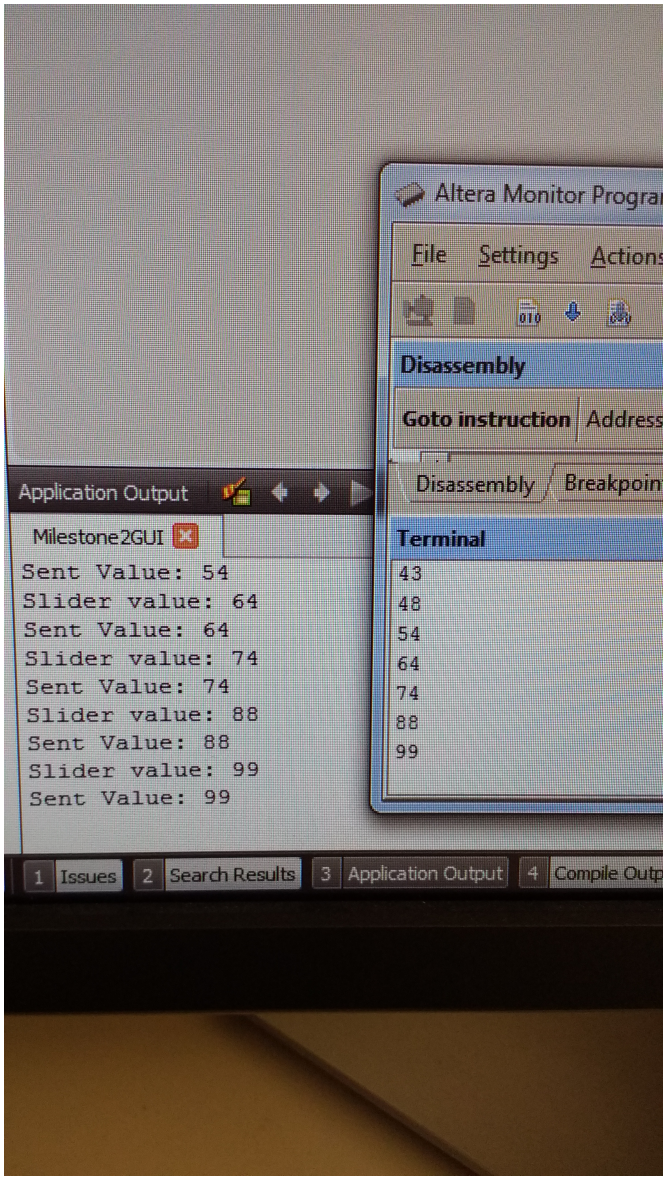

Figure 6: Value sent by the QT program (left) and recieved by the Altera board (right) through the RS232 serial port

IV. Discussion (Megan Rowland)

As was demonstrated in the Evaluation, the implemented design performed as intended. It is the belief of everyone involved in this research that there is not much room for improvement to the design. There was a big learning curve during the troubleshooting of the design, because it was difficult to identify if the problems arising occurred on the Qt side of the design or the Nios II side. One problem we had to overcome, was an unexpected faulty FPGA, which set the schedule of implementation back a bit. Additionally, during testing we noticed that the FPGA was receiving extra junk characters which was due to an incorrect setting of the parity bits on the Qt side.

IV. Appendix

```c
void put_jtag(volatile int *, char ); // function prototype

int main(void)
{
    /* Declare volatile pointers to I/O registers (volatile means that IO load and store
    instructions (e.g., ldwio, stwio) will be used to access these pointer locations) */
    volatile int * RS232_ptr = (int *) 0x10001010; // RS232 address
    volatile int * JTAG_UART_ptr = (int *) 0x10001000; // JTAG UART address

    volatile int * HEX0_ptr = (int *) 0x10000020; // HEX0 address
    volatile int * HEX1_ptr = (int *) 0x10000021; // HEX1 address

    int data;

    int count = 1;
    int dig1 = 0;
    /*
    int i;
    char text_string[ ] = "\nJTAG UART example code\n> \0";

    for (i = 0; text_string[i] != 0; ++i) // print a text string
        put_jtag (JTAG_UART_ptr, text_string[i] );
    */

    /* read and echo characters */
    while(1)
    {
        data = *(RS232_ptr); // read the JTAG_UART Data register
        if (data & 0x00008000) // check RVALID to see if there is new data
        {
```

```
31              if(count == 1)
32              {
33                  /*
34                  int i;
35                  char text_string[ ] = "\nfirst digit\n> \0";
36
37                  for (i = 0; text_string[i] != 0; ++i) // print a text string
38                      put_jtag (JTAG_UART_ptr, text_string[i] );
39                  */
40                  /* Display on the 7-segment display Most Significant Digit */
41                  if ((char) data == '0')
42                      *(HEX1_ptr) = 0b0111111; // 0 = 0b0111111
43                  else if ((char) data == '1')
44                      *(HEX1_ptr) = 0b0000110; // 1 = 0b0000110
45                  else if ((char) data == '2')
46                      *(HEX1_ptr) = 0b1011011; // 2 = 0b1011011
47                  else if ((char) data == '3')
48                      *(HEX1_ptr) = 0b1001111; // 3 = 0b1001111
49                  else if ((char) data == '4')
50                      *(HEX1_ptr) = 0b1100110; // 4 = 0b1100110
51                  else if ((char) data == '5')
52                      *(HEX1_ptr) = 0b1101101; // 5 = 0b1101101
53                  else if ((char) data == '6')
54                      *(HEX1_ptr) = 0b1111101; // 6 = 0b1111101
55                  else if ((char) data == '7')
56                      *(HEX1_ptr) = 0b0000111; // 7 = 0b0000111
57                  else if ((char) data == '8')
58                      *(HEX1_ptr) = 0b1111111; // 8 = 0b1111111
59                  else if ((char) data == '9')
60                      *(HEX1_ptr) = 0b1100111; // 9 = 0b1100111
```

```
61
62                  dig1 = *(HEX1_ptr);
63                  /* echo the character */
64                  put_jtag (JTAG_UART_ptr, (char) data & 0xFF );
65
66                  count = 0;
67              }
68              else
69              {
70
71                  /* Display on the 7-segment display Least Significant Digit */
72                  if ((char) data == '0')
73                      *(HEX0_ptr) = 0b1111111110111111; // 0 = 0b0111111
74                  else if ((char) data == '1')
75                      *(HEX0_ptr) = 0b1111111110000110; // 1 = 0b0000110
76                  else if ((char) data == '2')
77                      *(HEX0_ptr) = 0b1111111111011011; // 2 = 0b1011011
78                  else if ((char) data == '3')
79                      *(HEX0_ptr) = 0b1111111111001111; // 3 = 0b1001111
80                  else if ((char) data == '4')
81                      *(HEX0_ptr) = 0b1111111111100110; // 4 = 0b1100110
82                  else if ((char) data == '5')
83                      *(HEX0_ptr) = 0b1111111111101101; // 5 = 0b1101101
84                  else if ((char) data == '6')
85                      *(HEX0_ptr) = 0b1111111111111101; // 6 = 0b1111101
86                  else if ((char) data == '7')
87                      *(HEX0_ptr) = 0b1111111110000111; // 7 = 0b0000111
88                  else if ((char) data == '8')
89                      *(HEX0_ptr) = 0b1111111111111111; // 8 = 0b1111111
90                  else if ((char) data == '9')
```

```
91                      *(HEX0_ptr) = 0b111111111100111; // 9 = 0b1100111
92
93                  *(HEX1_ptr) = dig1;
94
95                  /* echo the character */
96                  put_jtag (JTAG_UART_ptr, (char) data & 0xFF );
97                  *(JTAG_UART_ptr) = '\n';
98                  count = 1;
99              }
100         }
101     }
102 }
103 /*****************************************************************************
104 * Subroutine to send a character to the JTAG UART
105 *****************************************************************************/
106 void put_jtag( volatile int * JTAG_UART_ptr, char c )
107 {
108     int control;
109     control = *(JTAG_UART_ptr + 1); // read the JTAG_UART Control register
110     if (control & 0xFFFF0000) // if space, then echo character, else ignore
111     {
112         *(JTAG_UART_ptr) = c;
113     }
114
115     //*(JTAG_UART_ptr) = '\n';
116 }
```