CENG3430 Rapid Prototyping of Digital Systems

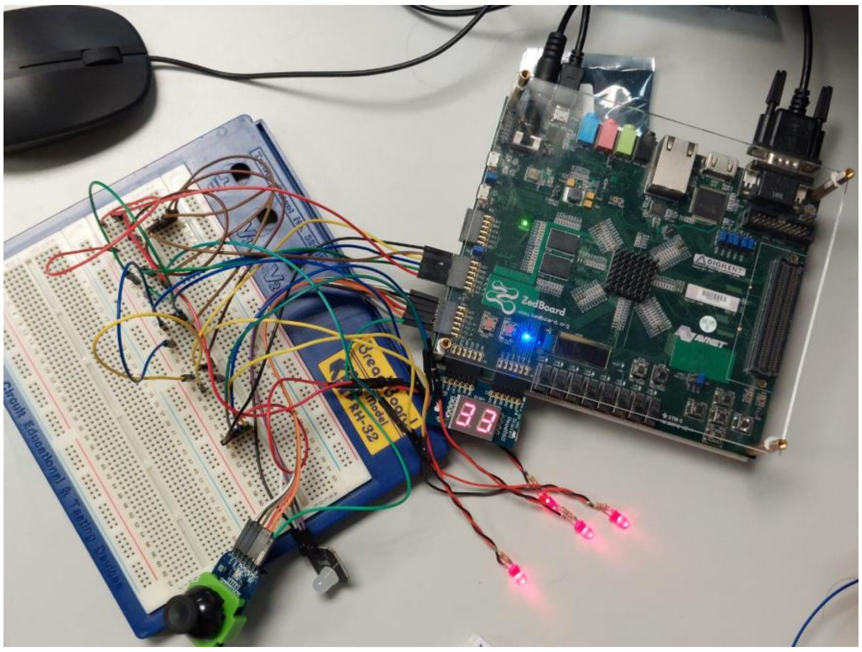
Project Report

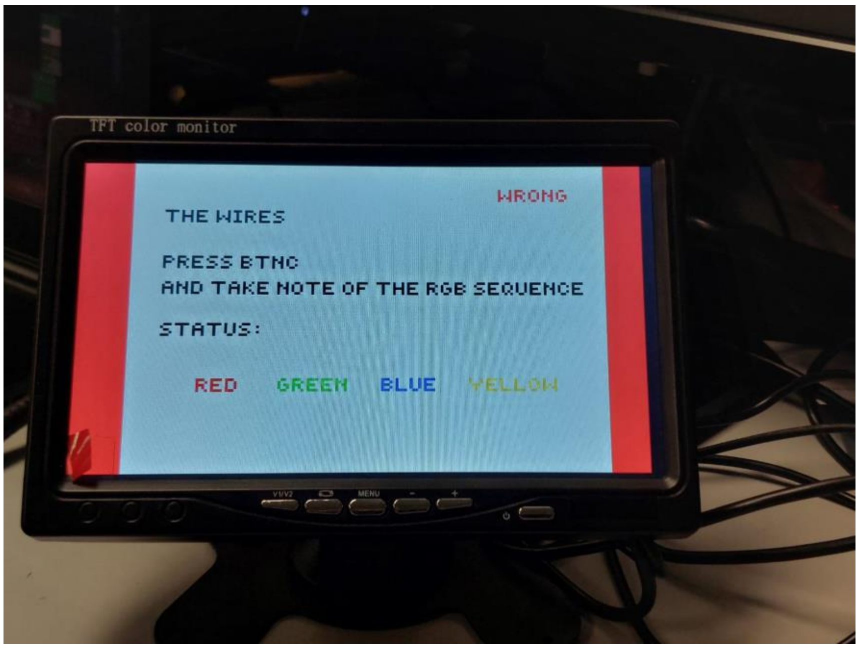
Shum Tsz Ching, 1155177034

Luk Hoi Chun, 1155176886

Demo video here

Keep Talking and Nobody Explodes





Introduction

The goal is to replicate a popular two player coop “bomb defusal” puzzle game, which requires both players’ efforts to complete puzzles on the bomb.
The general idea is that player 1 (“defuser”) holds the “bomb” puzzle but does not know how to defuse it. Player 2 (“expert”) holds the defusal manual but cannot see the bomb. Therefore, the defuser has to verbally describe the bomb to the expert so that the expert can instruct the defuser to defuse the bomb.

Design

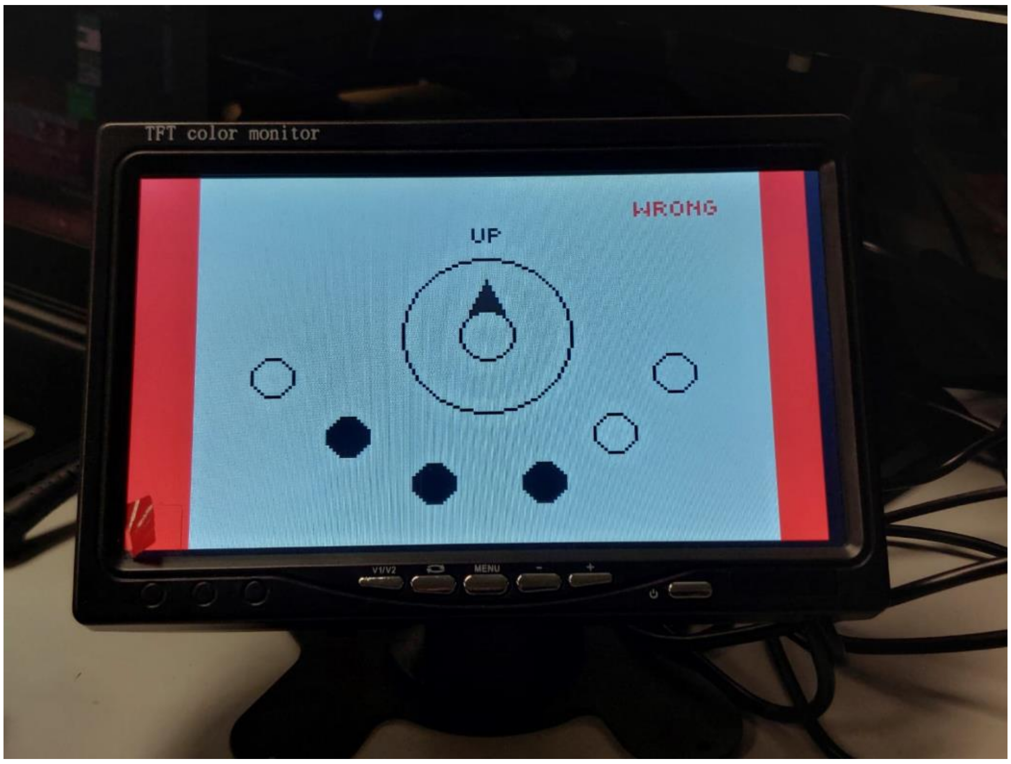
General Game Rule

In our version of the game, the players have 99 seconds to defuse the bomb. There are 3 puzzle modules to be completed – Knob, Keypad and Wires.
If the defuser manages to complete all puzzles within the time limit, the screen will display “defused.” Otherwise, the screen will display “boom.”

General Controls

On start, the defuser is prompted to press BTNC to start the game. This generates random numbers which randomises the puzzles. Then, the defuser uses BTND and BTNU on the ZedBoard to select the puzzle module to control.

Knob



Rule

The screen displays a cryptic dot pattern at the bottom of the screen. The defuser has to turn the dial at the centre of the screen to the correct position corresponding the dot pattern.

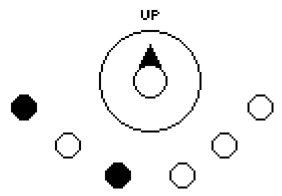


Figure 1. Up pattern

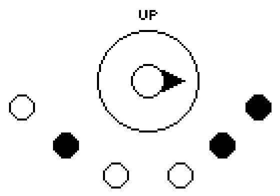


Figure 2. Right pattern

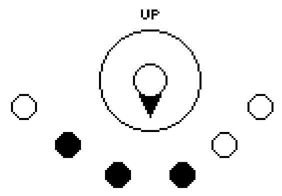


Figure 3. Down pattern

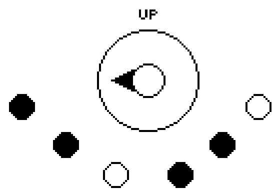


Figure 4. Left pattern

Controls

The defuser uses "Pmod JSTK2" joystick to control the direction of the dial on screen. Pressing the centre joystick button confirms selection of the dial direction. Pressing the black push button freezes the screen to make centre joystick button easier to press.

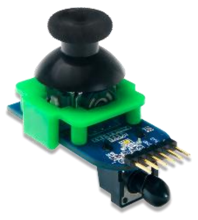
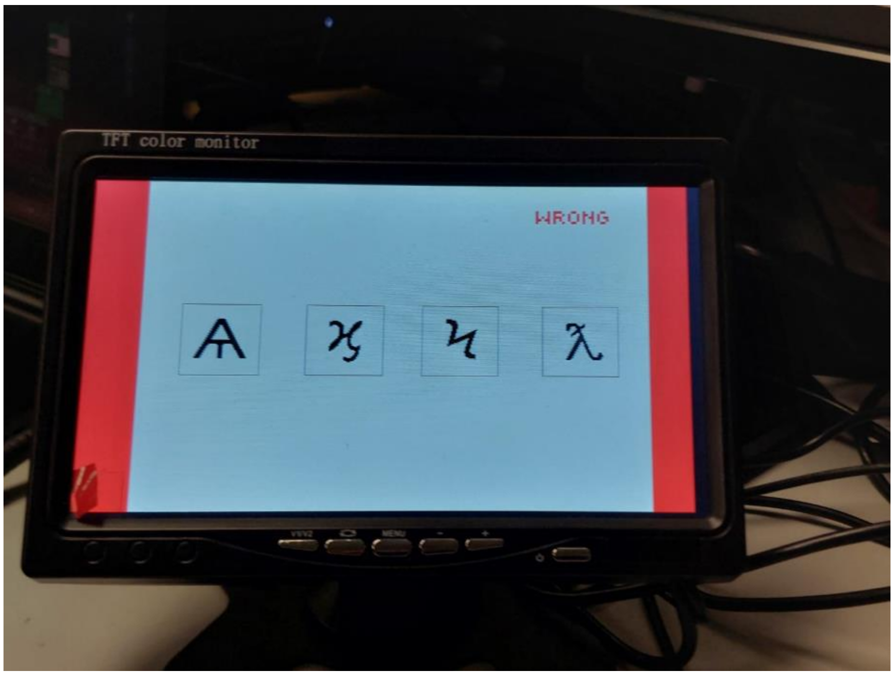


Figure 5. Pmod JSTK2 joystick

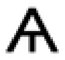
Keypad

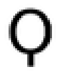


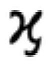
Rule

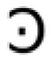
The screen displays 4 random cryptic symbols. The defuser has to describe the 4 symbols to the expert so that the expert can tell the defuser which flip flops on the ZedBoard to turn on. The puzzle is considered completed once the defuser turns on all 4 flip flops correctly.

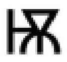
All 7 possible symbols:

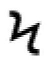
 AT – flip flop 1

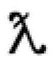
 Balloon – flip flop2

 Hookn – flip flop 3

 Leftc – flip flop 4

 Squidknife – flip flop 5

 Squigglyn – flip flop 6

 Upsidedowny – flip flop 7

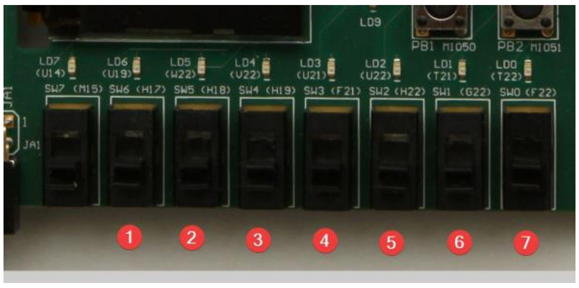
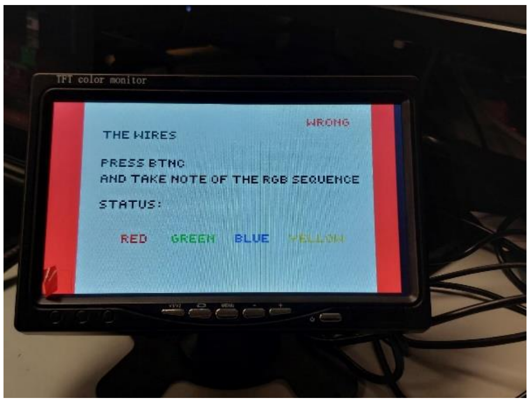
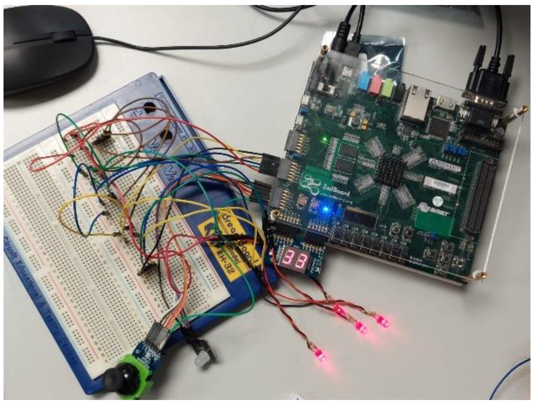


Figure 6. Flip flop positions (SW7 is unused)

Wires





Rules

The defuser is prompted to press BTNC and then take note of the colour sequence shown on the light module. The light module will display a random sequence of 3 colours - red, yellow, and blue. On the other hand, there are 4 colours of wires on the breadboard – red, green, blue, and yellow. The defuser has to disconnect coloured wires on the breadboard corresponding to the colour sequence displayed on the light module.
The coloured text on the screen turns to black once the corresponding wire is disconnected.

All 6 sequences and solution:

Colour sequence on light module	Wires to disconnect
Red Yellow Blue	blue
Blue Red Yellow	green, yellow
Yellow Blue Red	red, blue, yellow
Red Blue Yellow	red
Blue Yellow Red	red, green
Yellow Red Blue	red, yellow

Overall Structure

The following is a finite state machine diagram that shows the overall programme flow.
The puzzle modules are represented as “scenes” and the “selected_scene” integer variable controls which scene to be displayed and controlled by the player.

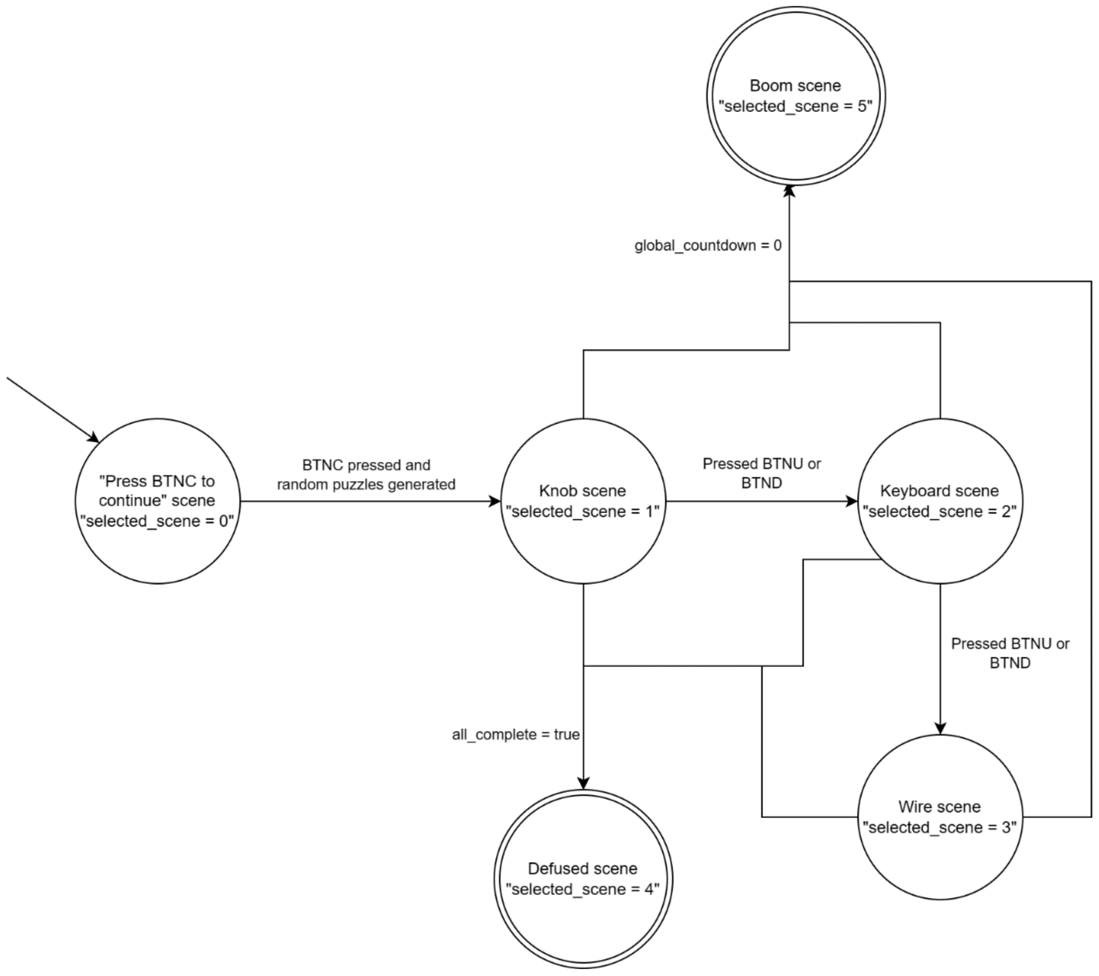

Boom scene "selected_scene = 5"
global_countdown = 0
"Press BTNC to continue" scene "selected_scene = 0"
BTNC pressed and random puzzles generated
Knob scene "selected_scene = 1"
Pressed BTNU or BTND
Keyboard scene "selected_scene = 2"
Pressed BTNU or BTND
all_complete = true
Wire scene "selected_scene = 3"
Defused scene "selected_scene = 4"

Figure 7.1. FSM diagram

And since the main logic is contained in vga_driver.vhd and mainly composed of sequential processes, the following is a block diagram that categorises each process.

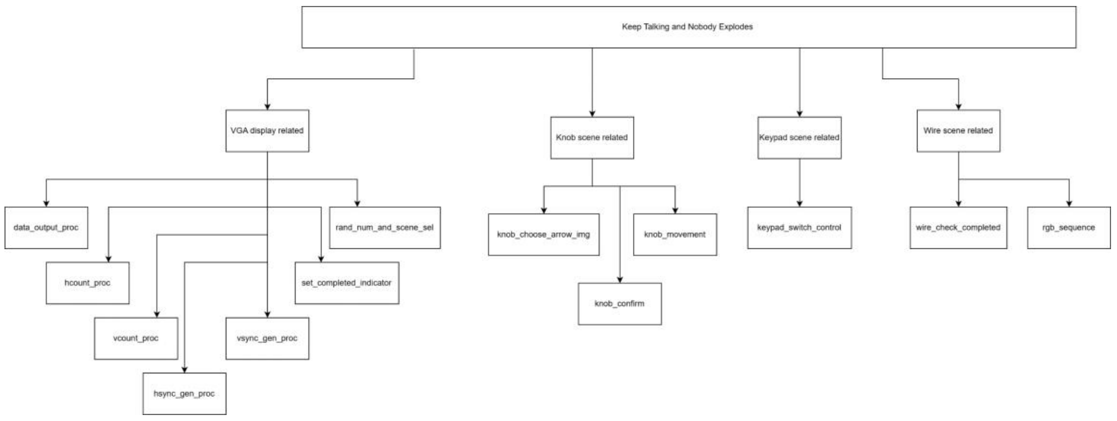


Figure 7.2. Block diagram

Implementation

Hardware used

Input devices: Buttons and flip flop switches on ZedBoard, Pmod JSTK2 joystick, wires on breadboard
Process device: ZedBoard
Output devices: VGA monitor, RGB LED light module, red LED lights on ZedBoard

Programme Implementation Details

Random Number Generation

At the beginning of the game, the player is prompted to press BTNC. This is because when the player presses BTNC, the “vcount” value at the time (used for VGA screen refreshing) is taken as the random number, since the vcount value is always changing and unpredictable. For example, “vcount mod 4” returns a random number in the range of 0 to 3.

VGA Monitor Display

Image Source

The images are drawn using a website called “pixilart”. The canvas size is 168px wide x 120px tall. Then, different image layers are exported as png and converted to “binary art” which represents white as “1” and black as “0”. The image size is enlarged to 840x600 (5x times) in the process. This binary art is used for expressing the colour of each pixel (“1” can represent white, “0” can represent black, green, red, or any other colour as defined in the VHDL code). The binary art files are then embedded into the VHDL code as “bit_vector” type.

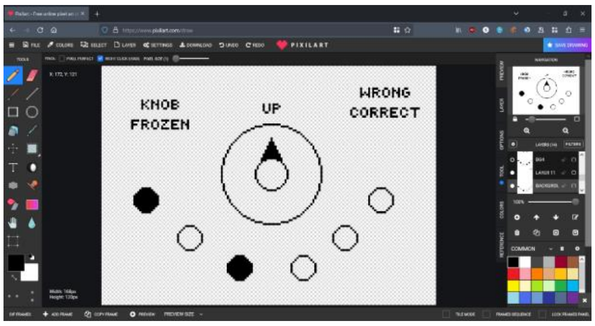
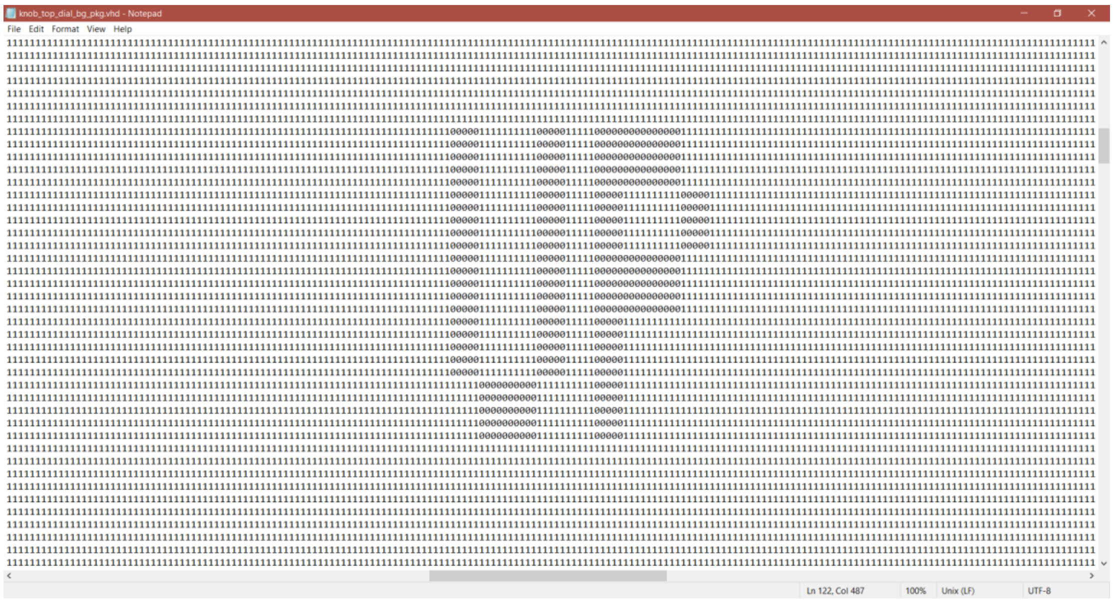


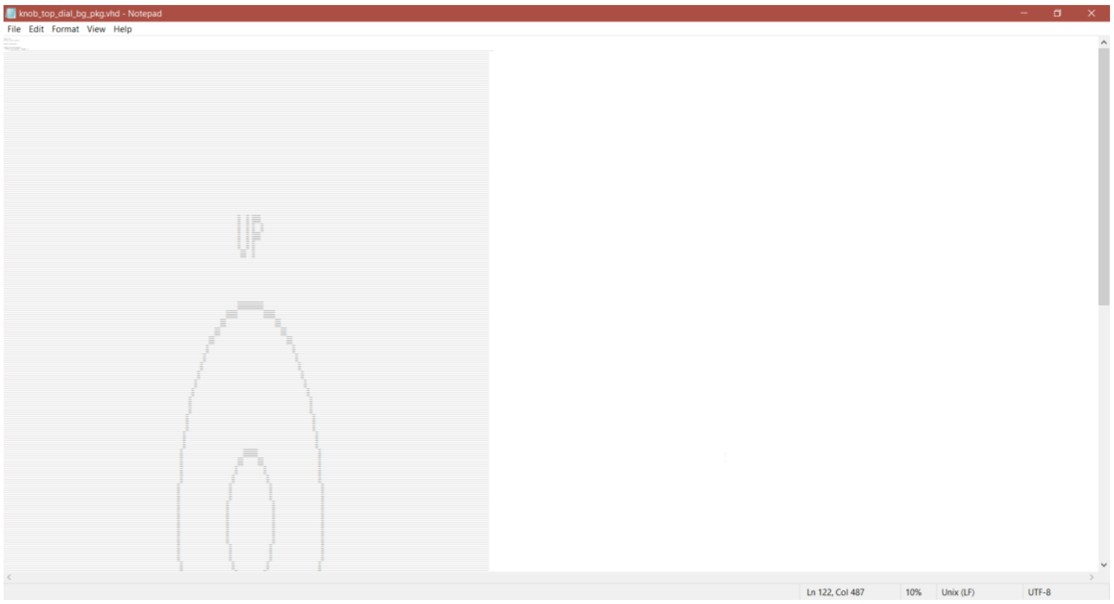
Figure 8. Drawing on pixilart


knob_top_dial_bg_pkg.vhd - Notepad

File Edit Format View Help

Ln 122, Col 487 100% Unix (LF) UTF-8

Figure 9. The “dial” layer from the knob scene, zoomed in


knob_top_dial_bg_pkg.vhd - Notepad

File Edit Format View Help

Ln 122, Col 487 10% Unix (LF) UTF-8

Figure 9. The “dial” layer from the knob scene, zoomed out

Display on VGA

The VGA display part is mainly based on lab5. Video output is handled by the “data_output_proc” process.

Since 840x600 images do not fill the whole screen, vcount is shifted to “vcount – 15” and hcount is shifted to “hcount + 620” to move the image to roughly the centre of the screen.

Multiple layers of images are stacked on top of each other using “or” operators. Colours such as black can be assigned to pixels defined as ‘0’.

Knob Scene Implementation

The knob_movement process reads the x_position and y_position variables returned by the joystick module. According to the values, a value will be assigned to "knob_arrow_sel" to determine which direction the knob has turned to (up, down, right or left). Then, the knob_choose_arrow_img process reads the value of "knob_arrow_sel" and controls the image displayed on the VGA monitor. The knob_confirm process reads the center button input of the joystick module for confirming the dial direction. The trigger_button freezes the screen to allow easier press of the knob button by changing the value of "refresh_screen" (if value is 0, then allow update the knob_arrow_sel; else, freezes the screen).

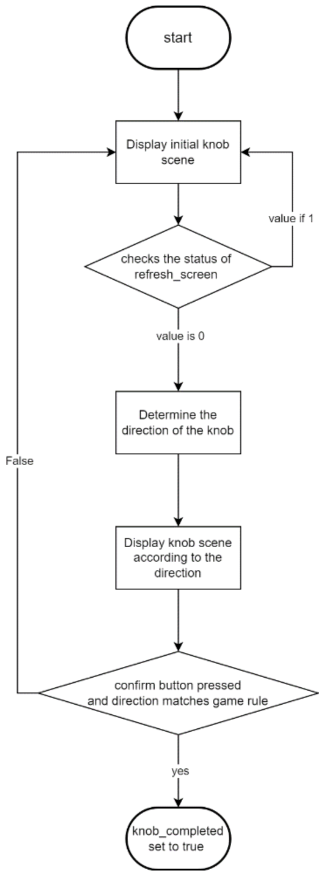

start
Display initial knob scene
value if 1
checks the status of refresh_screen
value is 0
Determine the direction of the knob
False
Display knob scene according to the direction
confirm button pressed and direction matches game rule
yes
knob_completed set to true

Figure 10. Flowchart describing the flow of "knob" puzzle

Keypad Scene Implementation

At the beginning of the game, the programme randomly picks one of 35 strings of 7-bit long bit_vector's, predefined in "keyboard_combination". The 7-bit long bit_vector's represents the 7 possible symbols that can be displayed. Each string only has four '1' bits. The positions of '1' bits pick which symbols to be displayed. This picking is done in a for loop in the "rand_num_and_scene_sel" process. Every time the flip flop switch pattern is changed, the "keypad_switch_control" process checks if the switch pattern matches the randomly selected "keyboard_combination". If match, keyboard_completed is set to true.

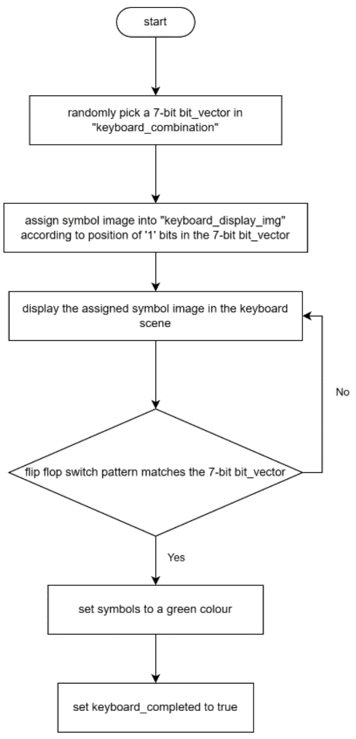

start
randomly pick a 7-bit bit_vector in "keyboard_combination"
assign symbol image into "keyboard_display_img" according to position of '1' bits in the 7-bit bit_vector
display the assigned symbol image in the keyboard scene
No
flip flop switch pattern matches the 7-bit bit_vector
Yes
set symbols to a green colour
set keyboard_completed to true

Figure 11. Flowchart describing the flow of "keyboard" puzzle

Wire Scene Implementation

The rgb_sequence process reads the random number generated (wire_rand_num) to determine the sequence of LED colour to be displayed, which is predefined as a 2D array called “wire_rgb_sqeuence”. When the player starts disconnecting wires on the breadboard, the “wire_check_completed” process checks if the pattern of disconnected wires matches the corresponding LED sequence according to the game rules (mentioned above). If match, wire_completed is set to true.

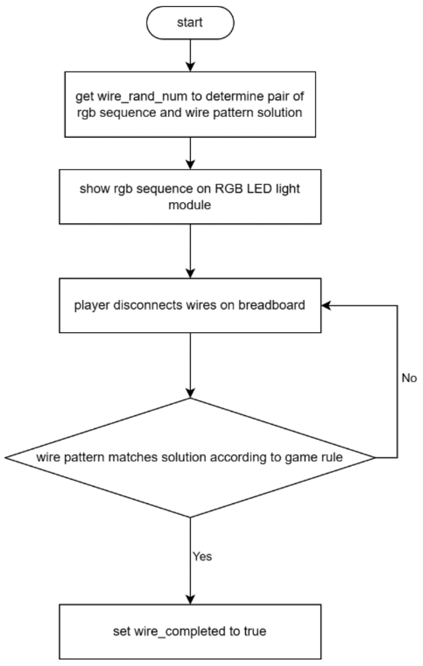

start
get wire_rand_num to determine pair of rgb sequence and wire pattern solution
show rgb sequence on RGB LED light module
player disconnects wires on breadboard
No
wire pattern matches solution according to game rule
Yes
set wire_completed to true

Figure 12. Flowchart describing the flow of “wire” puzzle

Results and Discussions

Overall, we managed to make 3 mini games out of the 5 proposed mini games. We think the gameplay flow and appearance matches what we imagined – an FPGA board that looks like a time bomb in itself, a 7-segment display that is typically seen on time bomb depicted in movies, deciding which coloured wires to be disconnected...

The development is not entirely without obstacles. We think it was difficult to interface with pmod peripherals, particularly the joystick module. It took lengthy research to finally find existing libraries online to make use of it in our project.

As for possible future improvements, we could implement the uncompleted mini games proposed originally. We could also implement a buzzer module for sound effects (adds tension to the game). Moreover, we could add a main menu to choose between mini games.

Division of Labor

Shum Tsz Ching – game design (retrofitting the pc game onto the ZedBoard), pixel art images, VGA output logic, keyboard puzzle module
Luk Hoi Chun – hardware interfacing including joystick, RGB LED module, 7-segment display, physical wire design on breadboard (for the “wire” puzzle module)

Reference

Game referenced:

All Games > Indie Games > Keep Talking and Nobody Explodes
Community Hub
KEEP TALKING and NOBODY EXPLODES
HOLD
MENU

Keep Talking and Nobody Explodes

Find yourself trapped alone in a room with a ticking time bomb. Your friends have the manual to defuse it, but they can't see the bomb, so you're going to have to talk it out – fast!

RECENT REVIEWS: Overwhelmingly Positive (110)
ALL REVIEWS: Overwhelmingly Positive (11,130)
RELEASE DATE: 9 Oct, 2015
DEVELOPER: Steel Crate Games
PUBLISHER: Steel Crate Games

Popular user-defined tags for this product:
Co-op Puzzle Local Co-Op Multiplayer VR +

Steam link:
https://store.steampowered.com/app/341800/Keep_Talking_and_Nobody_Explodes/

Keypad image source: https://ktane.fandom.com/wiki/Keypad

Game rules: https://www.bombmanual.com/web/index.html

Libraries:

Joystick and SPI master library: https://forum.digikey.com/t/joystick-jstk2-pmod-controller-vhdl/28986

AI generated random_number.vhd. Seems unused in the final programme.

Various codes from previous labs in CENG3430