connectal Documentation

Release 14.11.6

Jamey Hicks, Myron King, John Ankcorn

December 17, 2014

Contents:

CHAPTER

ONE

INTRODUCTION

Introduction goes here.

CHAPTER

TWO

CONNECTAL BSV LIBRARIES

2.1 CtrlMux Package

module `CtrlMux::mkInterruptMux` (*Vector#(numPortals, MemPortal#(aw, dataWidth)) portals*) → (ReadOnly#(Bool)

Used by BsimTop, PcieTop, and ZynqTop. Takes a vector of MemPortals and returns a boolean indicating whether any of the portals has indication method data available.

module `CtrlMux::mkSlaveMux` (*Vector#(numPortals, MemPortal#(aw, dataWidth)) portals*) → (PhysMemSlave#(addrWidth,dataWidth)

Takes a vector of MemPortals and returns a PhysMemSlave combining them.

2.2 HostInterface Package

The HostInterface package provides host-specific typedefs and interfaces.

2.2.1 Host-Specific Constants

typedef `HostInterface::DataBusWidth`

Width in bits of the data bus connected to host shared memory.

typedef `HostInterface::PhysAddrWidth`

Width in bits of physical addresses on the data bus connected to host shared memory.

typedef `HostInterface::NumberOfMasters`

Number of memory interfaces used for connecting to host shared memory.

2.2.2 Host-Specific Interfaces

interface `HostInterface::BsimHost`

Host interface for the bluesim platform

interface `HostInterface::PcieHost`

Host interface for PCIe-attached FPGAs such as vc707 and kc705

interface `HostInterface::ZynqHost`

Host interface for Zynq FPGAs such as zedboard, zc702, zc706, and zybo.

The Zc706 is a ZynqHost even when it is plugged into a PCIe slot.

2.3 Leds Package

interface `Leds::`**LEDS**

typedef `Leds::`**LedsWidth**
Defined to be the number of default LEDs on the FPGA board.

The Zedboard has 8, Zc706 has 4, ...

`Leds::`**leds** → Bit#(LedsWidth)

2.4 MemPortal Package

2.4.1 mkMemPortal Module

module **mkMemPortal** (*Bit#(slaveDataWidth) ifcId*, *PipePortal#(numRequests*, *numIndications*, *slaveDataWidth) portal*) → (MemPortal#(slaveAddrWidth, slaveDataWidth)
Takes an interface identifier and a PipePortal and returns a MemPortal.

2.5 MemTypes Package

2.5.1 Constants

typedef **Bit#(32) SGLId**

typedef **44 MemOffsetSize**

typedef **6 MemTagSize**

typedef **8 BurstLenSize**

typedef **32 MemServerTags**

2.5.2 Data Types

struct **PhysMemRequest#(numeric type addrWidth)**
A memory request containing a physical memory address

addr → Bit#(addrWidth)
Physical address to read or write

burstLen → Bit#(BurstLenSize)
Length of read or write burst, in bytes. The number of beats of the request will be the burst length divided by the physical width of the memory interface.

tag → Bit#(MemTagSize)

struct **MemRequest**
A logical memory read or write request. The linear offset of the request will be translated by an MMU according to the specified scatter-gather list.

sglId → SGLId
Indicates which scatter-gather list the MMU should use when translating the address

offset → Bit#(MemOffsetSize)
Linear byte offset to read or write.

burstLen → Bit#(BurstLenSize)
Length of read or write burst, in bytes. The number of beats of the request will be the burst length divided by the physical width of the memory interface.

tag → Bit#(MemTagSize)

struct MemData#(numeric type dsz)
One beat of the payload of a physical or logical memory read or write request.

data → Bit#(dsz)
One data beat worth of data.

tag → Bit#(MemTagSize)
Indicates to which request this beat belongs.

last → Bool
Indicates that this is the last beat of a burst.

2.5.3 Physical Memory Clients and Servers

interface PhysMemSlave (*numeric type addrWidth*, *numeric type dataWidth*)

read_server → PhysMemReadServer#(addrWidth, dataWidth)

write_server → PhysMemWriteServer#(addrWidth, dataWidth)

interface PhysMemMaster (*numeric type addrWidth*, *numeric type dataWidth*)

read_client → PhysMemReadClient#(addrWidth, dataWidth)

write_client → PhysMemWriteClient#(addrWidth, dataWidth)

interface PhysMemReadClient (*numeric type asz*, *numeric type dsz*)

readReq → Get#(PhysMemRequest#(asz))

readData → Put#(MemData#(dsz))

interface PhysMemWriteClient (*numeric type asz*, *numeric type dsz*)

writeReq → Get#(PhysMemRequest#(asz))

writeData → Get#(MemData#(dsz))

writeDone → Put#(Bit#(MemTagSize))

interface PhysMemReadServer (*numeric type asz*, *numeric type dsz*)

readReq → Put#(PhysMemRequest#(asz))

readData → Get#(MemData#(dsz))

interface PhysMemWriteServer (*numeric type asz*, *numeric type dsz*)

writeReq → Put#(PhysMemRequest#(asz))

writeData → Put#(MemData#(dsz))

writeDone → Get#(Bit#(MemTagSize))

2.5.4 Memory Clients and Servers

interface **MemReadClient** (*numeric type dsz*)

readReq → Get#(MemRequest)

readData → Put#(MemData#(dsz))

interface **MemWriteClient** (*numeric type dsz*)

writeReq → Get#(MemRequest)

writeData → Get#(MemData#(dsz))

writeDone → Put#(Bit#(MemTagSize))

interface **MemReadServer** (*numeric type dsz*)

readReq → Put#(MemRequest)

readData → Get#(MemData#(dsz))

interface **MemWriteServer** (*numeric type dsz*)

writeReq → Put#(MemRequest)

writeData → Put#(MemData#(dsz))

writeDone → Get#(Bit#(MemTagSize))

2.5.5 Memory Engine Types

struct **MemengineCmd**

A read or write request for a MemreadEngine or a MemwriteEngine. Memread and Memwrite engines will issue one or more burst requests to satisfy the overall length of the request.

sglId → SGLId
Which scatter gather list the MMU should use to translate the addresses

base → Bit#(MemOffsetSize)
Logical base address of the request, as a byte offset

burstLen → Bit#(BurstLenSize)
Maximum burst length, in bytes.

len → Bit#(32)
Number of bytes to transfer. Must be a multiple of the data bus width.

tag → Bit#(MemTagSize)
Identifier for this request.

2.5.6 Memory Engine Interfaces

interface **`MemwriteServer`** (*numeric type dataWidth*)

`cmdServer` → Server#(MemengineCmd,Bool)

`dataPipe` → PipeIn#(Bit#(dataWidth))

interface **`MemwriteEngineV`** (*numeric type dataWidth*, *numeric type cmdQDepth*, *numeric type numServers*)

`dmaClient` → MemWriteClient#(dataWidth)

`writeServers` → Vector#(numServers, Server#(MemengineCmd,Bool))

`dataPipes` → Vector#(numServers, PipeIn#(Bit#(dataWidth)))

`write_servers` → Vector#(numServers, MemwriteServer#(dataWidth))

typedef **`MemwriteEngineV#(dataWidth,cmdQDepth,1) MemwriteEngine#(numeric type dataWidth, numer`**

interface **`MemreadServer`** (*numeric type dataWidth*)

`cmdServer` → Server#(MemengineCmd,Bool)

`dataPipe` → PipeOut#(Bit#(dataWidth))

interface **`MemreadEngineV`** (*numeric type dataWidth*, *numeric type cmdQDepth*, *numeric type numServers*)

`dmaClient` → MemReadClient#(dataWidth)

`readServers` → Vector#(numServers, Server#(MemengineCmd,Bool))

`dataPipes` → Vector#(numServers, PipeOut#(Bit#(dataWidth)))

`read_servers` → Vector#(numServers, MemreadServer#(dataWidth))

typedef **`MemreadEngineV#(dataWidth,cmdQDepth,1) MemreadEngine#(numeric type dataWidth, numeric`**

2.5.7 Memory Traffic Interfaces

interface **`DmaDbg`**

`getMemoryTraffic` → ActionValue#(Bit#(64))

`dbg` → ActionValue#(DmaDbgRec)

2.5.8 Connectable Instances

instance **`Connectable`** (*MemReadClient#(dsz)*, *MemReadServer#(dsz)*)

instance **`Connectable`** (*MemWriteClient#(dsz)*, *MemWriteServer#(dsz)*)

instance **`Connectable`** (*PhysMemMaster#(addrWidth*, *busWidth)*, *PhysMemSlave#(addrWidth*, *busWidth)*)

instance **`Connectable`** (*PhysMemMaster#(32*, *busWidth)*, *PhysMemSlave#(40*, *busWidth)*)

2.6 Portal Package

2.6.1 PipePortal Interface

interface `Portal::`**`PipePortal`** (*numeric type numRequests*, *numeric type numIndications*, *numeric type slaveDataWidth*)

`messageSize` (*Bit#(16) methodNumber*) → Bit#(16)
Returns the message size of the methodNumber method of the portal.

`requests` → Vector#(numRequests, PipeIn#(Bit#(slaveDataWidth)))

`indications` → Vector#(numIndications, PipeOut#(Bit#(slaveDataWidth)))

2.6.2 MemPortal Interface

interface `Portal::`**`MemPortal`** (*numeric type slaveAddrWidth*, *numeric type slaveDataWidth*)

`slave` → PhysMemSlave#(slaveAddrWidth,slaveDataWidth)

`interrupt` → ReadOnly#(Bool)

`top` → WriteOnly#(Bool)

function `Portal::`**`getSlave`** (*MemPortal#(_a, _d) p*) → PhysMemSlave(_a,_d)

function `Portal::`**`getInterrupt`** (*MemPortal#(_a, _d) p*) → ReadOnly#(Bool)

function `Portal::`**`getInterruptVector`** (*Vector#(numPortals, MemPortal#(_a, _d)) portals*) → Vector#(16,ReadOnly#(Bool))

2.6.3 ShareMemoryPortal Interface

interface `Portal::`**`SharedMemoryPortal`** (*numeric type dataBusWidth*)
Should be in SharedMemoryPortal.bsv

`readClient` → MemReadClient(dataBusWidth)

`writeClient` → MemWriteClient#(dataBusWidth)

`cfg` → SharedMemoryPortalConfig

`interrupt` → ReadOnly#(Bool)

2.6.4 ConnectalTop Interface

interface `Portal::`**`ConnectalTop`** (*numeric type addrWidth*, *numeric type dataWidth*, *type pins*, *numeric type numMasters*)
Interface ConnectalTop is the interface exposed by the top module of a Connectal hardware design.

`slave` → PhysMemSlave#(32,32)

`masters` → Vector#(numMasters,PhysMemMaster#(addrWidth, dataWidth))

`interrupt` → Vector#(16,ReadOnly#(Bool))

`leds` → LEDS

`pins` → pins

2.6.5 StdConnectalTop Typedef

typedef `Portal::StdConnectalTop` (numeric type addrWidth) → ConnectalTop#(addrWidth,64,Empty,0)

Type StdConnectalTop indicates a Connectal hardware design with no user defined pins and no user of host shared memory. The "pins" interface is Empty and the number of masters is 0.

typedef `Portal::StdConnectalDmaTop` (numeric type addrWidth) → ConnectalTop#(addrWidth,64,Empty,1)

Type StdConnectalDmaTop indicates a Connectal hardware design with no user defined pins and a single client of host shared memory. The "pins" interface is Empty and the number of masters is 1.

CHAPTER

THREE

CONNECTAL EXAMPLES

3.1 Simple Example

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *pkgindex*
- *search*

Symbols

B

C

D

G

H

L

M

N

P

S

Z