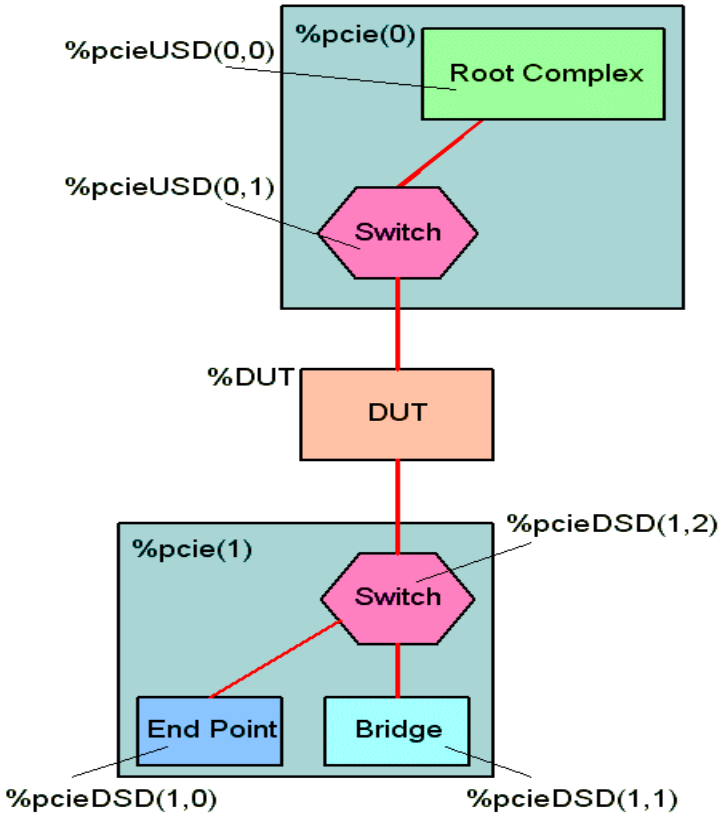
PCI Express Test Case in SDL

1.1 PCI Express Device Topology

This example is a test case for PCI Express ASICs illustrated in the following figure.



The port model 0, which is named as %pcie(0) in SDL, connects to the DUT via the upper PCIE link. The port model 1, %pcie(1) connects to the DUT via the lower link. Each port model contains a number of PCIE devices. Those PCIE devices are named as %pcieUSD(0,0) for the root complex instance, %pcieDSD(1,0) for the end point instance, etc.

On the next page, it is a test case in SDL with the above PCIE topology. The simulation represent a typical scenario that starts from link training, moves to flow control initialization for all the virtual channels after link up, then transmits and receives thousands of Transaction Layer Packets with proper sequence control. At end, the root complex requests a power-off. Both sides enter L2 states after the handshake protocol is completed.

2. Test Case

Test cases are inside text box and a header is given for each section.

```
// ---------------------------------------------------------------------
//
// Copyright (c) 2004-2006 Tarek Verification Systems. All Rights Reserved
//
// This file contains proprietary source code.
// This file may not be copied or redistributed in any form,
// or via any media, electronic, paper, or other, without he express written consent of Tarek
Verification Systems.
// Contact info:
//  www.tarek.com
//  info@tarek.com
//
// ---------------------------------------------------------------------
//
//  temp.sim for PCI Express functional model
//
//  pcie(0) is an upstream device with a downstream port
//    pcie(0) and pcieUSD(0,0) are internally connected

//  pcie(1) is a down stream device with an upstream port
//    pcieDSD(1,0) and pcieDSD(1,1) are internally connected to pcie(1)

//  pcie(0) and pcie(1) are connected directly via a PCIE link
//
```

2.1 The Upstream Device Object

```
//*********************************************************
// directly connected upstream device, link id 0
//*********************************************************
%pcie(0)

////////////////////////////////////////////////////////////
// Properties to set up this object
////////////////////////////////////////////////////////////

// This device use the following RDF file
rdf = $DRCHOME/dim/pciExp/rdf/usd.rdf:PCIE_USD

// internally connected devices
// default is 0 for an end point (DSD) or root complex (USD)
// When this property is 2, two objects,
```

```
// %pcieDSD(1,0) and %pcieDSD(1,1) are defined if this is a DSD
// %pcieUSD(1,0) and %pcieUSD(1,1) are defined if this is a USD
totalConnectedDevices = 1

// upstream device or downstream device
// 0 if this is a downstream device
// default is 0
upStreamDevice = 1

// bus, dev, and func
// default is 0, 0, 0
busId = 1
devId = 2
funcId = 3

// Compliance and functional Coverage
// default is on (1)
// set to 0 to exclude from the coverage report
coverageOn = 1
```

2.2 Serdes in the Upstream Device

```
////////////////////////////////////////////////////////
// Serdes properties
////////////////////////////////////////////////////////
// serdes reference clock frequency in MHz
// default is 2.5GHz
serdesRefClkFrequency = 2500

// serdes clock jitter margin in ps
// default is 10ps
serdesRxClkJitterMargin = 10

// number of signal transitions for serdes acquirement
// default is 4
serdesRxSyncAcquireThreshold = 4

// time interval when serdes period checking takes place
// default is 30 clock cycles
serdesRxClkPeriodCheckWindow = 30

// serdes tx clock initial delay in ps
// default is 1000ps
serdesTxInitialDelay = 1000
```

2.3 Physical Layer in the Upstream Device

```
// clock in Mhz
// default is 250
clockFrequency = 250

// timeout scaling
// timeOut = (realTimeOut * clockFrequency) / timeOutScale
timeOutScale = 250

// total available lanes
// must be <= hardwired lanes in Verilog or SystemC
totalLanes = 32

// supported link width
// bit 0 : x1
// bit 1 : x2
// bit 2 : x4
// bit 3 : x8
// bit 4 : x12
// bit 5 : x16
// bit 6 : x32
// default is 1 and totalLanes
supportedLinkWidth = 0x7f

// supported bit rate
// the value is 0x2 for gen1 and 0x6 for gen2
// default is 0x2
supportedBitRate = 0x6

// rx packet process time in clock
// default is 0
// rxProcessDelay = [100, 1000]
rxProcessDelay = 0

// LTS_POLLING_ACTIVE -> LTS_POLLING_CONFIG
// simulation speed-up
// default is 32 (spec is 1024)
// TS_PP1_1024 = 1024
TS_PP1_1024 = 32

// clock tolerance compensation
// the symbol time between two scheduled SKP OS
// default is between 1180 (0x49c) to 1538 (0x602)
// To disable, set to -1
SKPInterval = [1180, 1538]
```

```
// number of SKP K symbols
// default is 3 as the PCIE standard
// However, the phy may add or delete SKP due to clock jitter
// we need to generate 1 to 5 to make sure DUT can recognize all of them
SKPCount = [1, 5]

// number of FTS
// default is 1
numOfFTS = 5

// properties for ASPM support
// This is the register field defined in pcie configuration space
// no more txL0sAdj, use l0s_exit_latency instead
// l0s_exit_latency = 0

// legacy software
// If this is one a configuration request with CRS return status will
// cause a root complex to re-issue the request
legacySoftware = 0
```

2.4 LTSSM Control in the Upstream Device

```
////////////////////////////////////////////////////////
//  LTSSM state machine control properties
////////////////////////////////////////////////////////
// LTSSM state number
// 0 - detect quiet
// 1 - polling
// 2 - config
// 3 - L0
// 4 - recovery
// 5 - L1
// 6 - L2
// 7 - L0s (tx side)
// 8 - loopback (master only)
// 9 - hot reset
// 10 - disable
// 11:15 - preserved
// 16 - detect active

// LTSSM initial state (starting state when program starts)
// legal init states: detect quiet (0), L0 (3), recovery (4), detect active (16)
// default is 3
initLTSSMState = 3

// L0 state properties
// once the LTSSM has reached L0, we direct it to a new state
```

```
// default is 3 (stay in L0)
// legal values are 3 (L0), 4 (recovery), 5 (L0s), 6 (L1), 7 (L2)
// an USD can not initiate to L1/L2
// a new state will be picked after L0Period
L0NewState = [3, 5]
// The cycle time that LTSSM stays at L0 state before moves to new state
// default is -1 (forever)
L0Period = -1

// L0s state properties
// legal values are 3 (L0), 4 (recovery)
// a new state will be picked after L0sPeriod
L0sNewState = [3,4]
// The cycle time that LTSSM stays at L0s state before moves to new state
L0sPeriod = [1000,2000]

// L1 state properties
// it always moves to recovery
// default is 2048 for DSD
// USD can trigger DSD to request entry to L1 by writing
// a D state other than D0, e.g., D3_hot, to the power_state field
// L1Period = [1000,2000]

// L2 state properties
// it always moves to 0 (detect) after main power restored
// main power restored and simulated at the end of L2Period
// default is 2048 for DSD
// USD can trigger DSD to request entry to L2 by writing the PMTurn_Off message
// L2Period = [1000,2000]

// directed to configuration from recovery
// legal values are 3 (L0), 2 (config)
recoveryNewState = 2

// hotReset
// USD only
// when set to 1, LTSSM will enter hotReset state
// To enter hotReset state, LTSSM must be in recovery state
hotReset = 0
// hotReset period in cycle
// default is 2048
hotResetPeriod = 1000

// linkDisable
// USD only
// when set to 1, instruct LTSSM to enter the linkDisabled state
```

```
// to enter linkDisable state, LTSSM must be in config or recovery state
// default is 0
linkDisable = 0
// linkDisable period in cycle
// default is 2048
linkDisablePeriod = 1000

// loopback
// when set to 1, instruct LTSSM to enter the loopback state as master
// to enter loopback state, LTSSM must be in config or recovery state
// only one master is allowed
// default is 0
loopback = 0
// loopback period in cycle
// default is 2048
loopbackPeriod = 10000

// scramble disable
// when set to 1, instruct LTSSM to disable scramble
// default is 0
scramDisable = 0

// polling compliance trigger
// when set to 1, request other side to send polling compliance pattern
// default is 0 (off)
enablePollingCompliance = 0
// polling compliance period in cycle
// default is 2048
pollingCompliancePeriod = 1000

// let ltssm and dlcmsm be totally independent
// when ltssm move to detect state, dlcmsm can stay in DL_ACTIVE
// this is good for testing ltssm efficiently
// we can keep looping through all the states
// Note that both sides MUST set the same value
// default is 1
LTSSMIndependent = 0
```

2.5 Data Link Layer in the Upstream Device

```
// DLCMSM initial state, inactive (0) or active (2)
// default is 2
initDLCMSMState = 2
// The max number of sending the following PM DLLP repeatedly
// LNK_PM_ENT_L23, LNK_PM_ASR_L1, LNK_PM_ENT_L1, and LNK_PM_RQT_ACK
// default is -1, keep repeating until ack received
maxPMDLPRepeatCount = -1
```

2.6 Transaction Layer in the Upstream Device

2.6.1 TLP Traffic Generator

```
//////////////////////////////////////////////
// Automatic TLP Traffic Generator
//   as many generator as you want inside a pcie device
//////////////////////////////////////////////
// Only for memory read and write
// write-read-verify for every transaction

// set to 1 to enable the trafic
tlpTrafficEnable = 1

// traffic object 0
// number of transactions
// default is 0 (off)
tlpTrafficCount(0) = [1000, 3000]

// starting time of the traffic in cycle
tlpTrafficStart(0) = [0, 300]

// starting address
// no default, must be defined in rdf or -1
// if -1, the other side must have useSystemMemory set
tlpTrafficStartAddrHi(0) = 0x814
tlpTrafficStartAddrLo(0) = 0xc0000000

// the block that contains the traffic
// 1M, 10K, or 1024 bytes, etc.
// must be a constant
// no default, must be less than the memory size defined in rdf
tlpTrafficBlockSize(0) = 1M

// data length
// automatically reduced to maxPayloadLength
tlpTrafficDataLength(0) = [10,4096]

// next address
// automatically rewind
// default is 1
tlpTrafficNextAddrOffset = 1
// next data length increment
// can be negative
// default is 1
tlpTrafficDataLengthInc(0) = 1
```

```
//
// traffic object 1
// number of transactions
// default is 0 (off)
tlpTrafficCount(1) = [100, 256]

// starting time of the traffic in cycle
tlpTrafficStart(1) = [0, 300]

// starting address
// no default, must be defined in rdf or -1
// if -1, the other side must have useSystemMemory set
tlpTrafficStartAddrHi(1) = 0x814
tlpTrafficStartAddrLo(1) = 0xd0000000
// the block that contains the traffic
// 1M, 10K, or 1024 bytes, etc.
// must be a constant
// no default, must be less than the memory size defined in rdf
tlpTrafficBlockSize(1) = 1M

// data length
// automatically reduced to maxPayloadLength
tlpTrafficDataLength(1) = [10,4096]

// next address
// automatically rewind
// default is 1
tlpTrafficNextAddrOffset(1) = 1
// next data length increment
// can be negative
// default is 1
tlpTrafficDataLengthInc(1) = 1
//
// End of TLP traffic generation
```

2.6.2 Configuration Register in the Upstream Device

```
// number of RCB blocks per completion
// default is random from 1 to 3
// the value must be >= 1
RCBBlocks = [1,3]

// completion timeout in cycles
// default is 12500
// set to -1 to disable completion timeout
// [50us, 50ms] in spec
```

```
// 50us is 12500 clocks@250Mhz
compTimeOut = 12500

// For DMA to host memory or endpoint C model development
// When useSystemMemory sets to 1, real system addresses are used
// The correspoding UDFs will be called after TLPs are processed.
// Addresses, if defined in the rdf files, are ignored
// Warning! coredump may occur if the system memory are not properly allocated
// default is 0
useSystemMemory = 0

// override the field defined in rdf
// redefine the max_payload_size register
// max_payload_size = 0
// disable ECRC generation
// ecrc_generation_enable = 1
// disable ECRC checking
// ecrc_check_enable = 1

// tc to vc mapping is defined in PCIE config register (VC resource control register)
// Default in the rdf file is VCi=TCi, 0<=i<=7
// Follow the spec to set proper fields to define the mapping
// For example, To map TC1 and TC2 to VC1
// tc_vc1_map = 0x3
// vc1_id = 1
// vc1_enable = 1
```

2.6.3 Error Injection in the Upstream Device

```
////////////////////////////////////////////////////////
// error injection
////////////////////////////////////////////////////////
// default is no error (ErrorRate = 0)
```

2.6.3.1 Transaction Layer Error Injections

```
//////////////////////////////////////
// Transaction layer packet errors
//////////////////////////////////////
// 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
```

2.6.3.2 Data Link Layer Error Injections

```
////////////////////////////////////////
// Data link layer packet errors
////////////////////////////////////////
// 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
```

2.6.3.3 Physical Layer Error Injections

```
////////////////////////////////////////
// Physical layer 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
```

2.6.3.4 PIPE Interface Error Injections

```
// 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 DIM, 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]
```

2.7 The Root Complex Object

```
//************************************************************
// upstream device that is connected to %pcie(0)
// This one is RC
// device_port_type in config defines this is an rc
//
//************************************************************
%pcieUSD(0,0)
rdf = $DRCHOME/dim/pciExp/rdf/RootComplex/rc.rdf:PCIE_RC
// device_id is for only for ADG testing
device_id= 0xfc

// bus, dev, and func
// no default
busId = 0
devId = 0
funcId = 0
```

```
// Property for DMA to real host memory in the running process
// Good for driver-like C/C++ verification environment or endpoint C model
// development
// Good to NOT have a Verilog memory model for system memory to speed up
// When useSystemMemory sets to 1, real system addresses are used
// The correspoding UDFs will be called after TLPs are processed
// Note: Addresses defined in the rdf files are IGNORED
// Warning! coredump may occur if the system memory are not properly allocated
// default is 0
useSystemMemory = 1

// legacy software
// If this is one a configuration request with CRS return status will
// cause a root complex to re-issue the request
legacySoftware = 0
```

2.8 The Downstream Device Object

```
//*************************************************************
// directly connected downstream device, link id 1
//
//*************************************************************
%pcie(1)

rdf = $DRCHOME/dim/pciExp/rdf/DSD/dsd.rdf:PCIE_TYPE1
upStreamDevice = 0 // this is a downstream device

// bus, dev, and func
busId = 1
devId = 2
funcId = 0

// total available lanes
// must be <= hardwired lanes in Verilog or SystemC
totalLanes = 32

// supported link width
// bit 0 : x1
// bit 1 : x2
// bit 2 : x4
// bit 3 : x8
// bit 4 : x12
// bit 5 : x16
// bit 6 : x32
// default is 1 and totalLanes
supportedLinkWidth = 0x7f
```

```
// supported bit rate
// the value is 0x2 for gen1 and 0x6 for gen2
// default is 0x2
supportedBitRate = 0x6

// number of RCB blocks per completion
// default is random from 1 to 3
// the value must be >= 1
RCBBlocks = 1

// completion timeout in cycles
// default is 12500
// set to -1 to disable completion timeout
// [50us, 50ms] in spec
// 50us is 12500 clocks@250Mhz
compTimeOut = 12500

// internally connected devices
// default is 0 for an end point or root complex
// When this property is 2, two objects,
// %pcieDSD(1,0) and %pcieDSD(1,1) are defined if this is a DSD
// %pcieUSD(1,0) and %pcieUSD(1,1) are defined if this is a USD
totalConnectedDevices = 2

// rx packet process time in clock
// default is 0
// rxProcessDelay = [100, 1000]
rxProcessDelay = 0

// tx intra packet gap in clock
// txIpg is only for TLP and DLLP
// When TLP and DLLP transmitted at the same symbol time, this property is ignored.
// default is 0
// txIpg = [100, 1000]
txIpg = 0

////////////////////////////////////////////////////////////
//  LTSSM state machine control properties
// LTSSM state number
// 0 - detect quiet
// 1 - polling
// 2 - config
// 3 - L0
// 4 - recovery
// 5 - L1
```

```
// 6 - L2
// 7 - L0s (tx side)
// 8 - loopback (master only)
// 9 - hot reset
// 10 - disable
// 11:15 - preserved
// 16 - detect active

// LTSSM initial state (starting state when program starts)
// legal init states: detect quiet (0), L0 (3), recovery (4), detect active (16)
// default is 3
initLTSSMState = 3

// L0 state properties
// once the LTSSM has reached L0, we direct it to a new state
// default is 3 (stay in L0)
// legal values are 3 (L0), 4 (recovery), 5 (L0s), 6 (L1), 7 (L2)
// an USD can not initiate to L1/L2
// a new state will be picked after L0Period
L0NewState = [3, 7]
// The cycle time that LTSSM stays at L0 state before moves to new state
// default is -1 (forever)
L0Period = -1

// L0s state properties
// legal values are 3 (L0), 4 (recovery)
// a new state will be picked after L0sPeriod
L0sNewState = [3,4]
// The cycle time that LTSSM stays at L0s state before moves to new state
L0sPeriod = [1000,2000]

// L1 state properties
// it always moves to recovery
// default is 2048 for DSD
// USD is always off
L1Period = [1000,2000]

// L2 state properties
// it always moves to 0 (detect) after main power restored
// default is 2048
// main power restored and simulated at the end of L2Period
// default is 2048 for DSD
// USD is always off
L2Period = [1000,2000]

// linkDisable
```

```
// when set to 1, instruct LTSSM to enter the linkDisabled state
// default is 0
linkDisable = 0
// linkDisable period in cycle
// default is 2048
linkDisablePeriod = 10000

// loopback
// when set to 1, instruct LTSSM to enter the loopback state as master
// only one master is allowed
// default is 0
loopback = 0
// loopback period in cycle
// default is 2048
loopbackPeriod = 10000
```

```
// polling compliance trigger
// when set to 1, request other side to send polling compliance pattern
// default is 0 (off)
enablePollingCompliance = 0
// polling compliance period in cycle
// default is 2048
pollingCompliancePeriod = 1000

// LTS_POLLING_ACTIVE -> LTS_POLLING_CONFIG
// simulation speed-up
// default is 32 (spec is 1024)
// TS_PP1_1024 = 1024
TS_PP1_1024 = 32

// DLCMSM initial state, inactive (0) or active (2)
// default is 2 when ltssm is set to L0
initDLCMSMState = 2

// clock in Mhz
// default is 250
clockFrequency = 250

// LTSSM timeout scaling for speed up
// timeOut = (realTimeOut * clockFrequency) / timeOutScale
// 12ms becomes 48 us (12ms speed up 250 = 12*4 us)
timeOutScale = 250

// let ltssm and dlcmsm be totally independent
// when ltssm move to detect state, dlcmsm can stay in DL_ACTIVE
// this is good for testing ltssm efficiently
```

```
// we can keep looping through all the states
// Note that both sides MUST set the same value
// default is 1
LTSSMIndependent = 0

// number of FTS
// default is 1
// match the other side
numOfFTS = 5
```

2.9 The End Point Object

```
//*****************************************************************
//
// indirectly connected down stream pcie devices with config/mem/io spaces
//
//*****************************************************************
%pcieDSD(1,0)
// device 0
rdf = $DRCHOME/dim/pciExp/rdf/EndPoint/ep.rdf:PCIE_TYPE0

// bus, dev, and func
// default is busId%pcie(1)
busId = 3
devId = 4
funcId = 0
```

2.10 The Bridge Object

```
%pcieDSD(1,1)
// device 1
// This device use the following rdf file
rdf = $DRCHOME/dim/pciExp/rdf/Bridge/bridge.rdf:PCIE_TYPE1
device_id= 0xab02

// bus, dev, and func
// default is busId%pcie(1)
busId = 3
devId = 5
funcId = 1
```