THE UNIVERSITY of WISCONSIN MADISON

Mini-Project 1 Overview

done as Team of 2 or 3

Mini Project 1

- Design a Special Purpose Asynchronous Receiver/Transmitter (SPART) and its testbench in Verilog
- Simulate with dedicated testbench to ensure correct behavior
- Memory map as peripheral to your 552 proc
- Demonstrate correction functionality on FPGA
- Prepare a report on your design

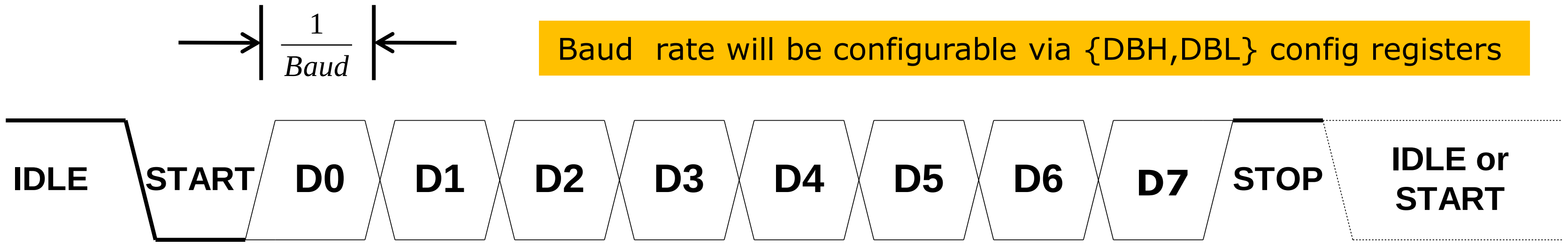
Miniproject 1 Objectives

- To get familiar with the lab environment prior to the class project
- To get practice using verilog in your designs
- To provide the basic I/O interface which might be useful for your project
- To get experience working with partners

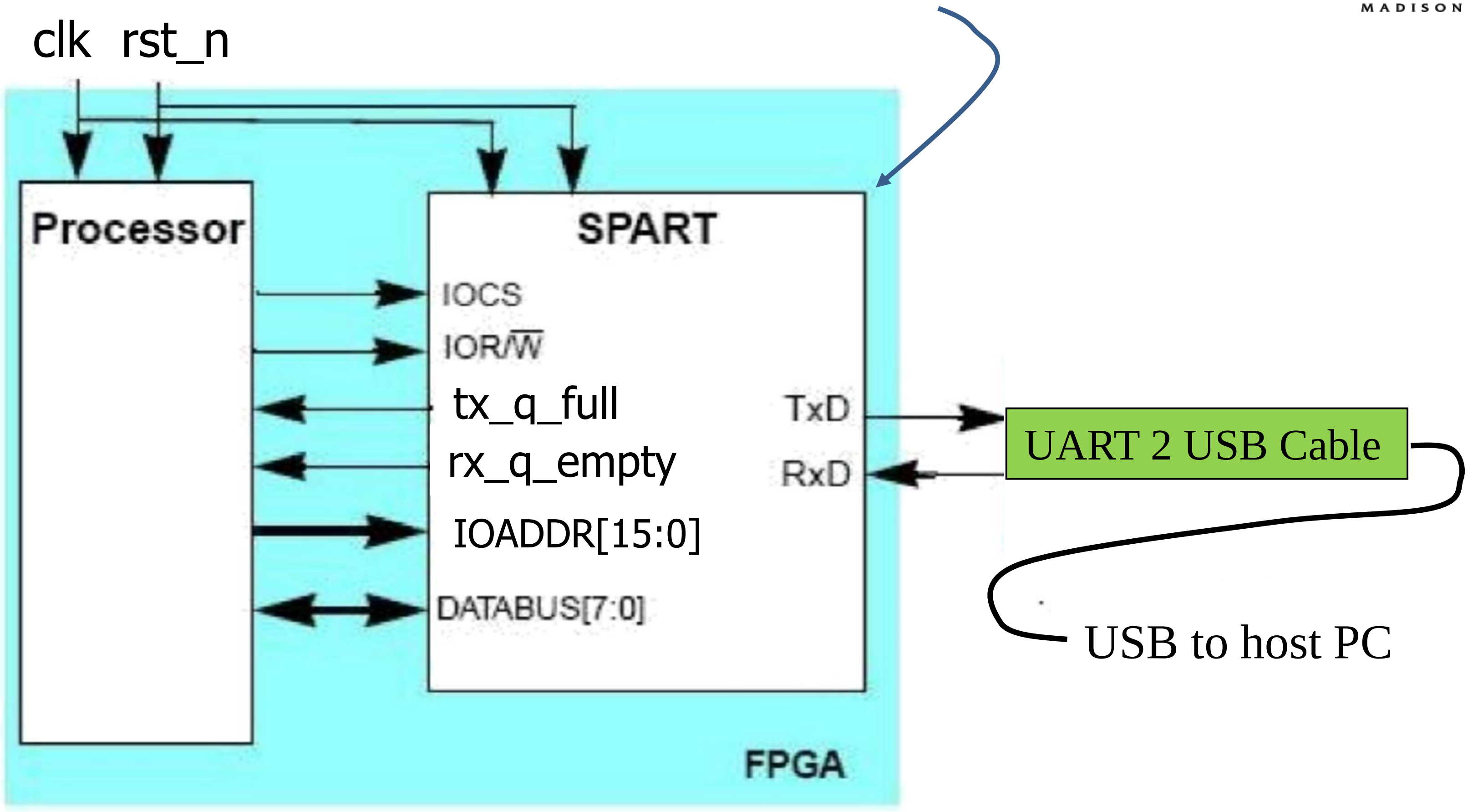
What is UART (Universal Asynch Receiver Transmitter)

- UART signal phases
  - Idle
  - Start bit
  - Data (8-data for our project)
  - Parity (no parity for our project)
  - Stop bit – channel returns to idle condition
  - Idle or Start next frame

Baud rate will be configurable via {DBH,DBL} config registers



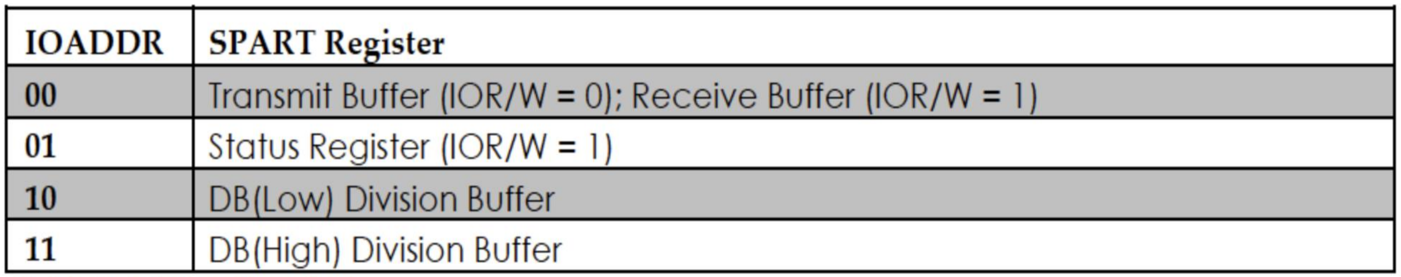
SPART (*SPART.sv*)



Download shell **spart.sv** as starting point.

SPART Interface

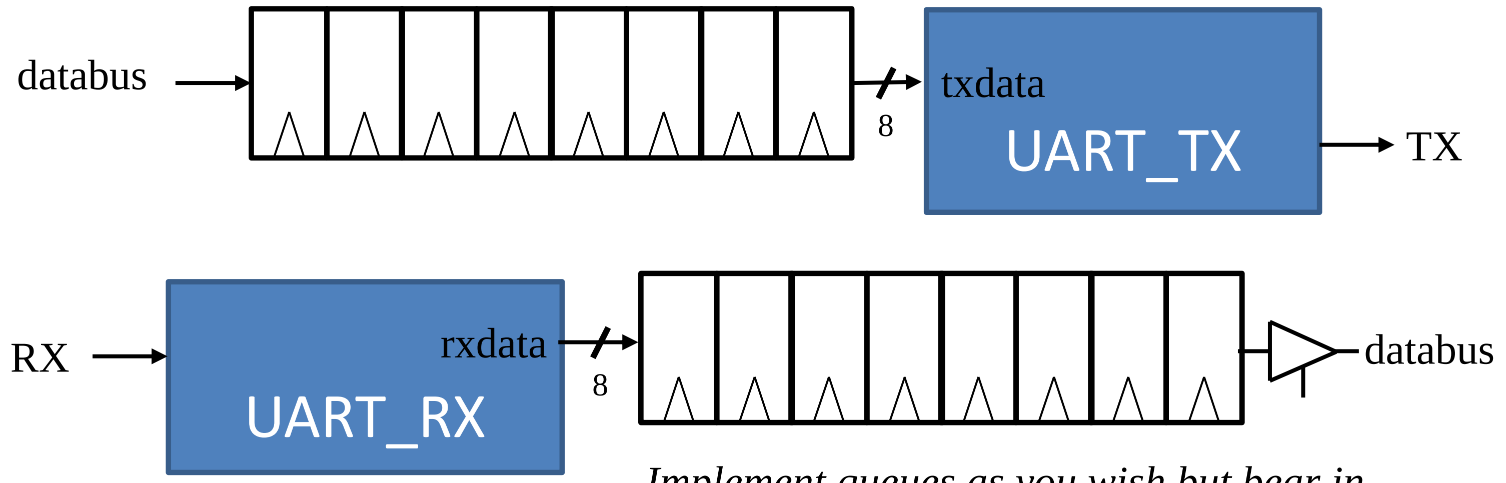
Signal:	Description:
databus[7:0]	An 8-bit, 3-state bidirectional bus used to transfer data and control information between the Processor and the SPART.
ioaddr[1:0]	A 2-bit address bus used to select the particular register that interacts with the **databus** during an I/O operation.
iorw_n	Determines the direction of data transfer between the Processor and SPART. For a **r**ead (iorw_n=1), data is transferred from the SPART to the Processor and for a **w**rite (iorw_n=0), data is transferred from the processor to the SPART.
iocs_n	Active low **ch**ip **s**elect. Writes or reads to registers have no effect unless active
rx_q_empty	If 1 then no receive data present to read.
tx_q_full	If 1 then transmit queue is full and cannot accept anymore bytes.



IOADDR	SPART Register
00	Transmit Buffer (IOR/W = 0); Receive Buffer (IOR/W = 1)
01	Status Register (IOR/W = 1)
10	DB(Low) Division Buffer
11	DB(High) Division Buffer

Table 1: Address Mappings

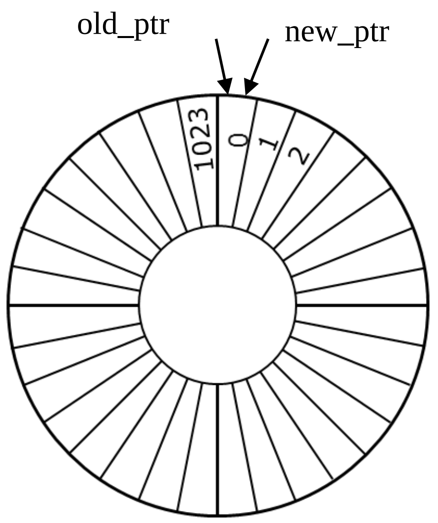
SPART Function



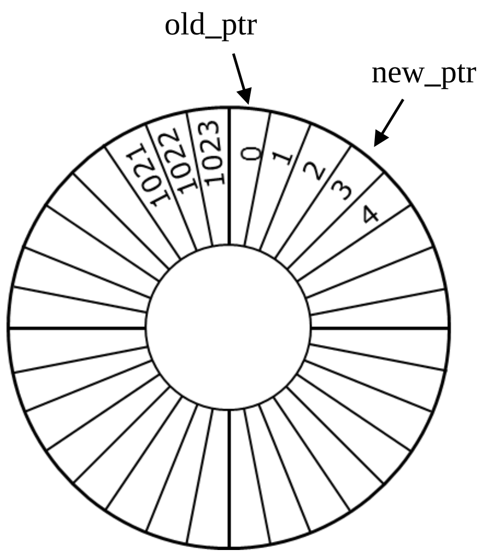
Implement queues as you wish but bear in mind real engineers use circular queues.

- It is likely someone on your team has basic UART transmitter and receiver blocks from ECE551. Use these as a foundation for your SPART.
- You have to add an 8 deep queue to TX so 8 *(actually 9)* consecutive writes can be buffered. You also add an 8-deep queue to the output of the receiver so multiple bytes could be received before the consumer has to read the buffer.
- In addition to these changes the transceiver needs to support configurable baud rates.

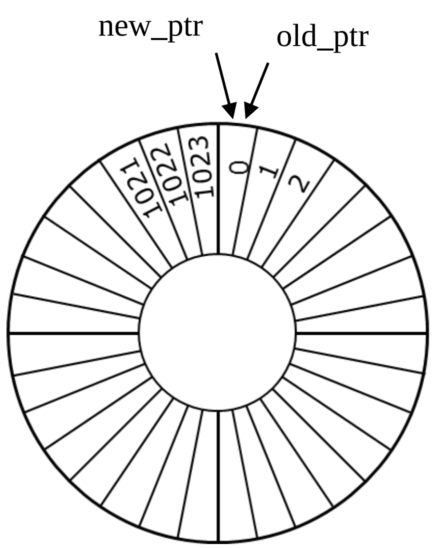
Use Circular Queues for Buffers



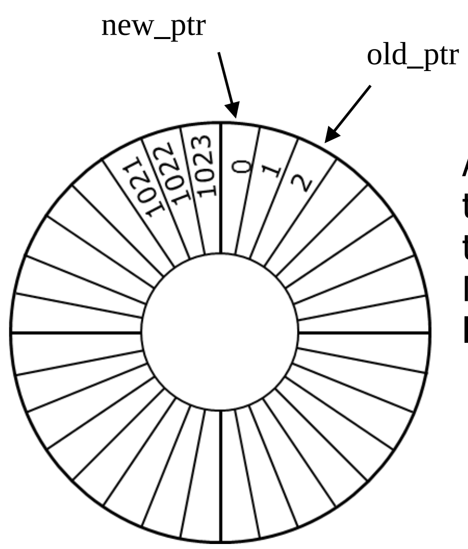
new_ptr points at next available location. ***old_ptr*** points at oldest data location. Circular queue starts like this.



As we write new entries into the queue **new_ptr** increments.



After having written 1024 locations the queue would look like this be full (**new_ptr==old_ptr**) *(wait… isn't that the condition for empty?)* This is why we would maintain 11-bit pointers not 10-bit pointers



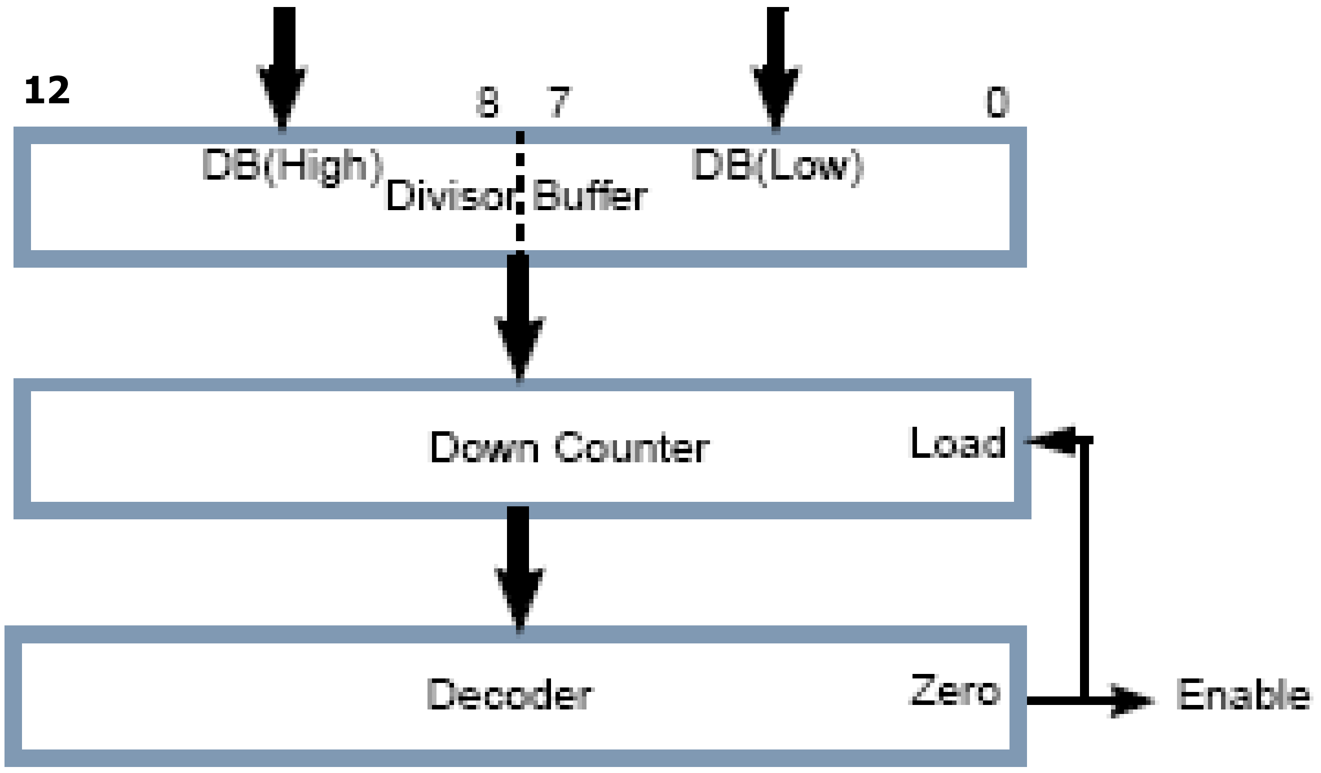
As the consumer of the queue data reads the **old_ptr** advances. Now the queue is no longer full.

SPART baudrate

Baud Division Buffer[12:0] {DBH[4:0],DBL[7:0]}	**Resulting baud rate** ***(assuming 50MHz clk)***
0x0036	921600
0x006C	460800
0x00D9	230400
0x01B2	115200* *reset to this baud rate*
0x0364	57600
0x0516	38400
0x0A2C	19200
0x1458	9600

- The DBH & DBL config registers are used in conjunction to set the configurable baud rate. The table above shows the common values to be used. The DBH register only needs 5-bit implemented to form a 13-bit division buffer.

Baud Rate Generator



- To implement the configurable baud rate it is recommended you modify the baud rate generator in both UART_TX and UART_RX to be down counters, and shift when they hit zero.

SPART Status Register

- The signals **tx_q_full** & **rx_queue_empty** can be used as handshaking signals to a hardware based driver of the SPART to know if there is room to transmit or if there is data to be read.
- A firmware based driver of SPART *(think a uController (like perhaps a version of your 552 processor))* can read the status register to obtain more detailed queue status.
- A read from **ioaddr**=2'b01 will return the status register. The status register is to be encoded as follows:

Bits [7:4]	Bits [3:0]
Number of entries remaining *(empty/available)* in tx queue	Number of entries filled *(that contain data)* in rx queue

SPART Status Register (addr=1'b01)

Testbench *(spart_tb.sv)*

- Create a testbench to thoroughly test the SPART.
  - Should test filling the TX queue to full.
  - Should test RX queue to near full.
  - Should only reset the DUT once at the beginning of the test sequence
  - Should test interleaved writes and reads at different rates
  - Should test configurable baud rates (at least a few)
  - Should test status reg contents
  - Should be self checking
  - Should be commented as to what it is testing.

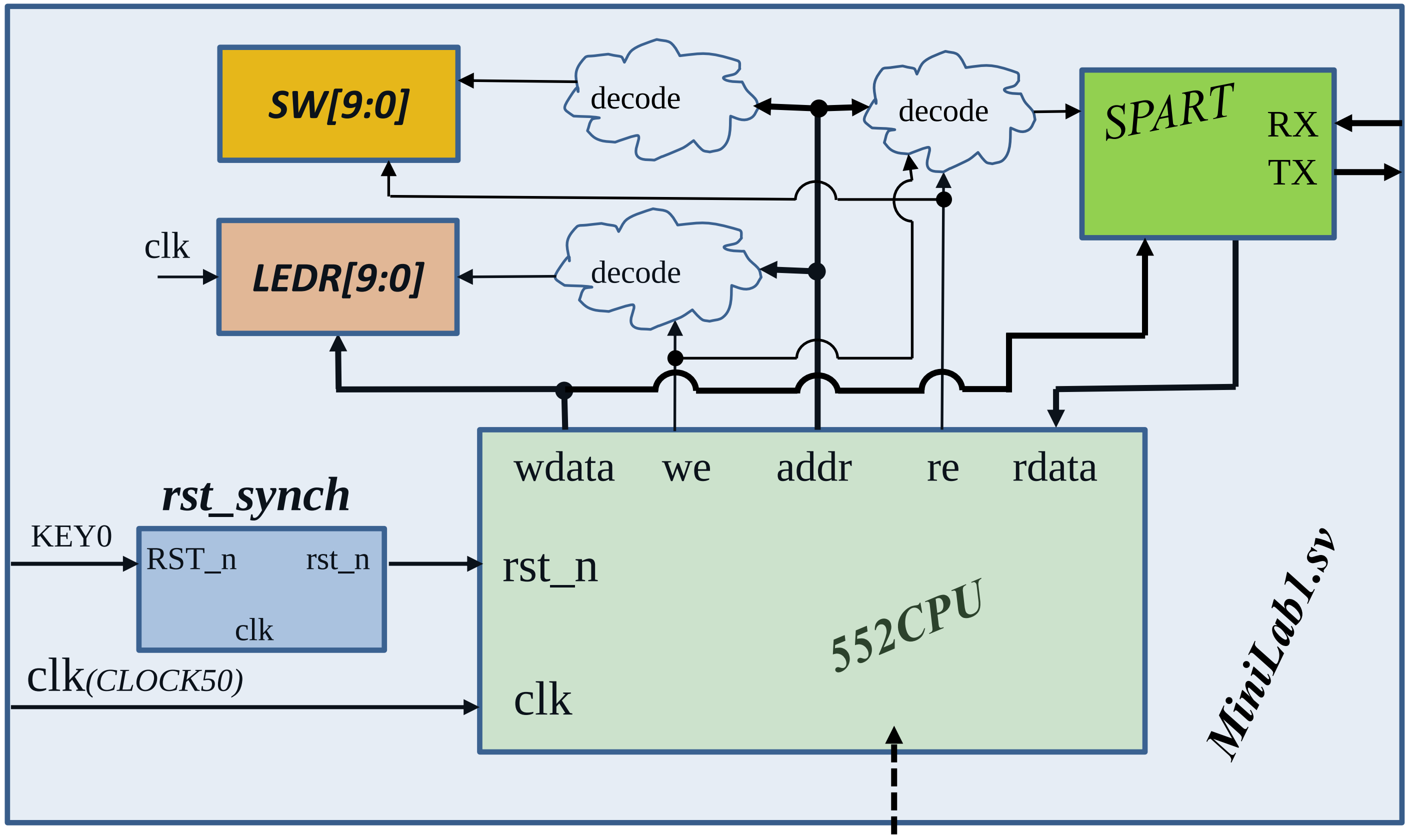
Memory Map to 552 Proc

- Memory Map to following addresses:

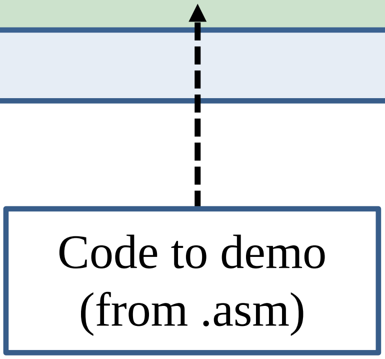
Address:	Description:
0xC004	TX/RX buffer. Read is a read from RX circular queue, write is write to TX circular queue.
0xC005	Status Register (read only)
0xC006	DB (low) Baud rate division buffer low byte
0xC007	DB (high) Baud rate division buffer high byte

- Test with code running on your 552 proc
  - Should clear the screen *(escape sequence <ESC>[2J) (VT100 escape codes)*
  - Should be print "Hello World" centered in the terminal
  - Should prompt user for their name and store response after receiving a <CR>
  - Should print "Hello <name>" to terminal

Toplevel *MiniLab1.sv*



Code should print "Hello World!\n" to a terminal emulator. Then should prompt the user for their name and print "Hello <name>" after <CR> detected.



Warning!

- Does a read from memory change the state of the machine?
- It can now! A read from the SPART queue removes an entry.
  - That is a change of state!

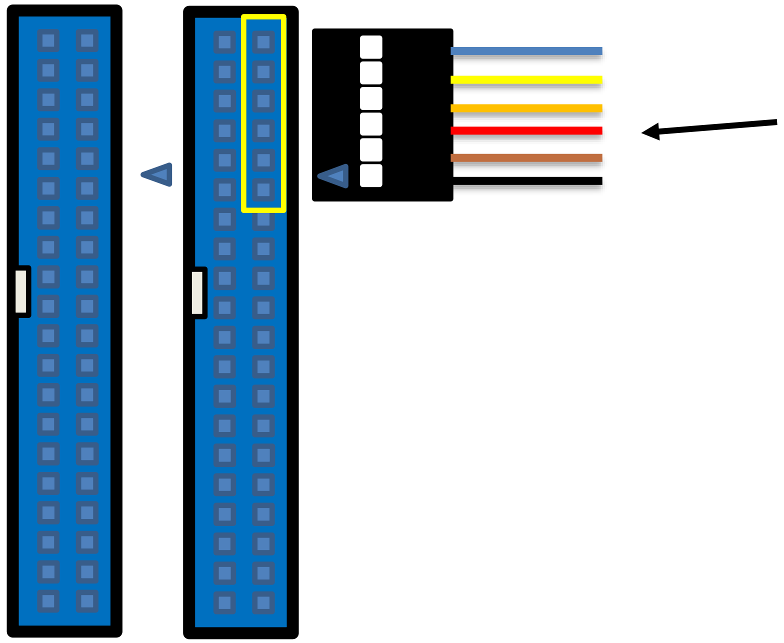
- On a flush does your processor inhibit the RE signal to memory? Mine did not. You need to ensure it does!

- I spent many many hours chasing down a bug related to this…you're welcome *(take this warning seriously)*

- Also…make sure your **C**hip **S**elect signal for the SPART is well qualified, looking at not just the proper address range, but also that either RE or WE is asserted.

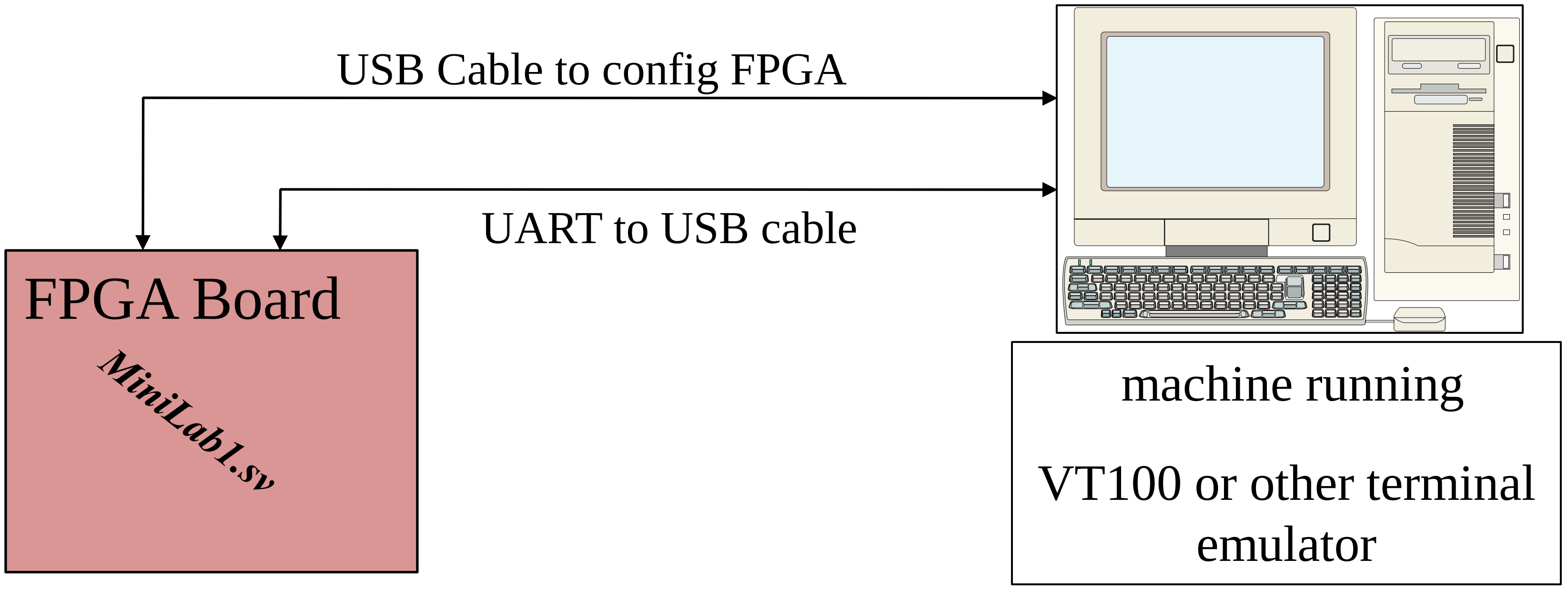
Lab Setup

- Download your design into the FPGA and following the picture to connect the USB-UART cable.



NOTE: The red wire is intentionally pulled from the socket. Having it connected would damage the board. It is 5V from USB and the I/O of the board is only tolerant to 3.3V.

Lab Setup



For full credit your firmware should make use of terminal escape sequences to locate to the center of the terminal emulator screen to print "Hello World!\n"

Miniproject-1 Report

- Verilog code for your design with clear comments. This includes **spart.sv**, **spart_tb.sv, MiniLab1.sv** and the firmware used to drive your 552 processor.
- Record of experiments conducted and how the design was tested
- Problems encountered and solutions employed