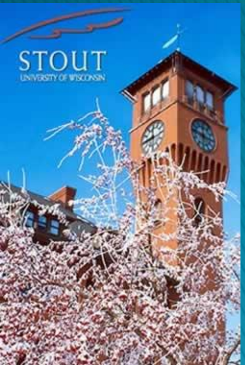
# CEE 445

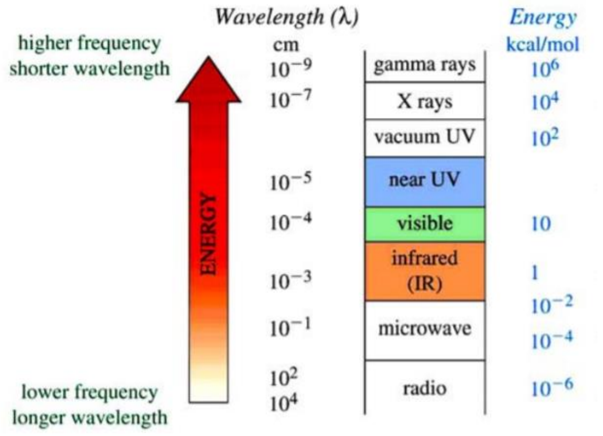
TV Remote Controller and Decoding Technique

Cheng Liu



# How TV Remote Control Works?

- TV Remote (transmitter and receiver operated between 38K and 56KHz)
  - Consists of an IR LED transmitter and a photo-sensitive detector/receiver (eg. Tsop38238, wavelength at 980nm)
  - Wireless communication (typically at 38KHz)
  - The messages travel via electromagnetic waves just above radio in frequency – the infrared.
- How it Works
  - The LED transmitter sends a special code assigned to that key as a beam of infrared light pulses.
  - The infrared receiver converts it into electrical impulses that correspond to the key code that controls the appliance.



Graphics source: Wade, Jr., Pearson Education Inc., 2003

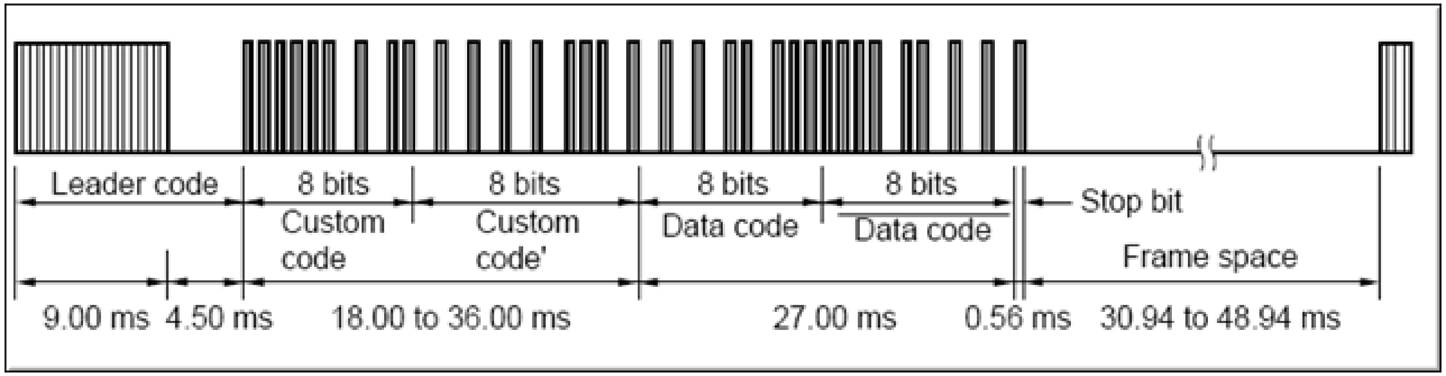
## Modulation

- Problem:
  - The sun and light bulbs radiate infrared lights at 980 nm. ☹
  - A solution needed for the appliance infrared receiver to differentiate between infrared light sent by the remote control and the infrared light from the sun.

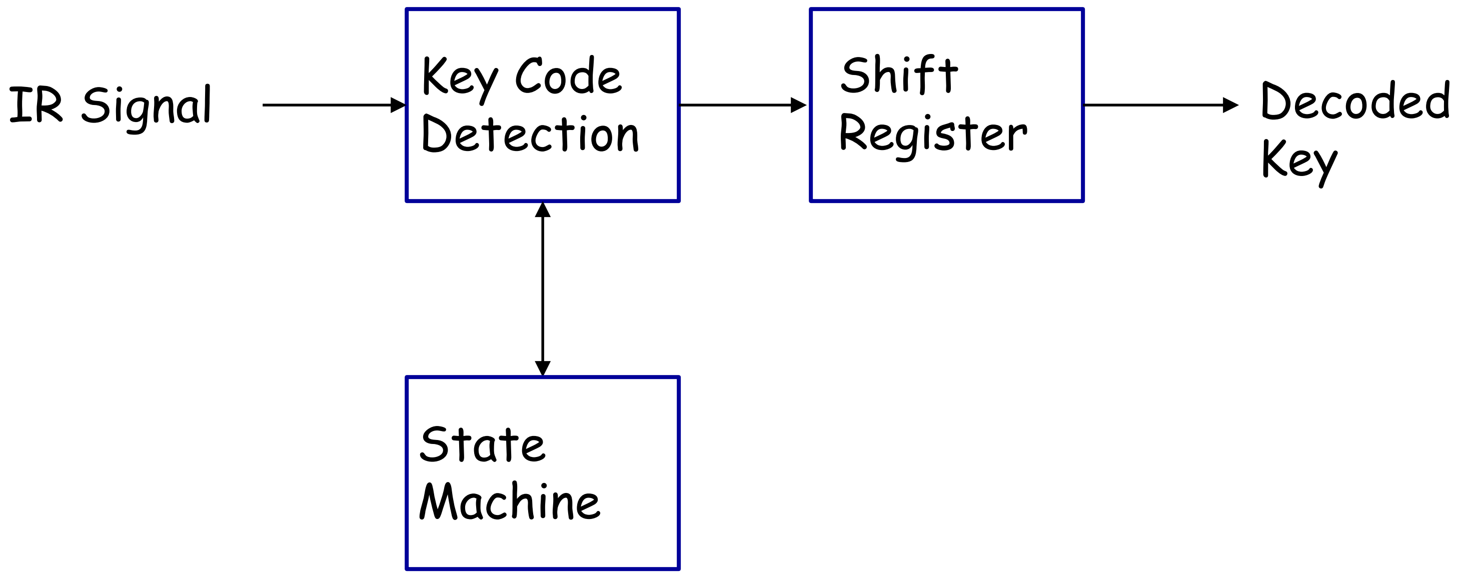
- Modulation to the rescue
  - The appliance infrared receiver is designed to respond not only to infrared light at a frequency of 980nm but also to light that is modulated at a frequency of 38KHz.
  - The remote control can ignore signals except the infrared light beam modulated at 38KHz.

## NEC IR Protocol

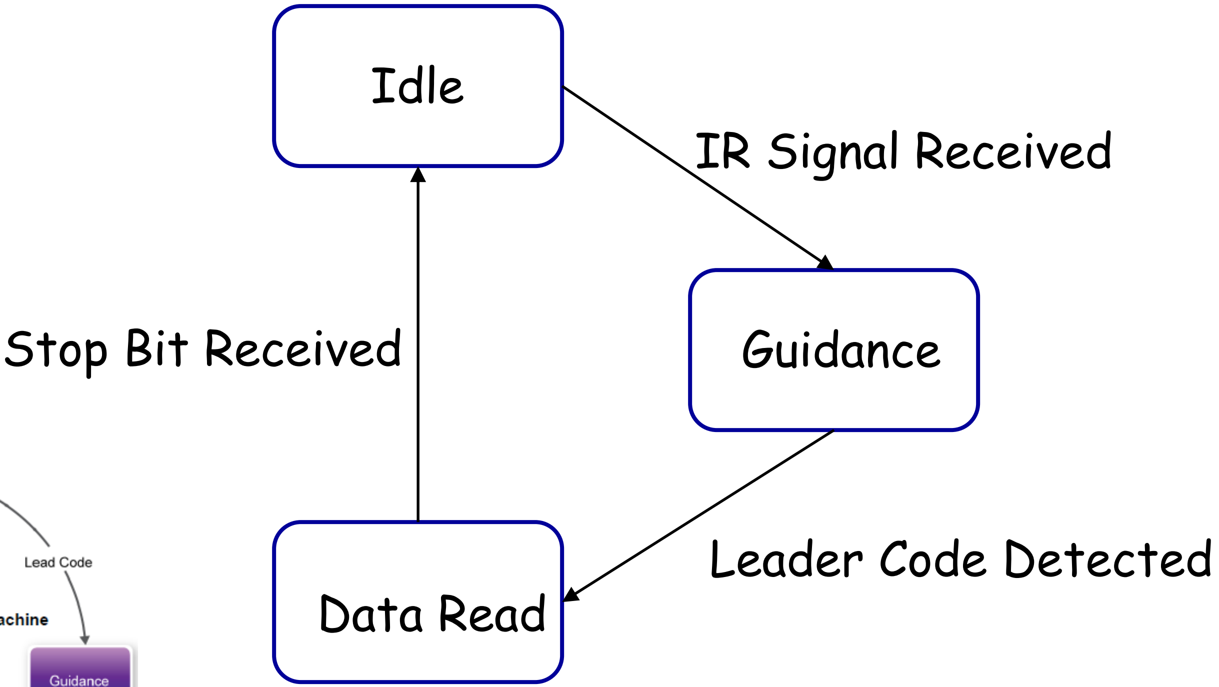
- One of the commonly used IR protocols
- NEC format:
  - 8-bit custom code and 8-bit data code
  - 38KHz – Carrier Frequency
  - Pulse modulation
  - Leader code has 9ms carrier waveform and 4.5ms OFF waveform, followed by a custom code and data code and their inverse.

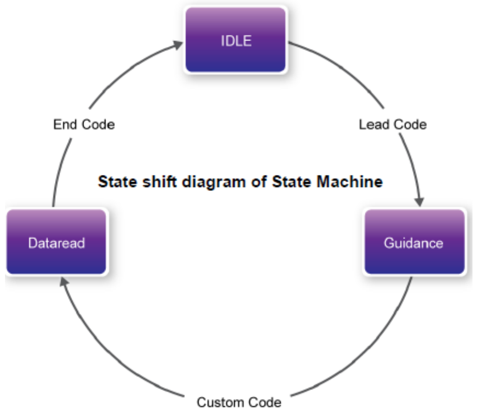


# IR Decoder


IR Signal
Key Code Detection
Shift Register
Decoded Key
State Machine

## State Machine



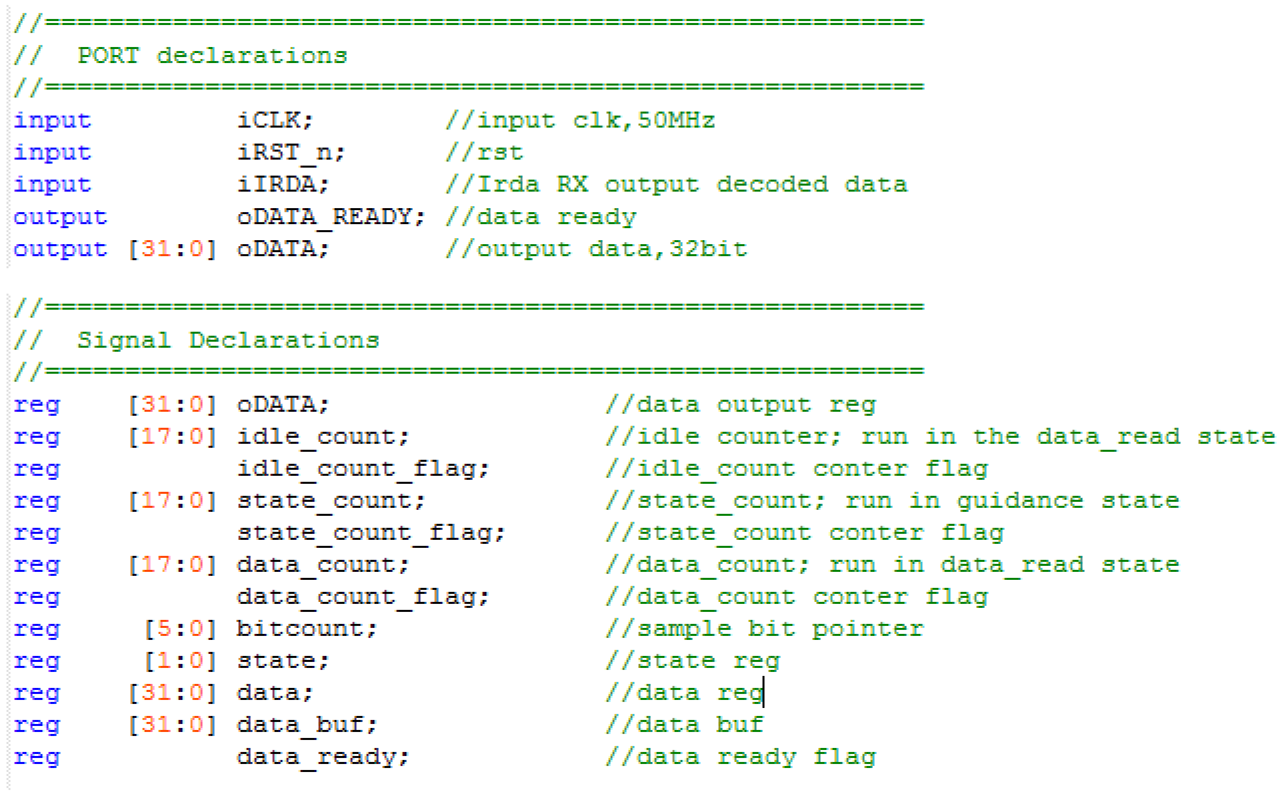


## Lab 8 IR Sensor Interface w/FPGA

# Verilog Model (IR Receiver)

```
  module IR_RECEIVE(
                 iCLK,          //clk 50MHz
                 iRST_n,        //reset
                 iIRDA,         //IR code input
                 oDATA_READY,   //data ready
                 oDATA          //decode data output
                 );

//=======================================================
//  PARAMETER declarations
//=======================================================
parameter IDLE              = 2'b00;  //state Idle: always at a high voltage level
parameter GUIDANCE          = 2'b01;  //state guidance; 9ms at a low voltage and 4.5 ms at a high voltage
parameter DATAREAD          = 2'b10;  //state; read IR data


parameter IDLE_HIGH_DUR     = 262143; // data_count    262143*0.02us = 5.24ms, threshold for DATAREAD-----> IDLE
parameter GUIDE_LOW_DUR     = 230000; // idle_count    230000*0.02us = 4.60ms, threshold for IDLE--------->GUIDANCE
parameter GUIDE_HIGH_DUR    = 210000; // state_count   210000*0.02us = 4.20ms, 4.5-4.2 = 0.3ms < BIT_AVAILABLE_DUR =
                                      // 0.4ms,threshold for GUIDANCE------->DATAREAD
parameter DATA_HIGH_DUR     = 41500;  // data_count    41500 *0.02us = 0.83ms, sample time from the posedge of iIRDA
parameter BIT_AVAILABLE_DUR = 20000;  // data_count    20000 *0.02us = 0.4ms,  the sample bit
                                      // pointer,can inhibit the interference from iIRDA signal
```

# Verilog Model (IR Receiver)



```
//=======================================================
//  PORT declarations
//=======================================================
input         iCLK;        //input clk,50MHz
input         iRST_n;      //rst
input         iIRDA;       //Irda RX output decoded data
output        oDATA_READY; //data ready
output [31:0] oDATA;       //output data,32bit

//=======================================================
//  Signal Declarations
//=======================================================
reg    [31:0] oDATA;                 //data output reg
reg    [17:0] idle_count;            //idle counter; run in the data_read state
reg           idle_count_flag;       //idle_count conter flag
reg    [17:0] state_count;           //state_count; run in guidance state
reg           state_count_flag;      //state_count conter flag
reg    [17:0] data_count;            //data_count; run in data_read state
reg           data_count_flag;       //data_count conter flag
reg     [5:0] bitcount;              //sample bit pointer
reg     [1:0] state;                 //state reg
reg    [31:0] data;                  //data reg
reg    [31:0] data_buf;              //data buf
reg           data_ready;            //data ready flag
```

# Verilog Model (IR Receiver)

```
//=========================================================
//   Structural coding
//=========================================================
assign oDATA_READY = data_ready;

//idle counter runs at IDLE state only
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         idle_count <= 0;
     else if (idle_count_flag)    //the counter runs when the flag is 1
          idle_count <= idle_count + 1'b1;
      else
          idle_count <= 0;          //the counter resets when the flag is 0
```

Idle count is to measure the IR LED "ON" time during the IDLE state

```
//idle counter job complete when iIRDA is low in the IDLE state
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         idle_count_flag <= 1'b0;
     else if ((state == IDLE) && !iIRDA)
          idle_count_flag <= 1'b1;
      else
          idle_count_flag <= 1'b0;
```

Idle count flag indicates IR LED "ON" time ends and the counter job and IR LED returns To LOW state.

# Verilog Model (IR Receiver)

```
//state counter runs in the GUIDANCE state only
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         state_count <= 0;
     else if (state_count_flag)     //the counter runs when the flag is 1
          state_count <= state_count + 1'b1;
      else
          state_count <= 0;            //the counter resets when the flag is 0
```

State count is to measure "OFF" time of the IR LED during the GUIDANCE state

```
//state counter job complete when iIRDA is high in the GUIDANCE state
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         state_count_flag <= 1'b0;
     else if ((state == GUIDANCE) && iIRDA)
          state_count_flag <= 1'b1;
      else
          state_count_flag <= 1'b0;
```

State count flag is to detect if the IR LED Changes from "ON" to "OFF"

# Verilog Model (IR Receiver)

Data count variable detects if IR data is '1' or '0'. This counter increments when a logic '1' is received.

```
//data read counter runs on iCLK clock
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         data_count <= 1'b0;
     else if(data_count_flag)      //the counter runs when the flag is 1
          data_count <= data_count + 1'b1;
      else
          data_count <= 1'b0;        //the counter resets when the flag is 0

//data counter job complete
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         data_count_flag <= 0;   |
     else if ((state == DATAREAD) && iIRDA)
          data_count_flag <= 1'b1;
      else
          data_count_flag <= 1'b0;
```

Data count flag detects if IR LED changes from "ON" to "OFF"

# Verilog Model

```
//data reg pointer counter
always @(posedge iCLK or negedge iRST_n)
    if (!iRST_n)
       bitcount <= 6'b0;
     else if (state == DATAREAD)
      begin
         if (data_count == 20000)
               bitcount <= bitcount + 1'b1; //add 1 when iIRDA posedge
      end
     else
        bitcount <= 6'b0;
```

Data reg. pointer counter determines how many data bit to read in the Data Read state.

```
//state change between IDLE,GUIDE,DATA_READ according to irda edge or counter value
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
        state <= IDLE;
     else
          case (state)
             IDLE     : if (idle_count > GUIDE_LOW_DUR)  // state chang from IDLE to Guidance when detect the negedge
                                                         //and the low voltage last for > 4.6ms
                          state <= GUIDANCE;
             GUIDANCE : if (state_count > GUIDE_HIGH_DUR)//state change from GUIDANCE to DATAREAD when detect
                                                         //the posedge and the high voltage last for > 4.2ms
                          state <= DATAREAD;
             DATAREAD : if ((data_count >= IDLE_HIGH_DUR) || (bitcount >= 33))
                              state <= IDLE;
           default  : state <= IDLE; //default
          endcase
```

State Machine

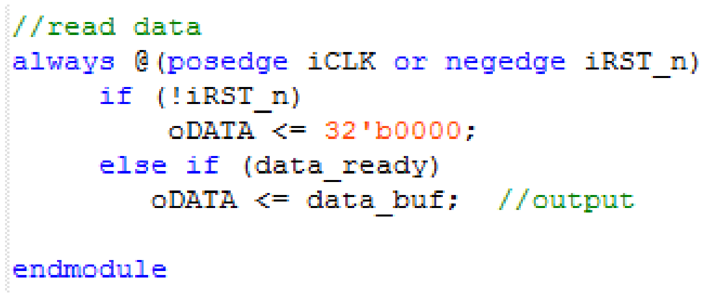
# Verilog Model (IR Receiver)

```
//data decoding
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
        data <= 0;
      else if (state == DATAREAD)
      begin
          if (data_count >= DATA_HIGH_DUR) //2^15 = 32767*0.02us = 0.64us
             data[bitcount-1'b1] <= 1'b1;  //>0.52ms  sample the bit 1
      end
      else
          data <= 0;


      //set the data_ready flag
      always @(posedge iCLK or negedge iRST_n)
           if (!iRST_n)
              data_ready <= 1'b0;
          else if (bitcount == 32)
            begin
                if (data[31:24] == ~data[23:16])
                begin
                     data_buf <= data;      //fetch the value to the databuf from the data reg
                    data_ready <= 1'b1;    //set the data ready flag
                end
                else
                    data_ready <= 1'b0 ;   //data error
            end
            else
               data_ready <= 1'b0 ;
```

Shift register – storing the IR data into a data array.

Data ready is set and IR data is stored in a data array only if the IR code matches with a key press on the remote control.

## Verilog Model (IR Receiver)

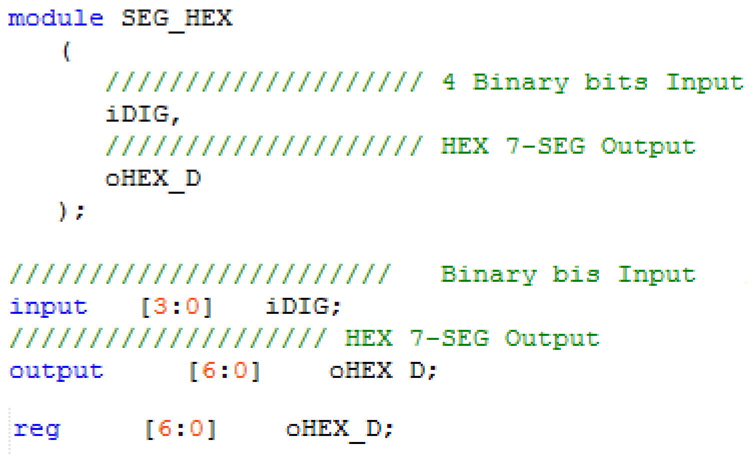


```
//read data
always @(posedge iCLK or negedge iRST_n)
     if (!iRST_n)
         oDATA <= 32'b0000;
     else if (data_ready)
        oDATA <= data_buf;  //output

endmodule
```

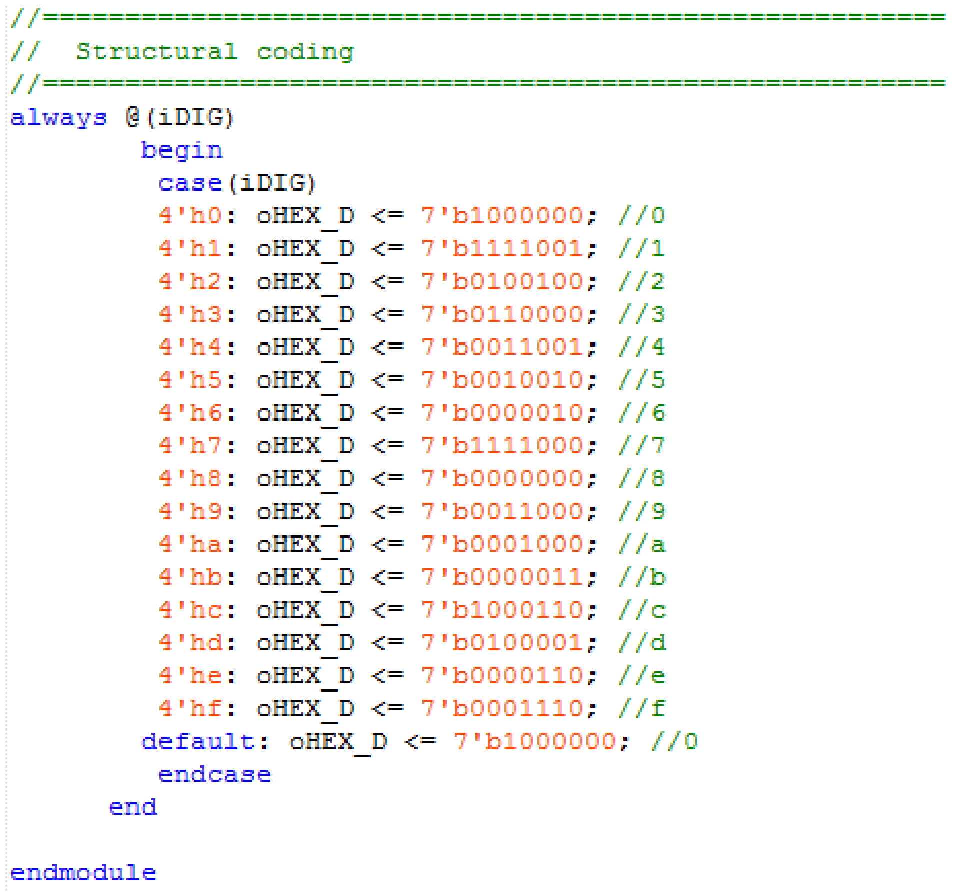
The oData variable contains a valid IR code

# Verilog Model (7-Seg Display)

A valid IR code is then displayed on a 7-seg. display

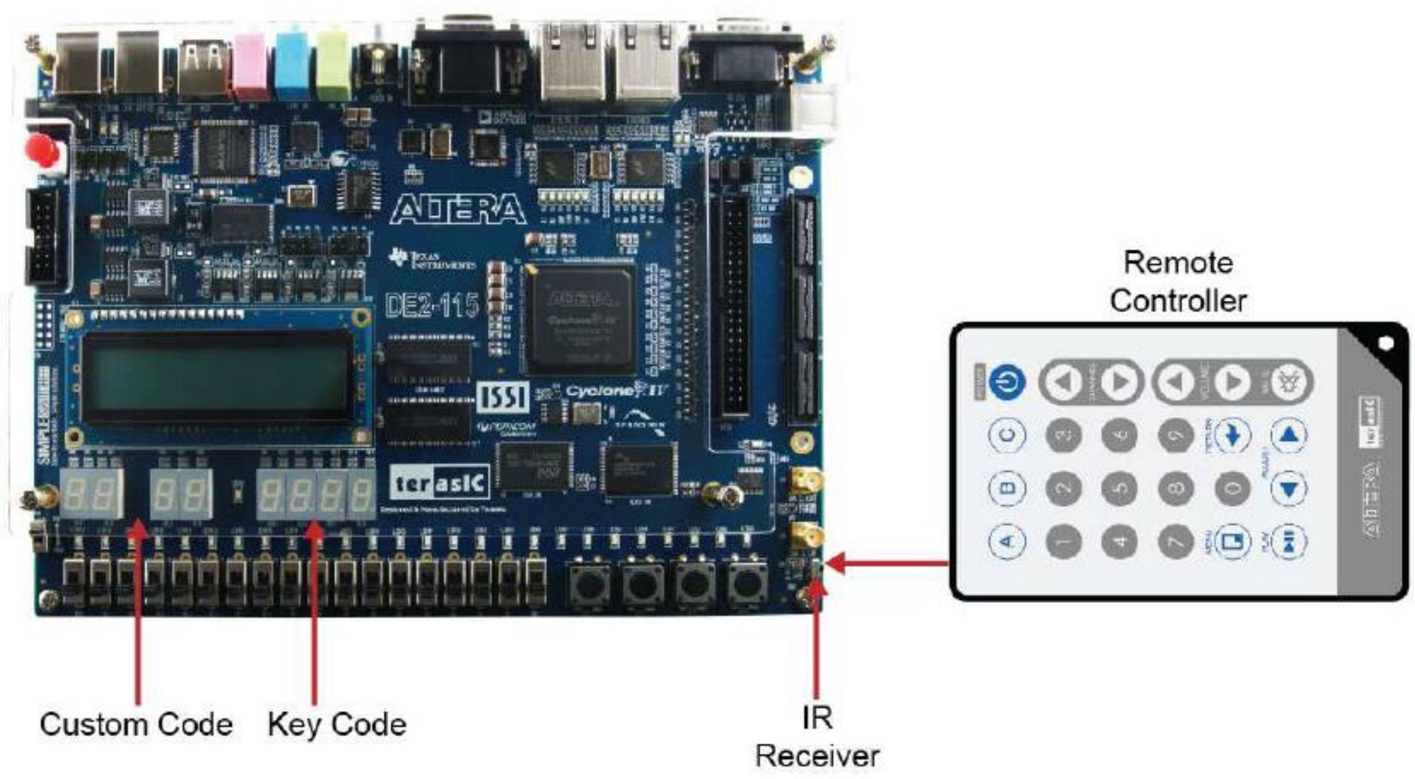


```
module SEG_HEX
   (
      ///////////////////// 4 Binary bits Input
      iDIG,
      ///////////////////// HEX 7-SEG Output
      oHEX_D
   );

////////////////////////   Binary bis Input
input    [3:0]    iDIG;
//////////////////// HEX 7-SEG Output
output      [6:0]     oHEX_D;

reg      [6:0]     oHEX_D;
```



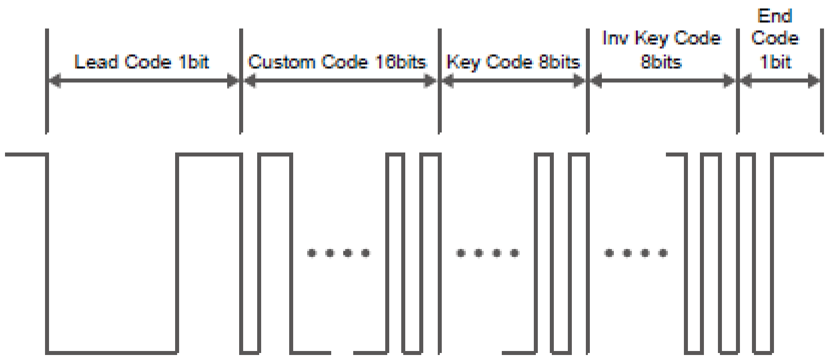
```
//=========================================================
//  Structural coding
//=========================================================
always @(iDIG)
        begin
         case(iDIG)
         4'h0: oHEX_D <= 7'b1000000; //0
         4'h1: oHEX_D <= 7'b1111001; //1
         4'h2: oHEX_D <= 7'b0100100; //2
         4'h3: oHEX_D <= 7'b0110000; //3
         4'h4: oHEX_D <= 7'b0011001; //4
         4'h5: oHEX_D <= 7'b0010010; //5
         4'h6: oHEX_D <= 7'b0000010; //6
         4'h7: oHEX_D <= 7'b1111000; //7
         4'h8: oHEX_D <= 7'b0000000; //8
         4'h9: oHEX_D <= 7'b0011000; //9
         4'ha: oHEX_D <= 7'b0001000; //a
         4'hb: oHEX_D <= 7'b0000011; //b
         4'hc: oHEX_D <= 7'b1000110; //c
         4'hd: oHEX_D <= 7'b0100001; //d
         4'he: oHEX_D <= 7'b0000110; //e
         4'hf: oHEX_D <= 7'b0001110; //f
        default: oHEX_D <= 7'b1000000; //0
         endcase
      end

endmodule
```

## Demo



Key code information for each Key on remote controller

| Key | Key Code | Key | Key Code | Key | Key Code | Key | Key Code |
|---|---|---|---|---|---|---|---|
| A | 0x0F | B | 0x13 | C | 0x10 | ⏻ | 0x12 |
| 1 | 0x01 | 2 | 0x02 | 3 | 0x03 | ▲ | 0x1A |
| 4 | 0x04 | 5 | 0x05 | 6 | 0x06 | ▼ | 0x1E |
| 7 | 0x07 | 8 | 0x08 | 9 | 0x09 | ▲ | 0x1B |
| ⧉ | 0x11 | 0 | 0x00 | ↩ | 0x17 | ▼ | 0x1F |
| ⏯ | 0x16 | ◀ | 0x14 | ▶ | 0x18 | 🔇 | 0x0C |



The transmitting frame of the IR remote controller