

AMBA AXI Design

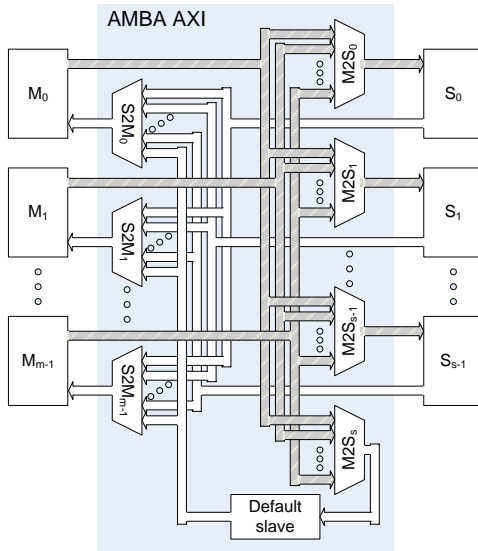
2013 – 2017 - 2018

Ando Ki
(adki@future-ds.com)

Agenda

- ❑ Structure of AMBA AXI
- ❑ Read channels
- ❑ Write channels
- ❑ Forward channels
- ❑ Backward channels
- ❑ AMAB AXI two-master and two-slave case

Structure of AMBA AXI

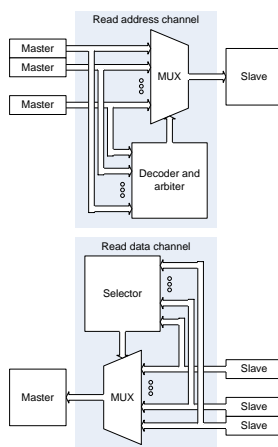


- ❑ Channel-based protocol
- ❑ 5 independent channels
- ❑ Single-clock edge operation: ACLK
- ❑ Two-way flow-control mechanism for each channel
- ❑ Valid/ready handshake mechanism
- ❑ Separate address/control and data phases
- ❑ Registers slices
- ❑ Easy to add register stages since each AXI channel transfers information in only one direction
- ❑ Burst-based protocol
- ❑ One address for burst
- ❑ Variable-length burst
- ❑ 1 to 16 data transfers per burst for AXI3; up to 256 for AXI4
- ❑ Multiple outstanding bursts
- ❑ Out-of-order transaction completion
- ❑ Data interleaving
- ❑ Lock and exclusive for atomic access in AXI3; only exclusive for atomic in AXI4

Copyright © 2013-2017-2018 by Ando Ki

Design AXI (3)

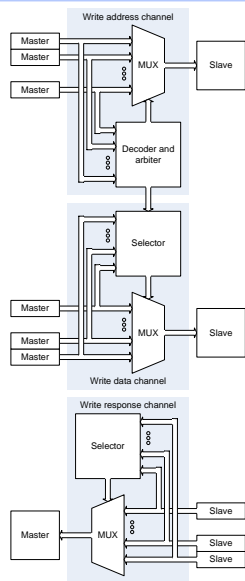
Read channels



Copyright © 2013-2017-2018 by Ando Ki

Design AXI (4)

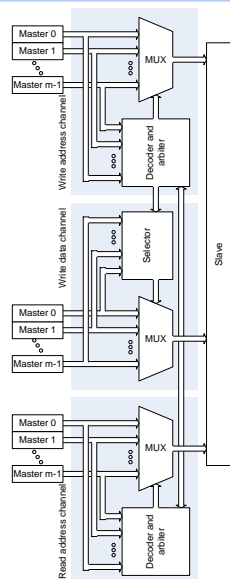
Write channels



Copyright © 2013-2017-2018 by Ando Ki

Design AXI (5)

Forward channels

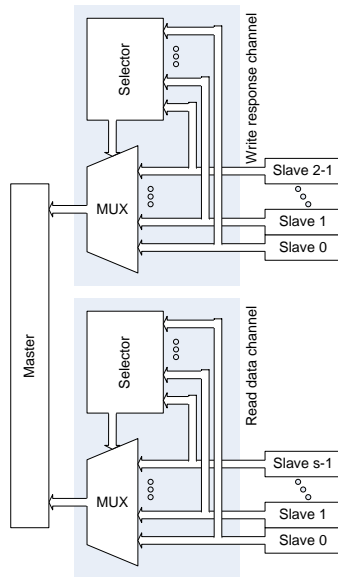


- Forward channels are dedicated to a specific slave and handles transactions driven by multiple masters. Write-address and read-address channel use address (AWADDR, ARADDR) to select a specific slave, while write-data channel uses information given by write-address channel.

Copyright © 2013-2017-2018 by Ando Ki

Design AXI (6)

Backward channels

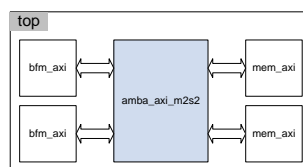


- Backward channels are dedicated to a specific master and handles transactions driven by multiple slaves. Write-response and read-data channels use tags (BID and RID) to select the master that is waiting for the transaction.

Copyright © 2013-2017-2018 by Ando Ki

Design AXI (7)

AMBA AXI two-master and two-slave case



Copyright © 2013-2017-2018 by Ando Ki

Design AXI (8)

Simulation with ModelSim (1/4)

```
# Makefile
SHELL = /bin/sh
MAKEFILE = Makefile

#-----
VLIB = $(shell which vlib)
VLOG = $(shell which vlog)
VSIM = $(shell which vsim)
WORK = work
#-----
TOP = top
#-----
all: vlib compile simulate

vlib:
    if [ -d $(WORK) ]; then /bin/rm -rf $(WORK); fi
    $(VLIB) $(WORK)

compile:
    $(VLOG) -lint -work $(WORK) -f modelsim.args

simulate: compile
    $(VSIM) -novopt -c -do "run -all; quit" $(WORK).$(TOP)
```

Modelsim commands

Specify where to store compile results

Compilation

Simulation

```
@ECHO OFF
REM RunMe.bat
SET MODELSIMWORK=work
SET MODELSIMVLIB=vlib
SET MODELSIMVSIM=vsim
SET MODELSIMVCOM=vcom
SET MODELSIMVLOG=vlog

SET DESIGNTOP=top

IF EXIST %MODELSIMWORK% RMDIR /S/Q %MODELSIMWORK%

%MODELSIMVLIB% %MODELSIMWORK%
%MODELSIMVLOG% -work %MODELSIMWORK% -lint^
-f modelsim.args
%MODELSIMVSIM% -novopt -c -do "run -all; quit"^
%MODELSIMWORK%.%DESIGNTOP%
```

Copyright © 2013-2017-2018 by Ando Ki

Design AXI (9)

Simulation with ModelSim (2/4)

```
//-----
+incdir+../design/verilog
./sim_define.v
../design/verilog/axi_switch_m2s2.v
//-----
+incdir+../bfm_axi_task/example/design/verilog
../bfm_axi_task/example/design/verilog/bfm_axi.v
../bfm_axi_task/example/design/verilog/mem_axi.v
//-----
// Below are test-bench
//-----
+incdir+../bench/verilog
../bench/verilog/top.v
//-----

`ifndef _SIM_DEFINE_V_
`define _SIM_DEFINE_V_
`define SIM // define this for simulation case if you are not sure
`define VCD // define this for VCD waveform dump
`define DEBUG
`define RIGOR
`define LOW_POWER
//-----
`endif
```

modelsim.args

sim_define.v

top

Copyright © 2013-2017-2018 by Ando Ki

Design AXI (10)

```
xterm adckiphpy
INFO: Start time: Sun Feb 03 22:44:29 2013

* * *
top.SLV_BK[0][u_wen_axi INFO 4K ( 4096) byte memory
top.SLV_BK[1][u_wen_axi INFO 4K ( 4096) byte memory
105 top.MST_BK[0][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
125 top.MST_BK[0][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
160 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000000 D#0x0
160 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000001 D#0x0
160 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000002 D#0x0
160 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000003 D#0x0
160 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000003 D#0x0
195 top.MST_BK[0][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
200 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000000 D#0x0
200 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000001 D#0x0
200 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000002 D#0x0
200 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000003 D#0x0
225 top.MST_BK[1][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
250 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000000 D#0x0
250 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000005 D#0x0
250 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000006 D#0x0
250 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000007 D#0x0
285 top.MST_BK[0][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
290 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000004 D#0x0
290 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x00000005 D#0x0
290 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x00000006 D#0x0
290 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x00000006 D#0x0
290 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x00000007 D#0x0
325 top.MST_BK[1][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
340 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000008 D#0x0
340 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x00000009 D#0x0
340 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000A D#0x0
340 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000B D#0x0
375 top.MST_BK[0][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
380 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x00000008 D#0x0
380 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x00000009 D#0x0
380 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x0000000A D#0x0
380 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x0000000B D#0x0
415 top.MST_BK[1][u_bfa_axi.write_resp_channel OK response for write=resp-channel: OKAY
420 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000C D#0x0
430 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000D D#0x0
430 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000E D#0x0
430 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000F D#0x0
430 top.MST_BK[0][u_bfa_axi.test_raw OK R#0x0000000F D#0x0
470 top.MST_BK[1][u_bfa_axi.test_raw From 0x0000000F to 0x0000000F 4-size 1-leng OK
470 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x0000000C D#0x0
470 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x0000000D D#0x0
470 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x0000000E D#0x0
470 top.MST_BK[1][u_bfa_axi.test_raw OK R#0x0000000F D#0x0
470 top.MST_BK[1][u_bfa_axi.test_raw From 0x0000000F to 0x0000000F 4-size 1-leng OK

** Note: Data structure takes 239932 bytes of memory
Process Time 0.02 secs
Finish ././../bench/werlog/top.v(673)
Time: 565 ns Iteration: 2 Instance: /top
INFO: End time: Sun Feb 03 22:44:29 2013
INFO: Total time: 0.000000 secs
INFO: CPU time: 0.015000 secs in simulation
WARNING: Can't find parameter "PROVE_ELF_FILE" in top module, iPROVE emulation was disabled.
[ackdchpy]
```

Design AXI (11)

❖ This example shows how to use BFM with tasks

- ❖ [Note that this design uses codes in “\\$\(PROJECT\)/codes/bfm_axi_task/design/verilog”](#)
 - ❖ bfm_axi.v and mem_axi.v
- ❖ Step 1: go to your project directory
 - ❖ [user@host] cd \$(PROJECT)/codes/amba_axi
- ❖ Step 2: see the codes
 - ❖ [user@host] cd \$(PROJECT)/codes/amba_axi/desing/verilog
 - ❖ [You should fill necessary code in “\\$\(PROJECT\)/codes/amba_axi/bench/verilog/top.v”](#)
- ❖ Step 3: compile and run
 - ❖ [user@host] cd \$(PROJECT)/codes/amba_axi/sim/modelsim
 - ❖ [user@host] make
- ❖ Step 4: waveform view
 - ❖ [user@host] gtkwave wave.vcd &

Design AXI (12)

Example: AMBA AXI case

```
xterm -dmi@mylife
Main Options VT Options VT Fonts
[adki@mylife] make
if [ -d work ]; then /bin/rm -rf work; fi
(/cygdrive/c/ModelSim/questasim_10.3/win32/vlib work || exit -1) 2>&1 | tee compil
e.log
(/cygdrive/c/ModelSim/questasim_10.3/win32/vlog -lint -work work\
-f modelsim.args || exit -1) 2>&1 | tee compile.log
QuestaSim vlog 10.3 Compiler 2014.01 Jan 6 2014
Start time: 22:17:20 on Aug 04,2015
vlog -lint -work work -f modelsim.args

#          430 top.MST_BLK[0].u_bfm_axi.test_raw test_raw from 0x00000000
to 0x0000000f 004-size 001-leng OK
#          470 top.MST_BLK[1].u_bfm_axi.test_raw OK A:0x0000000c D:0x0d
#          470 top.MST_BLK[1].u_bfm_axi.test_raw OK A:0x0000000d D:0x0e
#          470 top.MST_BLK[1].u_bfm_axi.test_raw OK A:0x0000000e D:0x0f
#          470 top.MST_BLK[1].u_bfm_axi.test_raw OK A:0x0000000f D:0x10
#          470 top.MST_BLK[1].u_bfm_axi.test_raw test_raw from 0x00000000
to 0x0000000f 004-size 001-leng OK
# ** Note: Data structure takes 4194352 bytes of memory
#          Process time 0.00 seconds
#          $finish      : ../../bench/verilog/top.v(673)
#          Time: 565 ns Iteration: 2 Instance: /top
# INFO: End time: Tue Aug 04 22:17:24 2015
# INFO: Total time: 0.000000 secs
# INFO: CPU time: 0.016000 secs in simulation
# WARNING: Can't find parameter 'iPROVE_EIF_FILE' in top module. iPROVE emulation
was disabled.
[adki@mylife]
```