Infrared Remote Controlled Robot

Goal

The goal of this example is to implement wireless controls on the robot used in class. A television remote from Samsung is selected that communicates using an infrared led. On the robot an infrared receiver (part no. TSOP 38238) receives infrared light signals with a 38 kHz frequency. The IR receiver protocol in Figure 1 is implemented according to the timing requirements for the Samsung remote controller. The receiver outputs pulses that represent the signal sent by the remote. The DE0-NANO FPGA receives the signals and determines a direction each of the servo motors should turn. The setup for this example is shown in Figure 2.

IR Signal

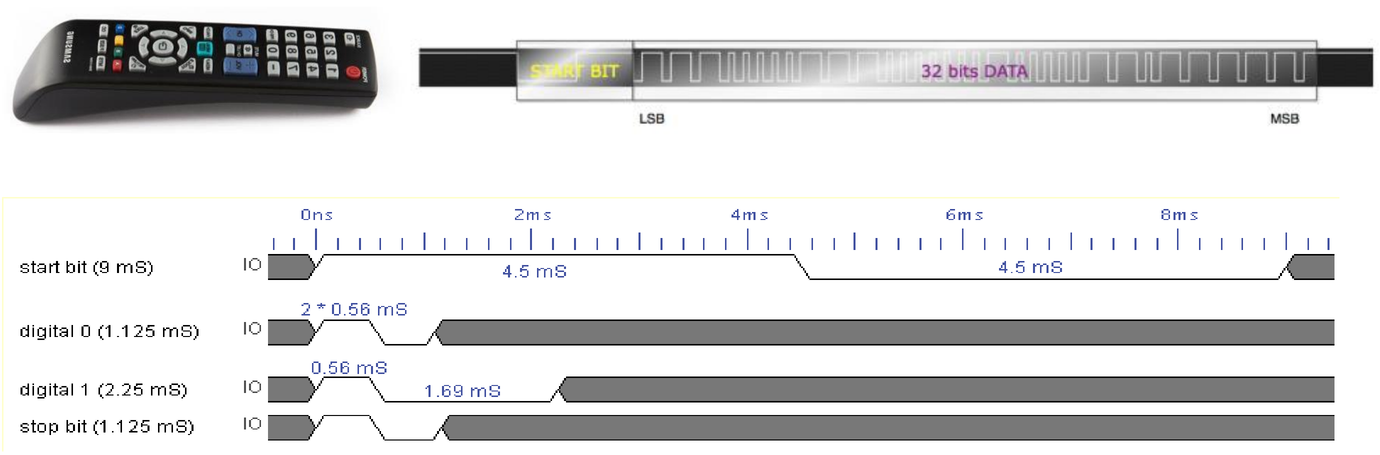


Fig. 1 Samsung remote control and its signal protocol

Schematic

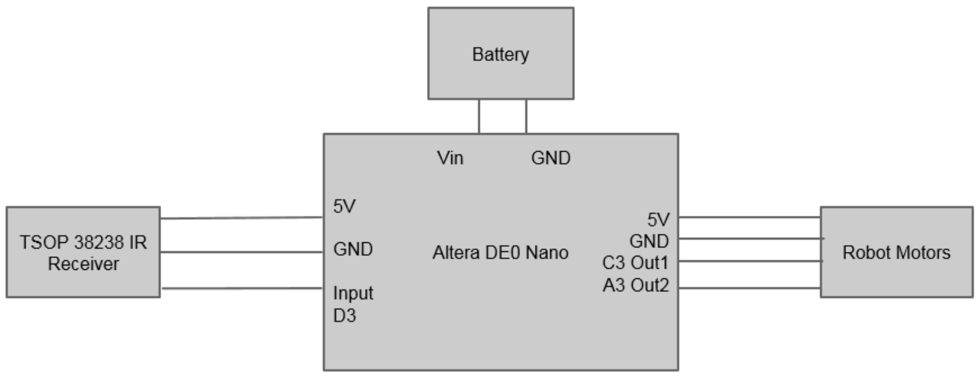


Fig. 2 schematic

Implementation:

Code

Top Module: Used for creating instances of the other modules as well as creating a clock to poll the IR receiver and a bit to stop motor movement after some time without receiving a command.

```
module RemoteSensor(clk, IR, led, pwm_left, pwm_right);
input clk;
input IR;
output [7:0] led;
output pwm_left; //Controls left motor
output pwm_right; //Controls right motor

//Generates clock for IR Receiver
reg [31:0] counter;
parameter delay = 4000;
reg OnOff;
wire clk2;

reg [7:0] lights; //Variable for Leds

wire [7:0] duration; //Duration of a high pulse
wire HighPulseCompleted; //High Pulse was received
wire [1:0] convertedState; //Result of high pulse (Start bit, 1, 0);
wire [31:0] command; //32 bit command from ir receiver
wire commandReceived; //Command received signal
wire [2:0] direction; //Direction robot should move
reg stop = 1; //Stops the robot if on
reg [31:0] delayCounter; //Lets robot move for a little bit after button release
```

```
//Generates clock for IR Receiver
always @(posedge clk)
begin
   if (counter == delay)
   begin
      OnOff <= !OnOff;
      counter <= 0;
   end
   else
      counter <= counter +1;
end

assign clk2 = OnOff; //(50000000/2/4000) = 6250hz clock

/*
IR RECEIVER
Input: clock and IR receiver input pin
Output: Duration of high pulse and signal when pulse is complete
*/
IR_Receive_Bits RECEIVER(.clk(clk2), .in(IR), .out(duration), .valid(HighPulseCompleted));
/*
Signal to Code
Input: Duration of high pulse
Output: Result of high pulse (Start bit, 1, 0)
*/
SignalToCode CONVERTER(.in(duration), .out(convertedState));
```

```
/*
Register
Input: pulse completed signal, state of pulse
Output: 32 bit command received by remote and signal that command is received
*/
Register REGISTER(.valid(HighPulseCompleted), .convertedState(convertedState), .command(command), .commandReceived(commandReceived));
/*
Decode Command
Input: 32 bit command and command received signal
Output: direction robot should move
*/
Decode_Command DECODER(.command(command), .commandReceived(commandReceived), .dir(direction));
/*
Robot Control
Input: Clock and direction
Output: PWM output for each motor
*/
Robot_Control ROBOT(.clk(clk),.direction(direction[2:0]+4*stop),.pwm_left(pwm_left),.pwm_right(pwm_right));
//Stops motors when button is let go
always @(posedge clk2 or posedge commandReceived)
begin
   if (commandReceived) //If button is pressed reset values and allow motors to move.
   begin
      stop <= 0;
      delayCounter <= 0;
   end
   else if (delayCounter == 1000) //If counter is reached stop motors.
   begin
      stop <= 1;
      delayCounter <= 0;
   end
   else //Increment counter.
      delayCounter <= delayCounter + 1;
end

//Displays direction status on leds
assign led = lights;
always @(*)
begin
      lights[7:0] <= direction;
end

endmodule
```

Receive Bits: Measures the duration of the positive signals sent by the IR receiver. Sets a bit when the signal goes low again.

```
module IR_Receive_Bits(clk, in, out, valid);
input clk;
input in;
output [7:0] out; //duration of a high pulse
output valid; //High pulse is finished

reg isValid;
reg [7:0] data;

always @(posedge clk)
begin
         if(in == 0) //If low input
            isValid <= 1; //High pulse is finished
         else
         begin
            if (isValid == 1) //If new pulse
               data = 0; //reset data
            isValid <= 0; //High pulse is in progress
            data <= data +1; //Count duration of pulse
         end
end

assign out = data;
assign valid = isValid;
endmodule
```

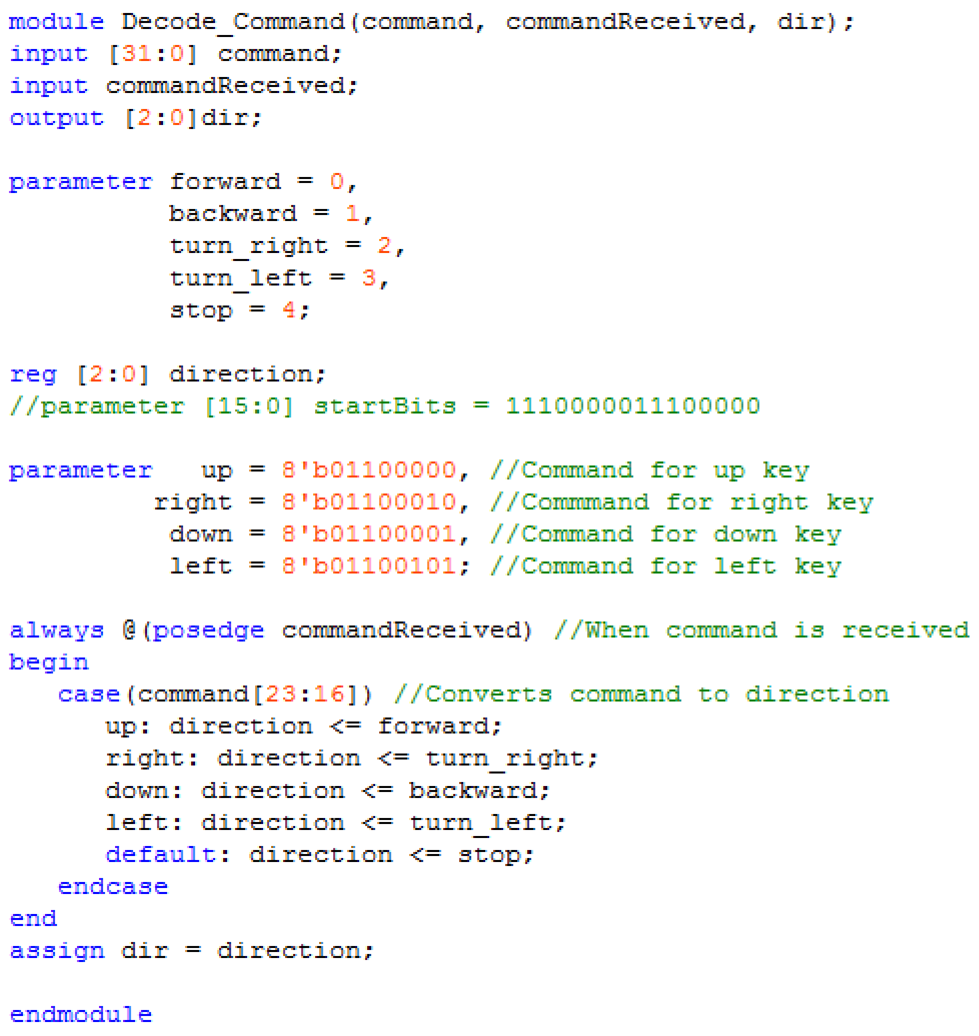
Signal duration to code value: Converts the duration of the high pulse into either a 1, 0, or start bit.

```
module SignalToCode(in, out);
input [7:0] in;
output [1:0] out;

reg [1:0] state; //State of high pulse
parameter start = 2,
          high = 1,
          low = 0,
          error = 3;

always @(*)
begin
   if (in > 25 && in < 30) //If this duration its a start bit
      state <= start;
   else if (in > 7 && in < 13) //Long duration = high bit
      state <= high;
   else if (in > 2 && in < 6) //Short duration = low bit
      state <= low;
   else
      state <= error;  //Error if none of the above
end

assign out = state;
endmodule
```

Register: Waits for a start bit then buffers the next 32 bits. After a complete code is received a bit is toggled indicating it is finished.

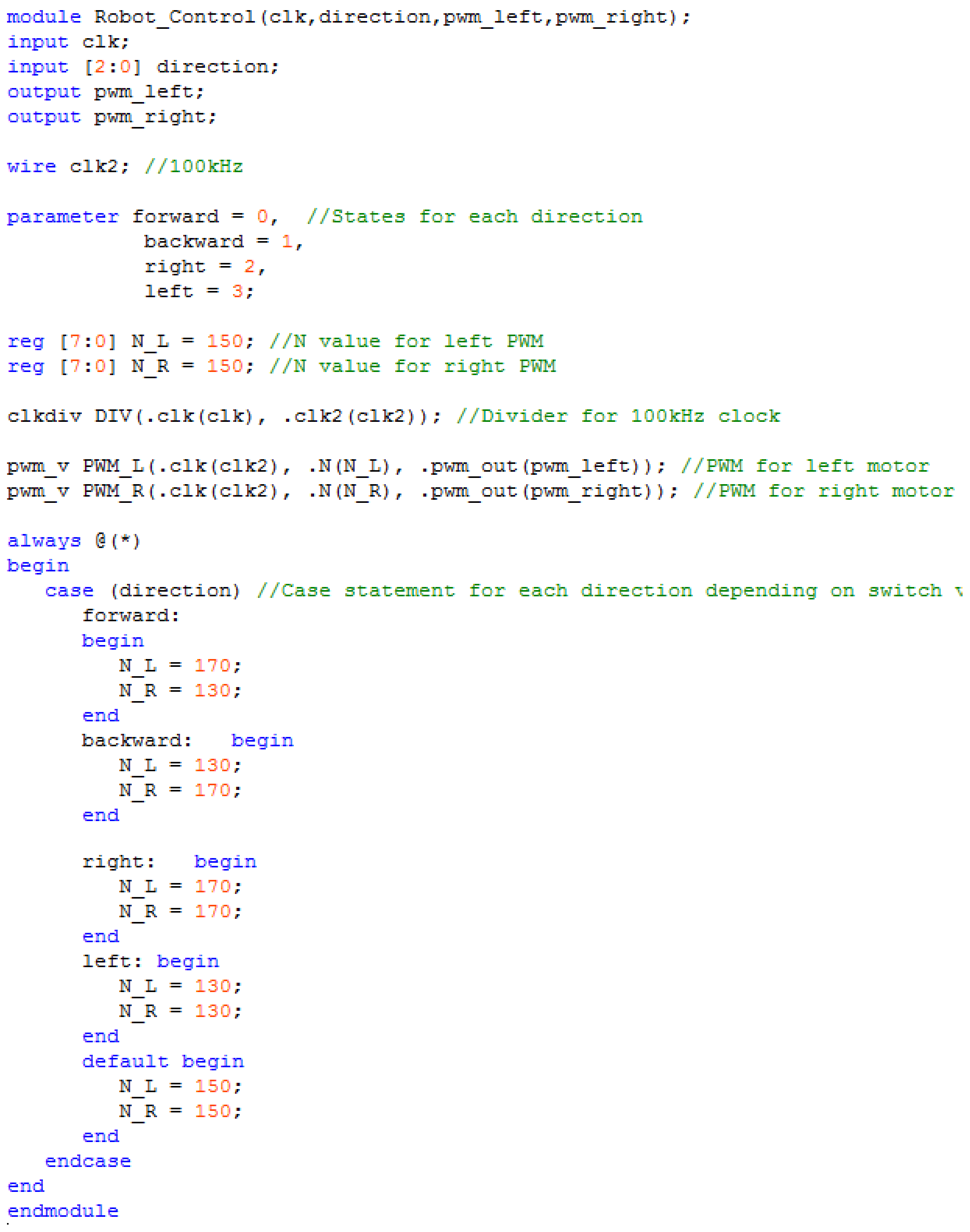
```
module Register(valid, convertedState, command, commandReceived);
input valid; //high pulse received signal
input [1:0] convertedState; //State of high pulse
output [31:0] command; //32 bit command
output commandReceived; //Entire command received

reg [1:0] state;
reg [4:0] i;
reg finished;
reg [31:0] data;
parameter waiting = 0, started = 1, last = 2; //States

always @(posedge valid) //Everytime high pulse is received
begin
   case (state)
      waiting:  //Waiting for command
      begin
         finished <= 0;
         if(convertedState == 2) //If start bit, change state
            state <= started;
         i <= 0;
      end
      started: //Receiving command
      begin
         if (convertedState == 0) //If state low add a 0
            data[i] <= 0;
         else if (convertedState == 1) //If state high add a 1
            data[i] <= 1;
         if(i > 30) //If received all the bits change state
            state <= last;
         if(convertedState == 2) //if received another start bit
            i <= 31; //Set i to end of commmand so it restarts
         i <= i + 1; //Increment index in command
      end
      last: //Receiving last bit
      begin
         finished <= 1;
         state <= waiting; //Wait for new command
      end
   endcase
end

assign commandReceived = finished;
assign command = data;
endmodule
```

Decoder: After a command is received the bits are decoded to determine which button was pressed and sets the direction depending on that result.



```verilog
module Decode_Command(command, commandReceived, dir);
input [31:0] command;
input commandReceived;
output [2:0]dir;

parameter forward = 0,
          backward = 1,
          turn_right = 2,
          turn_left = 3,
          stop = 4;

reg [2:0] direction;
//parameter [15:0] startBits = 1110000011100000

parameter   up = 8'b01100000, //Command for up key
         right = 8'b01100010, //Commmand for right key
          down = 8'b01100001, //Command for down key
          left = 8'b01100101; //Command for left key

always @(posedge commandReceived) //When command is received
begin
   case(command[23:16]) //Converts command to direction
      up: direction <= forward;
      right: direction <= turn_right;
      down: direction <= backward;
      left: direction <= turn_left;
      default: direction <= stop;
   endcase
end
assign dir = direction;

endmodule
```

Robot Controller: Once a direction is determined it is inputted into a module to move the robot in the proper direction.



```
module Robot_Control(clk,direction,pwm_left,pwm_right);
input clk;
input [2:0] direction;
output pwm_left;
output pwm_right;

wire clk2; //100kHz

parameter forward = 0,  //States for each direction
           backward = 1,
           right = 2,
           left = 3;

reg [7:0] N_L = 150; //N value for left PWM
reg [7:0] N_R = 150; //N value for right PWM

clkdiv DIV(.clk(clk), .clk2(clk2)); //Divider for 100kHz clock

pwm_v PWM_L(.clk(clk2), .N(N_L), .pwm_out(pwm_left)); //PWM for left motor
pwm_v PWM_R(.clk(clk2), .N(N_R), .pwm_out(pwm_right)); //PWM for right motor

always @(*)
begin
   case (direction) //Case statement for each direction depending on switch
      forward:
      begin
         N_L = 170;
         N_R = 130;
      end
      backward:   begin
         N_L = 130;
         N_R = 170;
      end

      right:   begin
         N_L = 170;
         N_R = 170;
      end
      left: begin
         N_L = 130;
         N_R = 130;
      end
      default begin
         N_L = 150;
         N_R = 150;
      end
   endcase
end
endmodule
```

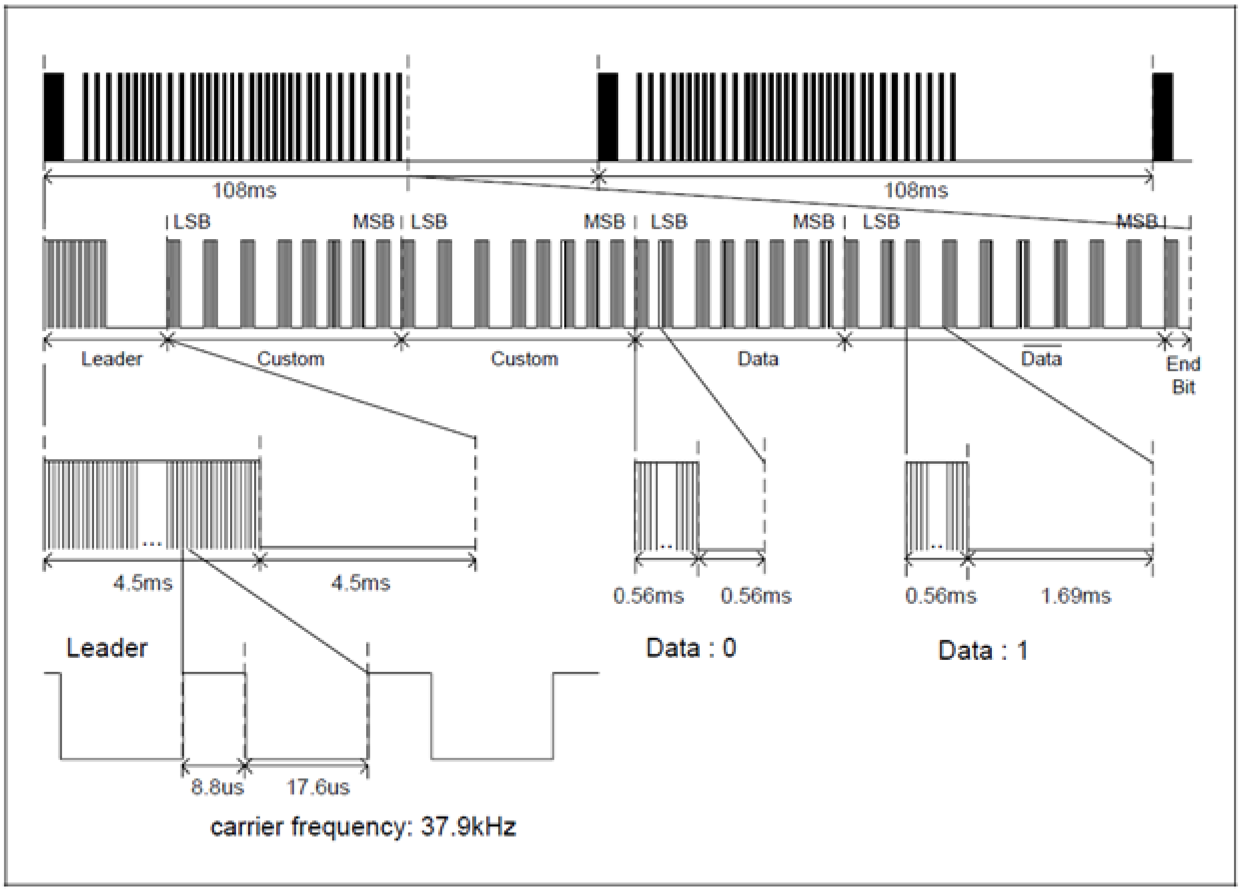
Procedure

To run this example an IR receiver (model no. TSOP 38238) needs to be connected to 3.3V, ground, and pin D3 on the DE0-NANO FPGA board. The left motor needs to be connected to pin C3 and the right to pin A3. After that is complete a TV remote can control the robot. A TV remote made by Samsung is used in this example.

IR SIGNAL:

All remote controller manufactures define IR protocol which is specified on their products or companies. This application note describes example of IR protocol. This IR protocol determines the bit 1 and 0 by pulse distance. Carrier frequency is 37.9 kHz and the frame period is 108ms. Leader is the start of frame. Following custom code is remote controller manufacture's customer code. It consists of 2 Bytes which is same each bytes. Data code and Data bar which is complemented Data code is sequentially transmitted. Last the end bit is transmitted.

The example protocol is shown Figure 1 below. (Custom code: 0x07H, Datacode: 0x02H)


108ms
108ms
LSB
MSB
LSB
MSB
LSB
MSB
LSB
MSB
Leader
Custom
Custom
Data
$\overline{\text{Data}}$
End Bit
4.5ms
4.5ms
0.56ms
0.56ms
0.56ms
1.69ms
Leader
Data : 0
Data : 1
8.8us
17.6us
carrier frequency: 37.9kHz