Bluespec SystemVerilog™
Design Example
TLMDMA
TLMDMA.bsv

- DMA based on TLM interfaces
- TLM slave receives read/write requests towards the configuration registers
- TLM master initiates a DMA transaction (read/write requests) to bus according to configuration registers
- Bypass FIFOs store read/write transactions
DMA Architecture

slave
TLMReadWriteRecvIFC

mkTLMDMA

mkTLMCBusAdapterToReadWrite
adapter

cfg
mkDMAConfigRegs

fifo_channel

local status registers

mkReg

fifo_write

mkSizedFIFO(16)

fifo_read

master
TLMReadWriteSendIFC

rx  tx  rx  tx
The DMA Interfaces

- Has master and slave subinterfaces
- Receives configuration requests through the slave interface
- Initiates DMA read/write requests and receives responses through the master interface

```c
interface TLMDMAIFC#(`TLM_TYPE_PRMS);
    interface TLMReadWriteSendIFC#(`TLM_TYPES) master;
    interface TLMReadWriteRecvIFC#(`TLM_TYPES) slave;
endinterface
```
Read and Write FIFOs

- **Master read FIFOs**
  - fifo_read_tx: Request FIFO for reads
  - fifo_read_rx: Response FIFO for reads

- **Master write FIFOs**
  - fifo_write_tx: Request FIFO for writes
  - fifo_read_rx: Response FIFO for writes

```
FIFO#(TLMRequestStd) fifo_read_tx <- mkBypassFIFO;
FIFO#(TLMResponseStd) fifo_read_rx <- mkBypassFIFO;

FIFO#(TLMRequestStd) fifo_write_tx <- mkBypassFIFO;
FIFO#(TLMResponseStd) fifo_write_rx <- mkBypassFIFO;
```
Channel FIFO

- Connects read response with write request
- Buffers read response data for the write side
- Allows for multiple concurrent read and write requests

```plaintext
FIFO#(TLMData#(`TLM_STD_TYPES)) fifo_channel <- mkSizedFIFO(16);
```
Local State Registers

- idle: DMA must be in idle (idle = True) for transfer to begin
- done: Set to False at start, set to True when transfer is complete
- reads_remaining: Holds number of reads remaining in the transfer
- writes_remaining: Holds number of writes remaining in the transfer
- addr: Holds the current destination address

```plaintext
Reg#(Bool) idle <- mkReg(True);
Reg#(Bool) done <- mkReg(False);
Reg#(DescLen) reads_remaining <- mkReg(0);
Reg#(DescLen) writes_remaining <- mkReg(0);
Reg#(TLMAddr#(`TLM_STD_TYPES)) addr <- mkReg(0)
```
Rules

- **start_transfer**: initiates local registers, sets idle to False
- **data_read**: initiates a read request
- **data_write**: initiates a write request
- **finish_transfer**: sets done to False, idle to true, resets configuration registers
- **get_read_response_data**: moves response data into channel FIFO
- **get_write_response_data**: removes response from fifo_write_rx
Rules

slave

master

mk TLMDMA

get_read_response_data

data_read

start_transfer

finish_transfer

data_write

get_write_response_data

rx

tx

mx

lx

L

M

D

M

A

start_transfer

finish_transfer
Testbench

- Instantiate source (mkTLMSource): generates random requests
- Instantiate DMA (mkTLMDMA)
- Instantiate reducer (mkTLMReducer): transforms requests to single reads and writes
- Connect source and dma.slave
- Instantiate RAM (mkTLMReadWriteRAM)
- Connect reducer, dma, and RAM
- Continue for 1000 cycles
```
mkTestDMA

source

random requests

slave

mkTLMDMA

master

stream of DMA read and write requests

reducer

DMA response

RAM

dsingle DMA read and write requests
```
module mkTestDMA ();

...  
TLMReadWriteSendIFC#(`TLM_STD_TYPES) source <- mkTLMSource;
TLMDMAIFC#(`TLM_STD_TYPES) dma <- mkTLMDMA;

TLMTransformIFC#(`TLM_STD_TYPES) reducer <- mkTLMReducer;

mkConnection(source, dma.slave);

TLMReadWriteRecvIFC#(`TLM_STD_TYPES) mem <- mkTLMReadWriteRam(0);

mkConnection(dma.master.read, reducer.in);
mkConnection(reducer.out, mem.read);
mkConnection(dma.master.write, mem.write);

...  
endmodule