CSC258 Computer Organization

HELICOPTER GAME

Final Report

Ajit Pawar
Lab station: #74

COMPUTER SCIENCE

UNIVERSITY OF TORONTO

April 05, 2013

1. Introduction

For our CSC258 project, we have built a helicopter game, whereby the user controls a virtual helicopter on the screen and guides it through a set of oncoming obstacles. The aim of the game is to have the user fly as long as possible without collision. The movement of the helicopter is controlled by a switch, which when flicked up or down, navigates the helicopter vertically on the screen. To keep the game interesting, five levels were implemented which incrementally increase the speed of the oncoming obstacles. As the levels go up, the user finds it increasingly harder to navigate these fast-moving obstacles. The final score is determined by the number of seconds the user stays in the air without collision and is shown on the hex display.

2. Planning

Our project was entirely implemented on the Altera DE2 FPGA and used a combination of schematic and Verilog to describe our circuit. To handle the complexity of our design, we approached it in a modular fashion by dividing it into six modules, namely Control, Paint, Offset, Score, Level, Collision and VGA. Each module is responsible for accomplishing a different task in the game. To complete the project in the allotted three weeks, we set the following milestones listed below:

- Draw a helicopter on the screen and move it based on user-input
- Draw three blocks (obstacles) on the screen and move them independently across the screen while maintaining user-control of the helicopter
- Detect collision between helicopter and all three obstacles. Implement levels and a scoring system

3. Design

In the following section, each of the seven modules is described in detail. Diagrams, screenshots and schematics can be found in the appendix.

Control Module (FSM)
Following are the nine states of our FSM (see Fig.1, Appendix):

1. Draw Helicopter
   This state basically allows us to draw the helicopter on the screen by taking in the initial position of the helicopter (0,0) and then plotting it. The colour to be plotted is retrieved from the RAM. This state is later used to redraw the helicopter at a different x-y location, to show that the helicopter has moved. When drawing is done, the next state Draw A is called.

2. Draw A
   Draw A is responsible for drawing the first of the three obstacles displayed on the screen. The procedure for drawing an obstacle is to enable the Paint module (which handles the process of drawing objects) and wait until the block is fully drawn. Once confirmation is received, then we move to the next state Draw B.

3. Draw B
   Draw B is used to draw the second obstacle. It uses the same code as Draw A, except that it signals the Paint module to draw this specific block. The colour and position parameters of this block are different from the previous one. Draw C is called when execution is done.

4. Draw C
   This state applies the same idea as Draw A and Draw B; however, it calls the Hold state after finishing drawing.

5. Hold
   This state makes use of a timer to output a 1Hz frequency signal. The purpose of this signal is to slow down the process of drawing and erasing on the screen so that the human eye can detect changes.

6. Erase
   The Erase state works by painting the entire screen black ie. changing the color of all the pixels on the screen to black (hex #000). This erases anything that was drawn on the screen. After Erase is done, the next state is assigned to Movement.

7. Movement
   In this state, the x-y coordinate values of all the objects on the screen are incremented or decremented to reflect their movement. For the helicopter, this depends on the input provided by the user. Similarly, the positions of the obstacles are also changed as they move simultaneously toward the helicopter. The Check state is called next.

8. Check
   Using the current position of the helicopter and the obstacles, this state will check for a collision on all possible sides of each block. If collision occurs, the next state will be assigned to End Game. Otherwise, we go back to the first state Draw A.

9. End Game
   In this state, the score freezes and the objects are no longer drawn or erased on the screen. The FSM perennially stays in this state until user resets the game.

Paint Module

As the name suggests, this module is responsible for drawing (or painting) objects on the screen. It stores the start location (top-left pixel) of each object to be drawn alongwith their respective width, height and colour. To draw an object, it sends these values pixel-by-pixel to

the VGA which then plots it on the screen. The paint module is aware of which block to draw based on signals from the FSM. See Fig.3, Appendix.

Offset Module
This module is a custom adder/subtractor whose sole purpose is to add or subtract from the current x-y coordinates of all the objects on the screen. For the helicopter, this means taking in the user input and incrementing or decrementing its y-value based on the position of the switch (up=increment and down=decrement). Only the y-value is changed since the helicopter moves only vertically. For the obstacles, it always decrements their x-value since they are moving from right to left (towards the origin). See Fig.4, Appendix.

Score Module
The score module is a simple up-counter that counts up from zero once the game begins. This score is displayed to the user using 7-segment display. When a collision occurs, the counter holds its current value until game is reset. The source clock for this module is a 1Hz clock, which means the score increments every second.

Level Module
The level module keeps track of the current "level" of the game. It is also a up-counter. It takes the score as an input and increments the level by one every 15 seconds. The levels begin at zero and go up until five. Once it reaches five, it holds this value perennially until the game is reset.

Collision Module
This module detects collision between the helicopter and the obstacles on any of its four sides. It takes in the x and y co-ordinates of each obstacle and compares them to that of the helicopter. Using a comparator, each pixel along the four edges of the helicopter is checked against those of the obstacles. The output of this comparison is the collision signal. See Fig.5, Appendix.

VGA Module
The VGA module is responsible for plotting pixels on the screen. It takes a 7-bit x-coordinate value, a 6-bit y-coordinate value and a 3-bit colour value from the Paint module. The 7-bit value (0 to 128) represents the horizontal axis of the screen while the 6-bit value (0 to 64) represents the vertical axis of the screen. Although the screen resolution is 640 x 480, the actual image drawn is of a resolution of 128 x 64. This is due to memory constraints of the DE2 RAM, which is further explained in the section below. The Paint module, in tandem with the FSM, instructs the VGA module to draw a single pixel of a specific colour at a specific location. This process is repeated in loops to draw blocks.

4. Challenges

The challenges we met while working on this project were both technical and non-technical in nature. Some of the technical challenges were related to designing the flow of execution and DE2 memory constraints.

Designing the flow of execution meant figuring out how and when the different modules interacted with each other. As some modules were dependent on others for input, it became apparent that there needed to be a standard way for modules to communicate with each other. Hence, a "handshake" protocol was implemented. However, this protocol quickly grew complex and timing issues started to creep up. For example, randomizing the sizes of obstacles was initially considered, but quickly dropped in favour of a constant size for all blocks. The problem here was that when the obstacles were drawn, their current height and width was passed on to the Collision module. While this module takes a few clock cycles to process this information, the obstacles currently on screen will have changed size. Waiting for the Collision module to finish its task would have made the obstacles on screen appear to have "stalled" momentarily, making the animation look jittery. The only solution here would have been to change the existing protocol, which was a tedious and time-consuming task. Another technical problem was the limited on-board memory of the DE2 board. Parameters such as pixel resolution and colour depth were directly impact by this constraint. The resolution of our game was a paltry 128 x 64, despite the VGA supporting upto 640 x 480. We were also forced to reduce our colours to a 3-bit value, meaning that only a max of 8 different colours could exist in our game.

5. Lessons Learnt

One of the biggest, but subtle lessons we learnt from this project was to understand the contrast between hardware and software. Inherently, things in hardware are sequential as opposed to the parallel nature of computing we often taken for granted in software. Fetching, reading, writing, executing, all these tasks are performed sequentially in hardware and hence need to be planned carefully. Modules can be built to work independently (and hence assumed to work in parallel), but their output must be gathered and analyzed sequentially. There must be a sequential flow to the execution process and often this is hard to design. If we were to redesign our circuit, what we would do differently is to estimate, using timing analysis, the approximate time needed for each module. Then, we would implement a communication protocol using this specific information such that there would be no race conditions between our modules. This would allow us to expand the scope our project to include more complex features such as random obstacle sizes and random block placement.

6. Conclusion

Working on this project was a thorough experience in hardware design. This project put to test all the concepts of logic design taught in class. It also allowed us to challenge our understanding of it and experiment with it. Using counters, timers, modules and FSM, we were successfully able to design and implement a game that was simple yet academically challenging. It gave us a great sense what it means to work on a project – achieving milestones, meeting deadlines and team work. We feel that this project brings a closure to the generally positive learning experience we had in this course. We hope to build on the knowledge gained here and share with our fellow friends the virtues of understanding things at the most fundamental level.

Appendix

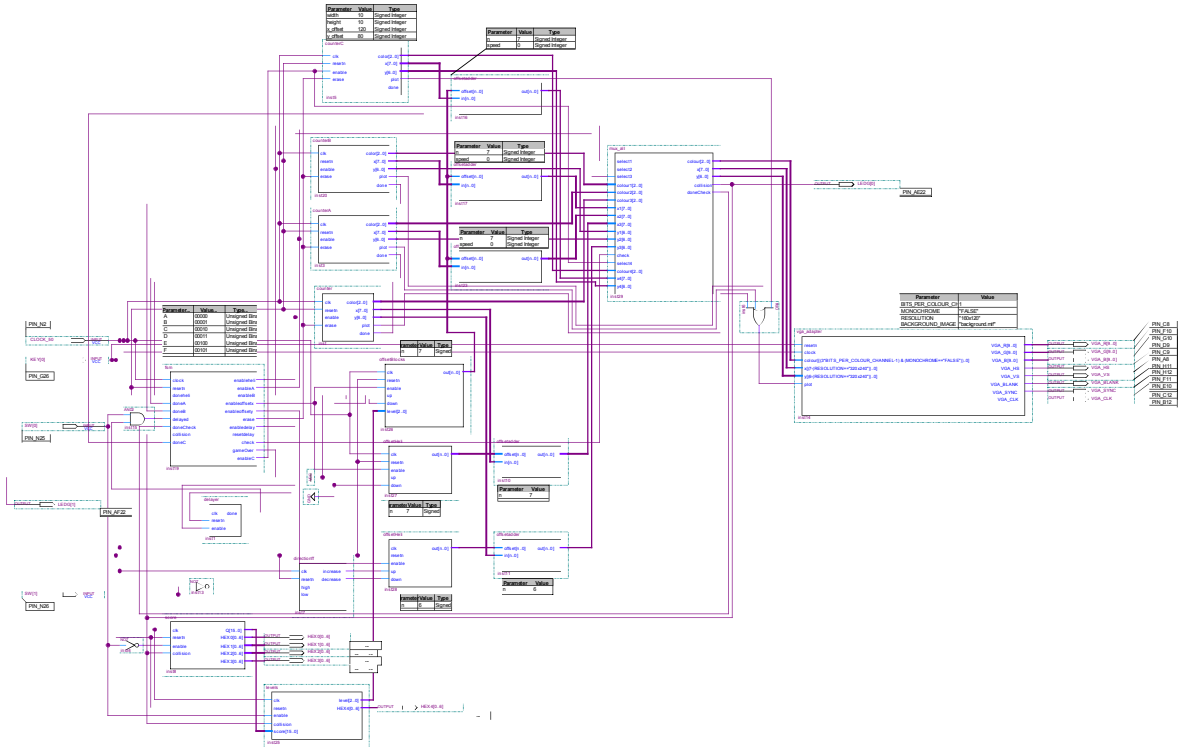

Fig.0 Schematic layout

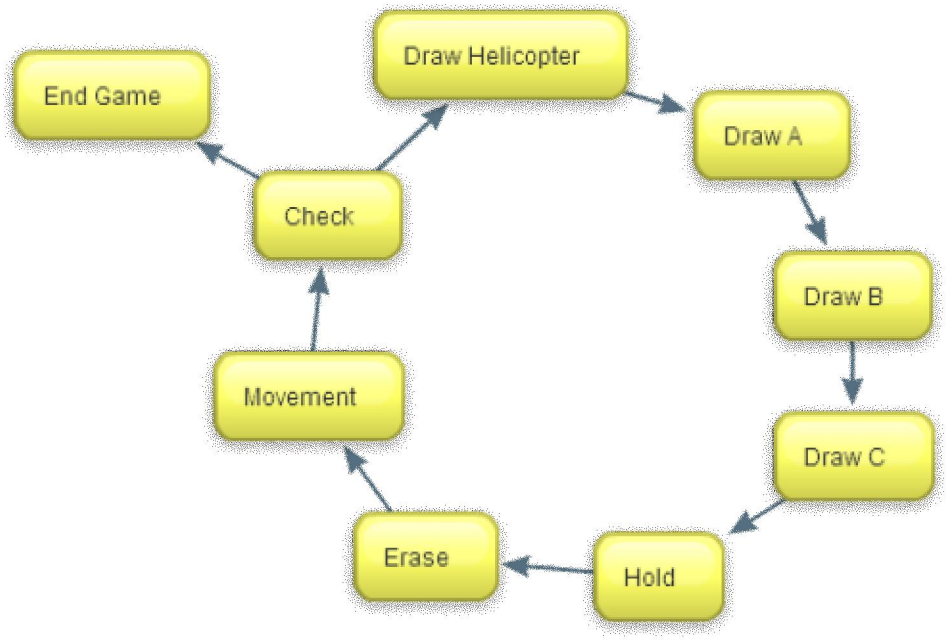

Fig.1 State-diagram of the FSM

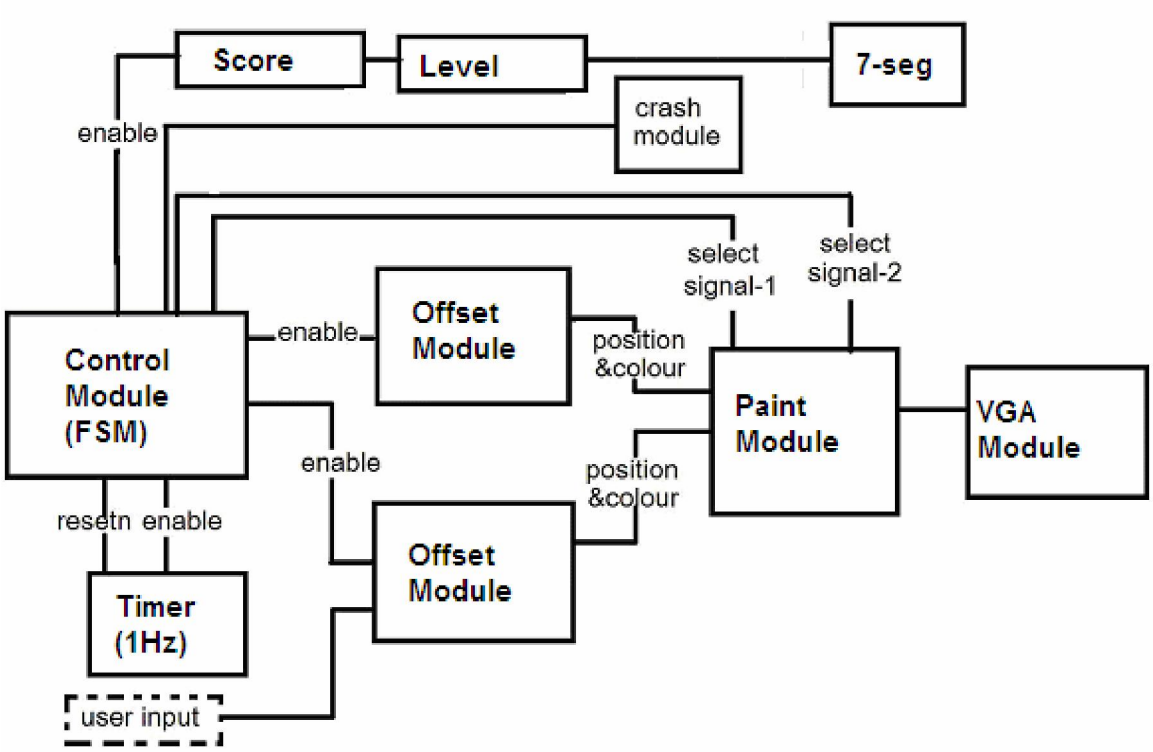

Fig.2 Block diagram of the overall circuit

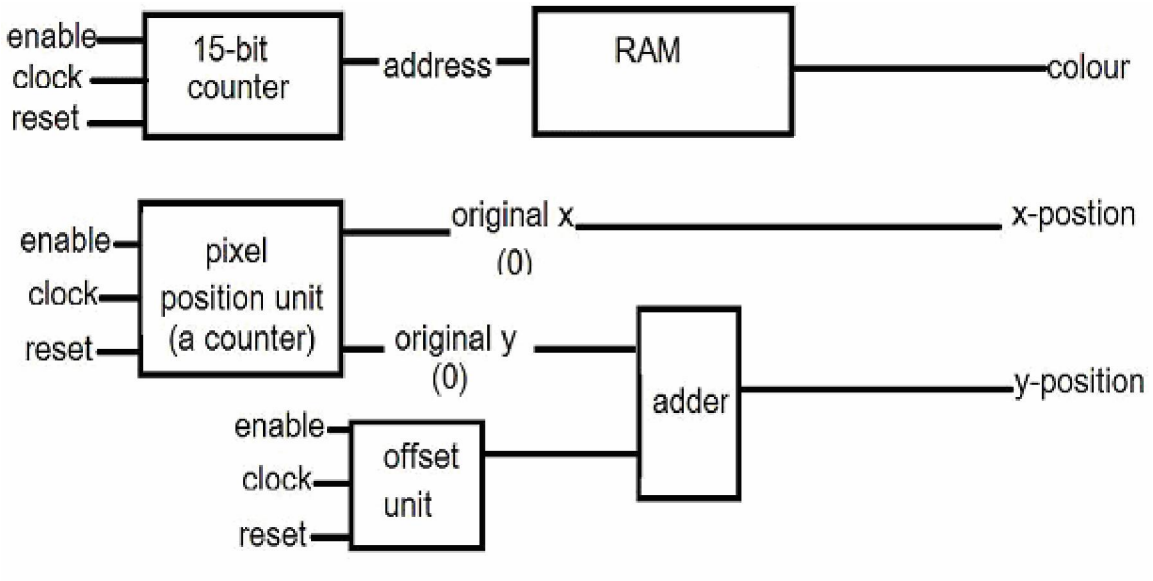

Fig.3 Block diagram of the Paint module

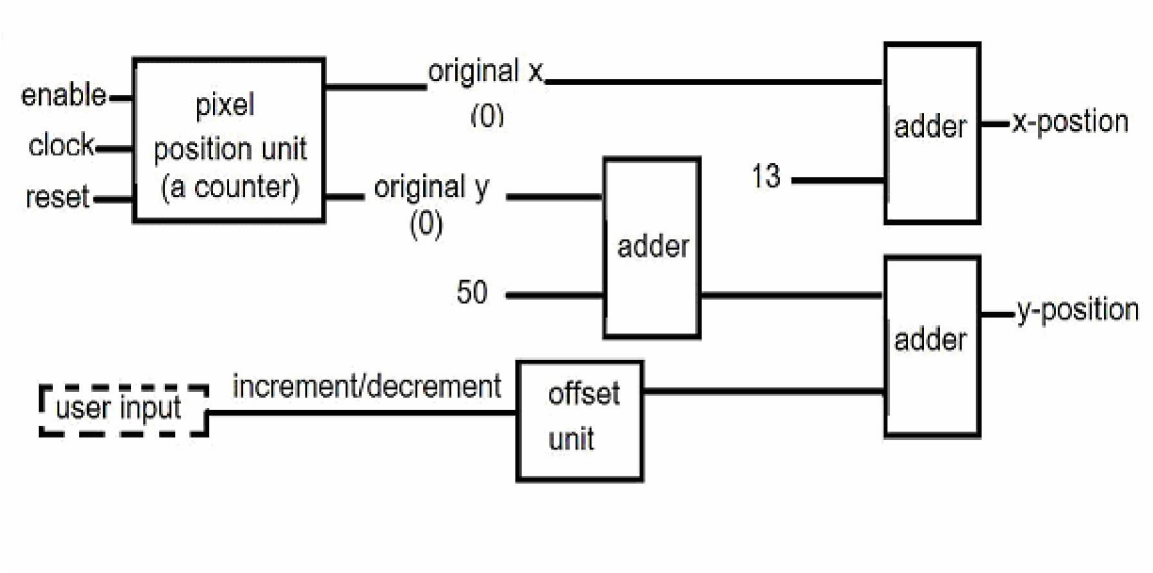

Fig.4 Block diagram of the Offset module

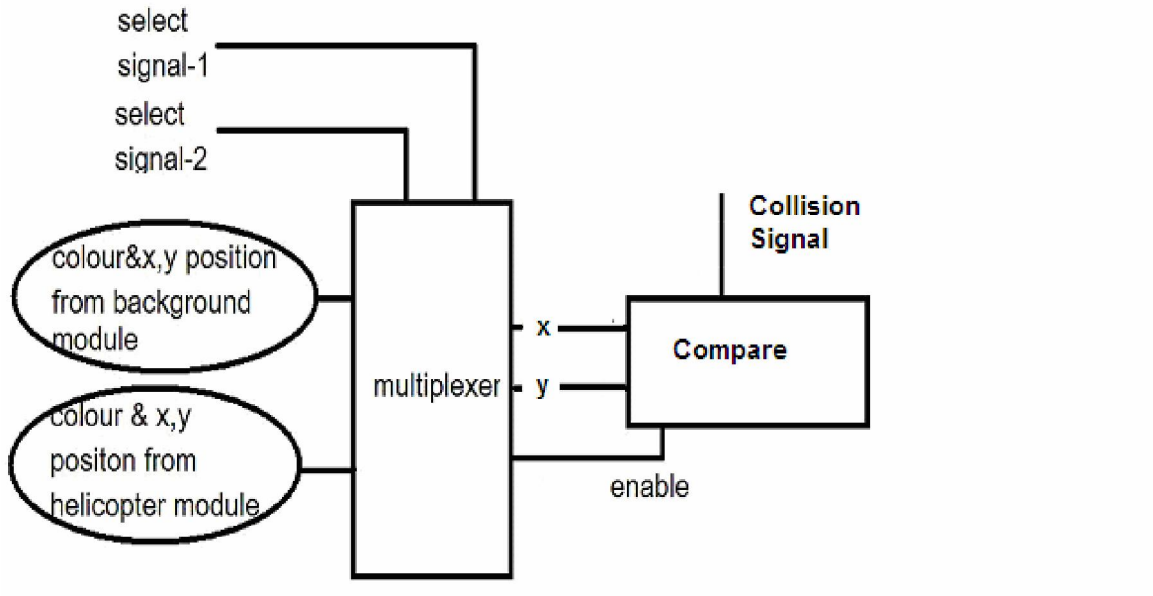

Fig.5 Block diagram of the Collision module

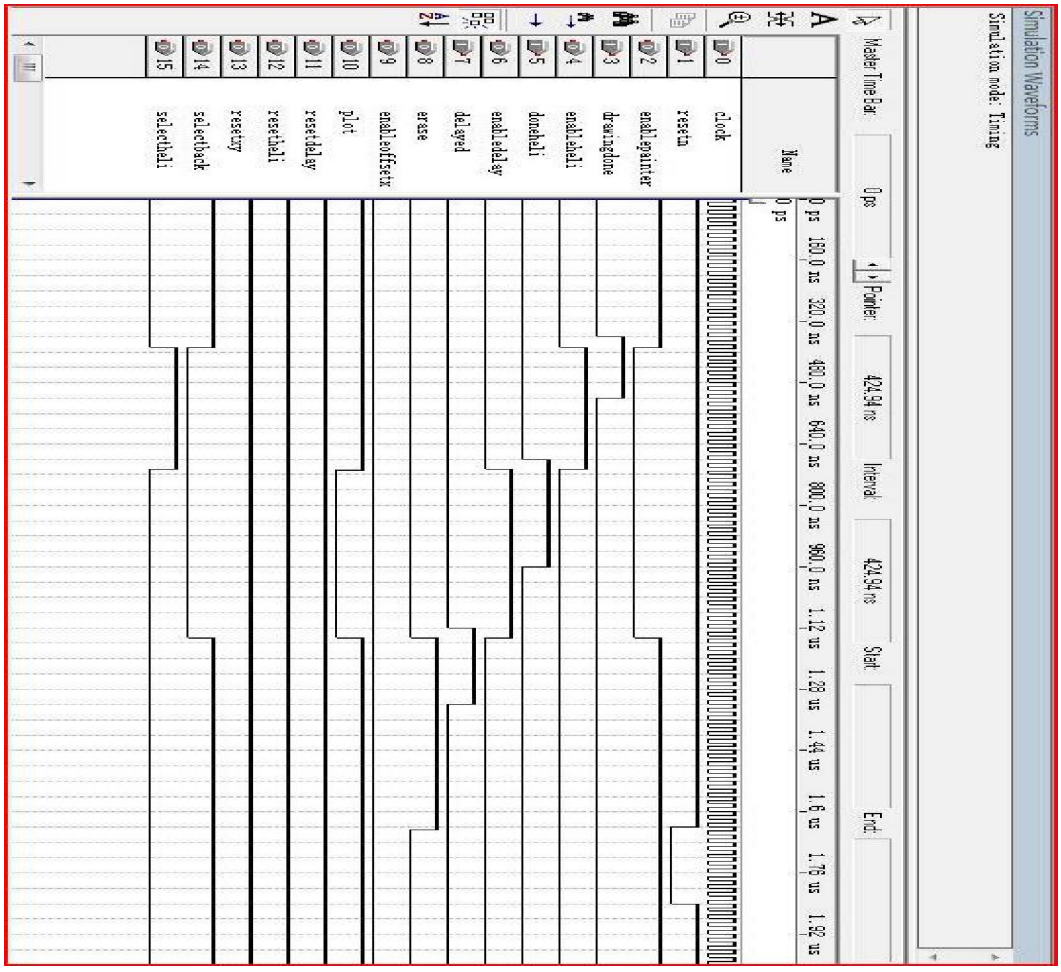

Fig.6 Wave simulation of Control Module (FSM)

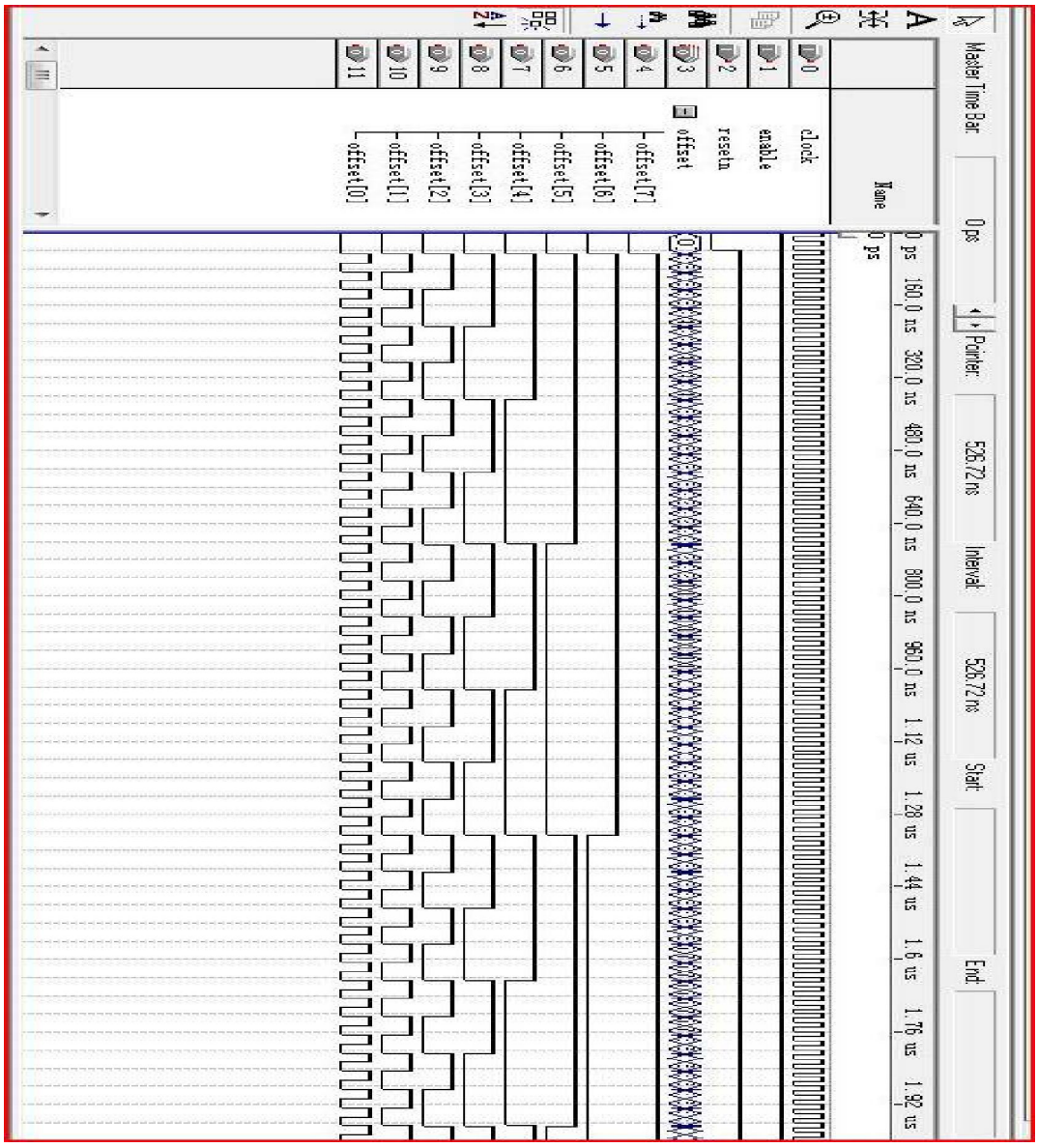

Fig.7 Wave simulation of Offset module