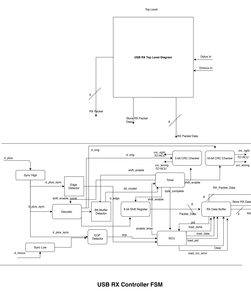
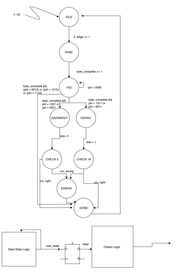
USB RX Functional diagram



USB RX Controller FSM



State	Control Signals	Output Signals Assertion
IDLE	d_edge	none
SYNC	byte_complete	none
PID	byte_complete, pid	load_pid
DATAPID	stop	load_data
DATAPIDFIN	stop	load_data
CHECK5	crc_correct, crc_wrong	timer
CHECK16	crc_correct, crc_wrong	load_crc_error
ERROR	none	load_crc_error
DONE	none	load_done

Control signals are what comes from the Decoder and what the next is going to be.

If for a state the control signal is not specified, the next state will simply move to the next state without the controls (i.e. d_rst is asserted).

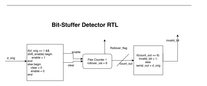
Next State Logic

```
always_comb
  case(state)
    IDLE: state = state;
    SYNC:
    PID:
      if(d_edge == 1) begin
        state = SYNC;
      end
      else begin
        next_state = IDLE;
      end
    DATAPID:
    DATAPIDFIN:
    CHECK5:
      if(byte_complete) begin
        if(pid == 8'b11) || (pid == 8'b10) || (pid == 8'b00) || (pid == 8'b01) || (pid == 8'b11) begin
          next_state = DATAPID;
        end
        else if(pid == 8'b10) || (pid == 8'b00) || (pid == 8'b01) begin
          next_state = DATAPIDFIN;
        end
        else begin
          next_state = PID;
        end
      end
      else begin
        next_state = PID;
      end
    CHECK16:
    ERROR:
      if(d_sync) begin
        state = CHECK16;
      end
      else begin
        next_state = DATAPID;
      end
    DONE:
      if(d_sync) begin
        state = CHECK16;
      end
      else begin
        next_state = DATAPID;
      end
    CHECK16:
      if(crc_error) begin
        next_state = ERROR;
      end
      else if(crc_ok) begin
        next_state = DONE;
      end
      else begin
        next_state = CHECK16;
      end
    ERROR:
      if(d_sync) begin
        next_state = DONE;
      end
      else begin
        next_state = IDLE;
      end
    DONE:
    default: state = IDLE;
  endcase
```

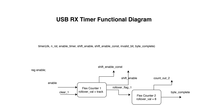
USB RX Timer Functional Diagram

Since the bit period is 8.33, the first two cycles are 8 while the third is 9 to average the bit.

If an invalid bit comes through, the timer has to be off for a cycle to fix the invalid bit. The shift_enable_const is the shift_enable, cause cannot clear

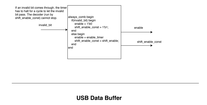


Bit-Stuffer Detector RTL



The general function is it looks at the output of the Decoder and if the value is one the flex counter counts up to 7.

USB RX Timer Functional Diagram



Since the bit period is 8.33, the first two cycles are 8 while the third is 9 to average the bit.

If an invalid bit comes through, the timer has to be off for a cycle to fix the invalid bit. The shift_enable_const is the shift_enable, cause cannot clear

USB Data Buffer

Logic:
Control signals (byte_done comes from the timer) 3 signifies a byte has fully shifted in (important especially with regards to the clear phase) and at least one the bits, n_packet, data valid
Clear signal is used to clear the buffer and is used when a packet has a error when things hold or just plain
Input done load_pid 2 control signal from the rx. signifies the incoming Packet_Data contains the pid and needs to be stored
Input done load_data 1 indicates to the data phase of
Input done load_crc error
Input done load_done
Output n_packet is high when a packet and then is used to operate the rest of the packet
Output stop_n_packet_data is used as the pin on what when the rest are enabled
Output n_packet_data is the output of the buffer, 8 bits, 1 byte, 8 bits, 1 byte, 8 bits, 1 byte, 8 bits, 1 byte
Only matters during the data phase or any given stream

```
logic
Control signals (byte_done comes from the timer) 3 signifies a byte has fully shifted in (important especially with regards
to the clear phase) and at least one the bits, n_packet, data valid
Clear signal is used to clear the buffer and is used when a packet has a error when things hold or just plain
Input done load_pid 2 control signal from the rx. signifies the incoming Packet_Data contains the pid and needs
to be stored
Input done load_data 1 indicates to the data phase of
Input done load_crc error
Input done load_done
Output n_packet is high when a packet and then is used to operate the rest of the packet
Output stop_n_packet_data is used as the pin on what when the rest are enabled
Output n_packet_data is the output of the buffer, 8 bits, 1 byte, 8 bits, 1 byte, 8 bits, 1 byte, 8 bits, 1 byte
Only matters during the data phase or any given stream

logic [7:0] n_packet_data;
logic [7:0] stop_n_packet_data;
logic [7:0] Packet_Data[7:0];
logic [2:0] cycle;
logic [2:0] cycle_prev;
always_comb begin
  if(load_crc_error) begin
    n_packet = 1'b0;
    stop_n_packet_data = 1'b0;
    n_packet_data = 1'b0;
  end
  else if(load_pid || load_data || load_done || n_packet || clear) begin
    n_packet = 1'b1;
    stop_n_packet_data = 1'b1;
    n_packet_data = 1'b1;
  end
  else begin
    n_packet = 1'b0;
    stop_n_packet_data = 1'b0;
    n_packet_data = 1'b0;
  end
end
always_comb
  begin : SET_PACKET_LOGIC
    if(n_packet || stop_n_packet_data) begin
      Packet_Data[0] = n_packet_data;
      Packet_Data[1] = n_packet_data;
      Packet_Data[2] = n_packet_data;
      Packet_Data[3] = n_packet_data;
      Packet_Data[4] = n_packet_data;
      Packet_Data[5] = n_packet_data;
      Packet_Data[6] = n_packet_data;
      Packet_Data[7] = n_packet_data;
    end
    else if(load_pid || load_data || load_done || n_packet || clear) begin
      Packet_Data[0] = 1'b0;
      Packet_Data[1] = 1'b0;
      Packet_Data[2] = 1'b0;
      Packet_Data[3] = 1'b0;
      Packet_Data[4] = 1'b0;
      Packet_Data[5] = 1'b0;
      Packet_Data[6] = 1'b0;
      Packet_Data[7] = 1'b0;
    end
    else if(cycle == 0) begin
      Packet_Data[0] = n_packet_data;
    end
    else if(cycle == 1) begin
      Packet_Data[1] = n_packet_data;
    end
    else if(cycle == 2) begin
      Packet_Data[2] = n_packet_data;
    end
    else if(cycle == 3) begin
      Packet_Data[3] = n_packet_data;
    end
    else if(cycle == 4) begin
      Packet_Data[4] = n_packet_data;
    end
    else if(cycle == 5) begin
      Packet_Data[5] = n_packet_data;
    end
    else if(cycle == 6) begin
      Packet_Data[6] = n_packet_data;
    end
    else if(cycle == 7) begin
      Packet_Data[7] = n_packet_data;
    end
    else begin
    end
  end
```

USB RX EOP Detector

States	Control Signals	Output Signals
IDLE	d_pkt_sync, d_rxrxus_sync, shift_enable	none
START	d_pkt_sync, d_rxrxus_sync, shift_enable	none
SE0	d_pkt_sync, d_rxrxus_sync, shift_enable	none
FINISH	none	eop

Output Logic

Next State Logic