Error Injections in PCIE-VR

PCIE-VR supports all possible error injections with high-level control in System Description Language (SDL). Error injections can be at any bit within any selected packet. Moreover, User-Defined Function (UDF) can be used for ASIC-specific error injections. The high-level control mechanism gives the user the capability to create hundreds of error injection test cases automatically.

1.1 Packet Selector

PCIE-VR implements a powerful packet selector that has unlimited 32-bit filters. With un-limited filters and AND/OR operation among the filters, virtually any special packet can be chose for error injection purpose.

The follow test case example shows how the packet selector works. The first property totalPktSelectors defines the number of filters. Each filter is numbered from 0 to totalPktSelectors-1. The property selectPkt(N) defines the 32-bit content of the filter N. The property selectLoc(N) defines the location and the property selectMask(N) defines the bits to be ignored by the filter N.
After all the filters are defined, the property selectFunc defines either AND or OR operation among the filters.

```
// how many packet selectors
// default is 4
totalPktSelectors = 4

// select a packet that has content 0x1234 at byte location 3
// default is all packets selected
// (loc[3] = 4, loc[4]=3, loc[5]=2, loc[6]=1)
selectPkt(0) = 0x12340000
// which 4 bytes from the beginning of the tlp header
// the property can have a negative value, counting backward from the end of packet
selectLoc(0) = 3
// bit[15:0] are ignored
selectMask(0) = 0x0000ffff

// packet selector 1
// select the packet that has content 0x1234 at byte location 3
// default is all packets selected
// (loc[3] = 4, loc[4]=3, loc[5]=2, loc[6]=1)
selectPkt(1) = 0x12340000
// which 4 bytes from the beginning of the tlp header
// the property can have a negative value, counting backward from the end of packet
selectLoc(1) = 3
// bit[15:0] are ignored
selectMask(1) = 0x0000ffff

// repeat up to errInjPktSelectorSize - 1

// packet selector AND/OR selections
// either AND (0) or OR (1)
// default is AND (0)
```

```
selectFunc = 0
```

1.2 Error Injections at the Transaction Layer

The following test case example shows how to inject errors to Transaction Layer Packets. For example, ECRC errors, data errors down to any bit, nullifying TLPs, Data poison, and completion return errors are supported.

```
// ECRC error rate per packet
// one crc error per 100 packets
// ECRCErrorRate = 100
ECRCErrorRate = 0

// inject TLP data errors but crcs are correct
tlpDataErrorRate = 0
// which byte from the begining of the tlp header
// the property can have a negative value, counting backward from the end of packet
// use for wrong packet length, none-zero reserved field, etc
tlpDataErrorLocation = 2
// within the byte, which bits should NOT have error injected
// default is 0
// for example, 0x05 means bit2 and bit0 are always good (protected)
tlpDataErrorMask = 0x05

// nullifying a TLP
// If value is [100,500], one nullified TLP per [100, 500] TLP packets
tlpNullifyRate = 0

// data error poison injection
// This property is used in UDF as an example
tlpErrorPoisonOn = 0

// different completion status testing
// If the value is [2,6], a specific completion return status is
// injected for every 2 to 6 completion packets
// UR: Unspported Request
// CA: Completion Abort
// CRS: Configuration Retry Status
compRetURRate = 0
compRetCARate = 0
compRetCRSRate = 0
```

1.3 Error Injections at the Data Link Layer

Error injections at the data link layer are LCRC errors, data errors down to any bit, and random TLP packet dropping for transmit retry buffer verification.

```
// LCRC error rate per TLP packet
LCRCErrorRate = 0

// 16-bit error rate per DLLP packet
// Good for NAK testing
DCRCErrorRate = 0

// inject DLLP data errors but 16-bit dcrc is correct
dllpDataErrorRate = 0
// which byte from the begining of the dllp header
// the property can have a negative value, count backward from the end of packet
// data = ~data
dllpDataErrorLocation = -2
// within the byte, which bits should NOT have error injected
// default is 0
// for example, 0x05 means bit2 and bit1 are always good (protected)
dllpDataErrorMask = 0x05

// tx retry buffer testing
// randomly drop rx good tlp to test tx retry buffer on DUT
// If the value is [1,7], a good tlp will be randomly dropped for
// every 1 to 7 received good tlp
tlpRxPktDropRate = 0
```

Go to the next page to see error injections at the physical layer.

1.4 Error Injections at the Physical Layer

Error injections at the physical layer are K symbol errors, training set errors, control bit errors,

```
// specific K symbol error injection
// KSymbolError is the K character to be replaced
// such as STP(0xfb), SDP(0x5c), END(0xfd), EDB(0xfe), COM(0xbc), etc.
// error rate is per the number of the OS sent if not PAD
// error rate is per the number of PAD sent if PAD
KSymbolErrorRate = 0
KSymbolError = 0x5c   // SDP

// TS error injection
// TSErrorLocation is the location that the good character to be replaced
// error rate is per the number of TS packets sent
// TSErrorLocation must be inside the TS packet from 0 to 15
// the property can have a negative value, count backward from the end of packet
// TS error injection also applies to compliance pattern, loopback data, and L2 beacon
TSErrorRate = 0
TSErrorLocation = 4

// control bit error injection
// error rate is per packet for any TLP, DLLP, and OS sent
ctrlErrorRate = 0
// the location is related to the packet that has error injected
// the property can have a negative value, count backward from the end of packet
// change the control bit at the last byte
ctrlErrorLocation = -1
```

1.5 Error Injections at PIPE Interface

Error injection at the PIPE interface is receive-status error.

```
// PIPE rxStatus errors
// supported only inside the back-to-back PIPE phy modules
// one rx status error for every 5000 to 10000 cycles
// PIPERxStatusErrorRate = [5000, 10000]
PIPERxStatusErrorRate = 0
// error injection applies to which interfaces
// 0: PCIE-VR, 1: DUT
PIPERxStatusErrorInterface = [0,1]
// same rx status as defined in the PIPE spec
PIPERxStatusErrorType = [4, 7]
// which lane should have the error injected
// Automatically adjust to the link width
PIPERxStatusErrorLane = [0,5]
```