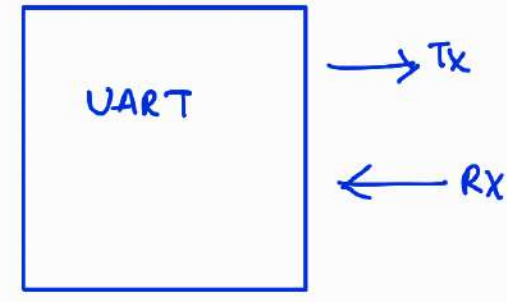
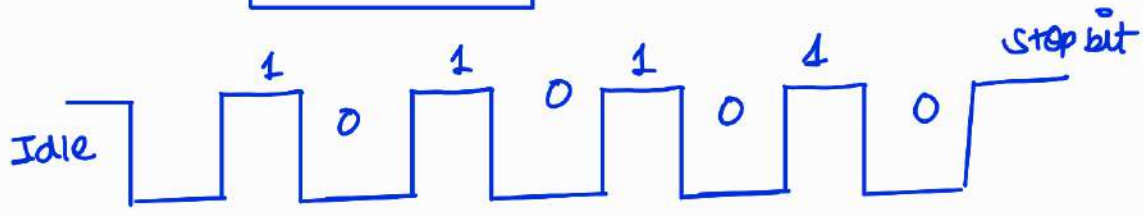
→ UART : Asynchronous Mode of communication

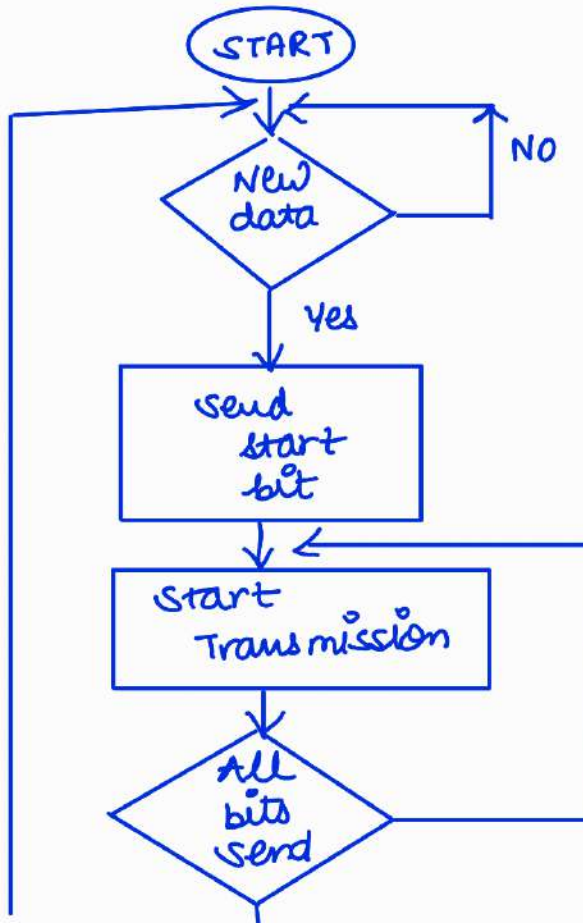
- Tx pin transmits data from transmitter to receiver (Default high state). for transmitting data a low will be send first in the form of start bit for a single bit duration



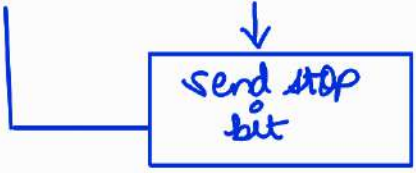


: FOR TRANSMISSION MODULE :

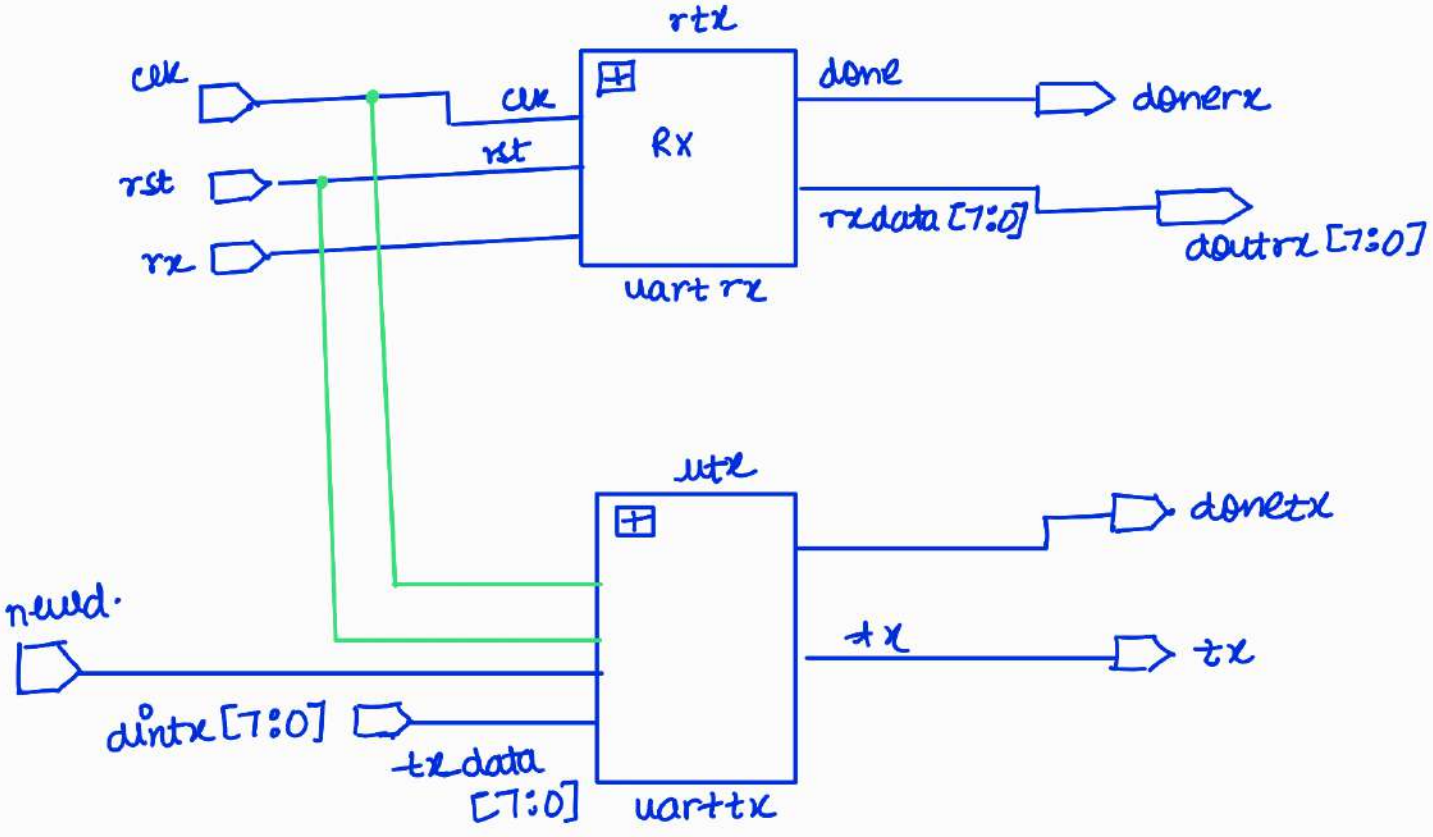
→ FLOWCHART FOR TRANSMISSION OF DATA: (GENERALIZED FLOWCHART)



During transmission LSB is send 1st.



→ SCHEMATIC OF OUR DESIGN :



→ Transmitter I/p port : clk, (rst) reset, newd, tx_data
(global)

→ Transmitter op port : donetx, tx

→ As soon as user have new data it'll tell this to the transmitter module by making newd signal high or sending 1 (High) to newd. which will sample data present on tx_data [7:0] & transmission of this data bit by bit happens on tx line as soon as transmission of data is done donetx will become high to indicate transmission

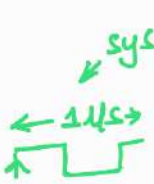
is done only for a single clock tick done tx will remain high (only)

CLOCK Generation Explanation:

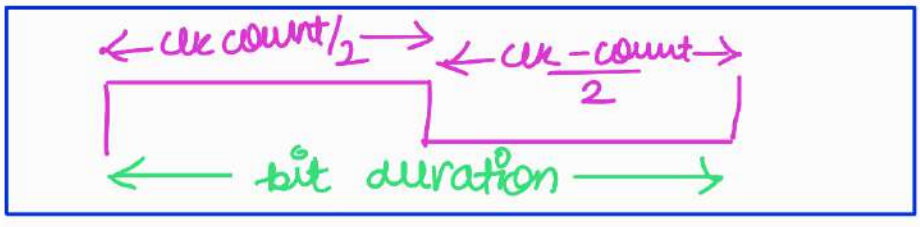
sys_clk = 1 MHz ⇒ clk_period = 1 µs

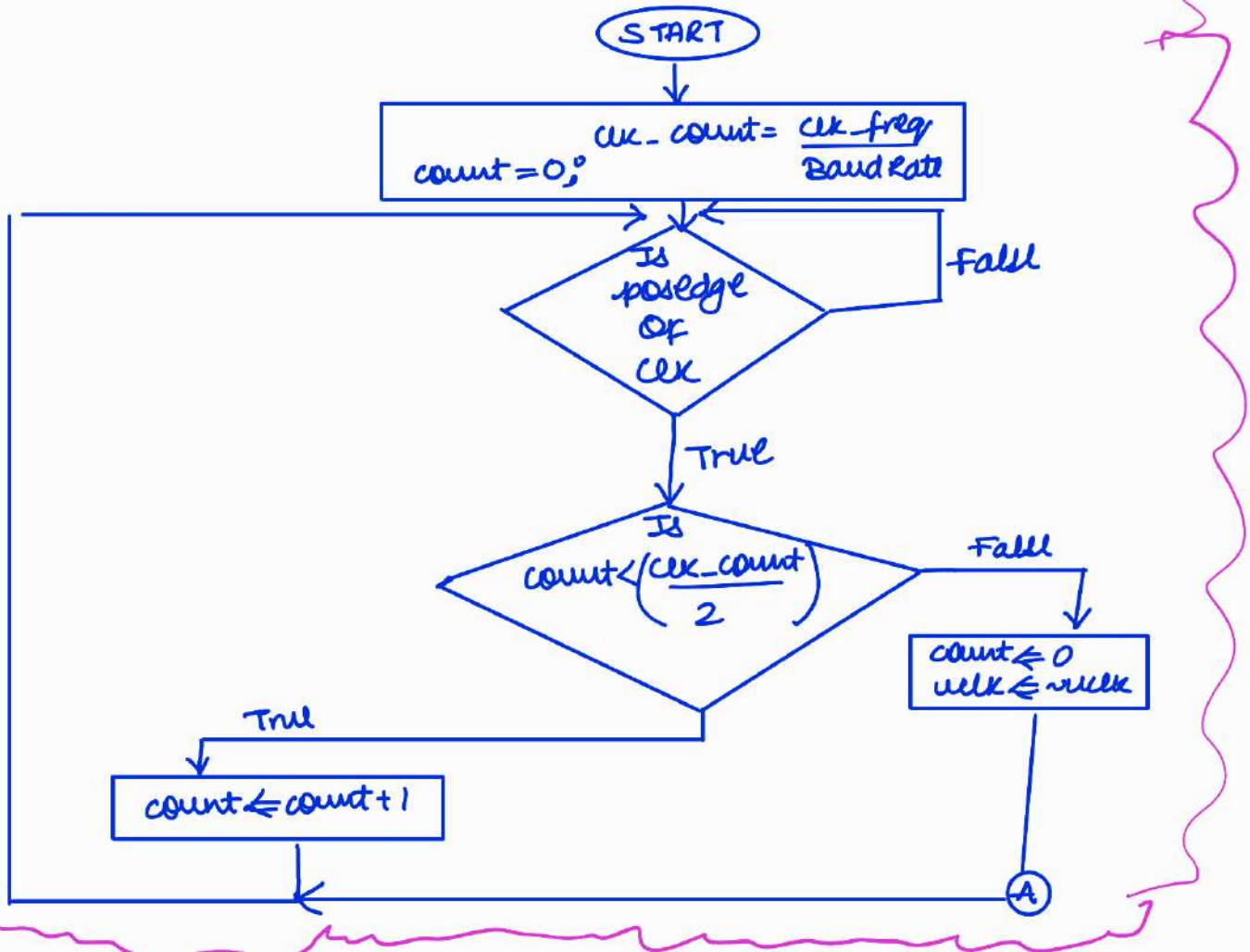
Baud Rate = 9600 ⇒ $\frac{1}{B.R} = 1.0416 \times 10^{-4}$

$= 0.104$ ms



Ratio = clk_count = $\frac{\text{clk\_freq (OR sys\_clk)}}{\text{Baud Rate}}$





→ CODE: (Transmitter)

```
Module uart_tx
#( parameter clk_freq = 1000000,
   parameter baud_rate = 9600
)
( input clk, rst,
  input newd,
  input [7:0] tx_data,
  output reg tx,
  output reg donetx
);
  localparam clkcount = (clk_freq/baud_rate);
  integer count = 0;
  integer counts = 0;
  reg uclk = 0; // Bit duration is represented by uclk.
enum bit [1:0] { idle = 2'b00, transfer = 2'b10} state;
```

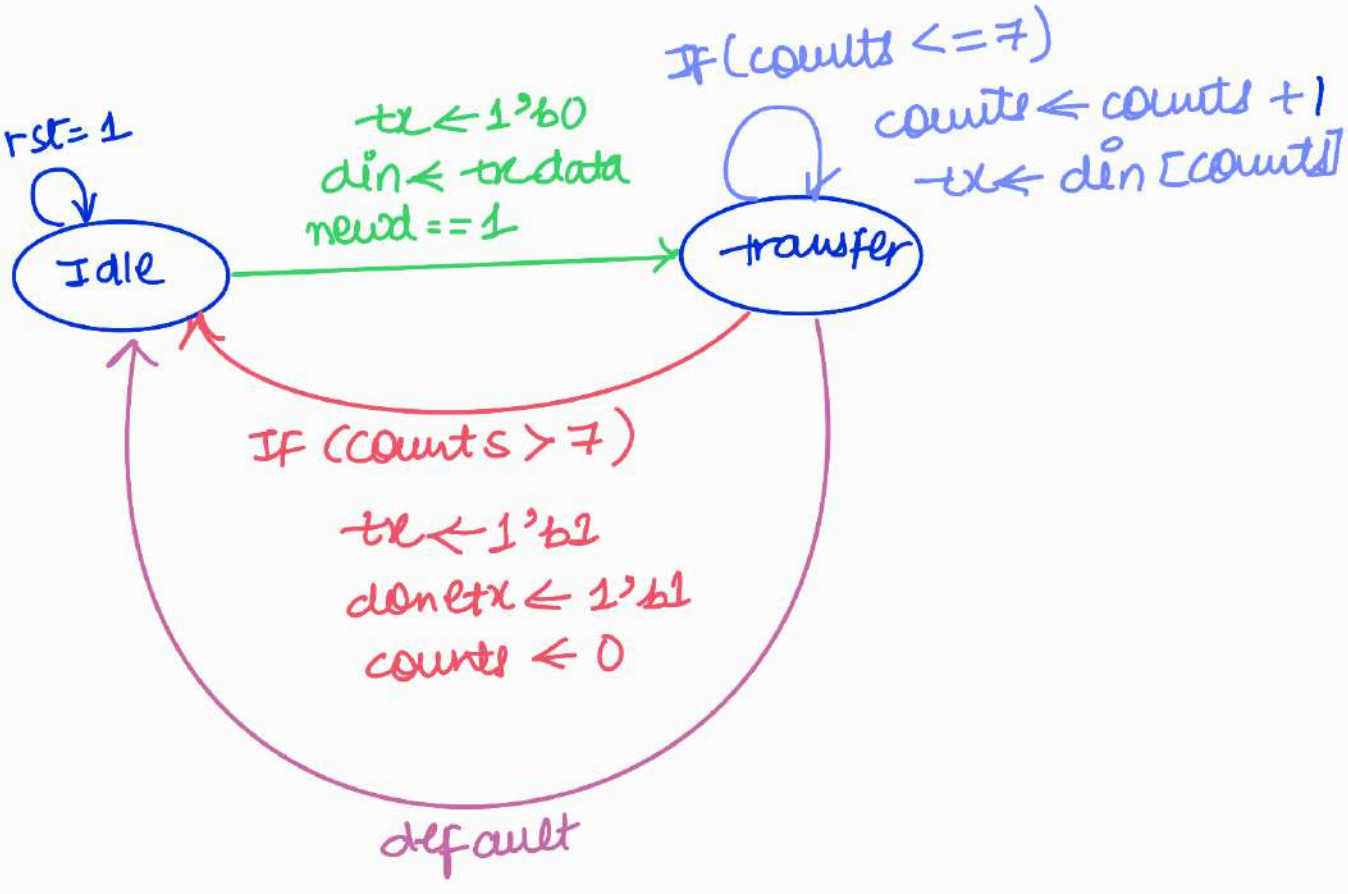
Can be Modified but are given default value

```
//// uart_clock_gen
    always @(posedge clk)
    begin
      if(count < clkcount/2)
        count <= count + 1;
      else begin
        count <= 0;
        uclk <= ~uclk;
      end
    end
////// Reset decoder
    always @ (posedge uclk)
```

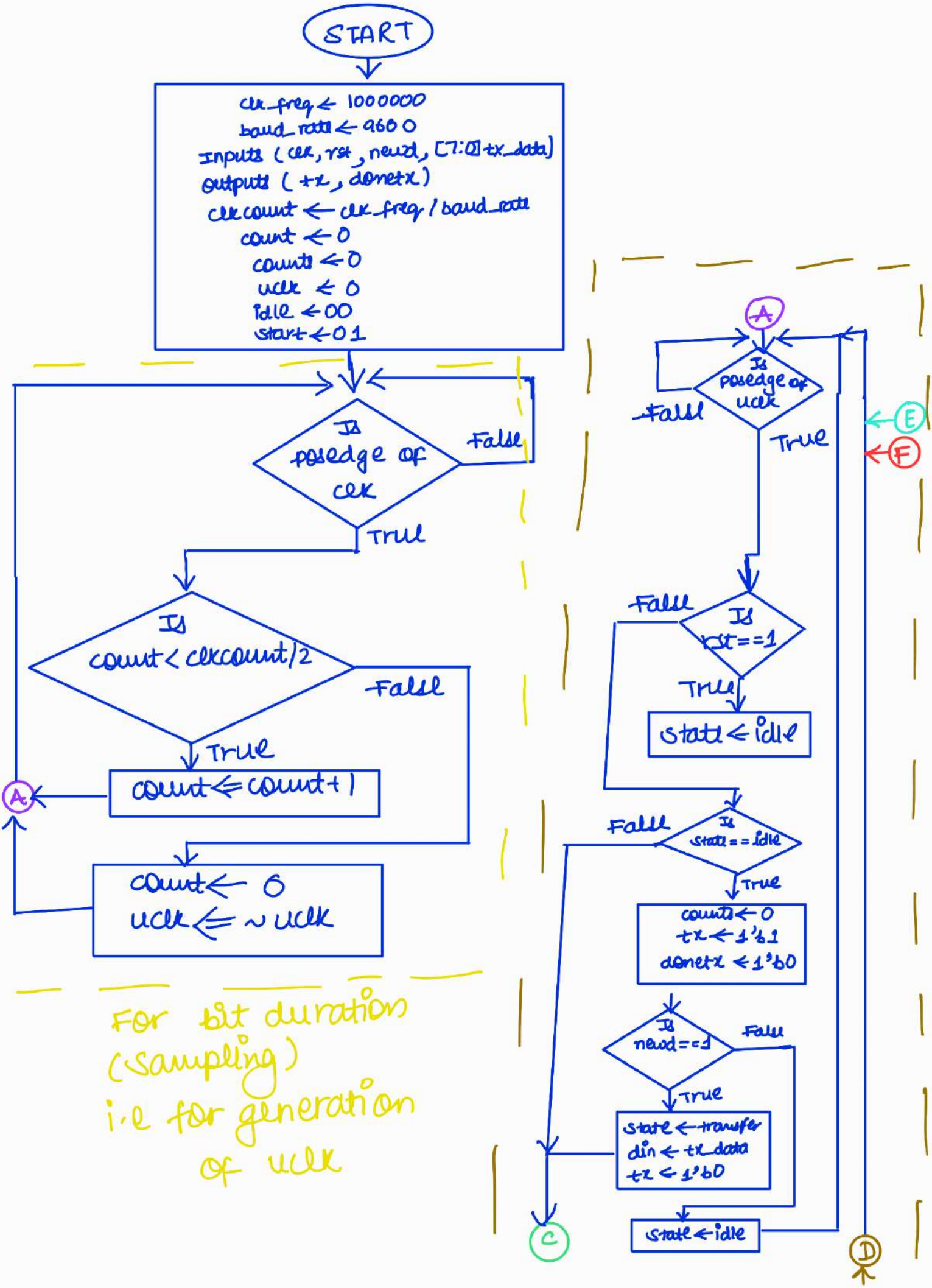
```
begin
 if (rst)
 begin
 state <= idle ;
 end
 else,
 begin
 case (state)
  idle :
    begin
    counts <= 0;
    tx <= 1'b1
    donetx <= 1'b0;
    if (newd)
     begin
     state <= transfer
     din <= tx_data; //sampling data
      tx <= 1'b0 ;  // start bit
      end
     else
      state <= idle;
     end

   transfer :
    if (counts <= 7) begin
     counts <= counts + 1;
     tx <= din [counts];
     state <= transfer;
     end
      else
      begin
       counts <= 0;
```

```
tx <= 1'b1;
state <= Pdle;
donetx <= 1'b1; // Indicates the
                // completion of data
                //transfer
.
end
end
default ; state <= idle
endcase
end
end
endmodule
```

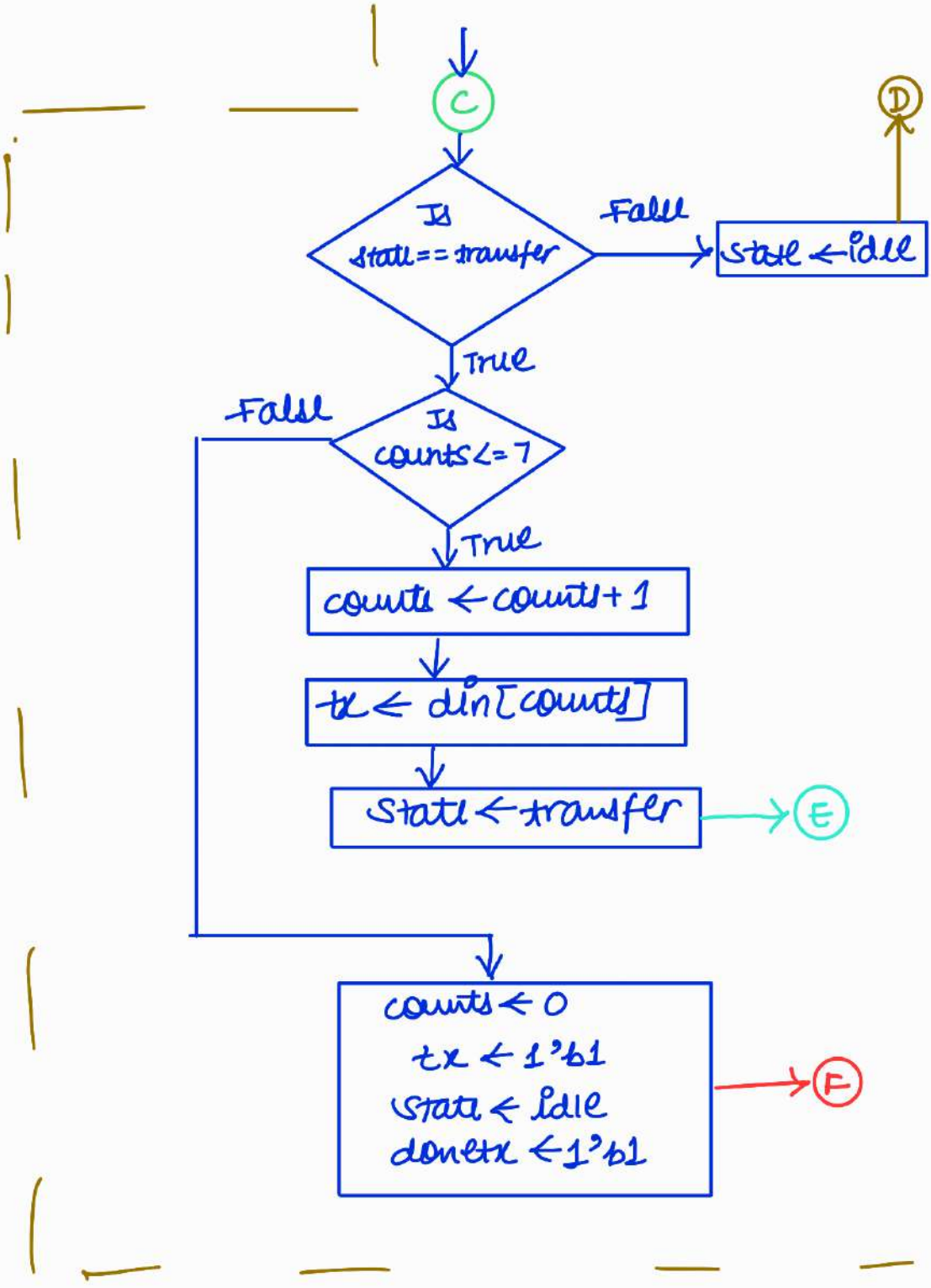
→ STATE DIAGRAM: [FOR TRANSFER MODULE] uart tx.



→ FLOWCHART FOR uarttx module: PROGRAM SPECIFIC



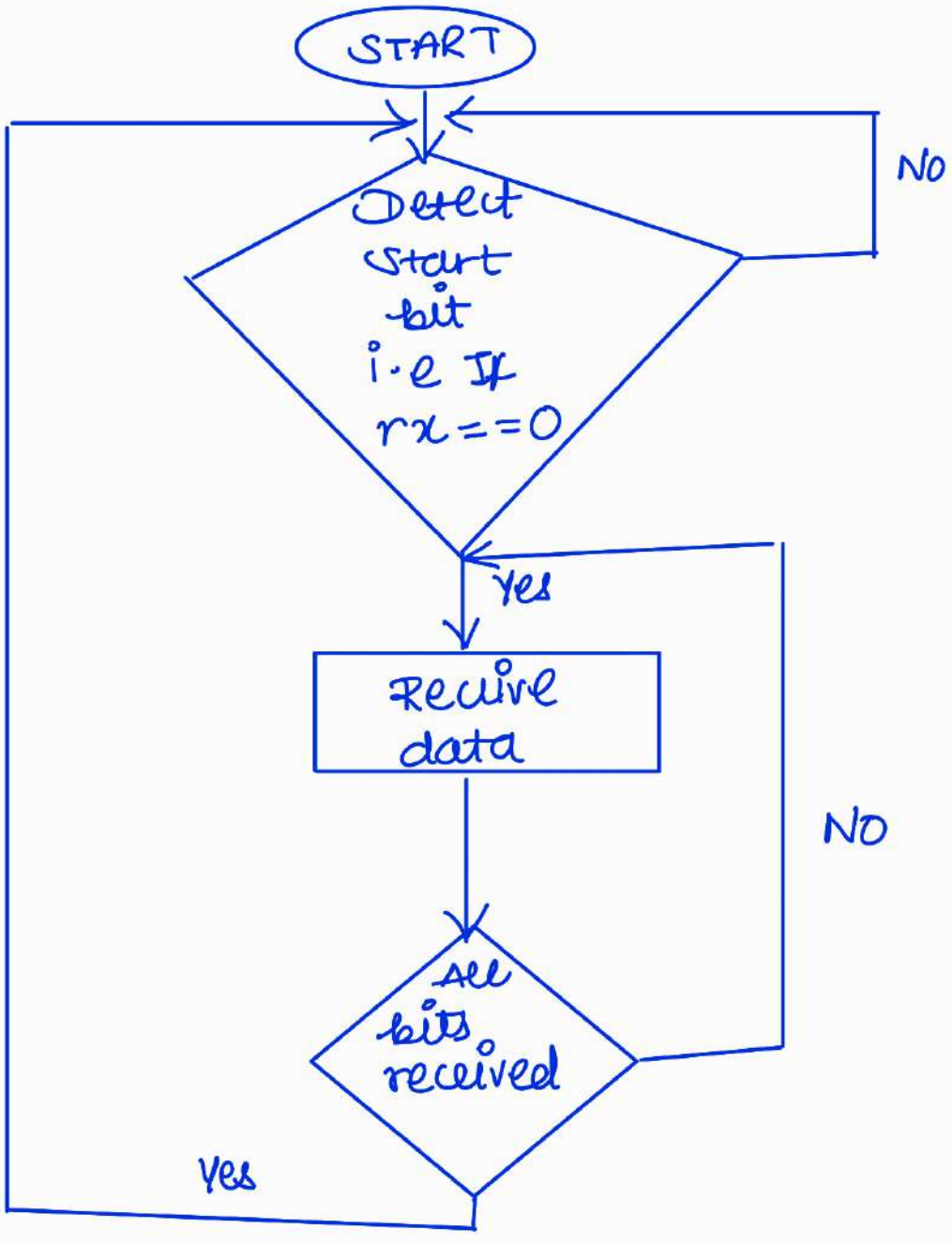
For bit duration (sampling) i.e for generation of uclk



For FSM Execution.

RECEPTION MODULE :

→ Flowchart for reception of Data: (GENERALIZED)



→ Receiver I/P signal: clk, rst, rx

global signals

→ receiver O/P signals: donerx, doutrx [7:0]

data will be received on rx pin & once data is being collected it will be sent on doutrx [7:0] & to mark completion of reception we have donerx.

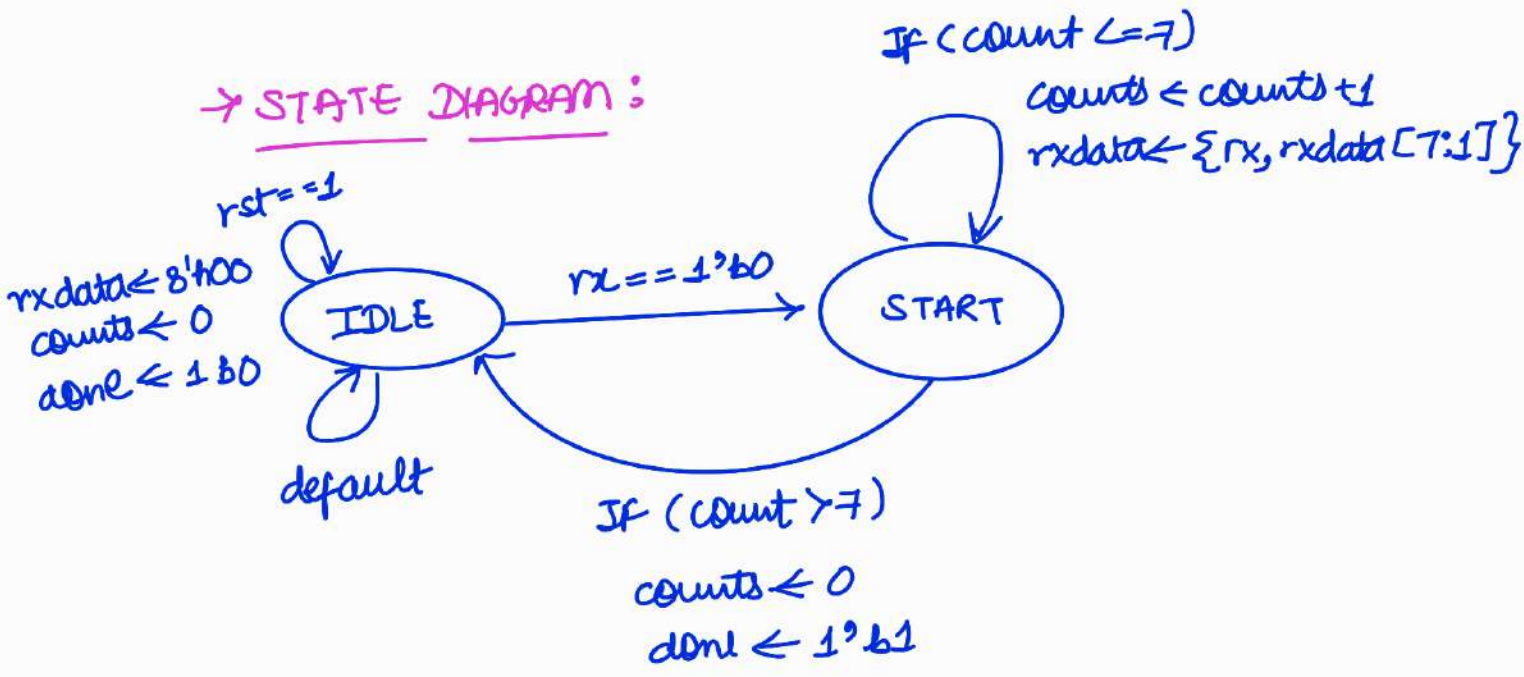
→ CODE FOR RECEIVER MODULE:

```
module uartrx
#(
  parameter clk_freq = 1000000, //MHZ
  parameter baud_rate = 9600
   )
( input clk,
  input rst,
  input rx,
  output reg done,
  output reg [7:0] rxdata
  );
 localparam clkcount = (clk_freq/baud_rate);
 integer count=0;
 integer counts = 0;
  reg uclk=0; // Bit duration is represented by uclk
enum bit[1:0] {idle = 2'b00, transfer = 2'b10} state;


//// uart_clock_gen
   always @(posedge clk)
   begin
    if(count < clkcount/2)
      count <= count +1;
    else begin
     count <=0;
     uclk <= ~uclk;

    end
```

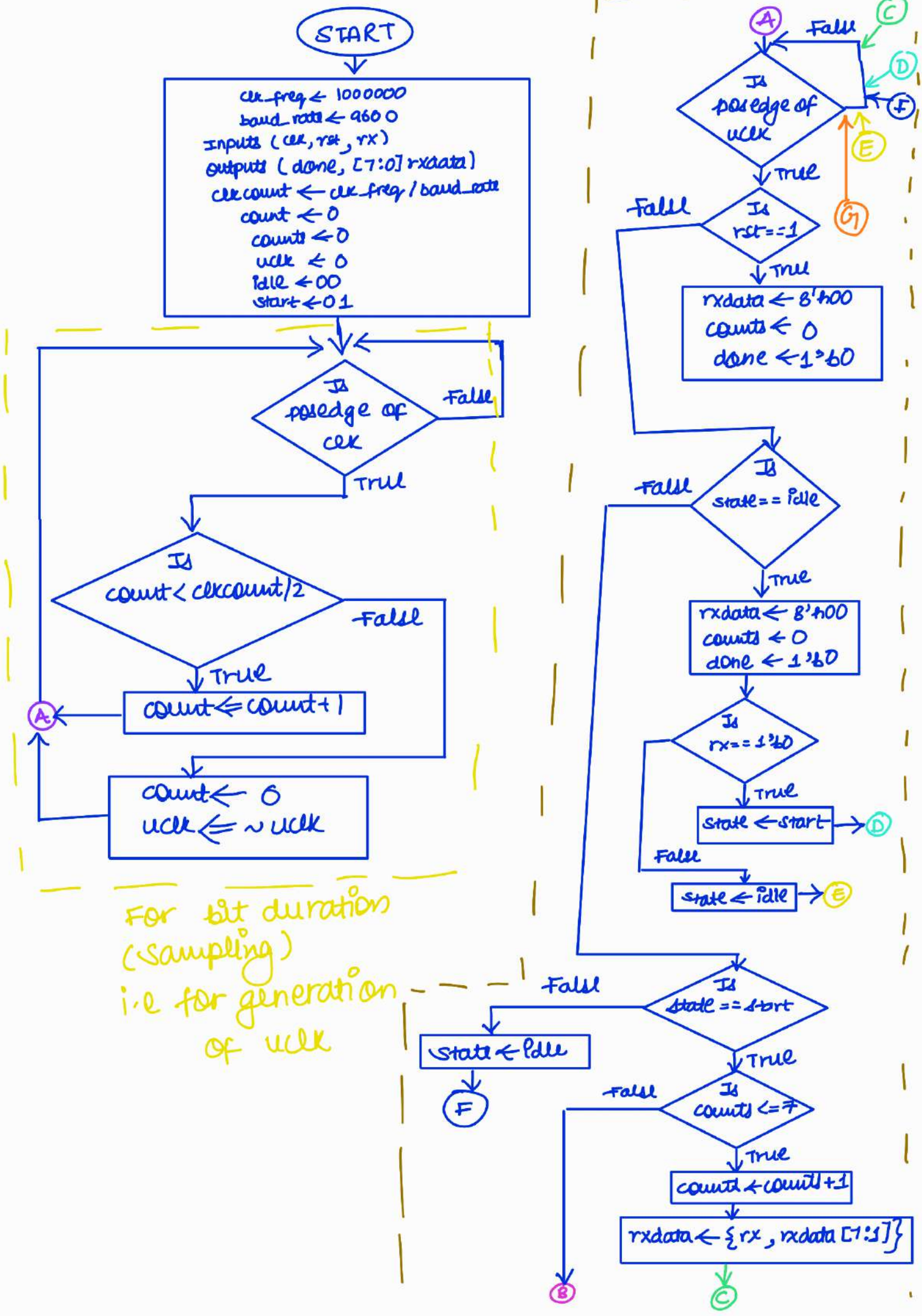
```
end
//FSM Execution
always @ (posedge clk)
begin
if (rst)
begin
rxdata <= 8'h00;
counts <= 0;
done <= 1'b0;
end
else
begin
case (state)
idle:
begin
rxdata <= 8'h00;
counts <=0;
done <= 1'b0;
if (rx == 1'b0) // start of transaction condition
state <= start;
else
state <= idle;
end

start:
begin
if (counts <=7)
begin
counts <= counts +1;
rxdata <= {rx, rxdata[7:1]} ;
  //Right shift register Implementation
end
else
begin
counts <= 0;
```

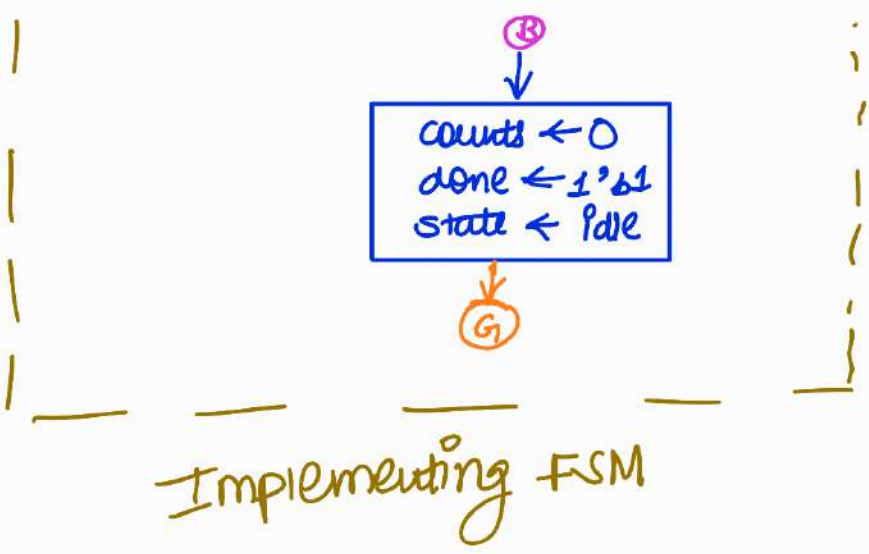
```
        done <= 1'b1;
        state <= idle;
      end
    end
    default : state <= idle;
  endcase
end
endmodule
```

→ STATE DIAGRAM :



→ FLOWchart for uartrx module : (PROGRAM SPECIFIC)


START
clk_freq ← 1000000
baud_rate ← 9600
inputs (clk, rst, rx)
outputs (done, [7:0] rxdata)
clkcount ← clk_freq / baud_rate
count ← 0
counts ← 0
uclk ← 0
idle ← 00
start ← 01
Is posedge of clk
Falsl
True
Is count < clkcount/2
Falsl
True
count ← count + 1
count ← 0
uclk ← ~uclk
A
For bit duration (sampling) i.e for generation of uclk
Is posedge of uclk
Falsl
True
Is rst == 1
Falsl
True
rxdata ← 8'h00
counts ← 0
done ← 1'b0
Is state == idle
Falsl
True
rxdata ← 8'h00
counts ← 0
done ← 1'b0
Is rx == 1'b0
True
state ← start
Falsl
state ← idle
Is state == start
Falsl
state ← idle
True
Is counts <= 7
Falsl
True
counts ← counts + 1
rxdata ← {rx, rxdata[7:1]}
B
C
D
E
F
G



Implementing FSM