1 AHB

Packages

```
import AHB :: * ;
```

Description

The AHB library includes interface, transactor, module and function definitions to implement the AHB protocol with Bluespec SystemVerilog. The BSV AHB library groups the AHB data and protocols into reusable, parameterized interfaces, which interact with TLM interfaces. An AHB bus is implemented using AHB transactors - interfaces which connect TLM interfaces on one side with AHB interfaces on the other side.

The AHB library supports the following AHB Bus protocol features:

- Basic and Burst Transfers
- Locked Transfers

The AHB library does not support the following AHB Bus protocol features:

- Early Burst Termination
- Split Transfers
- Retry Transfers

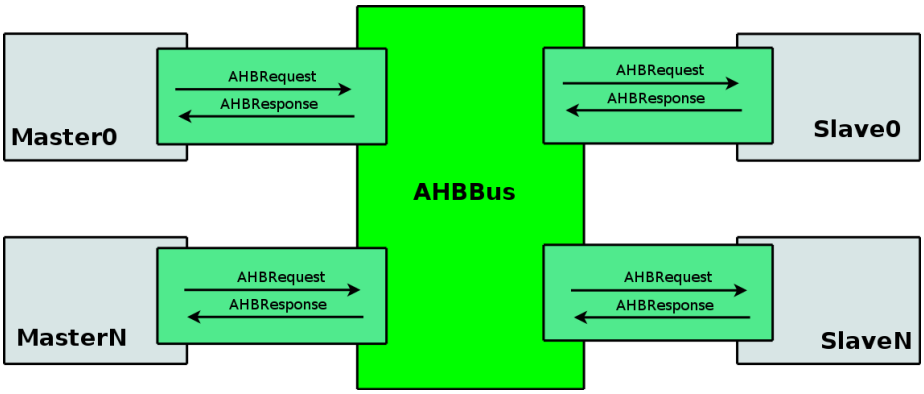

Figure 1: AHB Bus Example

Data Structures

Inside the transactor modules, the AHB data is organized into the following data structures: the address and control information is defined by `AHBCtrl`, the write data is defined by `AHBData`. These two structures are bundled into an `AHBRequest`. Finally, the response data is defined by `AHBResponse`.

AHBRequest An AHB request is defined by the `AHBRequest` structure as described below.

AHBRequest		
Member	DataType	Valid Values
cntrl	`AHBCtrl#(‘TLM_PRM)`	see above
data	`AHBData#(‘TLM_PRM)`	`Bit#(data_size)`

```
typedef struct {
                AHBCtrl#(‘TLM_PRM)    ctrl;
                AHBData#(‘TLM_PRM)    data;
               } AHBRequest#(‘TLM_PRM_DCL) ‘dv;
```

AHBCtrl The control fields in an `AHBRequest` are described by the `AHBCtrl` structure, the components of which are defined in the following table.

AHBCtrl		
Member	DataType	Valid Values
command	`AHBWrite`	READ, WRITE
size	`AHBSize`	BITS8, BITS16, BITS32, BITS64, BITS128, BITS256, BITS512, BITS1024
burst	`AHBBurst`	SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, INCR16
transfer	`AHBTransfer`	IDLE, BUSY, NONSEQ, SEQ
prot	`AHBProt`	`Bit#(4)`
addr	`AHBAddr#(‘TLM_PRM)`	`Bit#(addr_size)`

```
typedef struct {
                AHBWrite             command;
                AHBSize              size;
                AHBBurst             burst;
                AHBTransfer          transfer;
                AHBProt              prot;
                AHBAddr#(‘TLM_TYPES) addr;
               } AHBCtrl#(‘TLM_PRM_DCL) ‘dv;
```

AHBResponse An `AHBResponse` consists of a status fields and data (when responding to a read request). The components of the structure are described in the following table.

AHBResponse		
Member	DataType	Valid Values
status	`AHBResp`	OKAY, ERROR, RETRY, SPLIT
data	`AHBData`	`Bit#(data_size)`
command	`Maybe#(AHBWrite)`	READ, WRITE

```
typedef struct {
                AHBResp              status;
                AHBData#(‘TLM_PRM) data;
                Maybe#(AHBWrite)     command;
               } AHBResponse#(‘TLM_PRM_DCL) ‘dv;
```

Bus Interfaces

The two basic bus interfaces included in the AHB library are the `AHBMaster` interface and the `AHBSlave` interface.

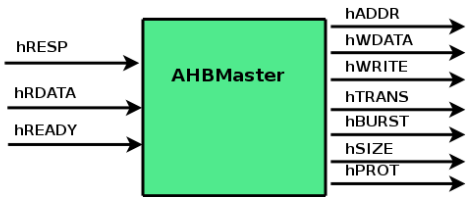

Figure 2: AHB Master Interface

AHBMaster The `AHBMaster` interface issues AHB requests and receives AHB responses.

```
(* always_ready, always_enabled *)
interface AHBMaster#(`TLM_PRM_DCL);
   // Outputs
   (* result = "HADDR" *)
   method AHBAddr#(`TLM_PRM)  hADDR;
   (* result = "HWDATA" *)
   method AHBData#(`TLM_PRM)  hWDATA;
   (* result = "HWRITE" *)
   method AHBWrite              hWRITE;
   (* result = "HTRANS" *)
   method AHBTransfer           hTRANS;
   (* result = "HBURST" *)
   method AHBBurst              hBURST;
   (* result = "HSIZE" *)
   method AHBSize               hSIZE;
   (* result = "HPROT" *)
   method AHBProt               hPROT;
   // Inputs
   (* prefix = "", result = "unused0" *)
   method Action      hRDATA((* port = "HRDATA" *) AHBData#(`TLM_PRM) data);
   (* prefix = "", result = "unused1" *)
   method Action      hREADY((* port = "HREADY" *) Bool value);
   (* prefix = "", result = "unused2" *)
   method Action      hRESP((* port = "HRESP" *) AHBResp response);
endinterface
```

AHBSlave The `AHBSlave` interface receives AHB requests and returns AHB responses.

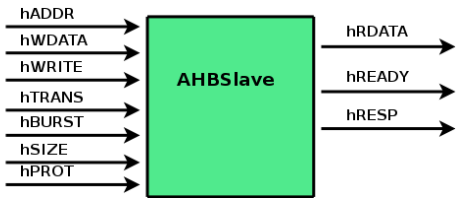

Figure 3: AHB Slave Interface

```
(* always_ready, always_enabled *)
interface AHBSlave#(`TLM_PRM_DCL);
```

```
    // Inputs
   (* prefix = "", result = "unused0" *)
   method Action      hADDR((* port = "HADDR" *)    AHBAddr#(`TLM_PRM) addr);
   (* prefix = "", result = "unused1" *)
   method Action      hWDATA((* port = "HWDATA" *)  AHBData#(`TLM_PRM) data);
   (* prefix = "", result = "unused2" *)
   method Action      hWRITE((* port = "HWRITE" *)  AHBWrite    value);
   (* prefix = "", result = "unused3" *)
   method Action      hTRANS((* port = "HTRANS" *)  AHBTransfer value);
   (* prefix = "", result = "unused4" *)
   method Action      hBURST((* port = "HBURST" *)  AHBBurst    value);
   (* prefix = "", result = "unused5" *)
   method Action      hSIZE((* port = "HSIZE" *)    AHBSize     value);
   (* prefix = "", result = "unused6" *)
   method Action      hPROT((* port = "HPROT" *)    AHBProt     value);

   // Outputs
   (* result = "HRDATA" *)
   method AHBData#(`TLM_PRM) hRDATA;
   (* result = "HREADY" *)
   method Bool               hREADY;
   (* result = "HRESP" *)
   method AHBResp            hRESP;
endinterface
```

The `AHBMaster` and `AHBSlave` interfaces are connectable.

```
instance Connectable#(AHBMaster#(`TLM_PRM), AHBSlave#(`TLM_PRM));
```

Fabric Interfaces

When used in the context of a bus or switch, AHB Master and Slave modules must communicate with the arbiter and with address decoding logic. Two additional interfaces are provided to support this communication.

AHBMasterArbiter The `AHBMasterArbiter` interface connects the master module with the bus arbiter. Through this interface, the master can request control of the bus and determine when control has been granted.

```
(* always_ready, always_enabled *)
interface AHBMasterArbiter;
   (* result = "HBUSREQ" *)
   method Bool        hBUSREQ;
   (* result = "HLOCK" *)
   method Bool        hLOCK;
   (* prefix = "" *)
   method Action      hGRANT((* port = "HGRANT" *) Bool value);
endinterface
```

AHBMasterArbiterDual

```
(* always_ready, always_enabled *)
interface AHBMasterArbiterDual;
   (* prefix = "", result = "unused7" *)
   method Action      hBUSREQ((* port = "HBUSREQ" *) Bool value);
   (* prefix = "", result = "unused8" *)
   method Action      hLOCK((* port = "HLOCK" *)     Bool value);
   (* result = "HGRANT" *)
   method Bool hGRANT;
endinterface
```

AHBSlaveSelector The `AHBSlaveSelector` interface provides an `addrMatch` method which given an AHB address returns an Boolean value indicating whether the given address maps to the associated slave. By polling this method for each slave on the bus, the decoding logic can determine the appropriate destination for each bus transaction. The `AHBSlaveSelector` interface also provides a `select` method by which the decoding logic can indicate which slave is the selected destination.

```
interface AHBSlaveSelector#(`TLM_PRM_DCL);
   method Bool   addrMatch(AHBAddr#(`TLM_PRM) value);
   (* prefix = "" *)
   method Action select((* port = "HSEL" *) Bool value);
endinterface
```

AHBFabricMaster The `AHBFabricMaster` interface bundles two subinterfaces, an `AHBMaster` interface and an `AHBMasterArbiter` interface. It is this interface that is provided as an argument when constructing an AHB bus and as the bus side interface of an AHB master transactor module.

```
interface AHBFabricMaster#(`TLM_PRM_DCL);
   (* prefix = "" *)
   interface AHBMaster#(`TLM_PRM)  bus;
   (* prefix = "" *)
   interface AHBMasterArbiter        arbiter;
endinterface
```

AHBFabricSlave The `AHBFabricSlave` interface bundles two subinterfaces, an `AHBSlave` interface and an `AHBSlaveSelector` interface. It is this interface that is provided as an argument when constructing an AHB bus and as the bus side interface of an AHB slave transactor module

```
interface AHBFabricSlave#(`TLM_PRM_DCL);
   (* prefix = "" *)
   interface AHBSlave#(`TLM_PRM)         bus;
   (* prefix = "" *)
   interface AHBSlaveSelector#(`TLM_PRM) selector;
endinterface
```

Transactor Interfaces

An AHB transactor module provides AHB and TLM interfaces to implement a translation between a stream of TLM operations and the AHB bus protocol. Each transactor has two subinterfaces: a subinterface for the connection with the AHB bus and a subinterface to send and receive TLM objects.

The AHB library package includes two transactor interfaces; The `AHBMasterXActor` interface for the master and `AHBSlaveXActor` interface for the slave. The AHB protocol doesn't separate read and write transactions, so there is a single transactor implementation for masters and a single implementation for slaves.

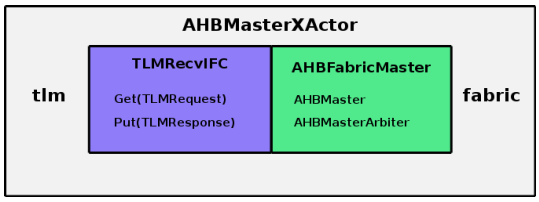

Figure 4: AHBMasterXActor Interface

AHBMasterXActor The `AHBMasterXActor` has two subinterfaces: an `AHBFabricMaster` subinterface and a `TLMRecvIFC` subinterface. The TLM interface is described in the TLM package. The transactor converts TLM requests into the AHB protocol, and converts the AHB response back into TLM.

```
interface AHBMasterXActor#(`TLM_RR_DCL, `TLM_PRM_DCL);
   interface TLMRecvIFC#(`TLM_RR)      tlm;
   (* prefix = "" *)
   interface AHBFabricMaster#(`TLM_PRM) fabric;
endinterface
```

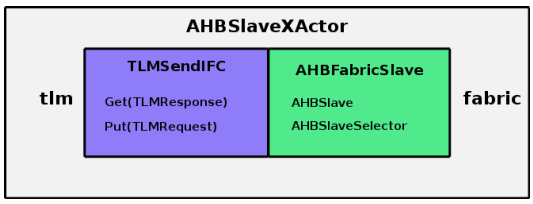

Figure 5: AHBSlaveXActor Interface

AHBSlaveXActor The `AHBSlaveXActor` has two subinterfaces: `AHBFabricSlave` subinterface and a `TLMSendIFC` subinterface. The TLM interface is described in the TLM package. The transactor converts an AHB request into TLM and the TLM response back into the AHB protocol.

```
interface AHBSlaveXActor#(`TLM_RR_DCL, `TLM_PRM_DCL);
   interface TLMSendIFC#(`TLM_RR)     tlm;
   (* prefix = "" *)
   interface AHBFabricSlave#(`TLM_PRM) fabric;
endinterface
```

Modules

The following constructors are used to create AHB transactor modules. Versions with associated synthesis boundaries are also available. These versions are called `mkAHBMasterStd`, and `mkAHBSlaveStd`. The specific TLM parameter values for these synthesized versions are as specified by the preprocessor macro `TLM_STD_TYPES`.

mkAHBMaster	Creates an AHB Master transactor module. Provides a AHBMasterXActor interface. This version is polymorphic.
	module mkAHBMaster (AHBMasterXActor#(‘TLM_RR, ‘TLM_PRM)) provisos(TLMRequestTC#(req_t, ‘TLM_PRM), TLMResponseTC#(resp_t, ‘TLM_PRM), DefaultValue#(TLMResponse#(‘TLM_PRM)), Bits#(req_t, s0), Bits#(resp_t, s1), Bits#(RequestDescriptor#(‘TLM_PRM), s2), AHBConvert#(AHBProt, cstm_type), AHBConvert#(AHBResp, cstm_type));

mkAHBMasterStd	Creates an AHB Master transactor module. Provides a AHBMasterXActor interface.
	module mkAHBMasterStd (AHBMasterXActor#(‘TLM_RR_STD, ‘TLM_PRM_STD));

mkAHBSlave	Creates an AHB Slave transactor module. Provides an AHBSlaveXActor interface. This version is polymorphic.
	module mkAHBSlave#(function Bool addr_match(AHBAddr#(‘TLM_PRM) addr)) (AHBSlaveXActor#(‘TLM_RR, ‘TLM_PRM)) provisos(TLMRequestTC#(req_t, ‘TLM_PRM), TLMResponseTC#(resp_t, ‘TLM_PRM), DefaultValue#(RequestDescriptor#(‘TLM_PRM)), Bits#(req_t, s0), Bits#(resp_t, s1), AHBConvert#(AHBProt, cstm_type));

mkAHBSlaveStd	Creates an AHB Slave transactor module. Provides an AHBSlaveXActor interface. This version is not polymorphic.
	module mkAHBSlaveStd#(function Bool addr_match(AHBAddr#(‘TLM_PRM_STD) addr)) (AHBSlaveXActor#(‘TLM_RR_STD, ‘TLM_PRM_STD));

mkAHBSlaveDummy	This is the recipient of everything that doesn’t have a slave destination.
	module mkAHBSlaveDummy (AHBFabricSlave#(‘TLM_PRM));

The following module constructor is used to create an AHB bus fabric.

`mkAHBBus`	Given a vector of `AHBFabricMaster` interfaces and a vector of `AHBFabricSlave` interfaces, `mkAHBBus` creates an AHB bus fabric.
	`module mkAHBBus#(Vector#(master_count, AHBFabricMaster#(‘TLM_PRM)) masters, Vector#(slv_count, AHBFabricSlave#(‘TLM_PRM)) slvs) (Empty) provisos(Add#(slv_count, 1, slave_count));`

The following module is used to add probe signals for each of the AHB bus signals. This facilitates debugging and waveform viewing of the created bus fabric.

`mkAHBMasterMonitor`	Adds a probe module for each of the AHB bus signals. The `include_pc` value indicates whether or not the monitor module should include an instantiation of an AHB protocol checker module (available from ARM). If the protocol checker is not available, the value of `include_pc` should be set to False.
	`module mkAHBMasterMonitor#(AHBFabricMaster#(‘TLM_PRM) master) (AHBMasterMonitor#(‘TLM_PRM));`

`getCurrentSlave`	Returns the `slave_num` of the current slave.
	`module getCurrentSlave#(AHBFabricMaster#(‘TLM_PRM) master, Vector#(slave_count, AHBFabricSlave#(‘TLM_PRM)) slaves) (ReadOnly#(LBit#(slave_count)));`