SysWip

Alternative Verification

CAN2.0 Bus Verification IP User Manual

Release 1.0

Table of Contents

1. Introduction

The CAN (Controller area network) Bus Verification IP (VIP) described in this document is a solution for verification of CAN2.0 designs. The provided verification package includes CAN2.0 verification IP, integration examples and extensive test suit which cover most of the scenarios defined in the ISO/DIS 16845(Conformance test plan). The VIP is fully compliant with CAN2.0B specification and can be very powerful tool to check, monitor and debug the CAN2.0-Bus protocol.

Package Hierarchy

After downloading and unpacking package you will have the following folder hierarchy:

- can_vip
  - docs
  - examples
    - sim
    - testbench
  - verification_ip

The Verification IP is located in the verification_ip folder.

Features

- Compliant to the CAN2.0B specification
- Supports all frame types
  - Data frames
  - Remote frames
  - Error frames
  - Overload frames
- Automatic response to the remote frames
- Supports all error types
  - Bit errors
  - Stuff errors
  - CRC error
  - Form errors
  - Acknowledgment errors
- Automatic re-transmission of the corrupted data or remote frames
- Programmable TX and RX error counters
- Supports all types of error injection
- Supports programmable bus idle insertion between data frames
- Supports data or remote frame transmission at the 3rd intermission bit
- Automatic overload frame transmission after data or remote frames.
- Separate RX data buffers for each identifiers
- Supports full debug frame transmission to test all possible corner cases
- Extensive status report
- Full duplex

- Bus protocol checker
- Easy integration and usage
- Free SystemVerilog source code

2. Functional Description

The VIP includes CAN-Bus transmitter and receiver. The transmitter transmits data and remote frames and generates error and overload frames if necessary.

The receiver receives data frames and put it to the internal buffers which are accessible by corresponding commands. It generates error and overload frames if necessary.

The VIP has a remote frame response buffer which keep data frames for transmission as a response to the remote frames.

Commands

Commands marked as a *queued* will be put in the one queue and executed sequentially.

Commands marked as a *blocking* will block other commands execution until finishing its own process.

- **Configuration Commands**
  - **configGlobal():** - *non queued, non blocking*
  - **configTransmitter():** - *non queued, non blocking*
  - **configReceiver(): -** *non queued, non blocking*
  - **configOverloadFrames():** - *non queued, non blocking*
- **Data Transfer Commands**
  - **txDataFrame(): -** *queued, non blocking*
  - **txRemoteFrame(): -** *queued, non blocking*
  - **putARFR_Buff(): -** *non queued, non blocking*
  - **getRxDataFrame():** - *non queued, blocking*
  - **txDebugFrame():** - *queued, non blocking*
  - **rxDebugFrame():** - *non queued, blocking*
  - **setIdle():** - *queued, non blocking*
- **Environment Commands**
  - **startEnv(): -** *non queued, non blocking*
  - **envDone(): -** *non queued, non blocking*
  - **setDebugMode():-** *non queued, non blocking*
  - **checkErrors():-** *non queued, non blocking*
  - **checkInfoBox():-** *non queued, non blocking*

Commands Description

All commands are *CAN_m_env* class methods.

- **txDataFrame()**
  - **Syntax**
    - *txDataFrame(int identifier11, identifier18, ide, bit[7:0] dataTx[], bit[18:0] paramTx)*
  - **Arguments**
    - *identifier11:* An *int* variable which specifies 11 bit identifier value.
    - *identifier18:* An *int* variable which specifies 18 bit identifier value. During standard frame format this field is ignored.
    - *ide:* An *int* variable which specifies frame format (0-standard, 1-extended)
    - *dataTx:* A data buffer. The data buffer size is from 0 to 8 bytes. If it is more than 8 then will be truncated till 8 bytes.
    - *paramTx:* An 19 bit vector which specifies parameters of the frame. This parameter can be used to insert different type of errors. Default value is: *paramTx* = 19'h0ff10.
      - *paramTx[3:0] – DLC value. If this field is 4'b0000 then frame DLC field will be set to the size of dataTx buffer. If this field is more than 8 then frame DLC field will be set to this value. If this field is more than 0 and is less than 9 then frame DLC field will be set to the random value in range 9:15. The default is 4'b0000.*
      - *paramTx[4]- SRR value. Frame SRR value. Default is 1'b1.Is ignored during standard frames.*
      - *paramTx[5]- R0 value. Frame R0 value. Default is 1'b0.*
      - *paramTx[6] – R1 value. Frame R1 value. Default is 1'b0. Is ignored during standard frames.*
      - *paramTx[7] – CRC error. When this bit is 1'b1 then frame CRC field will be corrupted to insert CRC error for receivers.*
      - *paramTx[8] – CRC delimiter value. Frame CRC delimiter value. Default is 1'b1.*
      - *paramTx[15:9] – End Of Frame (EOF) value. Frame EOF value.*
      - *paramTx[16] – Stuff error. When this bit is 1'b1 then the stuff bit will be inverted inserting stuff error for receivers.*
      - *paramTx[17] – Acknowledge error. When this bit is 1'b1 then acknowledge bit (dominant bit) generated by receiver will be forced to recessive. It will insert bit error.*
      - *paramTx[18] – Acknowledge delimiter error. When this bit is 1'b1 then acknowledge delimiter bit (recessive bit) generated by receiver will be forced to dominant. It will insert bit error.*
  - **Description**
    - Transmits data frame. If error occurred or arbitration lost during transmission the frame will be re-transmitted after bus is idle. Before re-transmission the *paramTx* will be set to its default value.
- **txRemoteFrame()**
  - **Syntax**
    - *txRemoteFrame(int identifier11, identifier18, ide, bit[18:0] paramTx)*
  - **Arguments**
    - See *txDataFrame* command. ***Note:*** Remote frame does not have data field.
  - **Description**
    - Transmits remote frame. If error occurred or arbitration lost during transmission

the frame will be re-transmitted after bus is idle. Before re-transmission the *paramTx* will be set to its default value.

- **putARFR_Buff()**
  - **Syntax**
    - *putARFR_Buff(int identifier11, identifier18, ide, bit[7:0] dataTx[], bit[18:0] paramTx)*
  - **Arguments**
    - See *txDataFrame* command.
  - **Description**
    - Put data frame to the automatically remote frame response (ARFR) buffer. If remote frame received and have the same identifiers and ide values as has the frame in the ARFR buffer the frame from the ARFR buffer will be transmitted automatically. Then it will be removed from the buffer. If the remote frame received but ARFR buffer is empty or the frames (received vs. ARFR buffer) have different identifiers or ides then remote frame will be ignored.
- **getRxDataFrame**()
  - **Syntax**
    - getRxDataFrame*(int identifier11, identifier18, ide, bit[7:0] dataRx[])*
  - **Arguments**
    - *identifier11:* An *int* variable which specifies 11 bit identifier value.
    - *identifier18:* An *int* variable which specifies 18 bit identifier value. During standard frame format this field is ignored.
    - *ide:* An *int* variable which specifies frame format (0-standard, 1-extended)
    - *dataRx:* Received data buffer.
  - **Description**
    - Wait until data frame with specified identifier is received then pass data field with the dataRx argument.
- **txDebugFrame**()
  - **Syntax**
    - *txDebugFrame(bit[511:0] debugData, int debugDataSize, bit[511:0] frEn)*
  - **Arguments**
    - *debugData:* The *bit* vector that specifies the debug information which should be transmitted.
    - *debugDataSize*: An int variable that specifies the debug data size in bits.
    - *frEn:* The *bit* vector that specifies the bits which should be forced to the recessive state.
  - **Description**

- Transmits debug information and returns to the IDLE state. Very useful function to test and debug corner cases.

- **rxDebugFrame**()
  - **Syntax**
    - rxDebugFrame*(bit[511:0] debugData)*
  - **Arguments**
    - *debugData:* The *bit* vector that specifies the received debug data
  - **Description**
    - Gets the debug data from the internal buffer which was stored during *txDebugFrame* transmission.
- **setIdle()**
  - **Syntax**
    - *setIdle(time idleTime)*
  - **Arguments**
    - *idleTime:* A *time* variable that specifies the idle time between data or remote frame transmissions
  - **Description**
    - Inserts idle time between data or remote frame transmissions. The bus idle will be interrupted if another CAN point starts transmission.
- **configGlobal()**
  - **Syntax**
    - *configGlobal(time t_bitTime, t_sample, int rstInfoBox,int rstErrBox, tec,rec, errMode, passiveModeVal)*
  - **Arguments**
    - *t_bitTime:* A *time* variable which specifies the nominal bit time
    - *t_sample:* A *time* variable which specifies the sample time
    - *rstInfoBox:* An *int* variable which cleans info buffer
    - *rstErrBox:* An *int* variable which does the following actions:
      - Cleans all error buffers
      - Sets TEC and REC to 0
      - Enables error active mode
      - Sets internal overload counter to 0
    - *tec:* An *int* variable which specifies TEC (transmit error counter) initial value
    - *rec:* An *int* variable which specifies REC (receive error counter) initial value
    - *errMode:* An *int* variable which specifies initial error mode. 0 is active, 1 is passive.

- *passiveModeVal:* An *int* variable which specifies the counters value(TEC or REC) to enter error passive mode. The default value is 127.

- **configTransmitter()**
  - **Syntax**
    - *configTransmitter(int startTXatIFS3bit, activeErrFlag,delErrFlag,ifsVal)*
  - **Arguments**
    - *startTXatIFS3bit:* An *int* variable which enables data or remote frame transmission at the 3rd bit of intermission.
    - *activeErrFlag:* An *int* variable which specifies active error flag value. The default value is 0. The MSB will be transmitted first.
    - *delErrFlag:* An *int* variable which specifies error delimiter value. The default value is 7'b1111111. The MSB will be transmitted first.
    - *ifsVal:* An *int* variable which specifies the intermission value. The default value is 3 (2'b11). The MSB will be transmitted first.
- **configReceiver()**
  - **Syntax**
    - *configReceiver(time t_readTimeOut, int ackErr, ackDelimiter)*
  - **Arguments**
    - *t_readTimeOut:* An *time* variable wich specify the time out value during *getRxDataFrame* task.
    - *ackErr:* An *int* variable which disables acknowledge bit. If this bit is 1 then receiver will not send dominant bit at the acknowledge slot.
    - *ackDelimiter:* An *int* variable which disables acknowledge delimiter bit. If this bit is 1 then receiver will send dominant bit at the acknowledge delimiter.
- **startEnv()**
  - **Syntax**
    - *startEnv()*
  - **Description**
    - Starts CAN-Bus environment. Don't use data transfer commands before the environment start.
- **envDone()**
  - **Syntax**
    - *envDone()*
  - **Description**
    - Displays error and info messages from the status buffer.
- **setDebugMode()**
  - **Syntax**

        - *setDebugMode(int debugEn)*
    - **Arguments**
        - *debugEn:* An *int* variable that enables or disables debug mode.
    - **Description**
        - Enables or disables debug mode. Debug mode turns on detailed messaging on the terminal.
- **checkErrors()**
    - **Syntax**
        - *checkErrors(string errType)*
    - **Arguments**
        - *errType:* A *string* variable that specify the error buffers or counters. This argument can only have the folllowing values:
            - "ALL" - returns the sum of all errors (CRC, STUFF, BIT, FORM, ACKNOLEDGE)
            - "CRC" - returns the amount of CRC errors
            - "BIT" - returns the amount of BIT errors
            - "STUFF" - returns the amount of STUFF errors
            - "FORM" - returns the amount of FORM errors
            - "ACK" - returns the amount of ACKNOLEDGE errors
            - "TEC" - returns the TEC counter value
            - "REC" - returns the REC counter value\
- **checkInfoBox()**
    - **Syntax**
        - *checkInfoBox(string infoType)*
    - **Arguments**
        - *infoType:* A *string* variable that selects info box or overload counter. This argument can only have the following values:
            - "infoBox" - returns the number of messages in the info buffer. The info messages will be generated during the following cases:
                1. SRR bit is dominant during extended frames
                2. The R0 or R1 bits are recessive
                3. DLC is more than 8
                4. The last bit of EOF is dominant
            - "ovld" - returns the value of the overload counter. The overload counter will be incremented if:
                1. There is dominant bit at the first or second bit of intermission.

2. The last bit of error or overload delimiter is dominant.

Integration and Usage

The CAN-Bus VIP integration into your environment is very easy. Instantiate the *can_txrx_if* interface in you testbench and connect interface ports to your DUT. Then during compilation don't forget to compile *can_txrx.sv* and *can_txrx_if.sv* files located inside the *can_vip/verification_ip/* folder.

For usage the following steps should be done:

1. Import *CAN_TXRX* package into your test.
    - **Syntax***: import CAN_TXRX::*;*
2. Create *CAN_txrx_env* class object
    - **Syntax***: CAN_txrx_env can_txrx= new(can_txrx_ifc, name);*
    - **Description:**
        - *can_txrx_ifc* is the reference to the CAN-Bus interface instance name.
        - *name:* A *string* variable.
3. Start CAN Environment.
    - **Syntax:** *can_txrx.startEnv();*