ECE 337 Lab 5: Design of a Receiver Block for a Universal Asynchronous Receiver/Transmitter (UART)

0. Lab Overview

In this lab, you will:

- Design and test via a test bench the timer unit for the Receiver Block of the UART
- Design and test via a test bench the receiver control unit (RCU) for the UART Receiver Block
- Combine the RCU block, timer block, and the given blocks in order to create the UART Receiver Block
- Develop a test bench from a given template to test the functionality of the UART Receiver Block created
- Synthesize the UART Receiver Block using Synopsys
- Test the Synthesized/Mapped version of the UART Receiver Block
- Submit electronically your completed UART receiver block to be graded

For this lab, you will be working with a partner with whom you are to meet and collaborate on the design and testing of your solution for lab 5. The collaboration ground rules are posted in the Lab Exercises section of Blackboard in the Team Reports document.

0.1. Lab Setup

Setup up your "Lab5" folder as in previous labs. Then in a UNIX terminal window, issue the following command:

```
setup5
```

Also make sure to import your fresh lab directory into SVN and checkout a working version before proceeding with the lab. **Once you have checked out a working copy of your lab 5 directory, have a TA check off your work up to this point.**

0.2. Required Preparation

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

- Create a complete state transition diagram for the Receiver Control Unit (RCU) described in section 3.3
- Create a complete pseudo-code for the timer unit described in 0
- Create complete RTL diagrams for the RCU block described in section 3.3
- Create a block diagram showing how you are going to build your timer unit using the flex_counter module from lab 3

You must have these diagrams both signed off and submitted via the "submit Lab5Prep" command no later than the following deadlines

- Tuesday Labs: **End of Thursday office hours**
- Wednesday Lab: **End of Monday office hours**
- Thursday Lab: **End of Tuesday office hours**

It is highly recommended that you complete all of these and have a TA sign off on them prior to starting to write any design code, as this should save you tremendous amounts of time debugging/rewriting code later on.

Note: All diagrams, both RTL and state transition diagrams, must be done as a digital drawing. Hand drawn diagrams (even if they are scanned) will receive a grade of zero points. There a multiple easy ways to make digital diagrams available to you in the ECN & ITAP labs and through free software. Two recommendations are Microsoft's Visio (available in Windows labs on campus) and the free ware program called DIA (available for both Linux and windows machines).

Note: All materials must be signed off as well as submitted electronically as stated above, with diagrams as either individual pdf files or image files.

0.3. Expectations Regarding Lab 5 and the Remaining Labs

In Lab 5 you will be working on part of the design of a Universal Asynchronous Receiver Transmitter (UART). From the first day of class, you have been gathering the knowledge and expertise with the tools to allow you to complete this design. At this point in the course, you should know, and will be expected to know, how to operate all the tools that were introduced to you in Labs 1 through 3.

This lab is structured to mimic what you would encounter should you choose to pursue a career as an ASIC/VLSI designer upon graduation. Essentially you, the designer, are being provided with a set of specifications for a design, including the protocol that it is supposed to support/functionality it is supposed to perform and an architecture for how to modularize the needed internals of the design. In general, as you gain experience design ASICs and systems you will be expected to play increasing larger roles in developing the architecture for a design instead of just implementing a provided architecture. In industry, you will most likely working together with other people in designing your ASIC. Therefore some blocks in your design will be written by other people and you will be required to understand how they work in order to be able to interface them to create a

working design. **In this lab, you have been given complete blocks for most of the blocks other than the top level block**, which are described in section 0. You may modify or rewrite any of the given blocks to make them interact better with your written blocks as you see fit. You also have been given a test bench template. **You will need to design and test the blocks described in section 3**, which includes the top-level block. **You are not and will not being specifically instructed on how to design these blocks. 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 8 building blocks to form the Receiver block.**

0.4. Grading Policy

The grading of Lab 5 will be different from what you have encountered in the previous labs. In order to receive 100% on the lab you must produce a fully functioning Receiver Block. A fully functioning Receiver Block is defined as a design that passes all the tests that are contained within the automated grading test bench. The code for this test bench will not be provided to you nor will you be told what each test case in the test bench is checking. Eighty percent (80%) of your grade for this lab will be determined from your MAPPED version. The automated grading system will run the grading test bench on the MAPPED version of your design. Thus in order to run the automatic grading script, your Receiver Block design must have an error-free run through Synopsys.

You will be allowed a **maximum of 4 passes through the Lab 5 grading script**. In the real world you will likely be designing something because you need to create the first instance of it and will not have a gold model to verify with and will have to perform all testing through a well-designed test bench. Because of this, in this lab you will not be given a gold model; however you will be given a starter test bench to guide you in the creation of well-designed test benches. 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.

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 3.1 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 5 possible runs.

You will need to score 50% (30/60) or higher in your mapped version test in order to satisfy the outcome for this lab.

0.5. Post Lab

Place your responses to the questions posed in the Team Reports document in a file called Teamwork.txt your docs directory, and run:

```
submit Lab5Post
```

Thought Question: For this design, what is the maximum allowable tolerance of the data transmission frequency? Why?

0.6. Submission Commands

- "**submit Lab5Prep**" submits the contents of your "docs" folder for the preparation phase of the design
    - Tuesday Labs: **Due by end of Thursday office hours**
    - Wednesday Lab: **Due by end of Monday office hours**
    - Thursday Lab: **Due by end of Tuesday office hours**
- "**submit Lab5**" submits your design for automated grading
    - **Due by the start of Lab 6**
- "**submit Lab5Post**" submits the contents of your "docs" to turn in the teamwork report
    - **Due by the start of Lab 6**
- "submit Lab5r" submits your design for automated grading for remediation purposes (this will only be activated after the regular deadline has passed for all lab sections)

0.7. Additional Comments

A test bench template (tb_rcv_block.sv) has been provided for you.

Your mapped design must correctly run at 400 Mhz in order to pass the grading test bench and script.

Also, please remember from Lab 4 that the synthesis program will stop optimizing once it exactly meets your specified clock period and that the delays during the mapped simulations are sometimes larger than those calculated in by the synthesis program. Thus you should choose optimization settings that yield a buffer of positive net slack in the critical path during synthesis to account for this.

1. Universal Asynchronous Receiver/Transmitter Protocol

1.1. Introduction to UART

The UART bus protocol is a simple unidirectional asynchronous serial bus, which allows for data to be sent between hardware devices. It is generally used as a full-duplex point-to-point link between devices as show below in figure 1. Additionally, a design performing both reception and transmission is generally referred to as a transceiver. Also, packet transmissions can start at any time, as long as another is not currently in progress.

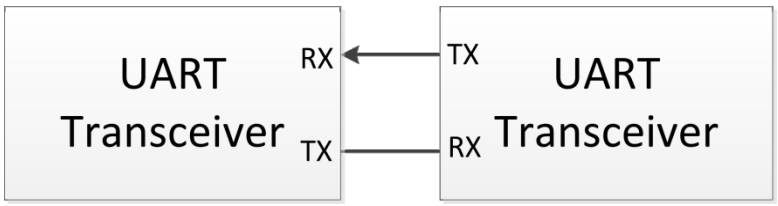

Figure 1: UART Transceiver Usage Diagram

1.2. Protocol Overview

The UART protocol is a fairly loose communication protocol in the sense that it really only regulates the general data packet format and size. All other parameters (specific timings, data rates, external connection media, voltage levels for wires, etc.) are left up to the designer and users. Although, there are several additional connection standards usually applied on top of the UART protocol for real systems, of which RS-232 (the common "serial" cable) is the most popular.

For this lab you will be using following parameters not defined by the protocol itself:

- The data rate will be 40 Mbps with a tolerance of +/- 4% (it is allowed to vary to a minimum of 38.4 Mbps and a maximum of 41.6 Mbps)
- The system will use a system clock 10x faster than the nominal data rate (400 MHz)
- The data packet will always contain 8 data bits
- The data packet will not contain a parity bit
- The data packet will terminate with 1 Stop-bit
- The transmitter will provide a minimum of 2 idle bit periods between a Stop-bit and the start bit of a new data packet
    - All errors must be reported through the corresponding flag signals by the ending of the first idle bit
- Data will be send Least Significant Bit (LSB) first
- All bits of a packet will be of uniform time length
- All flip-flops that store serial data or serial line samples must have each bit be reset to the idle line value when the asynchronous reset is activated

1.3. Data Packet Format

For a UART data transmission, the data is formatted into individual packets defined in order as:

- One (1) start bit that is always a logic '0'
- Five (5), seven (7), or eight (8) data bits, with the number of bits fixed for all transmission by an additional standard, the designer, or user. (RS-232 only allows either seven or eight bits)
- An optional parity bit
- One (1) or two (2) Stop-bits that are always a logic '1'

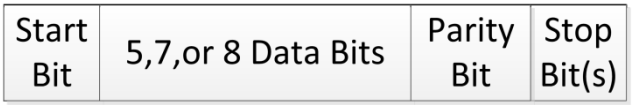

Figure 2: UART Protocol Packet Format

Also, between the last Stop-bit of a packet and the start bit of the next packet the connection is considered to be idle and a logic '1' must continually be transmitted.

If the parity bit is used and a packet's parity bit does not match the actual parity of the data then a "parity" error must be signaled to an external device of the UART receiver.

If a packet is incorrectly terminated (incorrect Stop-bit(s)) a Stop-bit error, also referred to as a framing error, must be signaled to an external device of the UART receiver.

1.4. Overrun Error

An overrun error occurs when a UART receiver receives a new packet before a valid previously received one was read from the internal data buffer of the UART receiver. When an overrun error condition occurs the old data should be overwritten by the new data, as the newer data is generally considered to be more important. The purpose of the overrun error flag is simply to alert the user/external device that it was too slow and thus has missed/lost some data.

1.5. General Sequence of Operations for UART Protocol

Since the UART protocol only governs the actual sending and receiving of a packet of data, each connection is treated as an independent connection and there is no required interaction between the receiver and transmitter of the same device, thus they can be treated as independent entities while designing a UART. Thus the sequence of operations for the receiver and transmitter are described in separate sections below.

1.5.1. Sequence of Operations for a UART Transmitter

1. The idle value (logic '1') is continuously supplied to the connection
2. When there is data to send:
   a. If parity is going to be used, then calculate and temporarily store the Parity-bit for the data
   b. Send the Start-bit (logic '0') for the packet
   c. Serially supply each of the data bits onto the connection
   d. If parity is going to used, then supply the Parity-bit
   e. Serially supply each of the Stop-bit(s) (all logic '1')
   f. Return to Step 1

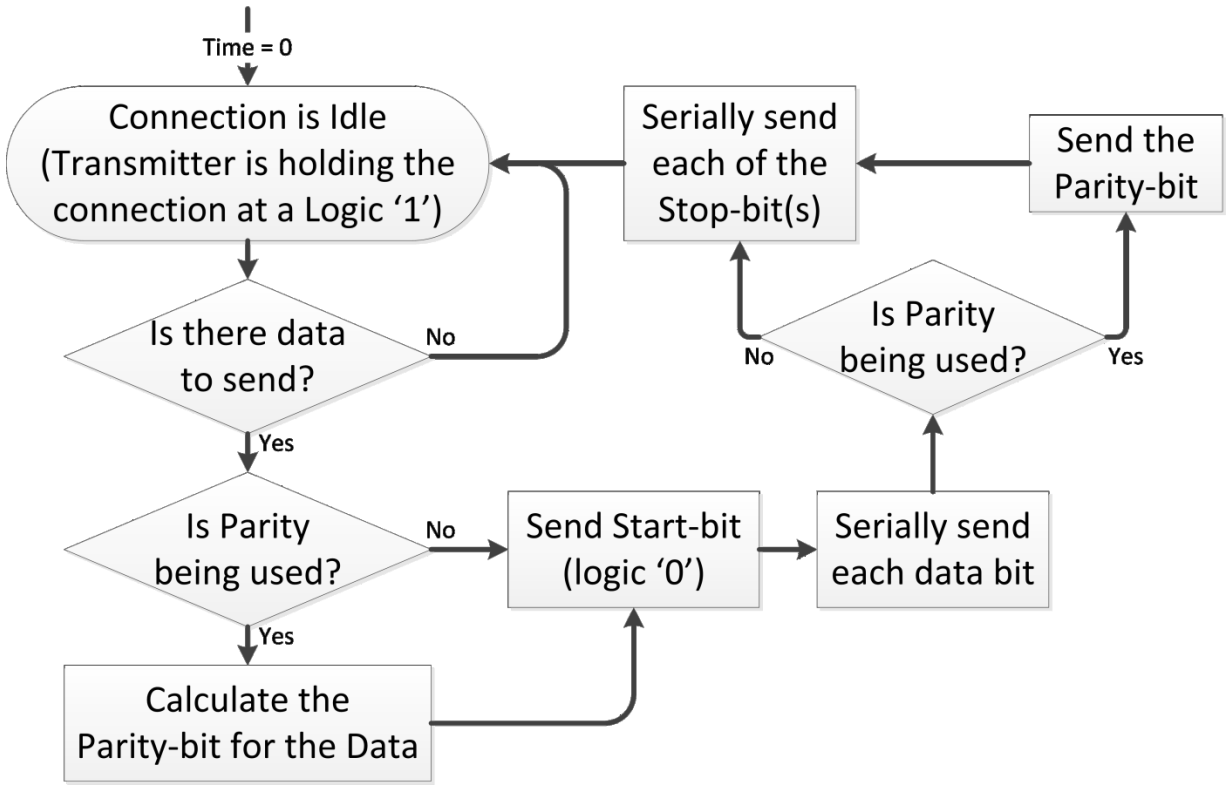

Figure 3: UART Transmitter General Sequence of Operations

1.5.2. Sequence of Operations for a UART Receiver

1. Wait for a falling edge to occur on the connection due to the presence of a Start-bit (logic '0') following either a Stop-bit (logic '1') or an Idle connection (logic '1')
2. Discard any prior packet based error condition (this is all errors except overrun error)
3. Sample and temporarily store each of the data bits
4. If Parity is used, then sample and temporarily store the Parity-bit
5. Sample and temporarily store each of the Stop-bit(s)
6. If the Stop-bit(s) are not correct then:
   a. Assert a Stop-bit or framing error signal to an external device of the device
   b. Discard the whole packet
   c. Return to step 1 and maintain the error signal during step 1
7. If parity is used and the Parity-bit does not match the actual parity of the data then:
   a. Assert a parity error signal
   b. Discard the whole packet
   c. Return to step 1 and maintain the error signal during step 1
8. If output previous data in the output buffer has not been read, then assert the overrun error signal
9. Move the temporarily stored data to an output data buffer
10. Return to step 1.

Also the overrun error should be cleared only when the output data buffer is read by an external device.

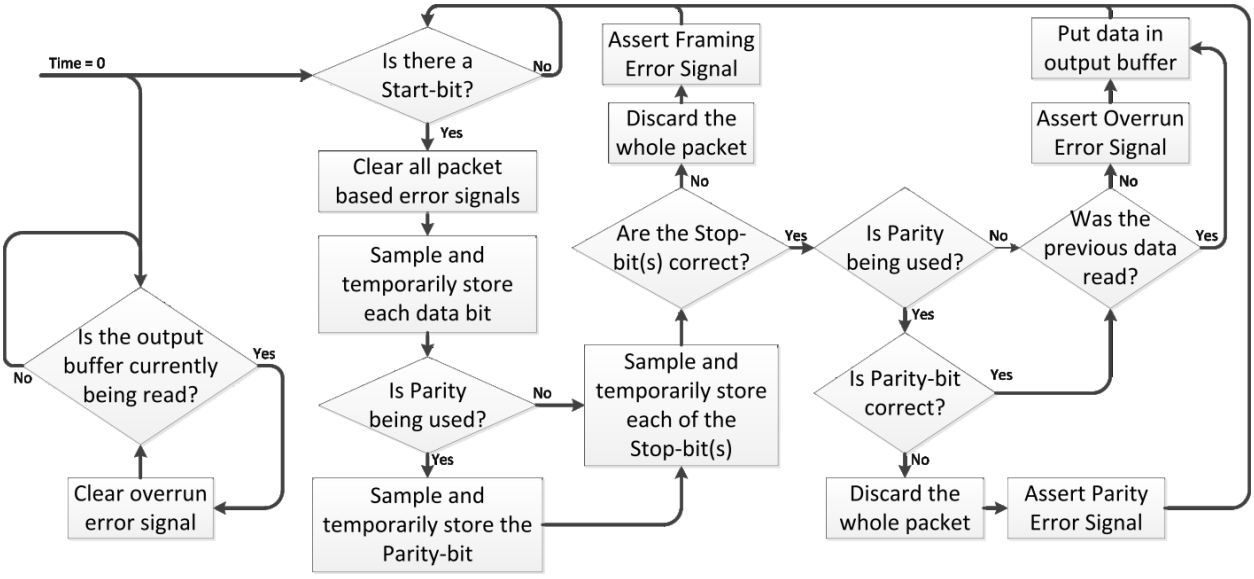

Figure 4: UART Receiver General Sequence of Operations

2. Design Architecture

2.1. UART Full Duplex Transceiver Architecture

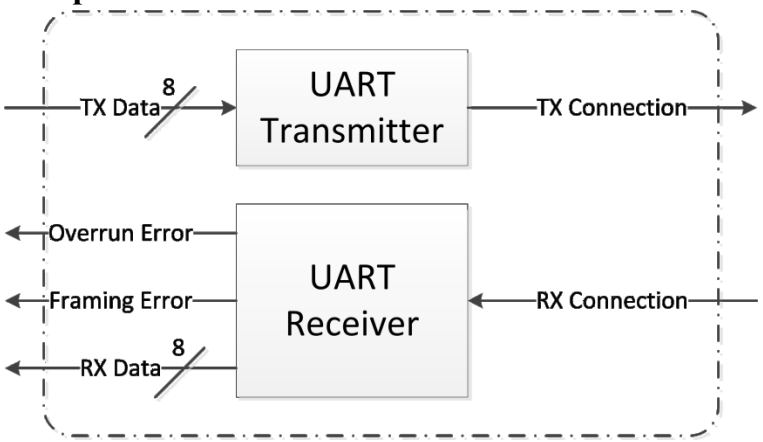

Figure 5: Full Duplex UART Transceiver Architecture Diagram

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

2.2. UART Receiver Block Architecture

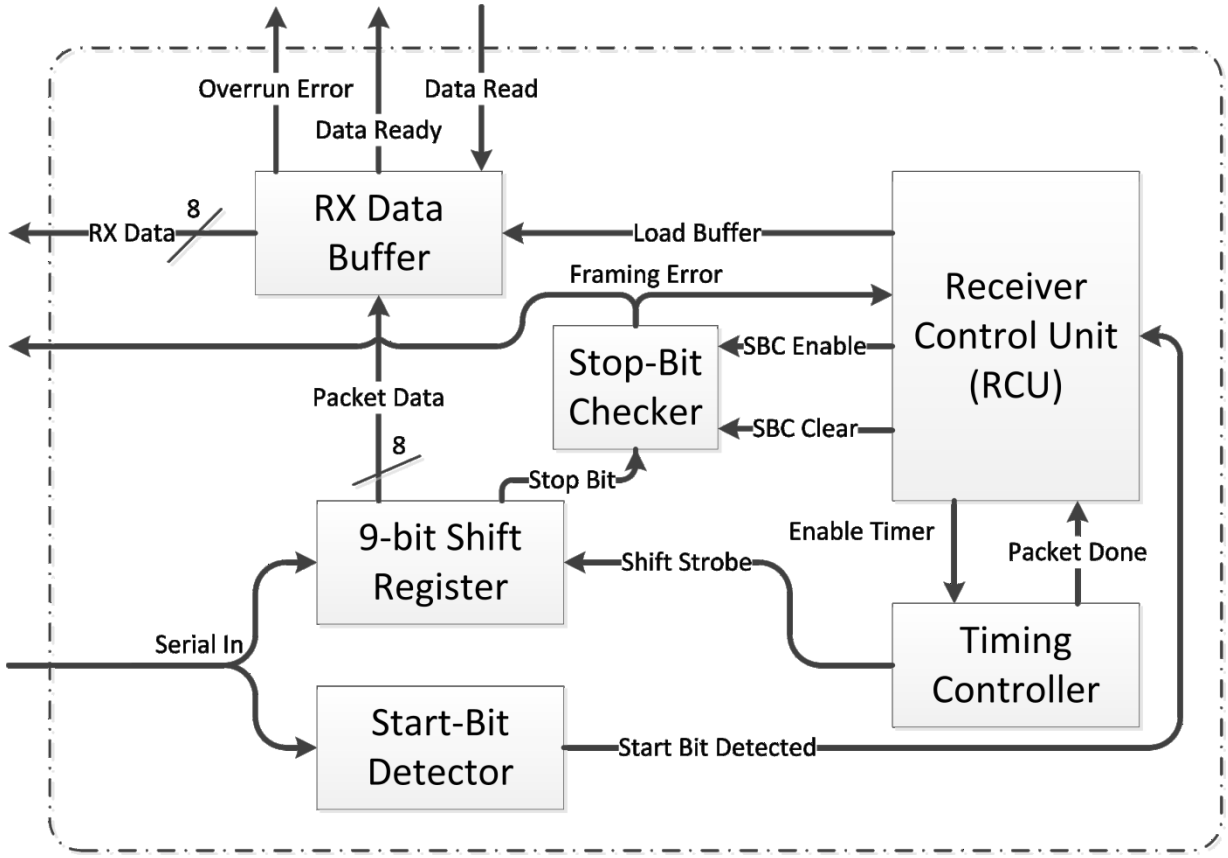

Figure 6: Receiver Block Architecture Diagram

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

2.2.1. List of the Top-Level Ports

- **Clk:** This is the system clock port. It should be connected to a 400 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
- **RX Data:** This is the 8-bit bus that holds the data byte currently available for reading from the output data buffer
- **Data Ready:** This is the signal that is used to tell an external device that data is available to be read
- **Data Read:** This is the signal that is asserted by an external device when it has read the available data
- **Overrun Error:** This is the signal that is asserted once an overrun error condition has occurred and it is cleared when an external device reads the currently available data
- **Framing Error:** This is the signal that is asserted once a framing error (invalid Stop-bit) has occurred for a packet and it is cleared once a new packet has started to be received

2.2.2. List of the Functional Units/Blocks

- **Receiver Control Unit (RCU):** This functional unit/block handles the decision logic regarding what to do with the packet that is being received
- **Timing Controller:** This functional unit/block handles the specific timing of when to sample the data from the serial input and signaling when a packet has been fully received
- **Output Data Buffer:** This functional unit/block stores received data for an external device to read and handles the hand shaking needed for evaluating and signaling an overrun error
- **Stop-Bit Checker:** This functional unit/block checks the received Stop-bit for correctness and signals if there is a framing error with the received packet
- **9-Bit Shift Register:** This functional unit/block shifts in/accumulates the needed 9 bits of the packet received over the serial input port
- **Start-Bit Detector:** This functional unit/block detects the falling edge that occurs between a Stop-bit and Start-bit

3. Block Specifications for Blocks You Must Design

3.1. Receiver Block

3.1.1. Block Description

The receiver block basically functions to serially receive data and store it in parallel into a register. This register will then be read by an external device. When idle, the serial input will always have the value of a logic ‘1’. The receiver knows to start receiving a packet when a start bit (logic ‘0’) has been detected. The start bit can and will arrive at any point in the system clock period. After the start bit, the following serial data should be sampled near the center of each bit. This is to minimize false data sampling due to the transmitter’s frequency variations. The serial data is transmitted LSB to MSB. The serial data is then followed by a Stop-bit (logic ‘1’). Also, all errors must be reported by the design using the designated flag signals by the end of the first idle bit following a

packet and their flag should remain asserted until the corresponding error condition has been removed (i.e. a new packet arriving removes the Stop-bit error condition).

Also, the serial_in signal is a completely asynchronous signal and must be synchronized to the clock edge. However, it is trivial to extend edge detection logic to also synchronize (which is already done in the provided start-bit detector block). Also, since the shift register should only ever shift in a value based on the timing from the timing controller, which is naturally synchronized to the clock as well as to the packet's timing, the shift register does not need have its serial_in directly synchronized to the clock. The other input signals well be coming from another device in the same system, mostly likely as part of a system on a chip (SOC) design and so will be already be synchronized to the clock.

3.1.2. Requirements

The required entity name is: rcv_block
The required filename is: rcv_block.sv

3.1.3. Port Descriptions

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
serial_in	Input	This is the serial input signal and thus will contain the data that is being serially transmitted to the UART. If a packet is not being transmitted, this line will be 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 by the external device.
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.

3.2. Timing Controller

3.2.1. Block Description

When the Timing Controller block is enabled, via the enable_timer signal, it will keep track of the timing of the serila_in data. It is worthwhile to note that this behavior can be done equally as well while using the enable_timer signal as one-time enable or continuous enable for the packet. When a data bit needs to be captured by the shift register, it will signal the 9-bit shift register to do so using the shift_strobe (hint: generate shift_strobe for 1 clk cycle in the middle of the data bit). When all nine (9) useful bits (eight (8) data and one (1) stop) have been captured by the shift register, the timing controller uses the packet_done signal to inform the RCU that all the data being received has been captured. It should also be stated that the name of this block may be slightly misleading. This block may actually consist of multiple timers that are working in conjunction to generate the desired output signals.

Hint: Since the timer is effectively just tracking the number of clock cycles that have happened and the number of bits that have been received, you quickly and easily build this using your flexible counter design from lab 3.

3.2.2. Requirements

The required entity name is: timer
The required filename is: timer.sv

3.2.3. Port Descriptions

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
enable_timer	Input	This is the enable signal for the timer.
shift_strobe	Output	This is the active-high one (1) clock cycle pulse signal that commands the 9-bit shift register to shift in the value of the serial_in signal.
packet_done	Output	This is the flag signal to the RCU that reports when all the needed bits from the current packet has been captured.

3.3. Receiver Control Unit

3.3.1. Block Description

This block is the portion of the design that dictates the current mode of operation for the Receiver Block of the UART. The name of the this block may make it sound intimidating to design, but if one sits down and rationally thinks out how the control unit should operate, the design of this unit should be relatively straight forward.

3.3.2. Requirements

The required entity name is: rcu
The required filename is: rcu.sv

3.3.3. Port Descriptions

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
start_bit_detected	Input	This is the active-high one (1) clock cycle pulse signal that reports when the falling edge between a Stop-bit or Idle line value and a Start-bit has occurred.
packet_done	Input	This is the flag signal that reports that all of the needed bits from the current packet have been captured.
framing_error	Input	This is the active-high flag signal that is updated by the Stop-bit checker on the rising edge of the clock while the sbc_enable signal is asserted. It is asserted when there is a framing error and is cleared on the clock cycle after the sbc_clear signal is asserted.
sbc_clear	Output	This is the active-high one (1) clock cycle pulse signal that tells the Stop-bit checker to clear its current framing error flag.
sbc_enable	Output	This is the active-high one (1) clock cycle pulse signal that tells the Stop-bit checker to check the current Stop-bit for correctness.
load_buffer	Output	This is the active-high one (1) clock cycle pulse signal that tells the output data buffer to load the data from the received packet.
enable_timer	Output	This is the enable signal for the timer.

3.1. 9-bit Shift Register

3.1.1. Block Description

This is a 9-bit Shift Register that will shift the serial data received.

Hint: Since this is simply a special case of the flexible shift register you designed in Lab 3, this should simply be a wrapper file that uses the that design, which is why that design file has been copied into your source folder by the setup script.

3.1.2. Requirements

The required entity name is: sr_9bit
The required filename is: sr_9bit.sv

3.1.3. Port Descriptions

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
shift_strobe	Input	This is the active-high one (1) clock cycle pulse signal that commands the 9-bit shift register to shift in the value of the serial_in signal.
serial_in	Input	This is the serial input signal and thus will contain the data that is being serially transmitted to the UART. If a packet is not being transmitted, this line will be a logic '1'.
packet_data[7:0]	Output	This is the 8-bit data that was sent in the currently received packet.
stop_bit	Output	This is the value of the Stop-bit that was received in the packet.

4. Block Specifications for Provided Blocks

The following are descriptions of the given components. You may modify or rewrite any of these components as you see fit as long as you keep the filenames same.

4.1. RX Data Buffer

4.1.1. Block Description

As the name implies, this block is a buffer that is utilized to store the 8-bit data that has been received by the UART and supply it for reading to an external device. It also checks if an overrun error has occurred each time it is loaded with new data. If an overrun error has occurred it reports the error using its error signal and only clears the error signal when the current contents of the buffer are read.

4.1.2. Requirements

The required entity name is: rx_data_buff
The required filename is: rx_data_buff.sv

4.1.3. Port Descriptions

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
load_buffer	Input	This is the active-high one (1) clock cycle pulse signal that tells the output data buffer to load the data from the received packet.
packet_data [7:0]	Input	This is the 8-bit data that was sent in the currently received packet.
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 by the external device.
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.

4.2. Start-Bit Detector

4.2.1. Block Description

The Start-Bit Detector is used to determine when the Start-Bit has occurred by checking for the falling edge that will occur between a Stop-bit or the Idle line value and a Start-bit. Since it is simply looking for a falling edge, it will assert the start_bit_detected signal whenever a falling edge occurs on the serial_in signal.

4.2.2. Requirements

The required entity name is: start_bit_det
The required filename is: start_bit_det.sv

4.2.3. Port Descriptions

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
serial_in	Input	This is the serial input signal and thus will contain the data that is being serially transmitted to the UART. If a packet is not being transmitted, this line will be a logic '1'.
start_bit_detected	Output	This is the active-high one (1) clock cycle pulse signal that reports when the falling edge between a Stop-bit or Idle line value and a Start-bit has occurred.

4.3. Stop-Bit Checker

4.3.1. Block Description

The purpose of the Stop-Bit Checker block is to determine if a proper Stop-bit has come across the connection. The Stop-bit concludes the transfer of data.

4.3.2. Requirements

The required entity name is: stop_bit_chk
The required filename is: stop_bit_chk.sv

4.3.3. Port Description

Signal	Direction	Description
clk	Input	The system clock. **(400 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 values.
sbc_clear	Input	This is the active-high one (1) clock cycle pulse signal that tells the Stop-bit checker to clear its current framing error flag.
sbc_enable	Input	This is the active-high one (1) clock cycle pulse signal that tells the Stop-bit checker to check the current Stop-bit for correctness.
stop_bit	Input	This is the value of the Stop-bit that was received in the packet.
framing_error	Output	This is the active-high flag signal that is updated by the Stop-bit checker on the rising edge of the clock while the sbc_enable signal is asserted. It is asserted when there is a framing error and is cleared on the clock cycle after the sbc_clear signal is asserted.