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.

NRZ
NRZI

- 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.

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:

<table>
<thead>
<tr>
<th>Raw Data</th>
<th>Sync Pattern</th>
<th>Packet Data</th>
<th>Packet Data</th>
<th>Bit-Stuff Pattern</th>
<th>NRZ Encoded Data</th>
<th>Packet Data</th>
<th>Packet Data</th>
</tr>
</thead>
</table>

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

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

clk
Ser clock
sclk
reset
start
sin
usbusser
(bit_insert
(De-serializer, provided by me)

Dout[7..0]
De-serialized byte output

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

usbserp2.gdf (part 2)

Part #2 Testbench (usbserp2.gdf)

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

usbserp2.gdf (part 2)

Part #2 Testbench (usbserp2.gdf)

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

usbserp2.gdf (part 2)

Part #2 Testbench (usbserp2.gdf)

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

usbserp2.gdf (part 2)

Part #2 Testbench (usbserp2.gdf)
DESerializer Operation
Understanding the DESerializer operation may help with implementation of the serializer.

NRZI Decode

Bit DeStuff (remove a ‘0’ after every 6 consecutive ‘1’s)

Shift Register

NRZI decoded serial stream

Dout[7:0]

NRZ bitstuffed serial stream

NRZI serial stream

NRZ bitstuffed serial stream

NRZ serialized stream

NRZI Decode

If last bit = this bit, then output a ‘1’ else ‘0’.

newbit is asserted when sin_nrz has valid data.

S0
sin_nrzi
en

S1

S2

S3

S4

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).

newbit

D
Q

D
Q

D
Q

D
Q

D
Q

D
Q

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.