Button Hero

Final Project Report

Team Members

Joo Han Chang [jc2685@columbia.edu]
Charles Lam [cgl201@columbia.edu]
George Liao [gkl2104@columbia.edu]
Kenneth Yu [khy2102@columbia.edu]

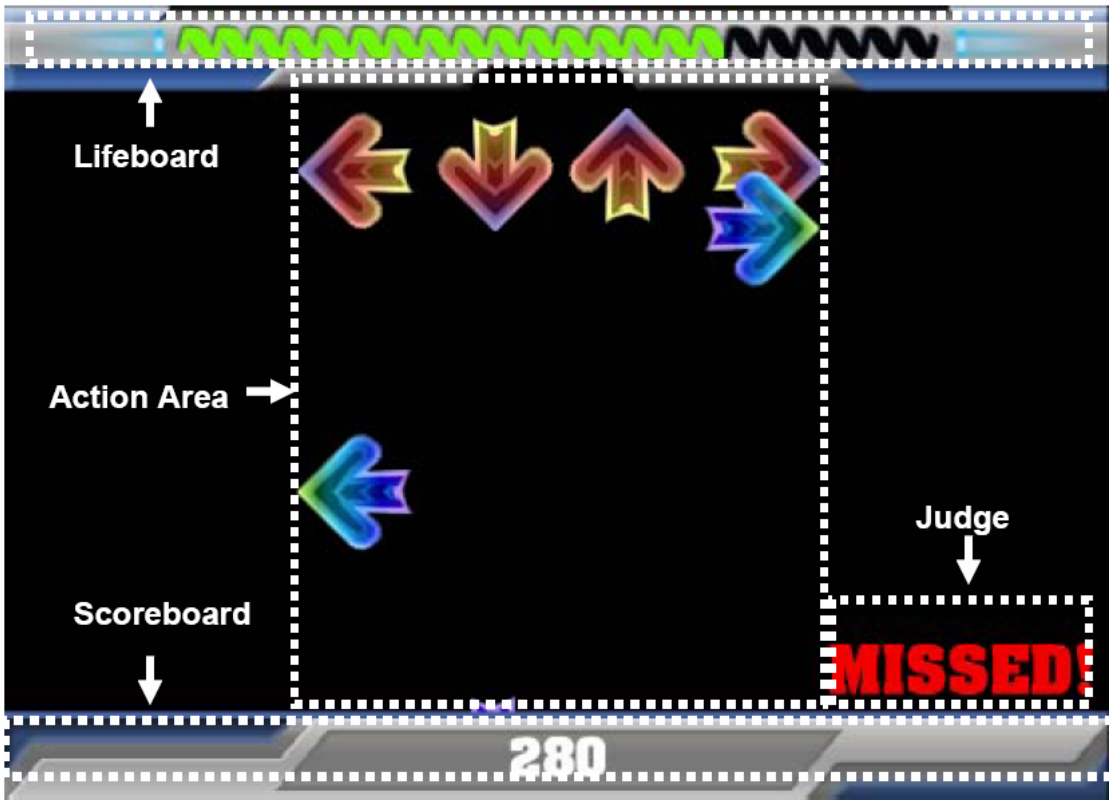
Table of Contents

1. OVERVIEW / MOTIVATION

Our group created an interactive game named “Button Hero” for the Spring 2007 CSEE 4840 class with Professor Stephen Edwards. This game was compiled and executed using the Quartus Cyclone II FPGA board. Our initial objective was to try and recreate the game “Guitar Hero” which was made popular by the Sony Playstation 2. An online version of the game can be found at the following link:

http://www.youtube.com/watch?v=cueXmJDbbvU

While the link shows the game “Guitar Hero”, a closer reference can actually be found in online flash-based DDR games. The following is an example screenshot of one:



Our game interface has the same “note” setup as this game, except our four buttons are lined up horizontally on the bottom center of the screen. These buttons correspond to the four push-buttons on the FPGA board. The colors were chosen arbitrarily with emphasis on being able to distinguish between them easily. The life bar, which goes up when the user pushes the correct notes and goes down when the user misses or pushes the incorrect notes, is situated on the left hand side of the screen. This is as opposed to the DDR game above, where the life bar is situated at the center top of the screen.

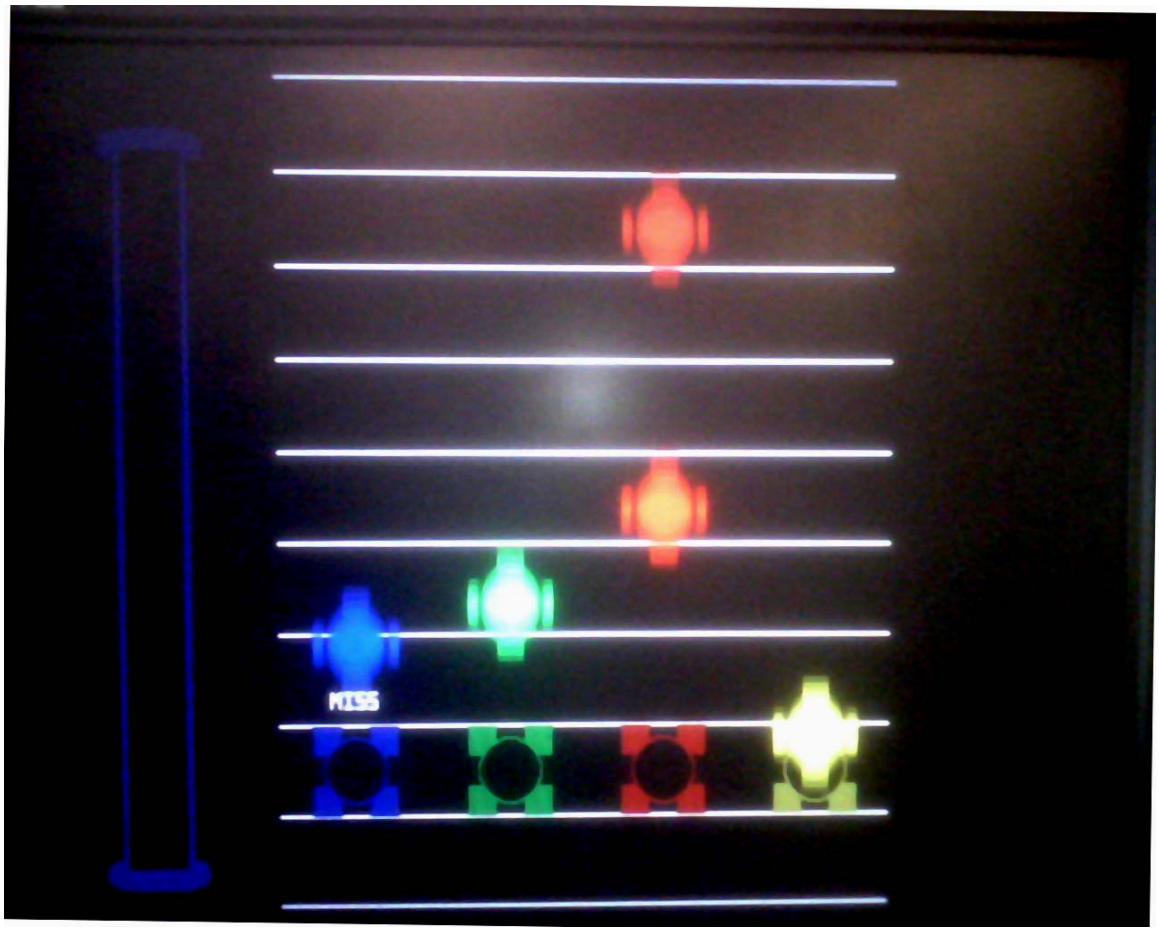
Meanwhile, the “Perfect”, “Good”, “Bad”, and “Miss” designations for each note that streams down appears right above the four anchor buttons on the bottom. There are four

of these accuracy text positions, one for each of the four buttons. We decided to do it this way as opposed to having one location displaying all the texts because the user can easily tell which button he/she pushed correctly or incorrectly.

Also, we created a start screen and an end screen to both greet the user and to signify the end of the game. The start screen prompts the user to push any button to start the game, while the end screen simply displays “game over”. Both of these messages are displayed at the center of the game interface.

We decided to add horizontal white lines to the section where the notes will stream down on because it adds to the musical feel of the program. Finally, we also used several components on the FPGA to display game information. The three right-most 7-segment displays keep track of the current score of the game, which is an integer between 0 and 100. The score will not be reset until the user starts a new game. Also, the LED’s are used to show the longest “Perfect” streak that the user has been able to maintain during the current game. This is also not reset until the user starts a new game.

The following is a sample screenshot of our game during execution. The accuracy text on the very left button is displaying “MISS”:



As for gameplay, the timing and the positioning of the notes for the song are all done by the C software. We created an excel script that allows us to generate a song that does not violate any of the hardware restrictions of our program. These restrictions include a maximum number of 8 notes at one time on the screen, as well as overlapping notes.

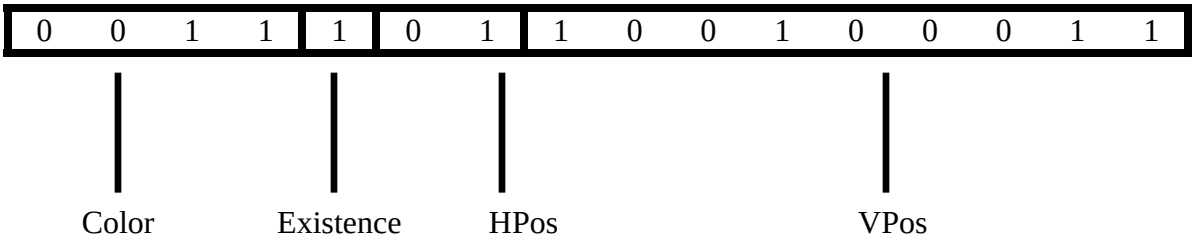
The game begins when the user presses a button when prompted by the start screen. Initial score is set to 50, which can be verified both by the 7-segment displays and the life bar. As the notes begin to stream down the screen, the user will need to push the correct button at the correct time in order to increase his score. A “perfect” will net him/her 4 points, a “good” 2 points, a “bad” -8 points, and a “miss” -10 points. The game will not stop until the full song has been played. We decided to let the game continue to run even if the user’s score is 0 because it gives the user another chance to catch up.

Overall, the inner workings of our program consist of two major components: the VHDL hardware and the C software. The hardware utilizes a VGA raster to display elements onto the monitor. The majority of the gameplay controls, however, is done by the software. Data is constantly being transferred from the software to the hardware, including the score, perfect streak, accuracy text, button position, and button anchor positions. The hardware then takes this information and outputs the data either on the VGA monitor or the FPGA board.

The following section will describe our program design in much more detail.

2. GRAPHICS ARCHITECTURE

2.a. 16-Bit "Falling Button" Packet



- **Color**

In choosing a coloring scheme, we first decided what level of accuracy is needed for the purpose of this project. We realized that the game only required around ten colors – (4 colors for falling buttons in 4 different lanes, 4 for button catchers in 4 different lanes, and black/white) which could be realized by a 16-color scheme, which is actually commonly used. (standard colors for HTML, for example)

This allowed to represent a color in just 4 bits of data, which was extremely convenient for the use of the 16-Bit packet shown above; in fact, 4 bits for color was the most we could use if we were to represent a falling block in a single 16-Bit packet. This, however, meant that we needed to hard code the respective RGB values for each of the 16 colors in hardware since passing the exact RGB values would have required 24 to 30 bits.

For a selection of the colors and their exact RGB values, we visited http://msdn2.microsoft.com/en-us/library/ms531197.aspx and used the hex values from there. An issue we faced was that the VGA interface we were using employed a 30-bit coloring scheme, rather than the usual 24-bit whose color table was available online. We resolved the issue by a manual conversion.

Map Within Hardware:

```
  constant colors : color :=
   ("00000000000000000000000000000000","00000000000000000000001000000000
","00000000000000000000001111111111","00000000001000000000000000000000",
   "00000000001000000000001000000000","00000000001111111111000000000000"
,"00000000001111111111111111111111","10000000000000000000000000000000",
   "10000000000000000000001000000000","10000000001000000000000000000000"
,"10000000001000000000001000000000","11000000001100000000011000000000",
   "11111111110000000000000000000000","11111111110000000000011111111111"
,"11111111111111111111000000000000","11111111111111111111111111111111");
```

Map Within Software:

```
//Color Map - 16
#define WHITE 0x0000
#define YELLOW 0x1000
#define FUCHSIA 0x2000
#define RED 0x3000
#define SILVER 0x4000
#define GRAY 0x5000
#define OLIVE 0x6000
#define PURPLE 0x7000
#define MAROON 0x8000
#define AQUA 0x9000
#define LIME 0xA000
#define TEAL 0xB000
#define GREEN 0xC000
#define BLUE 0xD000
#define NAVY 0XE000
#define BLACK 0XF000
```

Color Name	R(10) | G(10) | B(10)	Color
White	11111111111111111111111111111111	
Yellow	11111111111111111110000000000	
Fuchsia	11111111110000000001111111111	
Red	11111111110000000000000000000	
Silver	11000000001100000001100000000	
Gray	10000000001000000001000000000	
Olive	10000000001000000000000000000	
Purple	10000000000000000001000000000	
Maroon	10000000000000000000000000000	
Aqua	00000000001111111111111111111	
Lime	00000000001111111110000000000	
Teal	00000000001000000001000000000	
Green	00000000001000000000000000000	
Blue	00000000000000000001111111111	
Navy	00000000000000000001000000000	
Black	00000000000000000000000000000	

- **Existence**

This bit enabled us to toggle each falling button ON/OFF from the software. When the bit is set to 0, all other bits in the packet are irrelevant because the corresponding button is never displayed. What this ultimately accomplished was to eliminate the need for a dynamic allocation of falling buttons, which would have been quite a challenge to accomplish especially in hardware; with this mechanism, however, we simply needed to predetermine the number of falling blocks and declare them as arrays in both hardware and software.

- **HPos**

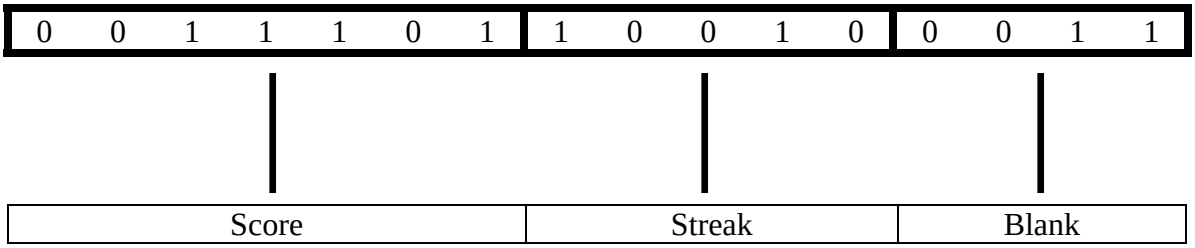
Representing the horizontal coordinate of a falling button by pixels would have required us 10 bits ($2^{10} > 640 > 2^9$), which meant that this information alone would occupy the majority of the 16-bit packet. Wanting to be as efficient as possible with the representation of data and especially wishing to represent the entire falling block in 16 bits, we realized that any falling block would be situated in one of the four predetermined lanes. Since we knew where the lanes were situated, the software only needed to inform the software in which of the 4 lanes the corresponding falling block should be. This meant that only 2 bits were needed, and that hardware would convert this information to a coordinate through the following relationship:

RECTANGLE_HSTART(i) = HPos*RECT_SPACING+RECT_OFFSET
Where RECT_SPACING = 90 and RECT_OFFSET = 160.

- **VPos**

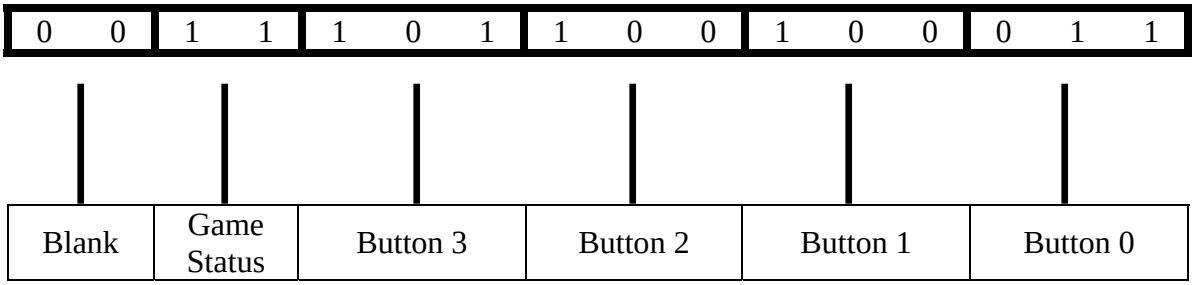
Representing the vertical coordinate was straight forward since there was no room for efficiency improvements; the full pixel coordinate had to be represented, which meant that this information required 9 bits ($2^9 > 480 > 2^8$).

2.b. 16-Bit "Scoring" Packet



This packet is sent from software to hardware whenever a change occurs. The 7 bits of "Score" allows up to 127 points to be kept, where only 100 are used. "Streak" marks the longest streak of "Perfects" that the player has achieved so far.

2.c. 16-Bit "Game State" Packet



* For each button, the MSB (represented here by the leftmost bit) represents whether a text-sprite should be displayed at all. The two remaining bits represent what text should be displayed on screen: 00 for “Miss”, 01 for “Perfect”, 10 for “Good”, and 11 for “Bad”. The Game Status block represents whether the game is currently in progress, whether the game has completed, or if the game is about to start.

2.d. Bitmap Generation and Control

The generation of bitmaps in our hardware began with the program IconEdit. We began by drawing our bitmap image in IconEdit, pixel by pixel, and then converting the 3-dimensional array to a 2-dimensional array in Matlab. We proceeded next by using a text editor to format the data into an array in VHDL syntax. Below shows the instantiation of the constant 15 x 20 text, “Perfect”:

```
constant perfect : perfect_text :=
        (("00000000000000000000000000000000000000000000000000",
        "00000000000000000000000000000000000000000000000000",
        "01111110111111011111101111101111101111101111101110",
        "01111110111111011111101111101111101111101111101110",
        "01100110110000011001101100001100000110000000011000",
        "01100110110000011001101100001100000110000000011000",
        "01111110111100011111101110001111000110000000011000",
        "01111110111100011111101110001111000110000000011000",
        "01100000110000011110001100000110000011000000011000",
        "01100000110000011011001100000110000011000000011000",
        "01100000111111011001101100000111111011111100011000",
        "01100000111111011001101100000111111011111100011000",
        "00000000000000000000000000000000000000000000000000",
        "00000000000000000000000000000000000000000000000000",
        "00000000000000000000000000000000000000000000000000");
```

Similar procedures were used to generate the other images. Although hard coding the bitmaps into our hardware may be considered inefficient, we thought that the implementation method was sufficient for our needs as the texts and buttons were only characterized by one color.

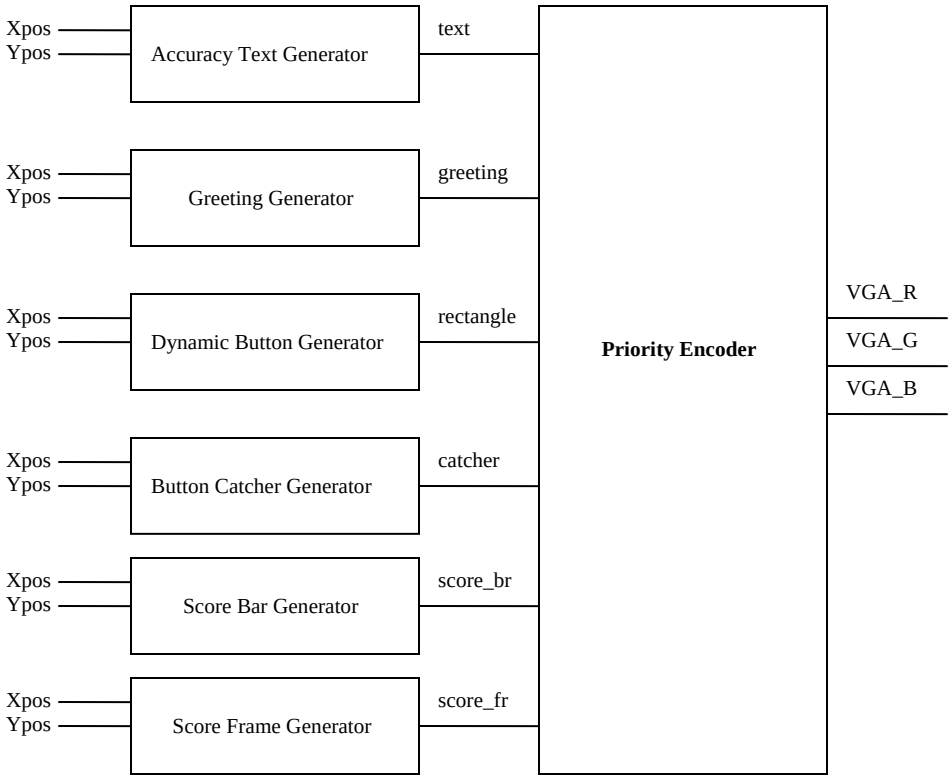
We determined which bitmap to display on the screen using various boundary ranges. While some of the ranges were dynamic such as the vertical positions of the falling buttons, other sprites such as the score bar, frame, text, back buttons, and even horizontal position of falling notes laid in static ranges. With boundary ranges in place, the VGA raster determines the video outputs given a horizontal and vertical position through a multiplexer. The following code below shows how we displayed our hard coded bitmaps:

```
text <= perfect(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH - 345))(49 - (conv_integer(HCount)-HSYNC-
        HBACK_PORCH-430));
```

Here, text is a std_logic that is either 1 or 0 depending on whether the look up table to the array returns a 1 or 0. The array is index by column and then row, with perfect(0)(0) representing the LSB. To display the text right side up, we needed to offset the indices accordingly.

In the last stage of the VGA display, all the outputs of the previous multiplexers feed another priority encoder with the VGA colors as its outputs. Our order of

precedence is as follows: Accuracy texts, “Start/Game Over Message”, falling buttons, button catchers (back buttons), score bar, and scoring frame. The overall architecture is shown below:



Video Controller

3. GAME PLAY

3.a. Software Implementation

The software portion of this project was designed around the manipulation of two types of information: 1) the position of each graphical sprite displayed by the hardware on the screen, and 2) the overall game logic (the score, notes, etc.). In order to accomplish this, however, careful attention had to be paid to not only the software that was being designed, but also the methods with which information between the hardware and the software was passed.

In order to accomplish this at a speed that was consistent with what we wanted to see on the screen after getting translated into hardware, we used one large loop that kept going while the game was in action (there were still notes left to be played or there were notes that were still on screen). In each iteration of the loop, the following tasks were completed:

- Each button's vertical position was incremented (moving the button downwards)
- Each button's vertical position was checked to insure that it was not past the bottom edge of the screen
- Each key press was recorded and appropriate action taken
    - A search for a button in the appropriate error was started
    - If found, the score would be calculated and information sent to hardware
    - If not found, score would be deducted and information sent to hardware
    - Score and number of streaks updated
- The next note is added to the queue of notes waiting to be played
- Finally, an inner loop is started as a means of slowing the game down to playable speed

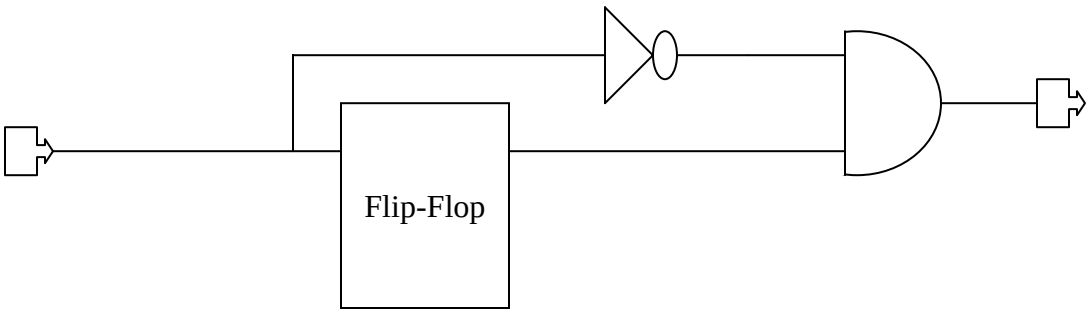
Although the game's loop appears pretty simple in its description, its actual implementation was in actuality pretty tricky. Each piece of data had to be stored in its own appropriate variable and then shifted bitwise to its appropriate location. In total, 10 packets were sent in each loop iteration (8 for the buttons, 2 more for extra information).

Not only was the sending of the data tricky, but so was the receiving of the information. Each key press should only be counted once, a difficult concept considering each key press lasted many iterations of the loop. Detecting this in hardware meant that each key press resulted in a signal lasting only one clock cycle – not time enough, it seemed, to pass the information back to software. Instead, we used the basic idea of a flip flop in software – comparing the values of each key press to its previous value. A change implied a second button pressed, no change implied same button being held down.

Finally, there were nuances in our language that needed to be dealt with as the program grew. For example, global variables had to be declared so that our functions (created as a means of simplifying our code) could access all the data in the main function without having them be passed, is just one example. Another example was our

choice to create a “buffer” for the lowest and highest scores. That is, while 100 was the maximum score displayed one screen, the actual score ranged from -20 to 120, a 20 point buffer on each side of the score that allowed a really good player a few chances to mess up without losing their perfect score (or, conversely, a bad player needing to do really well multiple times to gain a possible score). Both of these were design decisions on our part that at some times made the project more difficult, but we believe added overall to the project.

There were several design decisions during the course of the project where we initially planned to have hardware control the logic, but eventually decided that software would be the better choice. The first was how to detect a button press. We originally wanted to treat the key press the same way some of us did for Lab #1, which was to hold the ‘press’ state for only one clock cycle. This was done by feeding the key() signal, which would be 0 if the button is pressed down, into a flip-flop. The output of the flip-flop was AND’d with the NOT of the input. This would ensure that the new signal keypress(), which was the output of this circuit, would only bounce to 1 for one clock cycle. This solved two problems: one was if the user held the button down, and the other was the hundreds of clock cycles the 1 would be held for even if the user only tapped the button. The flip-flop diagram can be seen below:



We tested this code for lighting up successive LED’s, and it worked great. However, we soon realized that the vast majority of our control logic was being done in software due to its robustness, efficiency, and speed. We decided that it would make more sense for the software to determine whether or not a button has been pressed. We took out the flip-flop and simply sent 4 bits of information back to the software from the hardware, 1 bit for each of the four buttons. They simply signaled whether or not the button was pushed, and we used software to establish if it was being held down or it was a new push. We essentially moved the flip-flop to the software side.

This choice of software control over hardware control was also done for determining the accuracy of button presses. We originally coded detection into VHDL by ANDing the key press, button vertical position, and button catcher vertical position. Although this method worked, it required a bulky set of if-elsif statements and had one serious flaw: it couldn’t compensate for the case where the user pushes the button more

than once very fast. Thus, we once again decided to implement the accuracy texts using software control.

This was done by sending a 12-bit string of information from the software to the hardware. Those 12 bits are divided into four 3-bit groups, each one representing one of the four buttons. The first bit of each group controlled existence, while the last two controlled whether the accuracy text was “perfect”, “good”, “bad”, or “miss”. This way the software can not only control which accuracy text to display on which button, but it can also control the “blinking” of the text using the existence bit. The hardware simply displays the text-sprite in the appropriate position at the appropriate time.

3.b. Playfile Maker

- **Playfile Data Structure:**

The representation of the playfile within the software took a format such as the following:

```
NotesLane = {0,1,2,3,2,1,2,2};
NotesTime = {10,20,30,40,80,140,150,270};
```

Each element in the NotesLane array would be matched up with an element in NotesTime, and together they represent one "note" or one falling button within the game. Each note is played in chronological order from left to right in the array. NotesLane represents which of the four lanes (0 to 3; i.e. Lane 1 to 4) the note belongs to, while NotesTime represents the time at which the note appears and starts falling. "Time" here has no exact hard-coded units but its unit is effectively determined by the delay loop in the software.

Then, the data above would play the following "button song":

Button at Lane 1 at t = 10
Button at Lane 2 at t = 20
Button at Lane 3 at t = 30
Button at Lane 4 at t = 40
Button at Lane 2 at t = 80
Button at Lane 1 at t = 140
Button at Lane 2 at t = 150
Button at Lane 2 at t = 270

- **Motivation of Playfile Maker**

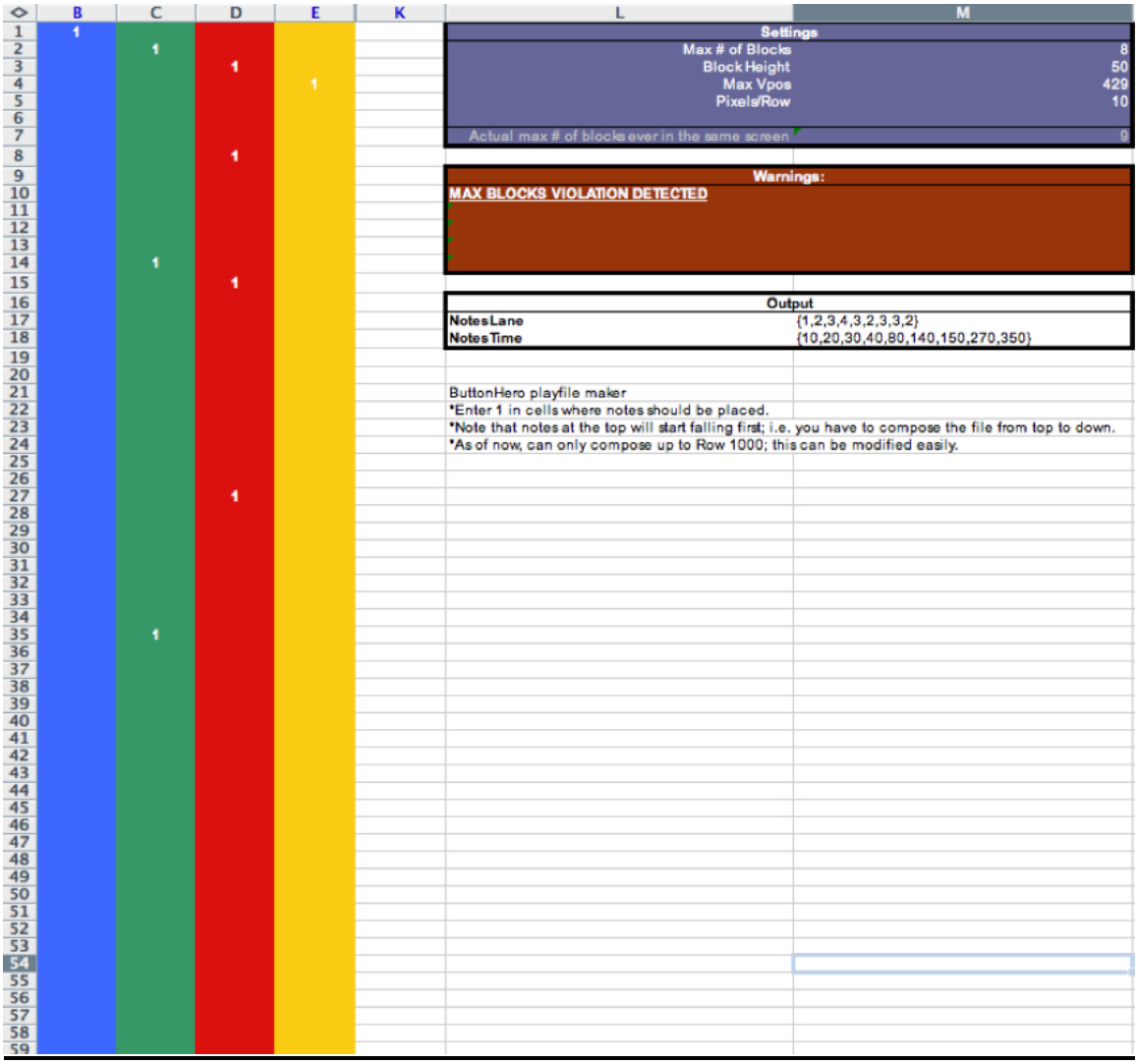
In practice, however, a playable song can get very long, which makes the manual composing very difficult and virtually impossible to visualize. Realizing that this would be a barrier in appreciating the true entertainment potential of the game and failing to make Button Hero a truly enjoyable and customizable game, we needed to find a solution. In particular, we needed a GUI to compose Button Hero music so that developers could visualize what is being created. While implementing such a system on the same platform as the game itself would have been ideal, the cost vs. benefit proved to be far too much, so we opted for a quick but efficient alternative: creating a Playfile Maker in MS Excel.

Besides the difficulty of visualization, another motivation behind the construction of Playfile Maker was the fact that the music had to follow two constraints:

1. The total number of falling buttons displayed on the same screen cannot exceed MAXBLOCKS, where MAXBLOCKS in the current version = 8. (MAXBLOCKS can be increased by adding repetitive lines of code in VHDL)

2. Falling buttons should not overlap.

While an error checking algorithm for the purpose of checking for constraint violation was embedded in the software, these limitation made the compilation of a legit playfile quite difficult, let alone an entertaining one. A GUI Playfile Maker would effective solve all these problems.



The above is the preliminary version of our Playfile Maker. It is programmed so that it takes a number of settings and a semi-graphic playfile (i.e. four leftmost columns in the spreadsheet) as the input, and outputs the playfile compatible with Button Hero (i.e. NotesLane and NotesTime) An advanced feature of PlayMaker is constraint violation detection; when either of the two constraints mentioned above is violated, it will display a warning so that the developer can make changes appropriately. Furthermore, it was made flexible enough so that it could still be useful when parameters within Button Hero was changed, e.g. the dimension of a falling button, the number of allowed buttons, screen resolution, etc.

- **Settings for Playfile Maker:**

Max # of Blocks: The maximum number of buttons as specified by the hardware.
Block Height: The pixel resolution of the falling block bitmap.
Max VPos: Vertical screen resolution.
Pixels/Row: Number of pixels that each row in the Spreadsheet should represent.

The Playfile Maker indeed proved to be useful in composing extended versions of playfiles for the demo. Furthermore, it would constitute a "total package" for a product along with Button Hero.

3.c. Scoring:

Game starts with 50 points. Maximum and minimum scores are 120 and -20, respectively, although the highest and lowest reported scores are 100 and 0, respectively. This is done to provide a buffer, rewarding a good player with leeway to mess up if he/she is doing well, and punish a bad player to perform a lot better to get out of his/her slump. Each button pressed results in a change in the score. Their values are as follows:

Button	Proximity	Score Change
Miss	Button not hit at all	-8
Perfect	-10 < Distance < +10	+4
Good	-15 < Distance < +15	+2
Bad	Otherwise	-10

4. GROUP MEMBER RESPONSIBILITES AND REFLECTIONS

Kenneth Yu:
Responsibilities:
I was in charge of the hardware on the FPGA board including the buttons, 7-segment LCD's, and LED's. During the initial design, this included the buttons triggering blocks moving. I also initially triggered the accuracy texts using hardware detection, but this was ditched in favor of software control. I also worked on the VHDL hardware code, including the accuracy texts (sprites, displaying, reading data) as well as the greeting and game over pages. As far as software, I made three songs for our program and helped organize the accuracy text/start page/end page data string.

Reflections:
The sheer size and complexity of this project really highlighted the essentials of being a successful team. I had read previous years' reports, and almost all of them emphasized the need for concrete and realistic milestones, constant communication, and to start as early as possible. I can only emphasize those points once again.

Attempting to complete this project as you would a purely software-based design will only end in additional frustration. Even though we decided to control as much of the game as possible through software, the intricacies involved with communicating between hardware and software still proved to be extremely difficult to deal with. In addition, VHDL requires exceedingly long compile times, which means that the guess-and-check used for Java/C simply isn't feasible.

The solution is to plan, plan, and plan. You must draw out and diagram exactly what you are trying to accomplish in VHDL *before* you actually write a single line of code. The extra 10 minutes you spend will save you hours in the long run. This also means that you must start early, and to schedule as many meetings with the TA/Professor as possible. It is very helpful to set realistic milestones for every member of your team on a weekly basis. This not only ensures that the project is moving along, but also serves as motivation.

Through this project, we also learned the true importance of communication. Coding in both VHDL and C requires the team members to constantly update each other on their progress and/or changes. I feel that our team significantly improved upon this aspect of teamwork, and it definitely helped our project proceed at a faster pace.

Working in a team is a unique experience because you are not in complete control of every aspect of your project. The most you can do it encourage, help, and hold accountable the other members of your team. I feel that our team was very successful in doing this, and we had very few disagreements.

I feel that I have become much more familiar with the engineering design process and the complexities that must be addressed early on. I have strengthened my knowledge of VHDL and C, as well as the advantages and disadvantages of both hardware and software.

Joo Han Chang

Responsibilities:
1. Designed and implemented the graphics driver in both HW/SW for falling button animation, e.g. coloring scheme and modified raster scanning. Efficiency was prioritized; i.e. opted for practicality rather than scalability, given the time constraints. This driver could then be applied to other objects in the game, e.g. score bar, messages, etc.
2. Designed the initial software architecture, customized to represent a time series of falling buttons in an array that would be interfaced with hardware.
3. Creating a Playfile Maker in MS Excel that mimics a true GUI in playfile design for the use of developers.

Reflections:
VHDL-wise, the key message to future students is "think hardware." I would highly recommend that students really learn their digital logic before taking this class. Software-style coding can create unpredictable problems which are extremely difficult to debug. For example, insensible sequential programming can possibly destroy the timing architecture and even cause tri-state signals, with unpredictable results. Meanwhile, there was most than just the technical side in the real education value of this class.

A specific but important skill that I picked up while completing this project was how to display good judgment in choosing between expandability and practicality. In the past, I had often opted for expandability; it seemed more elegant and made future development much easier, and therefore always appeared to be the superior choice. In writing a simple C program in the past, for example, I would often divide everything into different functions and be as general as possible in the implementation, so that the program could be modified and expanded in the future.

I almost fell into the same "trap" for this project. From common sense and from looking at successful past projects, I came under the impression that we should develop a fully fledged sprite system and spent much time in researching how such a system would work. Due to enormous time pressure for the 75% demo, however, I reached a compromise with myself and the rest of the group, and opted for "getting the job done," meaning that we found the most time-efficient way of developing a system that would do just enough to adhere to the project specifications. Because this meant that the architecture was not easily expandable – e.g. display of multi-color bitmaps, true text with font, etc – I wanted to upgrade the system for the final version.

The reality of project management, however, imposed on us a strict deadline, whose violation would have been very costly indeed when the stakes are as high as our timely graduation. This is analogous to real world engineering, like what Prof. Edwards mentioned in our very first class with regards to how products have to either meet market timing or face a dramatic loss in potential revenue. Furthermore, the nature of the project – low-level and difficult debugging – made the pace of progress extremely unpredictable. As a result, I permanently shifted my focus from creating an elegant system to be proud of to making sure that a working project would be completed in time. In retrospect, this

was the right choice. After all, engineering isn't about elegancy but about practicality; it isn't for the engineers but for the users.

George Liao:

Responsibilities: I was in charge of the software post-75% demo. This included the gameplay logic, sending bytes of information to the hardware to control the score, current perfect streak, accuracy texts, and the display of the greeting and game over pages.

Reflections: Working on this project was a tremendous learning experience, in respect to both technical knowledge as well as working in groups. The technical experience is obvious – experience designing a fairly large project, experience in VHDL, Quartus, and Nios, and more. I also learned how best to work in groups, communicate effectively with others, balance my priorities and those of my teammates.

Designing a project as large as Button Hero required a great deal of planning ahead. One of the issues that we failed to take into account for was paying more attention to detail. Implementation details need to be agreed upon beforehand if teammates are to be working on separate parts simultaneously so that there isn't any incongruity. VHDL itself posed many interesting problems – the largest being that we had to abandon many traditional programming ideas and realize that everything we did was in hardware. As for Quartus, Nios, and the applications that we used, it would have been a better idea to work earlier to figure out the ins and outs of those applications.

Working in groups, we learned that it was important to set milestones and deadlines and keep to them, despite the fact that it often becomes extremely tempting to push them back. A weekly (at least) meeting schedule needs to be agreed upon, not to mention times where the group can just sit down together and work on the project. Communication is key.

Charles Lam

Responsibilities: I was in charge of fixing bugs and adding features from software to the hardware post-75% demo. This included the addition of the score bar and accuracy text to the video display driver. I was also assigned to attempt implementing sounds through the on-board audio codec. However, this endeavor proved challenging and unfeasible due to time constraints.

Reflections: The project taught me a lot about the amount of time and patience one needs to succeed in the design world. Our initial unfamiliarity with syntax in VHDL was our most difficult challenge. In C, we could quickly through trial and error, compile and run code to discover logically errors. In VDHL, this process of trial and error, proved deadly as redrawing a line on the display required an upwards of ten to fifteen minutes of compilation time. This time lengthen as the project became more complex. Nonetheless,

I gained good experience programming in hardware and have developed much more confidence in attacking similar problems in the future.

If I could do this project over again, my team members and I would probably all agree on streamlining the process and communication from the beginning. Many times, hardware could not be developed because we were unsure how it was going to be interfaced between the software. All this factors can be attributed to our inexperience in the design process. For example in our 75% demo, we focused mainly on getting the video controller to have some functionality. The next 25% was to include the software to run the controller and to implement "simple" keyboard sounds when the buttons were pressed. We figured that once we figured how to interface the software and hardware, we could in theory just attach the audio codec and feed data to it from the software. However, the interfacing of the SW/HW suffered some delays and consequently, we could not begin testing of the audio component.

Ultimately, I am proud of what we accomplished and am pleased to be able to materialize a project that stayed true to our initial idea. The project taught the importance of being both a detail-oriented guru and a high level thinker. However, I feel beyond the technical knowledge gained from this project and class, the experience of struggling through the design process will prove to be invaluable going forward.

5. PROJECT FILES

5.a. Main VHDL File

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity vga_raster is

  port (

    reset : in std_logic;
    clk   : in std_logic;                    -- Should be 25.125 MHz

    avs_s1_clk        : in  std_logic;
    avs_s1_reset_n    : in  std_logic;
    avs_s1_read       : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(4 downto 0);
    avs_s1_readdata   : out std_logic_vector(15 downto 0);
    avs_s1_writedata  : in  std_logic_vector(15 downto 0);
      key               : in  std_logic_vector(3 downto 0);
    ledgreen                    : out std_logic_vector(8 downto 0);
-- Green LEDs
    ledred                : out std_logic_vector(17 downto 0);
    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment
displays
       : out std_logic_vector(6 downto 0);


    VGA_CLK,                                 -- Clock
    VGA_HS,                                  -- H_SYNC
    VGA_VS,                                  -- V_SYNC
    VGA_BLANK,                               -- BLANK
    VGA_SYNC : out std_logic;                -- SYNC
    VGA_R,                                   -- Red[9:0]
    VGA_G,                                   -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0) -- Blue[9:0]
  );

end vga_raster;

architecture rtl of vga_raster is

  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
```

```
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  type color is array(15 downto 0) of std_logic_vector(29 downto 0);
  type buttonmap is array(49 downto 0) of std_logic_vector(49 downto 0);
  type scoremap is array(19 downto 0) of std_logic_vector(59 downto 0);
  type perfect_text is array(14 downto 0) of std_logic_vector(49 downto
0);
  type good_text is array(14 downto 0) of std_logic_vector(49 downto 0);
  type bad_text is array(14 downto 0) of std_logic_vector(49 downto 0);
  type miss_text is array(14 downto 0) of std_logic_vector(49 downto 0);
  type start_screen is array(14 downto 0) of std_logic_vector(199
downto 0);
  type end_screen is array(14 downto 0) of std_logic_vector(49 downto
0);

  constant colors : color :=
      ("00000000000000000000000000000000","00000000000000000000001000000000
","00000000000000000000001111111111","00000000001000000000000000000000",
      "00000000001000000000001000000000","00000000001111111111000000000000"
,"00000000001111111111111111111111","10000000000000000000000000000000",
      "10000000000000000000001000000000","10000000001000000000000000000000"
,"10000000001000000000001000000000","11000000001100000000001100000000",
      "11111111110000000000000000000000","11111111110000000000001111111111"
,"11111111111111111111100000000000","11111111111111111111111111111111");

  constant start : start_screen :=
      ("00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000",
      "00011111010000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000010000000000000000000100000000000000000010000000000000000000000
00100000000100010000000000000000000000000000000000000000000000000000000000000000",
      "00010010100000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000010000000000000000000100000000000000000010000000000000000000000
00100000000100010000000000000000000000000000000000000000000000000000000000000000",
      "00010010100000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000010010000000000000000100000000000000000010000000000000000000000
00100000000100010000000000000000000000000000000000000000000000000000000000000000",
      "00011111010000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000010110000000000000000100000000000000000010000000000000000000000
00100000000100010000000000000000000000000000000000000000000000000000000000000000",
      "00010000101111011110011110111100011110111101111011110111100001111
11001111010010000111001111010010000111110111100001111011111011110011110
11111000011111011110111100001111011110011111011110000000000000000000",
      "00010000101001010010010000100100001001010000100101000010000000100
01001001010010000110001001010010000001000100100001000000100010010010000
00100000000100010010100100001001010010010101010010000000000000000000",
```

```
      "000100001011110100100111101111000010010100001111011110111100001
0100100101001000011100111110100100000100010010000111100010001001000
0010000000010001001011110000100101001001010101111000000000000000",
      "000100001010000100100000101000000010010100001000000010000010000
0100100101001000010110100001001000000100010010000000100010001001000
0010000000010001001010000000100101001001010101000000000000000000000",
      "000100001011110111111011110111100001111010000111101111011111000011
1110100101111100001001011110111110000001000111100001111000100011111010000
0010000000010001001011110000111101111101010101110001010100000000",
      "000000000000000000000000000000000010000000000000000000000000000000
0000000000001000000000000000001000000000000000000000000000000000000000
0000000000000000000000000000001000000000000000000000000000000000000",
      "000000000000000000000000000000000010000000000000000000000000000000
0000000000001000000000000000001000000000000000000000000000000000000000
0000000000000000000000000000001000000000000000000000000000000000000",
      "000000000000000000000000000000000010000000000000000000000000000000
0000000001111000000000000011110000000000000000000000000000000000000000
0000000000000000000000000001111000000000000000000000000000000000000");

  constant ends : end_screen :=
      ("00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000",
      "00011110111101111011111000011110100000101111011111",
      "00010000100101010101000000001001010000010100001001",
      "00010000100101010101000000001001011000110100001001",
      "00010000111101010101000000001001001000100100001001",
      "00010000100101010101110000001001001101100111001111",
      "00010110100101010101000000001001000101000100001100",
      "00010010100101000101000000001001000111000100001010",
      "00010010100101000101000000001001000010000100001001",
      "00011110100101000101111100001111000010000111101001",
      "00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000");

  constant perfect : perfect_text :=
      ("00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000",
      "01111110111111101111110111111101111110111111101111110",
      "01111110111111101111110111111101111110111111101111110",
      "01100110110000011001101100000110000011000000011000",
      "01100110110000011001101100000110000011000000011000",
      "01111110111110001111110111100011111000110000000011000",
      "01111110111110001111110111100011111000110000000011000",
      "01100000110000011110001100000110000011000000011000",
      "01100000110000011011001100000110000011000000011000",
      "01100000111111101100110110000011111110111111100011000",
      "01100000111111101100110110000011111110111111100011000",
      "00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000");

  constant good : good_text :=
      ("00000000000000000000000000000000000000000000000000",
      "00000000000000000000000000000000000000000000000000",
      "00000000000111111011111101111110111110000000000000",
```

```
    "00000000000111111011111101111110111111000000000000",
    "00000000000110000011001101100110110011000000000000",
    "00000000000110000011001101100110110011000000000000",
    "00000000000110111011001101100110110011000000000000",
    "00000000000110111011001101100110110011000000000000",
    "00000000000110011011001101100110110011000000000000",
    "00000000000110011011001101100110110011000000000000",
    "00000000000111111011111101111110111111000000000000",
    "00000000000111111011111101111110111111000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000");

constant bad : bad_text :=
    ("00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000011111100111100111111000000000000000",
    "00000000000000011111101111110111111000000000000000",
    "00000000000000011001101100110110011000000000000000",
    "00000000000000011001101100110110011000000000000000",
    "00000000000000011111001111110110011000000000000000",
    "00000000000000011111001111110110011000000000000000",
    "00000000000000011001101100110110011000000000000000",
    "00000000000000011001101100110110011000000000000000",
    "00000000000000011111101100110111111000000000000000",
    "00000000000000011111101100110111111000000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000");

constant miss : miss_text :=
    ("00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000011000110111111011111101111110000000000000",
    "00000000011101110111111011111101111110000000000000",
    "00000000011111110001100011000001100000000000000000",
    "00000000011111110001100011000001100000000000000000",
    "00000000011111110001100011111101111110000000000000",
    "00000000011010110001100011111101111110000000000000",
    "00000000011000110001100000001100000110000000000000",
    "00000000011000110001100000001100000110000000000000",
    "00000000011000110111111011111101111110000000000000",
    "00000000011000110111111011111101111110000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000",
    "00000000000000000000000000000000000000000000000000");

constant score_frame : scoremap :=
    ("0000000000000000000000000000000000000000000000000000000000000000",
    "0000000000000000000000000000000000000000000000000000000000000000",
    "0000000000000000000000000000000000000000000000000000000000000000",
    "0000000000000011111111111111111111111111111111111000000000000000",
    "0000000111111111111111111111111111111111111111111111100000000000",
    "0000111111111111111111111111111111111111111111111111111110000000",
    "0011111111111111111111111111111111111111111111111111111111100000",
    "0011111111111111111111111111111111111111111111111111111111100000",
    "0111111111111111111111111111111111111111111111111111111111110000",
```

```
    "111111111111111111111111111111111111111111111111111111111111",
    "111111111111111111111111111111111111111111111111111111111111",
    "111111111111111111111111111111111111111111111111111111111111",
    "111111111111111111111111111111111111111111111111111111111111",
    "111111111111111111111111111111111111111111111111111111111111",
    "111111111111111111111111111111111111111111111111111111111111",
    "111111111111100000000000000000000000000000000001111111111111",
    "011111111111100000000000000000000000000000000001111111111110",
    "001111111111100000000000000000000000000000000001111111111100",
    "001111111111100000000000000000000000000000000001111111111100",
    "000011100111100000000000000000000000000000000001111001110000");


constant button : buttonmap :=
    ("00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111000000000011100000000000000000",
    "00000000000000000100000000000000100000000000000000",
    "00000000000000000000011111111000000000000000000000",
    "00000000000000000001111111111110000000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000011111111111111111111000000000000000",
    "00000000000000111111111111111111111100000000000000",
    "00000000000001111111111111111111111110000000000000",
    "00000000000011111111111111111111111111000000000000",
    "00000000000111111111111111111111111111100000000000",
    "00000000000111111111111111111111111111100000000000",
    "11111111001111111111111111111111111111001111111111",
    "11111110001111111111111111111111111111000111111111",
    "11111110011111111111111111111111111111110011111111",
    "11111100011111111111111111111111111111110001111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100111111111111111111111111111111111100111111",
    "11111100011111111111111111111111111111110001111111",
    "11111110011111111111111111111111111111110011111111",
    "11111110001111111111111111111111111111000111111111",
    "11111111001111111111111111111111111111001111111111",
    "00000000000111111111111111111111111111100000000000",
    "00000000000111111111111111111111111111100000000000",
    "00000000000011111111111111111111111111000000000000",
    "00000000000001111111111111111111111110000000000000",
    "00000000000000111111111111111111111100000000000000",
    "00000000000000011111111111111111111000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000001111111111110000000000000000000",
    "00000000000000000000011111111000000000000000000000",
    "00000000000000000100000000000000100000000000000000",
```

```
    "00000000000000000111000000000111000000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000",
    "00000000000000000111111111111111100000000000000000");


  -- added rectangle controls
  constant RECT_HEIGHT : integer := 50;
  constant RECT_WIDTH  : integer := 50;
  constant RECT_SPACING : integer := 90; -- RECT_WIDTH + SPACE
  constant RECT_OFFSET : integer := 160;

  type rect_vtype is array(7 downto 0) of std_logic_vector(8 downto 0);
  type rect_htype is array(7 downto 0) of std_logic_vector(1 downto 0);
  type rect_single is array(7 downto 0) of std_logic;
  type rect_color is array(7 downto 0) of std_logic_vector(3 downto 0);
  type rect_integer is array(7 downto 0) of integer;


  signal RECT_E : rect_single;
  signal RECT_C : rect_color;
  signal packet : std_logic_vector(15 downto 0);
  signal packet_text : std_logic_vector(15 downto 0);
  signal text_data : std_logic_vector(11 downto 0);
  -- Signals for the rectangle


  signal RECTANGLE_VSTART : rect_integer;
  signal RECTANGLE_VEND   : rect_integer;
  signal RECTANGLE_HSTART : rect_integer;
  signal RECTANGLE_HEND   : rect_integer;


  -- Signals for the video controller

  signal Hcount : std_logic_vector(9 downto 0);  -- Horizontal position
(0-800)
  signal Vcount : std_logic_vector(9 downto 0);  -- Vertical position
(0-524)
  signal EndOfLine, EndOfField : std_logic;

  signal vga_hblank, vga_hsync,
       vga_vblank, vga_vsync : std_logic;  -- Sync. signals

  signal rectangle : std_logic;  -- rectangle area
  signal currrect : std_logic_vector(3 downto 0);
  signal catcher : std_logic;
  signal catchercolor : std_logic_vector(3 downto 0);
  signal bcounter : std_logic_vector(4 downto 0);
  signal ledgreens : std_logic_vector(8 downto 0);


  -- Signals for Score Frame, Score Bar
  signal score_br_height : integer;
```

```
  signal actual_score : integer;
  signal perfect_scores : std_logic_vector(4 downto 0);
  signal good_bad : std_logic_vector(1 downto 0);
  signal position_button : std_logic_vector(1 downto 0);
  signal score_br : std_logic;

  -- Signals for How_Good Texts
  signal how_good_1 : std_logic_vector(1 downto 0);
  signal how_good_2 : std_logic_vector(1 downto 0);
  signal how_good_3 : std_logic_vector(1 downto 0);
  signal how_good_4 : std_logic_vector(1 downto 0);
  signal text1E : std_logic;
  signal text2E : std_logic;
  signal text3E : std_logic;
  signal text4E : std_logic;
  signal startend : std_logic_vector(1 downto 0);

  signal score_fr : std_logic;  -- score frame top
  signal score_fr_color : std_logic_vector(3 downto 0);
  signal score_bar_color: std_logic_vector(3 downto 0);
  signal score_HSTART : integer := 40;
  signal score_HEND : integer := 100; --HSTART+60
  signal score_VSTART : integer := 40;
  signal score_VEND : integer := 60; --VSTART+20
  signal score_height : integer := 400;

  -- Signals for Text
  signal text : std_logic;
  signal greeting : std_logic;


  -- Signals for the buttons

--  signal q         : std_logic_vector(3 downto 0);
--  signal key_press : std_logic_vector(3 downto 0);
----  signal key_buff  : std_logic_vector(3 downto 0);
----  signal bcounter  : std_logic_vector(11 downto 0);


begin
  how_good_4 <= packet_text(1 downto 0);
  text4E <= packet_text(2);
  how_good_3 <= packet_text(4 downto 3);
  text3E <= packet_text(5);
  how_good_2 <= packet_text(7 downto 6);
  text2E <= packet_text(8);
  how_good_1 <= packet_text(10 downto 9);
  text1E <= packet_text(11);
  startend <= packet_text(13 downto 12);

  -- rectangle variables
  --RECTANGLE_HSTART <= conv_integer(RECT_H);
  --RECTANGLE_VSTART <= conv_integer(RECT_V);
  --RECTANGLE_HEND   <= RECTANGLE_HSTART + RECT_WIDTH;
  --RECTANGLE_VEND   <= RECTANGLE_VSTART + RECT_HEIGHT;
  actual_score <= conv_integer(packet(15 downto 9));
  score_br_height <= actual_score*4;
```

```
  good_bad <= packet(3 downto 2);
  position_button <= packet(1 downto 0);

  -- signal bus
  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
      if avs_s1_reset_n = '0' then
        --avs_s1_writedata <= (others => '0');
      else
        if avs_s1_chipselect = '1' then
              if (avs_s1_write = '1') then
                if (conv_integer(avs_s1_address) < 8) then
                    RECT_E(conv_integer(avs_s1_address)) <=
avs_s1_writedata(11);
                    RECT_C(conv_integer(avs_s1_address)) <=
avs_s1_writedata(15 downto 12);
                    RECTANGLE_HSTART(conv_integer(avs_s1_address)) <=
conv_integer(avs_s1_writedata(10 downto 9))*RECT_SPACING+RECT_OFFSET;
                    RECTANGLE_VSTART(conv_integer(avs_s1_address)) <=
conv_integer(avs_s1_writedata(8 downto 0));
                    RECTANGLE_HEND(conv_integer(avs_s1_address)) <=
conv_integer(avs_s1_writedata(10 downto
9))*RECT_SPACING+RECT_OFFSET+RECT_WIDTH;
                    RECTANGLE_VEND(conv_integer(avs_s1_address)) <=
conv_integer(avs_s1_writedata(8 downto 0))+RECT_HEIGHT;
                elsif (conv_integer(avs_s1_address) = 8) then
                  packet <= avs_s1_writedata;
                elsif (conv_integer(avs_s1_address) = 9) then
                  packet_text <= avs_s1_writedata;
                end if;
              end if;
            end if;
      end if;
    end if;
  end process;
  -- Horizontal and vertical counters

  HCounter : process (clk, reset)
  begin
    if reset = '1' then
      Hcount <= (others => '0');
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
    end if;
  end process HCounter;

  EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

  VCounter: process (clk, reset)
  begin
    if reset = '1' then
```

```
    Vcount <= (others => '0');
  elsif clk'event and clk = '1' then
    if EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk, reset)
begin
  if reset = '1' then
    vga_hsync <= '1';
  elsif clk'event and clk = '1' then
    if EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk, reset)
begin
  if reset = '1' then
    vga_hblank <= '1';
  elsif clk'event and clk = '1' then
    if Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk, reset)
begin
  if reset = '1' then
    vga_vsync <= '1';
  elsif clk'event and clk = '1' then
    if EndOfLine ='1' then
      if EndOfField = '1' then
        vga_vsync <= '1';
      elsif Vcount = VSYNC - 1 then
        vga_vsync <= '0';
      end if;
    end if;
  end if;
end process VSyncGen;
```

```
  VBlankGen : process (clk, reset)
  begin
    if reset = '1' then
      vga_vblank <= '1';
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;

  -- VGA GENERATOR

  VideoGen : process (clk, reset)
  begin
    if reset = '1' then
      rectangle <= '1';
    elsif clk'event and clk = '1' then

        -- Greeting end/start
        if ((startend="00") and (Hcount >= HSYNC + HBACK_PORCH + 220)
and (Hcount < HSYNC + HBACK_PORCH + 420) and (Vcount >= VSYNC +
VBACK_PORCH + 200) and (Vcount < VSYNC + VBACK_PORCH + 215)) then
          greeting <= start(14 - (conv_integer(VCount)-VSYNC-
VBACK_PORCH - 200))(199 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-220));
       elsif ((startend="11") and (Hcount >= HSYNC + HBACK_PORCH + 300)
and (Hcount < HSYNC + HBACK_PORCH + 350) and (Vcount >= VSYNC +
VBACK_PORCH + 200) and (Vcount < VSYNC + VBACK_PORCH + 215)) then
          greeting <= ends(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH
- 200))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-300));
        else
            greeting <= '0';
        end if;


        -- Buttons
      if ((RECT_E(0)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(0)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(0)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(0)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(0))) then
        rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
RECTANGLE_VSTART(0))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(0));
            currrect <= "0000";
      elsif ((RECT_E(1)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(1)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(1)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(1)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(1))) then
        rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
RECTANGLE_VSTART(1))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(1));
```

```
            currrect <= "0001";
      elsif ((RECT_E(2)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(2)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(2)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(2)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(2))) then
        rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
RECTANGLE_VSTART(2))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(2));
            currrect <= "0010";
      elsif ((RECT_E(3)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(3)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(3)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(3)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(3))) then
            rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH
- RECTANGLE_VSTART(3))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(3));
            currrect <= "0011";
      elsif ((RECT_E(4)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(4)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(4)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(4)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(4))) then
            rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH
- RECTANGLE_VSTART(4))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(4));
            currrect <= "0100";
      elsif ((RECT_E(5)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(5)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(5)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(5)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(5))) then
            rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH
- RECTANGLE_VSTART(5))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(5));
            currrect <= "0101";
      elsif ((RECT_E(6)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(6)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(6)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(6)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(6))) then
        rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
RECTANGLE_VSTART(6))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(6));
            currrect <= "0110";
      elsif ((RECT_E(7)='1') and (Hcount >= HSYNC + HBACK_PORCH +
RECTANGLE_HSTART(7)) and (Hcount < HSYNC + HBACK_PORCH +
RECTANGLE_HEND(7)) and (Vcount >= VSYNC + VBACK_PORCH +
RECTANGLE_VSTART(7)) and (Vcount < VSYNC + VBACK_PORCH +
RECTANGLE_VEND(7))) then
        rectangle <= button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
RECTANGLE_VSTART(7))(conv_integer(HCount)-HSYNC-HBACK_PORCH-
RECTANGLE_HSTART(7));
            currrect <= "0111";
        else
            rectangle <= '0';
```

```
        end if;


        -- Back Buttons
      if ((Hcount >= HSYNC + HBACK_PORCH + 160) and (Hcount < HSYNC +
HBACK_PORCH + 210) and (Vcount >= VSYNC + VBACK_PORCH + 366) and
(Vcount < VSYNC + VBACK_PORCH + 416) and (button(conv_integer(VCount)-
VSYNC-VBACK_PORCH - 366)(conv_integer(HCount)-HSYNC-HBACK_PORCH-
160)='0')) then
            catcher <= '1';
            catchercolor <= X"E"; -- Blue
        elsif ((Hcount >= HSYNC + HBACK_PORCH + 250) and (Hcount <
HSYNC + HBACK_PORCH + 300) and (Vcount >= VSYNC + VBACK_PORCH + 366)
and (Vcount < VSYNC + VBACK_PORCH + 416) and
(button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
366)(conv_integer(HCount)-HSYNC-HBACK_PORCH-250)='0')) then
            catcher <= '1';
            catchercolor <= X"C"; -- Green
        elsif ((Hcount >= HSYNC + HBACK_PORCH + 340) and (Hcount <
HSYNC + HBACK_PORCH + 390) and (Vcount >= VSYNC + VBACK_PORCH + 366)
and (Vcount < VSYNC + VBACK_PORCH + 416) and
(button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
366)(conv_integer(HCount)-HSYNC-HBACK_PORCH-340)='0')) then
            catcher <= '1';
            catchercolor <= X"8"; -- Maroon
        elsif ((Hcount >= HSYNC + HBACK_PORCH + 430) and (Hcount <
HSYNC + HBACK_PORCH + 480) and (Vcount >= VSYNC + VBACK_PORCH + 366)
and (Vcount < VSYNC + VBACK_PORCH + 416) and
(button(conv_integer(VCount)-VSYNC-VBACK_PORCH -
366)(conv_integer(HCount)-HSYNC-HBACK_PORCH-430)='0')) then
            catcher <= '1';
            catchercolor <= X"6"; -- Olive
        else
            catcher <= '0';
      end if;

        -- Score Frame
        if ((Hcount >= HSYNC + HBACK_PORCH + SCORE_HSTART) and (Hcount
< HSYNC + HBACK_PORCH + SCORE_HEND) and (Vcount >= VSYNC + VBACK_PORCH
+ SCORE_VSTART) and (Vcount < VSYNC + VBACK_PORCH + SCORE_VEND)) then
            score_fr <= score_frame(19 - (conv_integer(VCount)- VSYNC -
VBACK_PORCH - SCORE_VSTART) )(59 - (conv_integer(HCount)-HSYNC-
HBACK_PORCH - SCORE_HSTART));
        score_fr_color <= X"E";
      -- Draw reverse
        elsif ((Hcount >= HSYNC + HBACK_PORCH + SCORE_HSTART) and
(Hcount < HSYNC + HBACK_PORCH + SCORE_HEND) and (Vcount >= VSYNC +
VBACK_PORCH + SCORE_VSTART + score_height) and (Vcount < VSYNC +
VBACK_PORCH + SCORE_VEND + score_height)) then
            score_fr <= score_frame(conv_integer(VCount)- VSYNC-
VBACK_PORCH - SCORE_VSTART - score_height)(conv_integer(HCount)-HSYNC-
HBACK_PORCH - SCORE_HSTART);
          score_fr_color <= X"E";
        elsif ((Hcount >= HSYNC + HBACK_PORCH + SCORE_HSTART + 9) and
(Hcount < HSYNC + HBACK_PORCH + SCORE_HSTART + 13) and (Vcount >= VSYNC
+ VBACK_PORCH + SCORE_VEND) and (Vcount < VSYNC + VBACK_PORCH +
SCORE_VEND + score_height)) then
```

```
            score_fr <= '1';
          score_fr_color <= X"E";
        elsif ((Hcount >= HSYNC + HBACK_PORCH + SCORE_HEND - 13) and
(Hcount < HSYNC + HBACK_PORCH + SCORE_HEND - 9) and (Vcount >= VSYNC +
VBACK_PORCH + SCORE_VEND) and (Vcount < VSYNC + VBACK_PORCH +
SCORE_VEND + score_height)) then
            score_fr <= '1';
          score_fr_color <= X"E";
        else
            score_fr <= '0';
        end if;

        -- Score Bar
        if ((Hcount >= HSYNC + HBACK_PORCH + SCORE_HSTART + 13) and
(Hcount < HSYNC + HBACK_PORCH + SCORE_HEND - 13) and (Vcount >= VSYNC +
VBACK_PORCH + (400 - score_br_height) + SCORE_VEND - 5) and (Vcount <
VSYNC + VBACK_PORCH + SCORE_VSTART + score_height + 5)) then
              score_br <= '1';
        else
            score_br <= '0';
        end if;

        -- Greeting end/start

        -- Scoring Text
        if ((text1E='1') and (how_good_1="01") and (Hcount >= HSYNC +
HBACK_PORCH + 160) and (Hcount < HSYNC + HBACK_PORCH + 210) and (Vcount
>= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH + 360))
then
          text <= perfect(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH
- 345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-160));
        elsif ((text2E='1') and (how_good_2="01") and (Hcount >= HSYNC
+ HBACK_PORCH + 250) and (Hcount < HSYNC + HBACK_PORCH + 300) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= perfect(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH
- 345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-250));
        elsif ((text3E='1') and (how_good_3="01") and (Hcount >= HSYNC
+ HBACK_PORCH + 340) and (Hcount < HSYNC + HBACK_PORCH + 390) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= perfect(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH
- 345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-340));
        elsif ((text4E='1') and (how_good_4="01") and (Hcount >= HSYNC
+ HBACK_PORCH + 430) and (Hcount < HSYNC + HBACK_PORCH + 480) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= perfect(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH
- 345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-430));

        elsif ((text1E='1') and (how_good_1="10") and (Hcount >= HSYNC
+ HBACK_PORCH + 160) and (Hcount < HSYNC + HBACK_PORCH + 210) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= good(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-160));
```

```
        elsif ((text2E='1') and (how_good_2="10") and (Hcount >= HSYNC
+ HBACK_PORCH + 250) and (Hcount < HSYNC + HBACK_PORCH + 300) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= good(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-250));
        elsif ((text3E='1') and (how_good_3="10") and (Hcount >= HSYNC
+ HBACK_PORCH + 340) and (Hcount < HSYNC + HBACK_PORCH + 390) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= good(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-340));
        elsif ((text4E='1') and (how_good_4="10") and (Hcount >= HSYNC
+ HBACK_PORCH + 430) and (Hcount < HSYNC + HBACK_PORCH + 480) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= good(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-430));

        elsif ((text1E='1') and (how_good_1="11") and (Hcount >= HSYNC
+ HBACK_PORCH + 160) and (Hcount < HSYNC + HBACK_PORCH + 210) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= bad(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-160));
        elsif ((text2E='1') and (how_good_2="11") and (Hcount >= HSYNC
+ HBACK_PORCH + 250) and (Hcount < HSYNC + HBACK_PORCH + 300) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= bad(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-250));
        elsif ((text3E='1') and (how_good_3="11") and (Hcount >= HSYNC
+ HBACK_PORCH + 340) and (Hcount < HSYNC + HBACK_PORCH + 390) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= bad(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-340));
        elsif ((text4E='1') and (how_good_4="11") and (Hcount >= HSYNC
+ HBACK_PORCH + 430) and (Hcount < HSYNC + HBACK_PORCH + 480) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= bad(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-430));

        elsif ((text1E='1') and (how_good_1="00") and (Hcount >= HSYNC
+ HBACK_PORCH + 160) and (Hcount < HSYNC + HBACK_PORCH + 210) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= miss(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-160));
        elsif ((text2E='1') and (how_good_2="00") and (Hcount >= HSYNC
+ HBACK_PORCH + 250) and (Hcount < HSYNC + HBACK_PORCH + 300) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= miss(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-250));
```

```
        elsif ((text3E='1') and (how_good_3="00") and (Hcount >= HSYNC
+ HBACK_PORCH + 340) and (Hcount < HSYNC + HBACK_PORCH + 390) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= miss(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-340));
        elsif ((text4E='1') and (how_good_4="00") and (Hcount >= HSYNC
+ HBACK_PORCH + 430) and (Hcount < HSYNC + HBACK_PORCH + 480) and
(Vcount >= VSYNC + VBACK_PORCH + 345) and (Vcount < VSYNC + VBACK_PORCH
+ 360)) then
          text <= miss(14 - (conv_integer(VCount)-VSYNC-VBACK_PORCH -
345))(49 - (conv_integer(HCount)-HSYNC-HBACK_PORCH-430));
        else
            text <= '0';
        end if;

    end if;

  end process VideoGen;



  -- Registered video signals going to the video DAC

  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then
          -- accuracy texts
          if text = '1' then
              VGA_R <= "1111111111";
              VGA_G <= "1111111111";
              VGA_B <= "1111111111";
          elsif greeting = '1' then
                VGA_R <= "1111111111";
              VGA_G <= "1111111111";
              VGA_B <= "1111111111";

            elsif rectangle = '1' then
              VGA_R <=
colors(conv_integer(RECT_C(conv_integer(currrect))))(29 downto 20);
              VGA_G <=
colors(conv_integer(RECT_C(conv_integer(currrect))))(19 downto 10);
              VGA_B <=
colors(conv_integer(RECT_C(conv_integer(currrect))))(9 downto 0);
            -- backbuttons
            elsif catcher = '1' then
             VGA_R <= colors(conv_integer(catchercolor))(29 downto 20);
             VGA_G <= colors(conv_integer(catchercolor))(19 downto 10);
              VGA_B <= colors(conv_integer(catchercolor))(9 downto 0);
            -- score bar
          elsif score_br = '1' then
                VGA_R <= "1111111111";
              VGA_G <= "1111111111";
```

```
              VGA_B <= "1111111111";
          -- score frame
            elsif score_fr = '1' then
                 VGA_R <= colors(conv_integer(score_fr_color))(29
downto 20);
              VGA_G <= colors(conv_integer(score_fr_color))(19 downto
10);
             VGA_B <= colors(conv_integer(score_fr_color))(9 downto 0);

          -- note lines
            elsif ((HCount > HSYNC + HBACK_PORCH + RECT_OFFSET - 20)
and (HCount < HSYNC + HBACK_PORCH + 640 - RECT_OFFSET + 20) and
((conv_integer(VCount) mod 50) < 2)) then
              VGA_R <= "1111111111";
              VGA_G <= "1111111111";
              VGA_B <= "1111111111";
            else
          VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

  dataout: process (clk, reset)
  begin
    if reset = '1' then
      avs_s1_readdata <= "0000000000000000";
    elsif clk'event and clk = '1' then
--      if key_press(0) = '1' then
--        avs_s1_readdata(0) <= '1';
--
--      elsif key_press(1) = '1' then
--        avs_s1_readdata(1) <= '1';
--
--      elsif key_press(2) = '1' then
--        avs_s1_readdata(2) <= '1';
--
--      elsif key_press(3) = '1' then
--        avs_s1_readdata(3) <= '1';
--
--      end if;

        --bcounter <= bcounter + 1;
--      avs_s1_readdata(0) <= (not key(0));
--      avs_s1_readdata(1) <= (not key(1));
--      avs_s1_readdata(2) <= (not key(2));
--      avs_s1_readdata(3) <= (not key(3));
      avs_s1_readdata(3 downto 0) <= (not key);
        --avs_s1_readdata(3 downto 0) <= key(3 downto 0);
        avs_s1_readdata(15 downto 4) <= "000000000000";
```

```
    end if;
  end process dataout;

-----------------------------------
--Process that displays score on --
--     7 segment display         --
-----------------------------------

  seven_seg_score : process (clk, reset)
  begin
    if reset = '1' then
        HEX0 <= "1000000"; HEX1 <= "1000000"; HEX2 <= "1000000";
    elsif clk'event and clk = '1' then
            HEX3 <= "1111111"; HEX4 <= "0100011"; HEX5 <= "0101111";
            HEX6 <= "0000110"; HEX7 <= "0001001";
      if actual_score = 0 then
            HEX0 <= "1000000"; HEX1 <= "1000000"; HEX2 <= "1000000";
      elsif actual_score = 1 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "1111001";
      elsif actual_score = 2 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0100100";
      elsif actual_score = 3 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0110000";
      elsif actual_score = 4 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0011001";
      elsif actual_score = 5 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0010010";
      elsif actual_score = 6 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0000010";
      elsif actual_score = 7 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "1111000";
      elsif actual_score = 8 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0000000";
      elsif actual_score = 9 then
            HEX2 <= "1000000"; HEX1 <= "1000000"; HEX0 <= "0010000";
      elsif actual_score = 10 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "1000000";
      elsif actual_score = 11 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "1111001";
      elsif actual_score = 12 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0100100";
      elsif actual_score = 13 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0110000";
      elsif actual_score = 14 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0011001";
      elsif actual_score = 15 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0010010";
      elsif actual_score = 16 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0000010";
      elsif actual_score = 17 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "1111000";
      elsif actual_score = 18 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0000000";
      elsif actual_score = 19 then
            HEX2 <= "1000000"; HEX1 <= "1111001"; HEX0 <= "0010000";
      elsif actual_score = 20 then
            HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "1000000";
```

```
elsif actual_score = 21 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "1111001";
elsif actual_score = 22 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0100100";
elsif actual_score = 23 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0110000";
elsif actual_score = 24 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0011001";
elsif actual_score = 25 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0010010";
elsif actual_score = 26 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0000010";
elsif actual_score = 27 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "1111000";
elsif actual_score = 28 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0000000";

elsif actual_score = 29 then
      HEX2 <= "1000000"; HEX1 <= "0100100"; HEX0 <= "0010000";
elsif actual_score = 30 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "1000000";
elsif actual_score = 31 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "1111001";
elsif actual_score = 32 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0100100";
elsif actual_score = 33 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0110000";
elsif actual_score = 34 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0011001";
elsif actual_score = 35 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0010010";
elsif actual_score = 36 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0000010";
elsif actual_score = 37 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "1111000";
elsif actual_score = 38 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0000000";
elsif actual_score = 39 then
      HEX2 <= "1000000"; HEX1 <= "0110000"; HEX0 <= "0010000";
elsif actual_score = 40 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "1000000";

elsif actual_score = 41 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "1111001";
elsif actual_score = 42 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0100100";
elsif actual_score = 43 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0110000";
elsif actual_score = 44 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0011001";
elsif actual_score = 45 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0010010";
elsif actual_score = 46 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0000010";
elsif actual_score = 47 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "1111000";
elsif actual_score = 48 then
```

```
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0000000";
elsif actual_score = 49 then
      HEX2 <= "1000000"; HEX1 <= "0011001"; HEX0 <= "0010000";
elsif actual_score = 50 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "1000000";

elsif actual_score = 51 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "1111001";
elsif actual_score = 52 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0100100";
elsif actual_score = 53 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0110000";
elsif actual_score = 54 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0011001";
elsif actual_score = 55 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0010010";
elsif actual_score = 56 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0000010";
elsif actual_score = 57 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "1111000";
elsif actual_score = 58 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0000000";
elsif actual_score = 59 then
      HEX2 <= "1000000"; HEX1 <= "0010010"; HEX0 <= "0010000";
elsif actual_score = 60 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "1000000";
elsif actual_score = 61 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "1111001";
elsif actual_score = 62 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0100100";
elsif actual_score = 63 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0110000";
elsif actual_score = 64 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0011001";
elsif actual_score = 65 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0010010";
elsif actual_score = 66 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0000010";
elsif actual_score = 67 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "1111000";
elsif actual_score = 68 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0000000";
elsif actual_score = 69 then
      HEX2 <= "1000000"; HEX1 <= "0000010"; HEX0 <= "0010000";
elsif actual_score = 70 then
      HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "1000000";
elsif actual_score = 71 then
      HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "1111001";
elsif actual_score = 72 then
      HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0100100";
elsif actual_score = 73 then
      HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0110000";
elsif actual_score = 74 then
      HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0011001";
elsif actual_score = 75 then
      HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0010010";
elsif actual_score = 76 then
```

```
            HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0000010";
      elsif actual_score = 77 then
            HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "1111000";
      elsif actual_score = 78 then
            HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0000000";
      elsif actual_score = 79 then
            HEX2 <= "1000000"; HEX1 <= "1111000"; HEX0 <= "0010000";

      elsif actual_score = 80 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "1000000";
      elsif actual_score = 81 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "1111001";
      elsif actual_score = 82 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0100100";
      elsif actual_score = 83 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0110000";
      elsif actual_score = 84 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0011001";
      elsif actual_score = 85 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0010010";
      elsif actual_score = 86 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0000010";
      elsif actual_score = 87 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "1111000";
      elsif actual_score = 88 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0000000";
      elsif actual_score = 89 then
            HEX2 <= "1000000"; HEX1 <= "0000000"; HEX0 <= "0010000";
      elsif actual_score = 90 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "1000000";
      elsif actual_score = 91 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "1111001";
      elsif actual_score = 92 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0100100";
      elsif actual_score = 93 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0110000";
      elsif actual_score = 94 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0011001";
      elsif actual_score = 95 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0010010";
      elsif actual_score = 96 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0000010";
      elsif actual_score = 97 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "1111000";
      elsif actual_score = 98 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0000000";
      elsif actual_score = 99 then
            HEX2 <= "1000000"; HEX1 <= "0010000"; HEX0 <= "0010000";
      elsif actual_score = 100 then
            HEX2 <= "1111001"; HEX1 <= "1000000"; HEX0 <= "1000000";
      end if;
      end if;
end process seven_seg_score;
--            "1111001" WHEN   -- 1
--                "0100100" WHEN   -- 2
--            "0110000" WHEN  -- 3
--                "0011001" WHEN   -- 4
```

```
--                "0010010" WHEN  -- 5
--                "0000010" WHEN  -- 6
--                "1111000" WHEN  -- 7
--                "0000000" WHEN  -- 8
--                "0010000" WHEN  -- 9
--                  "1000000" WHEN -- 0

--------------------------------
--Process that lights up LED's--
--------------------------------
  buttonpress: process (clk, reset)
  begin
    if reset = '1' then
      bcounter <= "00000";
    elsif clk'event and clk = '1' then
            perfect_scores <= packet(8 downto 4);

            if perfect_scores = "00000" then
            ledred <= "0000000000000000";
            ledgreens <= "000000000";
        elsif perfect_scores = "00001" then
            ledred <= "1000000000000000";
        elsif perfect_scores = "00010" then
            ledred <= "1100000000000000";
        elsif perfect_scores = "00011" then
            ledred <= "1110000000000000";
        elsif perfect_scores = "00100" then
            ledred <= "1111000000000000";
        elsif perfect_scores = "00101" then
            ledred <= "1111100000000000";
        elsif perfect_scores = "00110" then
            ledred <= "1111110000000000";
        elsif perfect_scores = "00111" then
            ledred <= "1111111000000000";
        elsif perfect_scores = "01000" then
            ledred <= "1111111100000000";
        elsif perfect_scores = "01001" then
            ledred <= "1111111110000000";
        elsif perfect_scores = "01010" then
            ledred <= "1111111111000000";
        elsif perfect_scores = "01011" then
            ledred <= "1111111111100000";
        elsif perfect_scores = "01100" then
            ledred <= "1111111111110000";
        elsif perfect_scores = "01101" then
            ledred <= "1111111111111000";
        elsif perfect_scores = "01110" then
            ledred <= "1111111111111100";
        elsif perfect_scores = "01111" then
            ledred <= "1111111111111110";
        elsif perfect_scores = "10000" then
            ledred <= "1111111111111111";
            ledgreens <= "000000000";
```

```
        elsif perfect_scores = "10011" then
            ledgreens <= "010000000";
        elsif perfect_scores = "10100" then
            ledgreens <= "011000000";
        elsif perfect_scores = "10101" then
            ledgreens <= "011100000";
        elsif perfect_scores = "10110" then
            ledgreens <= "011110000";
        elsif perfect_scores = "10111" then
            ledgreens <= "011111000";
        elsif perfect_scores = "11000" then
            ledgreens <= "011111100";
        elsif perfect_scores = "11001" then
            ledgreens <= "011111110";
        elsif perfect_scores = "11010" then
            ledgreens <= "011111111";
        elsif perfect_scores = "11011" then
            ledgreens <= "011111111";
        elsif perfect_scores = "11100" then
            ledgreens <= "011111111";
        elsif perfect_scores = "11101" then
            ledgreens <= "011111111";
        elsif perfect_scores = "11110" then
            ledgreens <= "011111111";
        else
            perfect_scores <= "00000";
        end if;
    end if;
  end process buttonpress;
  ledgreen <= ledgreens;
end rtl;
```

5.b. Main Software(C) File

```
#include <io.h>
#include <system.h>
#include <stdio.h>

#define BASE 0x00080800

//Color Map - 16
#define WHITE 0x0000
#define YELLOW 0x1000
#define FUCHSIA 0x2000
#define RED 0x3000
#define SILVER 0x4000
#define GRAY 0x5000
#define OLIVE 0x6000
#define PURPLE 0x7000
#define MAROON 0x8000
#define AQUA 0x9000
#define LIME 0xA000
#define TEAL 0xB000
#define GREEN 0xC000
#define BLUE 0xD000
#define NAVY 0XE000
#define BLACK 0XF000

// Score points
#define MISS -8
#define PERFECT 4
#define GOOD 2
#define BAD -10
#define FLASHTIME 60

// function prototypes
unsigned int keypress();
void kill(int killed);
void pressed(int button);
void ChangeScore(int change);

// key information
unsigned int data = 0;

unsigned int key0_old = 0;
unsigned int key0_new = 0;
unsigned int key0 = 0;

unsigned int key1_old = 0;
unsigned int key1_new = 0;
unsigned int key1 = 0;

unsigned int key2_old = 0;
unsigned int key2_new = 0;
unsigned int key2 = 0;

unsigned int key3_old = 0;
unsigned int key3_new = 0;
```

```
unsigned int key3 = 0;

// note information

int NumButtons = 0;               // number of buttons on screen

/*********************
 *    Novice Mode     *
 *********************/
/*
int NumNotes = 28;                  // number of notes
unsigned int NotesLane[] = { 0, 1, 2, 3,
                             0, 1, 2, 3,
                             0, 1, 2, 3,
                             3, 2, 1, 0,
                             0, 1, 2, 3,
                             3, 2, 1, 0,
                             0, 1, 2, 3
                           } ;

unsigned int NotesTime[] = { 100, 200, 300, 400,
                             700, 700, 700, 700,
                             1000, 1000, 1050, 1050,
                             1300, 1350, 1400, 1450,
                             1700, 1800, 1800, 1850,
                             2100, 2200, 2300, 2300,
                             2600, 2600, 2600, 2600
                           } ;
*/


/*********************
 *  Intermediate Mode *
 *********************/

int NumNotes = 64;
unsigned int NotesLane[] = { 0, 1, 2, 3,
                             0, 1, 2, 3,
                             0, 1, 2, 3,
                             3, 2, 1, 0,
                             0, 1, 2, 3,
                             3, 2, 1, 0,
                             0, 1, 2, 3,
                             2, 1, 2, 3,
                             3, 0, 0, 1,
                             1, 3, 2, 0,
                             1, 2, 0, 3,
                             3, 0, 2, 1,
                             1, 0, 0, 3,
                             2, 3, 0, 2,
                             0, 2, 1, 2,
                             3, 1, 0, 3
                           } ;

unsigned int NotesTime[] = { 100, 100, 200, 200,
                             400, 450, 450, 450,
                             550, 550, 650, 700,
```

```
                              850, 950, 1000, 1000,
                              1150, 1150, 1200, 1250,
                              1400, 1400, 1400, 1500,
                              1700, 1750, 1750, 1800,
                              2000, 2150, 2150, 2150,
                              2300, 2350, 2400, 2400,
                              2700, 2750, 2850, 2900,
                              3150, 3200, 3250, 3250,
                              3300, 3450, 3500, 3550,
                              3700, 3750, 3800, 3800,
                              3900, 3950, 3950, 4000,
                              4200, 4200, 4250, 4250,
                              4400, 4450, 4500, 4500
                            } ;


/*********************
 *    Expert Mode     *
 *********************/
/*
int NumNotes = 132;                 // number of notes
unsigned int NotesLane[] = { 1, 2, 3, 0,
                             3, 1, 2, 0,
                             2, 3, 1, 0,
                             1, 2, 3, 0,
                             3, 0, 2, 3,
                             1, 2, 3, 0,
                             1, 2, 3, 0,
                             1, 2, 0, 3,
                             1, 3, 2, 0,
                             1, 3, 2, 0,
                             2, 0, 0, 3,
                             2, 1, 3, 0,
                             1, 3, 2, 0,
                             1, 2, 2, 0,
                             1, 2, 3, 0,
                             0, 3, 2, 3,
                             0, 2, 1, 2,
                             2, 1, 0, 3,
                             2, 3, 0, 1,
                             3, 0, 1, 2,
                             2, 0, 3, 1,
                             0, 2, 3, 0,
                             0, 3, 2, 1,
                             1, 2, 3, 0,
                             3, 0, 2, 3,
                             2, 1, 3, 2,
                             0, 3, 0, 1,
                             0, 2, 1, 3,
                             1, 0, 3, 2,
                             2, 3, 1, 0,
                             1, 3, 2, 1,
                             0, 1, 2, 3,
                             1, 1, 0, 2
                             } ;

unsigned int NotesTime[] = { 100,  100,  150,  200,
```

```
                                300,  350,  450,  500,
                                700,  750,  800,  850,
                                900,  900,  950,  950,
                                1050, 1050, 1100, 1150,
                                1300, 1300, 1400, 1400,
                                1600, 1600, 1650, 1700,
                                1800, 1800, 1850, 1900,
                                2000, 2100, 2150, 2150,
                                2300, 2300, 2400, 2400,
                                2500, 2550, 2600, 2600,
                                2750, 2800, 2800, 2950,
                                3100, 3100, 3150, 3150,
                                3300, 3300, 3350, 3400,
                                3600, 3650, 3700, 3750,
                                3900, 3900, 3950, 4000,
                                4200, 4200, 4250, 4250,
                                4350, 4400, 4400, 4450,
                                4550, 4550, 4600, 4650,
                                4750, 4800, 4850, 4900,
                                5050, 5150, 5150, 5200,
                                5300, 5350, 5350, 5400,
                                5450, 5500, 5500, 5550,
                                5650, 5700, 5700, 5750,
                                5850, 5850, 5900, 5950,
                                6050, 6050, 6100, 6150,
                                6250, 6250, 6350, 6400,
                                6450, 6500, 6550, 6550,
                                6600, 6750, 6750, 6800,
                                6900, 7000, 7050, 7050,
                                7150, 7150, 7250, 7300,
                                7450, 7500, 7500, 7550,
                                7600, 7700, 7700, 7700
                                } ;
*/


int i, j;                       // temp variables
unsigned int packet[10];        // packets used to send information

// button information
unsigned int ColorCode[] = {BLUE, LIME, RED, YELLOW};
unsigned int Color[]     = {0, 0, 0, 0, 0, 0, 0, 0};
unsigned int Existence[] = {0, 0, 0, 0, 0, 0, 0, 0};
unsigned int HPos[]      = {0, 0, 0, 0, 0, 0, 0, 0};
unsigned int VPos[]      = {0, 0, 0, 0, 0, 0, 0, 0};

// game information
unsigned int VMax = 429;
unsigned int Time = 0;            // current time
int CurrentNote;                  // current note

// score
int score = 50;
int real_score = 50;
int perfects = 0;
int max_perfects = 0;
int how_good = 0;
```

```
int position = 0;

int time0 = 0;
int time1 = 0;
int time2 = 0;
int time3 = 0;

int b0 = 0;
int b1 = 0;
int b2 = 0;
int b3 = 0;

int gameStatus = 0;
int started = 0;

int main()
{
  // game start
  while (1)
  {
    printf ("Welcome to Button Hero! Press any key to begin game!\n\n");
    gameStatus = 0;

    IOWR_16DIRECT(BASE, 18, 0);

    // update score
    ChangeScore(0);

    // pause until key pressed
    do
    {
      keypress();
    } while (!key0 && !key1 && !key2 && !key3);


    // update score
    ChangeScore(0);
    gameStatus = 2;

    printf ("Good luck!\n\n");

    // while there are still notes left to play, or there are buttons
to be pressed
    while (CurrentNote < NumNotes || NumButtons > 0)
    {
      for (j=0; j<8; j++)
      {
        // button exists
        if (Existence[j] == 0x0800)
        {
          VPos[j] = VPos[j]+1;           // move button

          // kill button
          if (VPos[j] == VMax)
          {
            //printf ("Missed!\t");
```

```
            // for score keeping
            perfects = 0;
            how_good = 0;
            position = HPos[j] >> 9;
            ChangeScore(MISS);
            kill(j);
            j--;
          }
        }
      }

      // send information to hardware
      for (j=0; j<8; j++)
      {
        packet[j] = Color[j] | Existence[j] | HPos[j] | VPos[j];
        IOWR_16DIRECT(BASE, 2*j, packet[j]);
      }

      if (time0 <= 0)
        b0 = b0 & 3;
      else
        b0 = b0 | 4;
      if (time1 <= 0)
        b1 = b1 & 3;
      else
        b1 = b1 | 4;
      if (time2 <= 0)
        b2 = b2 & 3;
      else
        b2 = b2 | 4;
      if (time3 <= 0)
        b3 = b3 & 3;
      else
        b3 = b3 | 4;

      packet[9] = (gameStatus << 12) | (b3 << 9) | (b2 << 6) | (b1 << 3)
| b0;
      if (Time > NotesTime[0]) //buffer time
        IOWR_16DIRECT(BASE, 18, packet[9]);



      // a key was pressed
      keypress();

      if (key0)
        pressed(3<<9);
      if (key1)
        pressed(2<<9);
      if (key2)
        pressed(1<<9);
      if (key3)
        pressed(0<<9);



      // look for more notes to add to queue
```

```
      while (Time >= NotesTime[CurrentNote] && NumButtons < 8 &&
CurrentNote < NumNotes)
      {
        // get next note
        Existence[NumButtons]=0x0800;
        HPos[NumButtons]=(NotesLane[CurrentNote])<<9;
        Color[NumButtons]=ColorCode[NotesLane[CurrentNote]];
        VPos[NumButtons]=0;

        NumButtons++;
        CurrentNote++;
      }

        for (i = 0 ; i < (2<<11) ; i++) ;
      Time++;
      time0--;
      time1--;
      time2--;
      time3--;

      // too many buttons on screen at once
      if(NumButtons>8)
      {
        printf ("Error! Button overload.");

        break;
      }
    }

    printf("\n\nEnd of game. Press any key to restart.\n\n");

    CurrentNote = 0;
    NumButtons = 0;
    Time = 0;
    real_score = 50;
    perfects = 0;
    max_perfects = 0;

    time0 = 0;
    time1 = 0;
    time2 = 0;
    time3 = 0;

    b0 = 0;
    b1 = 0;
    b2 = 0;
    b3 = 0;

    for (i = 0; i < 8; i++)
    {
      Color[i] = 0;
      Existence[i] = 0;
      HPos[i] = 0;
      VPos[i] = 0;
    }

    // send information to hardware (to clear remaining tiles)
```

```
    for (j=0; j<8; j++)
    {
      packet[j] = Color[j] | Existence[j] | HPos[j] | VPos[j];
      IOWR_16DIRECT(BASE, 2*j, packet[j]);
    }

    gameStatus = 3;
    packet[9] = gameStatus << 12;
    IOWR_16DIRECT(BASE, 18, packet[9]);

    // pause until key pressed
    do
    {
      keypress();
    } while (!key0 && !key1 && !key2 && !key3);

  }

  printf("Goodbye!\n");

  return 0;

}


unsigned int keypress()
{
  data = IORD_16DIRECT(BASE, 20);

  // if respective key is pressed
  key0_new = data & 1;
  key1_new = data & 2;
  key2_new = data & 4;
  key3_new = data & 8;

  // if it is on action of being pressed down
  if (key0_new == 0 && key0_old != 0)
    key0 = 1;
  else
    key0 = 0;

  if (key1_new == 0 && key1_old != 0)
    key1 = 1;
  else
    key1 = 0;

  if (key2_new == 0 && key2_old != 0)
    key2 = 1;
  else
    key2 = 0;

  if (key3_new == 0 && key3_old != 0)
    key3 = 1;
  else
    key3 = 0;

  // update information
```

```
  key0_old = key0_new;
  key1_old = key1_new;
  key2_old = key2_new;
  key3_old = key3_new;

  return data;
}

void kill(int killed)
{
  NumButtons--;

  int i;
  for(i = killed; i < NumButtons; i++)
  {
    Existence[i] = Existence[i+1];
    Color[i] = Color[i+1];
    HPos[i] = HPos[i+1];
    VPos[i] = VPos[i+1];
  }

  Existence[NumButtons] = 0;
  Color[NumButtons] = 0;
  HPos[NumButtons] = 0;
  VPos[NumButtons] = 0;
}


void pressed(int button)
{
  int button_exist = 0;

  for (j = 0; j < 8; j++)
  {
    if (Existence[j] == 0x0800 && HPos[j] == (button))    // button
exists in lane
    {
      if ((VPos[j] - 370) < 10 || (VPos[j] - 370) > -10)          //
perfect!
      {
        //printf("PERFECT!!!\t");
        how_good = 1;
        position = button >> 9;
        ChangeScore(PERFECT);
        perfects++;
        kill(j);
        button_exist = 1;
      }

      else if ((VPos[j] - 370) < 15 || (VPos[j] - 370) > -15)     //
good!
      {
        //printf("Good!!\t");
        perfects = 0;
        how_good = 2;
        position = button >> 9;
        ChangeScore(GOOD);
```

```
        kill(j);
        button_exist = 1;
      }
      else if ((VPos[j] - 370) < 50 || (VPos[j] - 370) > -50)     //
bad!
      {
        //printf("Good!!\t");
        perfects = 0;
        how_good = 3;
        position = button >> 9;
        ChangeScore(BAD);
        kill(j);
        button_exist = 1;
      }
    }
  }

  // Not perfect or good
  if (button_exist == 0)
  {
    how_good = 3;
    perfects = 0;
    position = button >> 9;
    //printf("Bad!\t");
    ChangeScore(BAD);
  }
}

void ChangeScore(int change)
{
  real_score += change;

  if (real_score > 120)
    real_score = 100;

  else if (real_score < -20)
    real_score = -10;

  if (real_score > 100)
    score = 100;
  else if (real_score < 0)
    score = 0;
  else
    score = real_score;

  if (position == 3)
  {
    b0 = how_good;
    time0 = FLASHTIME;
  }
  if (position == 2)
  {
    b1 = how_good;
    time1 = FLASHTIME;
  }
  if (position == 1)
  {
```

```
    b2 = how_good;
    time2 = FLASHTIME;
  }
  if (position == 0)
  {
    b3 = how_good;
    time3 = FLASHTIME;
  }

  perfects = perfects & 31;
  if (perfects > max_perfects)
    max_perfects = perfects;
  packet[8] = (score << 9) | (max_perfects << 4) | 0;

  IOWR_16DIRECT(BASE, 16, packet[8]);

  //printf ("Score = %d, Perfects = %d, Packet16 = %X\n", score,
perfects, packet[9]);


}
```