ARM®

AHB Example AMBA SYstem – ARM DUI 0092C **Addendum 01**

This addendum document details the implementation of AHB BusMatrix, which is an additional component in *Chapter 5 AHB Synthesis* in the *Example AMBA SYstem* (EASY) User Guide.

Table 1 Text additions

Page	Insert location	Inserted text
5-9	Insert after Section 5.3 Synthesizing new AHB modules	See attached text

June 2001

1 AHB BusMatrix

The bus matrix allows a number of AHB layers to communicate with a number of AHB shared slaves. The advantage of using the BusMatrix is that it provides parallel access paths between the various AHB layers and the shared slaves, giving improved overall system bandwidth.

1.1 Configuration

Different systems require different sizes of AHB BusMatrix and therefore the RTL design allows for this. The base design allows for the maximum size configuration, which is eight layer input ports and eight shared slave output ports. To allow for smaller sizes of BusMatrix, pragmas are included throughout the RTL to allow for the automatic generation of any configuration, within the following parameters:

- from two to eight layer input ports
- from one to eight shared slave output ports.

1.2 Structure

Figure 1 on page 3 shows the overall structure of the BusMatrix.

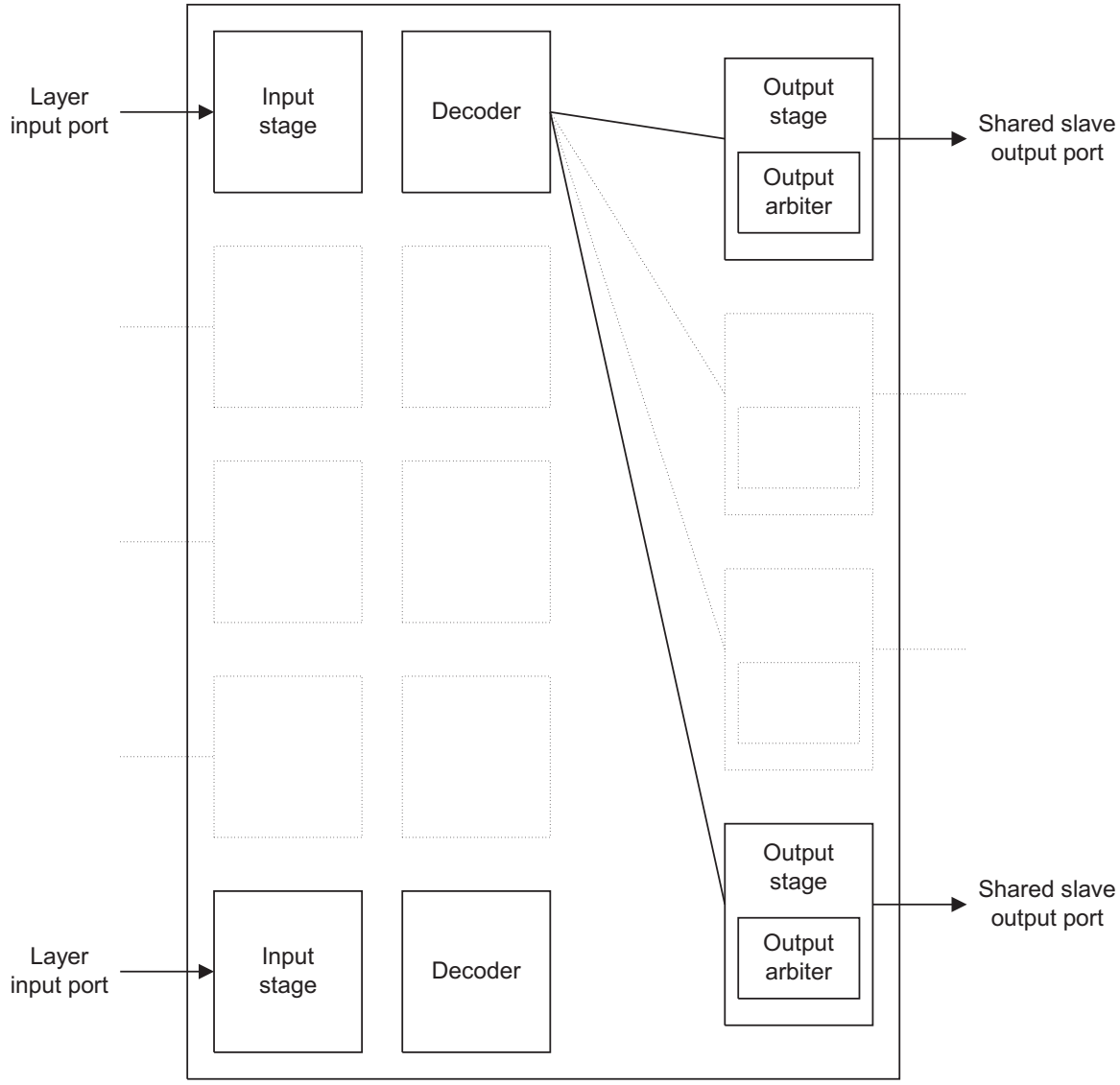

Figure 1 Top level block diagram

The design is made up of the following blocks:

Input stage There is one input stage for each layer input port. The input stage is responsible for holding the address and control information when the transfer to a shared slave is not able to commence immediately.

Decode stage Each layer input port also has a decoder associated with it. The decoder determines which shared slave a transfer is destined for.

Output stage Each shared slave has an output stage which is used to select which of the various input layers is routed to the slave.

Output arbiter Each output stage contains an output arbiter. The arbiter looks at which of the input stages has to perform a transfer to the shared slave and decides which is currently the highest priority.

Top level The top level of the BusMatrix connects each input stage or decoder to all the output stages.

1.3 Directory structure

The multi-layer AHB BusMatrix is supplied in the directory structure shown in Figure 2.

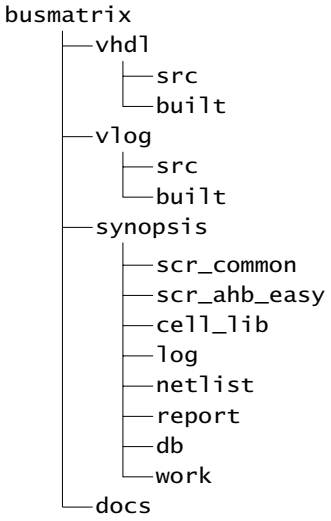

Figure 2 BusMatrix directory structure

The following directories are supplied:

- The vhdl and vlog directories contain the RTL for VHDL and Verilog respectively. In each of these areas, the full source for an 8-input, 8-output BusMatrix is in the src directory and a configuration script is supplied to allow any version of the BusMatrix to be generated in the built directory.
- The synopsys area contains all the Synopsys synthesis scripts that are required to synthesize the design. To perform synthesis on a version of the BusMatrix, use the run_BusMatrix.csh script.
- The docs directory contains the documentation that accompanies the BusMatrix.

1.4 Configuration

The BusMatrix is designed to allow for all configurations of input and output ports, up to a maximum of eight:

- from two to eight input ports
- from one to eight output ports
- round-robin or fixed arbitration.

This gives a total of 112 different configurations ($7 \times 8 \times 2$). A script called configmatrix.pl is supplied to allow the construction of any version that is required.

The script is located in both the vhdl and vlog directories. To obtain information on the usage of the script, run the script with the help switch:

```
> configmatix.pl --help
```

This prints the following information:

```
Purpose:  Builds particular configurations of the BusMatrix
          component.
Usage:
   Builds a BusMatrix component with a given number of input
   ports <inports>, a given number of output ports <outports>
   and a particular arbitration scheme.
Options:
   --inports=NUM           Number of input ports (2,3..8)
   --outports=NUM          Number of output ports (1,2..8)
   --arb=SCHEME            Arbitration scheme (f - fixed,
                           r - round-robin)
   --all                   Builds all possible configurations
   --verbose               Prints progress information
   --help                  Prints this help
```

To run the script and build all possible variants use the following command:

```
> configmatrix.pl --all --verbose
```

When run, the script generates the RTL for the required BusMatrix in the `built` directory. The name of the directory has the maximum number of the input and outputs as found in the RTL and this is one less than might be expected, because the ports are numbered from zero up to the maximum number. For example, a BusMatrix with three input ports and five output ports is found in the directory named `input2_by_output4_fixed`.

If only a single configuration of the BusMatrix is required, the script can be run to only build that option. Three parameters are required:

- number of input ports
- number of output ports
- arbitration scheme, which can be either fixed priority or round-robin

To build a BusMatrix with three input ports, five output ports, and a fixed priority arbitration scheme, use the following command:

```
> configmatrix.pl --inports=3 --outports=5 --arb==fixed
```

Source code pragmas

The different configurations are generated from a source design that is constructed for the maximum configuration of eight input ports and eight output ports. Throughout this design pragmas are used to indicate the portions of code that can be removed for smaller BusMatrix configurations.

For example, the code required for just one output port (port number 0) is followed by the pragma

```
-- busswitch output0
```

the code required for two output ports (up to port number 1) is followed by the pragma

```
-- busswitch output1
```

and so on, until finally the code required for all output ports (up to port number 7) is followed by the pragma

```
-- busswitch output7
```

Therefore, to generate the code required for a two output port configuration the script is used to remove all the RTL code between the `-- busswitch output1` and `-- busswitch output7` pragmas.

1.5 Design description

This section describes each of the design modules in detail.

Input stage

Figure 3 shows the input stage. All of the ports on the left side of the diagram are connected to the input layer and all the ports on the right of the diagram are connected to either the decoder or output stages within the BusMatrix.

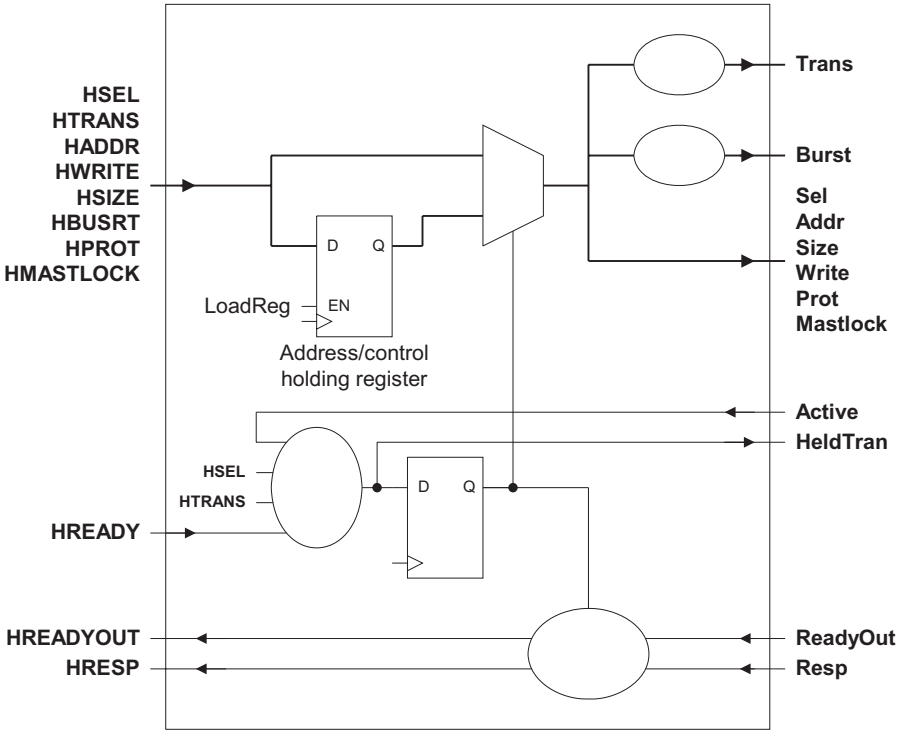

Figure 3 Input stage

The main function of the input stage is to hold the address and control information from the input layer if the transfer cannot be passed immediately to the appropriate shared slave. This is required because in the AHB protocol the duration of the address phase is controlled by the slave. The slave was accessed by the previous transfer and therefore the BusMatrix cannot extend the address phase of the transfer if the required shared slave is not available.

The loading of the holding register is controlled by the **Active** signal. Each output stage generates a set of **Active** signals, one per input stage, which indicates that the address/control signals from a given input stage are currently being driven on to the required shared slave.

Whenever a transfer arrives at the input stage it is either passed directly to the output stage, if **Active** is HIGH, or it is loaded in to the holding register if **Active** is LOW. The multiplexor in the address/control signal path simply selects the holding register when it is loaded or the straight-through path when it is empty.

The **HeldTran** signal is generated within the input stage and indicates if the holding register is full or empty. To be more precise, the **HeldTran** signal indicates if the holding register is full or empty in the following cycle. This signal is not only used within this block, but is also routed to the output arbitration block, because it shows that the input stage has a transfer that is ready to start.

The second main function of the input stage is to generate the **HREADYOUT** and **HRESP** signals for the input layer and this is done as follows:

- When a transfer has been routed to the appropriate output stage the **HREADYOUT** and **HRESP** are generated from the equivalent signals at the output stage.
- When a transfer is stored in the holding register **HREADYOUT** is driven LOW to stall the transfer and **HRESP** indicates OKAY.
- When the input stage is not being accessed or for an IDLE or BUSY transfer, **HREADYOUT** is driven HIGH and **HRESP** indicates OKAY, as required by the AHB protocol.

The final function provided by the input stage is shown in Figure 3 on page 7 by the two logic bubbles on the **HTRANS** and **HBURST** paths as they leave the input stage. These two patches of logic are used to override the transfer type information and the burst information if a fixed length burst to a shared slave is interrupted before it reaches completion. If this happens, the burst information is changed to indicate an INCR undefined length burst. The transfer type signals are only overridden to NONSEQ, if a wrapping burst has been changed to an INCR burst and has crossed the wrap boundary.

Decoder

Each input stage has a decoder associated with it, which is used to determine the output stage that is required to complete an access. Because the address map of every system can be different, the main address decode function of the decoder can be changed if required. Within the RTL this section of code, which converts from the incoming address bus to an output port number **AddrOutPort**, is located at the top of the file. Figure 4 on page 9 shows the decoder.

By default, the decoder is supplied with each of the output ports occupying 16MB of address space, that is, **HADDR[26:24]** is used to determine the output port required.

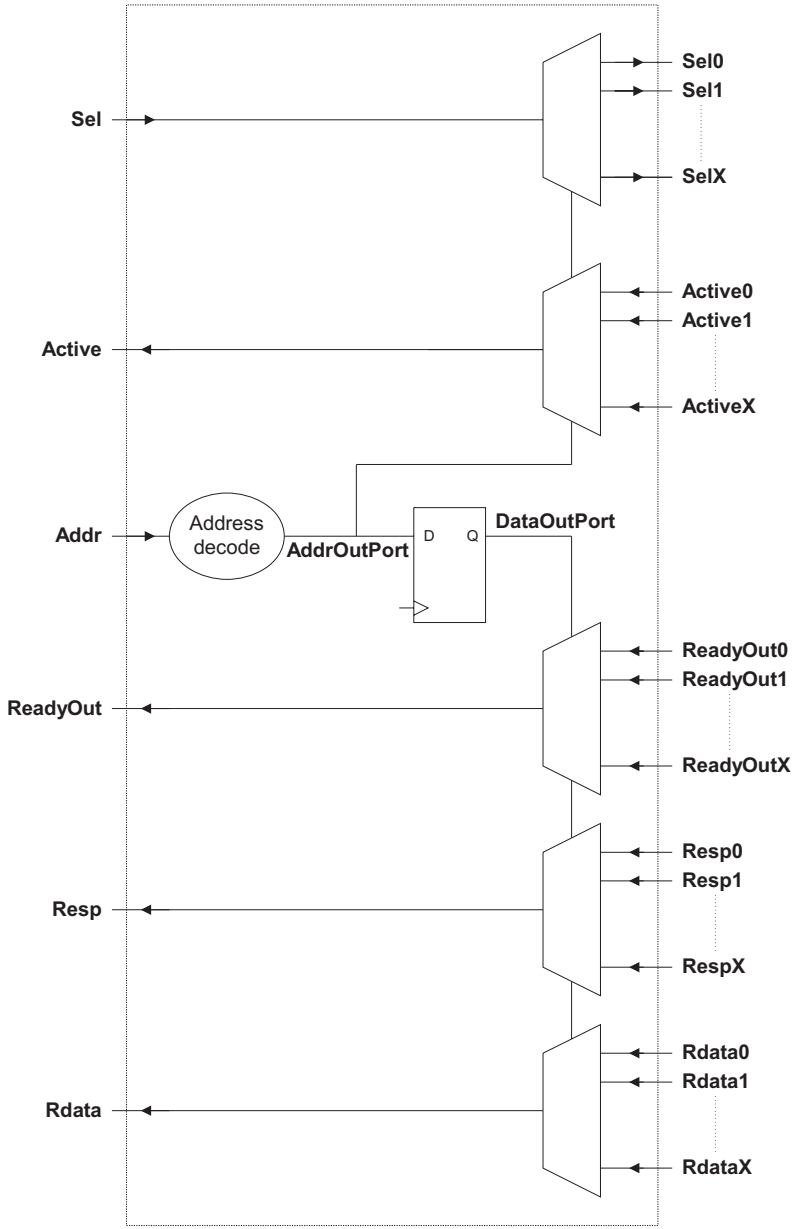

Figure 4 Decoder

Within the decoder, **AddrOutPort** is used to show which output port the current address and control information is destined for. **DataOutPort** is used to show which output port is being used for the data phase of the previous transfer.

AddrOutPort is used for two routing functions. Firstly, assuming that the input **Sel** signal is HIGH, **AddrOutPort** is used to determine which output stage select signal must be asserted. Only one output **Sel** signal can be asserted at a time. Each output stage has a **Sel** signal from each input layer and the output stage arbitration can use this to determine which input stages has to perform a transfer.

The second use of **AddrOutPort** is to route the appropriate **Active** signal back to the input stage. The **Active** signal indicates that the address of the input port is being actively driven to the shared slave, so the transfer does not have to be held in the input stage.

Each time an address phase completes, as indicated by **HREADY** being HIGH, the data phase of the transfer commences. Whenever **HREADY** is HIGH, the output port number in **AddrOutPort** is moved to **DataOutPort** to indicate the output port required to complete the transfer.

Within the decoder **DataOutPort** is then used to select the **HREADYOUT**, **HRESP**, and **HRDATA** from the appropriate output stage and route these back to the input stage.

If an input layer accesses one shared slave and then immediately follows this with an access to a different shared slave, the output port indicated by **AddrOutPort** is different from the output port indicated by **DataOutPort**.

Output stage

Each output stage gives access to a shared slave. In reality there can be more than one shared slave connected to an output stage, but from the perspective of the BusMatrix this is not important and it can treat multiple shared slaves on the same port as just one slave (see *Address decoding strategies* on page 14 for more information on connecting multiple shared slaves to one output stage).

Each output stage has two main functions:

- it contains an output arbitration block to decide which input stage is given access to the shared slave
- it contains a set of routing multiplexors.

Figure 5 on page 11 shows the output stage.

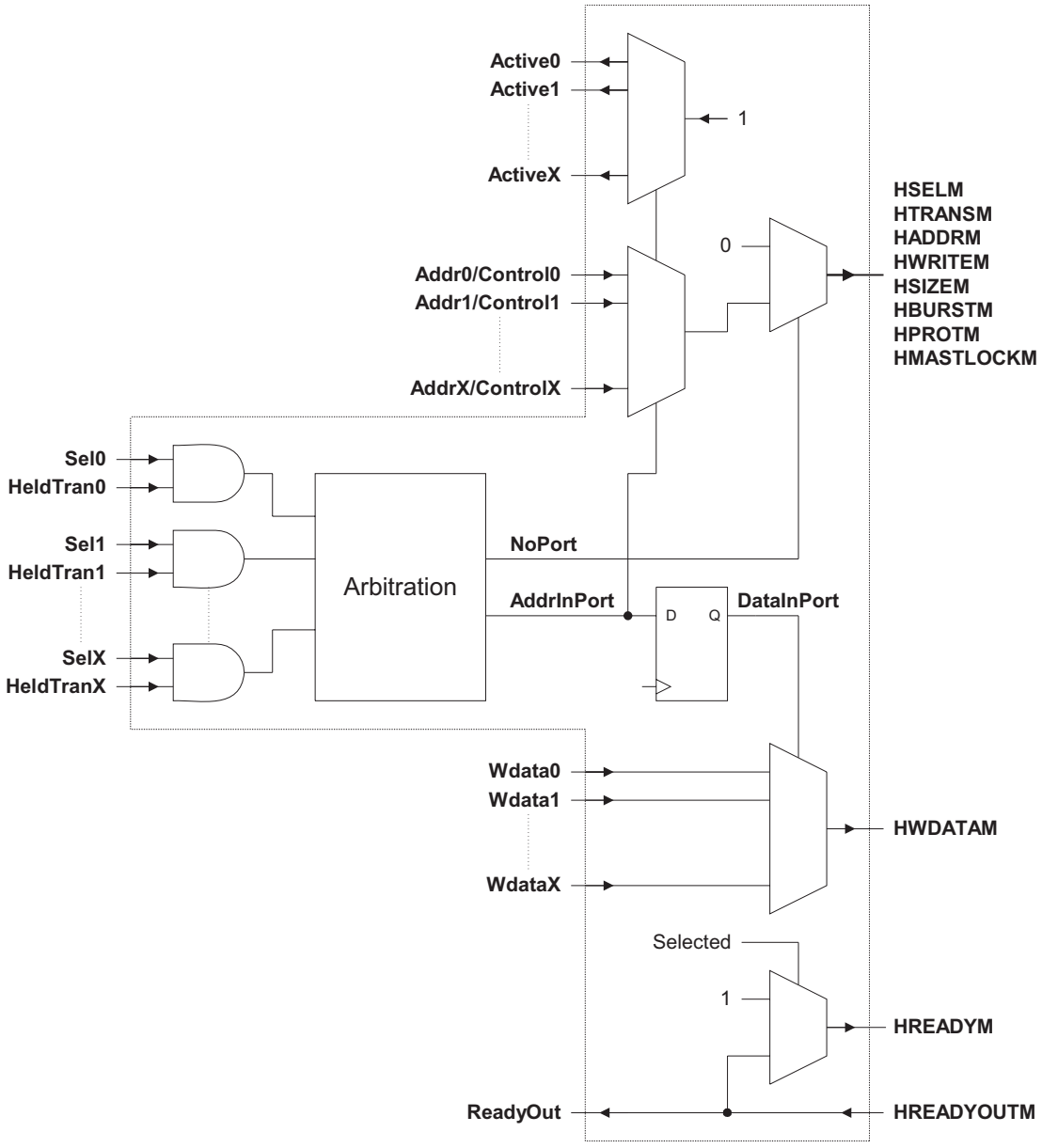

Figure 5 Output stage

The main routing multiplexors in the output stage are the address/control signals multiplexor, which is controlled directly using **AddrInPort**. The write data multiplexor is controlled using the data phase signal, **DataInPort**.

The address/control signals multiplexor is followed by another multiplexor, which drives all the address/control output signals to inactive levels when no input stages are selected, as indicated by the **NoPort** signal.

The two other functions contained within the output stage are the generation of the **Active** signals and the generation of the **HREADY** signal for the shared slave:

- The **Active** signals indicate back to the various input stages which one is currently driving out to the shared slave. Only one **Active** signal is asserted at any time.
- The **HREADY** signal for the shared slave is generated from the **HREADYOUT** signal of the slave when it is selected and at all other times it is driven HIGH.

Output arbitration

The **Request** signals for the output arbitration are generated within the output stage by ANDing the **HeldTran** signal with the **Select** signal from each input stage. The **HeldTran** signal shows that there is a transfer ready to commence and the **Select** signal shows that it is destined for this particular output stage.

It is important that the arbitration process uses the **HeldTran** signal, rather than just using **HTRANS(1)**, because the **HeldTran** signal also includes the fact that the **HREADY** signal on the input layer is HIGH. This is important because otherwise a transfer can be routed to the shared slave before it is has actually started on the input layer.

Within the output arbitration block all of the **Request** signals are combined to work out which input stage must be used for the next transfer. The arbitration process follows four steps:

1. If **HMASTLOCK** is asserted, the same input stage remains selected.
2. If **HMASTLOCK** is not asserted, all the different requests are examined and the highest priority input stage is selected. The algorithm that is used to choose between the input stages can be user-modified and two different schemes, fixed priority and round-robin, are supplied.
3. If no input stage is requesting access and the currently selected input stage is performing Idle transfers to the shared slave, that is, the **Sel** signal is still asserted, then the same input stage is selected.

4. If none of the above apply, the **NoPort** signal is asserted, which indicates that none of the input stages must be selected and the address/control signals to the shared slave must be driven to an inactive state.

Figure 6 shows the output arbitration.

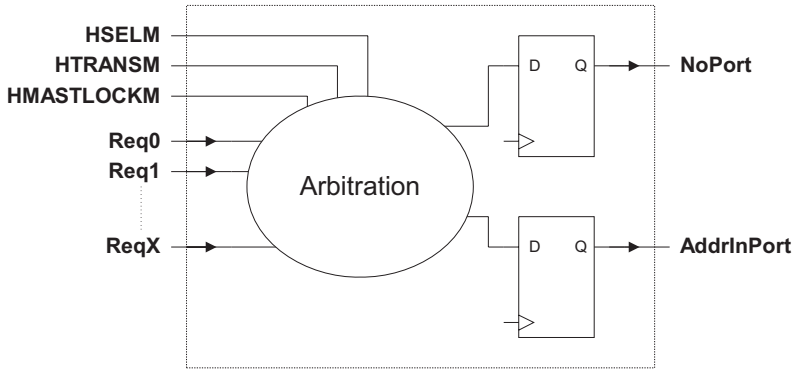

Figure 6 Output arbitration

The output arbitration registers the result of the arbitration process before passing this to the output stage multiplexors. This is done to avoid the critical path of attempting to:

- work out which input stages requires a transfer
- arbitrate between them
- switch the output multiplexors and still provide the address and control signals to the shared slave with adequate setup time.

However, this registering of the arbitration result can lead to a single cycle delay when an input layer first attempts access to a shared slave. The single cycle delay becomes hidden if the shared slave is already being accessed by another input layer.

The cycle delay only occurs on the first access to a shared slave. If an input layer performs a sequence of bursts to the shared slave, all of which are in the same address decode region then the single cycle penalty is only observed for the first access in the sequence.

It is possible in low clock frequency systems to remove the register stage. This removes the cycle delay that occurs to give an input stage access to the shared slave. However, this must only be done in circumstances where it is known that the critical arbitration timing path can be satisfied.

1.6 Address decoding strategies

Using a simple Multi-layer bus architecture with just one AHB master on each input layer and just one AHB slave on each output from the BusMatrix means that the entire system address decoding can be done within the decoder section of the BusMatrix.

However, typically more complex architectures are used to optimize the interconnect solution for a particular application and, in this case, an appropriate address decoding strategy has to be adopted.

Figure 7 shows the principle by showing an architecture that includes both local slaves on an input layer and multiple slaves on a single output port. The recommended approach is to use a simple decode at each stage, as shown.

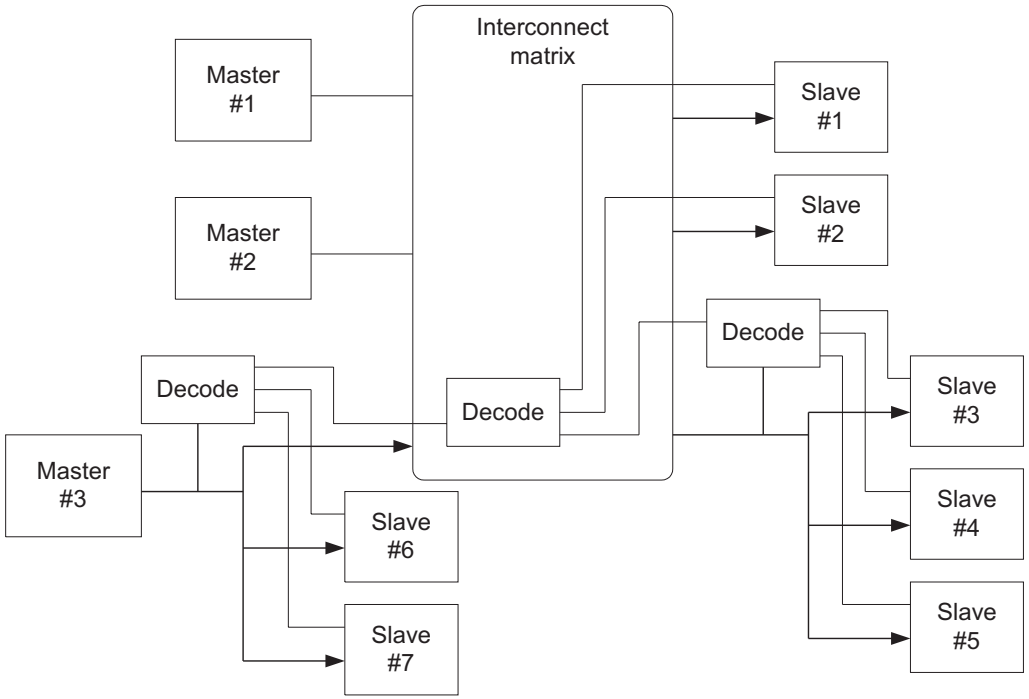

Figure 7 Address decode

1.7 Gate count optimization options

The BusMatrix can be configured for any number of input and output ports, up to a maximum of eight. This ensures that the size and complexity of the BusMatrix is well matched to that required for the target application. However, it is possible to make further small optimizations to the total size of the BusMatrix:

- If access to an output stage is not required from all input stages, it is possible to modify the top-level of the design so that the instantiation of the output stage has all of the signal connections to unrequired input stages tied off to appropriate levels. This should result in a gate count reduction during synthesis.
- If certain control signals are not used by any of the shared slaves, such as **HPROT**, **HBURST**, or possibly some of the address lines, it is possible that these signals can be tied-off at the top-level, again resulting in a gate count reduction during synthesis.

1.8 Additional arbitration schemes

Arbitration

The arbitration within the BusMatrix determines which input port has access to the shared slave and each shared slave has its own arbitration. Different arbitration schemes provide different system characteristics in terms of access latency and overall system performance.

The following arbitration schemes are supported by the slave switch:

- one port always has highest priority
- switch on every transfer if other port waiting.

The following arbitration schemes can also be considered for a particular application. However, these schemes are not supported and might require significant modification to the arbitration section of the switch:

- switch after *n* transfers if other port waiting
- switch if other port waiting for *n* transfers
- switch after *n* cycles if other port waiting
- switch at the end of fixed length bursts.

Locked transfers

The arbitration section of the BusMatrix ensures that when a master performs a sequence of locked transfer to a slave, no other master is allowed access to that slave until the master has completed the sequence.

Using a multi-layer AHB system requires certain restrictions to be placed on the use of locked transfers to prevent a deadlock situation. It is required that a sequence of locked transfers are all performed to the same slave within the system. A bus master can ensure this restriction is met by ensuring that a locked sequence of transfers remains within a 1KB address region.

1.9 Example timing diagrams

This section gives some example transfer sequences for the BusMatrix. For clarity, all transfer sequences shown are read operations, but the same cycle behavior occurs for writes.

Figure 8 shows simultaneous access from both ports with switching occurring every transfer. In this example, the shared slave inserts a wait state on the first transfer from each port, but all subsequent transfers are zero wait state.

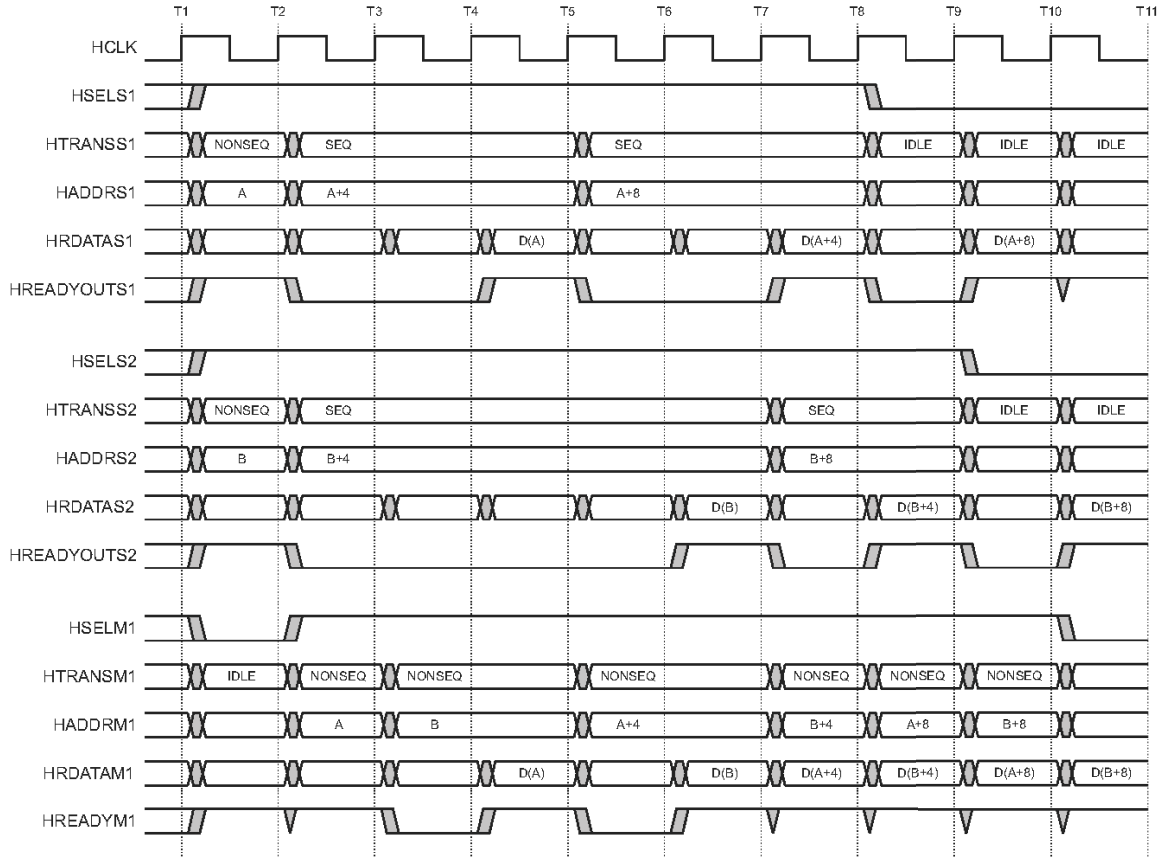

Figure 8 Parallel access timing diagram

Figure 9 shows a port accessing the shared slave when the other input port is idle. At the start of the sequence, a single wait state is inserted by the BusMatrix. This example shows the shared slave inserting one wait state for each of the first two accesses.

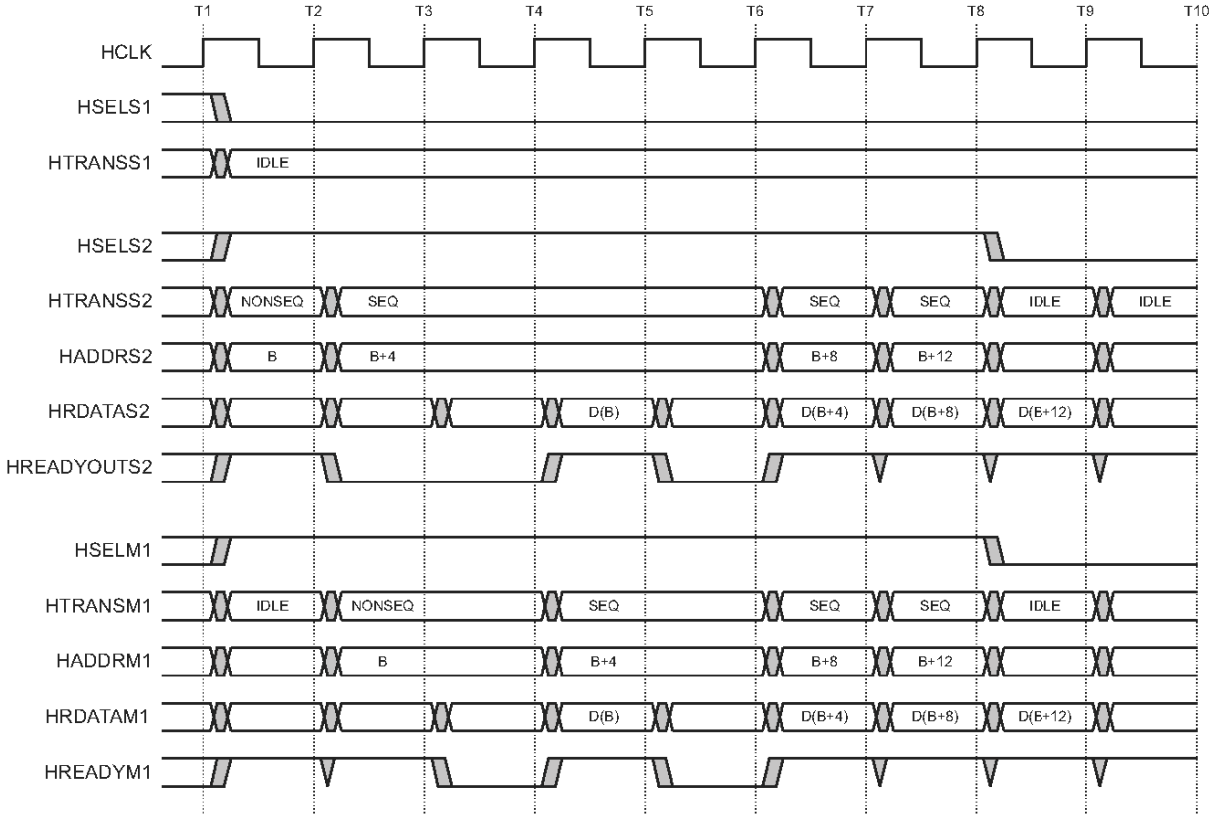

Figure 9 Single port access timing diagram

1.10 Interface diagram and signal list

Figure 10 shows the signal interface for the BusMatrix.

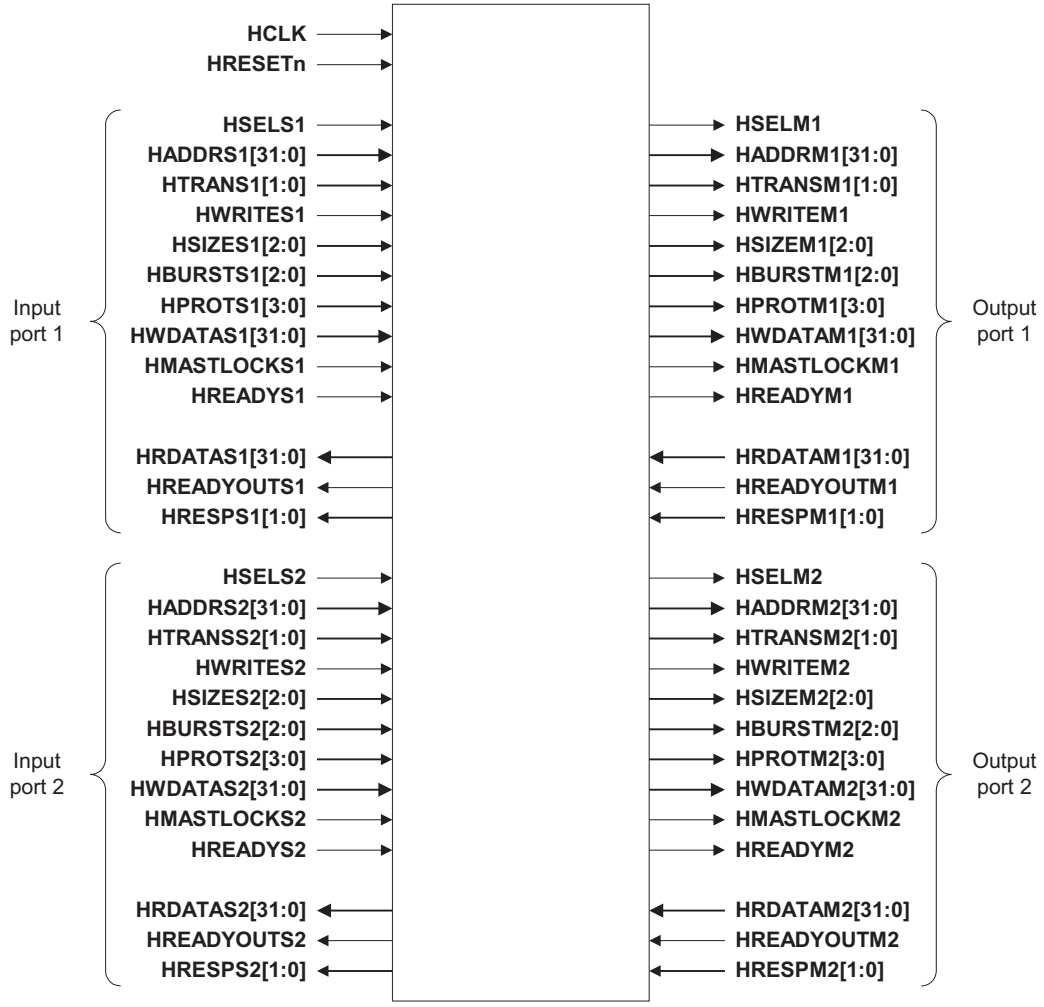

Figure 10 Signal interface diagram

Table 2 shows a list of the interface signals used by the BusMatrix. An identical set of signals can be found on each of the input ports and each of the slave output ports.

Table 2 BusMatrix interface signals

Name	Direction	Description
AMBA signals		
HCLK	Input	Bus clock
HRESETn	Input	Reset
Input port signals		
HSELSx	Input	Input port x select signal
HADDRSx[31:0]	Input	Input port x address
HTRANSSx[1:0]	Input	Input port x transfer type
HWRITESx	Input	Input port x transfer direction
HSIZESx[2:0]	Input	Input port x transfer size
HBURSTSx[2:0]	Input	Input port x burst type
HPROTSx[3:0]	Input	Input port x protection control
HWDATASx[31:0]	Input	Input port x write data
HMASTLOCKSx	Input	Input port x locked transfer
HREADYSx	Input	Input port x ready signal
HRDATASx[31:0]	Output	Input port x read data
HREADYOUTSx	Output	Input port x ready output
HRESPSx[1:0]	Output	Input port x response
Output port signals		
HSELMx	Output	Output port x select signal
HADDRMx[31:0]	Output	Output port x address
HTRANSMx[1:0]	Output	Output port x transfer type
HWRITEMx	Output	Output port x transfer direction
HSIZEMx[2:0]	Output	Output port x transfer size
HBURSTMx[2:0]	Output	Output port x burst type

Table 2 BusMatrix interface signals (continued)

Name	Direction	Description
HPROTMx[3:0]	Output	Output port x protection control
HWDATAMx[31:0]	Output	Output port x write data
HMASTLOCKMx	Output	Output port x locked transfer
HREADYMx	Output	Output port x ready signal
HRDATAMx[31:0]	Input	Output port x read data
HREADYOUTMx	Input	Output port x ready output
HRESPMx[1:0]	Input	Output port x response

1.11 AC characteristics

The BusMatrix conforms to the following AMBA timing parameters:

- combinatorial input to output paths less than 20% of **HCLK** cycle
- all outputs valid 60% of cycle after rising **HCLK**
- registered inputs setup 60% of cycle before rising **HCLK**.

The timing characteristics are confirmed by performing synthesis on the slave switch using the slow/slow process point of the Avanti cb25 cell library at a target speed of 100MHz.

1.12 Gate count

The gate count estimate depends on the configuration of the BusMatrix. Table 3 shows some examples of the boundaries.

Table 3 BusMatrix gate count

Matrix	Cell area	Interconnect area	Total area
2 input ports, 1 output port	1,300	600	1,900
2 input ports, 2 output pors	1,900	900	2,800
3 input ports, 3 output ports	3,200	1,700	4,900
8 input ports, 8 output ports	15,000	10,000	25,000

1.13 Test methodology

The AHB BusMatrix is designed for full scan test insertion. No dedicated test features are included.