Serial Communication

- Serial communication is as widely (or even more widely used) than parallel communication
  - Especially true if communication occurs over long wires
- Many new high speed serial communication standards have been developed
  - USB, IEEE Firewire, HyperTransport, etc.
- This lab will introduce you to some basic serial communication concepts, namely *bit-stuffing* and *NRZI encoding*
  - These techniques are used in the USB (Universal Serial Bus) interface.

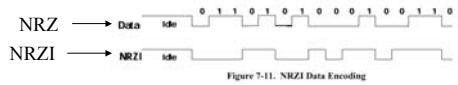


Figure 7-11. NRZI Data Encoding

Non-return to zero (NRZ) - normal data transitions.

NRZ Inverted (NRZI, not a good description, is not inverse of NRZ). A transition for every zero bit.

Strings of zeros means lots of transitions. Strings of '1's means steady line.

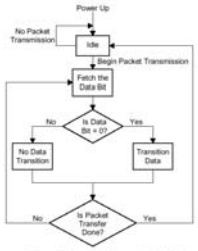


Figure 7-12. Flow Diagram for NRZI

Bit Stuffing – a '0' is inserted after every six consecutive '1's in order to ensure a signal transition so that receiver clock can remain synchronized to the bit stream.

Data Encoding Sequence:

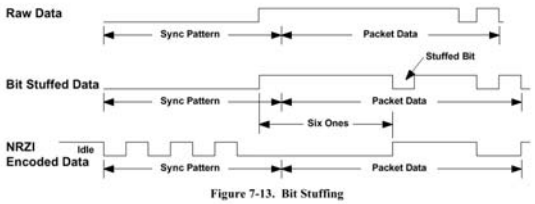
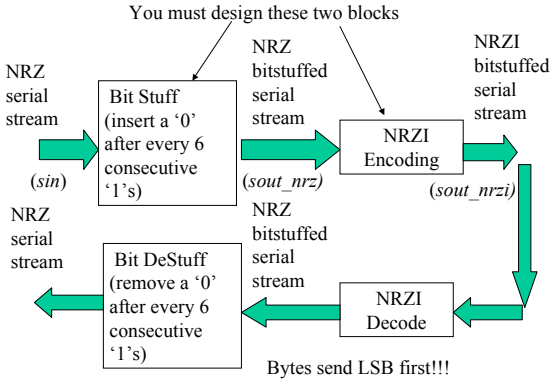


Figure 7-13. Bit Stuffing

Bit stuffing done automatically by sending logic. Sync pattern starts data transmission and is seven '0's followed by a '1'.



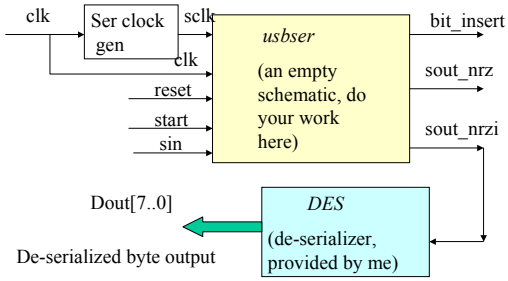
Bytes send LSB first!!!

The Task

- Design a block that performs bitstuffing of a '0' after every six consecutive '1's from an NRZ serial stream and does NRZI encoding of the output
- Inputs
  - *reset* – synchronous reset, high true
  - *clk* – clock signal
  - *serclk* – clock signal for serial stream (clk divided by 4, one pulse for every four clks)
  - *sin* - NRZ serial input stream
  - *start* – will be high for one clock cycle indicating start of valid data on serial NRZ stream. Serial bit is valid every time 'serclk' = '1'.
- Outputs
  - *Sout_nrzi* -- bit stuffed stream, NRZI encoding
  - *Sout_nrz* – bit stuffed stream, NRZ encoding (use this for debugging)
  - *Bit_insert* – assert high whenever a '0' bit is inserted (use this for debugging).

Testbench

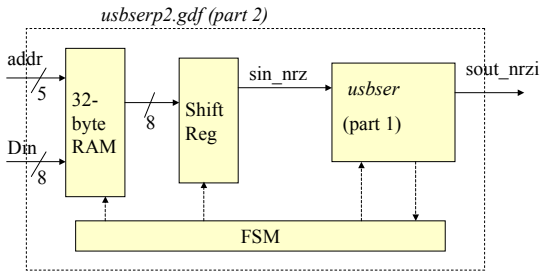
You are provided with a testbench called *tbusbser*



tbusbser_gold.scf

- *tbusbser_gold.scf* is the golden waveform
- Provides all input signals (start, reset, clk and most importantly, the serial input stream *sin*)
- The DES block is provided de-serializes the *sout_nrzi* stream into bytes
  - This byte stream shown as *dout[7..0]* on the golden waveform
- It might help to look at the structure of the DES block before attempting your design, it might give you some ideas.

Part 2: Read 32 Bytes from Ram and send over serial interface



Bytes send LSB first!!!

usbserp2.gdf Interface

- Exactly the same as used for bifilt interface
- Inputs
  - *Clk, reset* – clock and asynchronous reset
  - *we, addr[4..0], din[7..0]* – write enable, address, data for RAM
  - *start* – when start asserted, read 32 bytes from RAM from send over serial interface (*usbser* done in part 1)
- Outputs
  - busy – asserted when sending reading ram and sending serial data
  - sout_nrzi – serial output (bit stuffed, NRZI) from *usbser* block

usbserp2.gdf datapath, FSM

- RAM, counter for address lines, shift register already in *usbserp2.gdf* schematic
  - Clock generator for *serclk* included also
- May have to add other datapath elements as well (perhaps a 3 bit counter for determining when 8 bits are shifted out), maybe other components as well
- Have to add a FSM that will control reading of RAM and shift register
  - Will also have to monitor *bit_insert* output from *usbser* block – if inserting a bit into the serial data stream will have to halt shift operation

Part #2 Testbench (*tbusbserp2.gdf*)

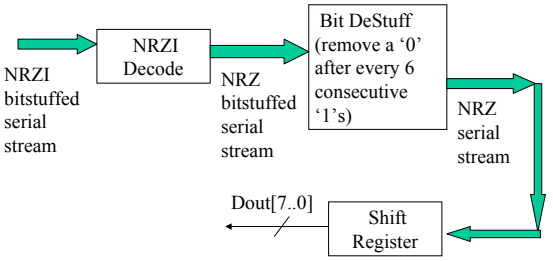
- *tbusbserp2.gdf* connects the *usbserp2* block to the *deserializer* block
- The output of the *deserializer* block will match the values read from RAM if the everything is working ok.
- Golden waveform is *tbusbserp2_gold.scf*

Due Dates

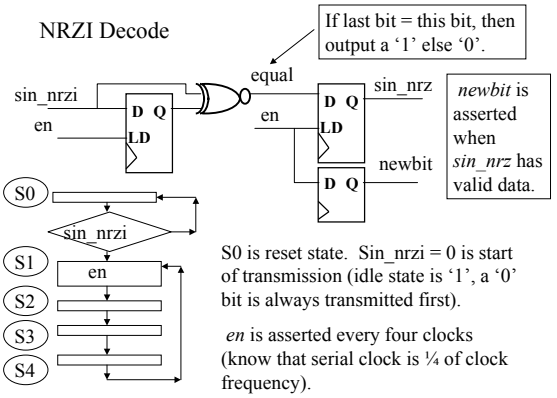
- One week for each part 1 and part 2.
- Each part is about the same difficulty
- Each part is worth 100 points.
- No extra credit for this lab.
- Only functional requirements, no clock frequency requirements.
- Can use any mixture of VHDL + schematic capture for the design.

DESerializer Operation

Understanding the DESerializer operation may help with implementation of the serializer.

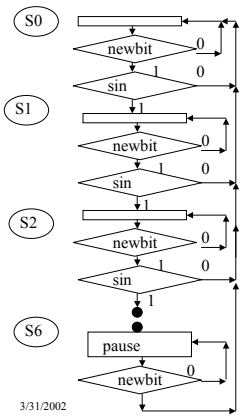


NRZI Decode



S0 is reset state. Sin_nrzi = 0 is start of transmission (idle state is '1', a '0' bit is always transmitted first).

en is asserted every four clocks (know that serial clock is ¼ of clock frequency).



Bit De-stuffing

pause asserted when six '1' bits detected. The pause signal used to halt shift register so that the '0' bit which was stuffed is not shifted into register.