ASIC Design Laboratory
Cooperative Design Lab (CDL)
2D Convolver End-point AHB-Lite SoC Module
Design Manual (2 and 3-Person Teams)

Spring 2026

The purpose of this multi-week cooperative lab is to help you gain experience working with a team of designers to implement and validate a larger scale design. The previous labs have all be focused on smaller self-contained designs to allow you to develop your system and hardware design skills, along with developing your ability to use an HDL to describe hardware systems. In this lab you will be directly responsible for designing and implementing a fairly sized portion of an overall design, with your team members implementing the other portions, and then working working you teammates to integrate your work into the overall design. Additionally, you will need to design the validation setup for the work one of your teammates implemented, and one of them will do so for the portion you designed. This setup is more similar to what you would experience in industry, where someone other than you will be responsible for validating your work according the standards and requirements that were set for your design, rather than based on knowing how it is implemented.

This design is going to be larger in design effort than your prior lab work and will have more opportunity for error propagation. Therefore it is paramount that you follow efficient debugging approaches based on clearly establishing cause-effect relationships through your design working backward from the chronologically first high-level problematic behavior. Other lazy or speculative debugging methods will most certainly result in vast amounts of wasted time, effort, and frustration and can easily increase debugging times by easily a factor of 10x.

In this lab, you (with your team) will perform the following tasks:

- Plan out the implementation of a larger design composed of multiple functional modules, that each are composed of multiple sub-modules, based on a provided starter architecture.
- Submit your implementation planning as a group via Brightspace submission
- Demonstrate and discuss your design planning with your TA.
- Implement the major modules of the design architecture as a hierarchy of smaller function-focused modules.
- Develop test benches to validate the major modules (the test bench you develop can not be for the major module you are responsible for).
- Directly validate the major modules of the design prior to integrating them to form the overall design.
- Integrate the major modules to form the overall required design.
- Synthesize the overall design (or at least its major modules) using Design Compiler®
- Validate the Synthesized/Mapped version of the overall design (or at least its major modules)
- Prove via demonstration to course staff that your design (or at least the major modules) works as required.
- Perform area and timing analysis of the major modules and overall design.
- Document any needed revisions to your design's architectural approach, your validation methodology, and the area and timing analysis of your design via a final report.
- Submit your final report as a group via Brightspace submission.

1 Lab Work Overview

1.1 Design Overview

For the Cooperative Design Lab you will be designing, implementing, and verifying a System-on-Chip peripheral module that would add 2D Convolver End-point support to an AHB-Lite based SoC. 2D Convolvers like the one you will design in this lab are useful for a variety of applications, from image processing to machine learning. Each of these peripheral devices will generally be an SoC of it's own, with it's own (usually light-weight) CPU/microcontroller, on-board volatile memory, potentially some non-volatile/permanent memory, and other peripheral modules for managing various other IO connections used within the device. Some devices might also have power-optimized compute accelerators if high-precision or sophisticated controls are needed, such as in printers or high resolution scanning equipment. An example of one such SoC system is shown in Figure 1 for a contextual reference. The AHB-Lite and 2D Convolver related specifications are discussed in dedicated sections, followed by sections discussing the required Cooperative Design Lab architecture, required core components, and required team workload allocations.

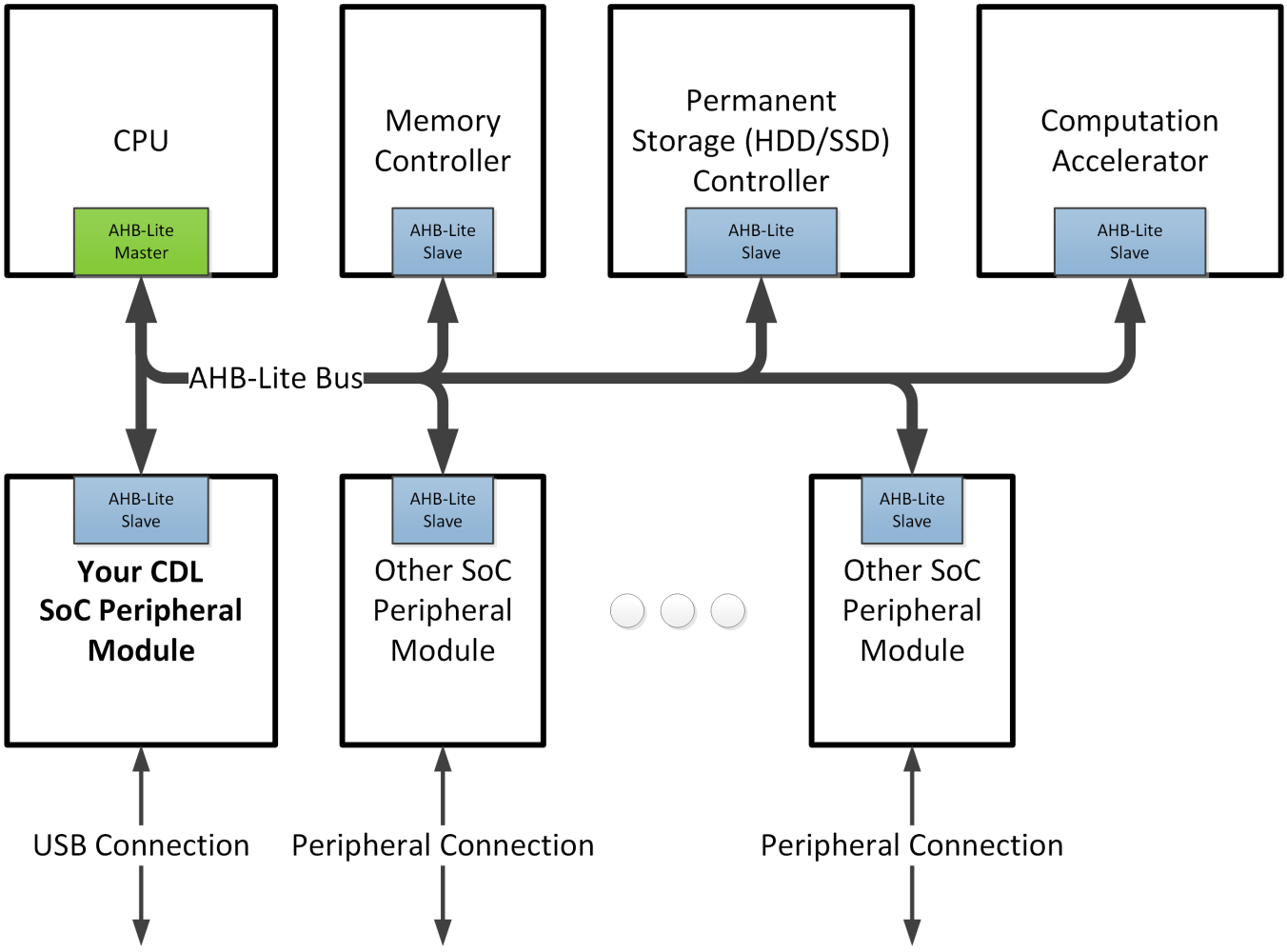

Figure 1: Example of CDL 2D Convolver End-point peripheral usage in an end-point device SoC

1.2 Expectations Regarding the Design Planning Stage

During the design planning stage you are required to:

- Create separate functional block or hierarchical RTL diagrams to describe the internal structure of each of the major modules shown in the design's overall architecture
- Create separate RTL diagrams for each block indicated in a major module's diagram.
- For each block provide a description of the function of that block in one of the following forms: pseudocode, state transition diagram, or flow-chart.
- Demonstrate and discuss your diagrams with your TA.
- Submit your prepared diagrams as a group via a Brightspace assignment.

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

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.

To guide you in your design work, you are provided with a starting architecture for the overall design and a list of required functional blocks that will be needed within each major module. Your task is to design how these pieces will interact with each other to execute the overall functionality needed. 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. You are allowed to create additional signals to link between major modules in the architecture, but must otherwise maintain the same structure for the top-level of the design.

1.3 Expectations Regarding the Implementation Stage

During the Implementation stage, you are expected to:

- Implement each module's functional sub-modules
- Verify the functionality of each module's functional sub-modules via a unit-testing style approach.
- Integrate the sub-modules to form their respective major module(s).

1.4 Expectations Regarding the Verification Stage

The verification stage is where you are expected to do full major module and full design verification and validation. To ensure correct verification of the AHB-Lite interface operation, you will be provided with and required to use an AHB-Lite bus model that has been updated to expect correct HREADY and Burst Transfer related behavior. This will be provided via a library that contains complied and verified modules for implementing the model. 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. At the start of this stage, you will be given a document that describes proper interfacing and usage of the provided bus model and the various verification and demonstration requirements that will be used to determine your CDL Technical Accomplishment Score.

1.5 Grading Policy

As stated in the syllabus, there are three aspects to the CDL grading: (1) technical accomplishment, (2) Documentation, (3) Demonstration. The technical accomplishment portion is based on successful completion of the required design process and the proof of correct functionality of your design's implementation, and is worth 60% of the CDL total score and 15% of your course grade. The documentation aspect is based on the completeness of your design

planning and final report, and is worth 20% of the CDL total score and 5% of your course grade. The Demonstration aspect is based solely on the effectiveness, correctness, and organization of your validation demonstration to course staff, and is worth 20% of the CDL total score and 5% of your course grade.

The workload distribution requirements are discussed in the design architecture section.

2 Advanced High-performance Bus Lite Protocol [teams of 3 only]

You should already be somewhat familiar with AHB-Lite from Lab 9. However, this lab will require supporting more of the full protocol and thus this section will both recover the portions of the protocol covered in the Lab 9 manual and also discuss the additional portions of the protocol that your design will be supporting during this Cooperative Design Lab.

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.

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.

2.1 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 its own dedicated one.

HTRANS This indicates the type of the current transfer (value encoding provided in Table 1).

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).

HBURST This indicates the nature of the current burst transfer (value encoding provided in Table 2).

Table 1: The value encoding scheme for the 'HTRANS' signal

Value	Label	Description
0	IDLE	The bus in an idle phase.
1	BUSY	There is currently a master triggered delay cycle within a variable length burst.
2	NONSEQ	This is either an individual transfer or a the first beat of a burst.
3	SEQ	This is one of the beats in a burst.

Table 2: The value encoding scheme for the 'HBURST' signal

Value	Label	Description
0	SINGLE	This is a single transfer.
1	INCR	This is an incrementing burst of variable length.
2	WRAP4	This is a 4-beat wrap-around burst.
3	INCR4	This is a 4-beat incrementing burst.
4	WRAP8	This is a 8-beat wrap-around burst.
5	INCR8	This is a 8-beat incrementing burst.
6	WRAP16	This is a 16-beat wrap-around burst.
7	INCR16	This is a 16-beat incrementing burst.

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.

2.2 Operation of a Basic Write transfer

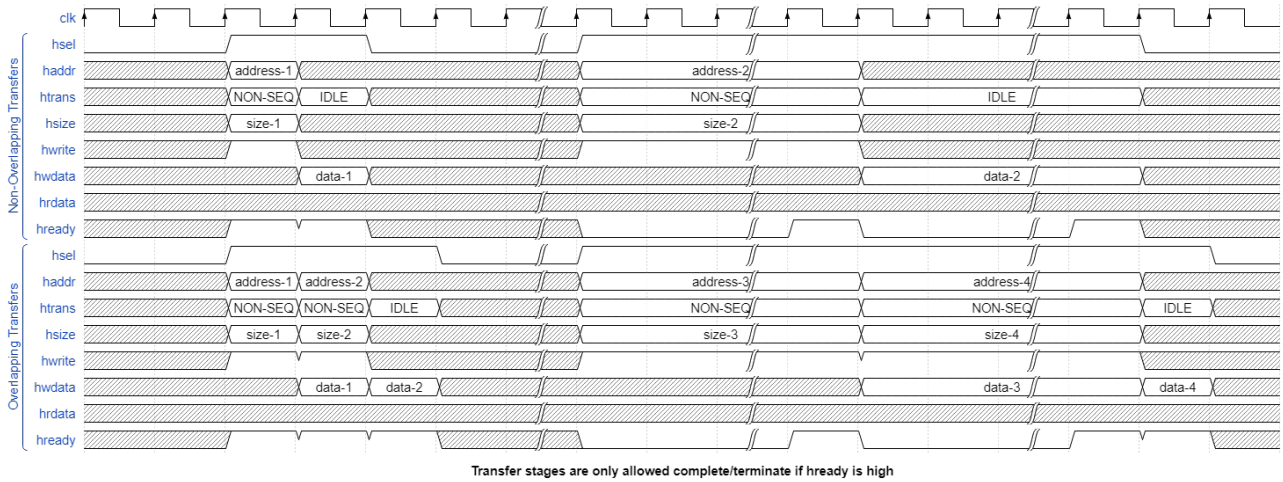

Figure 2: Example operation and timing of AHB-Lite Writes from the perspective of a 'slave'

Write transfers are only allowed to finish and release the bus once the 'slave' maintains the 'hready' signal (via its '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.

2.3 Operation of a Basic Read transfer

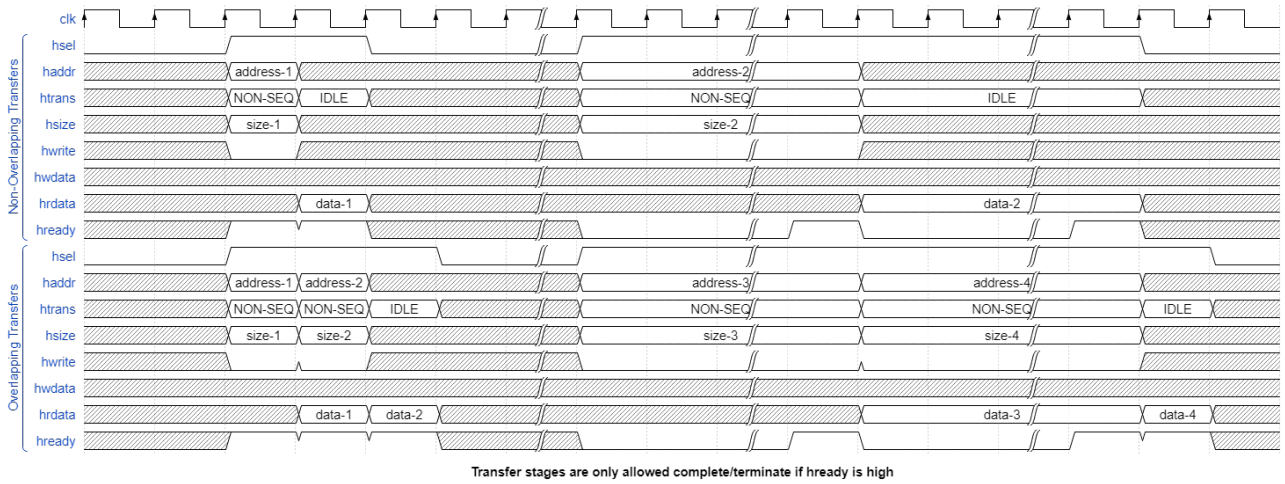

Figure 3: Example operation and timing of AHB-Lite Reads from the perspective of a 'slave'

Similar to write transfers, read transfers are only allowed to finish and release the bus once the 'slave' maintains the 'hready' signal (via its '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.

2.4 Transfer Error during Basic Read or Write

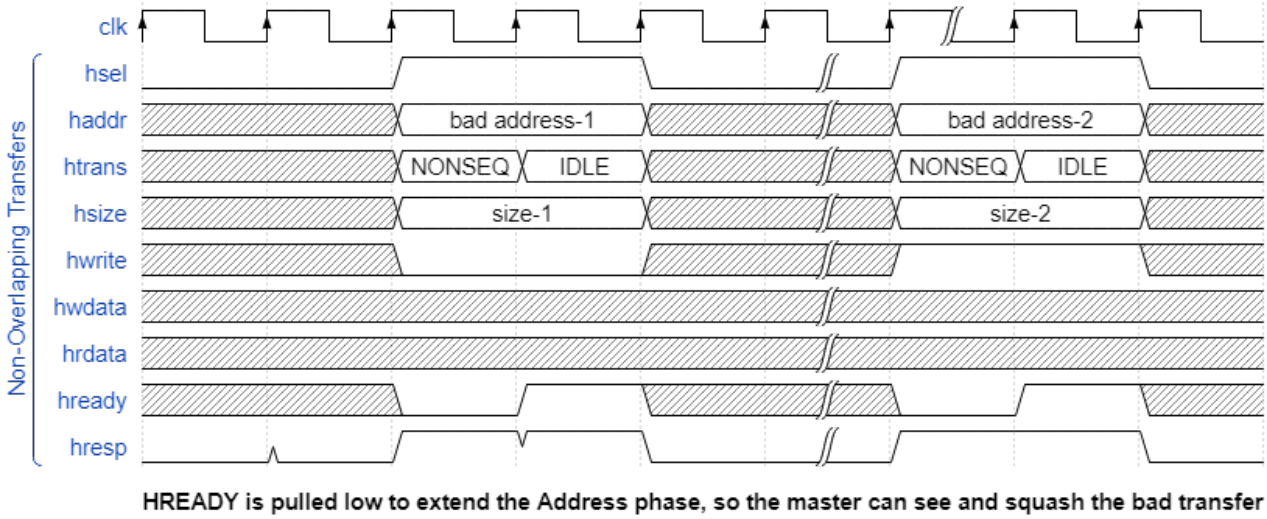

Figure 4: Example of erroneous read and write transfers.

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.

2.5 Burst Transfer Styles

Only applicable to four person projects as done in prior semesters. Not applicable to teams three or smaller.

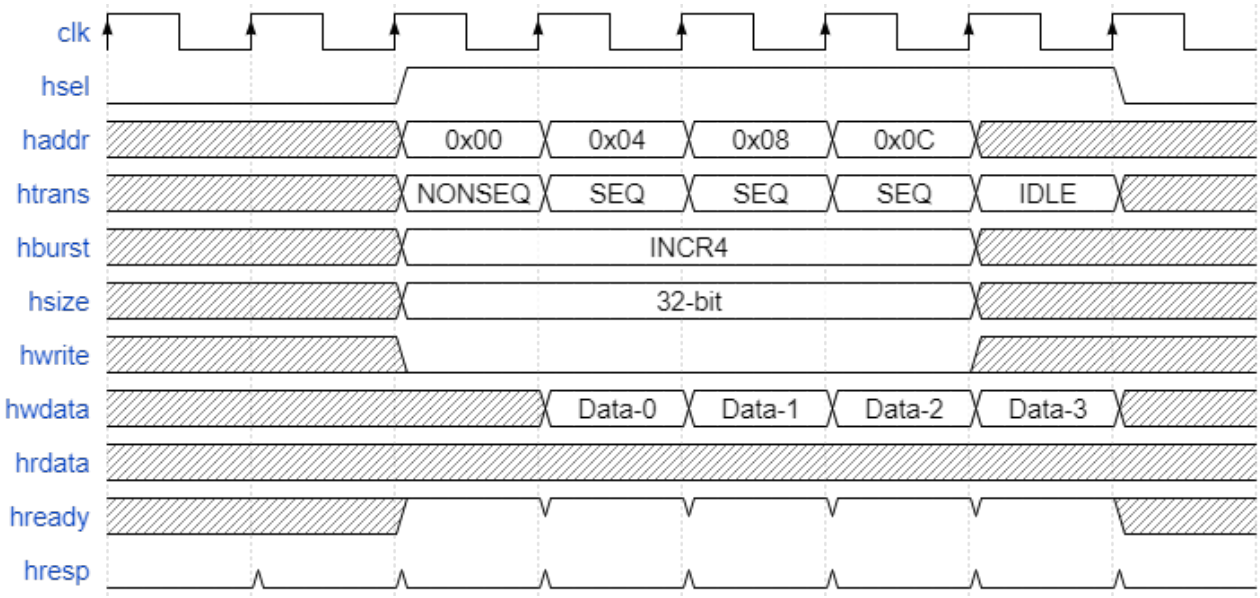

Figure 5: Example of a 4-beat incrementing write burst transfer

NOTE: There is no control or timing difference between read and write bursts, only which data bus is active.

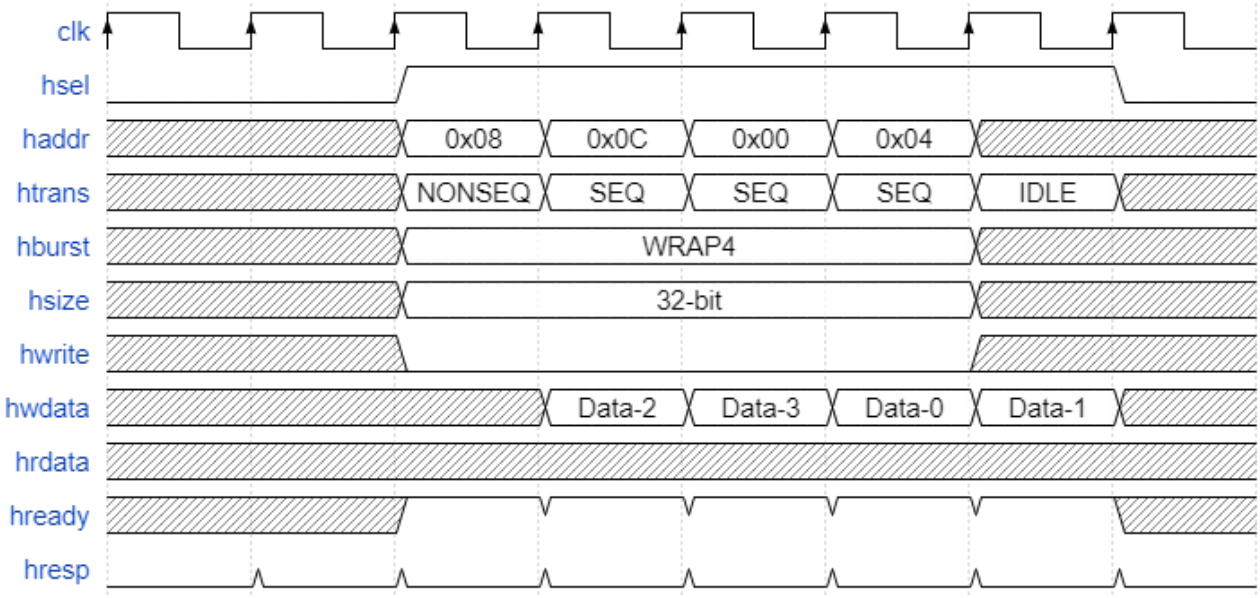

Figure 6: Example of a 4-beat wrap-around write burst transfer

3 Convolution Process

You are likely already familiar with the term "convolution" as mentioned in signal processing applications. Conceptually, it is a simple mathematical operation on two functions that produces a third function, which expresses how the shape of the first function is modified by the second. The implementation in this lab is similar to that of Lab 7, except the convolution will be carried out by modules of your own design, and the convolution is carried out across 3 rows of 3 coefficients and 3 samples.

3.1 Expectations for project

Your design is only required to implement the ability to input coefficients, input samples, and output convolution results as described in this manual. It is not up to your design to handle the organization and ordering of the sample rows of management of the overall convolution process. Instead, that is the job of the CPU in the SoC to handle.

3.2 Convolution Process

The convolution implementation in this lab uses a 3x3 "kernel" or convolution area. 3 rows of three coefficients each are multiplied with 3 rows of 3 samples each, and then the products from these 9 multiply operations are summed together to form the result. In an image processing or computer vision application, the kernel would start at the top-left of the image, perform a multiply/add operation, then shift to the right by one sample column and repeat. Once the kernel shifts in the last column available, it returns back to the left side of the image, moves down one sample row, and repeats the entire process again. This continues until the kernel has traversed the entire image.

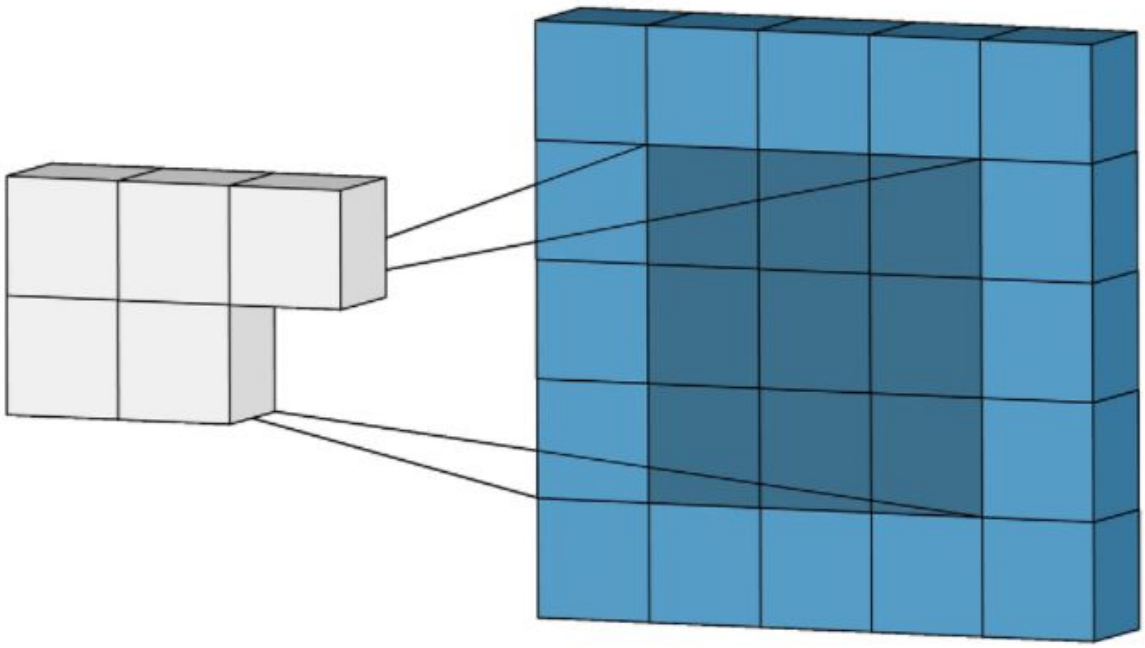

Figure 7: Snapshot of 3x3 kernel convolution (the blue grid are the samples, the white blocks are the results)[2]

Again, in this project you do not have to worry about the exact traversal of the convolution kernel. Your design simply needs to accept coefficients, samples, and control signals from the CPU through the AHB bus.

3.3 Convolution Packet Types

Sample Column A column of 3 4-bit samples (12-bit value total).

Coefficient Column A column of 3 4-bit coefficients (12-bit value total).

Command Byte A single 8-bit value, with certain bits corresponding to certain commands.

4 Design Architecture

A full SoC peripheral module contains both the SoC bus related interface module and the functional modules needed to implement and execute the peripherals communication standard. In this over all design the SoC bus interface is a more fully capable AHB-Lite bus, and the peripheral functions as a 2D Convolver.

NOTE: There are significant differences between the design requirements for 4-person and 3-person teams. Therefore, teams should only depend on the content of design manuals for their respective team size.

4.1 2D Convolver End-point AHB-Lite SoC Module Design Architecture

The starting architecture your team must follow (other than potential internal signal additions) is depicted in Figure 8.

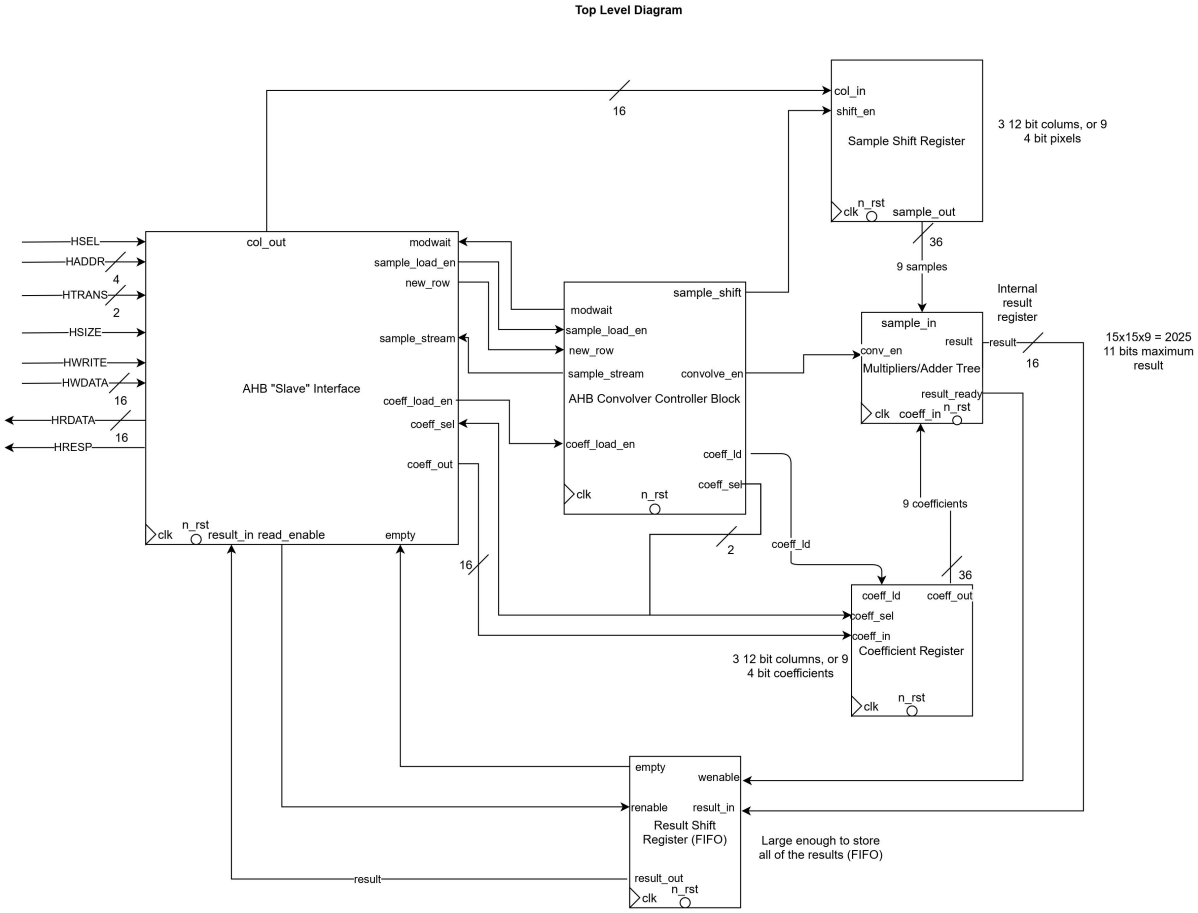

Figure 8: Architecture for 2D Convolver End-point AHB-Lite SoC Module

You may add, remove, or redefine the internal signals that go between the AHB-lite slave and other blocks in this design. Consequently it is up to you determine how exactly these signals will be used. It should be similar but not necessarily identical to what is presented here. What matters is that your design is capable of receiving the required data packets, performing the necessary convolutions, and responding appropriately to commands on the AHB-lite bus.

NOTE: Clock and Reset signals are assumed to be sent to the appropriate blocks and are represented with triangles and circles in Figure 8

Similar to during Lab 9, the security ('HPROT') and atomicity ('HMASTERLOCK') signals are absent from this design because they are not needed or relevant given the functionality of the design.

Unlike in Lab 9, both HREADY feedback (via 'HREADYOUT') and burst transfer control (via 'HBURST') are present on this design. However, even though HREADY feedback is present, the slave really only has a valid need for delaying/extending the bus transfers under error conditions as, there should not be any transfers that required longer than one address cycle or one data cycle to properly respond. All status checking or valid data supply/retrieval is expected to be enforced by software running on the SoC core and via polling-style data status checks.

Furthermore, in order to focus the design work in meaningful way for a 3-person team the burst transfer support is not required or for this design and thus the 'HBURST' signal is not present.

4.1.1 List of the Functional Units/Blocks and Purpose

Multiplier / Adder Tree This module handles all the mathematical work needed by multiplying all 9 coefficients with its respective sample then summing these products to obtain the result.

Coefficient Register This module handles the storage and supply of all 9 4-bit coefficients through the use of 3 12-bit coefficient column registers

Sample Shift Register This module handles the shifting, storage, and supply of all 9 4-bit samples through the use of 3 shifting 12-bit sample column indices.

Result FIFO Buffer This module handles the buffered storage of results, in a first-in-first-out manner, through the use of 1352 16-bit registers (the exact size of this buffer can vary based on the necessary application. This configuration is much larger than necessary for the purposes of this lab).

Controller This module handles the coordination of the various states needed to implement all the functionality of the 2D Convolver.

AHB-Lite-Slave This module handles all AHB-Lite-Slave Interface specific functionality. [THREE PERSON TEAM ONLY]

4.1.2 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.

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).

4.1.3 Description of internal interface signals

Below is a list of some of the internal signals that might need a bit more explanation. You are free to add, remove, or modify any of the internal signals of this design as long as your design exhibits the proper behavior at the top

level ports described previously. At the very least, you will have to define some of these signals in more detail than is provided here. However, for two member teams, most of these internal signals will be part of the DUT port map since the two person design will not include the AHB slave.

Modwait When the Convolver module is busy performing some task, this will be asserted to indicate this fact.

New Row This signal will indicate to the controller when the convolution kernel is skipping down to a new row (in which case three new sample columns need to be loaded before convolution continues).

Sample Stream Indicates when the Convolver module has convolved the current samples and is ready to shift in a new sample column.

Empty Indicates when the FIFO is empty, so that the CPU knows when it has read all results (You may want to consider adding a full indicator as well but it is not required).

Result Ready Indicates to the FIFO buffer that the result is ready to be stored (activates the write-enable "wenable" on the FIFO buffer).

4.2 Required AHB-Lite-Slave Address Mapping

Table 3: 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 '1' for bit-0 → Convolver is busy (modwait) Value of '1' for bit-7 → FIFO is empty (empty) Value of '1' for bit-8 → Sample streaming mode active (sample stream)
0x2	2	R	FIFO Result Register: Reading from this register will read from the FIFO. Renable will be automatically pulsed. Reading when empty will have no effect.
0x4	2	R/W	New Sample Column Register: Accepts 12-bit sample columns. Each column should contain 3 4-bit samples.
0x6	2	R/W	R0 Coefficient Column Register: Accepts 12-bit coefficient column R0. Each column should contain 3 4-bit coefficients.
0x8	2	R/W	R1 Coefficient Column Register: Accepts 12-bit coefficient column R1. Each column should contain 3 4-bit coefficients.
0xA	2	R/W	R2 Coefficient Column Register: Accepts 12-bit coefficient column R2. Each column should contain 3 4-bit coefficients.
0xC	1	R/W	Command / Control Register: Value of '1' for bit-0 → Load coefficients. Value of '1' for bit-1 → Load new sample column. Value of '1' for bit-2 → Begin load new convolution row. Value of '1' for bit-1 and bit-2 → Sample complete.

All values are to be handled according to 'Little Endian' notation.

4.3 Valid Design Usage

4.3.1 Coefficient Loading

The valid flow of operations for loading coefficients into this design is as follows:

1. The SoC core will write the first coefficient column (left-most column in the kernel) into the R0 Coefficient Column Register.
2. The SoC core will write the second coefficient column (middle column in the kernel) into the R1 Coefficient Column Register.
3. The SoC core will write the third coefficient column (right-most column in the kernel) into the R2 Coefficient Column Register.
4. Once all three columns have been written successfully, the SoC core will assert bit 1 of the Command / Control Register (address 0xC) to start the coefficient loading process.
5. The SoC core will check the status register repeatedly, checking bit 0 (modwait).
6. Once the SoC core sees that modwait has been deasserted, the coefficient loading process will be complete.

4.3.2 Convolution

The valid flow of operations for using this design to convolve a sample set is as follows:

1. The SoC core will ensure the coefficients have already been loaded.
2. The SoC core will write the first sample column (left-most column in the kernel) into the sample column register, then assert bit 0 of the Command / Control register (address 0xC) to shift the column into the sample register.
3. The SoC core will repeat the sample write and shift process for the other two sample columns in the kernel.
4. At this point, the first kernel will be convolved and the result will be placed into the FIFO buffer shortly (indicated by the deassertion of the "empty" bit in the status register).
5. The SoC core will continue to stream samples using the write and command process.
6. Once the kernel reaches the end of its current row in the sample, the SoC core will then assert the new row bit in the Command / Control register and load three samples using the standard procedure before convolution continues.
7. Upon reaching the end of the sample, the SoC core will then assert the appropriate bits in the Command / Control register to indicate completion of the sample to the convolver.
8. Finally, the SoC core will reach results from the FIFO result buffer one at a time until the empty bit is asserted in the status register to indicate that all results have been read.

5 Specifications for the Blocks You Must Design and Implement

5.1 2D Convolver End-point AHB-Lite SoC Module

5.1.1 Block Description

This is the full design module that connects the dedicated AHB-Lite-Slave interface and all of the other sub-blocks to form the full peripheral.

Since there will not be an electronic grading script for this design, the exact names used for the filename and ports does not matter as long as they are logical and descriptive of their intended function.

The target clock rate for this design must be 100 MHz.

5.2 AHB-Lite-Slave Interface [3 PERSON ONLY]

This block handles all AHB-Lite specific work and must be composed of at least the following component modules:

Address Decoder This block should handle the decoding of raw addresses into value specific selection and write control signals.

HRDATA Register The value for the HRDATA bus must be registered, this is to prevent critical path lengthening and bus synchronization issues for the AHB-Lite interconnect.

Value Registers Every accessible value that is not fully available via port on the module must be held in a dedicated register. These may be distinctly named or handled via an array of value registers.

5.3 Controller

This block handles all of the transitions between the various states of operation for the 2D Convolver. Its inputs must be closely tied to those of the Slave Interface module for quick and responsive operation. In the reference architecture, its outputs interface with almost every other block in the device, including the sample shift register, multiplier/adder tree, and coefficient register. However, you are free to alter this internal structure as long as the top-level behavior is correct.

5.4 Sample Shift Register

This block handles all work related to the shifting, storage, and supply of the 9 4-bit samples. It should make use of 3 12-bit shift positions, each holding a sample column of 3 4-bit samples. Its input in the reference design is 16 bits wide to allow for easier future expansion of sample/kernel size, however only the bottom 12 bits are used in the current implementation. All 36 bits of sample data are outputted simultaneously through its output in the reference architecture.

5.5 Multiplier / Adder Tree

This block handles all of the mathematical work needed to carry out the convolution. It does this by multiplying all 9 coefficients in the 3x3 kernel matrix with the corresponding 9 samples the kernel is aligned with. Below is an example calculation for reference.

Table 4: Table displaying a sample calculation conducted by the Multipier / Adder Tree sub-block

Sample Register	Coefficient Register	Calculation	Result
1 4 7 2 5 8 3 6 9	1 4 7 2 5 8 3 6 9	1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6 + 7*7 + 8*8 + 9*9	285

5.6 Coefficient Register

This block handles all work related to the storage and supply of the 9 4-bit coefficients. It makes use of 3 12-bit addressable registers to load and store the coefficients, and one 36-bit parallel output to supply all 36 bits of coefficient data. In the reference design, the input is 16 bits wide to allow for easier future expansion of sample/kernel size, however only the bottom 12 bits are used in the current implementation.

5.7 Result FIFO Buffer

This block handles all work related to the buffered storage of results outputted by the multiplier/adder tree. The reference design uses 1352 16-bit registers, however this is only to handle potentially large samples and not necessary for demonstration purposes. A write-enable and read-enable signal are used and control the position of write and read pointers within the buffer. Once central clock can be used for both the write and read controllers, as the input and outputs sides of the buffer are on the same clock domain. It may be useful to add a clear input, although this is not necessary as indicated by its absence in the reference architecture.

5.8 Required Workload Distribution

All design and full module verification work must be distributed in the following manner:

- Team Member 1:
  - Design, implement, and test the Coefficient Register Design, implement, and test the FIFO Result Buffer
- Team Member 2:
  - Design, implement, and test the Sample Shift Register
  - Design, implement, and test the Multiplier / Adder Tree
  - Design, implement, and test the Controller
- Team Member 3:
  - Design, implement, and use bus model to test the AHB-Lite Slave
  - Design and implement top level test bench.

6 Closing Remarks

- It is strongly recommended that you review RTL diagrams, pseudo-code, list of test cases, etc. with course staff as you develop them. Even though these items will be graded, you will need feedback as soon as possible to stay on track.

Bibliography

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

[2] Irhum Shafkat. Intuitively understand convolutions for deep learning. page 1.