

1 TLM

Packages

```
import TLM :: * ;
```

Description

The TLM package includes definitions of interfaces, data structures, and module constructors which allow users to create and modify bus-based designs in a manner that is independent of any one specific bus protocol. Bus operations are defined in terms of generic bus payload data structures. Other protocol specific packages include transactor modules that convert a stream of TLM bus operations into corresponding operations in a specific bus protocol. Designs created using the TLM package are thus more portable (because that they allow the core design to be easily applied to multiple bus protocols). In addition, since the specific signalling details of each bus protocol are encapsulated in pre-designed transactors, users are not required to learn, re-implement, and re-verify existing standard protocols.

Data Structures

The two basic data structures defined in the TLM package are `TLMRequest` and `TLMResponse`. By using these types in a design, the underlying bus protocol can be changed without having to modify the interactions with the TLM objects.

TLMRequest A TLM request contains either control information and data, or data alone. A `TLMRequest` is tagged as either a `RequestDescriptor` or `RequestData`. A `RequestDescriptor` contains control information and data while a `RequestData` contains only data.

```
typedef union tagged {RequestDescriptor#('TLM_TYPES) Descriptor;  
                      RequestData#('TLM_TYPES) Data;  
                      } TLMRequest#('TLM_TYPE_PRMS) deriving(Eq, Bits, Bounded);  
  
typedef TLMRequest#('TLM_STD_TYPES) TLMRequestStd;
```

RequestDescriptor The table below describes the components of a `RequestDescriptor` and the valid values for each of its members.

RequestDescriptor		
Member Name	Data Type	Valid Values
<code>command</code>	<code>TLMCommand</code>	READ, WRITE, UNKNOWN
<code>mode</code>	<code>TLMMode</code>	REGULAR, DEBUG, CONTROL
<code>addr</code>	<code>TLMAddr#('TLM_TYPES)</code>	<code>Bit#(addr_size)</code>
<code>data</code>	<code>TLMData#('TLM_TYPES)</code>	<code>Bit#(data_size)</code>
<code>burst_length</code>	<code>TLMUInt#('TLM_TYPES)</code>	<code>UInt#(uint_size)</code>
<code>byte_enable</code>	<code>TLMByteEn#('TLM_TYPES)</code>	<code>Bit#(TDiv#(data_size, 8))</code>
<code>burst_mode</code>	<code>TLMBurstMode</code>	INCR, CNST, WRAP, UNKNOWN
<code>burst_size</code>	<code>TLMBurstSize#('TLM_TYPES)</code>	<code>Bit#(TLog#(TDiv#(data_size, 8)))</code>
<code>prty</code>	<code>TLMUInt#('TLM_TYPES)</code>	<code>UInt#(uint_size)</code>
<code>lock</code>	<code>Bool</code>	
<code>thread_id</code>	<code>TLMId#('TLM_TYPES)</code>	<code>Bit#(id_size)</code>
<code>transaction_id</code>	<code>TLMId#('TLM_TYPES)</code>	<code>Bit#(id_size)</code>
<code>export_id</code>	<code>TLMId#('TLM_TYPES)</code>	<code>Bit#(id_size)</code>
<code>custom</code>	<code>TLMCustom#('TLM_TYPES)</code>	<code>cstm_type</code>

```

typedef struct {TLMCommand          command;
                TLMMode             mode;
                TLMAddr#('TLM_TYPES) addr;
                TLMDData#('TLM_TYPES) data;
                TLMUInt#('TLM_TYPES) burst_length;
                TLMByteEn#('TLM_TYPES) byte_enable;
                TLMBurstMode         burst_mode;
                TLMBurstSize#('TLM_TYPES) burst_size;
                TLMUInt#('TLM_TYPES) prty;
                Bool                 lock;
                TLMId#('TLM_TYPES) thread_id;
                TLMId#('TLM_TYPES) transaction_id;
                TLMId#('TLM_TYPES) export_id;
                TLMCustom#('TLM_TYPES) custom;
                } RequestDescriptor#('TLM_TYPE_PRMS) deriving (Eq, Bits, Bounded);

```

RequestData The table below describes the components of a RequestData and the valid values for its members.

RequestData		
Member Name	DataType	Valid Values
data	TLMDData#('TLM_TYPES)	Bit#(data_size)
transaction_id	TLMId#('TLM_TYPES)	Bit#(id_size)
custom	TLMCustom#('TLM_TYPES)	cstm_type

```

typedef struct {TLMDData#('TLM_TYPES) data;
                TLMId#('TLM_TYPES) transaction_id;
                TLMCustom#('TLM_TYPES) custom;
                } RequestData#('TLM_TYPE_PRMS) deriving (Eq, Bits, Bounded);

```

TLMResponse The table below describes the components of a TLMResponse and the valid values for its members.

TLMResponse		
Member Name	DataType	Valid Values
command	TLMCommand	READ, WRITE, UNKNOWN
data	TLMDData#('TLM_TYPES)	Bit#(data_size)
status	TLMStatus	SUCCESS, ERROR, NO_RESPONSE
prty	TLMUInt#('TLM_TYPES)	UInt#(uint_size)
thread_id	TLMId#('TLM_TYPES)	Bit#(id_size)
transaction_id	TLMId#('TLM_TYPES)	Bit#(id_size)
export_id	TLMId#('TLM_TYPES)	Bit#(id_size)
custom	TLMCustom#('TLM_TYPES)	cstm_type

```

typedef struct {TLMCommand          command;
                TLMDData#('TLM_TYPES) data;
                TLMStatus           status;
                TLMUInt#('TLM_TYPES) prty;
                TLMId#('TLM_TYPES) thread_id;

```

```

TLMId#('TLM_TYPES)      transaction_id;
TLMId#('TLM_TYPES)      export_id;
TLMCustom#('TLM_TYPES) custom;
} TLMResponse#('TLM_TYPE_PRMS) deriving (Eq, Bits, Bounded);

```

```
typedef TLMResponse#('TLM_STD_TYPES) TLMResponseStd;
```

Configurable Parameters

In the above BSV code definitions the compiler macros 'TLM_TYPE_PRMS and 'TLM_TYPES are used in the `typedef` statements. A 'define statement is a preprocessor construct used to place prepackaged text values into a file. In this case, the macros contain parameters to be used in the data definitions. Placing the parameters in a separate file allows them to be easily modified for different protocol requirements. For convenience, we have predefined a few useful definitions for use in the TLM package.

The TLM_TYPE_PRMS macro contains type definition parameters which are used in the interface definitions or as arguments to TLM types and interfaces.

The TLM_TYPES macro is used when providing the interface or using the data type. TLM_TYPES is still polymorphic.

The macro TLM_STD_TYPES provides specific values for the polymorphic values defined above. The values defined in TLM_STD_TYPES are common values. The user can change any of the values or define other corresponding macros (with different values) as appropriate for a given design.

The macros are found in the file `TLM.defines`. A sample of the contents of the file are displayed below.

```

'define TLM_TYPE_PRMS numeric type id_size, numeric type addr_size, \
                        numeric type data_size, numeric type uint_size, type cstm_type
'define TLM_TYPES id_size, addr_size, data_size, uint_size, cstm_type
'define TLM_STD_TYPES 4, 32, 32, 10, Bit#(0)

```

Interfaces

The TLM interfaces define how TLM blocks interconnect and communicate. The TLM package includes two basic interfaces: The `TLMSendIFC` interface and the `TLMRecvIFC` interface. These interfaces use basic `Get` and `Put` subinterfaces as the requests and responses. The `TLMSendIFC` interface generates (`Get`) requests and receives (`Put`) responses. The `TLMRecvIFC` interface receives (`Put`) requests and generates (`Get`) responses. Additional TLM interfaces are built up from these basic blocks.

TLMSendIFC The `TLMSendIFC` interface transmits the requests and receives the responses.

TLMSendIFC Interface		
Name	Type	Description
<code>tx</code>	<code>Get#(TLMRequest#('TLM_TYPES))</code>	Transmits a request through the <code>Get</code> interface
<code>rx</code>	<code>Put#(TLMResponse#('TLM_TYPES))</code>	Receives a response through the <code>Put</code> interface

```

interface TLMSendIFC#('TLM_TYPE_PRMS);
  interface Get#(TLMRequest#('TLM_TYPES)) tx;
  interface Put#(TLMResponse#('TLM_TYPES)) rx;
endinterface

```

TLMRecvIFC The TLMRecvIFC interface receives the requests and transmits the responses.

TLMRecvIFC Interface		
Name	Type	Description
tx	<code>Get#(TLMResponse#('TLM_TYPES))</code>	Transmits the response through the Get interface
rx	<code>Put#(TLMRequest#('TLM_TYPES))</code>	Receives the request through the Put interface

```
interface TLMRecvIFC#('TLM_TYPE_PRMS);
  interface Get#(TLMResponse#('TLM_TYPES)) tx;
  interface Put#(TLMRequest#('TLM_TYPES)) rx;
endinterface
```

As illustrated in Figure 1, a TLMSendIFC is connectable to a TLMRecvIFC, just as a **Get** is connectable to a **Put**. A transmitted request (**tx**) from a TLMSendIFC is received (**rx**) by the TLMRecvIFC and visa versa.

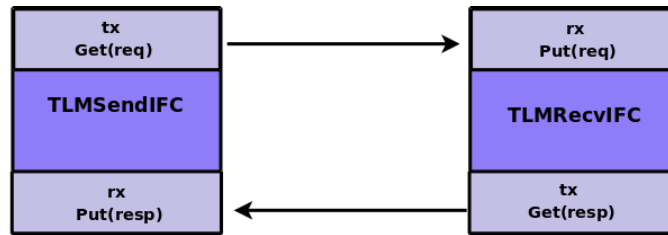


Figure 1: Connecting TLM Send And Receive Interfaces

```
instance Connectable#(TLMSendIFC#('TLM_TYPES), TLMRecvIFC#('TLM_TYPES));
```

A module with a TLMSendIFC interface creates a stream of requests. A module with a TLMRecvIFC interface receives the requests and transmits responses. Some bus protocols have separate channels for read and write operations. In these cases it is useful to have interfaces which bundle together two sends or two receives. The TLMReadWriteSendIFC interface includes two send interfaces while the TLMReadWriteRecvIFC interface bundles two receives.

TLMReadWriteSendIFC The TLMReadWriteSendIFC interface is composed of two TLMSendIFC subinterfaces, one for a read channel and one for a write channel.

```
interface TLMReadWriteSendIFC#('TLM_TYPE_PRMS);
  interface TLMSendIFC#('TLM_TYPES) read;
  interface TLMSendIFC#('TLM_TYPES) write;
endinterface
```

TLMReadWriteRecvIFC The TLMReadWriteRecvIFC interface is composed of two TLMRecvIFC subinterfaces, one for a read channel and one for a write channel.

```
interface TLMReadWriteRecvIFC#('TLM_TYPE_PRMS);
  interface TLMRecvIFC#('TLM_TYPES) read;
  interface TLMRecvIFC#('TLM_TYPES) write;
endinterface
```

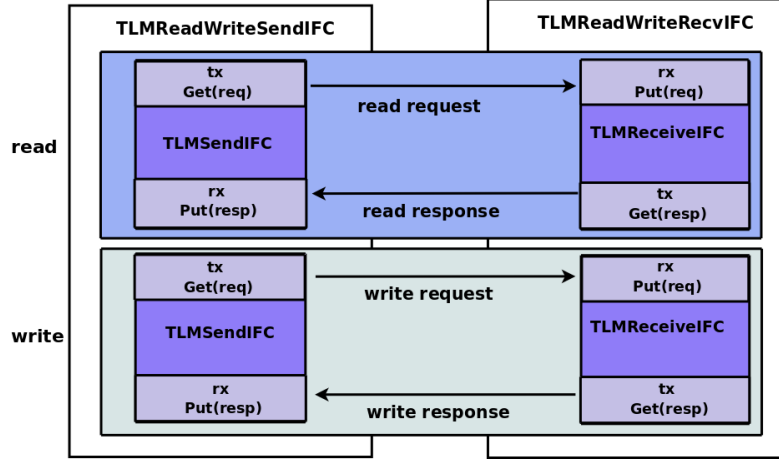


Figure 2: TLM Read/Write Interfaces

As illustrated in Figure 2, the **TLMReadWriteSendIFC** and **TLMReadWriteRecvIFC** interfaces are connectable as well.

```
instance Connectable#(TLMReadWriteSendIFC#('TLM_TYPES), TLMReadWriteRecvIFC#('TLM_TYPES));
```

TLMTransformIFC The **TLMTransformIFC** provides a single **TLMRecvIFC** interface and a single **TLMSendIFC** interface. This interface is useful in modules which convert one stream of TLM operations into another. It is the interface provided by **mkTLMReducer** module for instance.

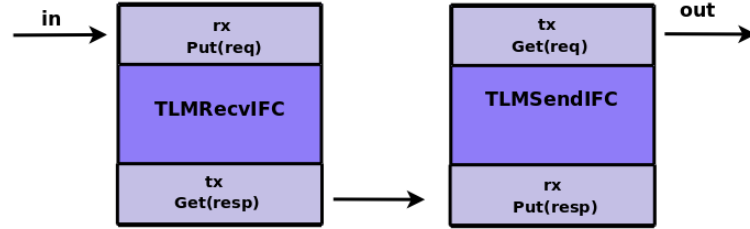


Figure 3: TLMTransformIFC Interface

```
interface TLMTransformIFC#('TLM_TYPE_PRMS);
  interface TLMRecvIFC#('TLM_TYPES) in;
  interface TLMSendIFC#('TLM_TYPES) out;
endinterface
```

Modules

The TLM package includes modules for creating and modifying TLM objects: **mkTLMLRandomizer**, **mkTLMSource**, and **mkTLMReducer**. Two TLM RAM modules are also provided: **mkTLMRam** which provides a single read/write port and **mkTLMReadWriteRam** which provides two ports, a separate one for reads and a separate one for writes.

mkTLMRandomizer	<p>Creates a stream of random TLM operations. The argument <code>m_command</code> is a <code>Maybe</code> type which determines if the <code>TLMRequests</code> will be reads, writes, or both. A value of <code>Valid READ</code> will generate only reads, a value of <code>Valid WRITE</code> will generate only writes, and an <code>Invalid</code> value will generate both reads and writes. The <code>Randomize</code> interface is defined in the <code>Randomizable</code> package.</p> <pre> module mkTLMRandomizer#(Maybe#(TLMCommand) m_command) (Randomize#(TLMRequest#('TLM_TYPES))) provisos(Bits#(RequestDescriptor#('TLM_TYPES), s0), Bounded#(RequestDescriptor#('TLM_TYPES)), Bits#(RequestData#('TLM_TYPES), s1), Bounded#(RequestData#('TLM_TYPES))); </pre>
------------------------	--

mkTLMSource	<p>Creates a wrapper around the <code>mkTLMRandomize</code> module. The provided interface is now a <code>TLMSendIFC</code> interface which both sends <code>TLMRequests</code> and receives <code>TLMResponses</code>. The argument <code>m_command</code> has the same meaning as in <code>mkTLMRandomizer</code>. The <code>verbose</code> argument controls whether or not <code>\$display</code> outputs are provide when sending and receiving TLM objects.</p> <pre> module mkTLMSource#(Maybe#(TLMCommand) m_command, Bool verbose) (TLMSendIFC#('TLM_STD_TYPES)); </pre>
--------------------	---

mkTLMReducer	<p>Converts a stream of (arbitrary) TLM operations into a stream with only single reads and single writes.</p> <pre> module mkTLMReducer (TLMTransformIFC#('TLM_TYPES)) provisos(Bits#(TLMRequest#('TLM_TYPES), s0), Bits#(TLMResponse#('TLM_TYPES), s1), Bits#(RequestDescriptor#('TLM_TYPES), s2)); </pre>
---------------------	--

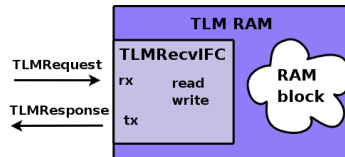


Figure 4: TLMRAM

mkTLMRam	<p>Creates a TLM RAM with a single port for read and write operations. Provides the TLMRecvIFC interface. The verbose argument controls whether or not \$display output is provided when performing a memory operation. The id argument provides an identifier for the instantiation which is used in the \$display output if the verbose flag is asserted.</p> <pre> module mkTLMRam#(parameter Bit#(4) id, Bool verbose) (TLMRecvIFC#('TLM_TYPES)) provisos(Bits#(TLMRequest#('TLM_TYPES), s0), Bits#(TLMResponse#('TLM_TYPES), s1)); </pre>
-----------------	--

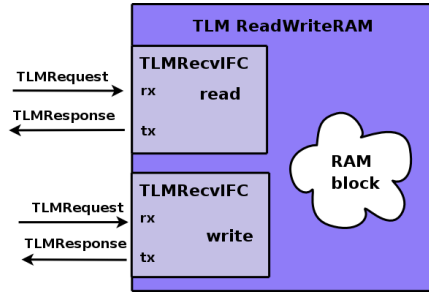


Figure 5: TLMReadWriteRAM

mkTLMReadWriteRam	<p>Creates a RAM with separate ports for read and write operations. Provides the TLMReadWriteRecvIFC interface. The verbose argument controls whether or not \$display output is provided when performing a memory operation. The id argument provides an identifier for the instantiation which is used in the \$display output if the verbose flag is asserted.</p> <pre> module mkTLMReadWriteRam#(parameter Bit#(4) id, Bool verbose) (TLMReadWriteRecvIFC#('TLM_TYPES)) provisos(Bits#(TLMRequest#('TLM_TYPES), s0), Bits#(TLMResponse#('TLM_TYPES), s1)); </pre>
--------------------------	--

The **mkTLMCbusAdapter** module creates an adapter which allows the CBus to be accessed via a TLM interface.

mkTLMCBusAdapter	Takes a TLMCBus interface as an argument. Provides the TLMRecvIFC interface.
	<pre> module mkTLMCBusAdapter#(TLMCBus#('TLM_TYPES, caddr_size) cfg) (TLMRecvIFC#('TLM_TYPES)) provisos(Bits#(TLMRequest#('TLM_TYPES), s0), Bits#(TLMResponse#('TLM_TYPES), s1), Add#(ignore, caddr_size, addr_size)); </pre>

mkTLMCBusAdapterToReadWrite	Takes a TLMCBus interface as an argument. Provides the TLMReadWriteRecvIFC interface. This configuration provides separate ports for read and write operations.
	<pre> module mkTLMCBusAdapterToReadWrite# (TLMCBus#('TLM_TYPES, caddr_size) cfg) (TLMReadWriteRecvIFC#('TLM_TYPES)) provisos(Bits#(TLMRequest#('TLM_TYPES), s0), Bits#(TLMResponse#('TLM_TYPES), s1), Add#(ignore, caddr_size, addr_size)); </pre>

Functions

createBasicRequestDescriptor	Returns a generic TLM request with default values.
	<pre> function RequestDescriptor#('TLM_TYPES) createBasicRequestDescriptor() provisos(Bits#(RequestDescriptor#('TLM_TYPES), s0)); </pre>

createBasicTLMResponse	Returns a generic TLM response with default values.
	<pre> function TLMResponse#('TLM_TYPES) createBasicTLMResponse() provisos(Bits#(TLMResponse#('TLM_TYPES), s0)); </pre>