Pong Game

Assignment date: 04.07.2016
Submission date: 24.07.2016
Group members: Sebastian Wittka: 1/3, VGA Controller, Image Generator, Computer Opponent Felix Kaiser: 1/3, HDMI Controller, Match Controller Habib Gahbiche: 1/3, Introduction, Sound Generator

Contents

1 Introduction **1**

2 Implementation **1**
2.1 Video Controller 1
2.1.1 VGA Controller 2
2.1.2 HDMI Controller 3
2.1.3 HDMI - Principle of Operation 3
2.1.4 HDMI with Spartan6 FPGAs on Atlys Boards 4
2.2 Computer Opponent 5
2.3 Image Generator 5
2.3.1 Multiplexer 6
2.3.2 Wall 6
2.3.3 Ball 6
2.3.4 Paddle 7
2.3.5 Score 7
2.4 Match Controller 8
2.4.1 Finite State Machine - Interface 8
2.4.2 Finite State Machine - Blockdiagram 9
2.5 Sound Generator 9

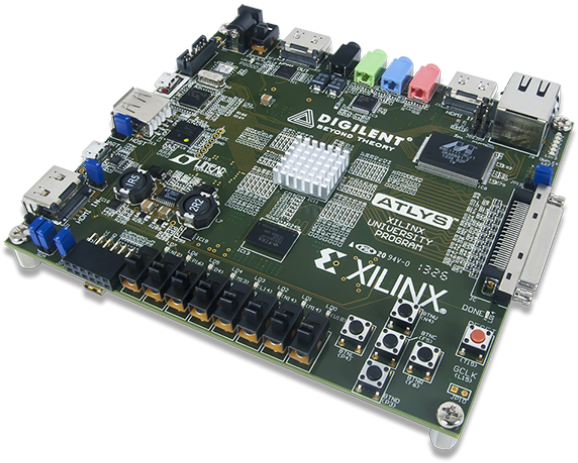
3 References **11**

1 Introduction

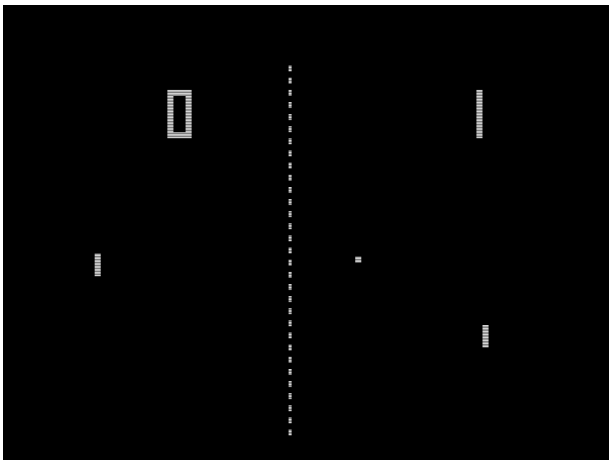
This project is about implementing the game Pong on the Atlys Spartan-6 FPGA board. Pong is a two dimensional multiplayer game that simulates table-tennis. Each of the two players controls an in game paddle by moving it vertically in order to hit a ball back and forth. A player scores a point when the opponent fails to return the ball.

We also took advantage of the built-in HDMI port and the AC-97 Codec to produce a better image and audio quality output.

Figure 1 shows a picture of the used board, and a screenshot of the (yet to be) realized game.



(a) Atlys Spartan-6 board



(b) Screenshot of the game Pong

Figure 1: Used board and screenshot of the game

2 Implementation

The hardware description of the implemented game pong is divided into six modules. Firstly the video controller, which implements the VGA respectively HDMI interface, provides the functionality to display the video output on a monitor. Two modules generate the inputs for the movement of the paddles. The right paddle is controlled by the debounced up and down buttons of the board and the right paddle is controlled by the computer opponent. Additionally the image generator creates all required objects like the ball and paddles as well as their movement. Furthermore the match controller manages the interactions between the objects and the state of the match itself. Lastly the audio output is created by the sound generator module.

2.1 Video Controller

The video controller provides an interface for the image generator so objects can easily be displayed on a monitor. Thereby it makes the coordinates of the current pixel available and hides all additional requirements of the protocol like timing. The point of origin of the pixel matrix is located at the top left corner of the monitor. Because our group has access to the

Atlys Spartan-6 board with HDMI connector as well as the Nexys4 board with VGA connector, we decided to implement a video controller for both protocols, so the remaining modules could be implemented using both boards. The main focus was laid on the above mentioned uniform interface for the image generator so the remaining development is independent of the used video controller.

2.1.1 VGA Controller

The Nexys4 board contains a VGA interface with 4-bits-per-color as well as the sync signals (horizontal sync - HS, vertical sync - VS). As shown in figure 2 the analog color signal for each of the 3 colors red, green and blue is generated by resistor-divider circuits. This results in 16 signal levels per color. Horizontal sync is used to get the timing for each row of the display and

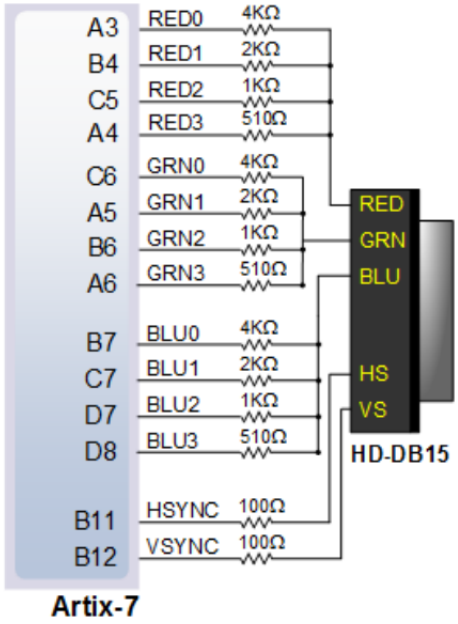
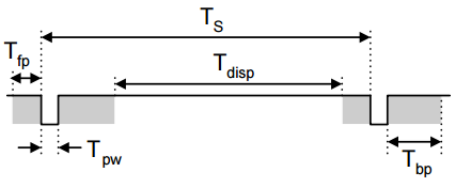


Figure 2: Nexys 4 VGA Interface

vertical sync for the timing for each frame. For a display resolution of 640x480 pixels a 25 MHz pixel clock is required for a vertical refresh rate of 60 Hz. Thereby the pixel clock is generated via a clock divider from the 100 MHz system clock. The required timings for both sync signals is shown in figure 3. Through sync pulse and pulse width the waveforms of the sync signals are specified. In addition display time as well as front and back porch are needed to identify the pixel coordinates on the display. Both porch timings are historically required, because CRT-based VGA displays used electron beams to display information on a phosphor-coated screen. Thereby the beam required these porch timings to get from the end of a row to the beginning of the next row for horizontal sync as well as from the end of a frame to the beginning of the next frame for vertical sync. Modern displays also use the same timings as CRT-displays for VGA.

The VGA controller module uses two counters to generate both of these sync signals (figure 4). In addition the display coordinates are provided for the image generator module to display all required objects.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 3: Signal timing for 640 x 480 pixels using a 25 MHz pixel clock at 60 Hz vertical refresh

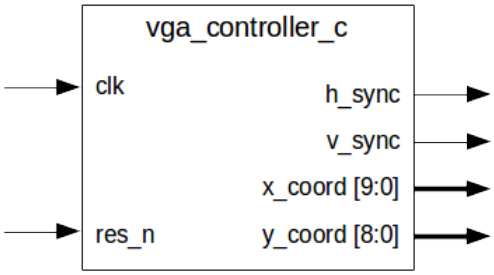


Figure 4: Schematic of the VGA Controller Module

2.1.2 HDMI Controller

For the first prototypes of the project VGA was used to stream the images to a screen. That needed to be changed because of two reasons: Firstly digital visual interfaces can get provide better resolution than analogue ones and secondly, and that is the actual main reason, the atlys board does not have a VGA port.

2.1.3 HDMI - Principle of Operation

HDMI(High Definition Multimedia Interface) is a digital interface which transport visual as well as audio data differential. The used encoding is the same as DVI called Transition-minimized differential signaling. TMDS is a kind of 8b/10b encoding but not the actual 8b/10b used by PCIE. It is optimized to create as less as possible transitions. Fast transitions are the main cause for electromagnetic interference which couples the twisted pair connection.
The TMDS encoder transforms the actual bits with the previous bits with a XOR or a XNOR function. It takes always the one which causes less transitions. This transformation happens for the 8 data bits while the ninth bit encodes which operation was used. Additionally the sustained average DC level should stay similar. Therefore the encoder can invert the data to even out the balance of ones and zeros. This is saved in the tenth bit.
Because of the fact that a HDMI connector just have 19 Pins, the data is transferred sequentially at a higher clockspeed. The first 12 Pins are for the data and the clock. For each color(RGB) and the clock there is a differential pair with one shield connection. That means that the bits have to be transferred with a 10 times faster clock than the clock which generates the pixel.

2.1.4 HDMI with Spartan6 FPGAs on Atlys Boards

The Atlys Board which is used in the project got 4 HDMI ports. The one which is called J2 is a usual Type A HDMI Connector. With this information it is possible to get the right constraints out of the following table(5).

HDMI Type A Connectors				HDMI Type D	
Pin/Signal	J1: IN	J2: Out	J3: IN	Pin/Signal	JA: BiDi
1: D2+	B12	B8	J16	1: HPD	JP3*
2: D2_S	GND	GND	GND	2: RES	VCCB2
3: D2-	A12	A8	J18	3: D2+	N5
4: D1+	B11	C7	L17	4: D2_S	GND
5: D1_S	GND	GND	GND	5: D2-	P6
6: D1-	A11	A7	L18	6: D1+	T4
7: D0+	G9	D8	K17	7: D1_S	GND
8: D0_S	GND	GND	GND	8: D1-	V4
9: D0-	F9	C8	K18	9: D0+	R3
10: Clk+	D11	B6	H17	10: D0_S	GND
11: Clk_S	GND	GND	GND	11: D0-	T3
12: Clk-	C11	A6	H18	12: Clk+	T9
13: CEC	NC	OK to Gnd	NC	13: Clk_S	GND
14: RES	NC	NC	NC	14: Clk-	V9
15: SCL	C13	D9	M16	15: CEC	VCCB2
16: SDA	A13	C9	M18	16: Gnd	GND
17: Gnd	GND	GND	GND	17: SCL	C13**
18: 5V	JP4*	5V	JP8*	18: SCA	A13**
19: HPD	1K to 5V	NC	1K to 5V	19: 5V	JP3

*jumper can disconnect Vdd **shared with J1 I2C signals via jumper JP2

EDK designs can use the xps_tft IP core (and its associated driver) to access the HDMI ports. The xps_tft core reads

Figure 5: Constraints for HDMI(out of the Atlys User Manual)

The actual design for the HDMI interface is shown in 6. This is a reference design by Xilinx itself. In the presented design the audio HDMI interface was not used, so ignore this part. Because of that there are just the video signals and the pixel clock there to mention. At first the RGB data is encoded by the TMDS module like it was described in the sections before. For a well synchronized transfer we need at the same time a perfectly timed pixel clock at a speed of 74.25 MHz. Because 74.25 MHz is hard to reach out of 100 MHz, 75 MHz were taken and it still worked fine. This clock was needed to be created by the clockwizard IP. In the next parts the TMDS encoded RGB data is converted and serialized into the differential TMDS signals. Because of the serialization we need a 10 times faster clock which is created out of the PLL shown in the schematic. Additionally to RGB data the clock is sent as well for synchronization.

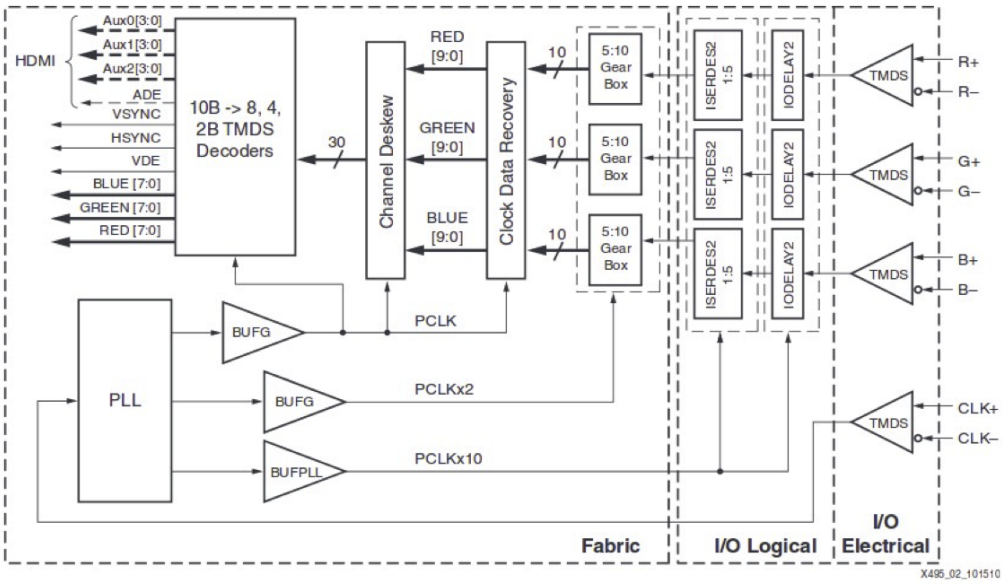


Figure 2: **TMDS Receiver Design**

Figure 6: Design how to build a HDMI interface(out of the TMDS User Interface Manual for Spartan6)

2.2 Computer Opponent

Only the right paddle of the pong game is controlled by the player. So the left paddle has to be controlled by some logic. For this purpose the computer opponent module generates the button signals to move the paddle up or down. It uses the vertical position of the paddle and the ball to determine the movement of the paddle (figure 7). A linear-feedback shift register (LFSR) is used to get some randomness into this control, so the computer opponent does not behave deterministically.

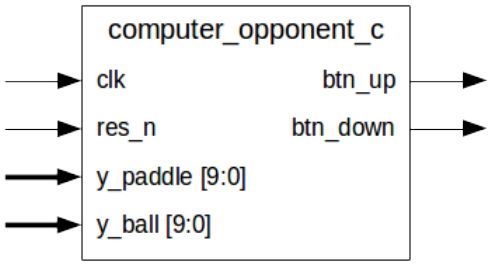


Figure 7: Schematic of the Computer Opponent Module

2.3 Image Generator

The Image Generator takes inputs from the players (`btn_up_left`, `btn_down_left`, `btn_up_right`, `btn_down_right`) and returns the RGB values that can be displayed through the HDMI or VGA interface of the boards (figure 8). The display coordinates are used in its subcomponents to show all objects at the right positions on the screen. Thereby each object is created by an individual module. Additionally four signals are received from the match controller module to ensure the right behaver of the game. The first two are `l_scored` and `r_scored`. These are

asserted if the left respectively right player obtains a point and the movement of the ball should be reset. The other signals are `l_paddle_hit` and `r_paddle_hit`, which are asserted if the ball hits the proper paddle. The match controller receives information about the vertical position of the paddles and the position of the ball from the image generator module to generate these four signals.
All vectors in the schematics of the image generator and its subcomponents are matching for the resolution of the HDMI controller with 1280x720 pixels using 24-bits-colors. The schematics for the VGA resolution are not shown separately, bacause their vectors are only slightly smaller and do not change the overall structure of the modules.

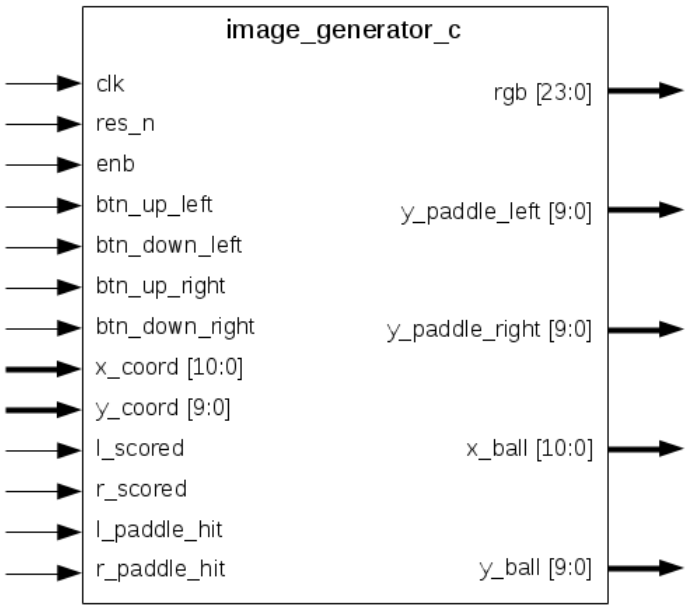


Figure 8: Schematic of the Image Generator Module

2.3.1 Multiplexer

Because the RGB values are generated for each object individual, they have to be combined into a single output to drive the VGA connector. For this purpose a multiplexer is used (figure 9). It forwards the RGB value of the object, which asserts its select signal. If no select is asserted, the background color is displayed. Multiple asserted select signals are resolved by using different priorities for the objects.

2.3.2 Wall

To restrict the movement of the ball and the paddles the wall builds the surrounding of the field. On the basis of the display coordinates this module asserts its select signal, if the coordinates are inside of the area of the wall (figure 10).

2.3.3 Ball

The ball entity moving across the field is created in the `ball_c` module (figure 11). It moves diagonally over the field and rebound from the walls. It also rebound from a paddle, of it hits it. To indicate this, the match controller asserts `l_paddle_hit` or `r_paddle_hit`. Like the wall module, the ball module asserts its select signal, if the display coordinates are inside of the

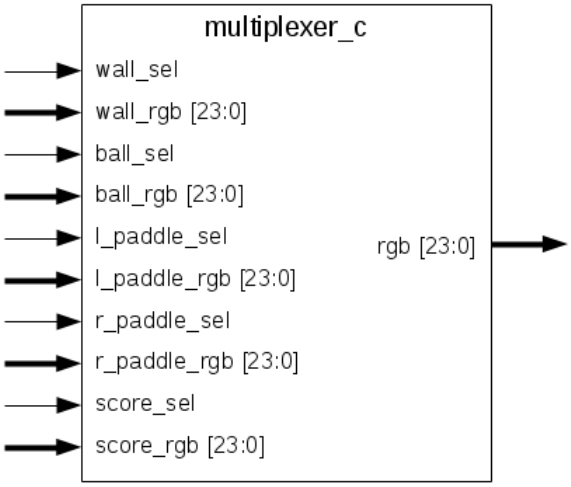


Figure 9: Schematic of the Multiplexer Module

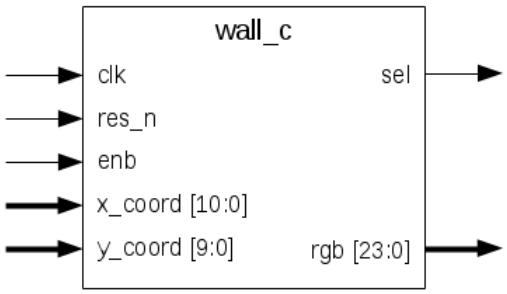


Figure 10: Schematic of the Wall Module

area of the ball. Additionally the coordinates of the ball (top left corner) are provided to the match controller and computer opponent modules. The right movement speed is generated by dividing the system clock down to the desired movement frequency of the ball. It then moves one pixel in x- and y-direction per movement clock cycle.

2.3.4 Paddle

For each player the paddle module is instantiated. It can be moved up and down via the up- and down-buttons of the board for the right player. The values for `btn_up` and `btn_down` for the left player are generated by the computer opponent module. Again the display coordinates are used to assert the select signal, if the coordinates are inside of the area of the paddle (figure 12). Furthermore the y-position is provided to the match controller to ensure the right behaver of the match. Additionally the computer opponent receives this signal from the left paddle to calculate its movement.

2.3.5 Score

A score is used to count the points each player achieves. Thereby a digit is displayed like a seven-segment display and each player can score up to 99 points (figure 13). Again the display coordinates are used to assert the select signal, if the coordinates are inside of the area of the score. Internally the score for each player is stored as a bit-vector. So it is firstly converted into BCD code and then this BCD code is used in an decoder to drive the digits of the seven-segment display.

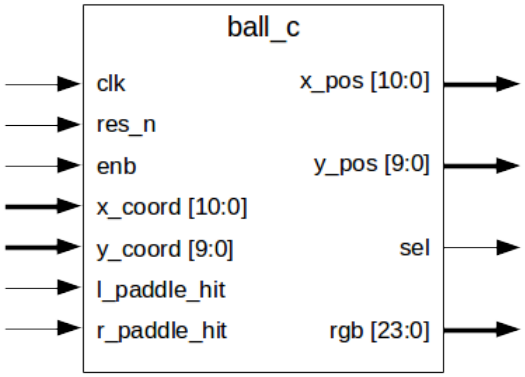


Figure 11: Schematic of the Ball Module

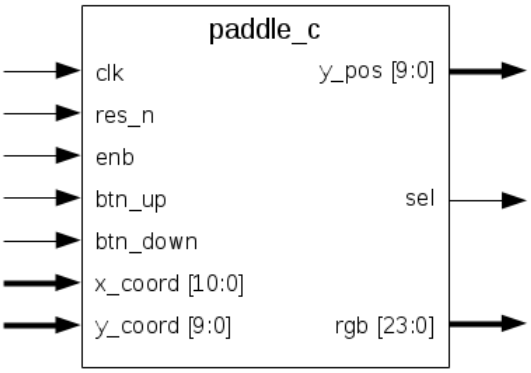


Figure 12: Schematic of the Paddle Module

2.4 Match Controller

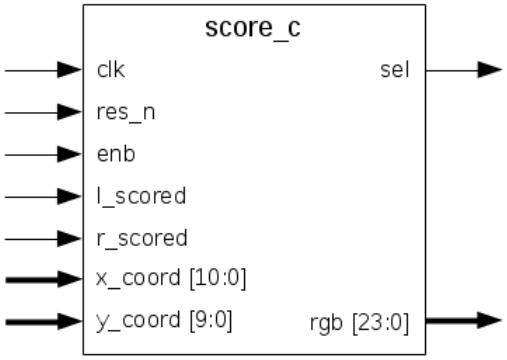
The game has to have structered rules to decide who of the players has won a game. For example it is obligatory to know exactly when a score is counted and how the movement changes when the ball hits a wall or one of the players panels. Furthermore it has to be clear how the game continues after scoring a point.
For the last point there are some different possbilities. The ball could either go "through" the goal and appear out of the other one, it could rebound or it respawns at another point near the center of the field. In the presented design the last option was chosen. This happened for no special reason but it seemed to be the most balanced possibility. The ball is spawning at the exact center of the x-axis. The y-coordinates are variating to avoid determinism during the game.

2.4.1 Finite State Machine - Interface

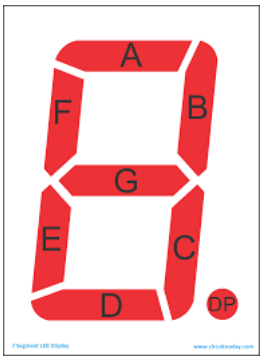
The interface of the FSM as it is shown in 14 consists of 4 input and 4 output signals.

Firstly there are two input signals `y_paddle_left` and `y_paddle_right`. These are the vertical paddle positions. The other two input signals are the x and the y position of the ball. Via these values it is possible to detect a collision between the ball and one of the paddles.

The output signals which are generated in this module are a `paddle_hit` and a scored signal for the left and the right side. In general this module has to detect if one of the paddles is hit or a point is scored. When one of these events happen it have to provide a signal to the top



(a) Schematic of the Score Module



(b) Seven-Segment Display

Figure 13: Schematic of the Score Module and Seven-Segment Display

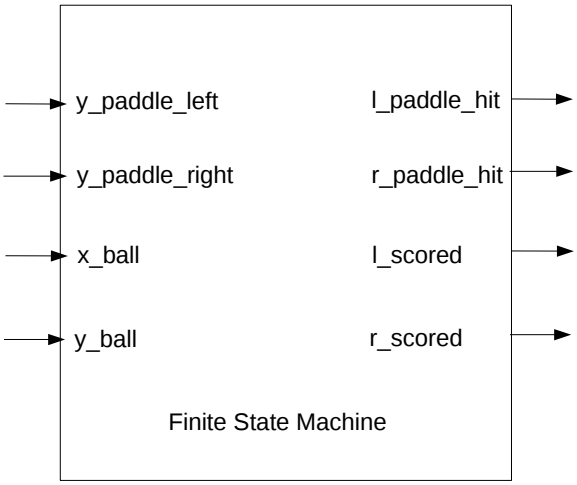


Figure 14: Interface of the FSM module

module.

2.4.2 Finite State Machine - Blockdiagram

The ball can actually be in one of two non-volatile states: Either it is moving left- or rightwards15. But before a game can start the ball needs to be spawn. Therefore is the startng state the so called `spawn_phase`. As soon as the ball either gets close enough to one of the paddles or to one of the sidewalls, a transition happens. If it is close enough to one of the paddles the corresponding `paddle_hit` signal is driven. In the upper module the moving direction of the ball is changed. On the other hand when one of the side walls is hit by the ball a score has to be counted. For this issue the design gets into the `goal_hit` state. Now the corresponding scored signal is driven. Furthermore the ball has to be spawned again. Because of that there is a transition back to the `spawn_phase`. The current moving direction is kept.

2.5 Sound Generator

In order to generate sound, we used the on board LM4550 chip. Figure 16c shows the block diagram of the sound generator. This module takes in the sound events from the Match

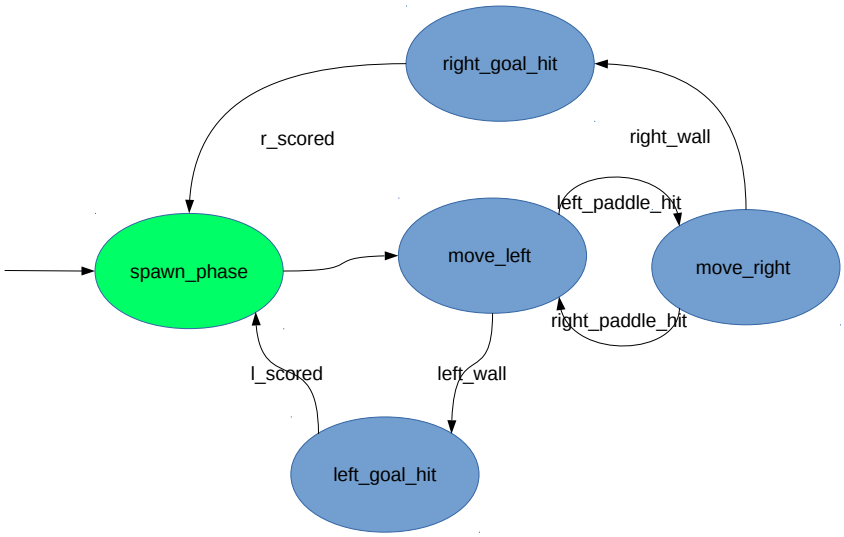


Figure 15: Diagram of the FSM

Controller module (i.e when sound generation should take place), reads a sound effect from a ROM and sends it to the LM4550 chip that in turn outputs the sound effect to a connected speaker.

The inputs to the `snd_gen_c` include a `clk`, an active low reset, a serial data in line `sdata_in`, a 12.288 MHz bit clock from the AC97 chip, 3 bit `snd_eff` signal and 5 bit `volume` control (will be connected to the switches of the Atlys board).

The module's output include a `sync` signal, serial data output `sdata_out` and an AC97 active low reset that initializes the AC97 (LM4550) chip.

Internally, the `snd_gen_c` module contains an AC97 controller and an AC97CMD submodules.

- **AC97 controller:** implements the AC Link serial interface protocol. Figure 16a shows an AC bidirectional audio frame, whereas figure 16b shows an AC output audio frame. In this project, we will be using the LM4550 chip for output only, however, the input audio frame (not shown here) has also been implemented for testing reasons. The next paragraph is a brief description of the AC link interface protocol. For more details about the AC97 link serial interface protocol, see `http://www.ti.com/lit/ds/symlink/lm4550.pdf`

  The AC Link Output Frame carries control and PCM data to the LM4550 control registers and stereo DAC. Output Frames are carried on the `sdata_out` signal which is an output from the AC97 Controller and an input to the LM4550 codec. As shown in Figure 16b, Output Frames are constructed from thirteen time slots: one Tag Slot followed by twelve Data Slots. Each Frame consists of 256 bits with each of the twelve Data Slots containing 20 bits. Input and Output Frames are aligned to the same SYNC transition.

- **AC97CMD command state machine:** is a state machine that creates the control data based on the inputs `snd_eff` and `volume` and passes it to the AC97 Controller. This control data controls among others the volume of the output audio and the DAC sample rate. Figure 16c shows the path of the audio data that gets transferred within the LM4550 chip and the registers that need to be configured along this path. The following code was extracted from the file `a97cmd_fsm.vhd` that implements the state machine necessary to

configure the registers of the LM4550 chip. It shows an example of how to activate the output `HP_OUT`. As shown in figure 16c `LINE_OUT` is muted by default.

...

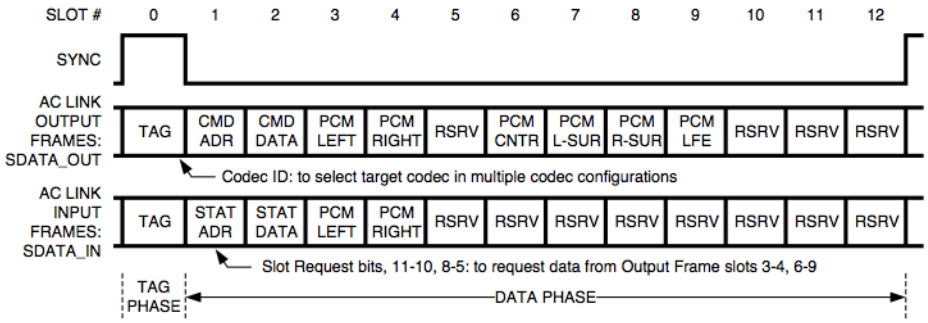
```
process (next_state, cur_state, atten)
    begin
        case cur_state is
            when S0 =>
            cmd <= X"02_8000";  -- master volume     0 0000->0dB atten,
                                                    1 1111->46.5dB atten
                next\_state <= S2;
            when S1 =>
                                                    -- HP_OUT volume
                cmd <= X"04" & "000" & atten & "000" & atten;
                next\_state <= S4;
            when S2 =>
```

...

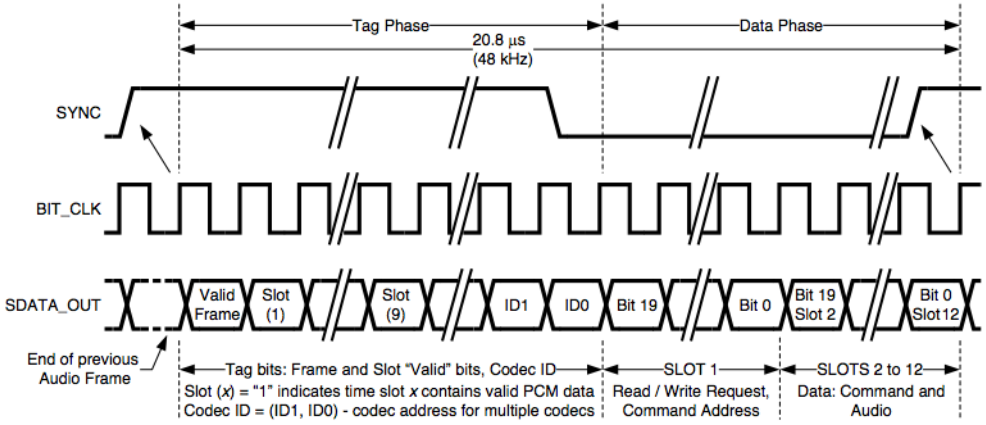
The AC97CMD module also reads the audio sound effects from a ROM that gets initialized when the FPGA is programmed. The right sound effect is chosen according to the signal `snd_eff`.

3 References

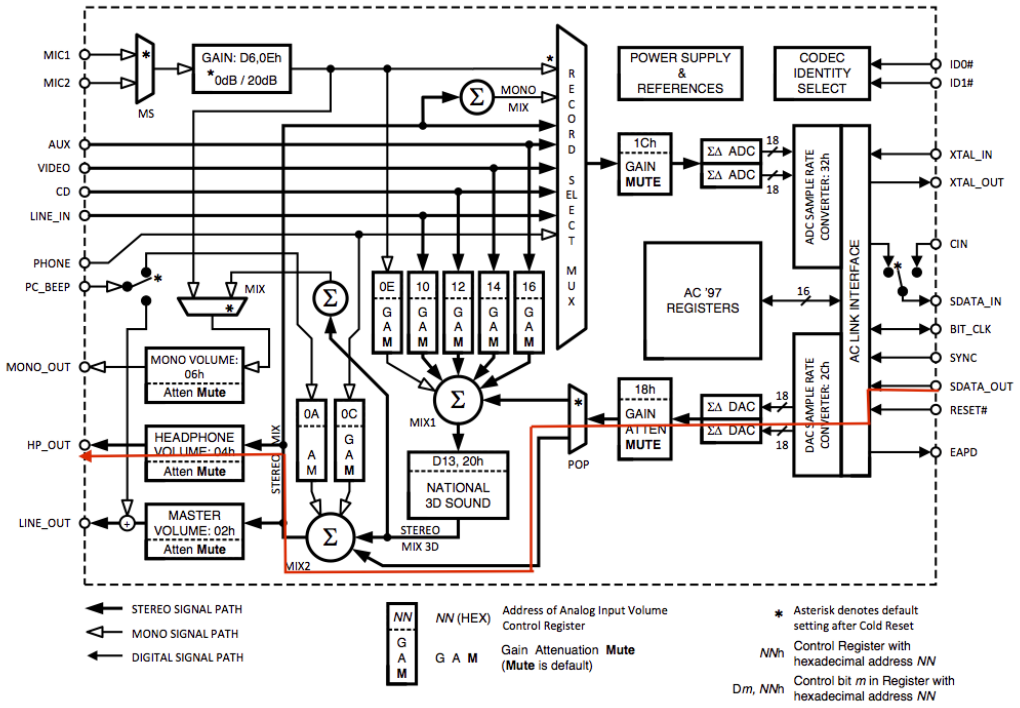
- The sound generator module is based on the sample project provided by Xilinx. `http://www.xilinx.com/support/university/boards-portfolio/xup-boards/AtlysBoard.html#docs`
- LM4550 `http://www.ti.com/lit/ds/symlink/lm4550.pdf`
- HDMI reference `https://reference.digilentinc.com/_media/atlys:atlys:atlys_rm.pdf`
- Nexys4 FPGA Board Reference Manual `http://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys4/documenatation/Nexys4_RM_VB1_Final_3.pdf`



(a) AC link bidirectional frame



(b) AC link output audio frame



(c) Block diagram of LM4550 chip. Red line is the path from FPGA output to `HP_OUT`. All registers along this path need to be configured.

Figure 16: AC link serial interface protocol