ASIC Design Laboratory
Lab 8 : Introduction to System-on-Chip Bus Protocols
(APB-Slave UART Receiver SoC Peripheral)
Lab Manual

Fall 2020

The purpose of this lab exercises is to help you become familiar with simple System-on-Chip bus protocols by working the APB protocol which is both one of the most widely used simple SoC bus protocols and contains a lot of the core aspects used in more advanced SoC bus protocols. Since SoC bus protocols are primarily used for integrating peripheral hardware modules with a core processor, you will be extending your completed UART design from the prior lab to become a APB-connected UART SoC peripheral module.

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 APB-Slave interface module
- Extend your prior UART Receiver design to allow some UART settings to be configurable via design ports.
- Integrate your completed APB-Slave interface module with your extended UART Receiver module to create an APB-connected (and configurable) UART Receiver module.
- Develop test benches for verifying the APB-Slave functionality of your design using a provided APB bus-model.
- Synthesize the APB-UART Receiver design using Design Compiler®
- Test the Synthesized/Mapped version of the APB-UART Receiver design
- Submit electronically your completed APB-UART Receiver design to be graded

1 Lab Setup

In a UNIX terminal window, issue the following commands, to setup your Lab 8 workspace:

mkdir –p ~/ece337/Lab8
cd ~/ece337/Lab8
dirset
setup8

The setup8 command is an alias to a script file that will check your Lab 8 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 APB Preparation

To prepare for implementing your APB-UART Receiver design you must complete the following:

- Create a complete RTL diagram for the APB-Slave interface module described in Section 5.2
- Create a complete State Transition Diagram for the APB-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 Lab8prep" 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 8 and the Remaining Labs

In this lab, you have been given access to a library that contains complied and verified modules for implementing the bus model you will be using to verify your slave-interface's functionality, which is described in Section 6. As these 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 APB-Slave Interface Verification

Since this lab is introducing the concept of using predesigned bus models to verify that your design complies with the required bus standard, you are provided with a starter test bench for testing your APB-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 APB-Slave Interface's data buffer
- Test for correct operation during Master reads of the APB-Slave Interface's status register
- Test for correct operation during Master reads of the APB-Slave Interface's configuration registers
- Test for correct operation during Master writes to the APB-Slave Interface's configuration registers
- Test for correct operation during Master writes to the APB-Slave Interface's stats register
- Test for correct operation during Master writes to the APB-Slave Interface's data buffer

2.3.2 Required Minimum APB-UART Peripheral Verification

Additionally, you will need to create a test bench that verifies the operation of the full APB-UART peripheral design. It is recommend that you do this by extending a copy of the APB-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 APB-Slave Interface and UART test benches. The minimum requirements for this test bench are as follows:

- Must use tasks for consistent execution of data streaming, power-on-reset, and output checking
- Must correctly use the provided APB bus model during APB bus related operation(s)
- Must correctly test for valid UART packet handling according the specifications in Lab6
- Must have a task for configuring the design via the APB interface
- Must correctly test for valid UART packet handling under the various configurable UART settings

2.4 Grading Policy

More than seventy percent (>70%) 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 8 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% (15/30) or higher on your most recent mapped version test in order to satisfy the outcome for this lab.

2.5 Submission Commands

submit Lab8prep Submits the contents of your "docs" folder for the preparation phase and will be due prior to the main design submission

submit Lab8 Submits your design for automated grading

submit Lab8re 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 Lab8r 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 Peripheral Bus Protocol

The Advanced Peripheral Bus (APB) protocol[1] is one of many that form the Advanced Microprocessor Bus Architecture (AMBA) Bus System design by ARM. At a high-level, APB 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. APB was designed to support low-speed peripherals where both latency and throughput are not strong concerns and typically the bus activity happens far faster than any external activity related to attached peripherals. For example, with UART entire packets would be read in one two-cycle bus transaction (or two of them if status checking is used), while the actual reception of the packet would take from hundreds to thousands of cycles depending on the selected UART baud-rate. For this reason, this standard is intentionally kept fairly simplistic for SoC buses and does not support any overlapping or bursting of transfers, unlike higher-throughput buses such as AHB-Lite (which you’ll work with in a later lab).

Like most SoC buses, APB 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 APB Signals

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

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

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

PADDR This is the address bus and can be up-to 32-bits wide.

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

PSELx 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.

PENABLE This signal indicates that the transaction is now in the second stage (’data’ stage)

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

PWDATA This is the data bus used for transferring the data written to the ’slave’ device and can be up to 32-bits wide.

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

PSTRB This is the set of byte-wise write enables which designate which byte(s) from the PWDATA bus should be written into the ’slave’ device during this transfer. There will be one for each byte present in the PWDATA bus. These must not be active during read transactions.

PREADY 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.

PSLVERR 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 APB 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.

3.2 Example Operation of a Master Writing to a Slave Device

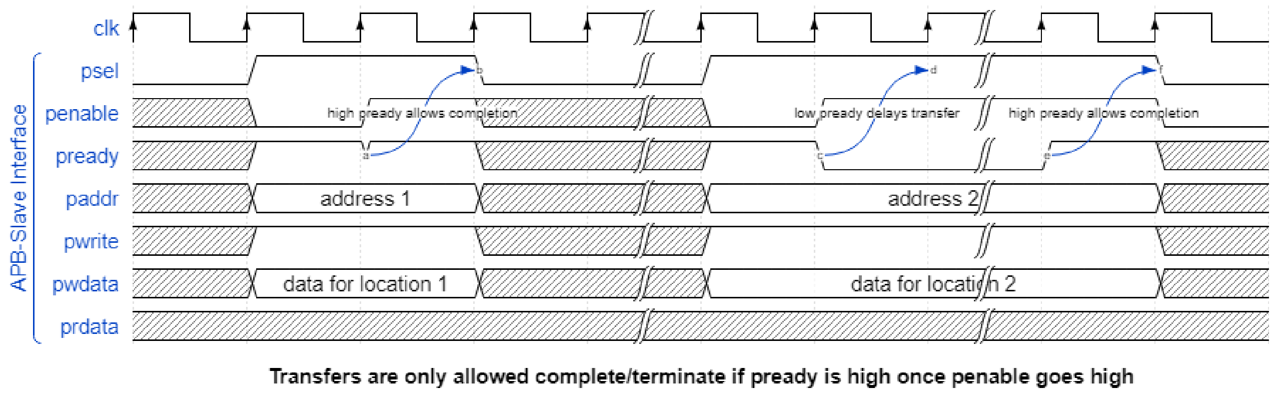
clk
psel
penable
pready
paddr
pwrite
pwdata
prdata
APB-Slave Interface
high pready allows completion
low pready delays transfer
high pready allows completion
address 1
address 2
data for location 1
data for location 2
Transfers are only allowed complete/terminate if pready is high once penable goes high

Figure 1: Example operation and timing of APB Writes from the perspective of a 'slave'

Write transfers are only allowed to finish and release the bus once the 'slave' maintains the 'pready' signal at a logic-high value during the second stage ('penable' is a logic-high), to indicate that the write was either completed or at least internally buffered depending on the design. If the 'pready' signal is at a logic low value then the bus (and involved master) must stall for as long as 'pready' stays at a logic low value. This can result in indefinite stall or 'freezing' of the bus if 'pready' is misused.

3.3 Example Operation of a Master Reading from Slave Device

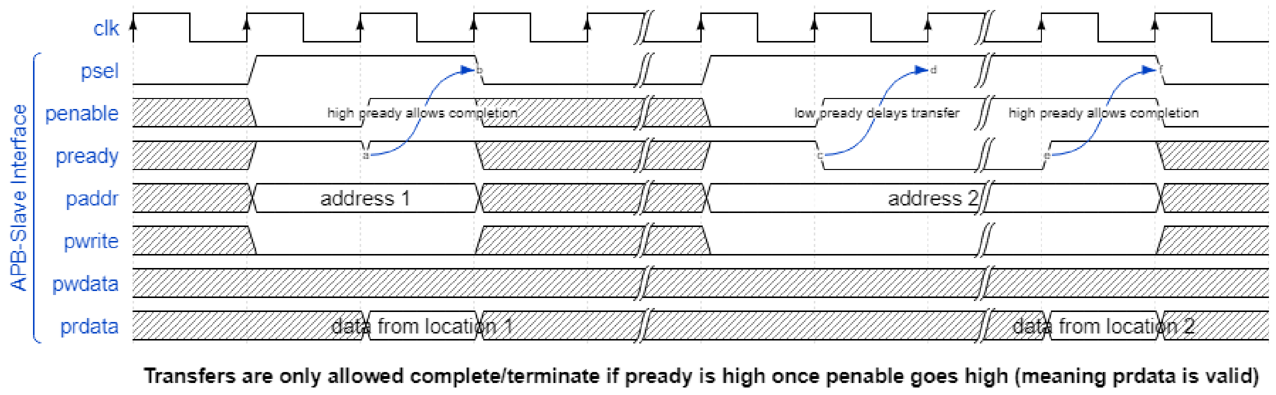
clk
psel
penable
pready
paddr
pwrite
pwdata
prdata
APB-Slave Interface
high pready allows completion
low pready delays transfer
high pready allows completion
address 1
address 2
data from location 1
data from location 2
Transfers are only allowed complete/terminate if pready is high once penable goes high (meaning prdata is valid)

Figure 2: Example operation and timing of APB 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 'pready' signal at a logic-high value during the second stage ('penable' is a logic-high), to indicate that the data requested has been available on the 'prdata' bus. If the 'pready' signal is at a logic low value then the bus (and involved master) must stall for as long as 'pready' stays at a logic low value. This can result in indefinite stall or 'freezing' of the bus if 'pready' is misused.

4 Design Architecture

A full SoC peripheral module contains both the SoC bus related interface module and the peripheral's functional hardware. To simplify verification and design, and re-usability with different SoC bus standards, all of the peripheral's functional hardware is bundled in it's own module (in this case your UART Receiver module). The means that the top-level file simply serves to bundle the chosen SoC bus interface module with the functional module.

4.1 APB-UART Receiver Peripheral Design Architecture

The architecture for the APB-UART Receiver Peripheral design you will be completing during this lab is shown in Figure 3.

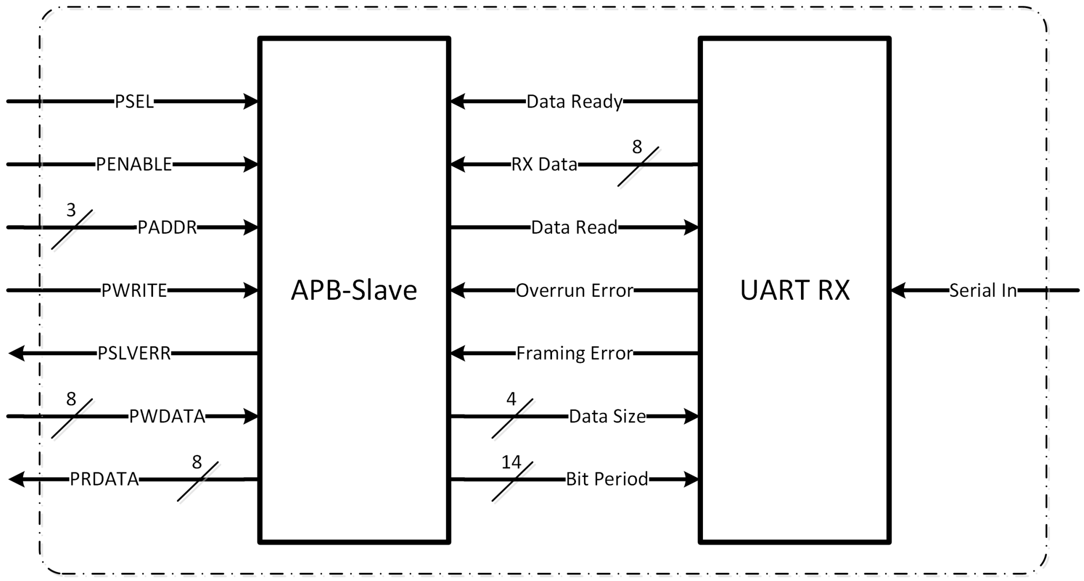

Figure 3: APB-UART Receiver Peripheral Architecture Diagram

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 APB protocol signals are missing from the 'slave' interface on this architecture. Within APB there are several signals that are optional for use depending on what the specific 'slave' interface is being designed to support. This design does not have a functional need or benefit from varying protection levels during APB transactions, and thus it will not use the the optional 'PPROT' signal. Since UART packets are at most one byte in size and we only buffer one packet at a time within the UART design, we will only use 8-bit data buses and do not need a 'PSTRB' signal since it would be only 1-bit and always match the value of the 'PWRITE' signal. Furthermore, the 'PREADY' signal will not be used for two reasons: (1) the UART protocol receives packets at a significantly slower rate than transactions happen on the APB bus and the only situation that might make sense for a UART related pause is while waiting for a new packet to arrive and (2) requiring the end user/system to poll (periodically check) status register(s) will allow other devices in the SoC to use the bus while the system waits for new data to arrive.

4.1.1 List of the Functional Units/Blocks and Purpose

UART RX This handles all UART specific functionality and is an extended version of your prior UART RX design.

APB-Slave This module handles all APB-Slave Interface specific functionality.

4.1.2 Required Configurable UART Settings

This design will be supporting more of the configurable options for UART that were previously fixed as specific values in the prior lab. It will retain the fixed settings of no parity checking and one stop bit but will allow the data rate and data size to be configured as indicated in Table 1.

Table 1: Table of required configurable UART settings

Setting Name	Accepted Value(s)	Description
Bit Period	[10, 16383]	Determines the nominal length of packet bits in terms of clock cycles
Data Size	5,7,8	Determines the number of bits in packet data payloads

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 'PREADY', 'PROT', and 'PSTRB'.

For this design, all unused bits of a register's value must be zero-filled before being put on the read data bus.

Furthermore, given that SoC data buses often are very long, involve a significant amount of routing circuitry, and still need to stay closely synchronized it is standard practice (and required for this class) to have the 'PRDATA' bus be driving directly from a register. This helps with bus drive strength (and thus update rate), allows for clean power saving design approaches when the bus is not supposed to be active, maximizes the amount of propagation time allocated for the long and slow bus wires and routing, and minimizes critical path issues by isolating any true data sources for the 'PRDATA' value behind the register.

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

Serial In This is the input port that is connected to the UART serial connection

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

PSEL This is the 'slave' selection signal.

PENABLE This indicates that the transaction is now in the second stage ('data' stage)

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

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

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

PSLVERR 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 APB-Slave Address Mapping

Table 2: Table of the required APB-Slave address-to-value mapping

Address	Value Size (Bytes)	Access Mode	Description
0x0	1	R	Data Status Register: Value of '0' → no data, Value of '1' → new data is present
0x1	1	R	Error Status Register: Value of '0' → no error, Value of '1' → framing error, Value of '2' → overrun error
0x2	2	R/W	Bit Period Configuration Register
0x4	1	R/W	Data Size Configuration Register
0x6	1	R	Data Buffer

NOTE: The address-to-value mapping in Table 2 was intentionally chosen so that it would naturally work well for both 8-bit and 16-bit data APB bus configurations, since SoC bus fabrics often handle connections of different sizes.

4.3 Design Operation

The typical flow of operation for this design is as follows:

1. Configuration of UART settings, unless they have already been set to the same values from prior operation. (Order of setting configuration will not matter)
2. Poll the Data Status Register until new data is present. The status must be updated to indicate the presence of new data as soon as it is available after being received validated by the UART component.
3. Read the data from the Data Buffer.

Additionally the host SoC may desire to perform error checking as well which would sensibly be in the form of checking the Error Status Register before it checks the Data Status Register during each polling attempt.

Required specific operational constraints:

- Given that the 'Data Status' and 'Data Buffer' registers are functionally coupled, their states must match after each APB transfer. For example, When data is read from the buffer, the state to the data status must transition back to 'no data' with the completion of the read.
- Since UART packet data values are normally considered to be unsigned integers, the value must be zero-extended to an 8-bit value before putting it on the APB bus during a read of the 'Data Buffer'.

5 Specifications for the Blocks You Must Design and Implement

5.1 APB-UART Receiver Peripheral

5.1.1 Block Description

This is the full design module that connects the dedicated APB-Slave interface and UART Reciever modules together to form the overall SoC peripheral.

5.1.2 Module Specifications

Required Module Name: apb_uart_rx

Required Filename: apb_uart_rx.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.
serial_in	input	This is the serial input signal and thus will contain the data that is being serially transmitted to the UART. This line’s idle value is a logic ‘1’.
psel	input	The slave’s selection signal.
paddr[2:0]	input	The slave’s address bus.
penable	input	Indicates that the transaction is now past the first stage.
pwrite	input	Indicates the transaction mode (’0’ for Read, ’1’ for Write)
pwdata[7:0]	input	The write data bus.
prdata[7:0]	output	The read data bus.
pslverr	output	The active-high transaction error feedback from the ’slave’ (must be asserted when a transaction error occurs (such as writing to a read-only address).

5.2 APB-Slave Interface

5.2.1 Block Description

5.2.2 Module Specifications

Required Module Name: apb_slave

Required Filename: apb_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.
rx_data[7:0]	input	This is the 8-bit data that has been received by the UART module and is available for reading.
data_ready	input	This is the active-high data ready signal for the rx_data port.
overrun_error	input	This is an active high flag signal that reports if an overrun error condition has occurred.
framing_error	input	This is an active-high flag signal that reports if a framing error occurred with the current/most recently received packet.
data_read	output	This is the active-high handshake signal for the UART module and is asserted when the SoC has read the available data.
psel	input	The slave’s selection signal.
paddr[2:0]	input	The slave’s address bus.
penable	input	Indicates that the transaction is now past the first stage.
pwrite	input	Indicates the transaction mode (’0’ for Read, ’1’ for Write)
pwdata[7:0]	input	The write data bus.
prdata[7:0]	output	The read data bus.
pslverr	output	The active-high transaction error feedback from the ’slave’ (must be asserted when a transaction error occurs (such as writing to a read-only address).
data_size[3:0]	output	This is the value from the Data Size configuration register
bit_period[13:0]	output	This is the value from the Bit Period configuration register

5.3 UART Receiver Module

5.3.1 Block Description

This is the extended version of your UART Receiver design from Lab 6. The only changes from your prior design should be enabling the operation of the timer to be adjusted based on the configuration register values.

5.3.2 Module Specifications

Required Module Name: rcv_block

Required Filename: rcv_block.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.
data_size[3:0]	input	This is the value from the Data Size configuration register
bit_period[13:0]	input	This is the value from the Bit Period configuration register
serial_in	input	This is the serial input signal and thus will contain the data that is being serially transmitted to the UART. This line’s idle value is a logic ‘1’.
data_read	input	This is the active-high handshake signal for the data buffer and is asserted by an external device when it has read the available data.
rx_data[7:0]	output	This is the 8-bit data that has been received by the UART signal and is available for reading by an external device.
data_ready	output	This is the active-high data ready signal for the rx_data port. It is asserted when new data is available to be read and cleared when the data is read.
overrun_error	output	This is an active high flag signal that reports if an overrun error condition has occurred. It is cleared when an external device reads the available data.
framing_error	output	This is an active-high flag signal that reports if a framing error occurred with the current/most recently received packet. It is cleared when a new packet is starting to be received.

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 APB 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 1)

ADDR_WIDTH The size of the slave's address bus, in terms of bits. (Default value of 3)

Module Name: apb_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 that you would like to have the bus model execute. There is no hard limit to the number of transactions that can be enqueued.
transaction_write	input	This must be a logic '1' if the transaction being enqueued is supposed to be a write and a '0' if it is 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_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.
psel	output	The selection signal for the slave.
paddr[#:0]	output	The address bus for the slave. (Sized based on 'ADDR_WIDTH')
penable	output	The enable signal for the slave.
pwrite	output	The write/read mode signal for the slave.
pwdata[#:0]	output	The write data bus for the slave. (Sized based on 'DATA_WIDTH')
prdata[#:0]	input	The read data bus for the slave. (Sized based on 'DATA_WIDTH')
pslverr	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 9.
- 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 APB Protocol Specification v1.0*. 2004.