Australian National University - Assignmemnt Cover Sheet

ENGN 6213 FPGA Assignment: Some Fun with FPGA VGA output

Hong-Bo Zhang

College of Engineering and Computer Science, Australian National University, Canberra, 2601, Australia
u6170245@anu.edu.au

Fan Yu

College of Engineering and Computer Science, Australian National University, Canberra, 2601, Australia
u5872700@anu.edu.au

UID	u6170245 u5872700
Course	ENGN6213 Digital System and Microprocessor
Assign No.	Assignment 1, FPGA Design: FPGA VGA output
Group No.	Group 17
Lecturer	Dr Nicolo Malagutti
Date	2017-May-01

Abstract: In this desgin, all four functions in the assignment description are implemented. They are also tested on both FPGA board Spartan3E and Nexys4. All of them function well. Especially, any figure in function 2 and be loaded in function 3. Both source code for Spartan3E and Nexys4 are submitted. ***However, we prefer the Nexys4 version to be assessed.***

*We declare that this work: upholds the principles of academic integrity, as defined in the ANU Policy: Code of Practice for Student Academic Integrity.

Contents

1. Overview **1**

2. User Interface **1**

3. Function 4 All Functions Works Together **2**

3.1 Top Module 2

3.2 Fun4 FSM 2

3.3 Module vgacontroller 3

3.4 Module lookupaddr 3

3.5 Module picrom 4

3.6 Highlighted combinational blocks in top module 5

4. Function 1 Output Uniform Color **5**

5. Function 2 Bit Map Drawing and Carton **5**

6. Function 3 Image Scramble Game **7**

6.1 Top module of function 3 7

6.2 State machine of function 3 7

6.3 Module arrow 8

7. Future Improvement **8**

1. Overview

In this desgin, all four functions in the assignment description are implemented. They are also tested on both FPGA board Spartan3E and Nexys4. All of them function well. Especially, any figure in function 2 and be loaded in function 3. We include the following files in the submitted archive.

- this report.
- all verilog source code with two ucf files and two ise project file $*.$ ise for Spartan3E and Nexys4 respectively,
- all the vector images of our design in pdf format in case that the figures in this report is too small to read.
- bit files for Spartan3E and Nexys4 respectively.
- all the python scripts used to assist this design, we use python to generate some lengthy but routinary verilog code.

The codes can be synthesized and implemented on both Spartan3E and Nexys4 once the relevant codes are uncommented and correct ucf file is used.

We prefer the Nexys4 version to be assessed.

It is recommaned to configure the LCD screen to an aspect ratio of $4:3$ and to mode *Auto Adjust* to get a better display effect. However, our design will also work well in the default setting. This design contains 13 bitmaps, all 13 images are stored in FPGA without using rom or ram. All the bitmap are output in 480×480 pixels via vga.

2. User Interface

The user inputs are five buttons (North, South, East, West and Middle) and four switches. The outputs are three LEDs at right most and vga port. All the user interface and their functions are shown in Fig 1 and Tab 1. We use Spartan3E as an example. The Nexys4 board is identical to Spartan3E. On Nexys4, the IO are the right most three LEDs, right most four switches, five buttons and vga port.

Notice We use East button to change (set) color in function 1, rather than using the North button as mentioned in assignment description. Furthermore, in function 3, we use the left most switch to check whether winning the game or not, which is not specified in assignment description.

Table 1: User interface

Item	Usage In FUN1	Usage In FUN2	Usage In FUN3	Usage In FUN4	Note
Switch 1	Screen on/off	Screen on/off	Screen on/off		Right-most switch
Switch 2	Red signal input	Function Reset	Function Reset		Second switch from right
Switch 3	Green signal input	Play Cartoon	Scramble/Move mode		Third switch from right
Switch 4	Blue signal input		Check if wins		Fourth switch from right
Btn North		Rotate picture	Move/Shift up		North (Up) button
Btn South		Inverse color	Move/Shift down		South (Down) button
Btn West		Previous picture	Move/Shift left		West (Left) button
Btn East	Change Color	Next picture	Move/Shift right		East (Right) button
Btn Middle				Change Function	Central Button
LED [2:0]	LED[0] light-up	LED[1:0] light-up	LED [2:0] light-up		LED[2:0]:Right-most 3 LEDs

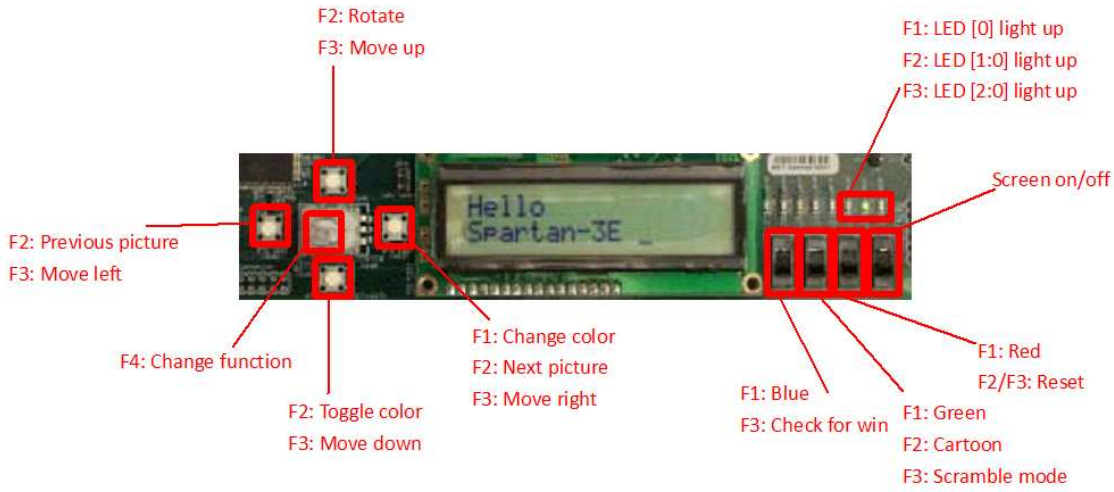

Figure 1: The IO description for a Spartan3E FPGA board.

3. Function 4 All Functions Works Together

3.1 Top Module

The overall design of the whole project is shown in Fig. 2. If this figure is too small to read, you can zoom out the file, or, you can find a vector image fun4.pdf in the submitted files. In this figure, every IO of each module is shown. However, many interconnection between these IOs are omitted for the sake of clarity.

There are two signals which may leads to confuse, pixelEN and romEN. In the porch, boader and sync region, the pixelEN is $1'b0$, while in the display region, it equals to $1'b1$. Since in our design, the bitmap is output in 480×480 pixels, and the vga ouput is 640×480 pixels, so we leave the left 80 pixels and right 80 pixels as blank. In these blank region, romEN is $1'b0$, and in the non-blank display region, romEN equals to $1'b1$.

The module lookupaddr and picrom are used in both function 2 and function 3. Consequently, we isolate them to avoid hardware redundant.

We use the code in HLab3 [1] to develope Module deboucer and doubledebouncer. Since we don't want the system to respond to button combinations, so we use double debouncer which doesn't allow button combinations.

This figure is too complicated, so we will introduct every part in details in the following of this report.

3.2 Fun4 FSM

The state machine of function 4 (top module of this project) is very simple. It has three states, *FUN1*, *FUN2* and *FUN3*, whose name is the same as its meaning. Three states are coded by one-hot coding, since it will simplify the design of many mux and demux in other parts. The transition between states is triggered by pressing the middle button. The outputs of FSM are signals to LEDs (indicating current function) and funSelect [2:0] , the later will be used in many mux and demux of filter/select signals of/to specific function. The initial state is set to *FUN1*. *Notice*: there is no reset in this state machine. The state transition diagram is shown in Fig. 3. It is a Moore machine, so we list the output in this diagram as well.

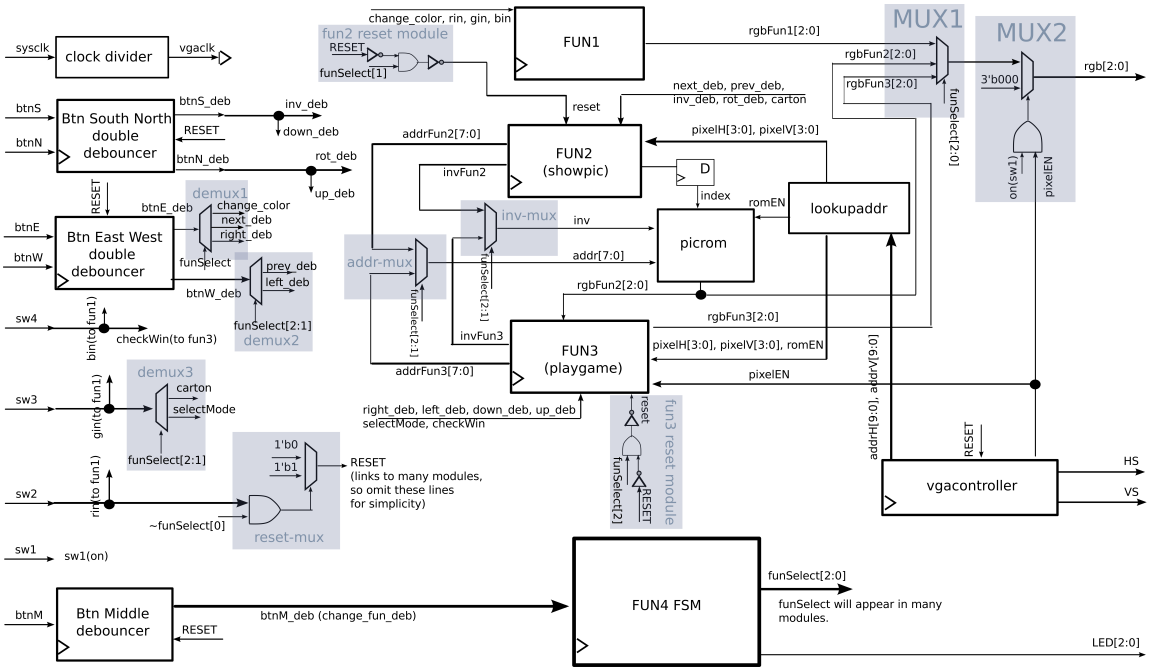

Figure 2: Vector image fun4.pdf can be found in the submitted files. Top module of this project. For clarity, we didn't connect all the wires. All the submodule will be introduced in details in the following of this report.

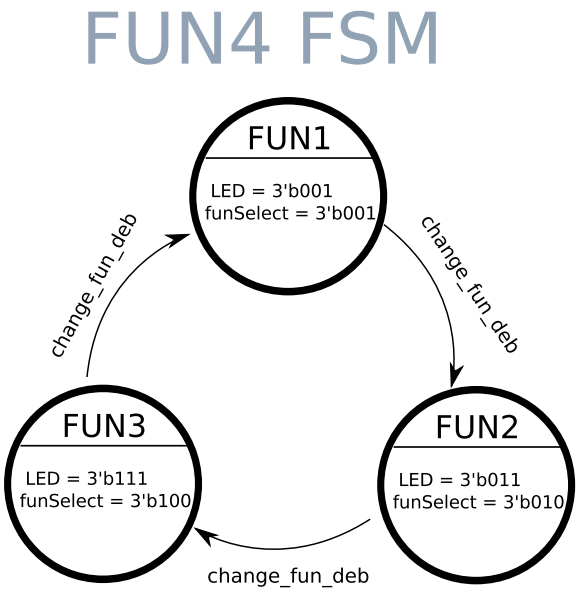

Figure 3: State transition diagram of FUN4 FSM. The output is also listed in the diagram.

3.3 Module vgacontroller

The block diagram of Module vgacontroller is shown in Fig. 4. The output addrH and addrV ranges in $0, 639$ and $0, 479$, respectively.

3.4 Module lookupaddr

The block diagram of Module lookupaddr is shown in Fig. 5. This module will decode addrH and addrV to pixelH and pixelV, which are both ranging from $0, 15$. It also gives the signal romEN as well.

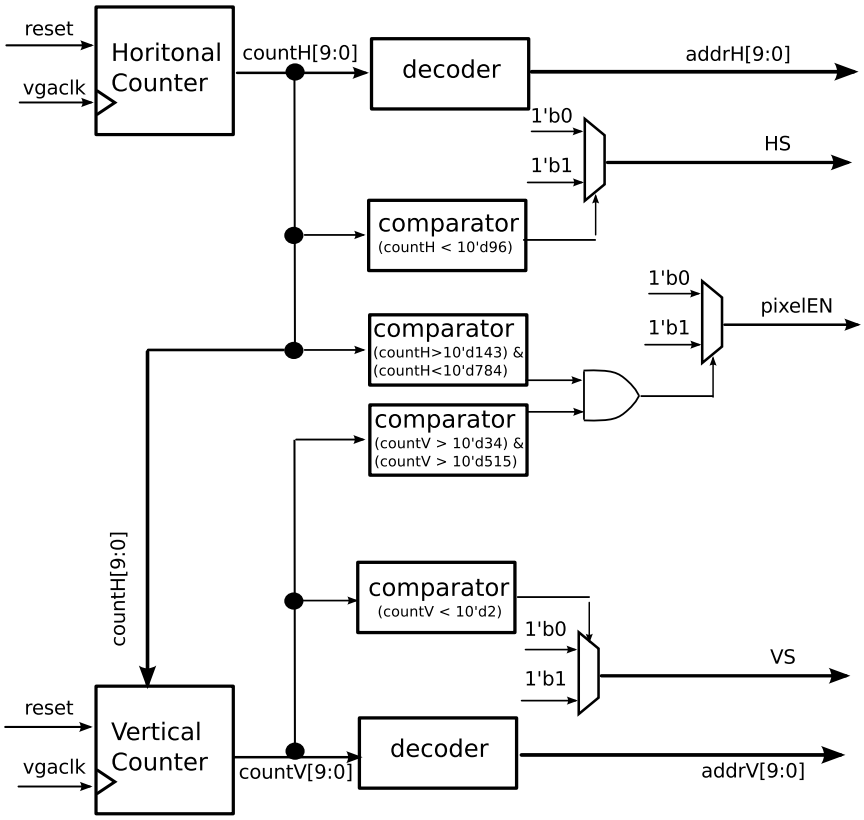

Figure 4: Module vgacontroller block diagram

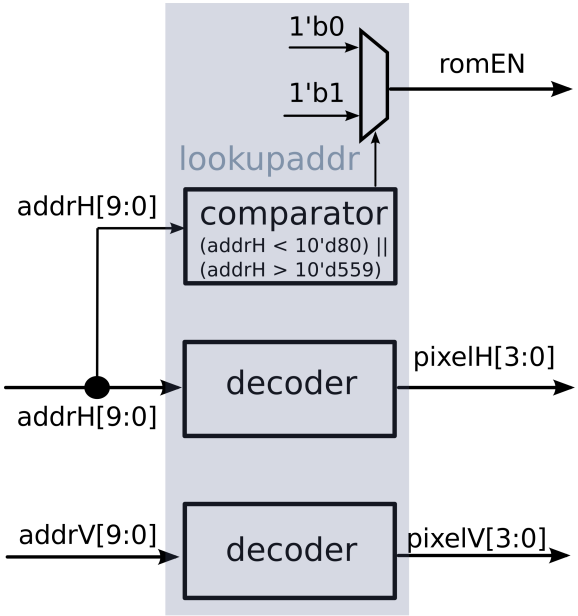

Figure 5: Module lookupaddr block diagram

3.5 Module picrom

The block diagram of Module picrom is shown in Fig. 6. It contains 13 bitmaps in this rom. Although we name this module "rom", we use FPGA only, rather than a real ROM. We use python scripts to generate the rom files (pic_*.v in source code).

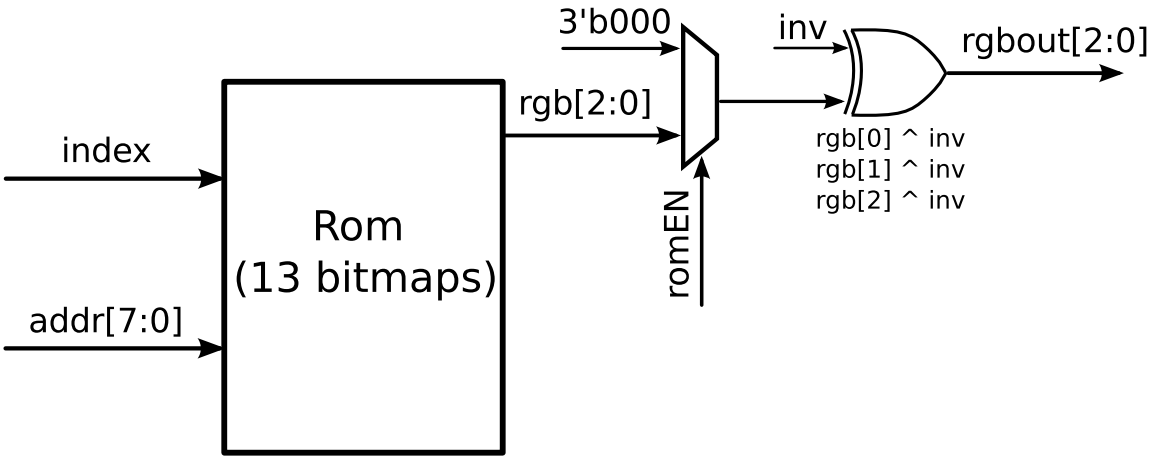

Figure 6: Module picrom block diagram

3.6 Highlighted combinational blocks in top module

There are many small but important combinational blocks in top design (Fig. 2. We will introduct each of them briefly.

- MUX1: select the rgb output according to the state machine among 3 functions
- MUX2: output black if on signal is $1'b0$ in display region.
- add−rmux: select the addr according to the state machine from function 2 and function 3, and give it to rom.
- inv−mux: the same as previous one, but in this case, select inv (inverse color) signal according to state machine.
- demux1 demux2 demux3: filter the input to function 1, 2 or 3, according to the state machine.
- reset−mux: since there is not reset in function 1. so we add this module.
- fun2 reset **module** and fun3 reset **module**: keep one function reset when the state machine is in the state of another function.

4. Function 1 Output Uniform Color

The block diagram of function 1 is shown in Fig. 7. We use three d flip-flop with a mux to store the color information.

5. Function 2 Bit Map Drawing and Carton

The block diagram of function 2 is shown in Fig. 8. We also plot lookupaddr and picrom in the figure, since they are also relevant in function 2.

The getpixel and selectpic are two simple state machines. Their state transition diagram are shown in Fig. 9. We also list the output in the diagram as well. In state machine getpixel, there are two decoupled sub state machine, so I seperate them in Fig. 9. These state machines are so simple that I will not explain them in details. The information provided in Fig. 9 is enough for understanding.

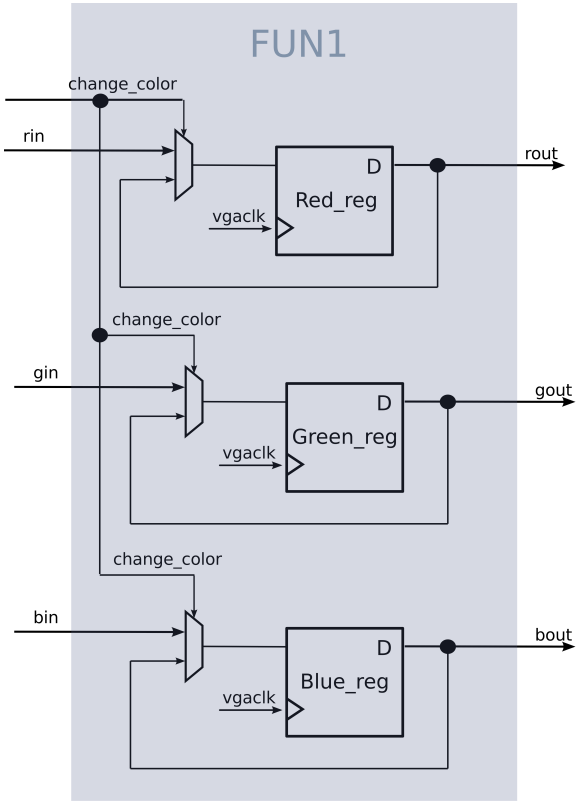

Figure 7: Block diagram of function 1.

The output index is stored in a register, so that when we switch to function 3, the same picture can be displayed and operated in function 3.

The output addrFun2 and invFun2 will become the input of addr−mux and inv−mux for signal selection according to the state machine of function 4. The selection results of these mux's will be input addr and inv of picrom.

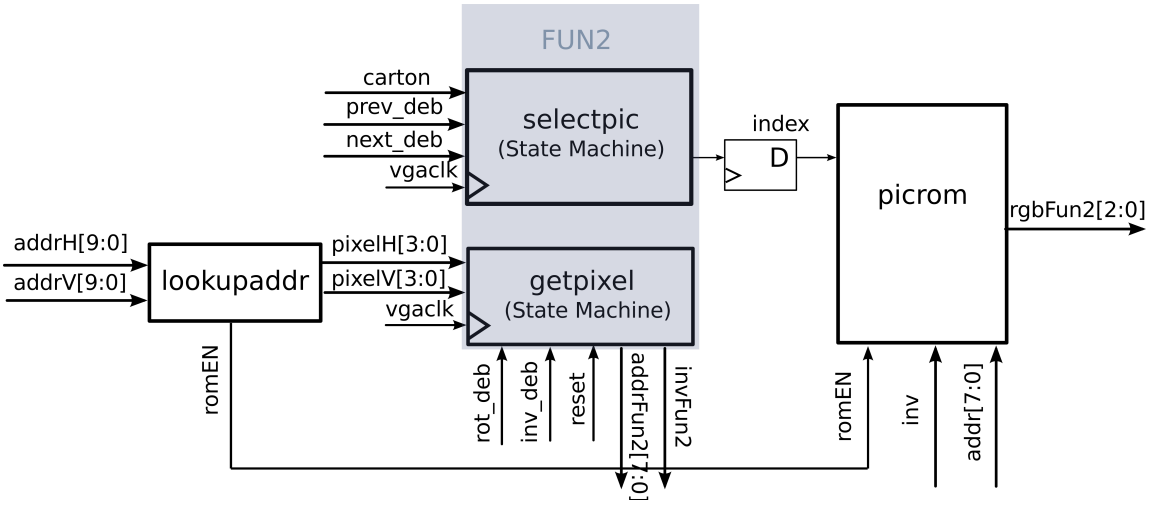

Figure 8: Block diagram of function 2.

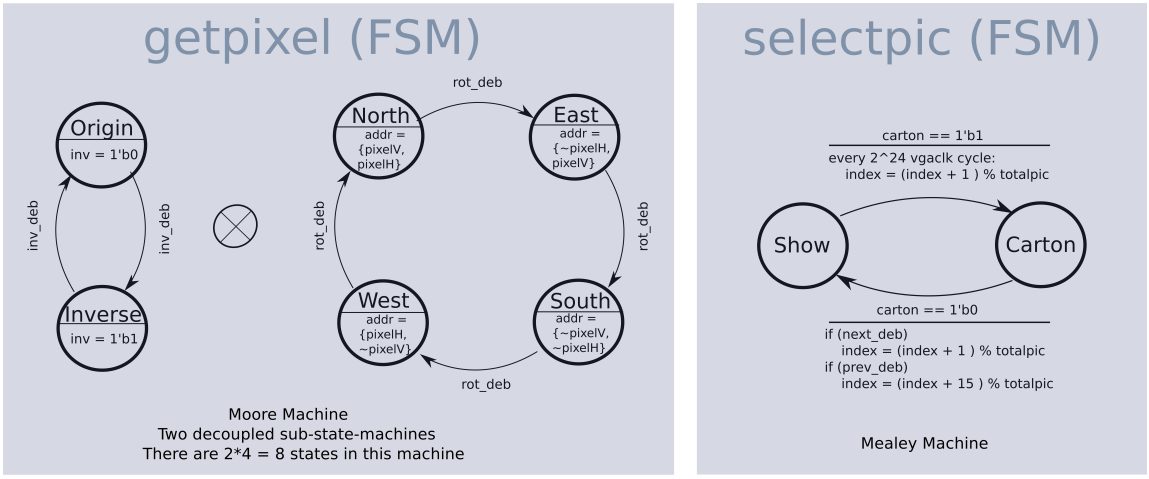

Figure 9: State transition diagram of state machine in function 2.

6. Function 3 Image Scramble Game

6.1 Top module of function 3

Function 3 is a little complicated than previous functions. The block diagram is shown in Fig. 10. Since looupaddr and picrom is also relevant to function 3, so we plot then in the figure.

Similar to function 2, the output addrFun3 and invFun3 will be directed to addr−mux and inv−mux for signal selction. And the results of selection will become input of picrom.

The index comes from a register, so that any image in function 2 can be displayed and operated in function 3.

The gamecontrol is a large state machine, and arrow is a module to output an arrow at correct position on the screen. These two modules will be introduced in following subsections.

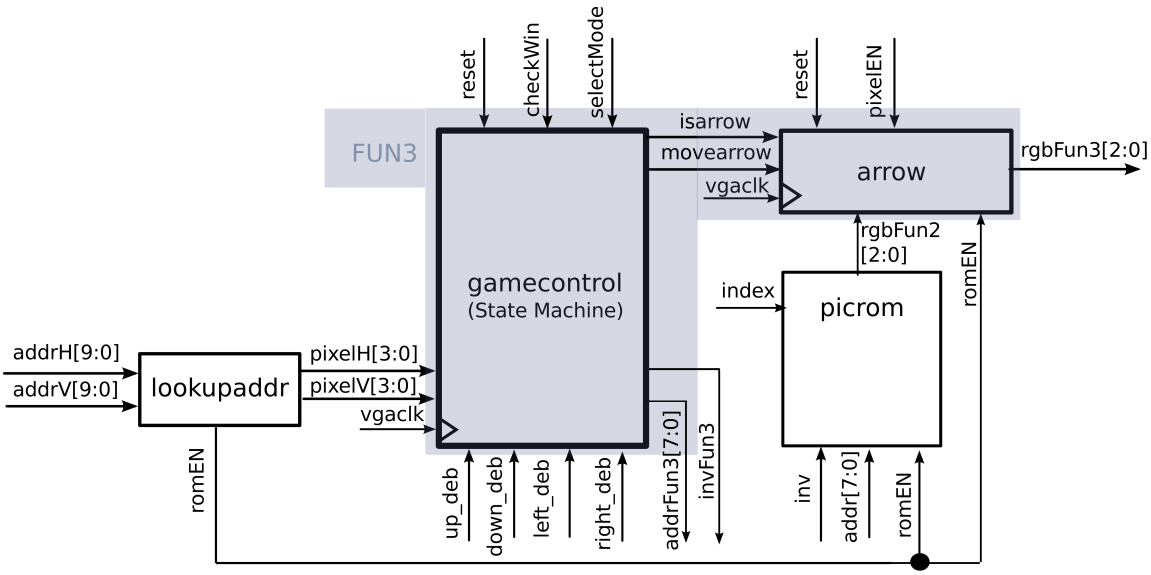

Figure 10: Block diagram of function 3.

6.2 State machine of function 3

There are four states, *Initial*, *Move*, *Scramble* and *Win*. *Initial* is the default initial state,

in this state, the image is just shown on the screen. After one vgaclk (40ns), the *Initial* state will transit to *Move* state. *Move* state is the state in which you can move the cursor. Once the switch selectMode is pulled, *Move* state will transit into *Scramble* state. In the *Scramble*, the image can be scrambled. If one wants to check whether he/she wins or not, he/she can pull switch checkWin. If it wins, state will transit into *Win* state. From any state, once reset is triggered, it will return *Initial*. We use grey code to encode the states.

Since the state machine of function 3 is so complicated, so we show the design of it in Fig. 11. And we also show the state transition diagram in Fig. 12 as well.

(We will discuss the further improvement of this state machine in the future in final section.)

In the design of this state machine, besides the ordinary *next state logic*, *state memory* and *output logic*, we also have *arrow counter*, *inv counter* and *imgScr mem* as well. The *arrow counter* is a counter used to generate the output isarrow and movearrow, which will be used in Module arrow to display an arrow on screen. The *inv counter* is a counter used to generate the output inv, which will be used in *Win* mode to blink the image. In the *imgScr mem* module, we will maintain and operate on a 2048 bits register imgScr[2047:0]. This register stores all the information of a scrambled image. Since there are $16 \times 16 = 256$ pixels in an image, and the image is scrambled, we can not simply look up its rgb value by these 256 addresses from rom. Therefore, for each pixel, we need 8 bit to store its address in picrom. Consequently, we need $256 \times 8 = 2048$ bits in total. In *Scramble* state, when a button is pressed, the imgScr will update itself by a complicated 2-dim shifting operations. If imgScr equals to its original value, which is set to be $2048'hfffefdfc...0403020100$, and switch checkWin is pulled, the condition signal (condition for winning) will be $1'b1$, and it wins. We use python scripts to generate these tedious but routinary verilog code of updating imgScr.

The register imgScr will be updated according to the following rules: in *Scramble* state, if right button is pressed, The $\{128 \cdot i + 7, 128 \cdot i\}$ bits (where i is arrowV) in imgScr will shift 8 bits to high rank, and the highest 8 bits will wrap around to the lowest in this index range. If the left button is pressed, the same range will shfit 8 bits to low rank, and the lowest 8 bits will wrap around to the highest in this index range. If the up button is pressed, the $\{128 \cdot i + 8 \cdot j + 7, 128 \cdot i + 8 \cdot j\}$ (where i is arrowV, j is arrowH) will shift to $\{128 \cdot (i-1) + 8 \cdot j + 7, 128 \cdot (i-1) + 8 \cdot j\}$ and the $i = 0$ will wrap around. If the down button is pressed, the operations are almost the same as up button case, except $(i-1)$ is changed to $(i+1)$ and the $i = 15$ will wrap around.

In Tab. 2, we show the state-output table. The output is expressed in the form of input, internal counters (arrowH, arrowV, invcount) and registers (imgScr).

6.3 Module arrow

The final module in this design is Module arrow, which is shown in Fig. 13. We prepare a 30×30 arrow bitmap by ourselves. The meaning of romEN and pixelEN has been discussed in section 2. The input isarrow is to indicate whether this 30×30 block has cursor (arrow) or not. The input movearrow equals to $1'b1$ when the arrow moves, so we need to reset the arrow counter. If the pixel to display is the location of arrow, the output rgb will be nxor of arrow rgb and image rgb.

7. Future Improvement

In the state machine of function 3 should be improved as following. The output logic of

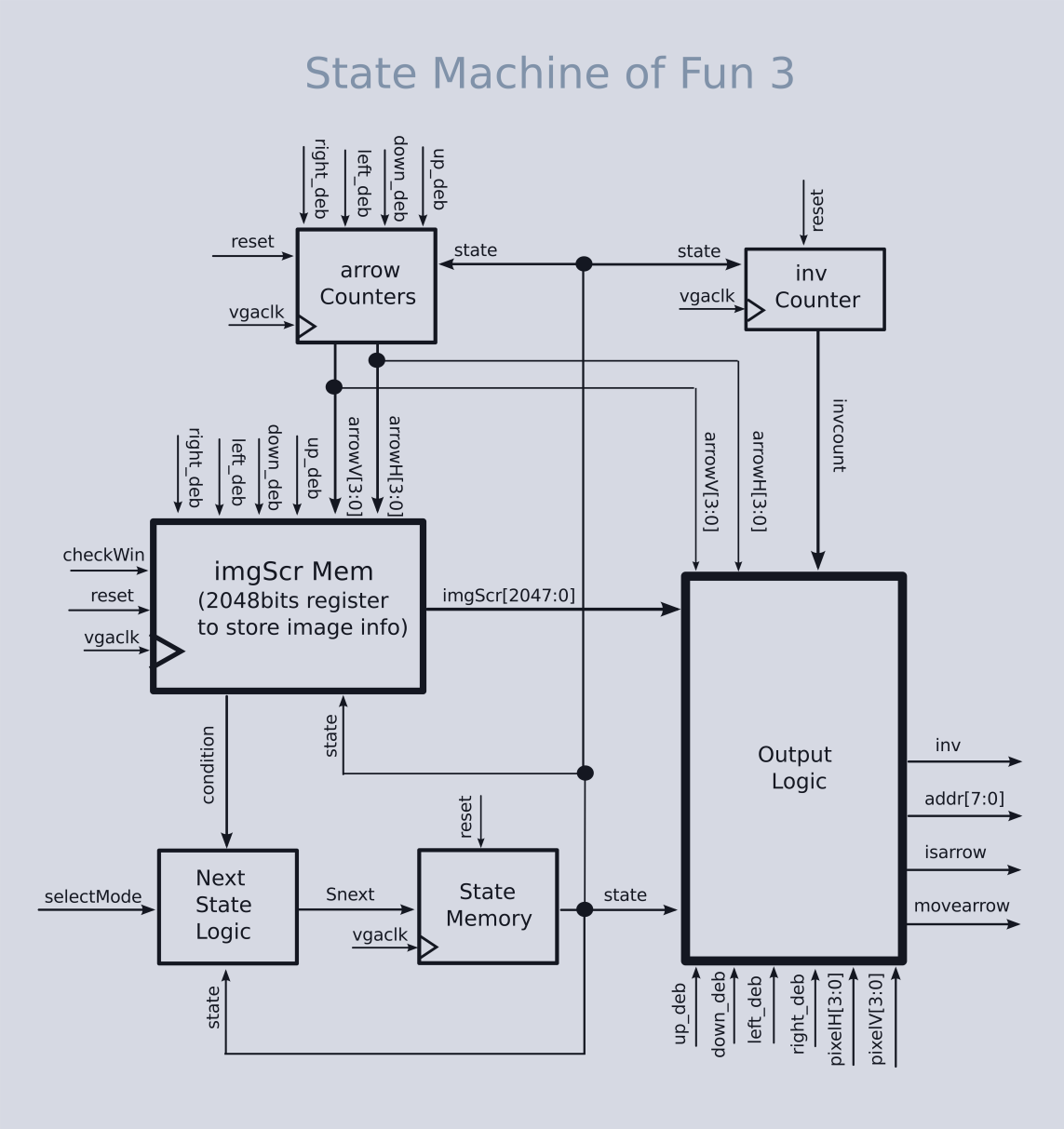

Figure 11: Block diagram of state machine in function 3.

state machine gives output move[1:0] to arrow Counter and imgScr mem. And at the same time, we can remove the up down left right inputs of arrow Counter and imgScr mem. The output move represents the direction of moving. In this case the state machine and module arrow Counter and imgScr mem could be decoupled better.

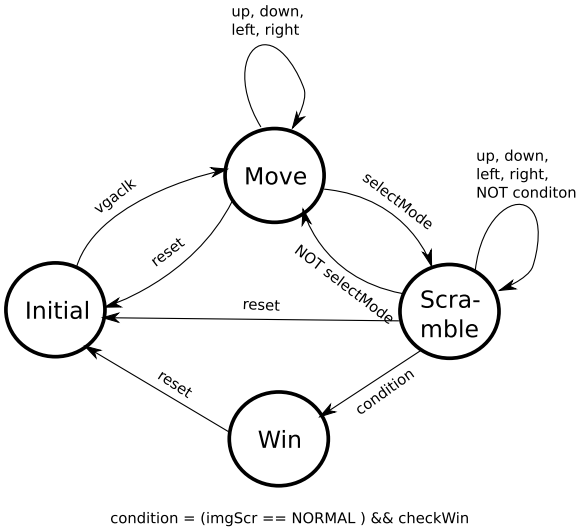

Figure 12: State transition diagram of state machine in function 3.

Table 2: State output table of Fun3 State Machine. Output is expressed as a function of input, where arrowH, arrowV, invcount and imgScr are internal counters and registers in this state machine

State	output as a function of input
Initial	addr = {pixelV, pixelH} inv = 1'b0 isarrow = 1'b0 movearrow = 1'b0
Move	addr = imgScr[(128·pixelV+8·pixelH+7):(128·pixelV+8·pixelH)] inv = 1'b0 isarrow = (arrowH == pixelH) && (arrowV == pixelV) movearrow = (up ‖ down ‖ right ‖ left)
Scramble	addr = imgScr[(128·pixelV+8·pixelH+7):(128·pixelV+8·pixelH)] inv = 1'b0 isarrow = (arrowH == pixelH) && (arrowV == pixelV) movearrow = 1'b0
Win	addr = {pixelV, pixelH} inv = (invcount $\leq 2^{23}$) ? 1'b0 : 1'b1 isarrow = 1'b0 movearrow = 1'b0

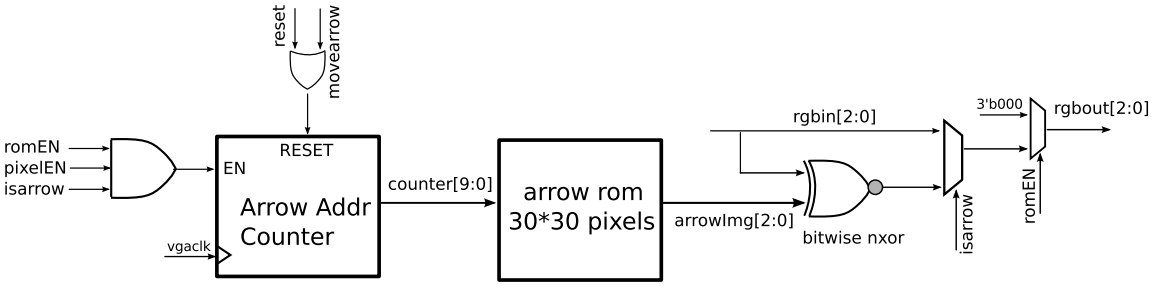

Figure 13: Block diagram of Module arrow.

Acknowledgments

Thanks the lecturer Dr. Nicolo Malagutti and all the tutors in this course.

References

[1] Malagutti, N., 2017, HLab3 code of ENGN6213 digital system and microprocessor, course material, ANU.