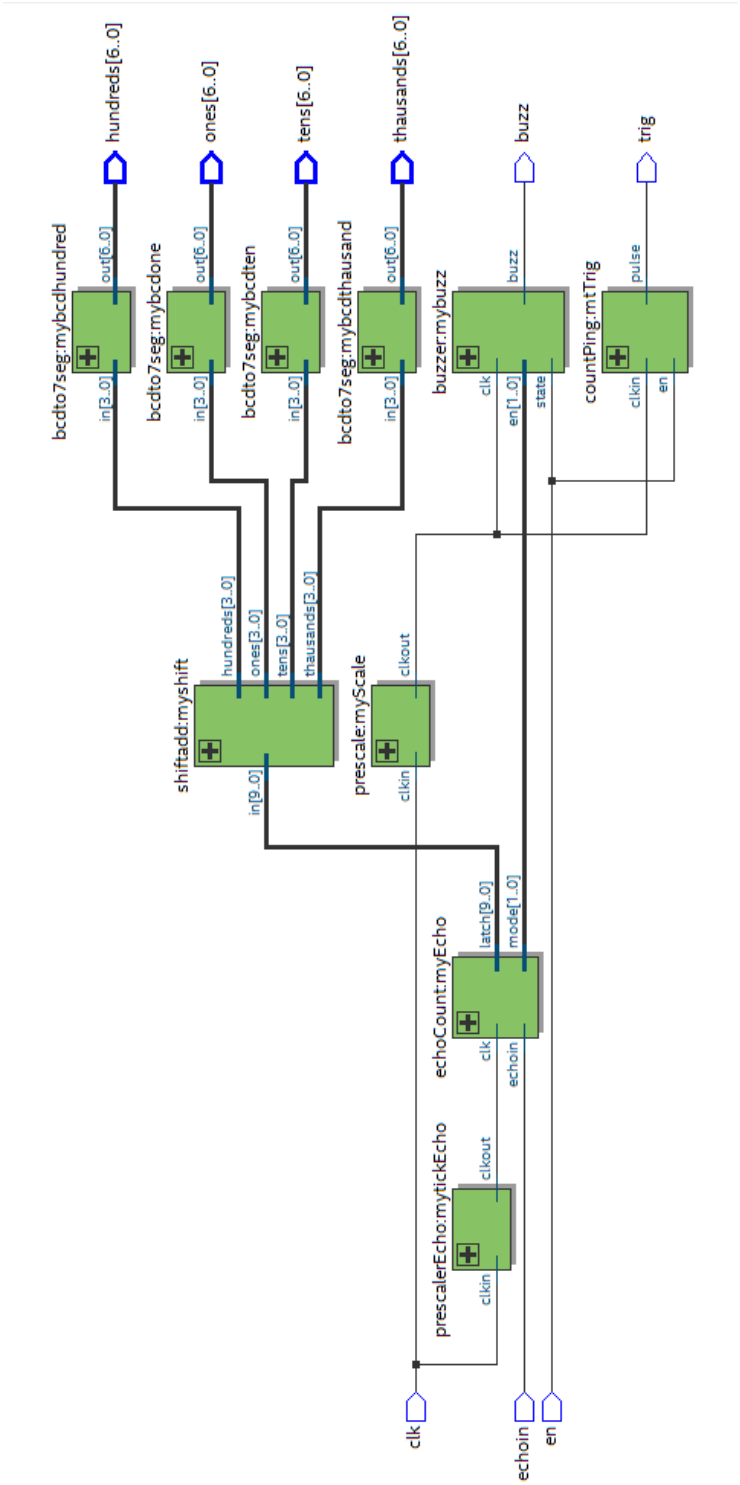
Muhammad Agung Nursyeha
07111850040005

Pengukuran Jarak Menggunakan Sensor Ping SR04 dan FPGA

1. Diagram Blok Sistem

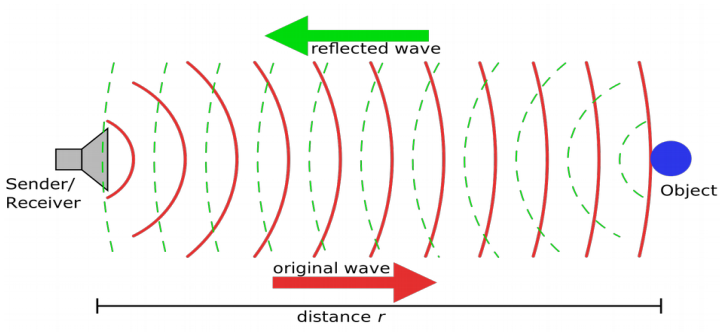


2. Cara kerja

Pada diagram blok, input sistem memiliki 3 pin yaitu “clk” untuk *clock*, “echoin” untuk input sinyal *echo* dari sensor jarak ultrasonik dan “en” untuk *enable* sistem. Output sistem berisi “buzz” yang akan tersambung pada *buzzer* aktif, “trigger” digunakan memberi *pulse* pada sensor ultrasonik, dan output untuk 4 *seven segment* yang menampilkan jarak yang terukur oleh sensor ultrasonik dalam 4 digit desimal.

Input *clock* dari FPGA memiliki frekuensi 50MHz, sedangkan untuk membangkitkan *pulse* trigger dibutuhkan lebar pulsa t_{on}= 10us, sehingga pulsa dengan frekuensi 50MHz diturunkan terlebih dahulu dengan prescaler 500 kali agar menjadi 100kHz.

Ketika “en” memiliki logika 1, maka pulsa *trigger* dengan t_{on}=10us akan dibangkitkan setiap 10ms. Setelah mengirim pulsa *trigger*, logika pada pin “echoin” adalah 1, dan counter untuk lebar pulsa *echo* mulai menghitung. Frekuensi counter untuk menghitung lebar *echo* disesuaikan dengan waktu tempuh suara dalam 1 mm.



Gambar 1. ilustrasi jarak tempuh gelombang mekanik pada sensor jarak SR04.

$$r = \frac{t}{2} * v$$

$$r = \frac{t}{2} * 340\, m/s$$

$$r = t * 170\, m/s$$

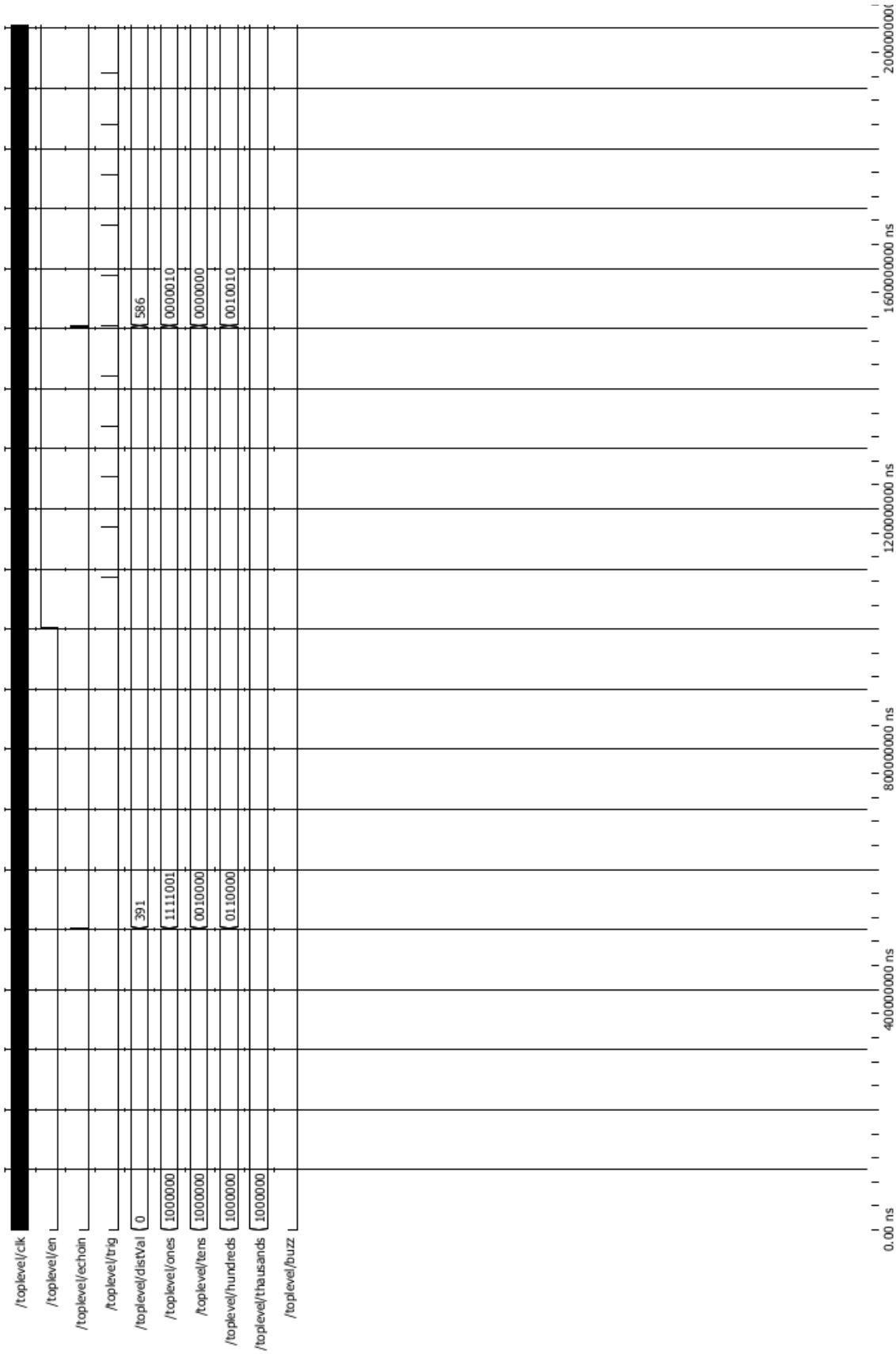
$$r = t * 0.17\, mm/us$$

$$t \simeq 5\, us/mm \qquad (1)$$

dari persamaan 1 diperoleh waktu tempuh suara dalam 1 mm adalah 5us, sehingga clock untuk menghitung lebar pulsa echo yang diperlukan memiliki periode yang sama yang menghasilkan frekuensi untuk menghitung lebar pulsa *echo* adalah 200kHz. Untuk mendapatkan *clock* dengan frekuensi 100kHz dari sumber 50MHz, maka diperlukan prescaler sebesar 250 kali. Hasil dari perhitungan lebar pulsa *echo* pada pin “echoin” merupakan jarak dalam milimeter. Untuk menampilkan lebar pulsa *echo* pada seven segment 4 bit, dibutuhkan rangkaian *shift and add 3* dan *binary* to *seven segment decoder.*

Ketika jarak diatas 280mm, maka sistem tidak membunyikan *buzzer.* Jika jarak diantara 251mm dan 280mm, maka buzzer akan berbunyi dengan frekuensi rendah. Frekuensi buzzer akan semakin tinggi apabila nilai jarak semakin kecil.

3. Simulasi


/toplevel/clk
/toplevel/en
/toplevel/echoin
/toplevel/trig
/toplevel/distVal
/toplevel/ones
/toplevel/tens
/toplevel/hundreds
/toplevel/thausands
/toplevel/buzz
0
391
586
1000000
1111001
0000010
0010000
0000000
0110000
0010010
0.00 ns
400000000 ns
800000000 ns
1200000000 ns
1600000000 ns
Entity:toplevel Architecture: Date: Fri May 31 10:49:05 ICT 2019 Row: 1 Page: 1

4. Implementasi Lampiran

ketika program dijalankan, saklar geser sw0 pada board altera DE1 SoC pada keadaan “off”, maka sistem tidak akan bekerja. Sedangkan ketika sw0 pada keadaan “on”, maka sistem makan menampilkan jarak terukur antar sensor dengan dinding dalam satuan milimeter yang ditampilkan pada seven segment 4 digit.

5. Listing Program

```
//program untuk menyalakan buzzer
//frekuensi buzzer semakin tinggi ketika objek semakin dekat

module buzzer(state,en,clk,buzz);

`define dis 2'b00
`define low 2'b01
`define med 2'b10
`define hi 2'b11

parameter bitScaler1= 14;    //untuk intensitas tinggi
parameter bitScaler2= 2;     //untuk intensitas menengah
parameter bitScaler3= 2;     //untuk intensitas rendah

input state;
input [1:0] en;
input clk;
output buzz;

wire prescale1;
wire prescale2;
reg buzz;

reg [bitScaler1-1:0] count=0;
reg [bitScaler2-1:0] count1=0;
reg [bitScaler3-1:0] count2=0;

assign prescale1= count[13];
assign prescale2= count1[1];

always @(posedge (clk))
       count<= count+1'b1;

always @(posedge (prescale1))
       count1<= count1+1'b1;

always @(posedge (prescale2))
       count2<= count2+1'b1;

always @(*)
       if(state==1'b1)
       begin
       case(en)
       `dis: buzz<= 1'b0;
       `low: buzz<= count2[1];
       `med: buzz<= count1[1];
       `hi: buzz<= count[13];
       endcase
       end else buzz<= 1'b0;

endmodule
```

```
//generate sinyal dengan lebar pulsa 10uS setiap 10ms

module countPing(en,clkin,pulse);

`define high 1
`define low 0

input en;
input clkin;
output pulse;


parameter bitSize= 13;
reg [bitSize-1:0] count= 0;
reg pulse;

always @(posedge (clkin))
	case(en)
		`low: begin
			count<= 1'b0;
			pulse<= 1'b0;
		end
		`high: begin
			count<= count + 1'b1;
			if(count > 8190)
				pulse<= 1'b1;
			else
				pulse<= 1'b0;
		end
	endcase

endmodule
```

```
//ini untuk menghitung echo yang diterima
module echoCount(echoin,clk,latch,mode);
parameter bitSize= 10;

input clk;
input echoin;
output [bitSize-1:0] latch;
output [1:0] mode;

reg [bitSize-1:0] dist= 0;
reg [bitSize-1:0] latch;
reg [1:0] mode;

always @(negedge (echoin))
begin
	latch<= dist;
	if((latch>251) && (latch<280)) mode<= 2'b01;
	else if((latch>100) && (latch<250)) mode<= 2'b10;
	else if((latch>1) && (latch<99)) mode<= 2'b11;
	else mode<= 2'b00;
end
always @(posedge (clk))
begin
	if(echoin== 1'b1)
	dist<= dist+1'b1;
	else
	dist<= 1'b0;
end
endmodule
```

```
//prescaler dari 50Mhz dibagi 500 kali menjadi 100kHz untuk membangkitkan trigger
prescale(clkin,clkout);

parameter bitScale= 9;
input clkin;
output clkout;
reg [bitScale-1:0] count= 0;

assign clkout= count[bitScale-1];

always @(posedge (clkin))
       count<= count+1'b1;

endmodule
```

```
//mengubah clock 50Mhz dibagi 250 menjadi 200kHz untuk menghitung lebar echo
module prescalerEcho(clkin,clkout);

parameter bitScale= 8;

input clkin;
output clkout;
reg [bitScale-1:0] count=0;

assign clkout= count[bitScale-1];

always @(posedge (clkin))
       count<= count+1'b1;

Endmodule
```

```
//untuk tampil dari binary to decimal
module bcdto7seg(in,out);

parameter bitIn= 4;
parameter bitOut= 7;

input [bitIn-1:0] in;
output [bitOut-1:0] out;

reg [bitOut-1:0] out;
always @(*)
       case(in)
       4'h0: out<= 7'b1000000;
       4'h1: out<= 7'b1111001;
       4'h2: out<= 7'b0100100;
       4'h3: out<= 7'b0110000;
       4'h4: out<= 7'b0011001;
       4'h5: out<= 7'b0010010;
       4'h6: out<= 7'b0000010;
       4'h7: out<= 7'b1111000;
       4'h8: out<= 7'b0000000;
       4'h9: out<= 7'b0010000;
       4'ha: out<= 7'b1000000;
       4'hb: out<= 7'b1000000;
       4'hc: out<= 7'b1000000;
       4'hd: out<= 7'b1000000;
       4'he: out<= 7'b1000000;
       endcase
endmodule
```

```
//shift and add3

module shiftadd(in,ones,tens,hundreds,thausands);

parameter bitIn= 10;
parameter bitOut= 4;

input [bitIn-1:0] in;
output [bitOut-1:0] ones;
output [bitOut-1:0] tens;
output [bitOut-1:0] hundreds;
output [bitOut-1:0] thausands;

wire [bitOut-1:0] c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12;
wire [bitOut-1:0] d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12;

assign d1= {1'b0,in[9:7]};
assign d2= {c1,in[6]};
assign d3= {c2,in[5]};
assign d4= {c3,in[4]};
assign d5= {c4,in[3]};
assign d6= {c5,in[2]};
assign d7= {c6,in[1]};
assign d8= {1'b0,c1[3],c2[3],c3[3]};
assign d9= {c8,c4[3]};
assign d10= {c9,c5[3]};
assign d11= {c10,c6[3]};
assign d12= {1'b0,c8[3],c9[3],c10[3]};

add3 m1(d1,c1);
add3 m2(d2,c2);
add3 m3(d3,c3);
add3 m4(d4,c4);
add3 m5(d5,c5);
add3 m6(d6,c6);
add3 m7(d7,c7);
add3 m8(d8,c8);
add3 m9(d9,c9);
add3 m10(d10,c10);
add3 m11(d11,c11);
add3 m12(d12,c12);

assign ones= {c7[2:0],in[0]};
assign tens= {c11[2:0],c7[3]};
assign hundreds= {c12[2:0],c11[3]};
assign thausands= {1'b0,1'b0,1'b0,c12[3]};

endmodule
```