ASIC Design Laboratory
Lab 9 : Pipelined SoC Bus Protocols (AHB-Lite FIR Filter Accelerator)
Lab Manual

Fall 2019

The purpose of this lab exercises is to help you become familiar with pipelined System-on-Chip bus protocols by working the AHB-Lite protocol, which is both one of the most widely used pipelined SoC bus protocols and contains a lot of the core aspects used in more advanced ones. Pipelined SoC bus protocols are primarily used for integrating high-speed peripheral hardware modules and computational accelerators with a core processor. Therefore, in this lab you will be connecting the your completed FIR Filter design from the prior lab with an AHB-Lite slave interface module to create a FIR Filter accelerator module for an SoC.

It is important to note, that given the fundamentally parallel and interactive nature of hardware designs, debugging designs described with HDL code requires a method that strictly identifies and leverages guaranteed cause-effect relationships with in the design's description. Other lazy or speculative debugging methods will generally result in vast amounts of wasted time, effort, and frustration and can easily increase debugging times by a factor of 10x.

In this lab, you will perform the following tasks:

- Design and test via a test bench an AHB-Lite-Slave interface module
- Integrate your completed AHB-Lite-Slave interface module with your FIR Filter module to create an AHB-Lite-connected FIR Filter accelerator module (AHB-Lite-FIR).
- Develop test benches for verifying the AHB-Lite-Slave functionality of your design using a provided AHB-Lite bus-model.
- Synthesize the AHB-Lite-FIR design using Design Compiler®
- Test the Synthesized/Mapped version of the AHB-Lite-FIR design
- Submit electronically your completed AHB-Lite-FIR design to be graded

1 Lab Setup

In a UNIX terminal window, issue the following commands, to setup your Lab 9 workspace:

mkdir –p ~/ece337/Lab9
cd ~/ece337/Lab9
dirset
setup9

The setup9 command is an alias to a script file that will check your Lab 9 directory structure and give you file needed for starting the lab. If you have trouble with this step please ask for assistance from your TA.

IMPORTANT: Make sure to add this new workspace into your 337 Repository, like you did in Lab1.

This way, you will always have the original copy in storage.

2 Lab Work Overview

2.1 Required AHB-Lite Preparation

To prepare for implementing your AHB-Lite-FIR design you must complete the following:

- Create a complete Hierarchical RTL diagram for the AHB-Lite-Slave interface module described in Section 5.2
- Create complete RTL and State Transition Diagrams for the AHB-Lite-Slave interface controller.

You must have these diagrams submitted as either PDF file(s) or image files using common standards (JPEG, PNG, or BMP) via "submit Lab9prep" in order to earn points for them.

It is highly recommended that you complete all of these diagrams prior to starting to write any design code, as this should save you tremendous amounts of time debugging/rewriting code later on.

NOTE: As in prior labs, All diagrams, must be done as a digital drawing. Hand drawn diagrams (even if they are scanned) will receive a grade of zero points.

2.2 Expectations Regarding Lab 9 and the Remaining Labs

In this lab, you have been given access to a library that contains complied and verified modules for implementing the model you will be using to verify your slave-interface's functionality, which is described in Section 6. As this module is contained in a library they must not be included in the various makefile variables, otherwise the makefile will error out trying to find local copies of the files. You will have to use the simulation rules from the makefile in order for library it is contained in to be linked against when starting the simulation.

You will need to design and test the blocks described in Section 5, which includes the top-level block. You are not and will not being specifically instructed on how to design these blocks, only their intended behavior/function. You are only told the expected architecture for the design, which is comprised of the inputs to each block, the outputs from each block and what function the block is to perform. It is up to you to come up with a working solution for your blocks and then integrate the building blocks to form the full design.

2.3 Required Minimum Verification

2.3.1 Required Minimum AHB-Lite-Slave Interface Verification

In order to facilitate correct usage of the provided bus model, you are provided with a starter test bench for testing your AHB-Lite-Slave interface module. This starter test bench will need to be extended to have additional test cases that at a minimum satisfy the following requirements:

- Test for correct operation during Master reads of the AHB-Lite-Slave Interface's result register
- Test for correct operation during Master reads of the AHB-Lite-Slave Interface's new sample register
- Test for correct operation during Master reads of the AHB-Lite-Slave Interface's status register
- Test for correct operation during Master reads of the AHB-Lite-Slave Interface's coefficient related registers
- Test for correct operation during Master writes to the AHB-Lite-Slave Interface's result register
- Test for correct operation during Master writes to the AHB-Lite-Slave Interface's new sample register
- Test for correct operation during Master writes to the AHB-Lite-Slave Interface's status register

- Test for correct operation during Master writes to the AHB-Lite-Slave Interface's coefficient related registers

2.3.2 Required Minimum AHB-Lite-FIR Peripheral Verification

Additionally, you will need to create a test bench that verifies the operation of the full AHB-Lite-FIR peripheral design. It is recommend that you do this by extending a copy of the AHB-Lite-Slave Interface starter test bench to have test cases design around end-to-end operation. In general, the structure and test cases within this test bench should be a blending of your stand-alone AHB-Lite-Slave Interface and FIR Filter test benches. The minimum requirements for this test bench are as follows:

- Must use tasks for consistent execution of sample streaming, power-on-reset, and result checking
- Must use test vector approach to manage testing the design on a set of coefficients and sample values
- Must correctly use the provided AHB-Lite bus model during AHB-Lite bus related operation(s)
- Must correctly test for valid sample handling according the specifications in Lab8
- Must have a task for configuring the design via the AHB-Lite interface

2.4 Grading Policy

More than seventy percent (>85%) of your grade for this lab will be determined from the mapped version of your full design implementation. The automated grading system will run the grading test bench on both the source and the mapped version of your design, but only the mapped version results will be used for grading. The code for this test bench will not be provided to you nor will you be told the details of how any test case operates. In order to run the automatic grading script, your design must have an error-free run through Design Compiler®. Your final grade will be determined by the most recent total grade you have obtained in a mapped test run and not a combination of different test runs. You will be allowed a maximum of 3 passes through the Lab 9 grading script.

The grading script is not there to for you to use to test your design, it is there to grade your design. Much like you are expected to check your own work prior to turning in homework in other classes, you are expected to do your own testing of your design prior to submission for grading. In regards to the design, you should ensure that you are naming the blocks the names that are specified in this lab. In addition, the interface signals for the top-level receiver block must be identical to those listed in Section 4 of this lab. Failure to name the interface signals correctly will result in the automated grading test bench failing and that corresponding run will count as 1 of your 3 possible runs. You will need to score 50% (30/60) or higher on your most recent mapped version test in order to satisfy the outcome for this lab.

2.5 Submission Commands

submit Lab9prep Submits the contents of your "docs" folder for the preparation phase and will be due prior to the main design submission

submit Lab9 Submits your design for automated grading

submit Lab9re Submits your design for automated grading for early remediation purposes and will only be activated after the regular deadline has passed for all lab sections

submit Lab9r Submits your design for automated grading for regular remediation purposes and will only be activated after the early remediation deadline has passed for all lab sections

3 Advanced High-performance Bus Lite Protocol

The Advanced High-performance Bus Lite (AHB-Lite) protocol[1] is one of many that form the Advanced Microprocessor Bus Architecture (AMBA) Bus System design by ARM.

Similar to APB transactions, AHB-Lite transactions are broken into two stages (nominally one cycle each) with the first stage being the setup or 'address' stage and the second one being the response or 'data' stage. However, unlike the APB bus, the AHB-Lite bus is pipelined to allow the address stage of a transaction to temporally overlap with the data stage of the prior transfer (regardless of which slave(s) are involved in the transfers) in order to improve throughput of the bus. This improved performance enables AHB-Lite to be used as a main SoC bus protocol for connecting the various high performance devices (such as memory and accelerators) to the core process, as well as whole peripheral subsystems via AHB-Lite to APB bridges.

Like most SoC buses, AHB-Lite has three main aspects or parts:

1. The 'master' interface device which is in charge of initiating any/all requests on the bus
2. At least one 'slave' interface device which responds to requests that are directed to it through the bus
3. The bus 'fabric' which handles how all of these devices are physically connected and how control and data signals for requests are routed between the two devices involved in an bus transaction.

3.1 AHB-Lite Signals

The following are the various signals (by name) that are used within the AHB-Lite protocol and their respective roles:

HCLK This is the bus (and commonly system) clock signal.

HRESETn This is the bus (and usually system) active-low reset signal.

HADDR This is used for providing the address (within the slave only) that the transaction involves.

HPROT This signal is used for selecting between protection levels for the transfers.

HMASTERLOCK This signal indicates that the transfer is 'locked' by the master and thus must be treated as an atomic operation.

HSELx This is 'slave' selection signal where the 'x' is replaced by a number for the slave it is connected to and each 'slave' device is given it's own dedicated one.

HTRANS This indicates the type of the current transfer.

HSIZE This indicates the size of the transfer. Typically is Byte, half-word, or full-word size and scales with the data bus size, with the number of bytes equal to 2^{HSIZE}.

HWRITE This indicates whether a transaction is a read (logic low value) or write (logic high value).

HWDATA This is used for transferring the data written to the 'slave' device.

HRDATA This is used for transferring the data read from the 'slave' device and can be up to 32-bits wide.

HREADY This is an active-high ready feedback signal from the 'slave' device which is pulled low by the 'slave' if it needs to stall/pause the transaction.

HRESP This is an active-high error feedback signal from the 'slave' device, and must be asserted when there is a transaction error (such as trying to write to read-only addresses).

In this lab you are going to be focusing on implementing a 'slave' device for the AHB-Lite protocol and so the following sections are going to focus on bus transfers as seen by the 'slave' devices after the bus 'fabric' has already handled the routing of signals and data to or from the 'master' and 'slave' devices. Also, since this lab will not be working with burst transfers, their timing and operation is not discussed below.

3.2 Example Operation of a Basic Write transfer

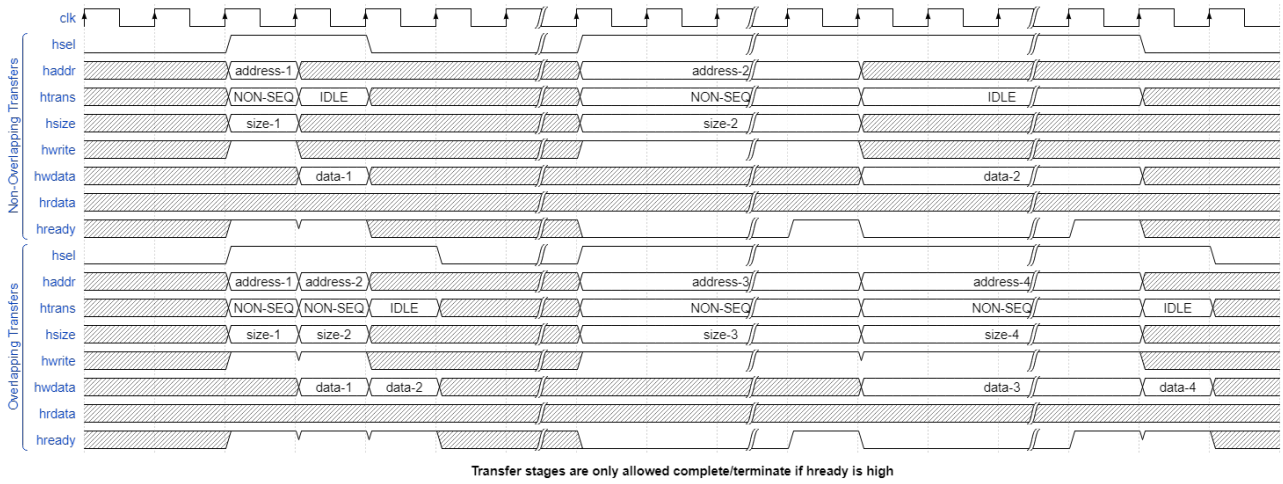

Figure 1: Example operation and timing of AHB-Lite Writes from the perspective of a 'slave'

Write transfers are only allowed to finish and release the bus once the 'slave' maintains the 'hready' signal (via it's 'hreadyout' at a logic-high value during the second stage, to indicate that the write was either completed or at least internally buffered depending on the design. If the 'hready' signal is at a logic low value then the bus (and involved master) must stall for as long as 'hready' stays at a logic low value. This requirement is true for both the address phase and the data phase of transfers and can result in indefinite stall or 'freezing' of the bus if 'hready' is misused. Additionally, subsequent transfers can be overlapped or spaced out by the master.

3.3 Example Operation of a Basic Read transfer

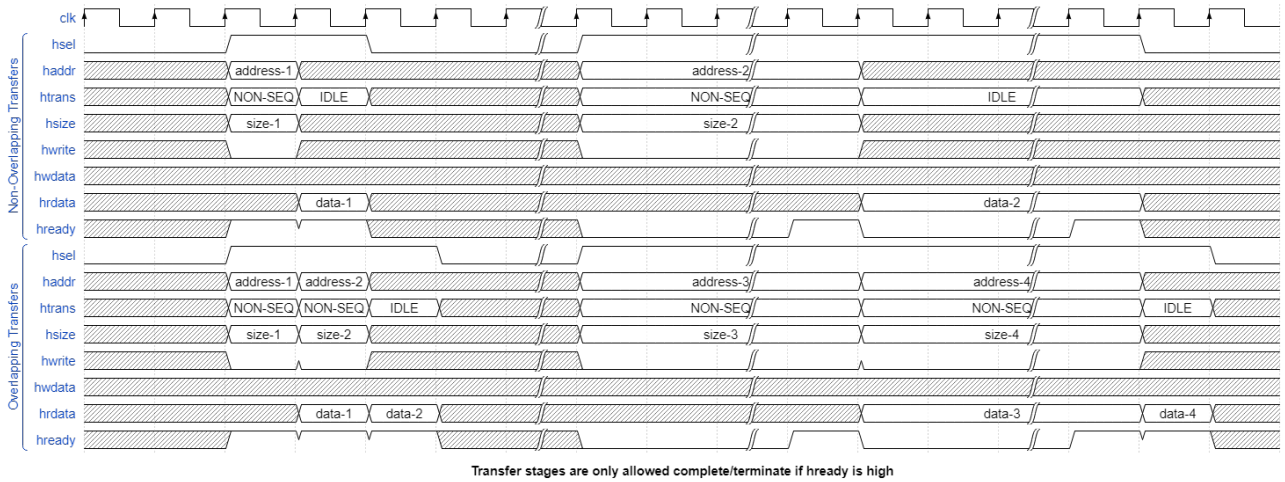

Figure 2: Example operation and timing of AHB-Lite Reads from the perspective of a 'slave'

Similar to write transfers, read transfers are only allowed to finish and release the bus once the 'slave' maintains the 'hready' signal (via it's 'hreadyout' at a logic-high value during the second stage, to indicate that the data requested has been available on the 'hrdata' bus. If the 'hready' signal is at a logic low value then the bus (and involved master) must stall for as long as 'hready' stays at a logic low value. This requirement is true for both the address phase and the data phase of transfers and can result in indefinite stall or 'freezing' of the bus if 'hready' is misused. Additionally, subsequent transfers can be overlapped or spaced out by the master.

3.4 Transfer Error during Basic Read or Write

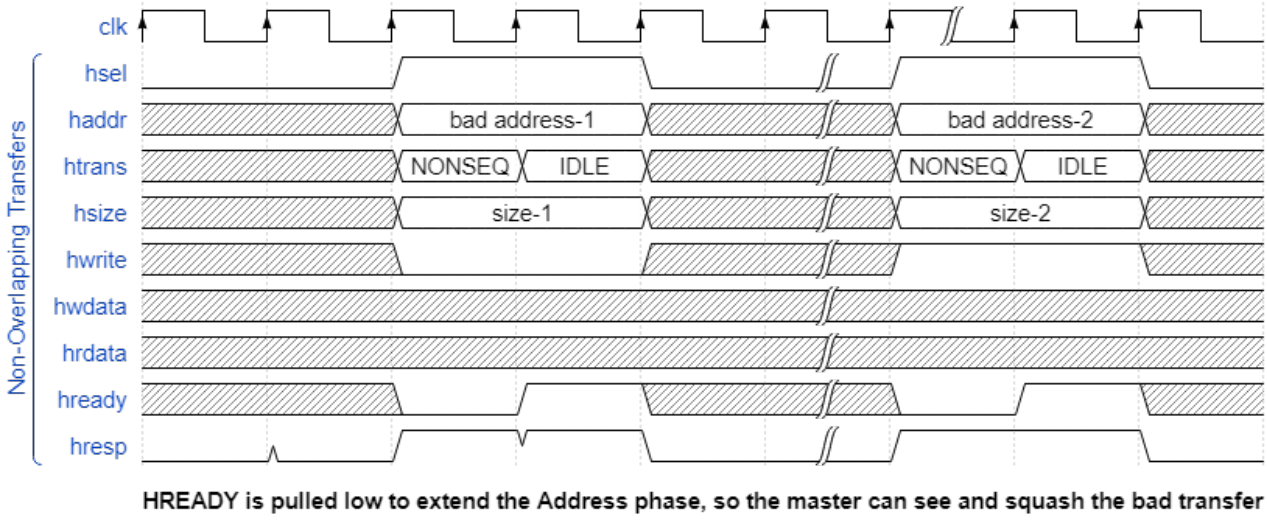

Figure 3: Example of erroneous read and write transfers.

In order for the master to be able to see the 'hresp' value (during an error) in time, the slave should pull 'hready' low to extend the address phase to be at least two cycles long. Once the master sees the error response it will generally choose to nullify the transfer by overriding it with an IDLE transfer value before the bus pipeline advances.

4 Design Architecture

A full SoC accelerator module contains both the SoC bus related interface module and the accelerator's functional hardware. To simplify verification and design, and re-usability with different SoC bus standards, all of the accelerator's functional hardware is bundled in it's own module (in this case your FIR Filter module). The means that the top-level file simply serves to bundle the chosen SoC bus interface module with the functional module and any 'glue-logic' needed (such as the FIR coefficient loader module in the following design).

4.1 AHB-Lite FIR Filter Accelerator Design Architecture

The architecture for the AHB-Lite FIR Filter Accelerator design you will be completing during this lab is shown in Figure 4.

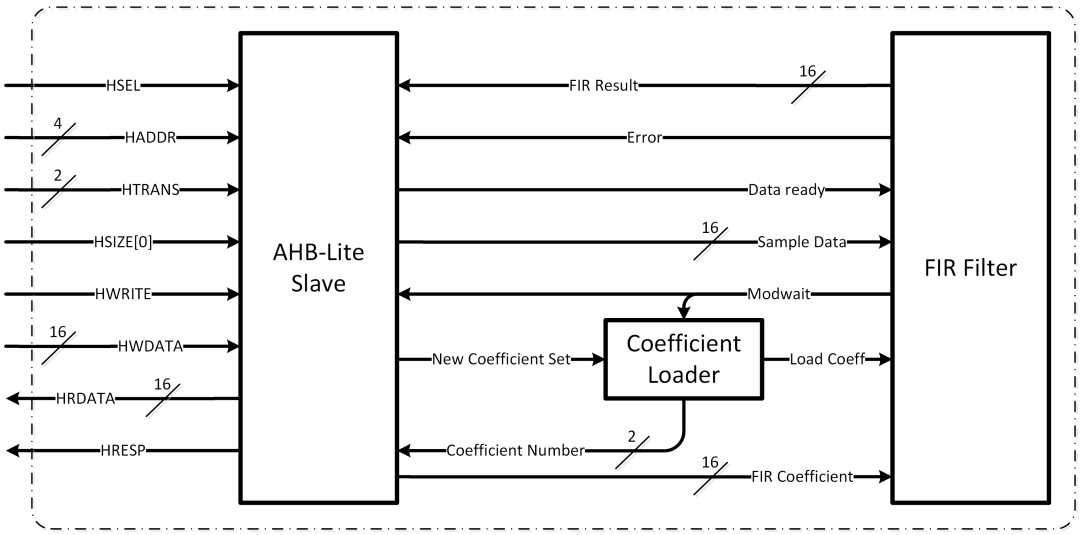

Figure 4: AHB-Lite FIR Filter Accelerator Architecture Diagram

NOTE: Clock and Reset signals are assumed to be sent to the appropriate blocks and are not shown

You may notice that some of the AHB-Lite protocol signals are missing from the 'slave' interface on this architecture. Within AHB-Lite there are several signals that are optional for use depending on what the specific 'slave' interface is being designed to support. Since this is your first attempt an AHB-Lite design and 'burst-mode' is performance optimization for when working with multi-masters on the same AHB-Lite bus, this design will not support 'burst-mode' in order to keep things focused on core functionality before optimization. This design does not have a functional need or benefit from varying protection levels during AHB-Lite transactions, and thus it will not use the the optional 'HPROT' signal. This design is not intended for use with multiple master modules, as it wouldn't make sense for FIR calculations, and thus does not need the 'HMASTERLOCK' signal. Furthermore, the 'HREADYOUT' signal will not be used for two reasons: (1) the FIR filter calculations complete at a significantly slower rate than transactions happen on the AHB-Lite bus and the only situation that might make sense for a FIR filter related pause is while waiting for calculations to complete and (2) requiring the end user/system to poll (periodically check) the status register will allow other devices in the SoC to use the bus while the system waits for a calculation to complete. Additionally, the 'one_k_samples' signal from the FIR is not connected because the software using this accelerator would naturally already know this status.

4.1.1 List of the Functional Units/Blocks and Purpose

AHB-Lite-Slave This handles all AHB-Lite-Slave Interface specific functionality.

Coefficient Loader This controls the FIR coefficient loading sequence.

FIR Filter This handles all FIR Filter specific functionality and is your prior FIR Filter design.

4.1.2 Suggested FIR Filter Design changes

Since the FIR Filter module is now used in a way where both 'sides' of the design operate on the same system clock, you should remove the synchronizers on the 'data_ready' and 'load_coeff' input signals to improve the responsiveness and ease of linking the FIR Filter module with the new AHB-Lite Slave interface module.

4.1.3 SoC Designer Requirements

In general, SoC standards provide the framework but require the actual SoC designers/architects (the course staff in this case) to make a variety of decisions about how values are handled and other design trade-offs that are more specific to a given SoC's operation.

Earlier we discussed the SoC design choices of not using some of the non-required bus signals, including 'HREADYOUT'. As a result of not explicitly controlling the 'HREADYOUT' bus signal in this SoC module, bus errors would also be handled a little bit differently, since the design won't be able to pull 'HREADY' low to extend the address phase of the transfer in order to make sure the master sees the error response. For this design the error response must still be asserted during the address phase of the erroneous transfer, but it assumed that the host will be fast enough to see the response before the natural end of the address phase, rather than needing extra time. Therefore, the expected behavior (which is check by the provided bus model) is that 'HRESP' is set to high for just the 1-cycle long address phase that naturally happens for the erroneous transfer, followed by it's corresponding data phase (on the next cycle) being discarded/ignored as long as the 'HRESP' was set during the address phase.

Another choice is what to do with portions of the data bus that are not active during transfers that don't use the full bus width. For this design, all non-active portions of the data bus during 8-bit transfers must be zero-filled and all unused bits of a register's value must be zero-filled before being put on the read data bus.

4.1.4 List of the Top-Level Ports and Purpose

Clk This is the system clock port. It should be connected to a 100 MHz clock.

N_Rst This is the active-low asynchronous system reset signal

HADDR This is used for providing the address (within the slave only) that the transaction involves.

HSEL This is the 'slave' selection signal.

HTRANS This indicates the type of the current transfer.

HSIZE This indicates the size of the transfer.

HWRITE This indicates whether a transaction is a read (logic low value) or write (logic high value).

HWDATA This is used for transferring the data written to the 'slave' device.

HRDATA This is used for transferring the data read from the 'slave' device and can be up to 32-bits wide.

HRESP This is an active-high error feedback signal from the 'slave' device, and must be asserted when there is a transaction error (such as trying to write to read-only addresses).

4.2 Required AHB-Lite-Slave Address Mapping

Table 1: Table of the required AHB-Lite-Slave address-to-value mapping

Address	Value Size (Bytes)	Access Mode	Description
0x0	2	R	Status Register: Value of '0' → FIR Filter is idle, Value of '1' for bit-0 → FIR Filter is busy, Value of '1' for bit-8 → FIR Filter encountered an error
0x2	2	R	Result Register
0x4	2	R/W	New Sample Register
0x6	2	R/W	F0 Coefficient Register
0x8	2	R/W	F1 Coefficient Register
0xA	2	R/W	F2 Coefficient Register
0xC	2	R/W	F3 Coefficient Register
0xE	1	R/W	New Coefficient Set Confirmation Register (Set to '1' to activate, will be cleared to '0' after coefficient loading has completed)

NOTE: The address-to-value mapping in Table 1 was intentionally chosen so that it would naturally work well for both 8-bit and 16-bit data AHB-Lite bus configurations, since SoC bus fabrics often handle connections of different sizes.

4.3 Valid Design Usage

The valid flow of operations for using this design is as follows:

1. Configuration of FIR Coefficients, unless they have already been set to the same values from prior operation.
2. Send the sample data to process.
3. Poll the Status Register until new data is present.
4. Read the data from the Result Buffer.
5. Repeat steps 2-4 until done with sample data to process with the current FIR coefficients.

Additionally the host SoC may desire to perform error checking as well which would sensibly be in the form of checking the Error Status bit of the status register during each polling attempt.

The FIR filter must be considered as busy during either a sample processing round or during the full duration of loading a set of coefficients, and the status register must be updated accordingly.

5 Specifications for the Blocks You Must Design and Implement

5.1 AHB-Lite FIR Filter Accelerator

5.1.1 Block Description

This is the full design module that connects the dedicated AHB-Lite-Slave interface and FIR Filter modules together to form the overall SoC accelerator module.

5.1.2 Module Specifications

Required Module Name: ahb_lite_fir_filter

Required Filename: ahb_lite_fir_filter.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_rst	input	This is an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.
hsel	input	The selection signal for the slave.
haddr[3:0]	input	The address bus for the slave.
hsize	input	This is indicates the size of the transfer
htrans[1:0]	input	This indicates the type of the current transfer.
hwrite	input	The write/read mode signal for the slave.
hwdata[15:0]	input	The write data bus for the slave.
hrdata[15:0]	output	The read data bus for the slave.
hresp	output	The transaction error feedback from the slave.

5.2 AHB-Lite-Slave Interface

5.2.1 Block Description

5.2.2 Module Specifications

Required Module Name: ahb_lite_slave

Required Filename: ahb_lite_slave.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_rst	input	This is an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.
sample_data[15:0]	output	The 16-bit unsigned data input
data_ready	output	Active high signal that indicates when a sample is ready to be processed
new_coefficient_set	output	Active high signal indicating to load a set of coefficients.
coefficient_num[1:0]	input	Which coefficient (0 -> F0, 1 -> F1, etc.) should be supplied to the FIR Filter
fir_coefficient[15:0]	output	16-bit fixed-point coefficient [0.0, 1.0] to load
modwait	input	Active high signal indicating that the design is busy
fir_out[15:0]	input	The 16-bit unsigned FIR filter of the last four samples
err	input	Active high signal indicating that an error occurred during the processing of the most recent sample
hsel	input	The selection signal for the slave.
haddr[3:0]	input	The address bus for the slave.
hsize	input	This is indicates the size of the transfer
htrans[1:0]	input	This indicates the type of the current transfer. Value of '0' $\rightarrow$ bus is idle, Value of '1' $\rightarrow$ delay within variable length burst (only valid for burst transfers), Value of '2' $\rightarrow$ non-sequential burst or individual transfer, Value of '3' $\rightarrow$ sequential burst (only valid for burst transfers)
hwrite	input	The write/read mode signal for the slave.
hwdata[15:0]	input	The write data bus for the slave.
hrdata[15:0]	output	The read data bus for the slave.
hresp	output	The transaction error feedback from the slave.

5.3 Coefficient Loader Module

5.3.1 Block Description

This manages loading a full set of coefficients from the AHB-Lite interface module into the FIR Filter module. It should be a simple FSM module that mimics the behavior previously done by the coefficient loading tasks in starter test bench provided during the FIR Filter lab.

5.3.2 Module Specifications

Required Module Name: coefficient_loader

Required Filename: coefficient_loader.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_reset	input	This is an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.
new_coefficient_set	input	Active high signal indicating to load a set of coefficients, as soon as possible
modwait	input	Active high signal indicating that the design is busy
load_coeff	output	Active high signal that indicates when a FIR coefficient must be loaded
coefficient_num[1:0]	output	Which coefficient (0 -> F0, 1 -> F1, etc.) should be supplied to the FIR Filter

5.4 FIR Filter Module

5.4.1 Block Description

This is your completed FIR Filter design from Lab 8. The only changes from your prior design should be removing the no longer needed synchronizers for 'data_ready' and 'load_coeff'.

5.4.2 Module Specifications

Required Module Name: fir_filter

Required Filename: fir_filter.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_reset	input	This is an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.
sample_data[15:0]	input	The 16-bit unsigned data input
data_ready	input	Active high signal that indicates when a sample is ready to be processed
fir_coefficient[15:0]	input	16-bit fixed-point coefficient [0.0, 1.0] to load
load_coeff	input	Active high signal that indicates when FIR coefficients must be loaded
one_k_samples	output	Active high signal indicating that 1,000 samples have been processed since the last assertion/coefficient load
modwait	output	Active high signal indicating that the design is busy
fir_out[15:0]	output	The 16-bit unsigned FIR filter of the last four samples
err	output	Active high signal indicating that an error occurred during the processing of the most recent sample

6 Specifications for Provided Blocks

When working with established bus standards and protocols, it is common to use a predesigned 'bus model' to aid in verification that the design complies with the formal standard, as this model is usually designed by someone whom is very experienced with any nuances of the standard. When using a bus model, the design's bus signals are connected to the model and not directly used by the test bench and the test bench then controls the bus model during testing.

The rest of this section discusses the AHB-Lite bus model provided to you for this lab via the Labs_IP library which your makefile will link against for simulations.

The bus model has two parameters:

DATA_WIDTH The size of the data buses, in terms of bytes. (Default value of 2)

ADDR_WIDTH The size of the slave's address bus, in terms of bits. (Default value of 4)

Module Name: ahb_lite_bus

Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
model_reset	input	This is an active-high reset for the bus model.
enqueue_transaction	input	This must be pulsed (0→1) for each new transaction to enqueue. There is no hard limit to the number of transactions that can be enqueued.
transaction_write	input	Logic '1' if the transaction being enqueued is a write and a '0' for a read.
transaction_fake	input	This must be a logic '1' if the transaction should act like it was intended for another device than the attached 'slave', and a logic '0' otherwise.
transaction_addr[#:0]	input	This is the intra-slave address to use during the transaction. (Sized based on 'ADDR_WIDTH')
transaction_size[2:0]	input	This is the value of 'HSIZE' to use for this transaction.
transaction_data[#:0]	input	This is the data value for the transaction. It will be written if the transaction being enqueued is a write, and expected as the response if it is a read. (Sized based on 'DATA_WIDTH')
transaction_error	input	This must be a logic '1' if the transaction is expected to result in a slave-side error, and a logic '0' otherwise.
enable_transactions	input	This is the active-high enable for the bus model to start running through enqueued transactions. As long as it is enabled, the bus model with run through it's internal queue in a back-to-back fashion.
current_transaction_num	output	This is an integer value that shows the ordinal number of the transaction currently be executed by the model. A value of '1' means first one since model reset.
current_transaction_error	output	Value of '1' → there was incorrect behavior during the current transfer.
hsel	output	The selection signal for the slave.
haddr[#:0]	output	The address bus for the slave. (Sized based on 'ADDR_WIDTH')
hsize[2:0]	output	This is indicates the size of the transfer
htrans[1:0]	output	This indicates the type of the current transfer
hwrite	output	The write/read mode signal for the slave.
hwdata[#:0]	output	The write data bus for the slave. (Sized based on 'DATA_WIDTH')
hrdata[#:0]	input	The read data bus for the slave. (Sized based on 'DATA_WIDTH')
hresp	input	The transaction error feedback from the slave.

7 Closing Remarks

- Turn in your check-off sheet at the beginning of your lab section during week 10.
- You will not be provided with nor will you be able to see the code for the provided bus model.
- Since the bus model will be checking for correct transaction-level behavior, there will be internal assertions that may result in QuestaSim® complaining about not being able to open the source file for the model during your simulation. This is an artifact of how QuestaSim® tries to 'helpfully' handle assertion messages and not an true error.

Bibliography

[1] ARM. *AMBA 3 AHB-Lite Protocol Specification v1.0*. 2006.