Alarm System

Niculescu Oana

1. Introduction

This project is an alarm system made on the Altera DE1 Development board using Verilog HDL and the Quartus II software. It uses an ultrasonic HC-SR04 sensor to detect the distance at which an object is placed from it. In the event that this object is less than 100 centimeters away, an alarm is triggered, which will play through the speakers connected to the Line Out of the DE1 board. There are only two ways the user can interact with the project: placing an object in front of the sensor to see the distance that is picked up and, thus, trigger the alarm, or resetting the system.

2. Top Module

The Top module is only used to create instances of the submodules that the project consists of.

2.1. Inputs

CLK – The board’s 50 MHz oscillator clock, will be used for clock dividers throughout the project

RST – The system reset mapped to one of the keys on the board (KEY0)

Echo_Sig – The signal the HC-SR04 sensor sends back to the FPGA, based on which we can measure the distance to the object placed in front of the sensor

2.2. Outputs

XCLK – The chip clock used by the Audio Decoder on the DE1 Board (Wolfson WM8731)

BCLK – The Bit-Stream clock used by the Audio Decoder

DACLRCK – The alignment clock used by the Audio Decoder for the left and right channels

Trigger_Sig – The trigger signal sent to the HC-SR04 sensor to begin measurement

Distance – The distance the object is placed from the sensor, to be shown on the 7-segment digit display

Sound_Data – The beep sent to the Audio Codec to be outputted through Line Out

2.3. Bidirectional

I2C_Data – The data signal that selects the Audio Codec as the current chip the FPGA interacts with and provides the write operation, while also receiving an acknowledge bit from the codec

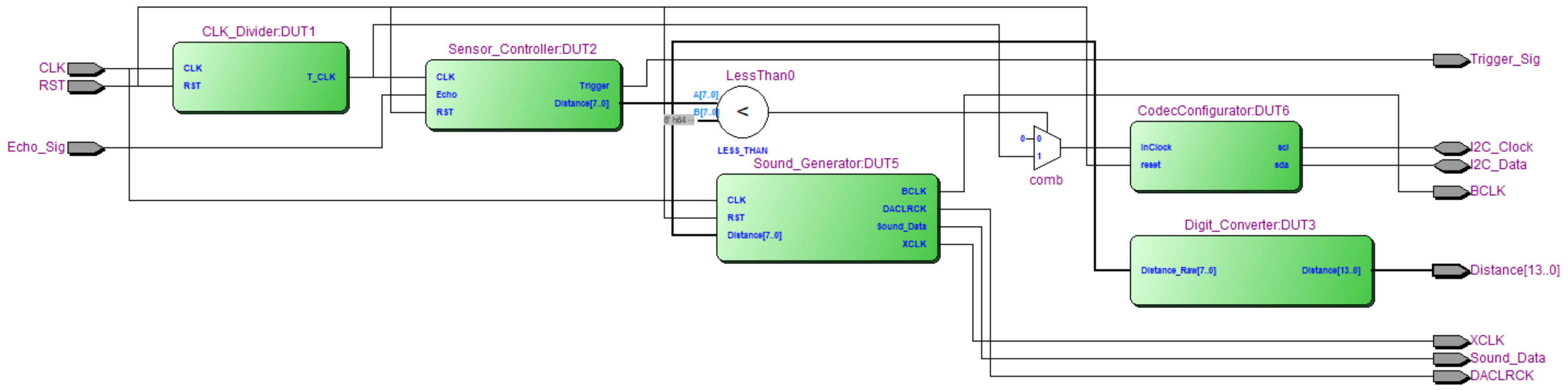
I2C_Clock – The clock signal sent to the Audio Codec for control

2.4. Internal wires

T_CLK – The trigger clock used in generating the trigger signal sent to the HC-SR04 sensor

Distance_Raw – The distance the object is placed from the sensor, to be transformed into the corresponding signals for the 7-segment displays

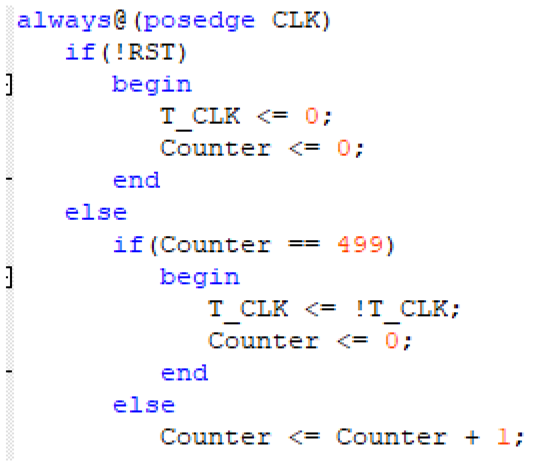
3. Block Diagram



4. Submodules

4.1. Clock Divider

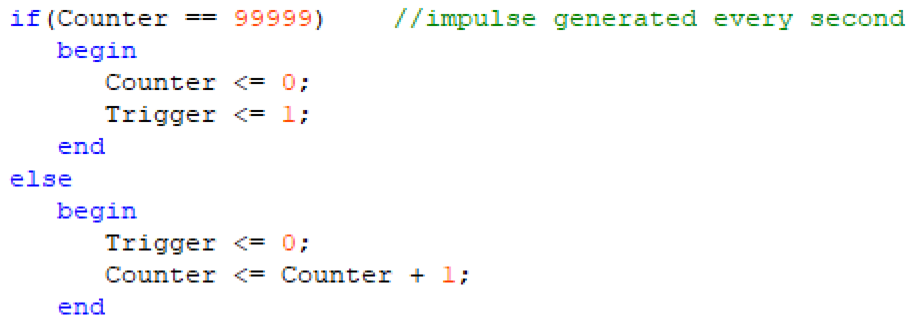
The **CLK_Divider** module generates the trigger clock, which is 100 KHz. To do this, a counter is used to invert a signal every 500 cycles of the main clock, **CLK**.



```
always@(posedge CLK)
   if(!RST)
      begin
         T_CLK <= 0;
         Counter <= 0;
      end
   else
      if(Counter == 499)
         begin
            T_CLK <= !T_CLK;
            Counter <= 0;
         end
      else
         Counter <= Counter + 1;
```

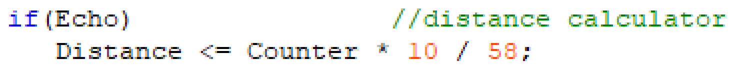
4.2. Sensor Controller

The **Sensor_Controller** module uses the previously generated trigger clock to send a trigger signal to the sensor. Since the sensor only needs one 10 µS impulse to begin measuring, a single cycle of the trigger clock is enough. For the purpose of this project, this impulse is generated once every second (the sensor only measures once every second).



```
if(Counter == 99999)     //impulse generated every second
   begin
      Counter <= 0;
      Trigger <= 1;
   end
else
   begin
      Trigger <= 0;
      Counter <= Counter + 1;
   end
```

Furthermore, a counter is used to measure the number of cycles between when the trigger signal was sent and when the echo was received. The following formula is used from the HC-SR04 datasheet to calculate the distance from the object:



```
if(Echo)                 //distance calculator
   Distance <= Counter * 10 / 58;
```

Since the counter only goes up every cycle of the trigger clock, each cycle counted is 10 µS.

4.3. Digit Converter

The **Digit_Converter** submodule transforms the distance calculated by the sensor controller into 7-segment display signals. Since our target range varies between 0-99 centimeters, two digits are used to display the value. First, the two digits are extracted:

```
assign Distance_Units = (Distance_Raw < 100) ? (Distance_Raw % 10) : 9;
assign Distance_Tens = (Distance_Raw < 100) ? ((Distance_Raw - Distance_Units) / 10) : 9;
```

Then, the digits are transformed into their respective signals using two instances of the **BCD** module.

```
BCD DUT1(.BINARY(Distance_Tens),
         .SEGMENTS(Distance[13:7]));

BCD DUT2(.BINARY(Distance_Units),
         .SEGMENTS(Distance[6:0]));
```

4.4. Sound Generator

The **Sound_Generator** module uses the main clock to send the necessary synchronization clocks and the sound data to the Audio Codec. It is essentially a clock divider module, but using a different method than the one used by the **CLK_Divider**. A 16-bit counter is used so that a sound signal of ~6KHz is generated.

```
always@(posedge CLK)
   if(!RST)
      Counter <= 0;
   else
      Counter <= Counter + 1;

assign Sound_Data = (Distance < 100) ? Counter[12] : 1'bz;     //frequency is about 6khz
```

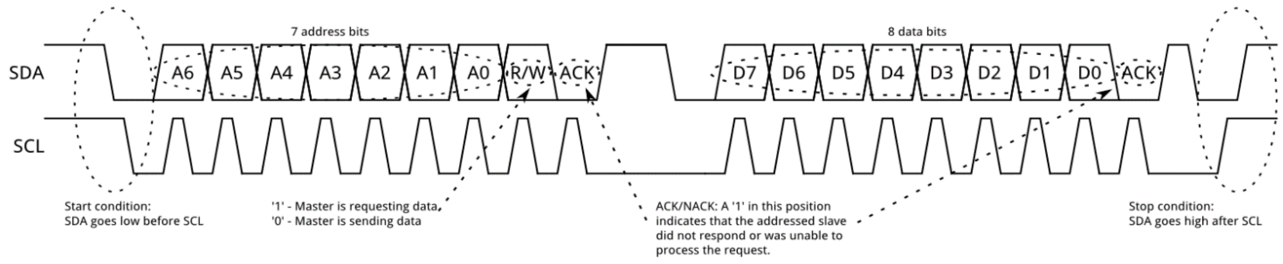
The synchronization clocks are different fractions of the main clock, needed by the codec to output the sound to the headphones.

```
assign XCLK = Counter[1];
assign BCLK = Counter[3];
assign DACLRCK = Counter[8];
```

4.5. Codec Configurator

The **CodecConfigurator** module is a third-party module [1] that acts as an interface for the I2C protocol used to control the Audio Codec. It is an automata that cycles through the stages needed to form a connection between the FPGA and the Wolfson Codec. The start and stop conditions are controlled, the device is selected, the specific operation data is sent (e.g. activation, powering down, left line in, right line in) and the final I2C clock is generated. These are connected to the bidirectional I2C buses.

The following figure [2] shows the stages the I2C automata goes through to ensure a connection to the Wolfson chip:


7 address bits
8 data bits
SDA
SCL
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
Start condition: SDA goes low before SCL
'1' - Master is requesting data, '0' - Master is sending data
ACK/NACK: A '1' in this position indicates that the addressed slave did not respond or was unable to process the request.
Stop condition: SDA goes high after SCL

In this situation, the master will always be sending data, since the microphone Line In is not used.

5. Observations and difficulties

Previous difficulties in the realisation of this project were the correct testing of the echo signal every cycle of the trigger clock within the **Sensor_Controller** module. As it stands, the measurement is not precisely accurate, but for the purpose of this project, it is sufficient to creating an alarm system.

A second problem was correctly generating the sound signal and the clocks used by the codec for synchronization, for which Albiach, J.'s thesis [3] on interfacing provided a solution. Thus, I was able to use a basic clock diving circuit to now correctly output a sound though the Line Out port to the headphones.

At last, the current state of the project does not allow changing the volume of the outputted sound, needing to alter the structure of the I2C automata to permit this operation to be done using the buttons on the DE1 board.

6. References

[1] - https://github.com/Goshik92/FFTVisualizer

[2] - https://learn.sparkfun.com/tutorials/i2c

[3] - Albiach, J. I. M., 2006. **Interfacing a processor core in FPGA to an audio system**. Linköping Institute of Technology.