

Pong Game

Assignment date: 04.07.2016

Submission date: 24.07.2016

Group members: Sebastian Wittka, Felix Kaiser and Habib Gahbiche.

Contents

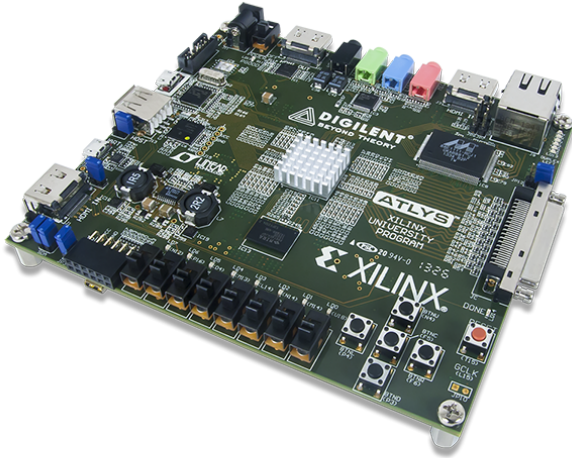
1	Introduction	1
2	Implementation	1
2.1	Video Controller	1
2.1.1	VGA Controller	2
2.1.2	HDMI Controller	3
2.2	Computer Opponent	3
2.3	Image Generator	4
2.3.1	Multiplexer	4
2.3.2	Wall	4
2.3.3	Ball	5
2.3.4	Paddle	5
2.3.5	Score	6
2.4	Match Controller	6
2.5	Sound Generator	7
3	Summary	8
4	references	9

1 Introduction

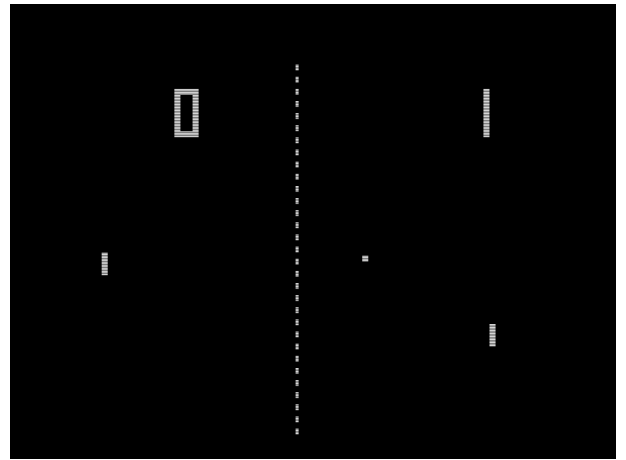
This project is about implementing the game Pong on the Atlys Spartan-6 FPGA board. Pong is a two dimensional multiplayer game that simulates table-tennis. Each of the two players controls an in game paddle by moving it vertically in order to hit a ball back and forth. A player scores a point when the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

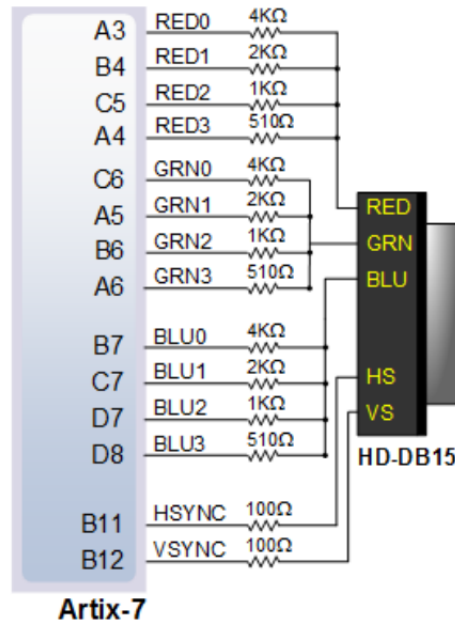


Figure 2: Nexys 4 VGA Interface

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.

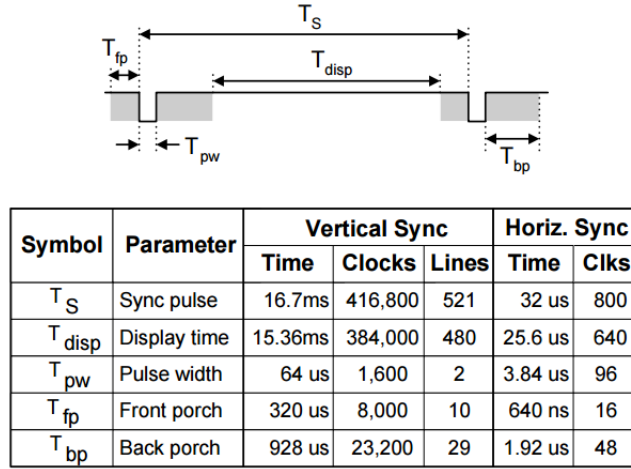


Figure 3: Signal timing for 640 x 480 pixels using a 25 MHz pixel clock at 60 Hz vertical refresh

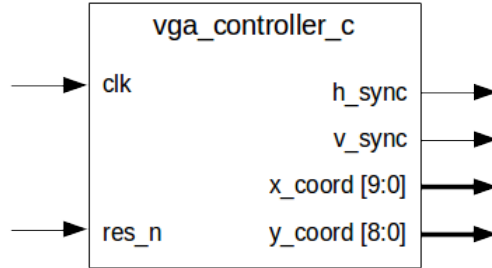


Figure 4: Schematic of the VGA Controller Module

2.1.2 HDMI Controller

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 5). A linear-feedback shift register (LFSR) is used to get some randomness into this control, so the computer opponent does not behave deterministically.

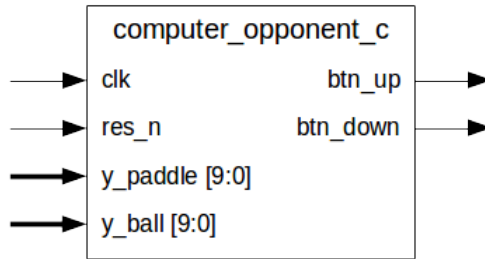


Figure 5: Schematic of the Computer Opponent Module

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 6). 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 behavior 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, because their vectors are only slightly smaller and do not change the overall structure of the modules.

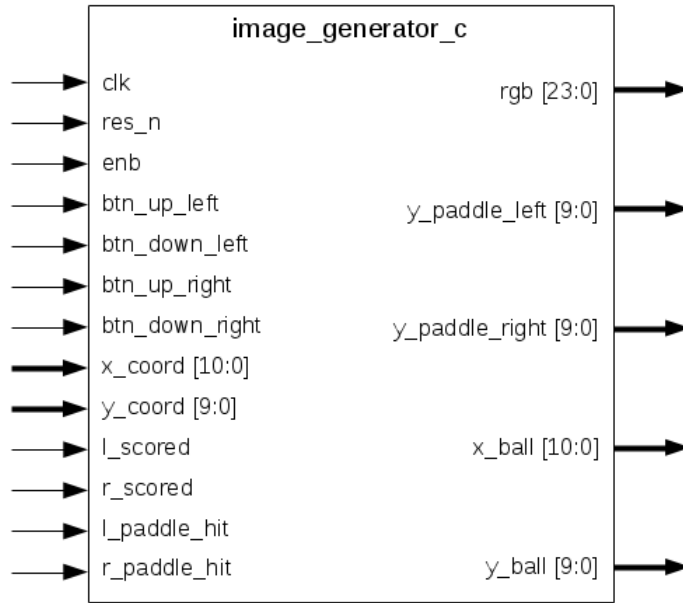


Figure 6: Schematic of the Image Generator Module

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 7). 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

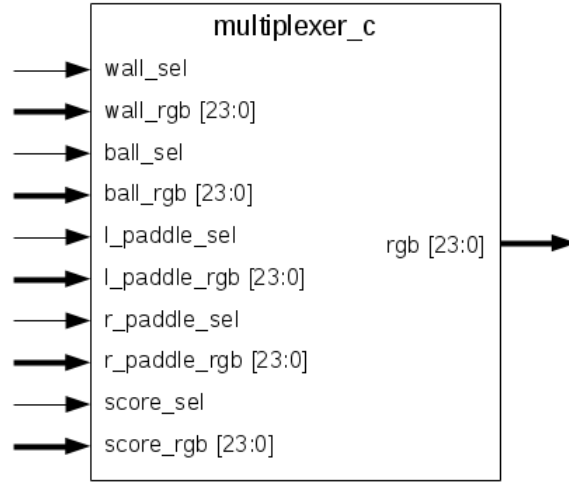


Figure 7: Schematic of the Multiplexer Module

are inside of the area of the wall (figure 8).

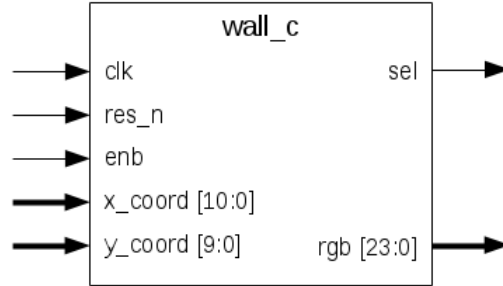


Figure 8: Schematic of the Wall Module

2.3.3 Ball

The ball entity moving across the field is created in the `ball_c` module (figure 9). It moves diagonally over the field and rebound from the walls. It also rebound from a paddle, if 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 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 10). Furthermore the y-position is provided to the match controller to ensure the right behavior of the match. Additionally the computer opponent receives this signal from the left paddle to calculate its movement.

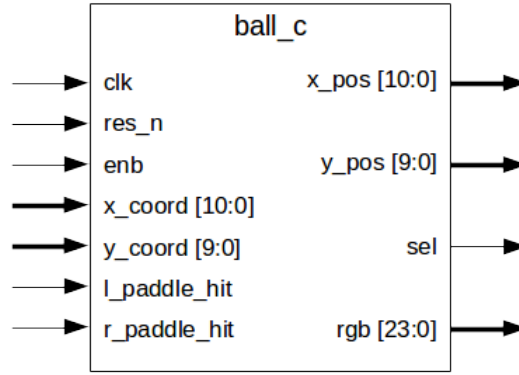


Figure 9: Schematic of the Ball Module

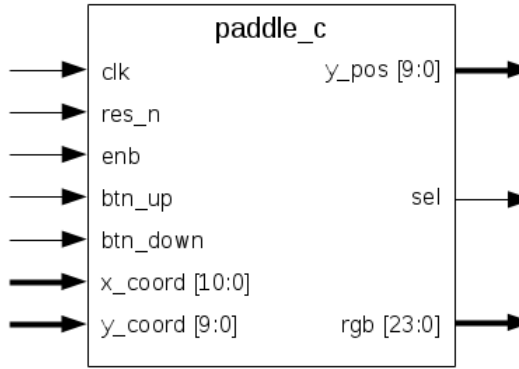
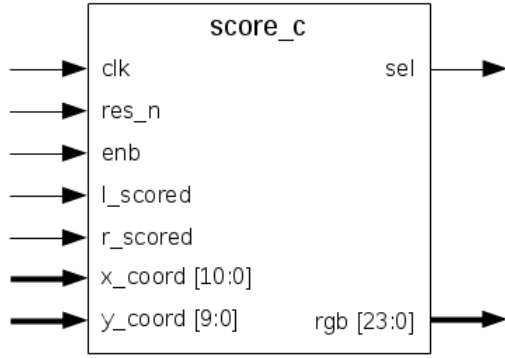


Figure 10: Schematic of the Paddle Module

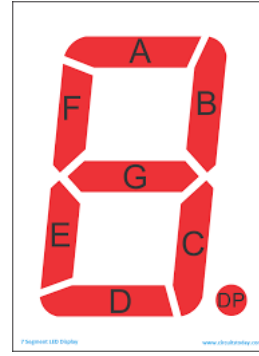
2.3.5 Score

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 11). 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.

2.4 Match Controller



(a) Schematic of the Score Module



(b) Seven-Segment Display

Figure 11: Schematic of the Score Module and Seven-Segment Display

2.5 Sound Generator

In order to generate sound, we used the on board LM4550 chip. Figure ?? shows the block diagram of the sound generator. This module takes in the sound events from the Match 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 12a shows an AC bidirectional audio frame, whereas figure 12b 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 12b, 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 12c 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 12c `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 Summary

4 References

- The VHDL implementation of the AC97 controller was based on the sample project provided by Xilinx. See <http://www.xilinx.com/support/university/boards-portfolio/xup-boards/AtlysBoard.html#docs>
- LM4550 manual <http://www.ti.com/lit/ds/symlink/lm4550.pdf>
- Nexys4 FPGA Board Reference Manual http://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys4/documenatation/Nexys4_RM_VB1_Final_3.pdf

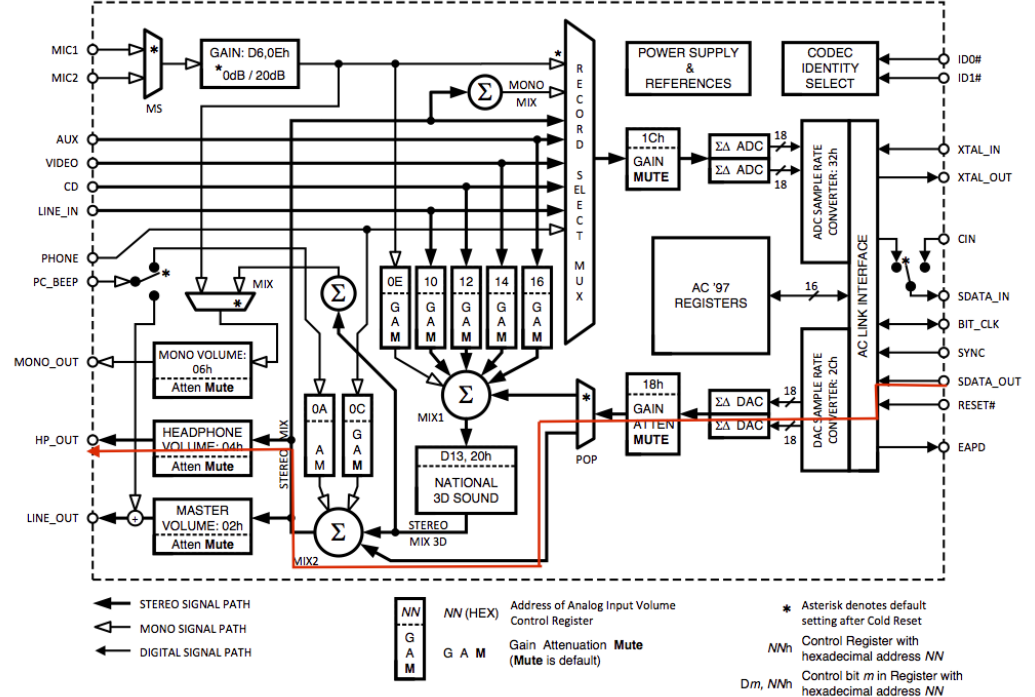
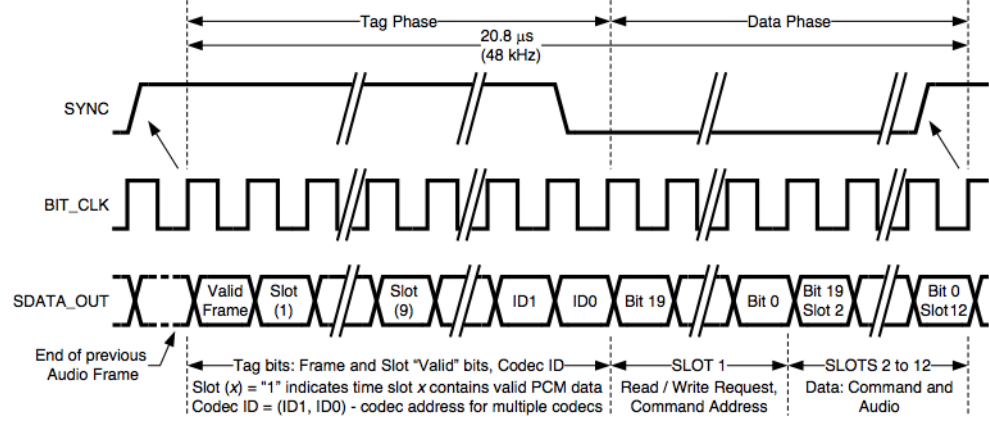
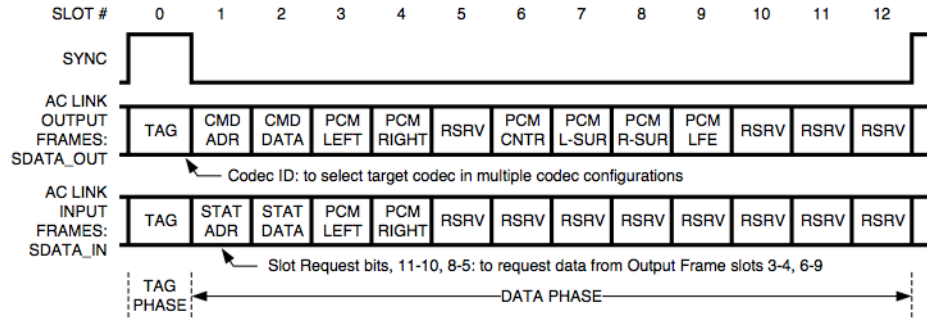


Figure 12: AC link serial interface protocol