UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

CÁTEDRA DE ELECTRÓNICA DIGITAL I

TRABAJO PRÁCTICO Nº 3

“Proyecto FPGA: Sumador - Restador - AND - OR de 4 bits”

Grupo Nº 3

Alumnos:
Langleib, Emilia

Rojo, Juan Pablo

Saillen, Simón

Vera, Violeta

NOTA:

Profesor:
Ing. Vrech, Ruben

Comisión: Viernes 9:30 hs

Mayo / 2022

Consigna

El objetivo de este Trabajo Práctico es diseñar y programar en una placa Digilent Basys2 un circuito ALU (*“Arithmetic Logic Unit”*) en una FPGA Xilinx Spartan 3E-100 bajo el lenguaje de programación VHDL *(“Very high speed integrated circuits Hardware Description Software”)*. Utilizando software de libre acceso debemos implementar un código para modificar en nivel hardware a la FPGA y así lograr la implementación de la ALU.

Desarrollo

1. Declaración de Variables

Partiendo de la consigna, se nos pide utilizar las siguientes *entradas*:

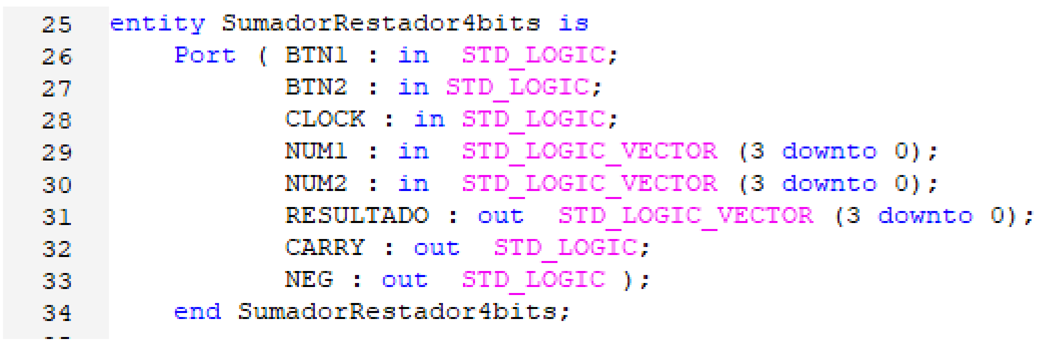
- **4 interruptores** para el primer número binario de 4 bits.
- **4 interruptores** para el segundo número binario de 4 bits.
- **2 pulsadores** para representar la operación binaria.

*(En nuestro grupo, para mayor comodidad a la hora de realizar las operaciones, decimos utilizar un **tercer pulsador**, asignado como "Clock").*

Siguiendo con la consigna, se nos pide utilizar las siguientes salidas:

- **4 LED** para representar el resultado de la operación binaria.
- **1 LED** para representar si el resultado es un número negativo.
- **1 LED** para representar si en la suma se utilizó Acarreo.

Entonces, para la implementación de nuestro código, declaramos las siguientes variables de entrada y de salida:

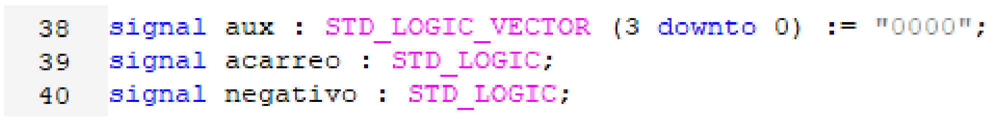


```
entity SumadorRestador4bits is
    Port ( BTN1 : in  STD_LOGIC;
           BTN2 : in STD_LOGIC;
           CLOCK : in STD_LOGIC;
           NUM1 : in  STD_LOGIC_VECTOR (3 downto 0);
           NUM2 : in  STD_LOGIC_VECTOR (3 downto 0);
           RESULTADO : out  STD_LOGIC_VECTOR (3 downto 0);
           CARRY : out  STD_LOGIC;
           NEG : out  STD_LOGIC );
    end SumadorRestador4bits;
```

Imagen 1: Declaración de Entradas y Salidas en VHDL

Optamos por separar las entradas BTN1 y BTN2 para tener un código más limpio y con menos comparadores.

También, debido a que programamos a un nivel hardware, tuvimos que declarar las siguientes variables internas, que son usadas dentro del *proceso* y luego asignadas al valor de salida correspondiente una vez terminado el mismo.



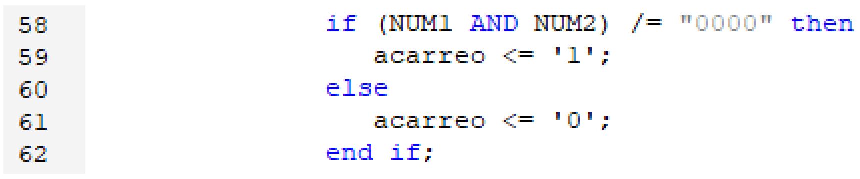
```
signal aux : STD_LOGIC_VECTOR (3 downto 0) := "0000";
signal acarreo : STD_LOGIC;
signal negativo : STD_LOGIC;
```

Imagen 2: Declaración de variables Internas (señales auxiliares del circuito) en VHDL

2. BITs de Acarreo y de Número Negativo

Basándonos en la definición de Acarreo, este ocurre cuando en la operación SUMA se suman dos bits de valor 1, dejando como resultado 0 y acarreando un 1 para el siguiente bit.
Por ejemplo: 0010 + 0011 = 0101, en el tercer bit ocurre un acarreo.

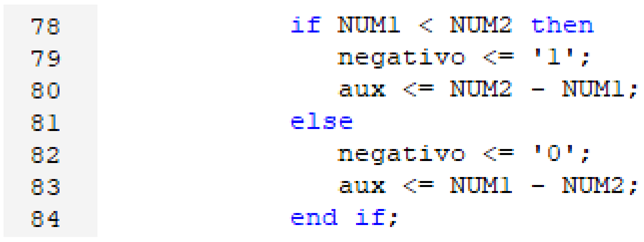
Entonces para implementar esto en código, optamos por realizar una operación AND entre los 2 números de 4 bits ingresados, esta nos dará un valor distinto de “0000” en caso de que ocurra algún acarreo. Quedándonos el código de la siguiente manera:



```
if (NUM1 AND NUM2) /= "0000" then
   acarreo <= '1';
else
   acarreo <= '0';
end if;
```

Imagen 3: Condición de BIT de Acarreo

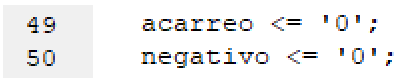
Para la condición de si el número resultado es negativo, al realizar la operación RESTA, comparamos si el primer número es menor al segundo número, en caso afirmativo, cambiamos el orden de la resta, restando el segundo termino (más grande) con el primer término (más chico), y el resultado de la resta dará un número negativo. Quedando el código:



```
if NUM1 < NUM2 then
   negativo <= '1';
   aux <= NUM2 - NUM1;
else
   negativo <= '0';
   aux <= NUM1 - NUM2;
end if;
```

Imagen 4: Condición de BIT de Número Negativo

(Aclaración: Debido a que cada vez que se realiza una operación se deben reiniciar los BITs, debimos agregar 2 líneas de código al principio del proceso).

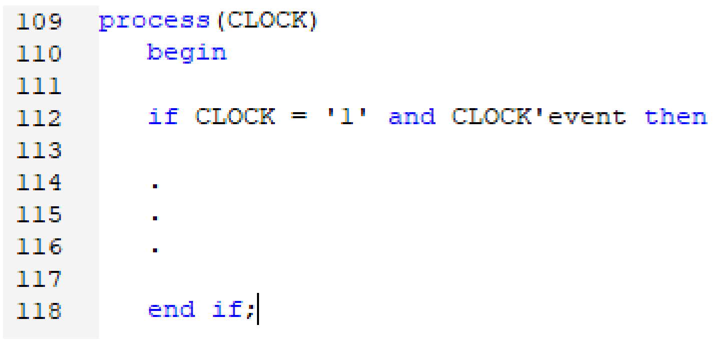


```
acarreo <= '0';
negativo <= '0';
```

Imagen 5: Reset de los BIT al principio de Process

3. Clock

Como explicamos anteriormente, debido a que en la consigna se puede optar por cualquier otra implementación aparte de la indicada, con nuestro grupos optamos por el uso de un Clock, esto se realizó para “congelar” el resultado de cada operación para tener mejor control y visibilidad del mismo.
Para realizarlo, decidimos dedicar el cuarto pulsador, entonces usamos los primeros 2 pulsadores para elegir la operación deseada, y el cuarto para procesar y congelar el resultado.
Entonces, como se verá posteriormente, el proceso lo empieza el Clock, quedando el código de la siguiente manera:



```
process(CLOCK)
   begin

   if CLOCK = '1' and CLOCK'event then

   .
   .
   .

   end if;
```

Imagen 6: Principio del Proceso y Condición de Comienzo
(La imagen es solo de forma ilustrativa).

4. Implementación de la ALU y Asignación de PINs

Luego de determinar las partes de la ALU, y las condiciones a tener en cuenta para su realización que explicamos anteriormente, procedemos a realizar el código de funcionamiento de la misma.
El mismo es agregado en la siguiente página del informe junto al archivo de asignación de Pines en la Basys 2:

```vhdl
----------------------------------------------------------------------------------
-- Company:        FCEFyN - UNC
-- Engineer:       Grupo 03 - ED1
--
-- Create Date:    20:51:56 05/15/2022
-- Design Name:    Trabajo Practico 3
-- Module Name:    SumadorRestador4bits - Behavioral
-- Project Name:   ALU con FPGA
-- Target Devices: Spartan 3E-100 CP-132
-- Tool versions:  Xilinx ISE 14.7
-- Description:    Sumador - Restador - AND - OR de 4 bits
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments: (Clock agregado para un control mas estable en las operaciones)
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SumadorRestador4bits is
    Port ( BTN1 : in  STD_LOGIC;
           BTN2 : in STD_LOGIC;
           CLOCK : in STD_LOGIC;
           NUM1 : in  STD_LOGIC_VECTOR (3 downto 0);
           NUM2 : in  STD_LOGIC_VECTOR (3 downto 0);
           RESULTADO : out  STD_LOGIC_VECTOR (3 downto 0);
           CARRY : out  STD_LOGIC;
           NEG : out  STD_LOGIC );
    end SumadorRestador4bits;

architecture Behavioral of SumadorRestador4bits is

signal aux : STD_LOGIC_VECTOR (3 downto 0)  := "0000";
signal acarreo : STD_LOGIC := '0';
signal negativo : STD_LOGIC := '0';

begin

   process(CLOCK)
   begin

   if CLOCK = '1' and CLOCK'event then

   acarreo <= '0';                                     -- Reseteamos los valores de...
   negativo <= '0';                                    -- ...acarreo y negativo para...
                                                       -- ...evitar arrastre del valor.
      if BTN1 = '0' then

         if BTN2 = '0' then

            aux <= NUM1 + NUM2;                        -- SUMA

```

```vhdl
                  if (NUM1 AND NUM2) /= "0000" then       -- Verifica si hubo Acarreo
                     acarreo <= '1';
                  else
                     acarreo <= '0';
                  end if;

            else

               aux <= NUM1 AND NUM2;                      -- AND

            end if;

         else

            if BTN2 = '0' then                            -- OR

               aux <= NUM1 OR NUM2;

            else

               if NUM1 < NUM2 then                        -- Verifica si la resta...
                  negativo <= '1';                        -- ...dará número negativo...
                  aux <= NUM2 - NUM1;                     -- ...y en ese caso invierte...
               else                                       -- ...el orden
                  negativo <= '0';
                  aux <= NUM1 - NUM2;
               end if;

            end if;
         end if;
      end if;
      end process;
                                                           -- Asigna los valores al...
    RESULTADO <= aux;                                      -- ...finalizar el proceso
    CARRY <= acarreo;
    NEG <= negativo;

  end Behavioral;
```

```

# PlanAhead Generated physical constraints

NET "NUM1[0]" LOC = G3;
NET "NUM1[1]" LOC = F3;
NET "NUM1[2]" LOC = E2;
NET "NUM1[3]" LOC = N3;
NET "NUM2[0]" LOC = P11;
NET "NUM2[1]" LOC = L3;
NET "NUM2[2]" LOC = K3;
NET "NUM2[3]" LOC = B4;
NET "BTN1" LOC = A7;
NET "BTN2" LOC = M4;
NET "CARRY" LOC = M5;
NET "NEG" LOC = M11;
NET "CLOCK" LOC = G12;
NET "RESULTADO[0]" LOC = N5;
NET "RESULTADO[1]" LOC = N4;
NET "RESULTADO[2]" LOC = P4;
NET "RESULTADO[3]" LOC = G1;
NET "CLOCK" CLOCK_DEDICATED_ROUTE = FALSE;
```

Diagrama Circuital

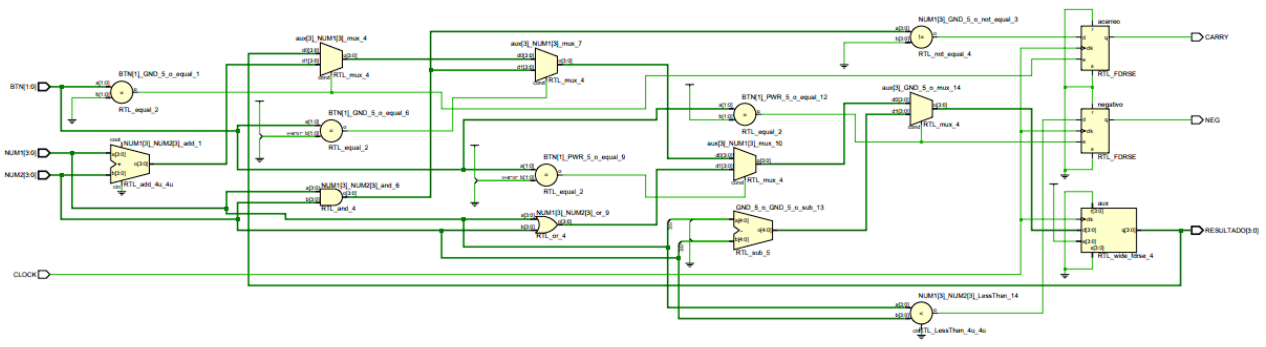


Imagen 7: Diagrama Circuital generado por el software sobre la ALU programada

Gráficos Topológicos

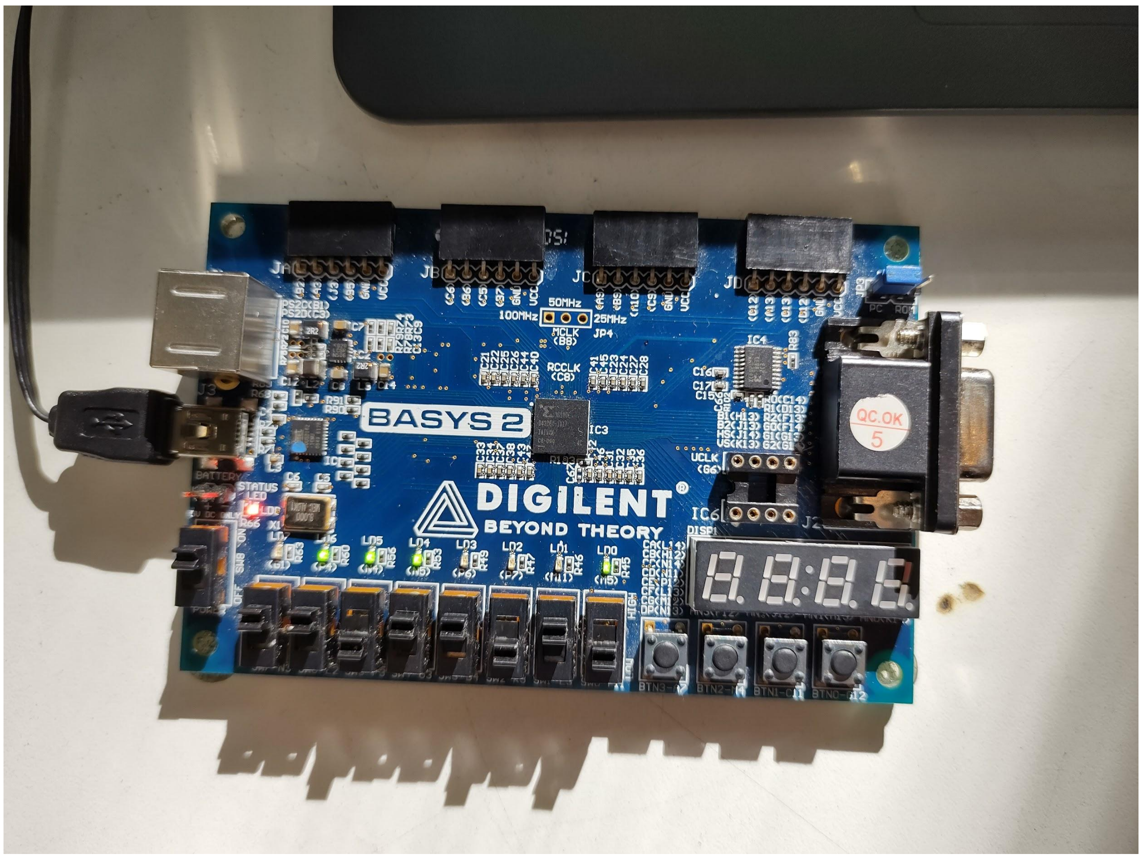


Imagen 8: Fotografía de la Basys 2 con el archivo .bit funcionando

Conclusiones

En este trabajo práctico se logró programar en la placa Basys 2, implementando conocimientos adquiridos en el lenguaje de programación VHDL y los software Xilinx ISE 14.7 y Digilent Adept, destacamos la facilidad que se tuvo a la hora de realizar este trabajo gracias a los conocimientos previos adquiridos en los trabajos prácticos anteriores.

Bibliografía y referencias

- [1] DIGILENT Reference https://digilent.com/reference/programmable-logic/basys-2/reference-manual

Hojas de Datos

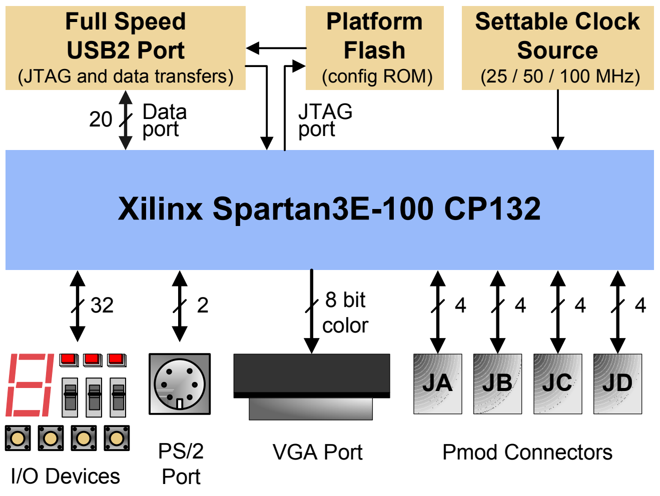
Digilent Basys2 Board Reference Manual

Revision: November 11, 2010

1300 Henley Court | Pullman, WA 99163
(509) 334 6306 Voice and Fax

Introduction

The Basys2 board is a circuit design and implementation platform that anyone can use to gain experience building real digital circuits. Built around a Xilinx Spartan-3E Field Programmable Gate Array and a Atmel AT90USB2 USB controller, the Basys2 board provides complete, ready-to-use hardware suitable for hosting circuits ranging from basic logic devices to complex controllers. A large collection of on-board I/O devices and all required FPGA support circuits are included, so countless designs can be created without the need for any other components.



- *100,000-gate Xilinx Spartan 3E FPGA*
- *Atmel AT90USB2 Full-speed USB2 port providing board power and programming/data transfer interface*
- *Xilinx Platform Flash ROM to store FPGA configurations*
- *8 LEDs, 4-digit 7-segment display, 4 buttons, 8 slide switches*
- *PS/2 port and 8-bit VGA port*
- *User-settable clock (25/50/100MHz), plus socket for 2nd clock*
- *Four 6-pin header expansion connectors*
- *ESD and short-circuit protection on all I/O signals.*

Figure 1. Basys2 board block diagram and features

Four standard expansion connectors allow designs to grow beyond the Basys2 board using breadboards, user-designed circuit boards, or Pmods (Pmods are inexpensive analog and digital I/O modules that offer A/D & D/A conversion, motor drivers, sensor inputs, and many other features). Signals on the 6-pin connectors are protected against ESD damage and short-circuits, ensuring a long operating life in any environment. The Basys2 board works seamlessly with all versions of the Xilinx ISE tools, including the free WebPack. It ships with a USB cable that provides power and a programming interface, so no other power supplies or programming cables are required.

The Basys2 board can draw power and be programmed via its on-board USB2 port. Digilent's freely available PC-based Adept software automatically detects the Basys2 board, provides a programming interface for the FPGA and Platform Flash ROM, and allows user data transfers (see www.digilentinc.com for more information).

The Basys2 board is designed to work with the free ISE WebPack CAD software from Xilinx. WebPack can be used to define circuits using schematics or HDLs, to simulate and synthesize circuits, and to create programming files. Webpack can be downloaded free of charge from www.xilinx.com/ise/.

The Basys2 board ships with a built-in self-test/demo stored in its ROM that can be used to test all board features. To run the test, set the Mode Jumper (see below) to ROM and apply board power. If the test is erased from the ROM, it can be downloaded and reinstalled at any time. See www.digilentinc.com/Basys2 for the test project as well as further documentation, reference designs, and tutorials.

Board Power

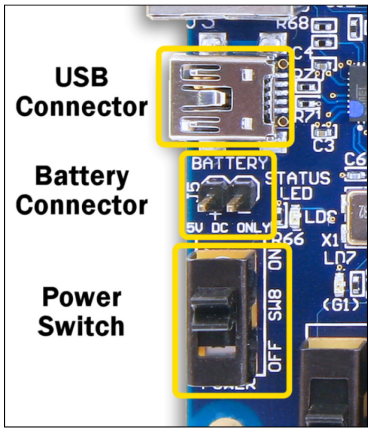


Figure 2. Basys2 power circuits

The Basys2 board is typically powered from a USB cable, but a battery connector is also provided so that external supplies can be used. To use USB power, simply attach the USB cable. To power the Basys2 using a battery or other external source, attach a 3.5V-5.5V battery pack (or other power source) to the 2-pin, 100-mil spaced battery connector (three AA cells in series make a good 4.5+/- volt supply). Voltages higher than 5.5V on either power connector may cause permanent damage.

Input power is routed through the power switch (SW8) to the four 6-pin expansion connectors and to a Linear Technology LTC3545 voltage regulator. The LTC3545 produces the main 3.3V supply for the board, and it also drives secondary regulators to produce the 2.5V and 1.2V supply voltages required by the FPGA. Total board current is dependant on FPGA configuration, clock frequency, and external connections. In test circuits with roughly 20K gates routed, a 50MHz clock source, and all LEDs illuminated, about 100mA of current is drawn from the 1.2V supply, 50mA from the 2.5V supply, and 50mA from the 3.3V supply. Required current will increase if larger circuits are configured in the FPGA, or if peripheral boards are attached.

The Basys2 board uses a four layer PCB, with the inner layers dedicated to VCC and GND planes. The FPGA and the other ICs on the board have large complements of ceramic bypass capacitors placed as close as possible to each VCC pin, resulting in a very clean, low-noise power supply.

Configuration

After power-on, the FPGA on the Basys2 board must be configured before it can perform any useful functions. During configuration, a “bit” file is transferred into memory cells within the FPGA to define the logical functions and circuit interconnects. The free ISE/WebPack CAD software from Xilinx can be used to create bit files from VHDL, Verilog, or schematic-based source files.

Digilent’s PC-based program called Adept can be used to configure the FPGA with any suitable bit file stored on the computer. Adept uses the USB cable to transfer a selected bit file from the PC to the FPGA (via the FPGA’s JTAG programming port). Adept can also program a bit file into an on-board non-volatile ROM called “Platform Flash”. Once programmed, the Platform Flash can automatically transfer a stored bit file to the FPGA at a subsequent power-on or reset event if the Mode Jumper (JP3) is set to ROM. The FPGA will remain configured until it is reset by a power-cycle event. The Platform Flash ROM will retain a bit file until it is reprogrammed, regardless of power-cycle events.

To program the Basys2 board, set the mode jumper to PC and attach the USB cable to the board. Start the Adept software, and wait for the FPGA and the Platform Flash ROM to be recognized. Use the browse function to associate the desired .bit file with the FPGA, and/or the desired .mcs file with the Platform Flash ROM. Right-click on the device to be programmed, and select the "program" function. The configuration file will be sent to the FPGA or Platform Flash, and the software will indicate whether programming was successful. The "Status LED" LED (LD_8) will also blink after the FPGA has been successfully configured. For further information on using Adept, please see the Adept documentation available at the Digilent website.

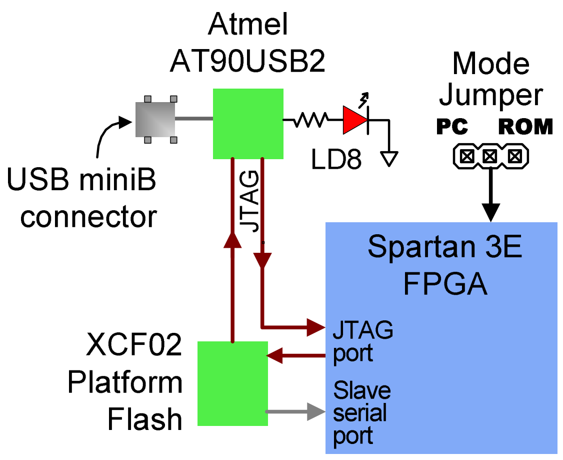


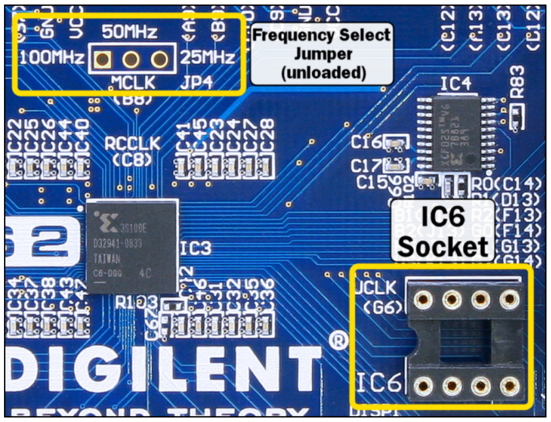
Figure 4. Basys2 Programming Circuits

Oscillators

The Basys2 board includes a primary, user-settable silicon oscillator that produces 25MHz, 50MHz, or 100MHz based on the position of the clock select jumper at JP4. Initially, this jumper is not loaded and must be soldered in place. A socket for a second oscillator is provided at IC6 (the IC6 socket can accommodate any 3.3V CMOS oscillator in a half-size DIP package). The primary and secondary oscillators are connected to global clock input pins at pin B8 and pin M6 respectively.

Both clock inputs can drive the clock synthesizer DLL on the Spartan 3E, allowing for a wide range if internal frequencies, from 4 times the input frequency to any integer divisor of the input frequency.

The primary silicon oscillator is flexible and inexpensive, but it lacks the frequency stability of a crystal oscillator. Some circuits that drive a VGA monitor may realize a slight improvement in image stability by using a crystal oscillator installed in the IC6 socket. For these applications, a 25MHz (or 50MHz) crystal oscillator, available from any catalog distributor, is recommended (see for example part number SG-8002JF-PCC at www.digikey.com).



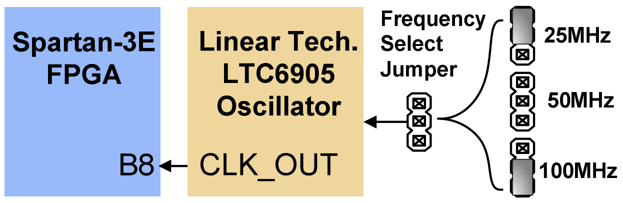


Figure 5. Basys2 oscillator circuits

User I/O

Four pushbuttons and eight slide switches are provided for circuit inputs. Pushbutton inputs are normally low and driven high only when the pushbutton is pressed. Slide switches generate constant high or low inputs depending on position. Pushbuttons and slide switches all have series resistors for protection against short circuits (a short circuit would occur if an FPGA pin assigned to a pushbutton or slide switch was inadvertently defined as an output).

Eight LEDs and a four-digit seven-segment LED display are provided for circuit outputs. LED anodes are driven from the FPGA via current-limiting resistors, so they will illuminate when a logic ‘1’ is written to the corresponding FPGA pin. A ninth LED is provided as a power-indicator LED, and a tenth LED (LD-D) illuminates any time the FPGA has been successfully programmed.

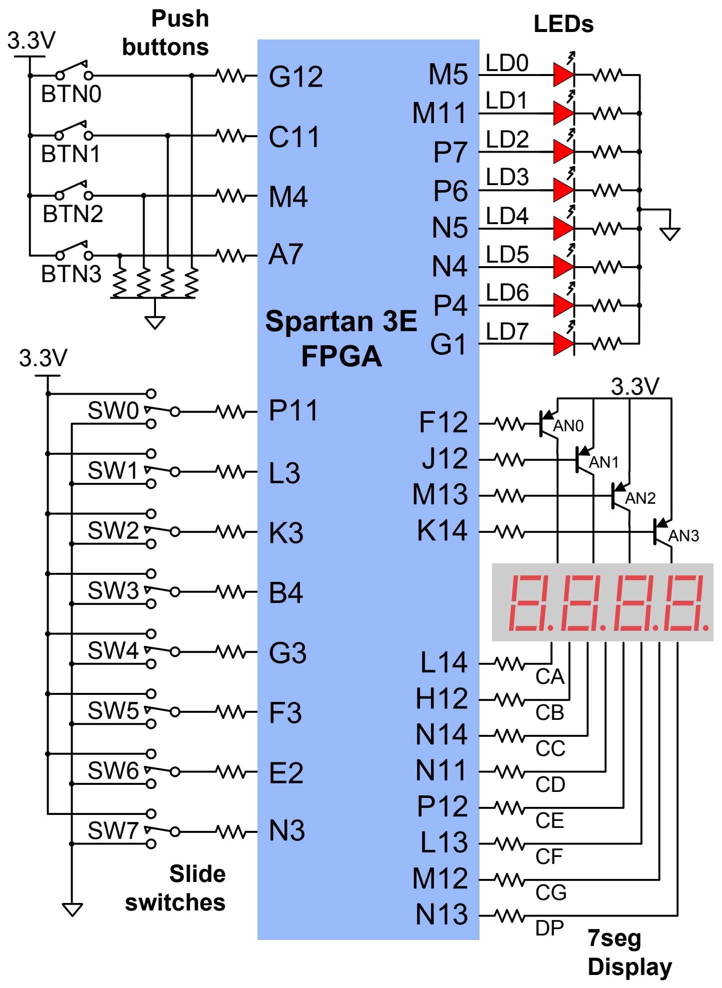


Figure 6. Basys2 I/O circuits

Seven-segment display

Each of the four digits of the seven-segment LED display is composed of seven LED segments arranged in a “figure 8” pattern. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark. Of these 128 possible patterns, the ten corresponding to the decimal digits are the most useful.

The anodes of the seven LEDs forming each digit are tied together into one common anode circuit node, but the LED cathodes remain separate. The common anode signals are available as four “digit enable” input signals to the 4-digit display. The cathodes of similar segments on all four displays are connected into seven circuit nodes labeled CA through CG (so, for example, the four “D” cathodes from the four digits are grouped together into a single circuit node called “CD”). These seven cathode signals are available as inputs to the 4-digit display. This signal connection scheme creates a multiplexed display, where the cathode signals are common to all digits but they can only illuminate the segments of the digit whose corresponding anode signal is asserted.

A scanning display controller circuit can be used to show a four-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession, at an update rate that is faster than the human eye response. Each digit is illuminated just one-quarter of the time, but because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears continuously illuminated. If the update or “refresh” rate is slowed to a given point (around 45 hertz), then most people will begin to see the display flicker.

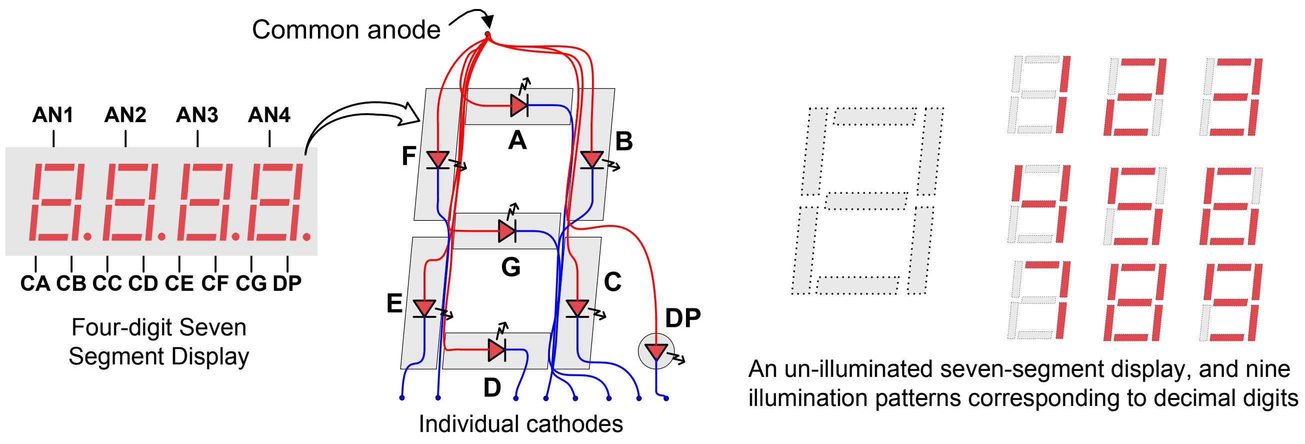

Common anode
AN1
AN2
AN3
AN4
CA CB CC CD CE CF CG DP
Four-digit Seven Segment Display
A
B
C
D
E
F
G
DP
Individual cathodes
An un-illuminated seven-segment display, and nine illumination patterns corresponding to decimal digits

Figure 7. Seven-segment display

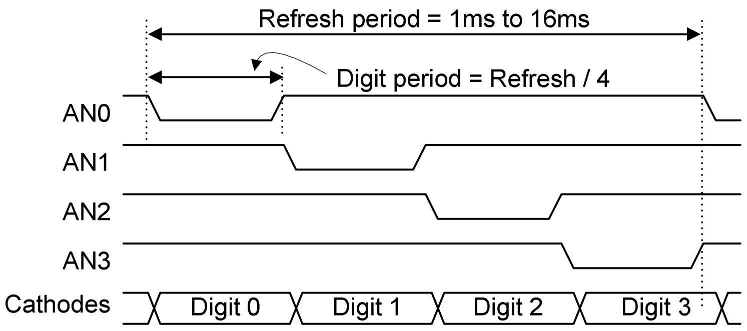


Figure 8. Multiplexed 7seg display timing

For each of the four digits to appear bright and continuously illuminated, all four digits should be driven once every 1 to 16ms (for a refresh frequency of 1KHz to 60Hz). For example, in a 60Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for ¼ of the refresh cycle, or 4ms. The controller must assure that the correct cathode pattern is present when the corresponding anode signal is driven. To illustrate the process, if AN1 is asserted while CB and CC are asserted, then a “1” will be displayed in digit position 1. Then, if AN2 is asserted while CA, CB and CC are asserted, then a “7” will be displayed in digit position 2. If A1 and CB, CC are driven for 4ms, and then A2 and CA, CB, CC are driven for 4ms in an endless succession, the display will show “17” in the first two digits. Figure 8 shows an example timing diagram for a four-digit seven-segment controller.

PS/2 Port

The 6-pin mini-DIN connector can accommodate a PS/2 mouse or keyboard. The PS/2 connector is supplied with 5VDC.

Both the mouse and keyboard use a two-wire serial bus (clock and data) to communicate with a host device. Both use 11-bit words that include a start, stop and odd parity bit, but the data packets are organized differently, and the keyboard interface allows bi-directional data transfers (so the host device can illuminate state LEDs on the keyboard). Bus timings are shown in the figure.
The clock and data signals are only driven when data transfers occur, and otherwise they are held in the “idle” state at logic ‘1’. The timings define signal requirements for mouse-to-host communications and bi-directional keyboard communications. A PS/2 interface circuit can be implemented in the FPGA to create a keyboard or mouse interface.

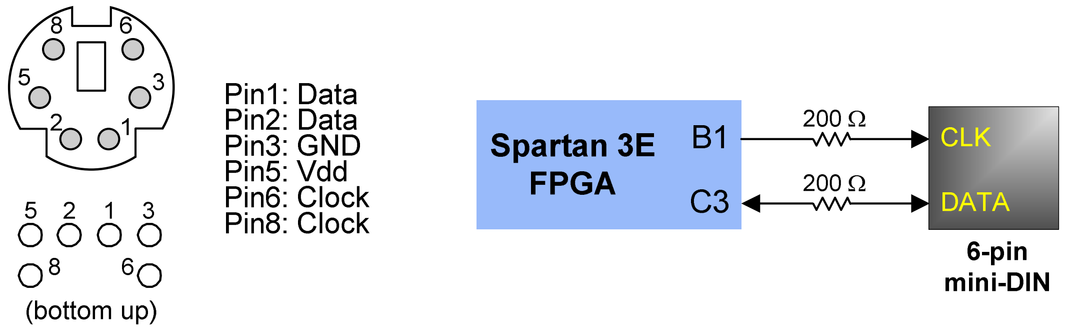
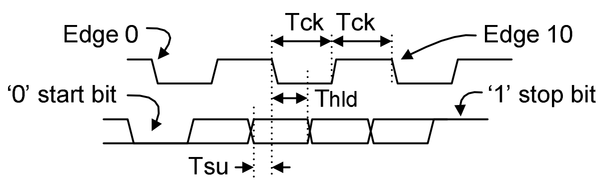


Figure 9. PS/2 connector and Basys2 PS/2 circuit

Keyboard

The keyboard uses open-collector drivers so the keyboard or an attached host device can drive the two-wire bus (if the host device will not send data to the keyboard, then the host can use input-only ports).



Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

Figure 10. PS/2 signal timing

PS2-style keyboards use scan codes to communicate key press data. Each key is assigned a code that is sent whenever the key is pressed; if the key is held down, the scan code will be sent repeatedly about once every 100ms. When a key is released, a "F0" key-up code is sent, followed by the scan code of the released key. If a key can be "shifted" to produce a new character (like a capital letter), then a shift character is sent in addition to the scan code, and the host must determine which ASCII character to use. Some keys, called extended keys, send an "E0" ahead of the scan code (and they may send more than one scan code). When an extended key is released, an "E0 F0" key-up code is sent, followed by the scan code. Scan codes for most keys are shown in the figure. A host device can also send data to the keyboard. Below is a short list of some common commands a host might send.

ED	Set Num Lock, Caps Lock, and Scroll Lock LEDs. Keyboard returns "FA" after receiving "ED", then host sends a byte to set LED status: Bit 0 sets Scroll Lock; bit 1 sets Num Lock; and Bit 2 sets Caps lock. Bits 3 to 7 are ignored.
EE	Echo (test). Keyboard returns "EE" after receiving "EE".
F3	Set scan code repeat rate. Keyboard returns "F3" on receiving "FA", then host sends second byte to set the repeat rate.
FE	Resend. "FE" directs keyboard to re-send most recent scan code.
FF	Reset. Resets the keyboard.

The keyboard can send data to the host only when both the data and clock lines are high (or idle). Since the host is the "bus master", the keyboard must check to see whether the host is sending data before driving the bus. To facilitate this, the clock line is used as a "clear to send" signal. If the host pulls the clock line low, the keyboard must not send any data until the clock is released.

The keyboard sends data to the host in 11-bit words that contain a '0' start bit, followed by 8-bits of scan code (LSB first), followed by an odd parity bit and terminated with a '1' stop bit. The keyboard generates 11 clock transitions (at around 20 - 30KHz) when the data is sent, and data is valid on the falling edge of the clock.

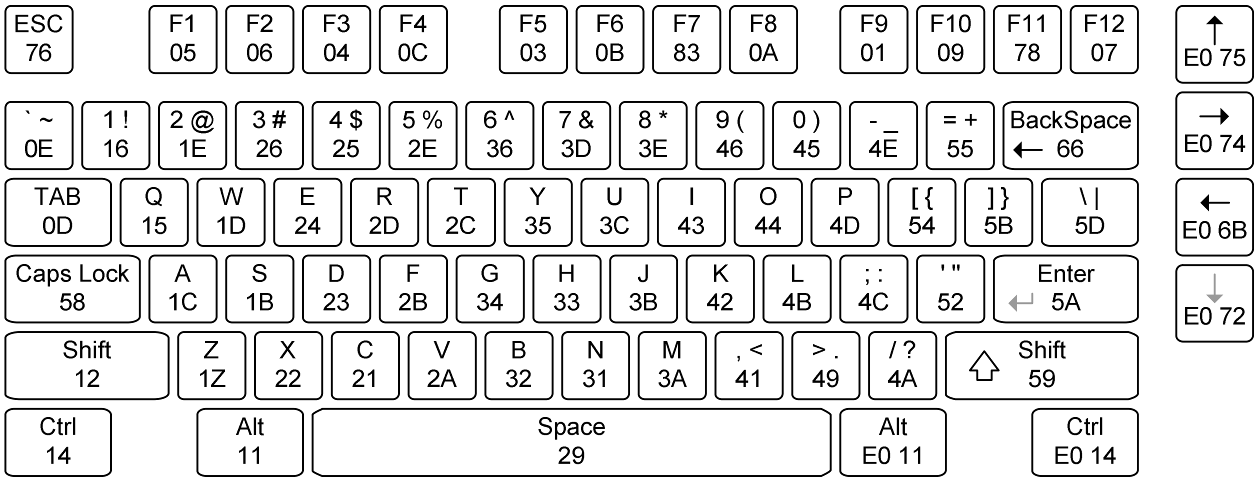


Figure 11. Keyboard scan codes

Mouse

The mouse outputs a clock and data signal when it is moved; otherwise, these signals remain at logic '1'. Each time the mouse is moved, three 11-bit words are sent from the mouse to the host device. Each of the 11-bit words contains a '0' start bit, followed by 8 bits of data (LSB first), followed by an odd parity bit, and terminated with a '1' stop bit. Thus, each data transmission contains 33 bits, where bits 0, 11, and 22 are '0' start bits, and bits 11, 21, and 33 are '1' stop bits. The three 8-bit data fields contain movement data as shown in the figure above. Data is valid at the falling edge of the clock, and the clock period is 20 to 30KHz.

The mouse assumes a relative coordinate system wherein moving the mouse to the right generates a positive number in the X field, and moving to the left generates a negative number. Likewise, moving the mouse up generates a positive number in the Y field, and moving down represents a negative number (the XS and YS bits in the status byte are the sign bits – a '1' indicates a negative number). The magnitude of the X and Y numbers represent the rate of mouse movement – the larger the number, the faster the mouse is moving (the XV and YV bits in the status byte are movement overflow indicators – a '1' means overflow has occurred). If the mouse moves continuously, the 33-bit transmissions are repeated every 50ms or so. The L and R fields in the status byte indicate Left and Right button presses (a '1' indicates the button is being pressed).

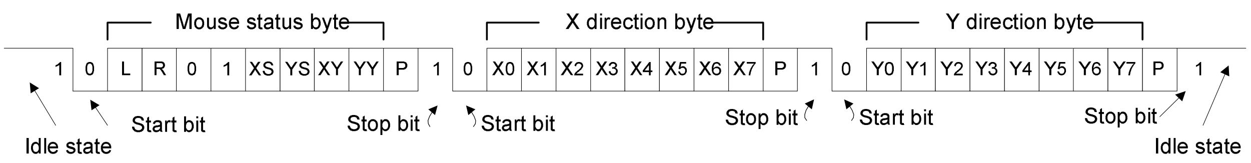
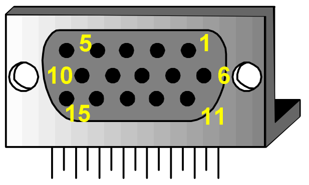


Figure 12. Mouse data format

VGA Port

The Basys2 board uses 10 FPGA signals to create a VGA port with 8-bit color and the two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). The color signals use resistor-divider circuits that work in conjunction with the 75-ohm termination resistance of the VGA display to create eight signal levels on the red and green VGA signals, and four on blue (the human eye is less sensitive to blue levels). This circuit, shown in figure 13, produces video color signals that proceed in equal increments between 0V (fully off) and 0.7V (fully on). A video controller circuit must be created in the FPGA to drive the sync and color signals with the correct timing in order to produce a working display system.



Pin 1: Red	Pin 5: GND
Pin 2: Grn	Pin 6: Red GND
Pin 3: Blue	Pin 7: Grn GND
Pin 13: HS	Pin 8: Blu GND
Pin 14: VS	Pin 10: Sync GND

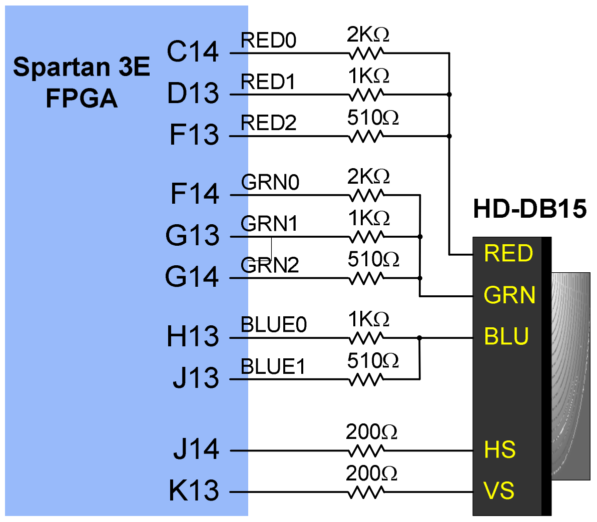


Figure 13. VGA pin definitions and Basys2 circuit

VGA System Timing

VGA signal timings are specified, published, copyrighted and sold by the VESA organization (www.vesa.org). The following VGA system timing information is provided as an example of how a VGA monitor might be driven in 640 by 480 mode. For more precise information, or for information on other VGA frequencies, refer to documentation available at the VESA website.

CRT-based VGA displays use amplitude-modulated moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by-pixel basis. Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays (so the "signals" discussion below pertains to both CRTs and LCDs). Color CRT displays use three electron beams (one for red, one for blue, and one for green) to energize the phosphor that coats the inner side of the display end of a cathode ray tube (see illustration). Electron beams emanate from "electron guns" which are finely-pointed heated cathodes placed in close proximity to a positively charged annular plate called a "grid". The electrostatic force imposed by the grid pulls rays of energized electrons from the cathodes, and those rays are fed by the current that flows into the cathodes. These particle rays are initially accelerated towards the grid, but they soon fall under the influence of the much larger electrostatic force that results from the

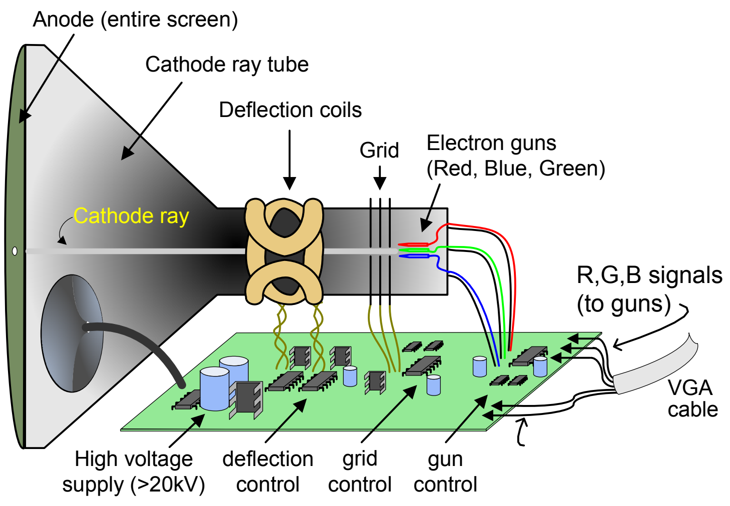


Figure 14. CRT deflection system

entire phosphor-coated display surface of the CRT being charged to 20kV (or more). The rays are focused to a fine beam as they pass through the center of the grids, and then they accelerate to impact on the phosphor-coated display surface. The phosphor surface glows brightly at the impact point, and it continues to glow for several hundred microseconds after the beam is removed. The larger the current fed into the cathode, the brighter the phosphor will glow.

Between the grid and the display surface, the beam passes through the neck of the CRT where two coils of wire produce orthogonal electromagnetic fields. Because cathode rays are composed of charged particles (electrons), they can be deflected by these magnetic fields. Current waveforms are passed through the coils to produce magnetic fields that interact with the cathode rays and cause them to transverse the display surface in a “raster” pattern, horizontally from left to right and vertically from top to bottom. As the cathode ray moves over the surface of the display, the current sent to the electron guns can be increased or decreased to change the brightness of the display at the cathode ray impact point.

Information is only displayed when the beam is moving in the “forward” direction (left to right and top to bottom), and not during the time the beam is reset back to the left or top edge of the display. Much of the potential display time is therefore lost in “blanking” periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass. The size of the beams, the frequency at which the beam can be traced across the display, and the frequency at which the electron beam can be modulated determine the display resolution. Modern VGA displays can accommodate different resolutions, and a VGA controller circuit dictates the resolution by producing timing signals to control the raster patterns. The controller must produce synchronizing pulses at 3.3V (or 5V) to set the frequency at which current flows through the deflection coils, and it must ensure that video data is applied to the electron guns at the correct time. Raster video displays define a number of “rows” that corresponds to the number of horizontal passes the cathode makes over the display area, and a number of “columns” that corresponds to an area on each row that is assigned to one “picture element” or pixel. Typical displays use from 240 to 1200 rows and from 320 to 1600 columns. The overall size of a display and the number of rows and columns determines the size of each pixel.

Video data typically comes from a video refresh memory, with one or more bytes assigned to each pixel location (the Basys2 uses three bits per pixel). The controller must index into video memory as the beams move across the display, and retrieve and apply video data to the display at precisely the time the electron beam is moving across a given pixel.

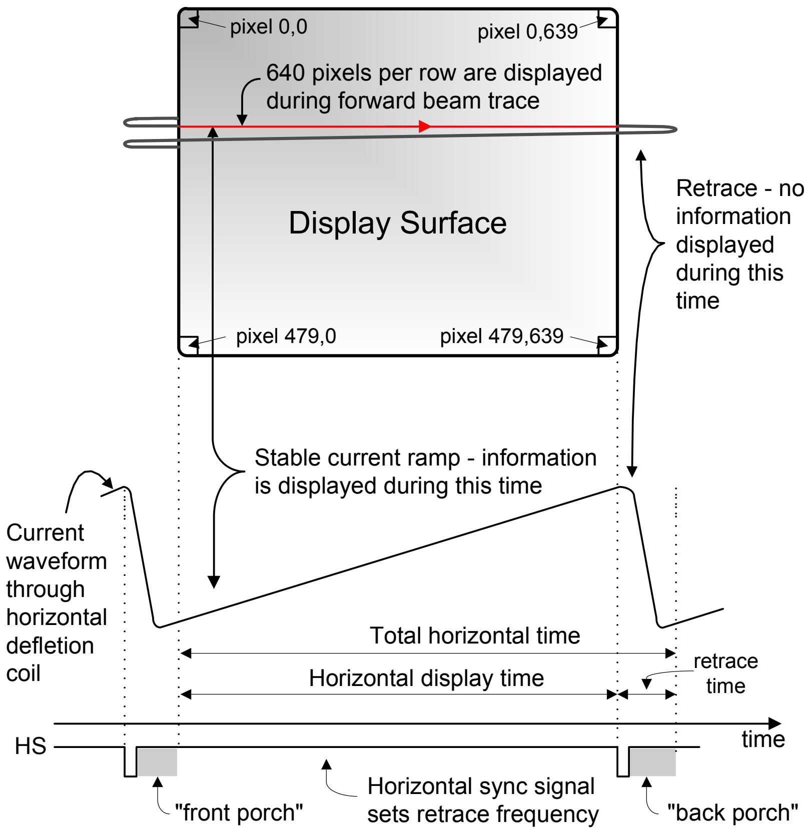
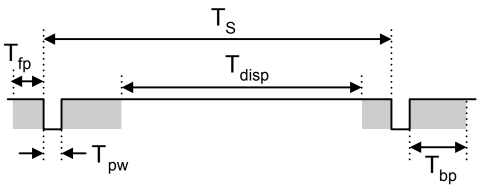

pixel 0,0
pixel 0,639
640 pixels per row are displayed during forward beam trace
Display Surface
Retrace - no information displayed during this time
pixel 479,0
pixel 479,639
Stable current ramp - information is displayed during this time
Current waveform through horizontal defletion coil
Total horizontal time
Horizontal display time
retrace time
HS
time
"front porch"
Horizontal sync signal sets retrace frequency
"back porch"

Figure 15. VGA system signals

A VGA controller circuit must generate the HS and VS timings signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the "refresh" frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display's phosphor and electron beam intensity, with practical refresh frequencies falling in the 50Hz to 120Hz range. The number of lines to be displayed at a given refresh frequency defines the horizontal "retrace" frequency. For a 640-pixel by 480-row display using a 25MHz pixel clock and 60 +/-1Hz refresh, the signal timings shown in the table at right can be derived. Timings for sync pulse width and front and back porch intervals (porch intervals are the pre- and post-sync pulse times during which information cannot be displayed) are based on observations taken from actual VGA displays.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 16. VGA system timings for 640x480 display

A VGA controller circuit decodes the output of a horizontal-sync counter driven by the pixel clock to generate HS signal timings. This counter can be used to locate any pixel location on a given row. Likewise, the output of a vertical-sync counter that increments with each HS pulse can be used to generate VS signal timings, and this counter can be used to locate any given row. These two continually running counters can be used to form an address into video RAM. No time relationship between the onset of the HS pulse and the onset of the VS pulse is specified, so the designer can arrange the counters to easily form video RAM addresses, or to minimize decoding logic for sync pulse generation.

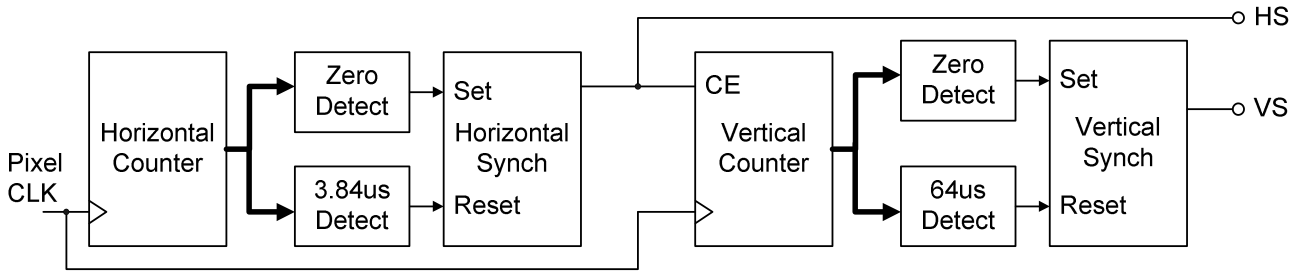


Figure 17. Schematic for a VGA controller circuit

Expansion Connectors (6-pin headers)

The Basys2 board provides four 6-pin peripheral module connectors. Each connector provides Vdd, GND, and four unique FPGA signals. Several 6-pin module boards that can attach to this connector are available from Digilent, including A/D converters, speaker amplifiers, microphones, H-bridge amplifiers, etc. Please see www.digilentinc.com for more information.

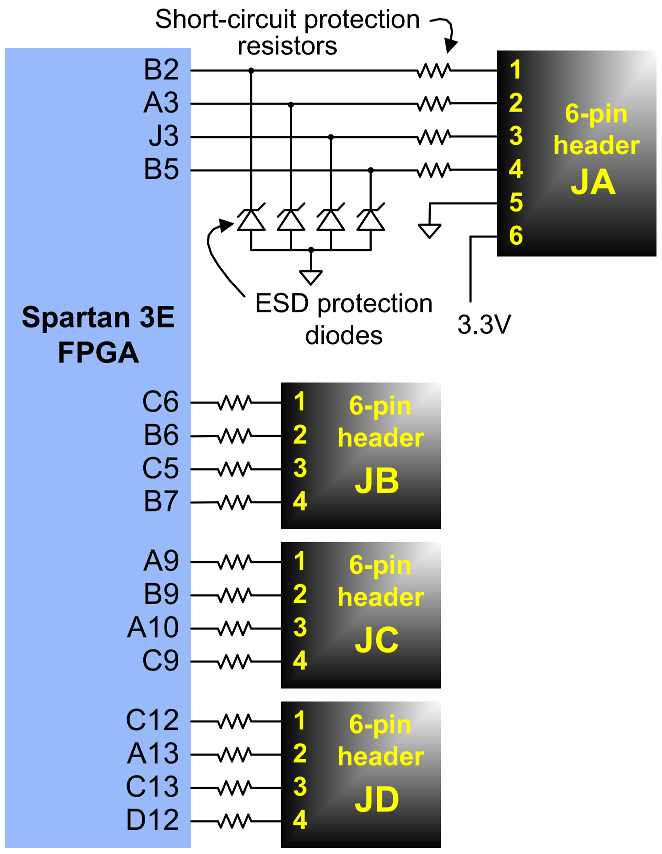


Figure 18. Basys2 Pmod connector circuits

FPGA Pin Definitions

The table below shows all pin definitions for the Spartan-3E on the Basys2 board. Pins in grey boxes are not available to the user

FPGA pin definition table color key		
Grey	▨	Not available to user
Green	🟩	User I/O devices
Yellow	🟨	Data ports
Tan	🟧	Pmod connector signals
Blue	🟦	USB signals

Basys2 Spartan-3E pin definitions

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
C12	JD1	P11	SW0	N14	CC	B2	JA1	P8	MODE0	M7	GND
A13	JD2	M2	USB-DB1	N13	DP	C2	USB-WRITE	N7	MODE1	P5	GND
A12	NC	N2	USB-DB0	M13	AN2	C3	PS2D	N6	MODE2	P10	GND
B12	NC	M9	NC	M12	CG	D1	NC	N12	CCLK	P14	GND
B11	NC	N9	NC	L14	CA	D2	USB-WAIT	P13	DONE	A6	VDDO-3
C11	BTN1	M10	NC	L13	CF	L2	USB-DB4	A1	PROG	B10	VDDO-3
C6	JB1	N10	NC	F13	RED2	L1	USB-DB3	N8	DIN	E13	VDDO-3
B6	JB2	M11	LD1	F14	GRN0	M1	USB-DB2	N1	INIT	M14	VDDO-3
C5	JB3	N11	CD	D12	JD4	L3	SW1	P1	NC	P3	VDDO-3
B5	JA4	P12	CE	D13	RED1	E2	SW6	B3	GND	M8	VDDO-3
C4	NC	N3	SW7	C13	JD3	F3	SW5	A4	GND	E1	VDDO-3
B4	SW3	M6	UCLK	C14	RED0	F2	USB-ASTB	A8	GND	J2	VDDO-3
A3	JA2	P6	LD3	G12	BTN0	F1	USB-DSTB	C1	GND	A5	VDDO-2
A10	JC3	P7	LD2	K14	AN2	G1	LD7	C7	GND	E12	VDDO-2
C9	JC4	M4	BTN2	J12	AN1	G3	SW4	C10	GND	K1	VDDO-2
B9	JC2	N4	LD5	J13	BLU2	H1	USB-DB6	E3	GND	P9	VDDO-2
A9	JC1	M5	LD0	J14	HSYNC	H2	USB-DB5	E14	GND	A11	VDDO-1
B8	MCLK	N5	LD4	H13	BLU1	H3	USB-DB7	G2	GND	D3	VDDO-1
C8	RCCLK	G14	GRN2	H12	CB	B14	TMS	H14	GND	D14	VDDO-1
A7	BTN3	G13	GRN1	J3	JA3	B13	TCK-FPGA	J1	GND	K2	VDDO-1
B7	JB4	F12	AN0	K3	SW2	A2	TDO-USB	K12	GND	L12	VDDO-1
P4	LD6	K13	VSYNC	B1	PS2C	A14	TDO-S3	M3	GND	P2	VDDO-1

Built in Self Test

The Basys2 board comes preloaded with a simple self test/demonstration project stored in its ROM. The demo project (available at the website) shows how the Xilinx CAD tools connect FPGA signals to Basys2 circuits. Since the project is stored in ROM, it can also be used to check board functions. To run the demo, set the ROM/USB jumper (JP3) to ROM and apply power to the board; the seven-segment display will show counting digits, the switches will turn on individual LEDs, the buttons will turn off individual digits on the seven segment display, and a test pattern is driven on the VGA port.

If the self test is not resident in the Platform Flash ROM, it can be programmed into the FPGA or reloaded into the ROM using the Adept programming software.