Field Programmable Gate Array Implementation of a Motherboard for Data Communications and Networking Protocols

Rosula S. J. Reyes, Ph.D., Carlos M. Oppus, Jose Claro S. Monje, Noel S. Patron, Raphael A. Gonzales, Oscar Idaño and Mark Glenn Retirado

***Abstract*—**Reconfigurable devices, such as the field programmable gate arrays (FPGA), have provided electrical, electronics and computer engineers with a versatile and cost-effective platform for designing circuits, developing devices and implementing electronic, communications, computer and other related systems. Presented in this paper is the use of FPGA in the development of a motherboard to introduce the concepts of data and network communications protocol through different interfaces. Some of the protocols implemented are VGA, PS/2, serial communications and parallel communications. Since the motherboard is FPGA-based, it can be reconfigured to perform other protocols making it open to a lot of possibilities.

***Keywords*—**communications, data, field programmable gate array, motherboard, network, protocols, trainers.

I. Introduction

DATA handling and network communications has risen in importance in this time and age[1]. Almost all electronic equipment has some means of storing, manipulating, and communicating data with users and with other devices. With computer networks at the core of modern communication, different networks, as well as different technologies within them, need to be able to connect and understand data going to and through them. Most of this processes goes unnoticed by the regular user, even most of today's engineers know only the theory on most of this basic communication protocols and focus on the higher level logic of systems today. A basic understanding of these concepts is required for efficient high level programming. However, systems today that use these said communication protocols, such as personal computers, are already designed for other purposes. These same systems make these protocols transparent to their users, and using them as training tools for these basic protocols will be costly. Thus, a low cost basic protocol motherboard will be beneficial especially for laboratories since low costs can mean more units and more time for hands-on exercises. Moreover, these motherboards will be FPGA-based and can be expanded with daughter boards for other purposes as the need arises.

II. Using FPGAs

Ten years ago, the thought of a single-chip design fulfilling all of the needs of a certain project was unimaginable. However, at present, the use of FPGAs has been a viable option.

An FPGA or field programmable gate array is a semiconductor device containing programmable logic components and interconnects[2]. These logic components can be programmed to duplicate the functionality of digital circuits. To configure the FPGA, a circuit diagram or source code is written using a hardware description language (HDL) that describes the functionality of the FPGA. Thus, there is no need to think about the gate level implementation of the circuit.

With its wide-range of capabilities and ease of use, it has been able to tackle nearly any type of application imaginable. One can implement a 16-bit microprocessor core inside an FPGA [3]. FPGA's are now used in teaching microprocessor and digital design courses [4]. Moreover, FPGAs are used to design products, to test products or even using FPGA-based hardware for controlling and manufacturing products. .

Manuscript received March 22, 2011. This work was funded by the Philippine Department of Science and Technology under the Engineering Research and Technology Program and the Ateneo de Manila University.

R. S.J. Reyes is with the Department of Electronics, Computer, and Communications Engineering, Ateneo de Manila University, Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: rsjreyes@ateneo.edu) and Blue Chip Designs, Inc, Unit 208 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines

C. M. Oppus is with the Department of Electronics, Computer, and Communications Engineering, Ateneo de Manila University, Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: coppus@ateneo.edu) and Blue Chip Designs, Inc, Unit 208 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines

J. C. Monje is with the Department of Electronics, Computer, and Communications Engineering, Ateneo de Manila University, Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: jcmonje@ateneo.edu) and Blue Chip Designs, Inc, Unit 208 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines

N. S. Patron is taking up a Master's degree in Electronics Engineering in Ateneo de Manila University and is with Blue Chip Designs, Inc, Unit 208 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: npatron@bcdph.com)

R. A. Gonzales is taking up a Master's degree in Electronics Engineering in Ateneo de Manila University and is with Blue Chip Designs, Inc, Unit 208 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: rgonzales@bcdph.com).

O. Idaño is taking up a Master's degree in Electronics Engineering in Ateneo de Manila University and is with Bughaw Electronics Solutions and Technologies , Unit 203 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: oidano@bestinc.ph).

M. G. Retirado is with Bughaw Electronics Solutions and Technologies , Unit 203 Xanland Place 323 Katipunan Avenue, Loyola Heights, Quezon City, 1108 Philippines (e-mail: mgretirado@bestinc.ph).

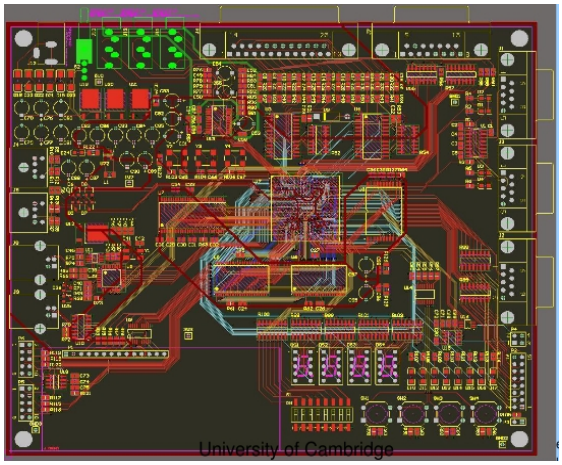

Fig. 1 Motherboard PCD Layout Diagram

Because of this flexibility, a motherboard to be used as a protocol trainer was designed using a FPGA by Blue Chip Designs (BCD).

Using a FPGA as a Central Processing Unit (CPU, various level converters and appropriate ports used by different communications standards, a communications protocol trainer can be developed.

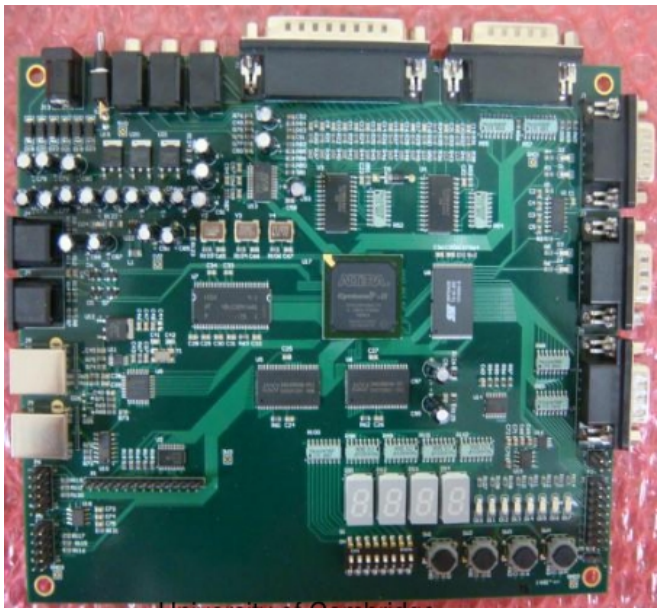
Fig. 2 FPGA Motherboard

For instance, connecting a DE-15 port and several lines of HDL, a very cheap 4096-color VGA controller is also created[5]. Another is to use the USB transceiver chip with appropriate DAC/ADC circuits and lines of HDL code, USB communication can be achieved. In addition, internal clocks can also be generated by the FPGA using phase-locked loop (PLL) circuits and most FPGAs today contain 1 to 8 PLLs[6]. Thus, only basic knowledge, coding skills, and good execution is needed to come up with an innovation.

A. VGA Display

This trainer is able to generate the needed signals for a VGA display monitor. There are five signals needed to display an image using the VGA port. These are the red signal, green signal, blue signal, horizontal synchronization, and vertical synchronization.

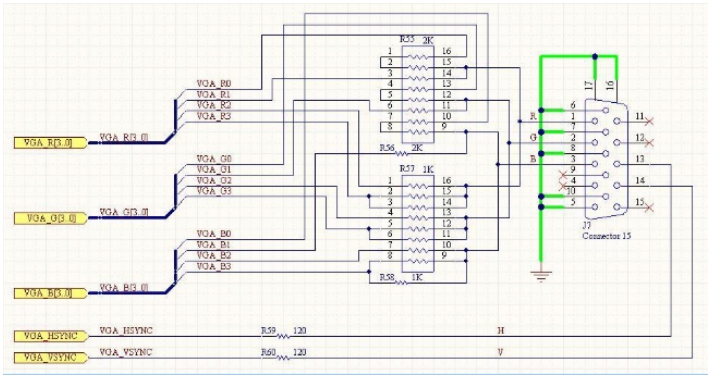

Fig. 3 VGA Port Schematic Diagram

Data for the red, green, and blue (RGB) signals corresponds to the pixel data which represents the color and intensity of the pixel. This data is saved in a memory module. Then, data is retrieved from the memory module, formatted into lines of pixels, and sent to the display device with the appropriate horizontal and vertical synchronization pulses.

In addition, the pixel data used to form the image is accessed from the RAM using horizontal and vertical counters that are also used to generate the synchronization pulses for VGA output.

Table I. Sample text map for the character 'A'

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0
4	0	0	1	1	1	0	0	0
5	0	1	1	0	1	1	0	0
6	1	1	0	0	0	1	1	0
7	1	1	0	0	0	1	1	0
8	1	1	1	1	1	1	1	0
9	1	1	0	0	0	1	1	0
10	1	1	0	0	0	1	1	0
11	1	1	0	0	0	1	1	0
12	1	1	0	0	0	1	1	0
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0

1) Text Display

Text is generated using a pixel map of individual characters. Each character is mapped on a grid with a size of 8 dots wide by 16 dots high. This map is used to create the pixel data of the needed character. Retrieval of the map is done by issuing the appropriate ASCII code for the character, as well as the coordinates of the individual pixel being generated. Table 1 shows a sample map for the character 'A.' 1 designates a pixel to be active on the display while a 0 otherwise.

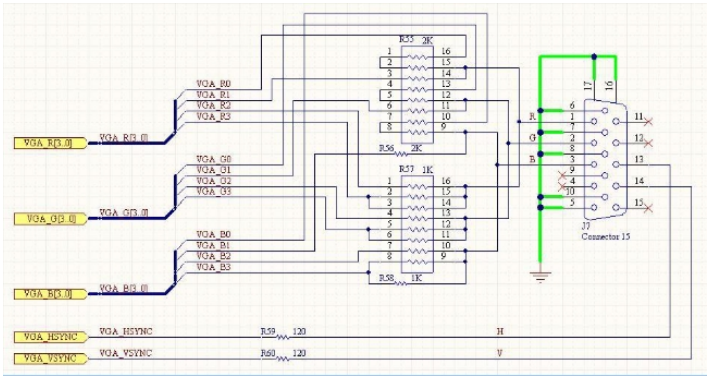

Fig. 4 UART Schematic Diagram

B. Serial Port

Another communication method included in the motherboard is the serial communication. This is done through the inclusion of three (3) DE9 connectors.

In contrast to parallel communication, serial communication sends data one (1) bit at a time[7]. Although it may refer in general to all types of devices which transmits data serially, the most common standard associated with the term serial port is the RS232 standard.

Although all the ports are designed to be UARTs, the three ports on-board the motherboard are configured differently. The first two (2) ports are level converted to be able to communicate with a standard RS232 port on a Personal Computer. The difference between these two ports is that the first port is configured to emulate a Data Terminal Equipment (DTE) while the second is configured to emulate a Data Circuit-terminating Equipment (DCE). The third port is configured for data transfer at TTL voltage levels.

The default operation for the serial ports on this training board is an implementation of an universal asynchronous receiver and transmitter(UART). A UART is a circuit that sends parallel data over a serial line. As its name implies, this operation contains a receiver module and a transmitter module. Since the transfer is asynchronous, meaning there is no clock information data sent over the line, the transmitter and receiver should beforehand agree on a set of parameters, such as transfer rate, number of data bits, stop bits and error correction bits and such, for a successful transfer of data.

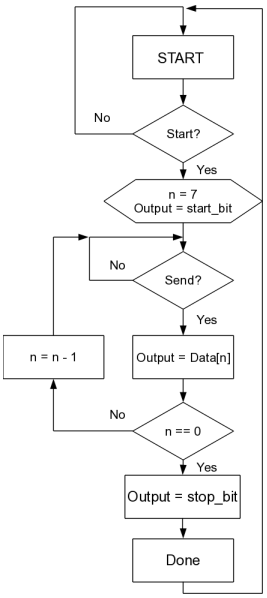

Fig. 5 UART Transmit Process

1) UART Transmitter

The UART Transmitter is basically a shift register that shifts of bits at the agreed upon transfer rate. This transmitter also needs a timing generator. Unlike the receiver, this timing can be at the same frequency as the transfer rate as this only signals when the next bit should be transmitted. Transfer is also done by a FSM, reading bits from an array one by one and transmitting them over the serial interface.

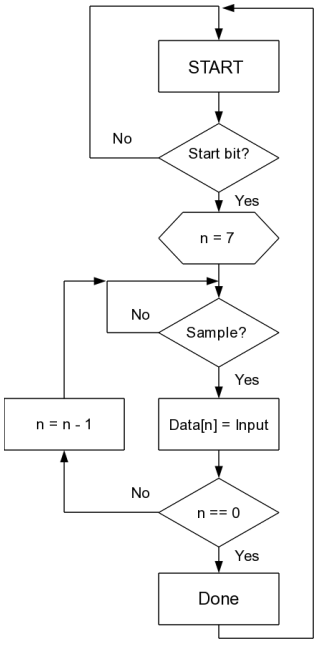

Fig. 6 UART Receive Process

2) UART Receiver

For a UART receiver, the timing of reading the serial input is essential to be able to receive the correct data. The timing is done using a baud rate generator. This generator creates a sampling clock that is a multiple of the transfer rate parameter that was agreed upon beforehand. Once a start bit is detected,

this sampling clock is used to approximate the middle point of a bit to sample it appropriately. Data bits are stored as parallel data of a fixed length. A finite-state machine(FSM) is used to sample the serial data and store it in a buffer until all bits are received.

C. Personal System/2 (PS/2)

The Personal System/2 (PS/2) connector is used for connecting a keyboard and a mouse to a compatible computer system. The PS/2 port was first used on the Personal System/2 line of IBM computers first introduced in 1986, hence the name.

The PS/2 port contains two connections for communication purposes. One connection is for bi-directional data, which is transmitted serially. The other connection is for the clock, which specifies the timing when the data from the data connection is valid and can be sampled. The information is transmitted packets of eleven (11) bits. This packet contains a start bit, eight (8) data bits, an odd parity bit, and a stop bit[8].

The PS/2 designs on keyboard and mouse interfaces are electrically similar and employ the same communication protocol. However, a given system's keyboard and mouse port may not be interchangeable since the two devices use a different set of commands.

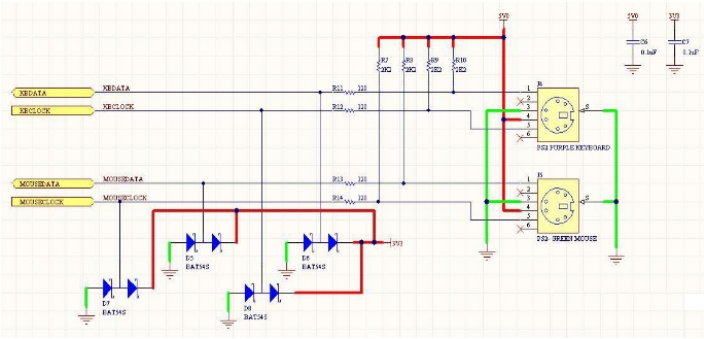

Fig. 7 PS/2 Ports Schematic Diagram

The motherboard is allocated with two ports for PS/2. It is designed for use with a keyboard and a mouse but it is reconfigurable through the FPGA.

1) PS/2 Keyboard

A keyboard consists of an array of keys. Activity done on a keyboard is monitored, or scanned, by a micro-controller. This micro-controller then sends the corresponding scan code over the PS/2 line.

The scan code consists of a sequence of codes that corresponds to the different activities that occur on the keyboard. When a key is pressed, the make code of the key is sent. If it is continued to be held down longer that 500ms, the make code is continually sent every 100ms. When the key is released, a break code is sent.

2) PS/2 Mouse

The mouse is used to detect motion on a two-dimensional surface. A mouse has an internal circuit that measures the relative distance the unit has traveled as well as the status of the buttons.

A mouse streams the data at a predetermined interval over the PS/2 port, however, upon power-up, the mouse needs to be sent commands to initialize the device. The PS/2 interface for the mouse needs to be bi-directional in this case, different from the keyboard which only sends data from the keyboard to the receiving device.

Over the predetermined interval, the mouse stores the relative movements over the x-axis and y-axis in an internal counter, as well as the three buttons, left, right and middle, states. These data are then sent through the PS/2 port in three bytes[8]. Once the data are sent, these internal counters are reset.

The x-axis movement is denoted by x8-x0 with xV signaling an overflow on the counter. The y-axis movement is contained in y8-y0 with yV as the overflow signal. Middle, right and left buttons status signals are mB, rB and lB respectively.

Table II. Mouse Data packet format

Byte 1	y_V	x_V	1	y_8	x_8	m_B	r_B	l_B
Byte 2	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Byte 3	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

D. Universal Serial Bus (USB)

Universal Serial Bus (USB) is a specification to establish communication between devices and a host controller, usually personal computers. It was developed to interface external devices to a host controller by eliminating the different types of connectors and using a single standard for physical and logical connections.

A USB transceiver chip is used for the USB module. It is prepared on the motherboard with its corresponding circuit for signal preparation. This transceiver is controlled using the FPGA. Also, handshaking and data transfer is also controlled by the FPGA.

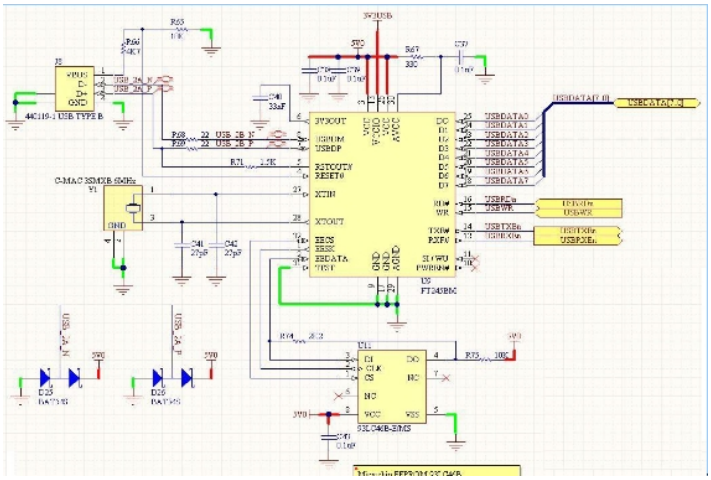

Fig. 8 USB schematic diagram using a transceiver chip

E. Parallel Port

This trainer is equipped with a sixteen (16) pin parallel

interface or otherwise called a General Purpose Input/Output (GPIO) port. Aside from the Vcc pin and the ground pin, other pins are customizable to the users specifications. For this motherboard, the default configuration is the same as IEEE1284 standard for parallel communications.

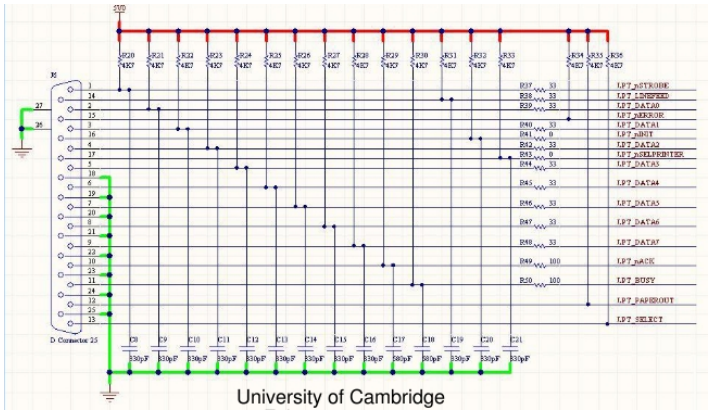

Fig. 9 Parallel Port Pin Assignment

Before USB, parallel communication was used to interface a wide range of peripheral devices to a computer. The most common of these devices was the printer. Currently, other interfaces were developed and are used other than the parallel port, predominantly the USB.

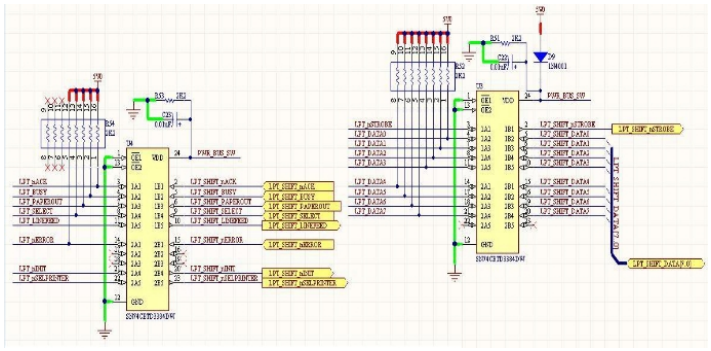

Fig. 10 Parallel Port Schematic Diagram

Recently, manufacturers consider parallel communications to be outdated and respective hardware to be legacy devices, opting to use more recent interfaces, such as USB or Firewire. As such, they removed parallel communication devices altogether.

F. Analog-to-Digital/Digital-to-Analog Converters

An Analog-to-Digital Converter(ADC) is a device that converts an analog signal, such as voltages or currents, into discrete digital signals representative of the analog signal being converted. The reverse is true of a Digital-to-Analog Converter(DAC) where discrete signals of a predetermined coding scheme is converted into analog signals.

For this motherboard a MAX1240 IC was used for the ADC component while MAX550 IC was used for the DAC.

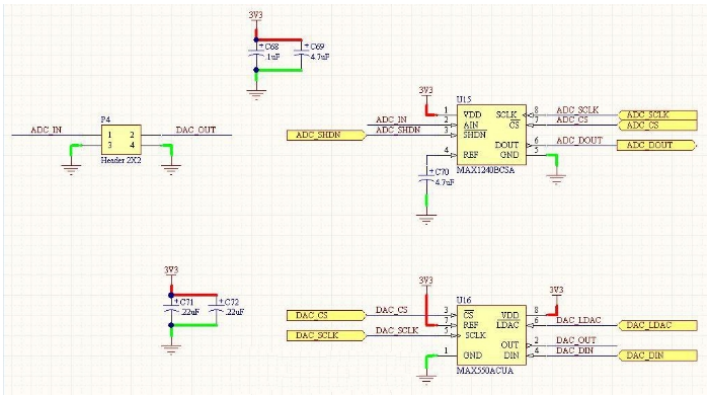

Fig. 11 ADC/DAC schematic diagram

The following the the HDL codes for the controllers of the ICs in Verilog HDL language:

```
module MAX_550(RST_n, CLK, START, CTRL_BYTE,
    DT_BYTE, MAX_SCLK, MAX_CS_n, MAX_LDAC_n,
    MAX_DIN);
 input  RST_n;
 input  CLK;
 input  START;
 input  [7:0] CTRL_BYTE;
 input  [7:0] DT_BYTE;
  output  MAX_SCLK;
  output  MAX_CS_n;
  output  MAX_LDAC_n;//output if MAX550
  output  MAX_DIN;

  reg  MAX_SCLK;
  reg  MAX_CS_n;
  reg  MAX_DIN;

  reg  [3:0] CLK_CNT;
  reg  [4:0] BIT_CNT;
  reg  [15:0] TRANS_DATA;

  wire  START_PE;
  wire  MAX_CS_n_PE;
  wire  MAX_SCLK_PE;
 wire  MAX_SCLK_NE;


 wire  START_GO;
 wire  MAX_SCLKfr_NE;

 reg  [15:0] BUSY_n_shift;
 reg  BUSY;

 always @(posedge CLK)
  if (!RST_n) BUSY_n_shift <= 16'hffff;
  else if (START_GO) BUSY_n_shift <= 16'h0000;
          else  if  (MAX_SCLKfr_NE)  BUSY_n_shift  <=
   {BUSY_n_shift[14:0], MAX_CS_n};
```

```
always @(posedge CLK)
  BUSY <= ~|BUSY_n_shift[1];

always @(posedge CLK)
  if (~RST_n) CLK_CNT <= 0;
  else if (START_GO) CLK_CNT <= 0;
  else CLK_CNT <= CLK_CNT + 1;

always @(posedge CLK)
  if (~RST_n) MAX_SCLK <= 0;
  else MAX_SCLK <= CLK_CNT[3] & (BIT_CNT < 16);

always @(posedge CLK)
  if (~RST_n) MAX_CS_n <= 1;
  else if (START_GO) MAX_CS_n <= 0;
  else if (&BIT_CNT) MAX_CS_n <= 1;

//if MAX550
 reg   MAX_LDAC_n;
 reg MAX_LDAC_n_dn1, MAX_LDAC_n_dn2;
 always @(posedge CLK)
 begin
  MAX_LDAC_n_dn2 <= MAX_LDAC_n_dn1;
    MAX_LDAC_n    <= MAX_LDAC_n_dn2;
 end

 always @(posedge CLK)
  if (~RST_n) MAX_LDAC_n_dn1 <= 1'b1;
  else if (MAX_CS_n_PE) MAX_LDAC_n_dn1 <= 1'b0;
  else if (MAX_SCLKfr_NE) MAX_LDAC_n_dn1 <= 1'b1;

 always @(posedge CLK)
  if (~RST_n) MAX_DIN <= 0;
   else if (~MAX_CS_n && (BIT_CNT < 16)) MAX_DIN <=
    TRANS_DATA[BIT_CNT];

 always @(posedge CLK)
  if (~RST_n) TRANS_DATA <= 16'b0000000000000000;
    else if (START_GO) TRANS_DATA <= {CTRL_BYTE,
    DT_BYTE};

 always @(posedge CLK)
  if (~RST_n) BIT_CNT <= 5'b11111;
  else if (START_GO) BIT_CNT <= 5'd15;
    else if (MAX_SCLK_NE & ~&BIT_CNT) BIT_CNT <=
    BIT_CNT - 1;

 assign START_GO = START_PE & ~BUSY;

    edge_detector #1 edge_detector0(CLK, 1'b1, START,
    START_PE);
 edge_detector #0 edge_detector1(CLK, 1'b1, CLK_CNT[3],
    MAX_SCLKfr_NE);
  edge_detector #1 edge_detector2(CLK, 1'b1, MAX_CS_n,
    MAX_CS_n_PE);
 edge_detector #1 edge_detector3(CLK, 1'b1, CLK_CNT[3] &
    ~MAX_CS_n, MAX_SCLK_PE);
 edge_detector #0 edge_detector4(CLK, 1'b1, CLK_CNT[3] &
    ~MAX_CS_n, MAX_SCLK_NE);

endmodule

module MAX_1240(RST_n, CLK, START,
   MAX1240_SCLK, MAX1240_CS_n, MAX1240_DOUT,
   DT_VALID, DT, NEXT);
 input  RST_n;
 input  CLK;
 input  START;
 output MAX1240_SCLK;
 output MAX1240_CS_n;
 input  MAX1240_DOUT;
 output DT_VALID;
 output [11:0] DT;
 output NEXT;

 reg    MAX1240_SCLK;
 reg    MAX1240_CS_n;
 reg    [11:0] DT;
 reg    DT_VALID;

 reg    [4:0] CLK_CNT;
 reg    [4:0] BIT;
 reg    [15:0] DT_LATCH;
 reg    DT_RDY;

 wire   MAX1240_CS_PE;
 wire   MAX1240_CS_NE;
 wire   MAX1240_SCLK_PE;
 wire   MAX1240_SCLK_NE;
 wire   MAX1240_DOUT_PE;
 wire   START_PE;
 wire   DT_RDY_PE;
 reg MAX1240_CS_nx;
 reg SCLKenable;
 reg [9:0] cnten;
 reg enable;
 reg [9:0] cntenx;

 always @(posedge CLK)
  if (~RST_n)          cntenx <= 0;
  else if (!MAX1240_CS_n) cntenx <= 0;
  else if (cntenx == 60) cntenx <= 60;
  else                cntenx <= cntenx + 1;


 always @(posedge CLK)
  if (~RST_n)          enable <= 0;
  else if (cntenx > 58) enable <= 1;
  else                enable <= 0;


  edge_detector1 #1 _MAX1240_NEXT(CLK, 1'b1, enable,
   NEXT);
     edge_detector1   #1   _MAX1240_CS_PE(CLK,   1'b1,
   MAX1240_CS_n, MAX1240_CS_PE);
     edge_detector1   #0   _MAX1240_CS_NE(CLK,   1'b1,
   MAX1240_CS_nx, MAX1240_CS_NE);
    edge_detector1   #1   _MAX1240_SCLK_PE(CLK,   1'b1,
```

```
    MAX1240_SCLK, MAX1240_SCLK_PE);
     edge_detector1  #0  _MAX1240_SCLK_NE(CLK,  1'b1,
    MAX1240_SCLK, MAX1240_SCLK_NE);
     edge_detector1  #1  _MAX1240_DOUT_PE(CLK,  1'b1,
    MAX1240_DOUT, MAX1240_DOUT_PE);
     edge_detector1  #1  _START_PE(CLK,  1'b1,  START,
    START_PE);
    edge_detector1  #1  _DT_RDY_PE(CLK,  1'b1,  DT_RDY,
    DT_RDY_PE);

//*****************************************
// MAX1240 CS_n pin

  always @(posedge CLK)
    if (~RST_n)                  MAX1240_CS_nx <= 1;
              //else    if    (START_PE    &    MAX1240_CS_n)
     MAX1240_CS_nx <= 0;
    else if (START_PE)             MAX1240_CS_nx <= 0;
    else if (&BIT & MAX1240_SCLK_NE) MAX1240_CS_nx
     <= 1;

reg [3:0] scnt;
initial scnt = 0;
  always @(posedge CLK)
     if (MAX1240_CS_PE) scnt <= 0;
     else              scnt <= scnt + 1;

  always @(posedge CLK)
   if (!RST_n)       MAX1240_CS_n <= 1;
   else if (&scnt)   MAX1240_CS_n <= MAX1240_CS_nx;

//*****************************************
// clock division, clock position markers

  always @(posedge CLK)
    if (~RST_n) CLK_CNT <= 0;
     else if (MAX1240_CS_NE | DT_RDY_PE) CLK_CNT <=
     0;
     else if (SCLKenable && !MAX1240_CS_nx ) CLK_CNT
     <= CLK_CNT + 1;

  always @(posedge CLK)
    if (~RST_n)          cnten <= 0;
    else if (MAX1240_CS_n || !DT_RDY) cnten <= 0;
    else if (cnten == 50) cnten <= 50;
    else                 cnten <= cnten + 1;


  always @(posedge CLK)
   if (~RST_n)          SCLKenable <= 0;
   else if (cnten > 48) SCLKenable <= 1;
   else                 SCLKenable <= 0;


  always @(posedge CLK)
    if (~RST_n)        MAX1240_SCLK <= 0;
    else if (SCLKenable) MAX1240_SCLK <= CLK_CNT[4];

//*****************************************
// data ready marker

  always @(posedge CLK)
    if (~RST_n) DT_RDY <= 0;
    else if (MAX1240_CS_NE) DT_RDY <= 0;
    else if (MAX1240_DOUT_PE) DT_RDY <= 1;

//*****************************************
// BIT marker

  always @(posedge CLK)
    if (~RST_n) BIT <= 15;
    else if (DT_RDY_PE) BIT <= 15;
    else if (MAX1240_SCLK_PE & ~&BIT) BIT <= BIT - 1;

//*****************************************
// digital data

  always @(posedge CLK)
    if (~RST_n) DT_LATCH <= 0;
        else  if  (MAX1240_SCLK_PE)  DT_LATCH[BIT]  <=
      MAX1240_DOUT;

  always @(posedge CLK)
    if (~RST_n) DT <= 0;
    else if (DT_VALID) DT <= DT_LATCH[14:3];
    else       DT <= 0;

  always @(posedge CLK)
    if (~RST_n)          DT_VALID <= 0;
    else if (START_PE)     DT_VALID <= 0;
    else if (MAX1240_CS_PE) DT_VALID <= 1;

endmodule
```

G. Digital Display Devices

Other display devices that are included in the motherboard are the seven-segment display and the liquid crystal display(LCD).

A seven segment display is a component that is composed of seven elements that can be turned on or off. In combination these elements can represent numerals or alphabets.

In the case of the motherboard the seven-segment display elements are light emitting diodes (LEDs) that can be controlled by registers within the FPGA, these registers are used as latches to tell the individual elements if they should be on or off.

The liquid crystal display uses the same concept for display and relaying information. Rather that limiting it to seven segments, a LCD screen uses more elements in a dot-matrix environment capable of displaying more complex representations on the display.

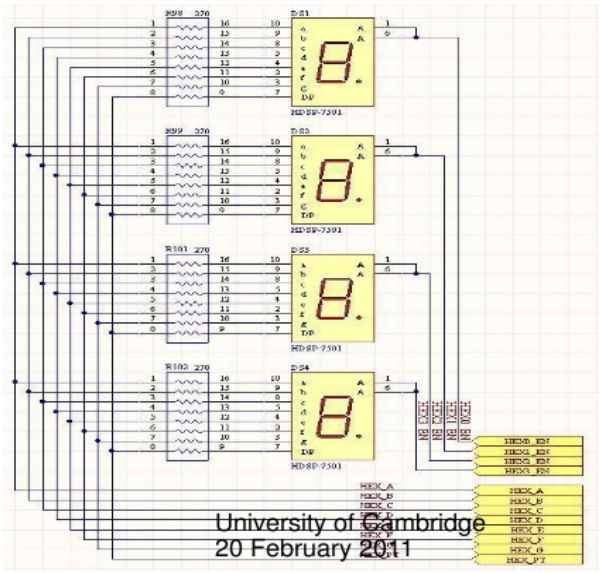

Fig. 12 7-Segment Display schematic diagram

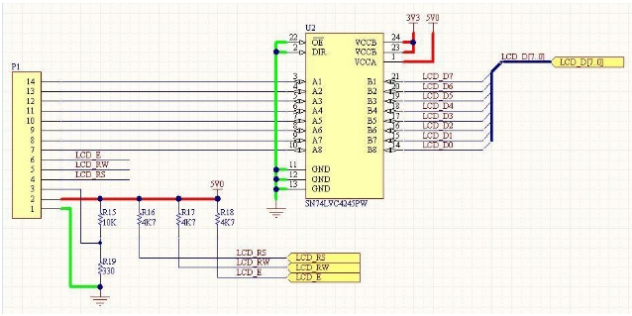

Fig. 13 Liquid Crystal Display schematic diagram

III. Summary

The Data and Network Communications Protocol Motherboard Using Reconfigurable Hardware was designed to develop a firm base of knowledge by giving more of a hands-on approach to learning.

The core of this motherboard is a reconfigurable FPGA. Using a compiler and some basic skills. A user will be able to design and implement his own project using the different devices on the board.

The board is outfitted with a VGA port to be connected to a VGA ready monitor. The user will design his own timing signals to be able to implement a VGA controller as well as design the display itself.

Two PS/2 ports are provided. The user will be able to use these as ports for a keyboard as well as for a mouse.

A USB port is also provided. This USB is already designed to transmit over a USB cable and what is needed from the user are the control signals as well as the data to be sent over the line.

A general input/output port is also provided. Primarily it will be used as a parallel port for parallel communications. It will also be used as an interface for daughter boards to be designed and implemented for additional functions for the motherboard.

For serial communications three DB9 ports are provided. They are designed to emulate communication devices using serial communications such as DTE and DCE devices.

Acknowledgment

The authors wish to extend their gratitude to the Philippine Council for Advanced Science and Technology Research and Development of the Department of Science and Technology for funding this study and to the Department of Electronics, Computer, and Communications Engineering Department, Ateneo de Manila University.

References

[1] Carlson, A. Bruce, *Communication Systems*, McGraw-Hill, 2002.

[2] Smith, Doug J., *HDL Chip Design,* Doone Publications, 2001.

[3] E. Alaer, A. Tangel, and M. Yakut., ?MIB-16? FPGA Based Design and Implementation of a 16-Bit Microprocessor for Educational Use, WSEAS Transactions on Advances in Engineering Education, Vol.5, No.5, 2008, pp.326-330.

[4] A. J. Araujo and J. C. Alves, A Project Based Methodology to Teach a Course on Advanced Digital Systems Design. WSEAS Transactions on Advances in Engineering Education, Vol.5, No.6, 2008, pp. 437-446

[5] Rosula Reyes, Carlos Oppus, Jose Claro Monje, Noel Patron, Reynaldo Guerrero, and Jovilyn Therese Fajardo. "FPGA Implementation of a Telecommunications Trainer System". International Journal of Circuits, Systems and Signal Processing. NAUN Press, Vol. 2, Issue 2, 2008, pp. 87-94. (ISSN: 1998-4464).Rosula Reyes, Carlos Oppus, Jose Claro Monje, Noel Patron, Reynaldo Guerrero, and Jovilyn Therese Fajardo. "FPGA Implementation of a Telecommunications Trainer System". International Journal of Circuits, Systems and Signal Processing. NAUN Press, Vol. 2, Issue 2, 2008, pp. 87-94. (ISSN: 1998-4464).

[6] Maxfield, Clive, *The Design Warrior's Guide to FPGAs: Devices, Tools, and Flows*, Elsevier Science and Technology, 2004.

[7] Brown, Stephen, Zvonko Vranesic, *Fundamentals of Digital Logic with Verilog Design*, McGraw-Hill, 2008.

[8] Chu, Pong P., *FPGA Prototyping by Verilog Design*, John Wiley & Sons, Inc., 2008.

[9] Rosula Reyes, Carlos Oppus, Jose Claro Monje, Noel Patron, Raphael Gonzales, and Jovilyn Therese Fajardo. "FPGA-based DSP Trainer," CSIE 2009, 2009 WRI World Congress on Computer Science and Information Engineering, March 31 - April 2, 2009, Los Angeles, California, USA, 7 Volumes, IEEE Computer Society 2009, Volume 5, pp. 343-347 (ISBN 978-0-7695-3507-4).