Introduction:

Slot machine is one of the most popular gaming medium in every casino. These machines have colorful bright lights and produce loud sounds which attract people. An example of a slot machine is shown on Figure 1. Slot machine is very easy to play. The user has only to press a button or pull a lever to initiate a game. In order for the user to win, the slot machine must show three similar symbols on the pay line which is set on the middle of the reels. These reels are like wheels which rotate when the machine is triggered and will stop at random location which affects the probability of the user winning a game.

Most modern slot machines use lcd monitors as display and have 5 reels on them. Furthermore, these machines have multiple pay lines and other features like extra game bonuses. A typical slot machine will cost for about $300 for the low-end ones and can
get up to $1000 for the high end ones [VEGAS10]. This price range is very expensive. But, there is an alternative way of manufacturing a slot machine that is more economic.

This project will focus on designing and building a slot machine that will cost less than a typical unit but will perform as well. The idea is to use an alternative embedded system or computer to implement the functionality of a slot machine. The embedded system that will be used for this project will be discussed further in the following sections. To give an overview, an embedded system is a computer that is designed and programmed to do specific functions. The program that runs an embedded system is called firmware and is typical coded in a high level programming language like C and Java. Figure 2 shows an example of an embedded system.

![Figure 2 - Embedded System by PlantAutomation](PLANTAUTO)
Background:

In this section, the Embedded System and the algorithm for implementing the slot machine will be discussed further.

**Embedded System:**

Embedded Systems are computers that are designed to perform a specific or various tasks. For example, a modern cellular phone, like the iPhone by Apple shown in Figure 3, is an embedded system that has multiple computers and each computer performs different function.

![iPhone by Apple](APPLE)

One of the computers handles the calling functionality of the phone. This particular computer decodes the incoming voice data and then sends the decoded voice to another computer which transforms the data into sound. Furthermore, another computer may serve as the MP3 player of the phone. This computer handles the music playing and organizes the play-lists and music files. On the other hand, another computer will handle
image processing because most modern cellular phones have cameras that can take photos or live video. Modern x-ray and MRI, which are shown in Figures 4a and 4b, are other examples of systems that use computer to process images. Embedded systems are versatile which can be used in many different applications.

Figure 4a and 4b - Digital X-ray (left picture) and MRI (right picture)

[FHM10] [IPOH10]

**Nexys 2:**

For this project, the Nexys 2 development board, manufactured by Digilent, is the embedded system that will be used for implementing the slot machine. Figure 5 shows an image of Nexys 2 and Table 1 gives the specification of Nexys 2. This development board uses the Xilinx Spartan 3e FPGA which is going to be programmed to mimic the behavior of a slot machine. In order to achieve this, a firmware will be programmed using the ISE version 9.1i web pack and Very High Speed Descriptive Language or VHDL, a programming language used in embedded systems.
Figure 5 - Nexys 2 Development Board

<table>
<thead>
<tr>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price: $99.00</td>
</tr>
<tr>
<td>Xilinx Spartan-3E FPGA, 500K or 1200K gate</td>
</tr>
<tr>
<td>USB2 port providing board power, device configuration, and high-speed data transfers Works with ISE/Webpack and EDK</td>
</tr>
<tr>
<td>16MB fast Micron PSDRAM</td>
</tr>
<tr>
<td>16MB Intel StrataFlash Flash R</td>
</tr>
<tr>
<td>Xilinx Platform Flash ROM</td>
</tr>
<tr>
<td>High-efficiency switching power supplies (good for battery-powered applications)</td>
</tr>
<tr>
<td>50MHz oscillator, plus a socket for a second oscillator</td>
</tr>
<tr>
<td>75 FPGA I/O’s routed to expansion connectors (one high-speed Hirose FX2 connector with 43 signals and four 2x6 Pmod connectors)</td>
</tr>
<tr>
<td>All I/O signals are ESD and short-circuit protected, ensuring a long operating life in any environment.</td>
</tr>
<tr>
<td>On-board I/O includes eight LEDs, four-digit seven-segment display, four pushbuttons, eight slide switches</td>
</tr>
</tbody>
</table>

Table 1 - Nexys 2 Specification

[DIGILENT]
Additional Hardware:

Display Monitor – will display the reels of the slot machine. Figure 6 shows the image of the display monitor and Table 2 shows its specification.

![Display Monitor Image](Newegg10)

<table>
<thead>
<tr>
<th>Manufactured by Samsung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price: $159.99</td>
</tr>
<tr>
<td>Model: P2050</td>
</tr>
<tr>
<td>Color: Black</td>
</tr>
<tr>
<td>Resolution: 1600x900</td>
</tr>
<tr>
<td>Brightness: 300 cd/m2</td>
</tr>
<tr>
<td>Contrast: DC 50000:1</td>
</tr>
<tr>
<td>Response time: 2ms</td>
</tr>
<tr>
<td>Connectors: D-Sub, DVI-D</td>
</tr>
</tbody>
</table>

Table 2 - Specification for the display monitor

![Keyboard Image](Newegg10)

Keyboard – will serve as the input device of the slot machine. Figure 7 shows the image of the keyboard and Table 3 shows its specification.

![Keyboard Image](Newegg10)

Table 3 - Specification for the keyboard
Algorithm:

After doing some research, slot machines use *pseudorandom numbers* that are mapped to the symbols on the reels. Pseudorandom numbers are not actually random numbers. They are generated by using formulas which take seed numbers and produce different numbers based on these seed numbers. The randomness of the numbers will depend on the formulas used and the seed or input numbers. To increase the randomness of the numbers, the resulting pseudorandom numbers can be used as input to another formula which in turn produces new pseudorandom numbers.

For slot machines, the Lehmer’s congruential iteration [TURNER04] is used and the algorithm for the formula is as follows:

**Given:** three very large numbers
- a multiplier (a)
- an added number (b)
- a divider/modulus (m)

**Steps:**
1. Start with a seed number, for example, time of day.
2. Multiply it by one number (a) and add another number (b)
3. Divide by the modulus (m)
4. The remainder is the first random number
5. Translate the remainder in a useful range, for example, 0 to 1, 1 to 128, etc.
6. Use the remainder as the seed for the next number

[TURNER04]
When various pseudorandom numbers are generated, different symbols will be mapped to each of these numbers. For example, when the machine generated a number between the ranges of 0-10, a cherry symbol will show on the reels. If the generated number is in between 20-30, the letter ‘A’ will be displayed on the reels. Figure 4 shows an example of how mapping symbols is done on a slot machine.

Figure 8 - Example of mapping symbols in a slot machine [TURNER04]
Description:

Price Comparison:

<table>
<thead>
<tr>
<th>Type</th>
<th>Price</th>
<th>Type</th>
<th>Materials</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Slot Machine</td>
<td>$300</td>
<td>Nexys 2 Slot Machine</td>
<td>Nexys 2 Development Board</td>
<td>$99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LCD Monitor</td>
<td>$159.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Keyboard</td>
<td>$14.99</td>
</tr>
<tr>
<td>Total</td>
<td>$300</td>
<td></td>
<td></td>
<td>$273.98</td>
</tr>
</tbody>
</table>

Table 4 - Price comparison

If the specified materials are used to build the alternative slot machine, the typical low-end slot machine cost more. The price difference between the Nexys 2 slot machine and the typical low-end slot machine is about $27. The cost to manufacture the Nexys 2 slot machine is about 9% less than the usual slot machine. In terms of price, the Nexys 2 slot machine will be able to compete with other slot machines in the market.

Implementation:

![Figure 9 - Top level block diagram of Nexys 2 slot machine](image)
Figure 9 shows the top level block diagram of the Nexys 2 slot machine. It has to inputs which are the clock and the trigger. The clock signal, which is supplied by the onboard oscillator, produces 50Mhz of frequency. The clock will be used to synchronize the modules of the slot machine. On the other hand, the input signal will be coming from the button labeled “BTN3” on the Nexys 2 development board. When “BTN3” is asserted, it will initiate the start of the game and when released, the reels will stop and display the corresponding symbols. Meanwhile, the outputs of the system are 8-bit display output, HS, VS, and LED. The 8-bit display output contains the data that will be displayed on the monitor. These bits are RGB bits which are responsible for displaying colors and images onto the monitor. The three most significant bits correspond to the color red. The next three bits correspond to the color green and the last 2 bits correspond to the color blue. Different bit patterns will display different colors and also with different depths. The signals HS, horizontal sync, and VS, vertical sync, are responsible for telling the monitor when and where to write a new frame. HS gives the monitor the location of the line that needs to be written. VS will tell the monitor to start writing a new frame, starting from upper left corner of the monitor. The resolution of the display will be 640x480 so; there will be 640 horizontal lines and 480 vertical lines. The 8-bit display output, HS, and VS are sent to the VGA port of the Nexys 2 development board which connected to the display monitor. Lastly, the LED output signal is just an indicator that tells whether the system is on or off. The LED signal is mapped to the led labeled “LD0” on the development board. If the “LD0” is lit up, the system is on and off, if it isn’t.
The next level block diagram includes the mapping, clock divider, random generator, and Synchro. This level is shown on Figure 10. The structure for the Nexys 2 slot machine is modular because the clock divider, random generator, and Synchro have their own entities and are defined in separate projects. Meanwhile, the mapping section is defined in this level.

The block diagram for the mapping module is shown on Figure 11. The input signals used for this module are CLK, 3 Bit Random Number, Trigger, and 10 Bit
HCount and VCount. As discussed above, the CLK and Trigger signals are coming from onboard oscillator, which is 50Mhz, and “BTN3” of the development board, respectively. The 3 Bit Random Number is coming from randomGenerator module which will be explained later. The 3 Bit Random Number signal is going to be used to map symbols for the reels. If it falls in the range from 0 to 2, inclusively, a rectangle shape will be mapped on the reels. If it falls in the range from 3 to 5, inclusively, a square shape will be displayed on the reels. And then, if the number is either 6 or 7, a letter “T” will be mapped and shown on the reels.

In order to display the right symbol on the right position, the 10 Bit HCount and VCount signals are needed. These signals indicate the current pixel that is being drawn onto the monitor and they are converted to integers so that the different regions of the slot machine can be accessed easily. If HCount < 640 and VCount < 480, the slot machine is being drawn and displayed on the monitor. If HCount is in the range from 110 to 530 and VCount is in between 105 and 375, the reels are being displayed. If HCount is in the range from 110 to 530 and VCount is in between 195 and 285, the symbols are drawn onto the monitor. Depending on the current location of HCount and VCount, different colors will be used in order to distinguish the different areas of the slot machine.

The 8 Bit Display Output signal relies on the region that is currently being drawn. For example, if the current region is in the reels area, the 8 Bit Display Output signal is set to “11100000” which correspond to the color red. If rectangle shape is being drawn, the display signal is set to “00000011” which indicates blue. If square shape is being drawn, the 8 Bit Display Output signal is set to “00011100” which is the color green. If it’s the letter “T”, the output signal is set to “00011111” which is cyan. If it is not in the
reels region, the 8 Bit Display Output is set to “11111111” which is the color puke. (See mapping in the Appendix for the source code of the mapping module.)

Figure 12 - Block diagram of the Clock Divider module

The block diagram for the Clock Divider module is shown in Figure 12. The input is a clock signal produced by the onboard oscillator which generates 50MHz of frequency. The goal of the Clock Divider is to drop the 50MHz of frequency to 25MHz which is needed by the Synchro module in order to create the appropriate sync signals. In order to achieve 25MHz, the Clock Divider uses the following algorithm: on every rising edge of the standard clock (50MHz), a signal is set to its inverse. The resulting signal is the new clock signal which has 25MHz of frequency. (See Clock Divider in the Appendix for the source code of this module.)

Figure 13 - Block diagram for the randomGenerator module
The block diagram for the randomGenerator module is shown above. The module takes in the standard clock which is 50MHz of frequency and produces a 3 Bit Random Number. There are three randomGenerator modules in this system, in order to increase the variety of the random numbers. The three modules use similar algorithm which is different from the Lehmer’s congruential iteration. Furthermore, the three randomGenerator modules use different bits in computing the random number. The algorithm used is as follows: XOR two bits together and use the resulting bit as the first bit. And then, shift the other two bits either to the right or to the left. The first randomGenerator XORS bit 2 and bit together and stores the resulting bit into bit 0. Then, it shifts bits 1 and 0 to bits 2 and 1. The second randomGenerator XORS bit 2 and bit 0 together and stores the resulting bit to bit 0. It also shifts bits 1 and 0 to bits 2 and 1. Lastly, the third randomGenerator XORS bit 2 and bit 0 together and stores it to bit 2. Then, it shifts bits 2 and 1 to bits 1 and 0. They all start with “100”. The code for this module is based on the code found in VHDL Guru website [VHDL]. (See randomGenerator in Appendix for the source code of this module.)

![Figure 14 - Block diagram for Synchro module](image)

The block diagram for the Synchro module is shown in Figure 14. It uses the 25 MHz clock as an input which is needed to produce a 640x480 resolution. On the other
hand, HS, VS, HCount, and VCount are the outputs which are used to display images on
the monitor. HS and VS are sent to the VGA port of the development board while
HCount and VCount are used for mapping symbol. The definition for this module is
taken from the Nexys 2 demo created by Digilent [DIGILENT]. (See Synchro in the
Appendix for the source code of this module.)
Evaluation:

Figure 15 shows the set-up for the Nexys 2 slot machine. In this configuration, a 24 inch LCD monitor is used for displaying the slot machine. The Nexys 2 is powered by the USB connected to the laptop. The USB is also used to upload the firmware into the FPGA of the Nexys 2 development board with the help of the ExPort software provided by Digilent.

After the software is uploaded, the slot machine will be initialized so that it will be ready to be used. Figure 16 below shows the initial state of the system which also displays that the reels will start with rectangle symbols on them. To start a game just press the button labeled “BTN3.” Figure 17 shows the reels having different symbols and Figure 18 shows a winning combination of the symbols.
Figure 16 - Image of the initial state of the Nexys 2 slot machine

Figure 17 - Image of the Nexys 2 slot machine in action
Figure 18 - Image of a winning combination of the reels

After playing the Nexys 2 slot machine for awhile, the reels displays the symbols in a pattern. The same symbol combination will be seen after playing the machine for a few times. The cause of this behavior is the numbers produced by the randomGenerator repeat after playing the machine several times. Another behavior of the system, when the user holds “BTN3” the reels will keep on triggering and flashing the symbols. On the other hand, the Nexys 2 slot machine performed as expected, overall.
Conclusion:

The Nexys 2 slot machine was able to mimic the basic functionality of a typical slot machine. In terms of pricing, the Nexys 2 slot machine has the potential to compete with other slot machines in the market. In terms of performance, the Nexys 2 slot machine also has the potential to compete with others. The current design of the system can still be improved and be able to out perform the competition.

Improvements:

randomGenerator – increase the range and randomness of the random numbers by improving or changing the algorithm that is currently in use so that the symbol combination will not repeat.

“BTN3” – change the algorithm so that when the user holds the button it will not keep the reels triggering.

images – upload real images. The Nexys 2 has the ability to stream good quality images. These images can replace the current symbols. Having these images will make the Nexys 2 slot machine more realistic.

sound system – a sound amplifier module can be integrated to the Nexys 2 development board which can be used to produce slot machine sounds, adding a realistic feel to the system.

MicroBlaze – redesign the system using the Xilinx EDK which provides additional power and flexibility to the development of the hardware and firmware.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity monitorDemo is
Port ( clk : in  STD_LOGIC;
       input : in std_logic;
       led : out  STD_LOGIC;
       uHS : out std_logic;
       uVS : out std_logic;
       pixel : out std_logic_vector (7 downto 0));
end monitorDemo;

architecture Behavioral of monitorDemo is

component clk25Mhz
Port ( clk : in  STD_LOGIC;
       sclk: out  STD_LOGIC);
end component;

component Synchro
port ( clk25MHz: in std_logic; -- ck 25MHz
       HS: out std_logic; -- horizontal synchro signal
       VS: out std_logic; -- vertical synchro signal
       Hcnt: out std_logic_vector (10 downto 0); -- horizontal counter
       Vcnt: out std_logic_vector (10 downto 0) -- vertical counter
       );
end component;

component randomGen
Port ( clk : in  STD_LOGIC;
       randomNum : out  STD_LOGIC_VECTOR (2 downto 0));
end component;

component randomGen2 is
Port ( clk2 : in  STD_LOGIC;
       randomNum2 : out  STD_LOGIC_VECTOR (2 downto 0));
end component;
randomNum2 : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component randomGen3 is
Port ( clk3 : in  STD_LOGIC;
        randomNum3 : out  STD_LOGIC_VECTOR (2 downto 0));
end component;

component getSymbol
Port ( randomNumber : in  STD_LOGIC_VECTOR (2 downto 0);
       outpix : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

constant PAL:integer:=640;  --Pixels/Active Line (pixels)
constant LAF:integer:=480;  --Lines/Active Frame (lines)
constant UP:integer:=105;
constant DOWN:integer:=375;
constant LEFT:integer:=110;
constant RIGHT:integer:=530;
constant FIRST:integer:=250;
constant SECOND:integer:=390;
constant UPPER:integer:=195;
constant LOWER:integer:=285;

signal slow_clk : std_logic;
signal rand : std_logic_vector(2 downto 0);
signal rand2 : std_logic_vector(2 downto 0);
signal rand3 : std_logic_vector(2 downto 0);
signal randN : std_logic_vector(2 downto 0);
signal randN2 : std_logic_vector(2 downto 0);
signal randN3 : std_logic_vector(2 downto 0);
signal randN : std_logic_vector(2 downto 0);
signal out1 : std_logic_vector (7 downto 0);
signal out2 : std_logic_vector (7 downto 0);
signal out3 : std_logic_vector (7 downto 0);
signal pixout1 : std_logic_vector (7 downto 0);
signal pixout2 : std_logic_vector (7 downto 0);
signal pixout3 : std_logic_vector (7 downto 0);
signal hcount : std_logic_vector (10 downto 0);
signal vcount : std_logic_vector (10 downto 0);
signal intHcnt: integer range 0 to 800-1; --PLD-1 - horizontal counter
signal intVcnt: integer range 0 to 521-1; -- LFD-1 - verical counter
signal random1: integer range 0 to 7;
signal random2: integer range 0 to 7;
signal random3: integer range 0 to 7;

begin

M1 : clk25Mhz port map (clk, slow_clk);
M2 : Synchro port map (slow_clk, uHS, uVS, hcount, vcount);
M3 : randomGen port map (clk, rand);
M4 : randomGen2 port map (clk, rand2);
M5 : randomGen3 port map (clk, rand3);

process(input)
begin
if input = '1' then
  random1 <= conv_integer(rand);
  random2 <= conv_integer(rand2);
  random3 <= conv_integer(rand3);
  randN <= rand;
  randN2 <= rand2;
  randN3 <= rand3;
else
  randN <= randN;
  randN2 <= randN2;
end if;
end process;

randN3 <= randN3;
random1 <= random1;
random2 <= random2;
random3 <= random3;

end if;
end process;

--M6 : getSymbol port map (randN, out1);
--
--M7 : getSymbol port map (randN2, out2);
--
--M8 : getSymbol port map (randN3, out3);

intHcnt <= conv_integer(hcount);
intVcnt <= conv_integer(vcount);

process(intHcnt, intVcnt, input)
begin
if intHcnt < PAL and intVcnt < LAF then
if intHcnt > LEFT and intHcnt < RIGHT then
if intVcnt > UP and intVcnt < DOWN then
-- reels' area
if intHcnt = FIRST or intHcnt = SECOND then
pixel <= "11111111";
else
if intHcnt < FIRST then
-- individual area
if intVcnt > UPPER and intVcnt < LOWER then
-- symbol goes here
if random1 >= 0 and random1 <= 2 then
-- rectangle
if intHcnt > LEFT+20 and intHcnt < FIRST-20 then
if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
pixel <= "00000011";
else
pixel <= "11100000";
end if;
else
pixel <= "11100000";
end if;
else
if intHcnt >= 3 and random1 <= 5 then
-- square
if intHcnt > LEFT+35 and intHcnt < FIRST-35 then
if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
pixel <= "000111100";
else
pixel <= "11100000";
end if;
else
pixel <= "11100000";
end if;
else
if intVcnt > UPPER+10 and intVcnt < LOWER-50 then
if pixel <= "000111100";
else
pixel <= "11100000";
end if;
else
if intVcnt > UPPER+10 and intVcnt < LOWER-50 then
if pixel <= "000111100";
else
pixel <= "11100000";
end if;
else
pixel <= "11100000";
end if;
else
pixel <= "11100000";
end if;
else
if intVcnt > UPPER+10 and intVcnt < LOWER-50 then
if pixel <= "000111100";
else
pixel <= "11100000";
end if;
else
pixel <= "11100000";
end if;
else
pixel <= "11100000";
end if;
else
  pixel := "11100000";
  end if;
else
  pixel := out1;
end if;
else
  pixel := "11100000";
end if;
elsif intHcnt < SECOND then
  -- individual area
  if intVcnt > UPPER and intVcnt < LOWER then
    -- symbol goes here
    if random2 >= 0 and random2 <= 2 then
      -- rectangle
      if intHcnt > FIRST+20 and intHcnt < SECOND-20 then
        if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
          pixel := "00000011";
        else
          pixel := "11100000";
        end if;
      else
        pixel := "11100000";
      end if;
    elsif random2 >= 3 and random2 <= 5 then
      -- square
      if intHcnt > FIRST+35 and intHcnt < SECOND-35 then
        if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
          pixel := "00011100";
        else
          pixel := "11100000";
        end if;
      else
        pixel := "11100000";
      end if;
    elsif random2 = 6 or random2 = 7 then
      -- triangle
      if intHcnt > FIRST+35 and intHcnt < SECOND-35 then
        if intHcnt > FIRST+50 and intHcnt < SECOND-50 then
          if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
            pixel := "00011111";
          else
            pixel := "11100000";
          end if;
        else
          if intVcnt > UPPER+10 and intVcnt < LOWER-50 then
            pixel := "00011111";
          else
            pixel := "11100000";
          end if;
        end if;
      else
        pixel := "11100000";
      end if;
    else
      pixel := "11100000";
    end if;
  else
    pixel := "11100000";
  end if;
elsif random2 >= 3 and random2 <= 5 then
  -- symbol goes here
  if random3 >= 0 and random3 <= 2 then
    -- rectangle
    if intHcnt > FIRST+20 and intHcnt < SECOND-20 then
      if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
        pixel := "00000011";
      else
        pixel := "11100000";
      end if;
    else
      pixel := "11100000";
    end if;
  else
    pixel := "11100000";
  end if;
else
  pixel := "11100000";
end if;
pixel <= "11100000";
end if;
else
pixel <= "11111111";
end if;
else
pixel <= "11100000";
end if;
elif random3 >= 3 and random3 <= 5 then
  --square
  if intHcnt > SECOND+35 and intHcnt < RIGHT-35 then
    if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
      pixel <= "00011100";
    else
      pixel <= "11100000";
    end if;
  else
    pixel <= "11100000";
  end if;
elif random3 = 6 or random3 = 7 then
  --triangle
  if intHcnt > SECOND+35 and intHcnt < RIGHT-35 then
    if intHcnt > SECOND+50 and intHcnt < RIGHT-50 then
      if intVcnt > UPPER+10 and intVcnt < LOWER-10 then
        pixel <= "00011111";
      else
        pixel <= "11100000";
      end if;
    else
      if intVcnt > UPPER+10 and intVcnt < LOWER-50 then
        pixel <= "00011111";
      else
        pixel <= "11100000";
      end if;
    end if;
  else
    if intVcnt > UPPER+10 and intVcnt < LOWER-50 then
      pixel <= "00011111";
    else
      pixel <= "11100000";
    end if;
  end if;
end if;
end if;
end if;
end if;
end if;
elif intHcnt < PAL and intVcnt < LAF then
led <= '1';
ellse
led <= '0';
end Behavioral;
Clock Divider:

-- Company:
-- Engineer:
-- Create Date: 12:47:51 05/20/2010
-- Design Name:
-- Module Name: clk25Mhz - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity clk25Mhz is
  Port ( clk : in  STD_LOGIC;
         sclk : out  STD_LOGIC);
end clk25Mhz;

architecture Behavioral of clk25Mhz is
signal slow_clk : std_logic;

begin
  process(clk)
  variable count : Integer := 0;
  begin
    if rising_edge(clk) then
      if (count = 0) then
        slow_clk <= not slow_clk;
        count := 0;
      else
        count := count + 1;
      end if;
    end if;
  end process;
  sclk <= slow_clk;
end Behavioral;
Random Generator:
----------------------------------------------------------------------------------
-- Company: 
-- Engineer: 
-- Create Date:  20:39:37 06/04/2010
-- Design Name: 
-- Module Name: randomGen - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision: 
-- Revision 0.01 - File Created
-- Additional Comments: 
-- 
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity randomGen is
  Port ( clk : in  STD_LOGIC;
         randomNum : out  STD_LOGIC_VECTOR (2 downto 0));
end randomGen;

architecture Behavioral of randomGen is
begin
  process(clk)
  variable rand_temp : std_logic_vector(2 downto 0):=(2 => '1',others => '0');
  variable temp : std_logic := '0';
  begin
    if(rising_edge(clk)) then
      temp := rand_temp(2) xor rand_temp(1);
      rand_temp(2 downto 1) := rand_temp(1 downto 0);
      rand_temp(0) := temp;
    end if;
    randomNum <= rand_temp;
  end process;

end Behavioral;
Synchro:

-- synchro.vhd --
-- Author : Mircea Dabacan
-- Copyright 2005 Digilent, Inc.
-- This is the source file for the synchro component,
-- provided by the Digilent Reference Component Library.

-- Revision History:
-- 15/01/2005(MirceaD): created
-- 2/12/2006 (MirceaD): modified for std_logoc_vector Hcnt and Vcnt

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Synchro is
  port (
    ck25MHz: in std_logic;       -- ck
    25MHz
    HS: out std_logic;
    -- horizontal synchro signal
    VS: out std_logic;
    -- verical synchro signal
    Hcnt: out std_logic_vector (10 downto 0); -- horizontal counter
    Vcnt: out std_logic_vector (10 downto 0) -- verical counter
  );
end Synchro;

architecture Behavioral of Synchro is
  -- constants for Synchro module
  constant HPW:integer:=96;  --Horizontal synchro Pulse Width (pixels)
  constant HFP:integer:=16;  --Horizontal synchro Front Porch (pixels)
  constant HBP:integer:=48;  --Horizontal synchro Back Porch (pixels)
  constant VPW:integer:=2;  --Verical synchro Pulse Width (lines)
  constant VFP:integer:=10;  --Verical synchro Front Porch (lines)
  constant VBP:integer:=29;  --Verical synchro Back Porch (lines)
  constant PAL:integer:=640;  --Pixels/Active Line (pixels)
  constant LAF:integer:=480;  --Lines/Active Frame (lines)
  constant PLD: integer:=800; --Pixel/Line Divider
  constant LFD: integer:=521; --Line/Frame Divider

  signal intHcnt: integer range 0 to PLD-1; -- horizontal counter
  signal intVcnt: integer range 0 to LFD-1; -- verical counter

begin
  syncro: process (ck25MHz)
  begin
if ck25MHz'event and ck25MHz='1' then
  if intHcnt=PLD-1 then
    intHcnt<=0;
    if intVcnt=LFD-1 then intVcnt<=0;
    else intVcnt<=intVcnt+1;
  end if;
  else intHcnt<=intHcnt+1;
end if;

-- Generates HS - active low
if intHcnt=PAL-1+HFP then
  HS<='0';
elsif intHcnt=PAL-1+HFP+HPW then
  HS<='1';
end if;

-- Generates VS - active low
if intVcnt=LAF-1+VFP then
  VS<='0';
elsif intVcnt=LAF-1+VFP+VPW then
  VS<='1';
end if;
end if;
end process;

-- mapping internal integers to std_logic_vector ports
Hcnt <= conv_std_logic_vector(intHcnt,11);
Vcnt <= conv_std_logic_vector(intVcnt,11);

end Behavioral;
Bibliography


- Picture for an iPhone


- Image and specification for Nexys 2
- Source code for the Synchro module


- This book gives an overview about the history of slot machines.


- Picture for a digital X-ray


- This book discusses how State Machines behave and how it can be implemented on Embedded Systems. The book also shows some applications of Embedded State Machines.


- This website illustrates the basic parts and functionalities of slot machines. It also describes the object of the game and the payout.


- Picture of a MRI


- Price, specification, and image for the keyboard.

- Price, specification, and image for the LCD monitor


- This book teaches readers how to start a design process from the top-level model up to the production of the prototype. The design process uses different methods such as real-time analysis and modeling and validation in order to achieve the desired outcome.

[PLANTAUTO] PlantAutomation. 29 April 2010
http://www.plantautomation-technology.com/contractors/motors/js/

- Picture for an embedded system


- This article explains how slot machines operate. This article also shows some tips on how to beat slot machines. Lastly, this article describes how slot machines can be addicting to users.


- Price list for slot machines


- Random Number Generator