ECE 571
CAN Protocol Controller: Design & Verification

By

Vijay E Arputharaj

Akshay S Kotian

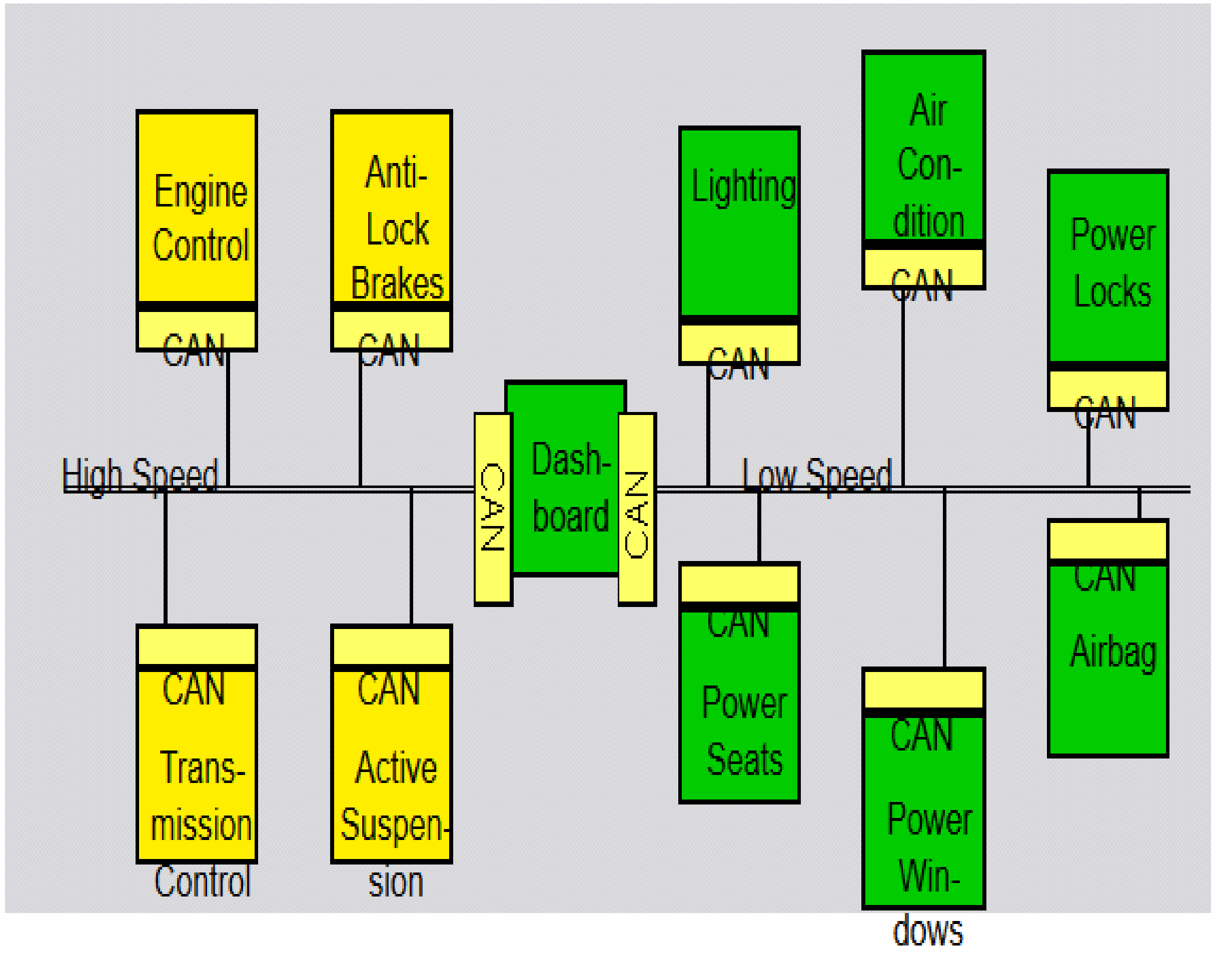
INTRODUCTION

CAN Bus

- CAN(Controller Area Network) vehicle bus standard allows devices to communicate without host computer
- Typical Automobile has around 70 ECU’s so communication requires robust but simple protocol
- CAN, developed and patented by Bosch, fills this need

Components

- CPU: Host processor decides what messages to send & receive
- Controller: Protocol Implementation
  - Packets, Serializes and transmits on bus
  - Stores received bits and hands it over to CPU
- Transceiver: Physical layer signaling



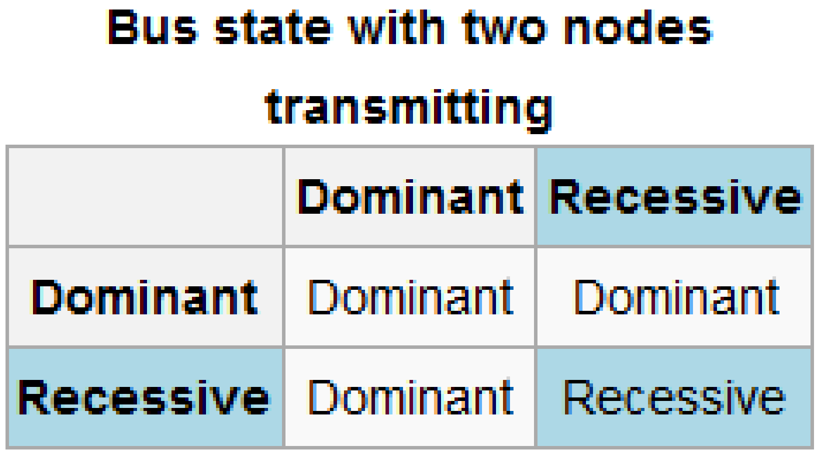
Features

- CAN is multi-master broadcast serial standard, no limitations on number of devices
- Nodes have no specific address and message identifiers are content specific
- CRC check for Error Handling, NRZ bit code with stuffing for synchronization
- CSMA/BA(Bitwise Arbitration), prioritized arbitration with no loss in time

BASIC CONCEPTS

Bus Characteristics

- Two Bus states
  - "1" = Recessive
  - "0" = Dominant



Bus state with two nodes transmitting

	Dominant	Recessive
Dominant	Dominant	Dominant
Recessive	Dominant	Recessive

- Bus logic is 'Wired-AND' i.e. Dominant bits overwrite Recessive bits

Bus Access and Arbitration

- Arbitration is non-destructive and prioritized by message
- Bitwise Arbitration scheme

Bits	A[3]	A[2]	A[1]	A[0]
Node 1				
Node 2				
Node 3				

Bus Access and Arbitration

- Arbitration is non-destructive and prioritized by message
- Bitwise Arbitration scheme

Bits	A[3]	A[2]	A[1]	A[0]
Node 1	1			
Node 2	1			
Node 3	1			

Bus Access and Arbitration

- Arbitration is non-destructive and prioritized by message
- Bitwise Arbitration scheme

Bits	A[3]	A[2]	A[1]	A[0]
Node 1	1	0		
Node 2	1	0		
Node 3	1	1		

Bus Access and Arbitration

- Arbitration is non-destructive and prioritized by message
- Bitwise Arbitration scheme

Bits	A[3]	A[2]	A[1]	A[0]
Node 1	1	0	0	
Node 2	1	0	1	
Node 3	1	1	S	

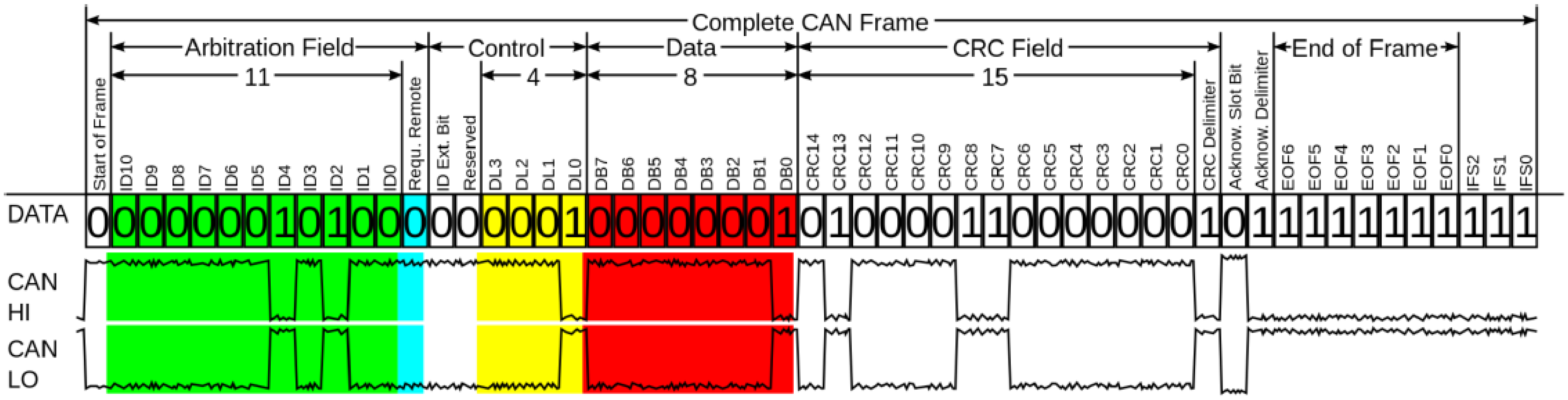
Bus Access and Arbitration

- Arbitration is non-destructive and prioritized by message
- Bitwise Arbitration scheme

Bits	A[3]	A[2]	A[1]	A[0]	
Node 1	1	0	0	0	Bus Master!
Node 2	1	0	1	S	
Node 3	1	1	S	S	

Frame Formats

- Data Frame



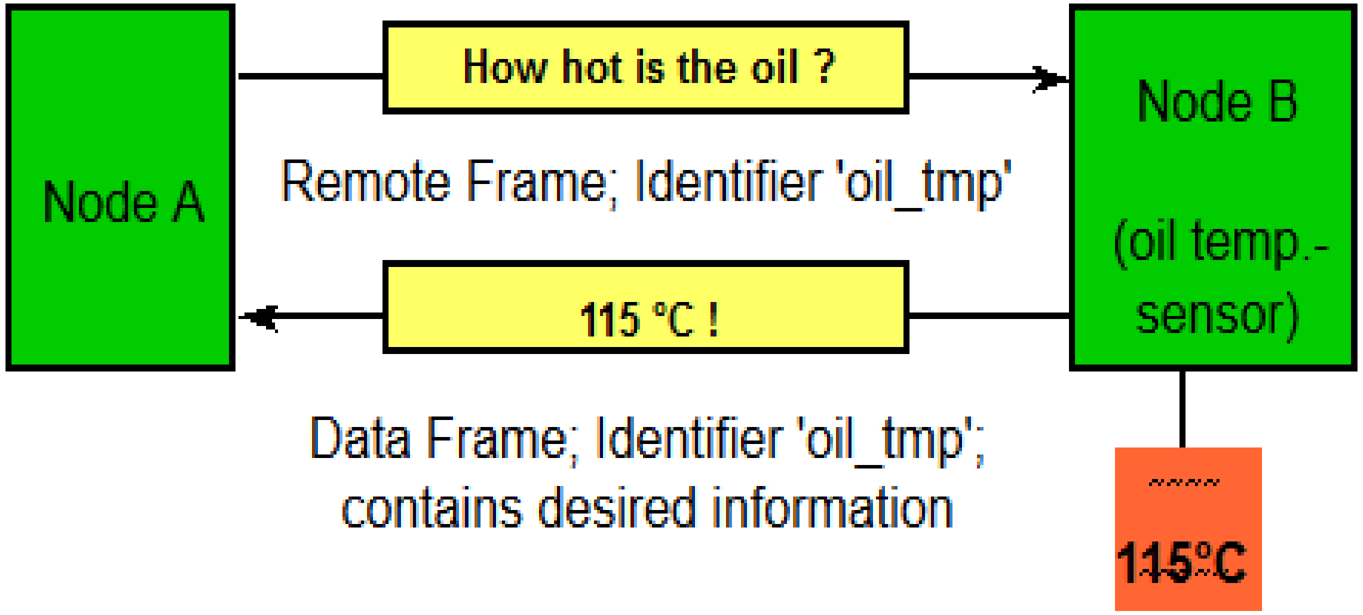
Frame Formats

- Request Frame

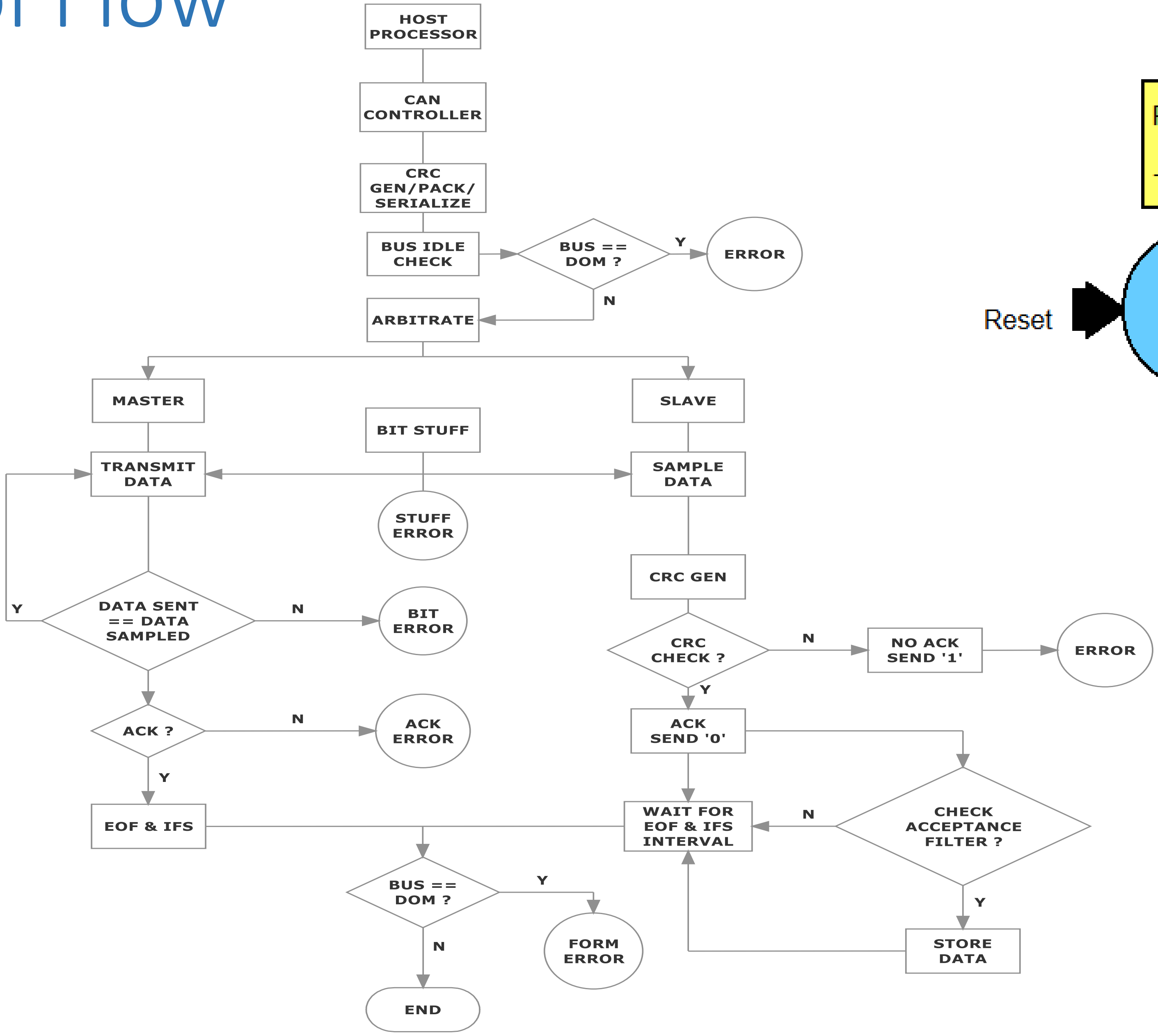
  - Similar to Data frame format but no Data field

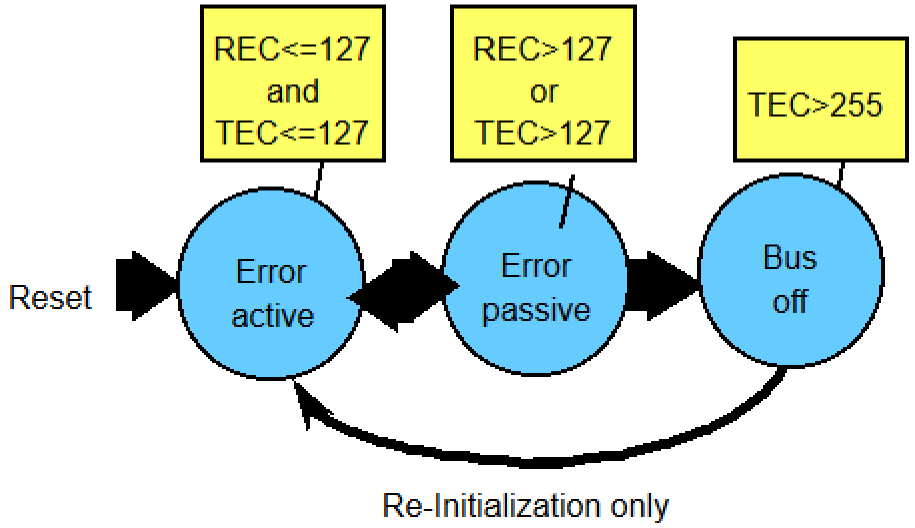
- Error Frame

  - Fixed Frame format sent by all nodes on bus error


Node A
How hot is the oil ?
Remote Frame; Identifier 'oil_tmp'
115 °C !
Data Frame; Identifier 'oil_tmp'; contains desired information
Node B
(oil temp.-sensor)
115°C

Control Flow



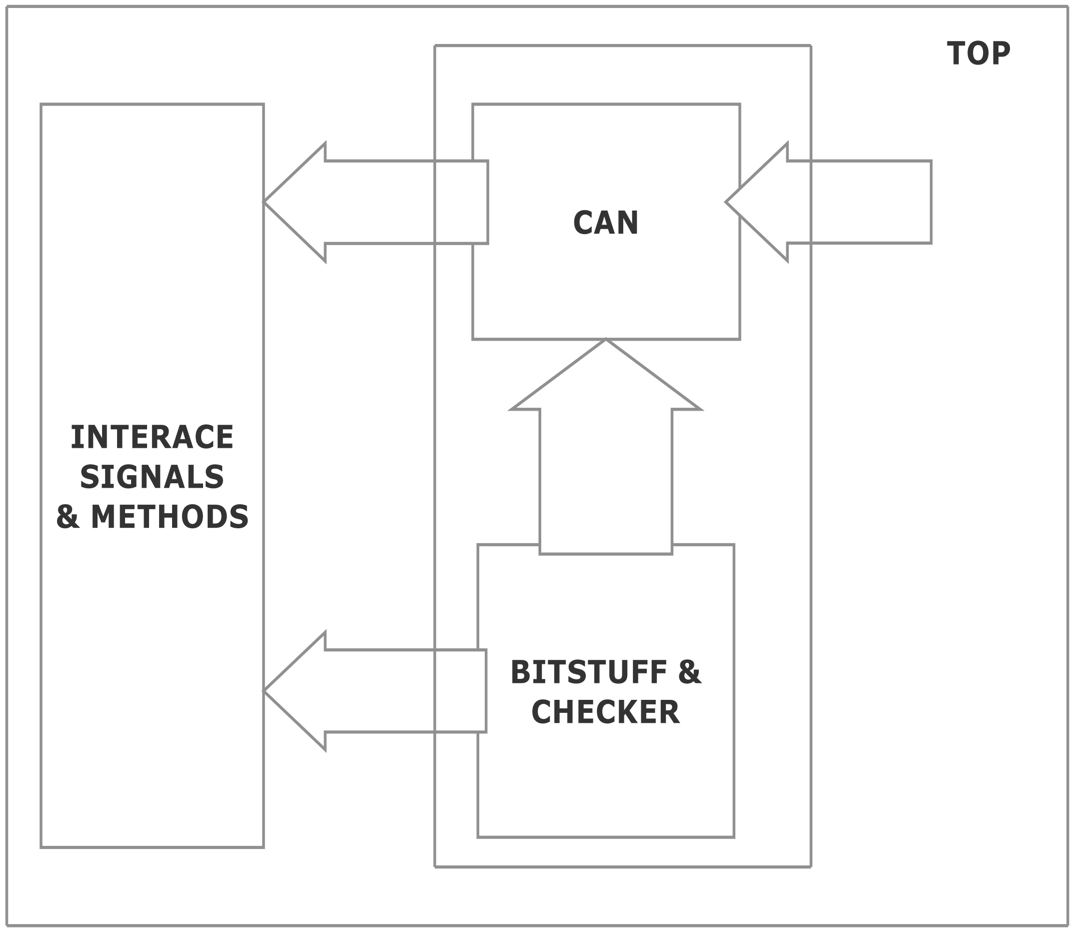


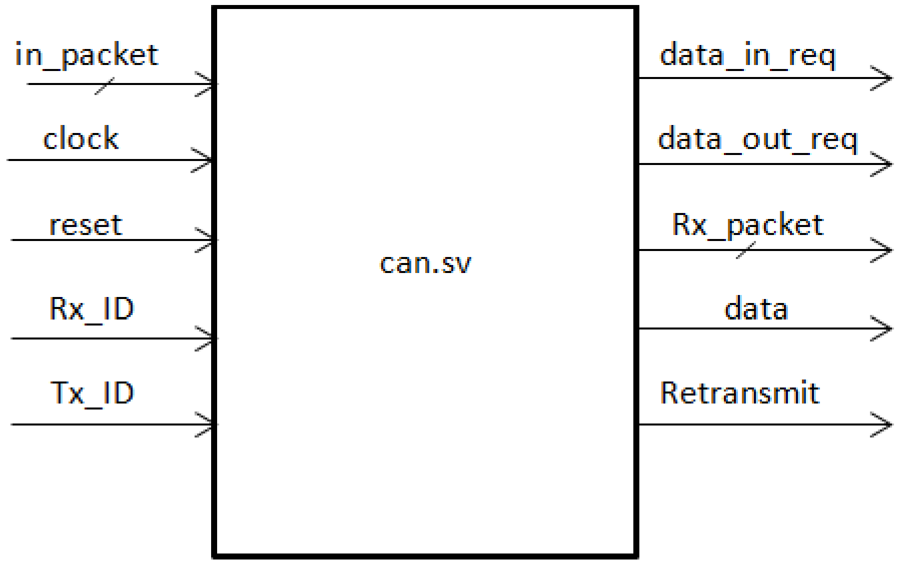
IMPLEMENTATION

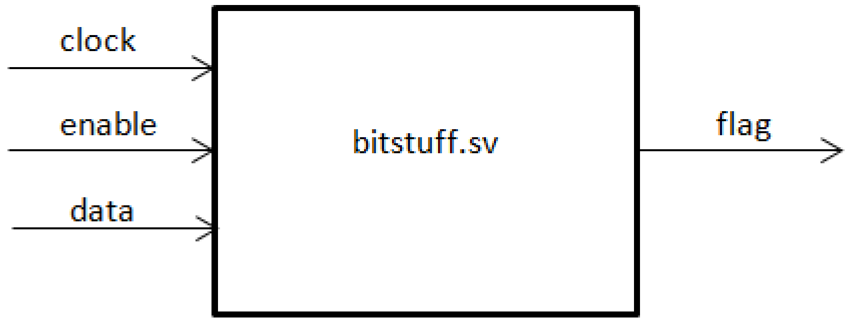
Objective

- Synthesizable Behavioral model of CAN Protocol Controller
- Use System Verilog constructs for functional verification
- Perform Veloce emulation in TBX mode
- Studied Bosch’s CAN specification and designed from scratch , no existing code used

Design Blocks



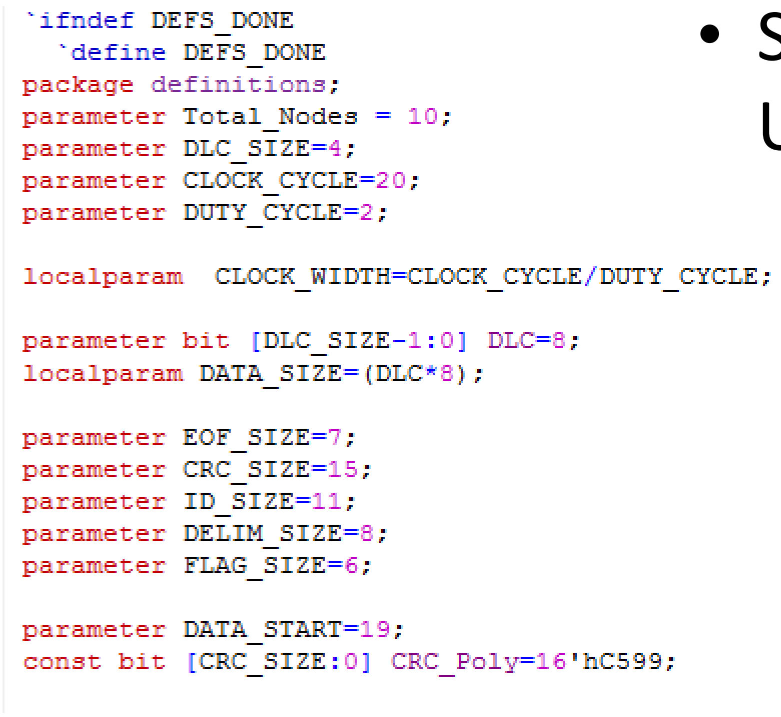




DESIGN CONSTRUCTS

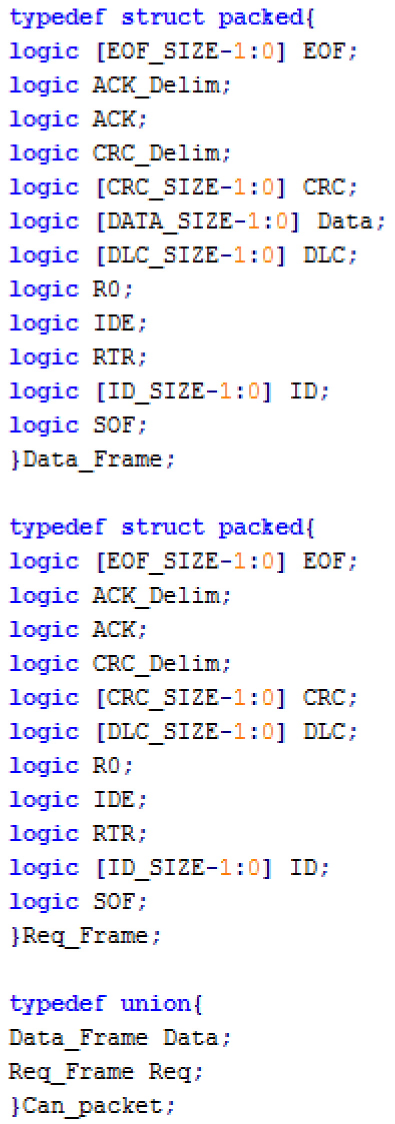
Packages

- Parameters



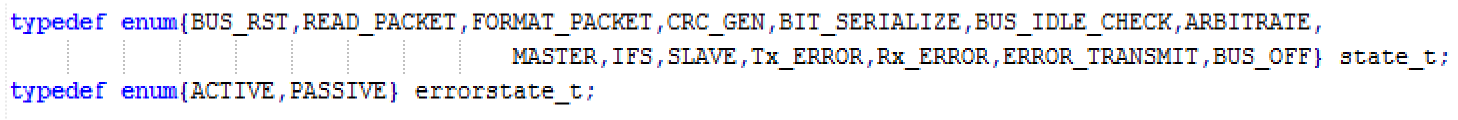
```
`ifndef DEFS_DONE
  `define DEFS_DONE
package definitions;
parameter Total_Nodes = 10;
parameter DLC_SIZE=4;
parameter CLOCK_CYCLE=20;
parameter DUTY_CYCLE=2;

localparam  CLOCK_WIDTH=CLOCK_CYCLE/DUTY_CYCLE;

parameter bit [DLC_SIZE-1:0] DLC=8;
localparam DATA_SIZE=(DLC*8);

parameter EOF_SIZE=7;
parameter CRC_SIZE=15;
parameter ID_SIZE=11;
parameter DELIM_SIZE=8;
parameter FLAG_SIZE=6;

parameter DATA_START=19;
const bit [CRC_SIZE:0] CRC_Poly=16'hC599;
```

- Structures & Unions



```
typedef struct packed{
logic [EOF_SIZE-1:0] EOF;
logic ACK_Delim;
logic ACK;
logic CRC_Delim;
logic [CRC_SIZE-1:0] CRC;
logic [DATA_SIZE-1:0] Data;
logic [DLC_SIZE-1:0] DLC;
logic R0;
logic IDE;
logic RTR;
logic [ID_SIZE-1:0] ID;
logic SOF;
}Data_Frame;

typedef struct packed{
logic [EOF_SIZE-1:0] EOF;
logic ACK_Delim;
logic ACK;
logic CRC_Delim;
logic [CRC_SIZE-1:0] CRC;
logic [DLC_SIZE-1:0] DLC;
logic R0;
logic IDE;
logic RTR;
logic [ID_SIZE-1:0] ID;
logic SOF;
}Req_Frame;

typedef union{
Data_Frame Data;
Req_Frame Req;
}Can_packet;
```

- Enums



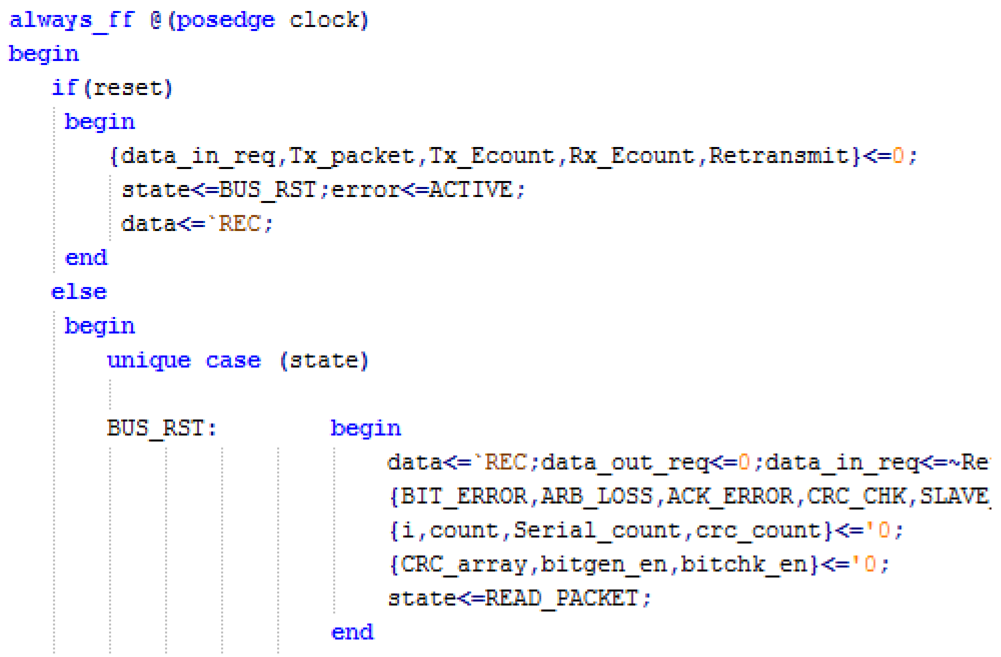
```
typedef enum{BUS_RST,READ_PACKET,FORMAT_PACKET,CRC_GEN,BIT_SERIALIZE,BUS_IDLE_CHECK,ARBITRATE,
                        MASTER,IFS,SLAVE,Tx_ERROR,Rx_ERROR,ERROR_TRANSMIT,BUS_OFF} state_t;
typedef enum{ACTIVE,PASSIVE} errorstate_t;
```

Interface

```
//
//DESCRIPTION: CAN Bus Interface. Has a single wire-AND line
// 0 is Dominant state and 1 is recessive state on bus

`include "def.pkg"
interface can_bus(input [Total_Nodes-1:0] input_data);

wire data = (1'b1 & (&input_data));

//Error handle function for CAN node
function automatic void Error_Handle(const ref errorstate_t ERROR,ref int error_count,ref Error_Frame Error_packet);
error_count=error_count+8;
Error_packet.Delim='1;
if(ERROR==ACTIVE) Error_packet.Flag='0;
if(ERROR==PASSIVE)   Error_packet.Flag='1;
endfunction

//CRC generator - code re-used from Bosch's CAN specification
function automatic void CRC_gen(const ref logic [DATA_CRC_LEN:0] CRC_array,ref int CRC_Len,output logic [CRC_SIZE-1:0] CRC_RG);
  automatic logic CRCNXT=0;
  parameter N_CRC=15;
  CRC_Len=`RTR_BIT?REQ_CRC_LEN:DATA_CRC_LEN;
  for(int i=0;i<=CRC_Len;i++)
  begin
  CRCNXT=CRC_array[i] ^ CRC_RG[N_CRC-1];
  CRC_RG<<=1;
  if(CRCNXT)
  CRC_RG=CRC_RG ^ CRC_Poly;
  end
  CRC_RG={<<{CRC_RG}};
endfunction
endinterface
```

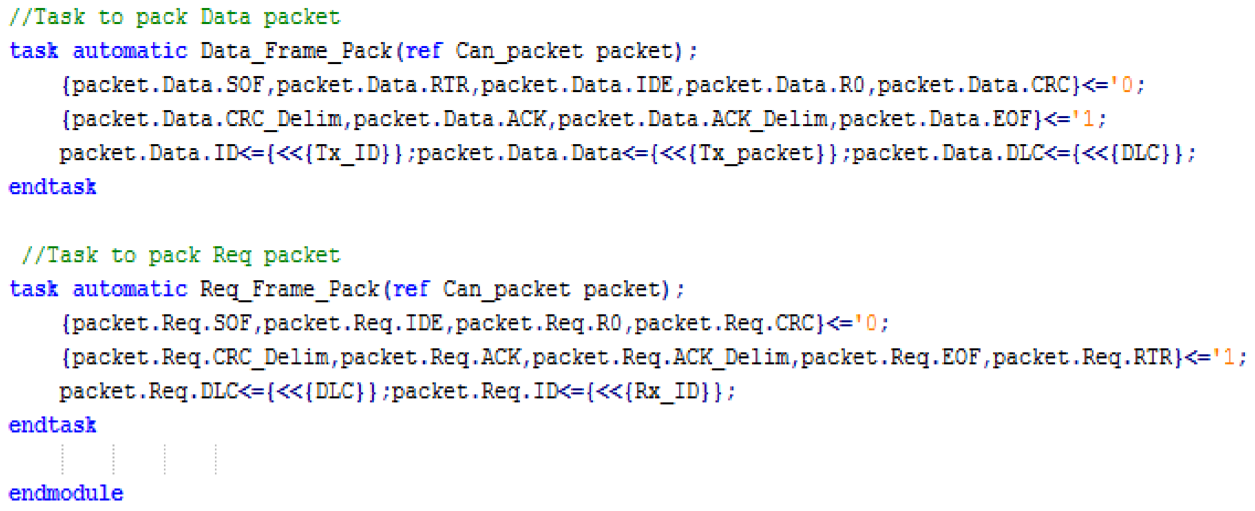
Procedural blocks

- always_ff & unique



```
always_ff @(posedge clock)
begin
   if(reset)
    begin
       {data_in_req,Tx_packet,Tx_Ecount,Rx_Ecount,Retransmit}<=0;
        state<=BUS_RST;error<=ACTIVE;
        data<=`REC;
    end
   else
    begin
       unique case (state)

       BUS_RST:         begin
                            data<=`REC;data_out_req<=0;data_in_req<=~Re
                            {BIT_ERROR,ARB_LOSS,ACK_ERROR,CRC_CHK,SLAVE_
                            {i,count,Serial_count,crc_count}<='0;
                            {CRC_array,bitgen_en,bitchk_en}<='0;
                            state<=READ_PACKET;
                        end
```

- Tasks



```
//Task to pack Data packet
task automatic Data_Frame_Pack(ref Can_packet packet);
    {packet.Data.SOF,packet.Data.RTR,packet.Data.IDE,packet.Data.R0,packet.Data.CRC}<='0;
    {packet.Data.CRC_Delim,packet.Data.ACK,packet.Data.ACK_Delim,packet.Data.EOF}<='1;
    packet.Data.ID<={<<{Tx_ID}};packet.Data.Data<={<<{Tx_packet}};packet.Data.DLC<={<<{DLC}};
endtask

 //Task to pack Req packet
task automatic Req_Frame_Pack(ref Can_packet packet);
    {packet.Req.SOF,packet.Req.IDE,packet.Req.R0,packet.Req.CRC}<='0;
    {packet.Req.CRC_Delim,packet.Req.ACK,packet.Req.ACK_Delim,packet.Req.EOF,packet.Req.RTR}<='1;
    packet.Req.DLC<={<<{DLC}};packet.Req.ID<={<<{Rx_ID}};
endtask

endmodule
```

Others….

- .* operator
- 2 state & logic types
- Queues & Associative arrays for debugging & Scoreboard
- Foreach loops
- Final blocks
- timeunit & timeprecision

Design Features

- Used generate blocks to allow multiple node instantiations
- All CAN specifications such as Field widths are parameterized for code reusability
- Flags for Debug, Assertion, Error inject & Veloce modes

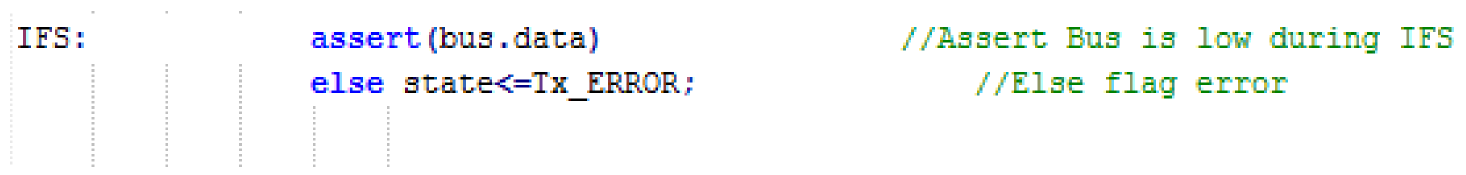
VERIFICATION

Assertions

- Employed both immediate & concurrent assertions
- Helped us immensely to localize bugs at the early stages of design as each transaction took multiple cycles, looking at waveforms or $monitor output is tedious
- In the later part, developed separate assertion module which runs in parallel
- Ensure nodes adhered to protocol throughout simulation and also prints out a scoreboard at end

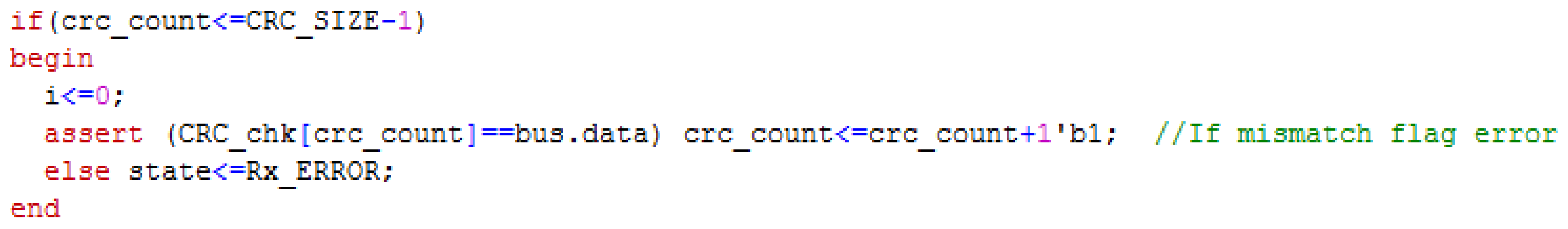
Immediate Assertions

- IFS Check:



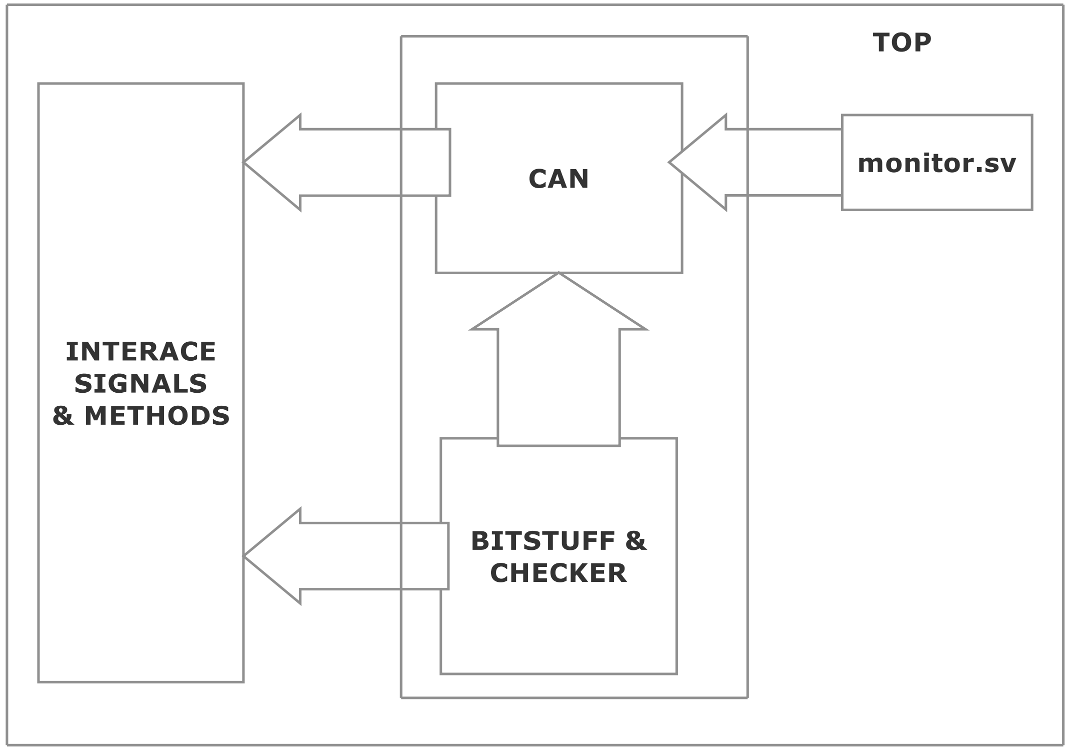
```
IFS:            assert(bus.data)                //Assert Bus is low during IFS
                else state<=Tx_ERROR;              //Else flag error
```

- CRC Check:



```
if(crc_count<=CRC_SIZE-1)
begin
  i<=0;
  assert (CRC_chk[crc_count]==bus.data) crc_count<=crc_count+1'b1;  //If mismatch flag error
  else state<=Rx_ERROR;
end
```

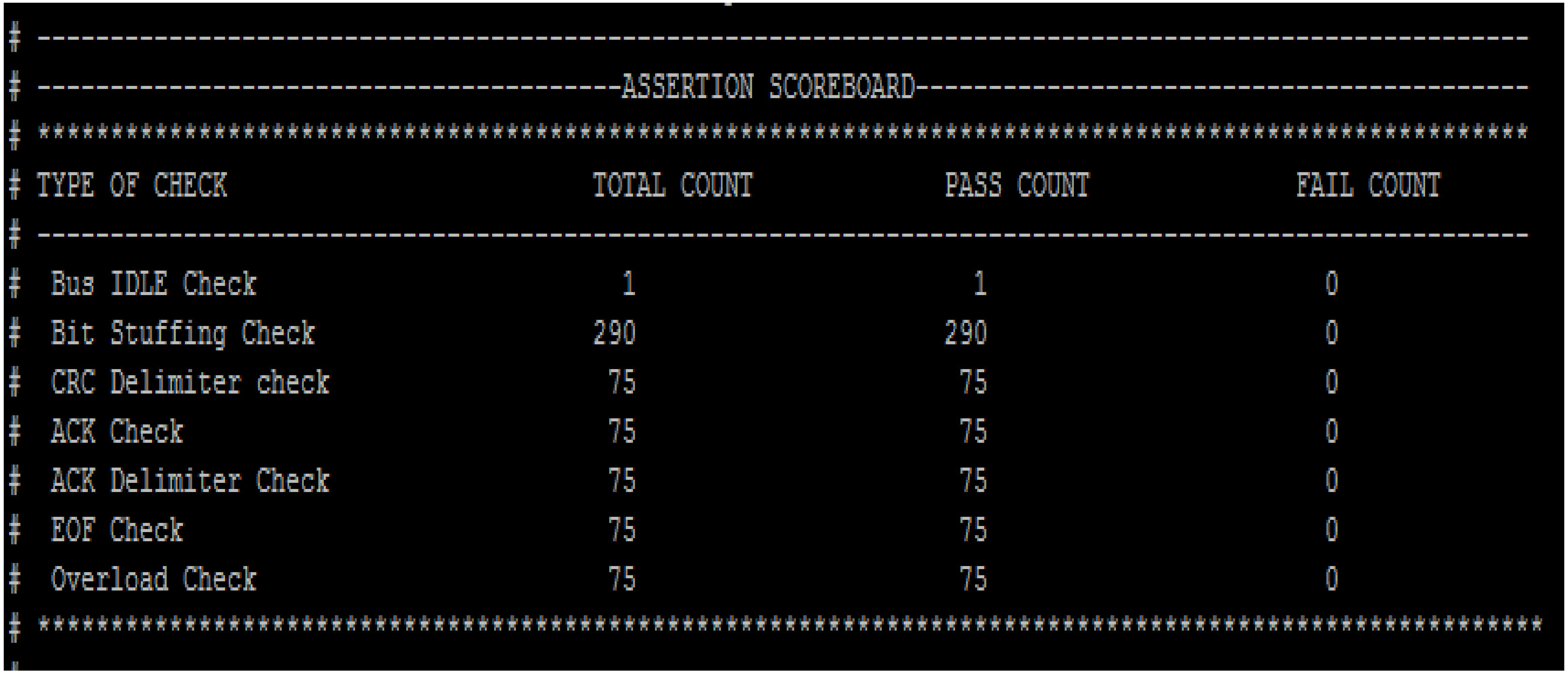
Concurrent Assertions



```
 property bus_idle_check;
@(posedge clock) $rose(reset)|=> bus.data;
endproperty

property  bit_stuffing_check_high;
@(posedge clock) disable iff (!BIT_CHK)
(bus.data[*5] |=>  !bus.data)   ;
endproperty

property  bit_stuffing_check_low;
@(posedge clock) disable iff (!BIT_CHK)
(!bus.data[*5] |=> bus.data)   ;
endproperty

property EOF_check;                        // End of Frame check
@(posedge clock) disable iff (reset)
( $rose(ACK) |-> ##2 bus.data[*7])   ;
endproperty

property ACK_check;
@(posedge clock) disable iff (reset)
( ACK |=> !bus.data)     ;
endproperty
```

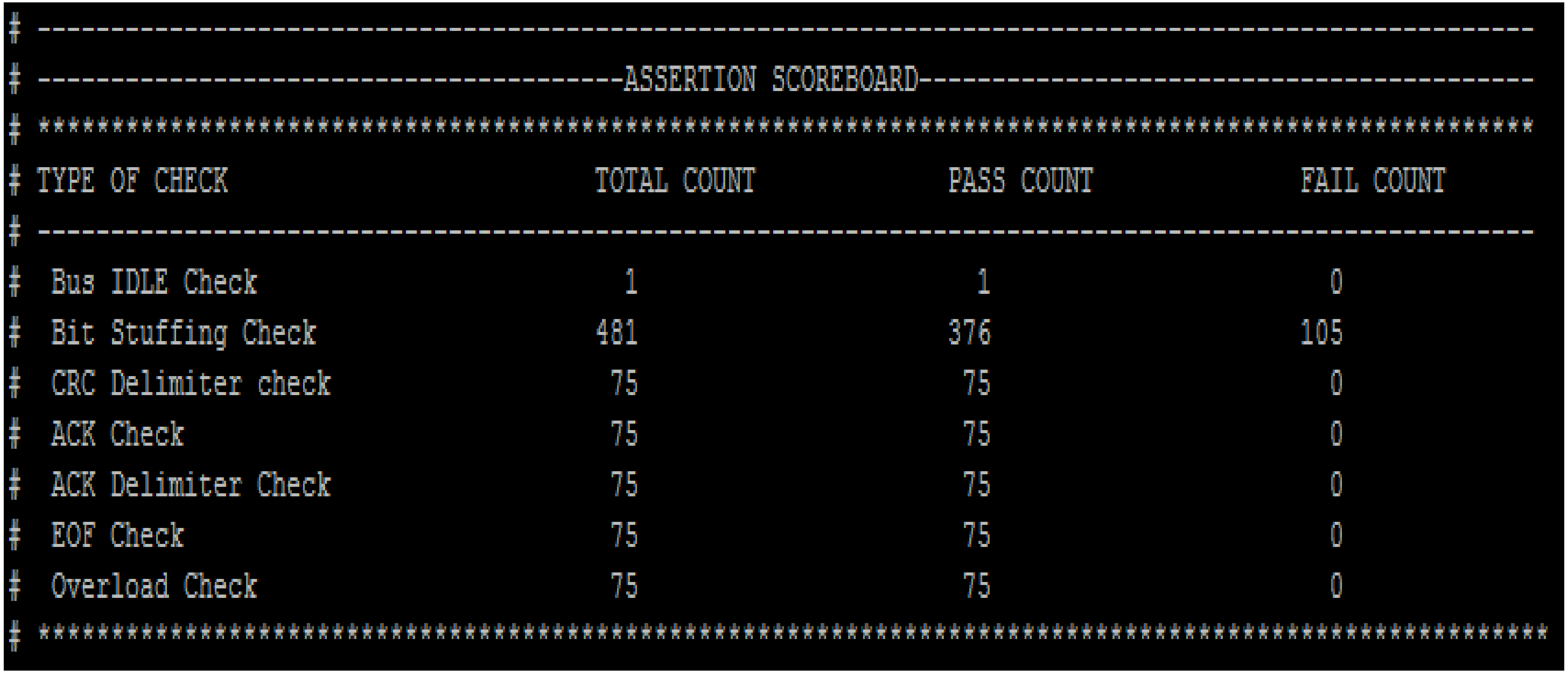
```
property ACK_delimiter_check;
@(posedge clock) disable iff (reset)
( ACK |=> ##1 bus.data) ;
endproperty


property CRC_delimiter_check;
@(posedge clock) disable iff (reset)
( ACK |-> bus.data) ;              // Read the value on the data bus
endproperty                                   //at the same clk edge

property Overload_check;
@(posedge clock) disable iff (reset)
( $rose(ACK) |-> ##9 bus.data[*3])  ;   //Check if bus goes dominant during IDLE phase
endproperty
```

Scoreboard (Normal Mode)



```
# -----------------------------------------------------------------------------------------
# ----------------------------------ASSERTION SCOREBOARD-----------------------------------
# *****************************************************************************************
# TYPE OF CHECK                       TOTAL COUNT           PASS COUNT            FAIL COUNT
# -----------------------------------------------------------------------------------------
#  Bus IDLE Check                       1                     1                     0
#  Bit Stuffing Check                 290                   290                     0
#  CRC Delimiter check                 75                    75                     0
#  ACK Check                           75                    75                     0
#  ACK Delimiter Check                 75                    75                     0
#  EOF Check                           75                    75                     0
#  Overload Check                      75                    75                     0
# ******************************************************************************************
```

Scoreboard (Error inject Mode)

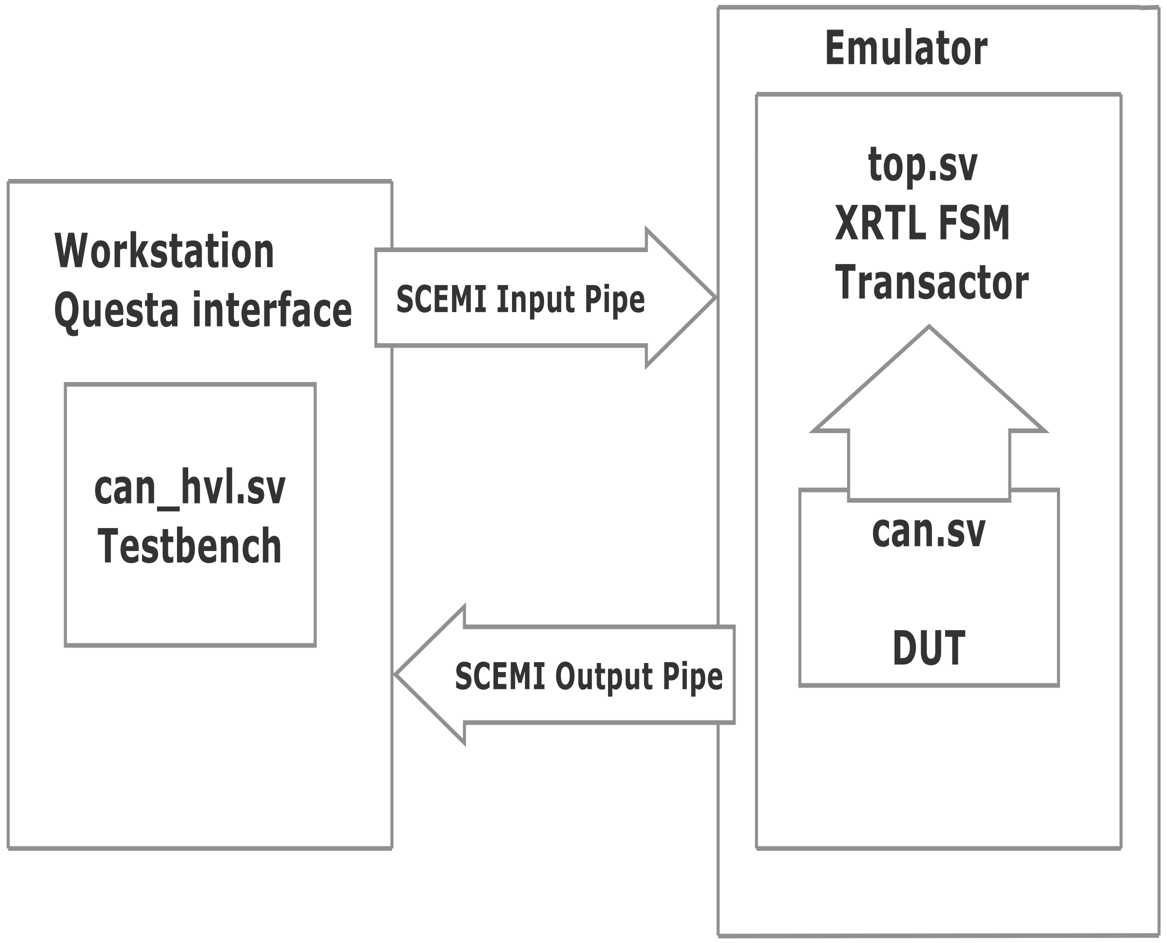


```
# ----------------------------------------------------------------------------------------
# --------------------------------ASSERTION SCOREBOARD-------------------------------------
# *****************************************************************************************
# TYPE OF CHECK                       TOTAL COUNT          PASS COUNT          FAIL COUNT
# ----------------------------------------------------------------------------------------
#  Bus IDLE Check                         1                    1                   0
#  Bit Stuffing Check                   481                  376                 105
#  CRC Delimiter check                   75                   75                   0
#  ACK Check                             75                   75                   0
#  ACK Delimiter Check                   75                   75                   0
#  EOF Check                             75                   75                   0
#  Overload Check                        75                   75                   0
# *****************************************************************************************
```

VELOCE EMULATION

TBX Mode

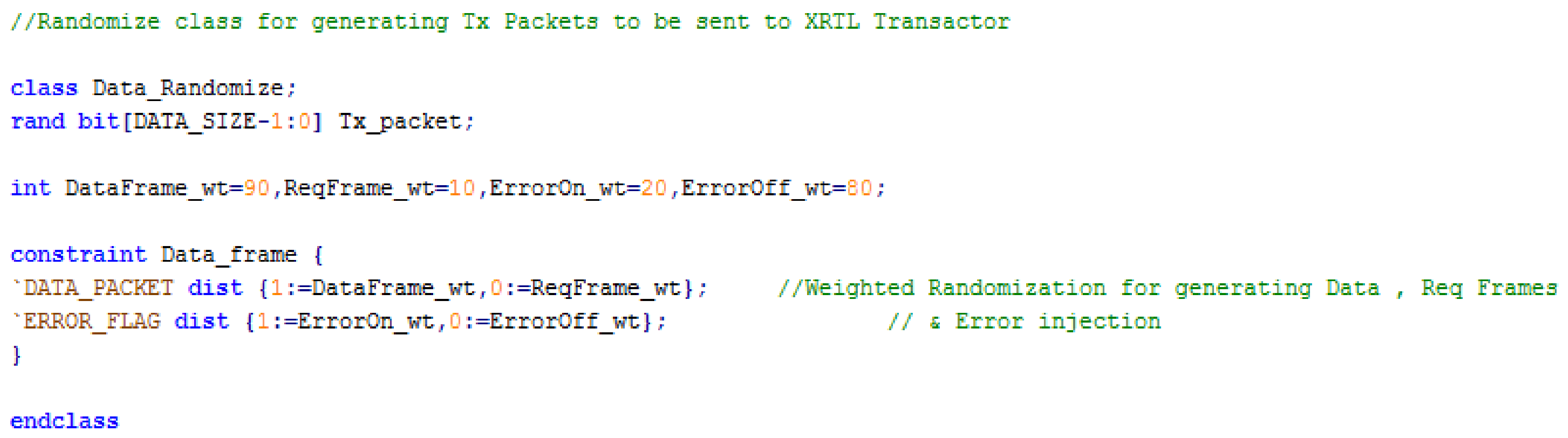
- XRTL Transactor configured as host processor stub for CAN nodes
- Input SCEMI pipe used to get data packets generated from HVL side
- Received packets, after bus transaction, are sent through output SCEMI
- HVL then does scoreboard of sent and received packets and prints out log



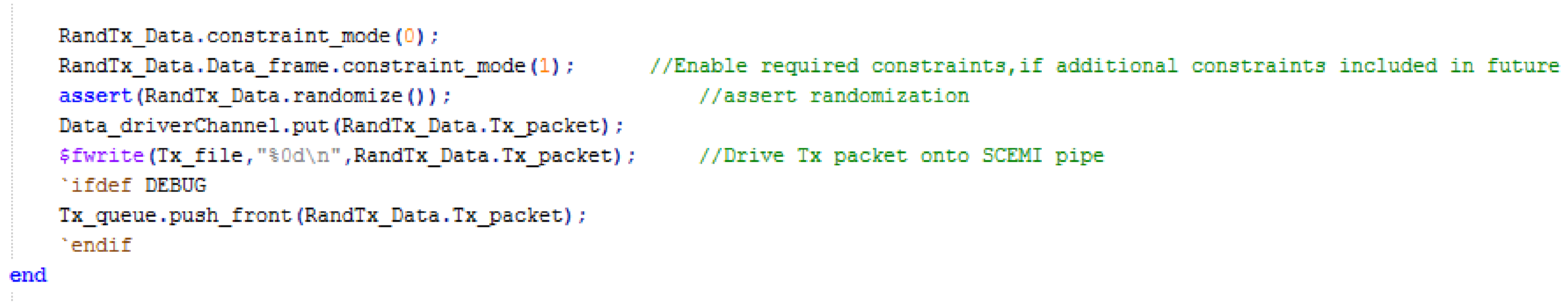
HVL

- Implements Constrained Randomization & OOP techniques for generating test packets & Identifiers
- Monitors received packets and produces a scoreboard at end of simulation run
- Logging of input and output packets in files

Randomization & OOP Constructs

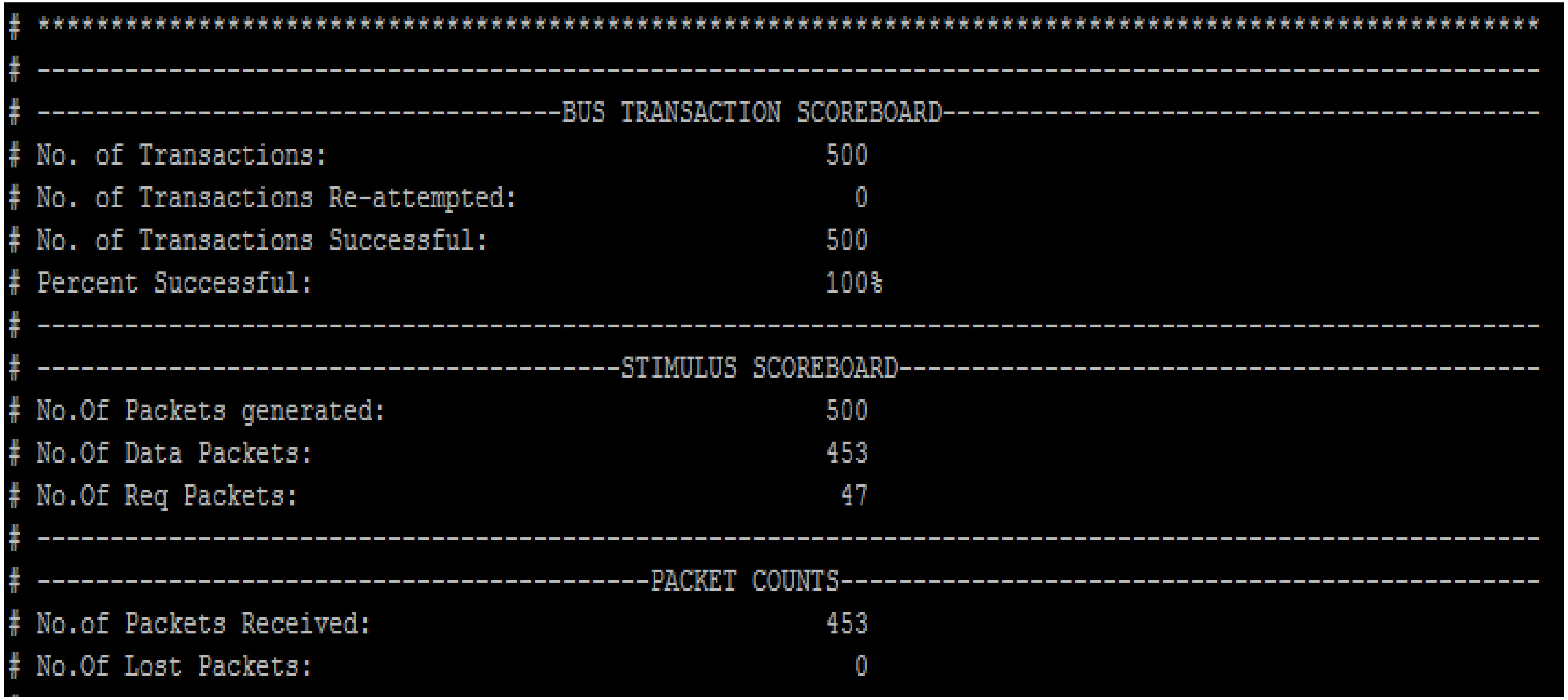


```
//Randomize class for generating Tx Packets to be sent to XRTL Transactor

class Data_Randomize;
rand bit[DATA_SIZE-1:0] Tx_packet;

int DataFrame_wt=90,ReqFrame_wt=10,ErrorOn_wt=20,ErrorOff_wt=80;

constraint Data_frame {
`DATA_PACKET dist {1:=DataFrame_wt,0:=ReqFrame_wt};     //Weighted Randomization for generating Data , Req Frames
`ERROR_FLAG dist {1:=ErrorOn_wt,0:=ErrorOff_wt};                //  & Error injection
}

endclass
```



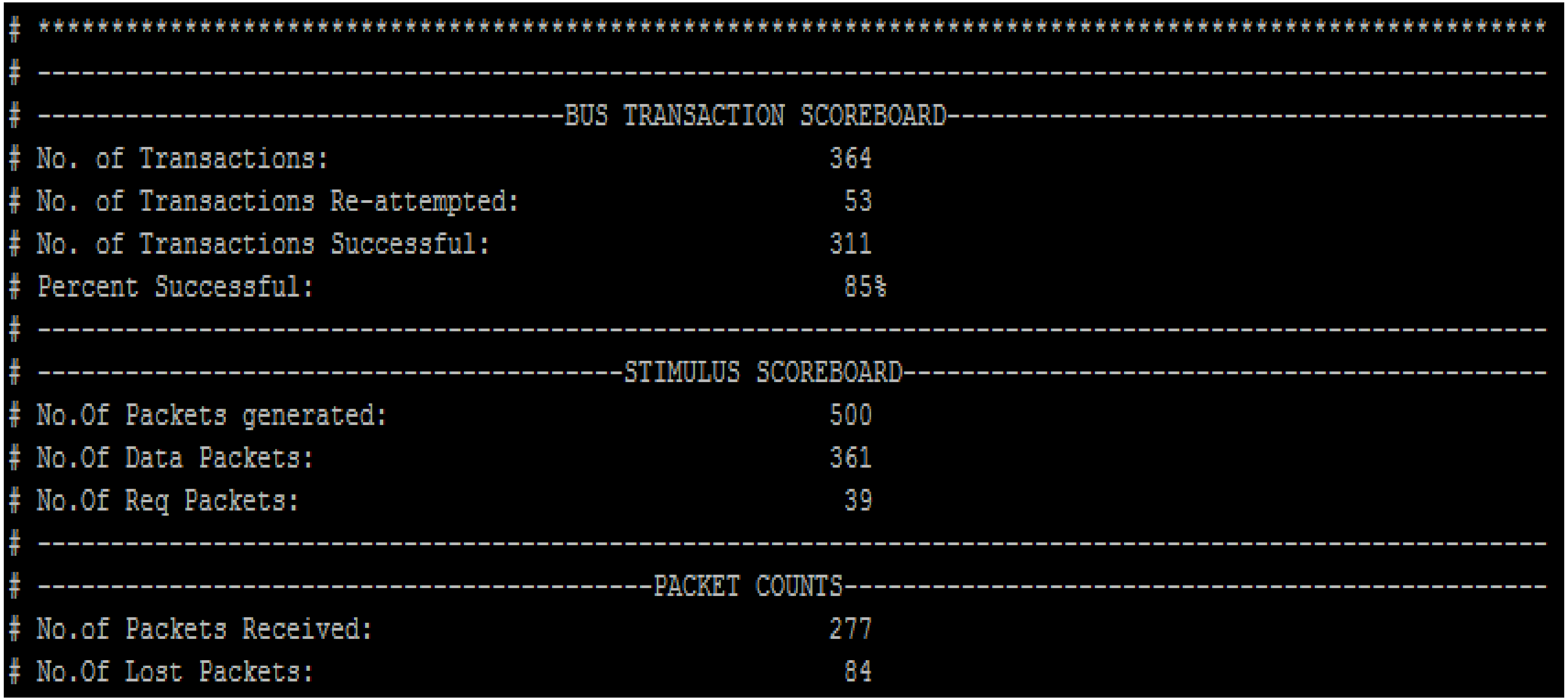
```
    RandTx_Data.constraint_mode(0);
    RandTx_Data.Data_frame.constraint_mode(1);      //Enable required constraints,if additional constraints included in future
    assert(RandTx_Data.randomize());                    //assert randomization
    Data_driverChannel.put(RandTx_Data.Tx_packet);
    $fwrite(Tx_file,"%0d\n",RandTx_Data.Tx_packet);     //Drive Tx packet onto SCEMI pipe
    `ifdef DEBUG
    Tx_queue.push_front(RandTx_Data.Tx_packet);
    `endif
end
```

Scoreboard: Normal Mode (with 4 Nodes)



```
# ************************************************************************************************
# ------------------------------------------------------------------------------------------------
# ----------------------------------BUS TRANSACTION SCOREBOARD-------------------------------------
# No. of Transactions:                                   500
# No. of Transactions Re-attempted:                        0
# No. of Transactions Successful:                        500
# Percent Successful:                                    100%
# ------------------------------------------------------------------------------------------------
# --------------------------------------STIMULUS SCOREBOARD----------------------------------------
# No.Of Packets generated:                               500
# No.Of Data Packets:                                    453
# No.Of Req Packets:                                      47
# ------------------------------------------------------------------------------------------------
# ----------------------------------------PACKET COUNTS--------------------------------------------
# No.of Packets Received:                                453
# No.Of Lost Packets:                                      0
```

Scoreboard: Error inject Mode (with 4 Nodes)



```
# *******************************************************************************************************
# -------------------------------------------------------------------------------------------------------
# ------------------------------------BUS TRANSACTION SCOREBOARD-----------------------------------------
# No. of Transactions:                                  364
# No. of Transactions Re-attempted:                      53
# No. of Transactions Successful:                       311
# Percent Successful:                                    85%
# -------------------------------------------------------------------------------------------------------
# ----------------------------------------STIMULUS SCOREBOARD--------------------------------------------
# No.Of Packets generated:                              500
# No.Of Data Packets:                                   361
# No.Of Req Packets:                                     39
# -------------------------------------------------------------------------------------------------------
# ------------------------------------------PACKET COUNTS------------------------------------------------
# No.of Packets Received:                               277
# No.Of Lost Packets:                                    84
```

Verification Features

- Monitor module running concurrent assertions to trap protocol violations
- Weighted randomizations allows more control on test vector generation
- Randomization done on:
  - Environment configuration – Multiple nodes instantiated using generate blocks and simple ID assignment algorithm employed by transactor to configure acceptance filters
  - Device configuration – Random ID's (hence randomized priority), Error injection, type of frames

Summary

- Build design from scratch using specification document
- Developed test environment for generating random stimulus and scoreboard
- Exercised most of the system Verilog constructs gained from the course in both design and functional verification

References

- http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf
- http://en.wikipedia.org/wiki/CAN_bus