ECE 482 Final Project

Layout for a 45nm Serializer and Deserializer Pair

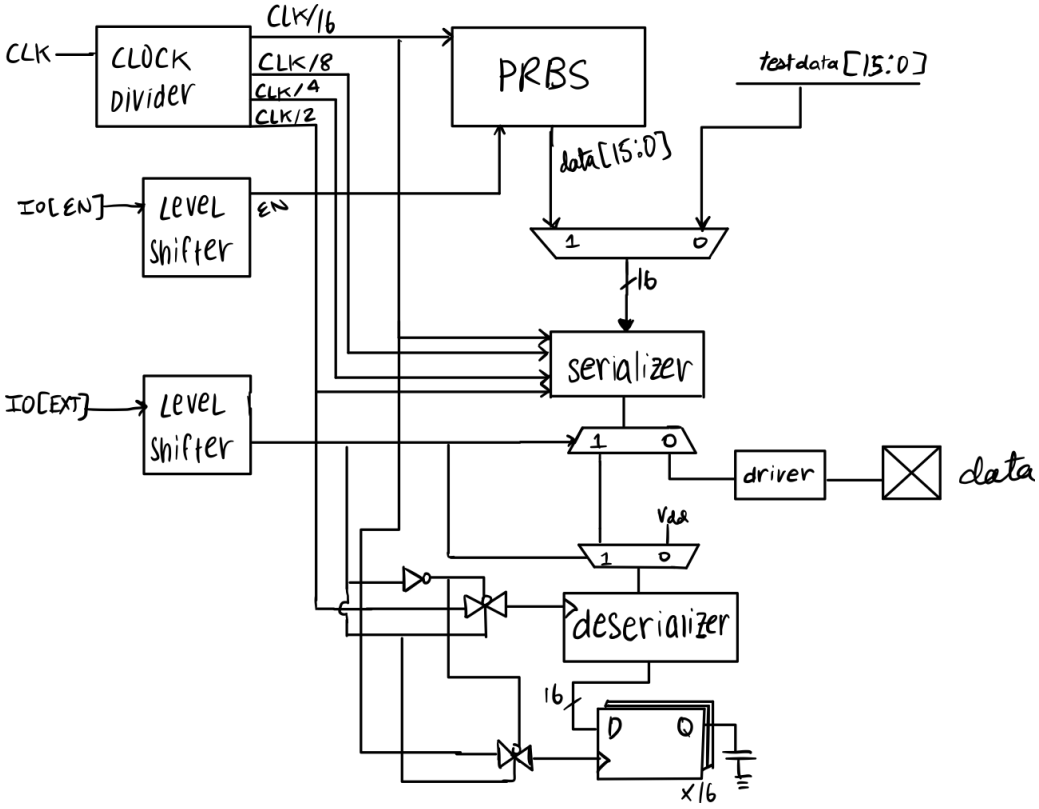
Derek Chaw, Qingyuan Liu, Kevin Shi, Pradyun Narkadamilli

December 7, 2023

Contents

1 General Design

Below is depicted the expected top-level diagram for our serializer/deserializer layout. Throughout this section, we will slowly decompose each of the components involved in this layout. Then in the next section, we will cover the transistor-level schematics and sizing necessary for the components in this design, such that we have transistor layouts available for all the general subcomponents..



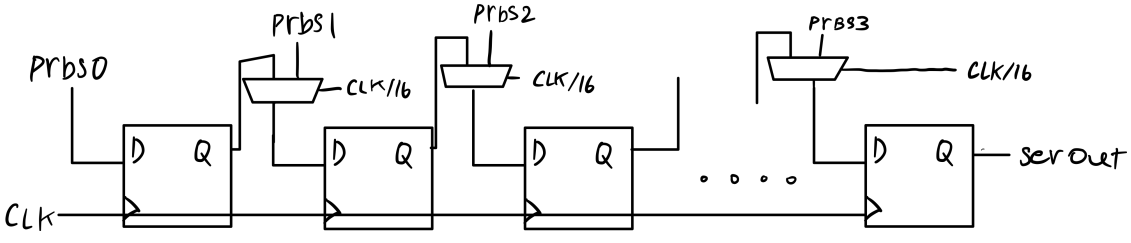
Block Diagram of Top Level Design

The inputs to our layout are the 1600MHz clock, denoted simply as CLK, and the 1.8V EXT/EN signals, denoted as IO[EN] and IO[EXT] respectively. As specified in the design document, the IO signals and level shifters operate on 1.8V, while the rest of the design operates on a 1.1V core voltage.

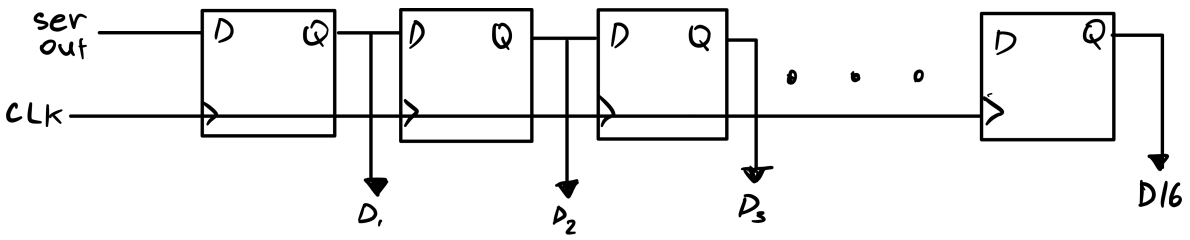
It should also be noted that for simplicity, the schmit trigger inverter buffers have been omitted - however, each of the top-level incoming data/control signals in our layout are passed through the provided buffer layout.

1.1 Serializer/Deserializer Pair

When initially approaching this design, we realized that from an architectural perspective, the naive implementation for a serializer/deserializer pair was exceedingly simple from an architectural perspective - we could simply use a shift register with parallel load capabilities for the Serializer, and another shift register (this time without parallel load) for the Deserializer. This would look something like the general schematic below.



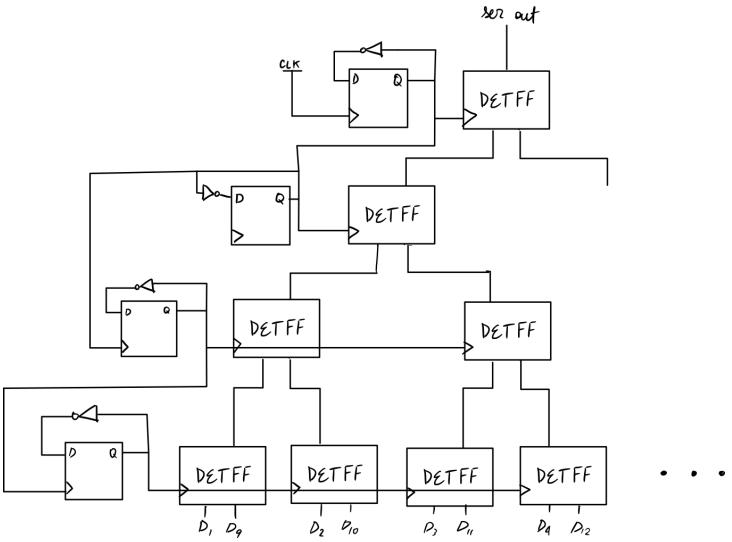
Block Diagram of Naive Serializer



Block Diagram of Naive Deserializer

However, we were concerned with the fact that there is very high switching activity on both ends of this design - every register is updated on every cycle of the serializer's output, as well as on every cycle of the deserializer's shift-in. As a result, we looked abroad to see if we could find any papers on modern Serializer/Deserializer implementations that minimized switching activity. In donig so, we found [this paper] (make sure to insert link here) about high-speed serializer and deserializer design. While the paper discusses benefits like more optimal clock distribution networks and better noise immunity, we were attracted to it due to the fact that it advertised low power consumptions while continuing to use C^2MOS registers - something that we are very familiar with. Furthermore, by using C^2MOS registers, we are able to minimize the effect of CLK-to- $\overline{CLK}$ skew.

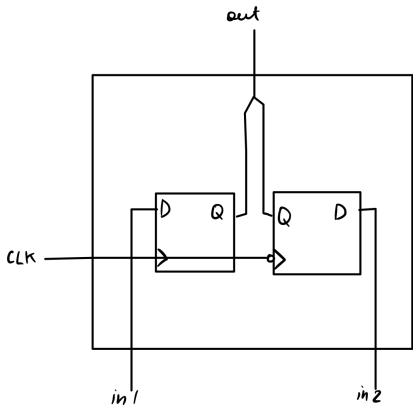
We start by discussing the serializer implementation. To summarize the paper, by creating a tree-like structure of dual-edged flip flops that are clocked by progressively slower/divided clocks. The high-level design for this can be seen below.



Block Diagram of our Serializer

Some odd symbols may be immediately noticed.The "..." is meant to infer the duplication of the bottom 3 rungs of the tree to the right side, and the DETFF symbol denotes a dual edged register. The general idea goes that for every four values that an upper level shifts out, each DETFF on the level directly below it will only change its values twice.

Let us continue traversing down the design hierarchy for our serializer. As seen earlier, this our dual edged registers are designed as follows.



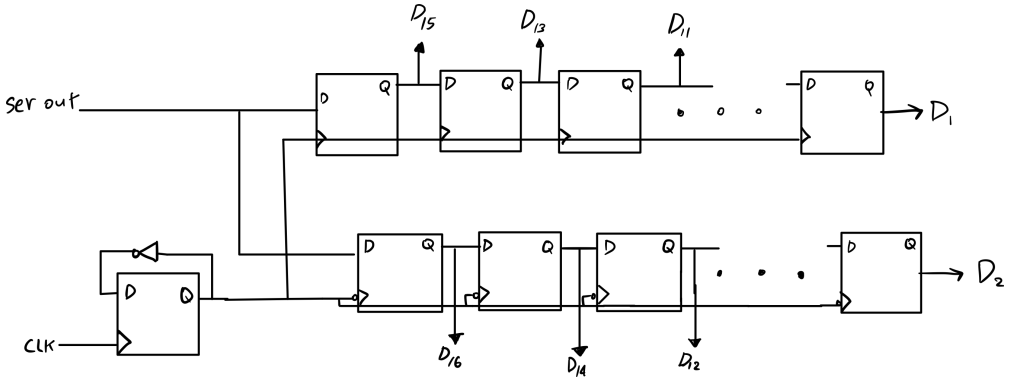
Block Diagram of Dual-Edged FF/Register

The dot on the clock terminal of the right-hand register indicates that the CLK and $\overline{CLK}$ terminals will be swapped on both stages of the register, making it negative edge triggered. Note that the registers are bridged in the dual edge FF schematic - this is no mistake. Since the C^{2}MOS register only strongly drives the output on the *negative* pulse of the clock, only one register will be driving this line at any given moment. This may become more obvious after seeing the register schematic in section 2.

Looking at the register schematic, something that should be addressed with this design is the additional area overhead - our tree structure requires 15 dual-edged registesr, i.e 30 C^{2}MOS registers, whereas the naive implementation would only need *16* registers. However, whereas all 16 registers would be updated on every cycle in the native implementation, in this case you update an average of 6 registers per clock cycle throughout the rest of the serializer operation. The power savings are so drastic that we considered the tradeoff worth it.

The dual edged register, which will be referred to as a DETFF going forwards, uses standard sized C^{2}MOS registers consistent with section 2. It should be noted that the paper suggests using a latch in between the negative-edged register and the output multiplexer - however preliminary testing in our design indicated that this was introducing issues in our simulations, and that the DETFF did not encounter any significant issues when operated without the latch. As a result, we have removed that latch altogether.

Since our serializer operates on a dual-edged output signal, we must create a counterpart deserializer capable of receiving data on both edges of a slower 800 MHz clock. According to the earlier mentioned paper, this slower clock distributed within the deserializer helps reduce power consumption in the clock distribution network - however since we did not have time to lay out both of the designs we proposed, we unfortunately cannot provide comparative results at this time.



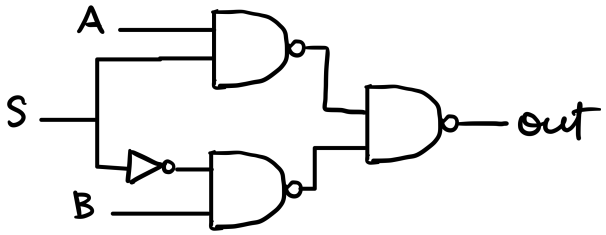
Block Diagram of our Deserializer

The D registers shown in this diagram are once again standard C^{2}MOS registers - the top line of registers is positive edged, whereas the bottom line of registers is driven on the positive edge of $\frac{\overline{\text{CLK}}}{2}$, making it effectively negative edged. We also observe a standard clock divider in the bottom left, such that the dual edge operating clock of the deserializer is matched with the operating clock of the serializer.

It should be noted that there were (comparatively) no novel architectural decisions taken with regards to the serializer - it has the exact same area as our naive serializer (barring the clock divider), and has simply been adapted to sync well with our dual edged operation.

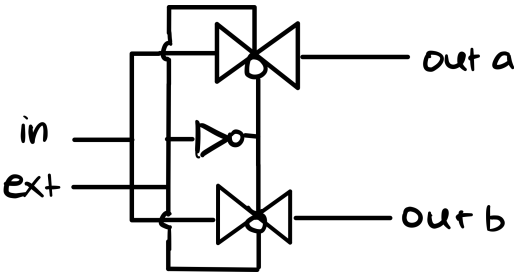
1.2 Multiplexer and Demux

Note that the linkage between the serializer and deserializer requires a 2-to-1 multiplexer and a 1-to-2 demux (to route the serializer output). The logical diagrams for these components are listed below. These constructs, like the rest of the design, uses the default gate parameters specified in section 2.



Logical Diagram of 2-to-1 Mux

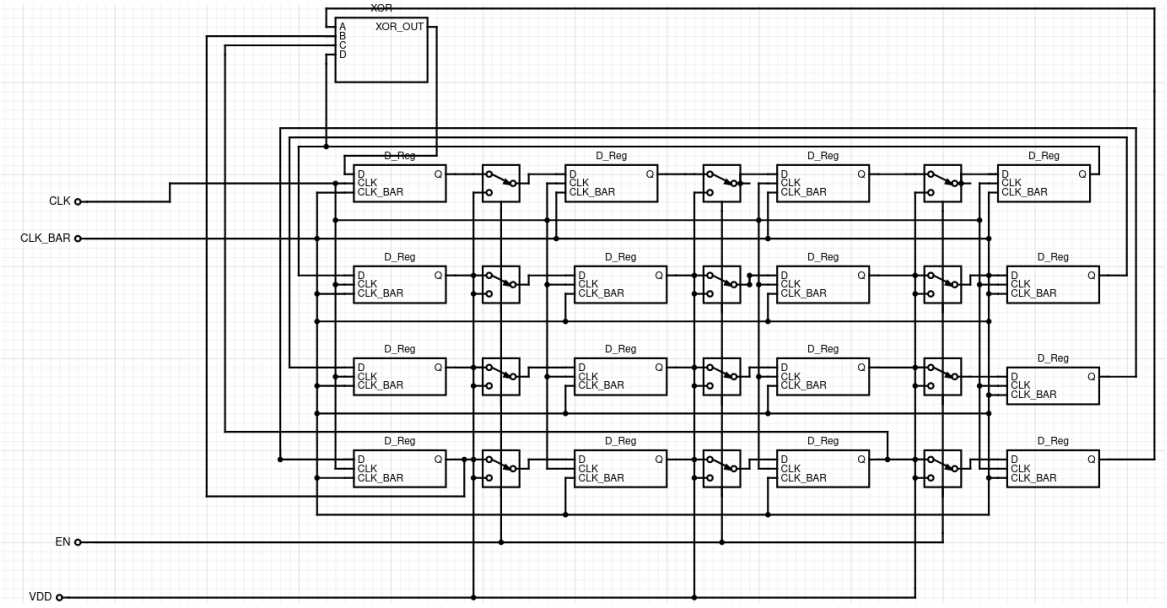
For the demux implementation, we chose to opt for a transmission gate based approach to minimize propagation delay - however since this can potentially drive a high-impedance signal, we have piped the final signal back into a multiplexer to strengthen it. In hindsight, our top level design could have potentially been modified to be two chained inverters instead to increase the drive strength.



Logical Diagram of 1-to-2 Demux

1.3 PRBS Design

Architectural design for this part of our circuit was relatively simplistic, since we were given the schematic from Digi-Key. A schematic of our PRBS architecture has been shown for clarity.



Block Diagram of PRBS

Do note that in the image about, the box-like symbol with 3 circles is a 2-to-1 multiplexer. Unfortunately the tool used to construct this schematic did not have the traditional trapezoid symbol.

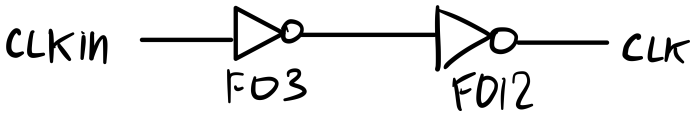
The biggest addition to our PRBS was reset logic - in order to fit the spec of the PRBS starting with all 1s, we position muxes before each register with a shifted down version of EN used as input. When disabled, the PRBS constantly loads 1s, and when enabled the PRBS will operate like normal. Ideally we would havel iked to put the reset in our register, but we had initially forgotten to account for the fact that the register does not naturally reset to high - by this point, adding resets into our registers directly would have completely changed our layout topology. While the mux-based resets are inoptimal in terms of area, they are functional, which is all we can ask for.

In our PRBS, we continue to use C^2MOS registers since they allowed for fast prototyping/layout, without having to worry too much about clock skew between the inverted and no-inverted clock signals.The main decision we made in the design of our PRBS was was what kind of XOR topology to use. We noticed that the output of each XOR gate was relatively low-fanout and did not have a very large load to drive. This seemed like an ideal usecase for a transmission-gate based XOR gate - it helps us reduce our area and power consumption in the PRBS without any real downside in our application. More on this schematic in section 2!

1.4 Top Level Integration

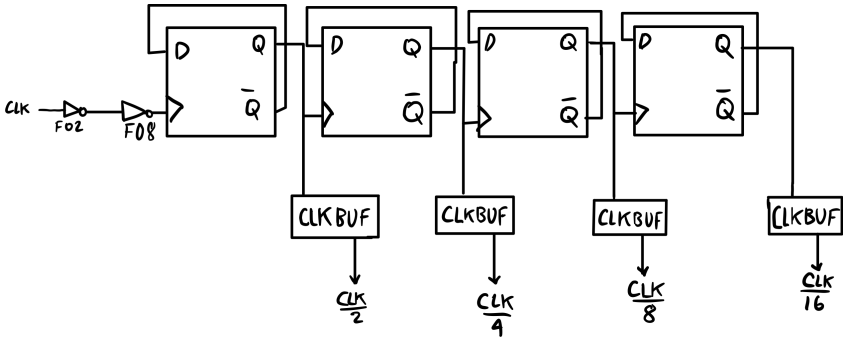
Until now, we have discussed most of the complex blocks in our system that required architectural forethought - here we detail some of the smaller components necessary to integrate all of these components together in the final layout.

For starters, driving CLK to every device in our layout was a very real issue. It requires a large driver to be able to reliably drive the high fanout nets in our system. To increase the drive strength of each divded clock signal, we use an even-N superbuffer, consisting of two inverters sized 3 and 12. This is depicted below.



General Clock Buffer

Additionally, to make sure that sub-1600MHz clocks are available as necessary across the layout, we created a unified clock divider with large clock buffers to drive the output. This way, we can avoid requiring 1600MHz clock dividers in every component that uses a sub-1600MHz clock (which, to be frank, is all of them).This final clock driver/divider circuit looks like as follows.



Clock Divider and Driver Circuit

Note that the CLK input to this clock divider is not strengthened as much, since the only node it directly clocks is the register it is directly connected to. For the most part, these buffered clocks are directly wired to receiver modules without any strengthening in between, and we have not observed any adverse effects due to board skew of the clock or insufficient drive strength.

The biggest exception to this rule was transmission gates on the clock inputs of the deserializer and the output registers. The idea was that if these clock inputs are locked by the EXT signal, such that when the design is driving the serializer output off chip, none of the deserializer/output register circuitry

is clocked. This in turn reduces our power usage in the EXT-on case, since you reduce switching activity. However, when running SS-corner simulations, we realized that the minimal sized transmission gate we had used had insufficient drive strength - it was unable to provide a clean clock waveform to the deserializer. Since the transmission gate was initially minimally sized, we were able to experimentally verify that on the SS corner, resizing it to size 4 transistors for the PMOS and NMOS components in the transmission gate sufficiently increases our clock drive speed.

There is also the matter of transmitting data to/from the layout. The EN and TX signals must be stepped down from the 1.8V IO voltage to the 1.1V core voltage, and the serializer output must be driven off-chip. Both of these components were moreso a sizing challenge than an architectural challenge - as such we'll address them in Section 2. For now, we simply address them as generic blocks.

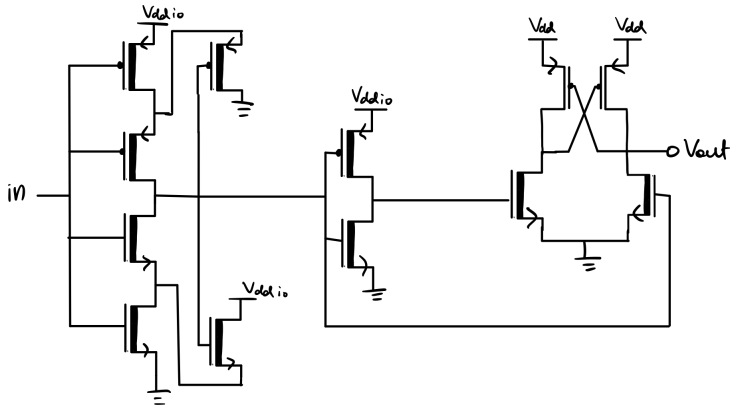
With all these modules in mind, and implicit modifications made to the serializer and deserializer to receive their divided clocks as inputs rather than performing internal divsiion, we have now created well-defined logical definitions for almost every component in our layout. Next, we will establish the topology used for all of our combinational gates and registers.

2 Designing the Cell Library

Unless otherwise mentioned, all of our logic was created at FO1, with $\beta = 2$. This was done in an effort to keep area to a minimum, and to avoid insufficient drive strength issues as much as possible (when driving this logic).

For most gates we used the SCMOS implementations (most of our design is based on NAND/NOT logic). However, we elected to use the transmission gate-based implementation for the XOR gate, for the reasons discussed in section 1.

2.1 1.8V to 1.1V Level Shifter



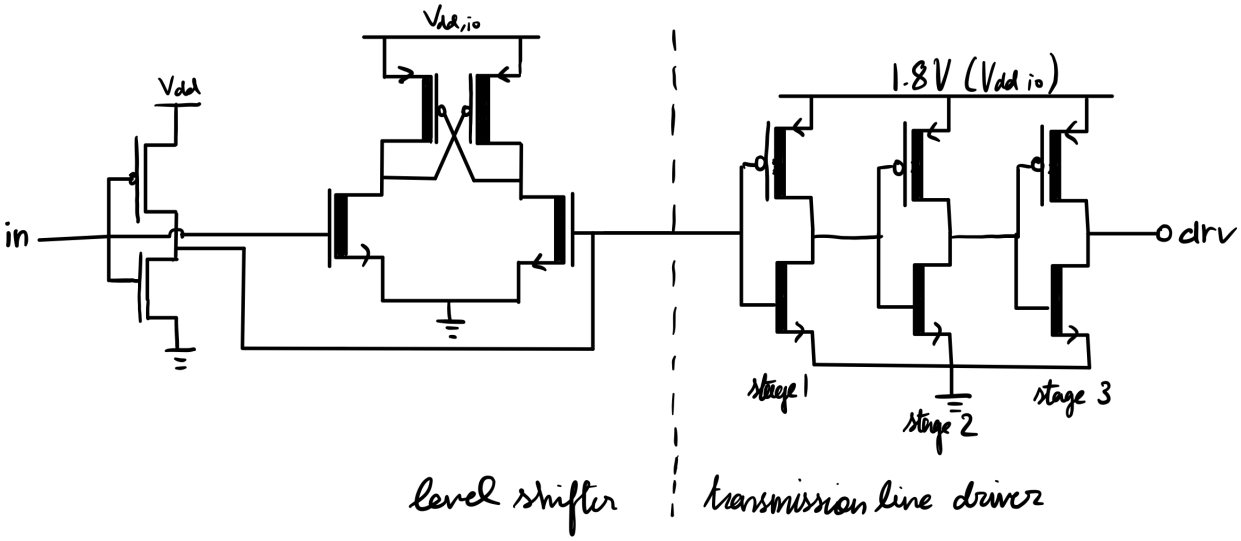
Transistor-Level Schematic of 1.8V $\rightarrow$ 1.1V Level Shifter

Note that all the transistors in this schematic are sized minimally - thick bars on the transistor indicate IO transistors (1.8V operation) whereas the thin lines are normal 1.1V transistors. Their sizing is detailed in the table below.

Type	PMOS Width	NMOS Width	Channel Length
1.8V Transistor	640 nm	320 nm	150 nm
1.1V Transistor	n/a	120 nm	45 nm

2.2 Off-Chip Driver

The off-chip driver consists of two components in serial - a minimally sized 1.1V $\rightarrow$ 1.8V upshifter connected to a superbuffer of inverters to drive the external transmission line.



Transistor-Level Schematic of Off-Chip Driver

The sizing varies between the level shifter and off-chip driver - we separate the sizing for these subcomponents into two tables for readability.

Type	PMOS Channel Width	NMOS Channel Width	Channel Length
1.8V Transistors	320 nm	480 nm	150 nm
1.1V Transistors	240 nm	120 nm	45 nm

Transistor Sizing for Level Shifter

Type	PMOS Channel Width	NMOS Channel Width	Channel Length
Stage 1 Transistors	2.56 µm	1.28 µm	150 nm
Stage 2 Transistors	10.24 µm	5.12 µm	150 nm
Stage 3 Transistors	40.96 µm	20.48 µm	150 nm

Transistor Sizing for Superbuffer

2.2.1 Sizing the Superbuffer

The spec given for the off-chip driver states that the output should be raised from low-to-high within 625 ps - thus, we needed to find appropriate sizing for the last stage inverter such that its $t_{p,HL}$ is 625 ps.

We first needed emprirical parameters for R_{dr}, $C_{g,min}$, and C_{int}. By connecting an FO1 inverter constructed with 1.8V transistors to a 100 fF capacitor, we are able to extract an approximation for R_{dr}, assuming that C_{int} is negligible compared to the load capacitance. After that, we perform simulation on an unloaded and cascaded FO1 inverter(s) to extract its parasitic capacitances. Transient simulation results yielded the following parameters.

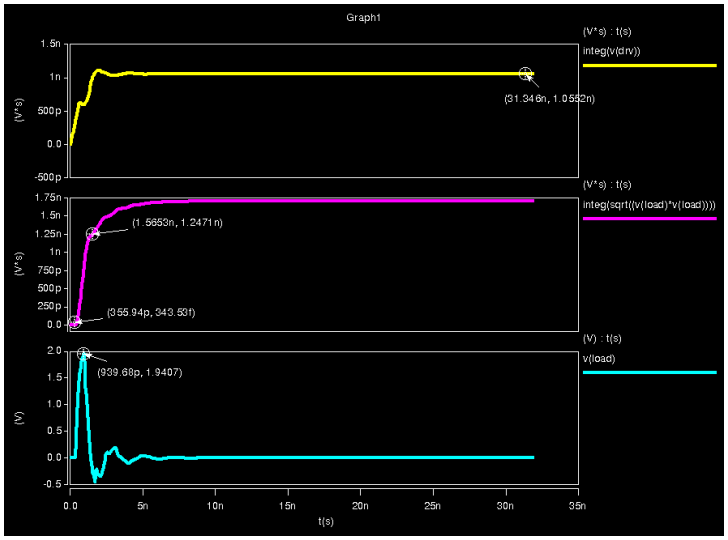
$$R_{dr} \approx 6.4\,\text{k}\Omega$$
$$C_{int} \approx 2\,\text{fF}$$
$$C_{g,min} \approx 2.1\,\text{fF}$$

Given this, we can find an approximate initial sizing for the last-stage inverter in the off-chip driver.

$$t_{p,HL} = 625\,\text{ps} = 2.2 \cdot \frac{R_{dr}}{S} \cdot (SC_{int} + 2.3\,\text{pF})$$
$$S \approx 55$$

Note the modified 2.2 constant - this is used because we are looking for 10% to 90% propagation delay rather than 50-to-50%.

Using this as a starting point for testing, we experimentally found that size 64 inverter created the best balance between having sufficient drive strength, and being able to create a tidy superbuffer afterwards. Given our empirical parameters, we were able to extract that in a superbuffer of 1.8V inverters, there is an optimal $f_i = 4$ - working backwards from this, we decided to create a 3-stage superbuffer with FO4, FO16, and FO64 inverters.

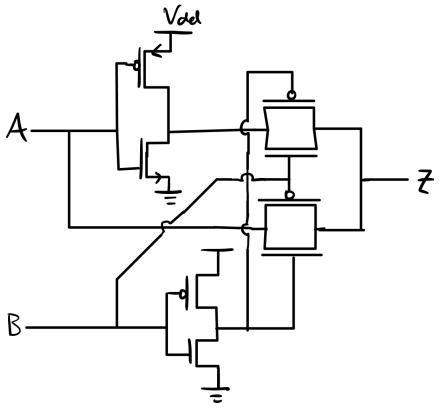


Measuring FOM from 625ps pulse into Driver

Using the procedure detailed in the initial design document and based on the graph above, we were able to confirm that we indeed achieve an approximate 85

2.3 4-input XOR gate

Our entire design requires a single four-input XOR gate, for the PRBS. This is constructed by XORing the first two and last two inputs, then XORing those two intermediate results. As such, when creating the cell library we created a cell for a 2-way transmission gate-based XOR for greater potential reuse. Then in the construction of the PRBS, we cascade these XORs as necessary. The basic 2-way XOR schematic is shown below.



Transistor-Level Schematic of 2-input XOR gate

Note that all the transistors in this schematic are minimum-sized 1.1V MOSFETS, with $\beta = 2$. Refer to the following table for specific sizing details.

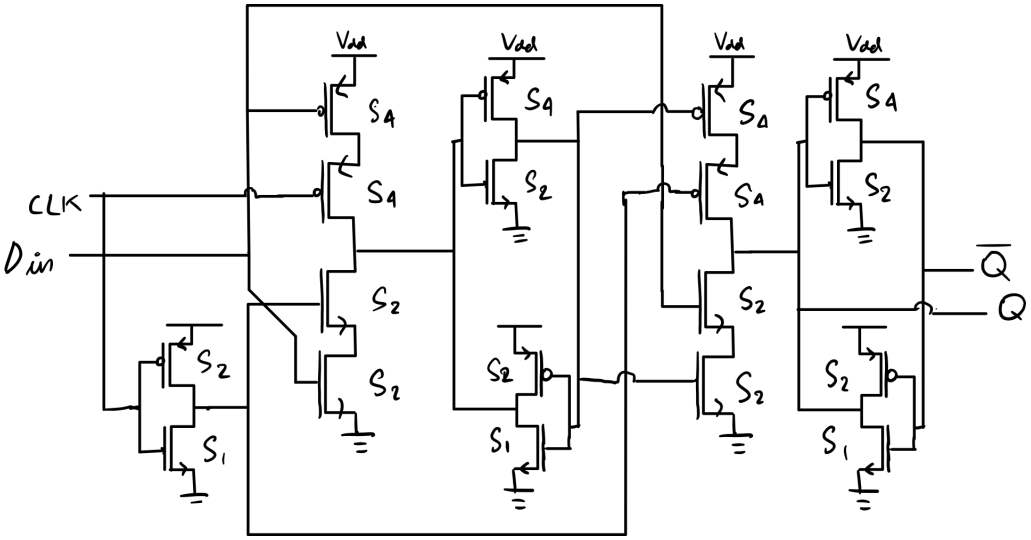
PMOS Width	NMOS Width	Channel Length
240 nm	120 nm	45 nm

Transistor Sizing for 2-input XOR gate

2.4 C^{2}MOS Register

For the most part, this is a standard C^{2}MOS register with a flipflop in between. This is done to avoid some of the concerns with sizing, timing, etc. that crop up when working with dynamic logic. We have also added another flipflop before the output in order to increase our output drive strength.

Also note that our register has an *internal* inverter for CLK. While this does add to our area and power consumption, we made this decision in order to simplify our clock routing throughout the design - on such a tight schedule with such a long list of components to create, we needed to strike a balance between design viability and efficiency - this tradeoff seemed worth it in order to avoid another set of design issues in the form of routing congestion/difficulties. Though most naive implementations are able to get away with using a single clock rail that distributes to every reigster in the design, This decision ended up working in our favor - as is, our layout already has extremely complex routing, and adding another high-fanout line would have only made that routing harder.



Transistor-Level Schematic of C^{2}MOS register

Due to the varying transistor sizing across different parts of the schematic, we've tagged each transistor with a sizing, and list the measurements of each one below.

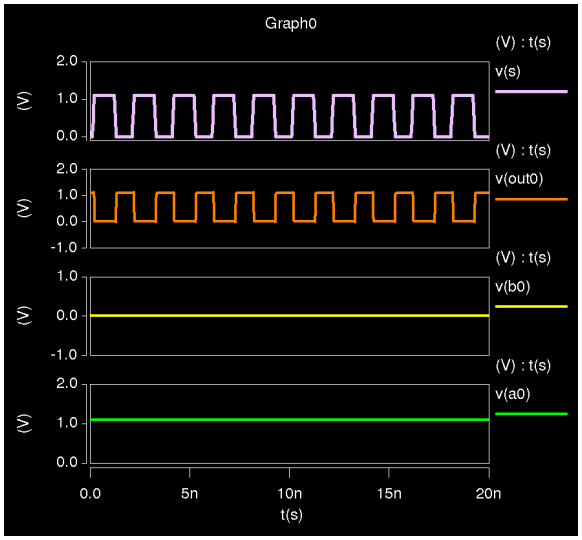
Type	Channel Width	Channel Length
S1	120 nm	45 nm
S2	240 nm	45 nm
S4	480 nm	45 nm

Transistor Sizing for C^{2}MOS register

Note that we also have an alternate version of this register used in certain parts of our design, like the PRBS and the Serializer - this register is essentially a *dynamic* version of the above register. If one were to remove the flip-flops (circular inverter chains) between the two halves of the normal C^{2}MOS topology and the one after the second "latch", then we achieve a normal C^{2}MOS register. This modified topology was used in sections of the design where drive strength was (less so) of a concern.

3 Simulation Results [Multiplexer]

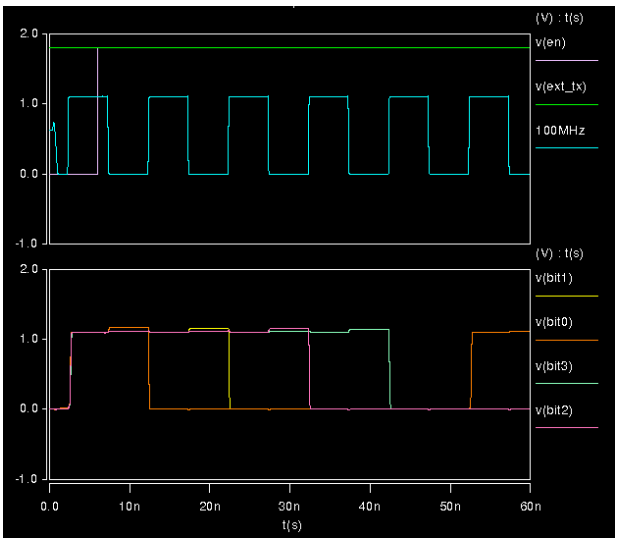
Since our 16-wide MUX is simply many parallel instantiations of our 2-to-1 MUX, we below show a waveform detailing behavioral verification of our 2-to-1 MUX in the TT process corner. It is assumed that if this operationally passes, then going forwards our MUX is logically sound, and is a prerequisite to the successful functioning of our SerDes stack.



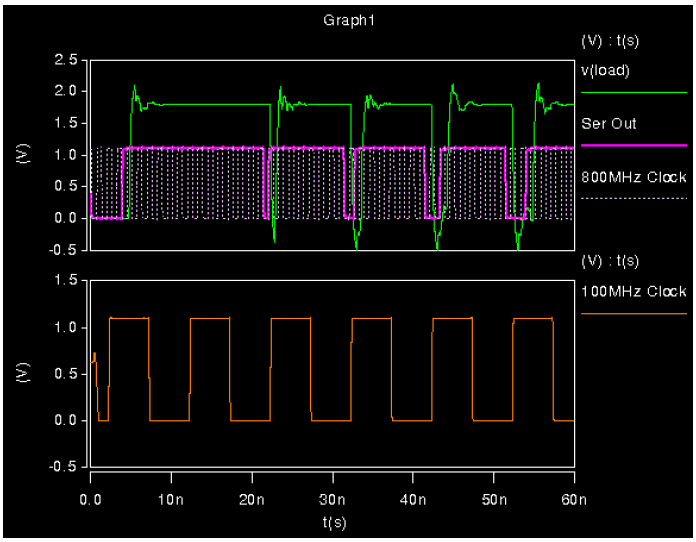
Simulation Results for 2-to-1 MUX

In the above simulation, the signal S is our select input, the a/b signals are our data inputs, and the out signal is the selected MUX output.

4 Simulation Results [TT Corner]

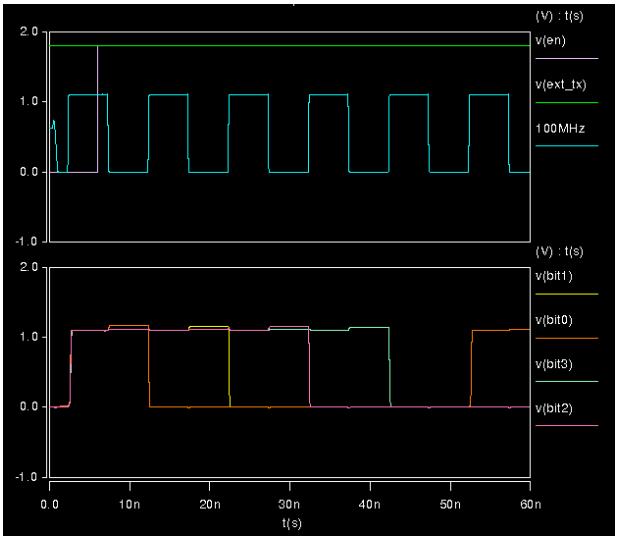


TT Off-Chip Simulation: Input stimulus and Low 4 PRBS values

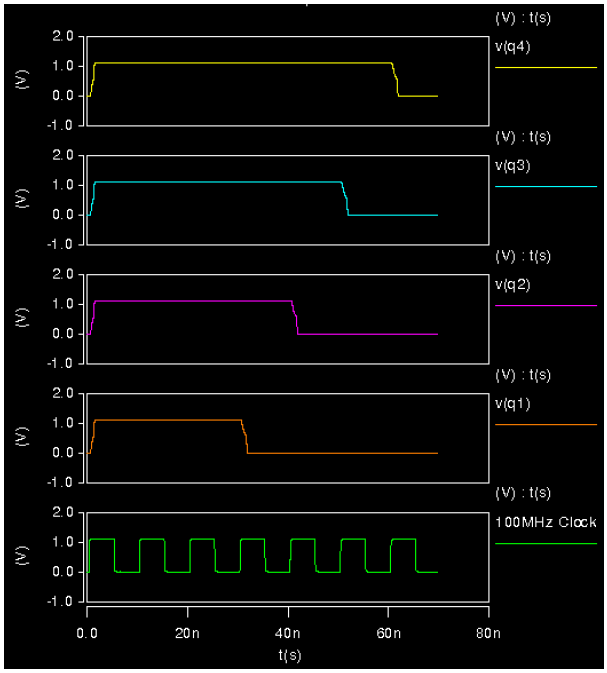


TT Off-Chip Simulation: Serializer Output, Load Transmission Line output with 800MHz clock guideline

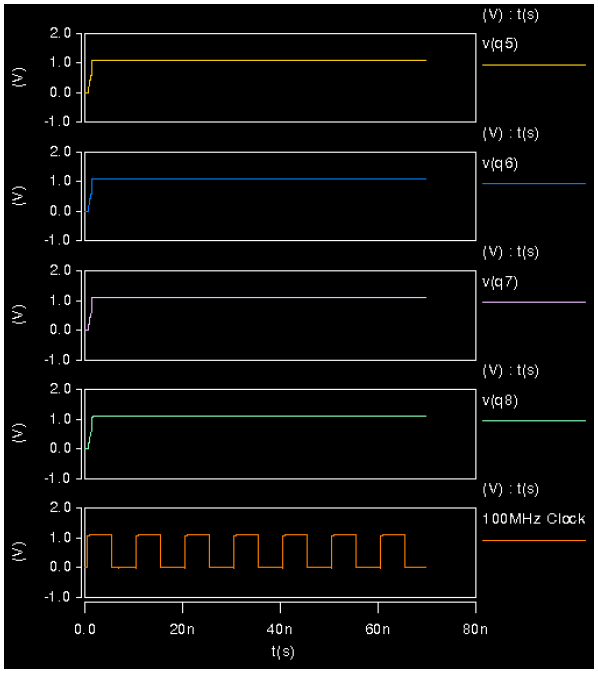
As can be seen in the images above, we first reset the chip by leaving EN low. Once EN is raised, the PRBS immediately latches a new value, and in the subsequent cycle the PRBS shifts out that data to data out. Since the PRBS is guaranteed to reset to active high, we have only displayed the bottom 4 bits of the PRBS, to show that the active high reset works and that the 4 LSBs of the PRBS are being updated over 4 cycles of the 100MHz clock. As the PRBS values update, we can see that this is reflected on the load terminal, which in our netlist is connected to the transmission line. We have shown both the direct serializer output and the transmission line node both for completeness - the graph shows that the FOM reflects similar behavior to our intiial testing, and that there is a noticeable delay before making it off-chip, as expected. We show a trace of the 800MHz clock for clarity, and we can see that a new value is displayed on the output on every alternating edge of this 800MHz clock, as expected of our dual-edge design.



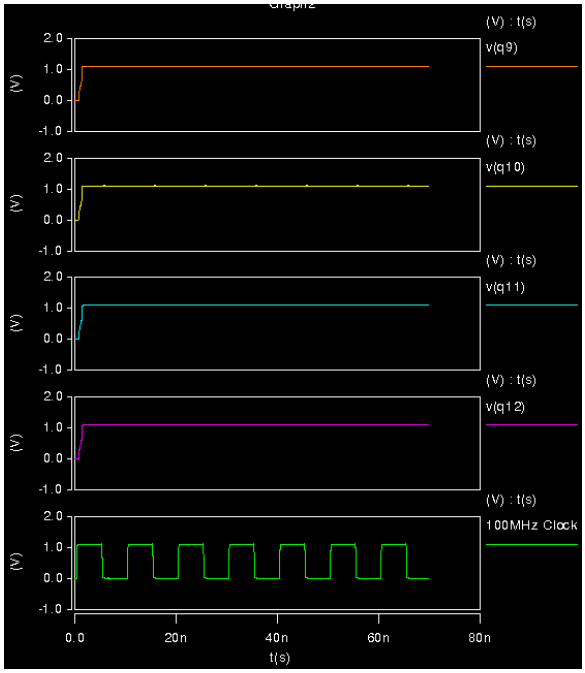
TT simulation results for Loopback Mode - Input Stimulus and PRBS state



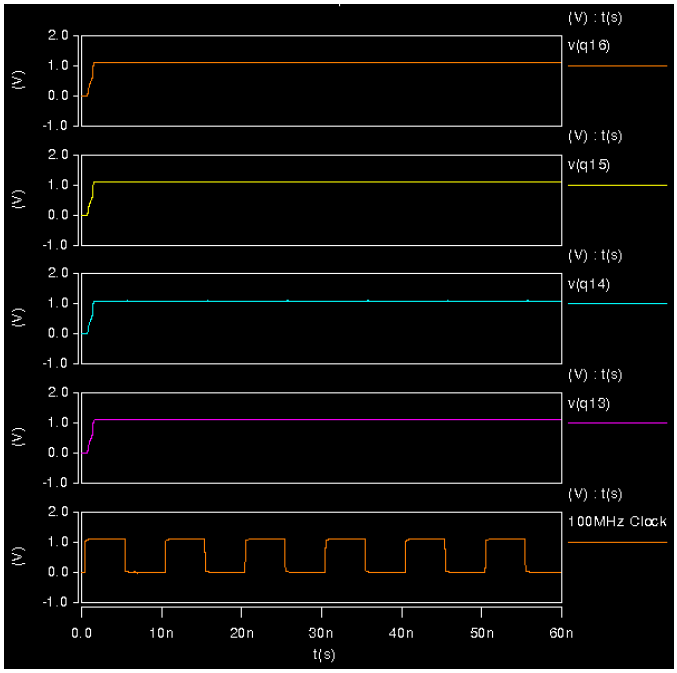
TT simulation results for Loopback Mode - Low 4 bits of output D registers



TT simulation results for Loopback Mode - Bits 4-8 of output D registers



TT simulation results for Loopback Mode - Bits 9-12 for output D registers



TT simulation results for Loopback Mode - Bits 13-16 for output D registers

Analagously, we attach above simulation results for loopback mode operation of our SerDes stack. The input stimulus is the same as before, and we omit the upper bits of the PRBS (which do not update during our simulation period) again. We can see that on q1-q4, the lowest bits of the output register, the value shown on the PRBS is latched two cycles later - thi is due to the fact that if the PRBS is updated on cycle 1, the deserializer will latch the entirety of that data by the end of cycle 2, and then the FF will capture that data on cycle 3 - which is the expected behavior. We can also see that this effect cascades, with the registers all the way up to Q4 latching zeroes in alignment with the PRBS cycle. The simulation period has been increased in order to keep up with the additional delay required to monitor the D register array.

4.1 Power Report

Mode	Total Power
EXT=0, EN=0	187 µW
EXT=0, EN=1	414.6 µW
EXT=1, EN=1	2.59 mW

Total power for different operating modes in the TT process corner

At this time, we are unable to surmise where the extreme power consumption is coming from when EXT and EN are both on - our off-chip driver was the first and primary suspect, however probing its individual power consumption as well as separately testbenching it with "worst-case" parameters both yielded very reasonable results, which are insignificant compared to the overall power consumption. For greater accuracym, and to avoid error propagation from our odd simulation results, we have separately testbenched the off-Chip driver for the power results found ahead.

PRBS Power	Serializer	Deserializer	Total Power
17.65 µW	19.86 µW	372.2 µW	414.6 µW

Subcircuit Power for EXT-TX=0 and EN=1 operating in TT process corner

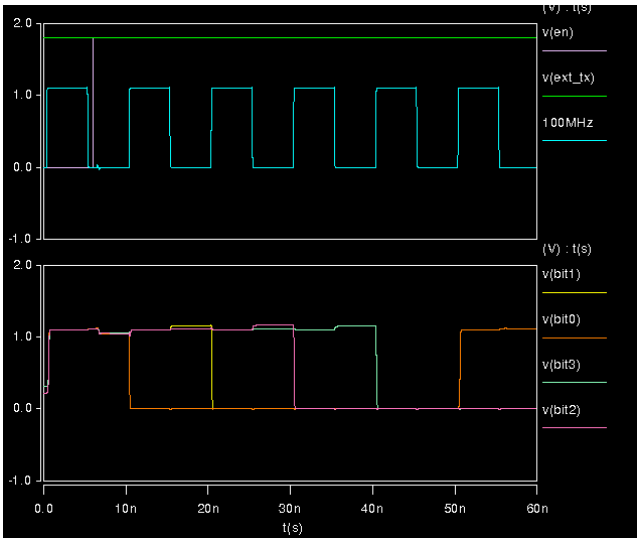
We were initially extremely surprised by the operational power difference between the Serializer and Deserializer - however, it made sense once we realized that we had used an older model of DETFF/D-register for the PRBS and Serializer, where the register did not have flip-flops. As a result, despite the fact that the Deserializer has only double the average switching activity of the Serializer under similar operating conditions, it had significantly more power consumption due to the approximately 8 more transistors per register that it has to drive (nearly double the transistor count per register).

Off-Chip Driver
7.32 µW

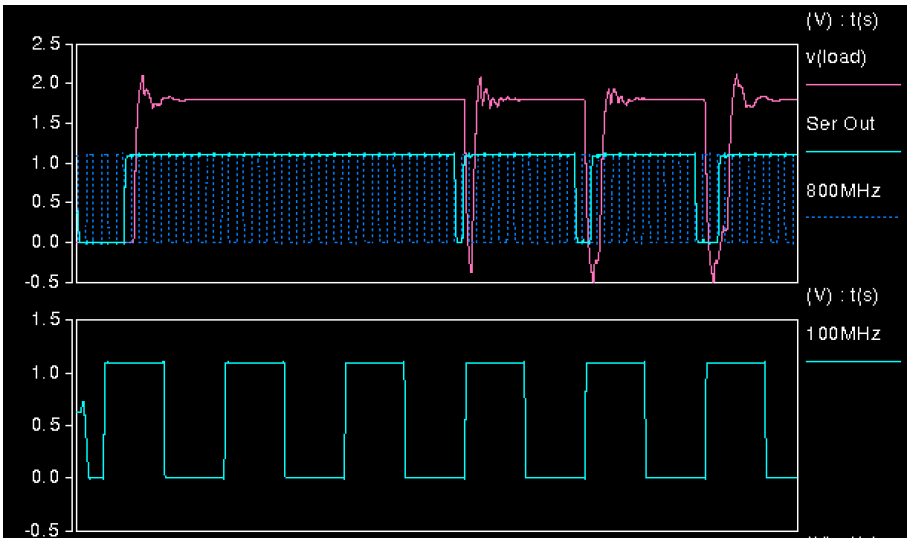
Subcircuit power for EXT-TX=1 and EN=1 operating in TT process Corner

5 Simulation Results [FF Corner]

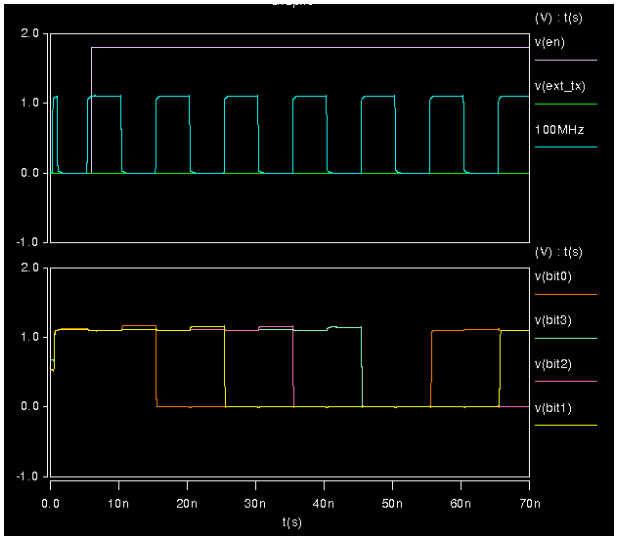
The simulation results for this process corner were virtually the exact same as our TT testing. The waveforms have been listed below. It should be noted that we initially ran into some bugs with our MUX propagation delay being too slow, however that was correct by strengthening the size of the inverter after the EN level shifter.



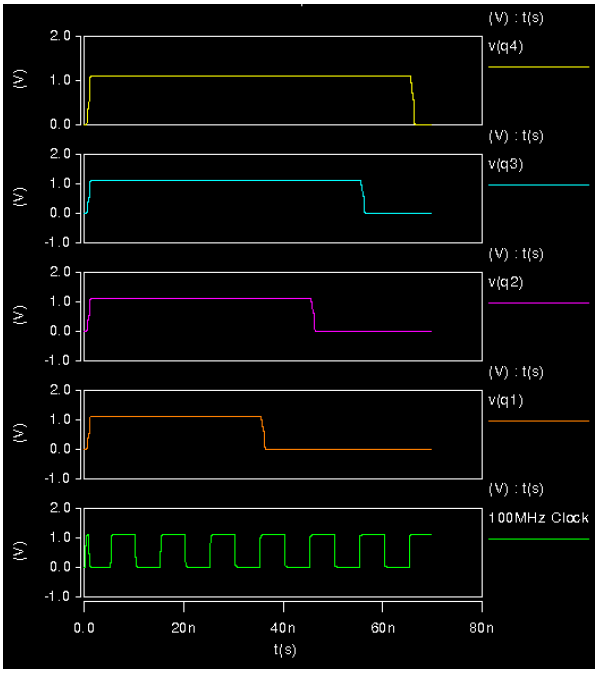
FF Simulation for Off-Chip - Input Stimulus and PRBS state



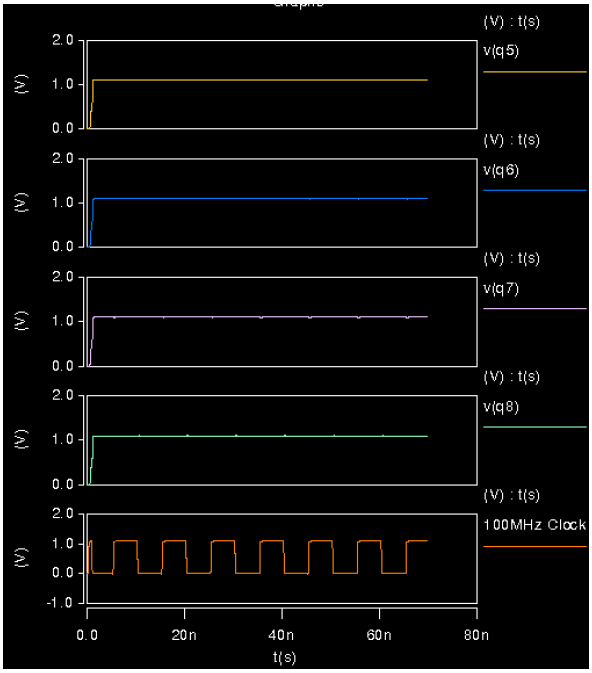
FF simulation results for Off-Chip - Serializer Output and Transmission Line Load Voltage



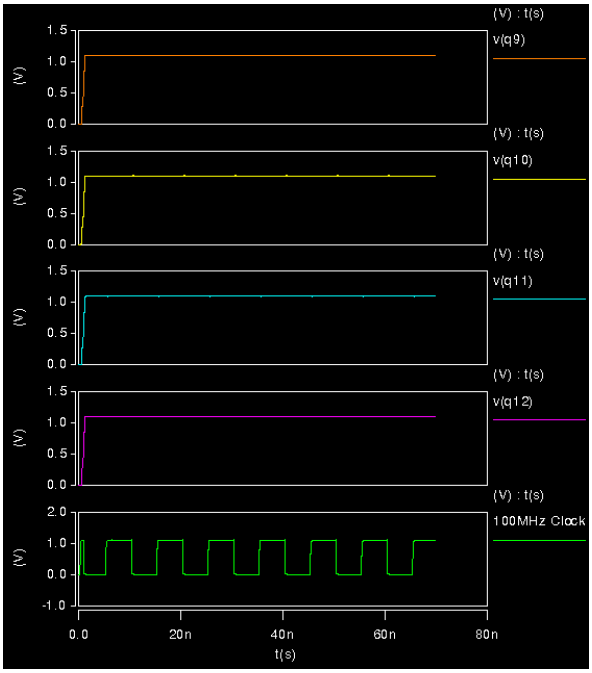
FF simulation results for Loopback Mode - Input Stimulus and PRBS state



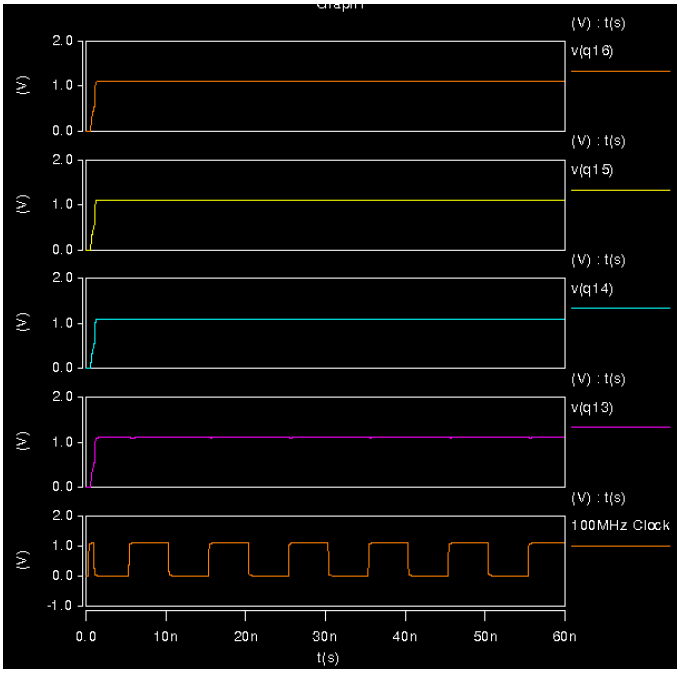
FF simulation results for Loopback Mode - Bottom 4 bits of output D registers



FF simulation results for Loopback Mode - Bits 5-8 of output D registers



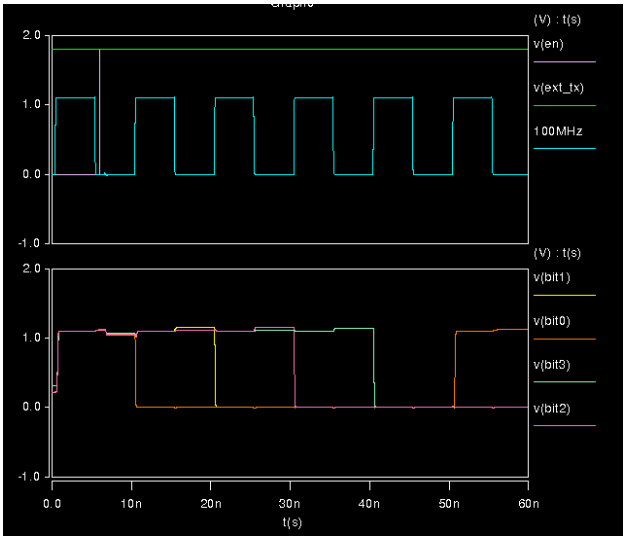
FF simulation results for Loopback Mode - Bits 9-12 of output D registers



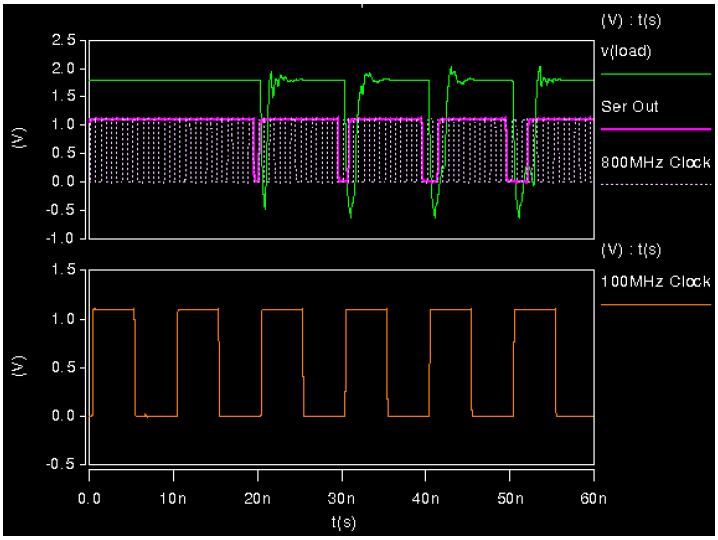
FF simulation results for Loopback Mode - Top 4 bits of output D registers

6 Simulation Results [FS Corner]

The simulation results for Off-Chip driving the FS corner were relatively standard - they match whatever simulation results we achieved on the FF and TT corners.

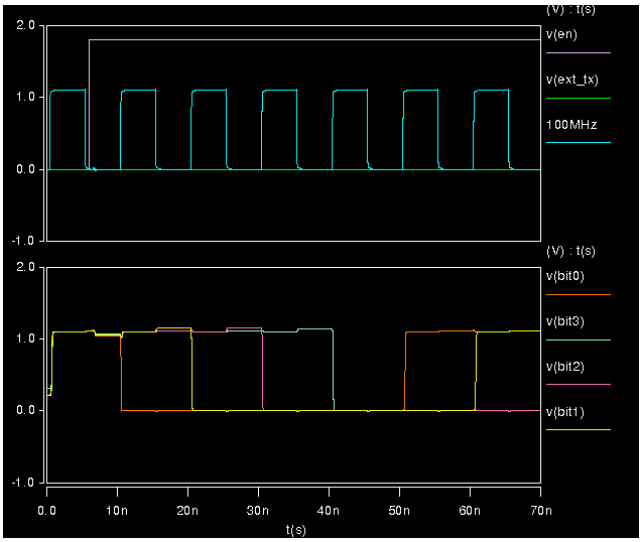


FS simulation results for Off-Chip Mode - Input Stimulus and PRBS state

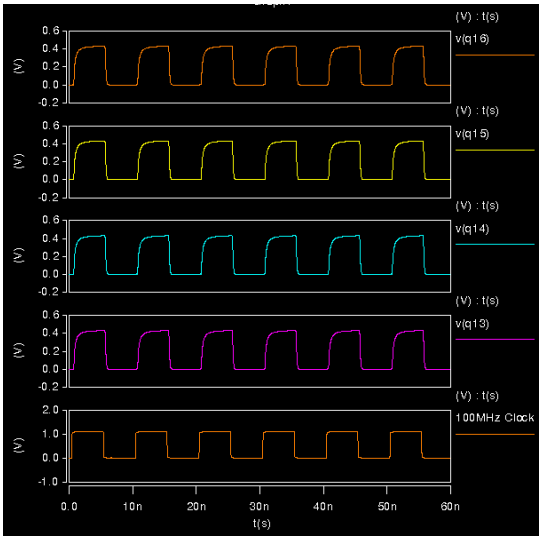


FS simulation results for Off-Chip - Serializer and Output Contact Values

However the on-chip driver simulation is where things started to go downhill. We applied the same stimulus as usual, however the following results were observed.

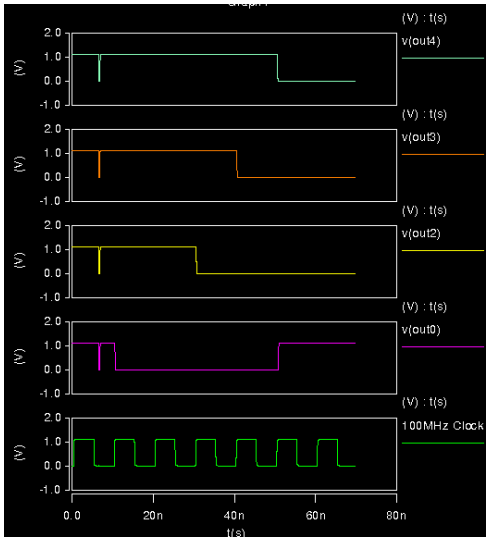


FS simulation results for Loopback Mode - Input Stimulus



FS simulation results for Loopback Mode - Incorrect D Register Values

For some reason, the register output array is oscillating in tangent with the clock, regardless of what the driver input is. We noticed this behavior for all 16 registers. This felt odd - we were sure that we had designed the DeSerializer correctly. What was going on? We looked one layer deeper, and checked the Deserializer output.

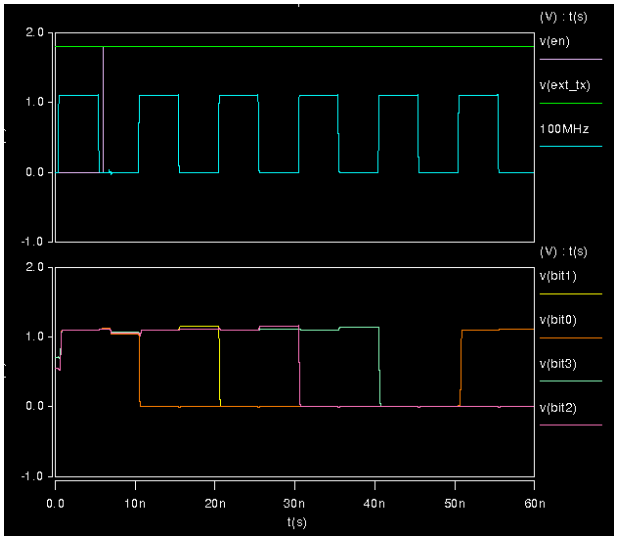


FS simulation results for Loopback Mode - Observing Direct Deserializer Outputs

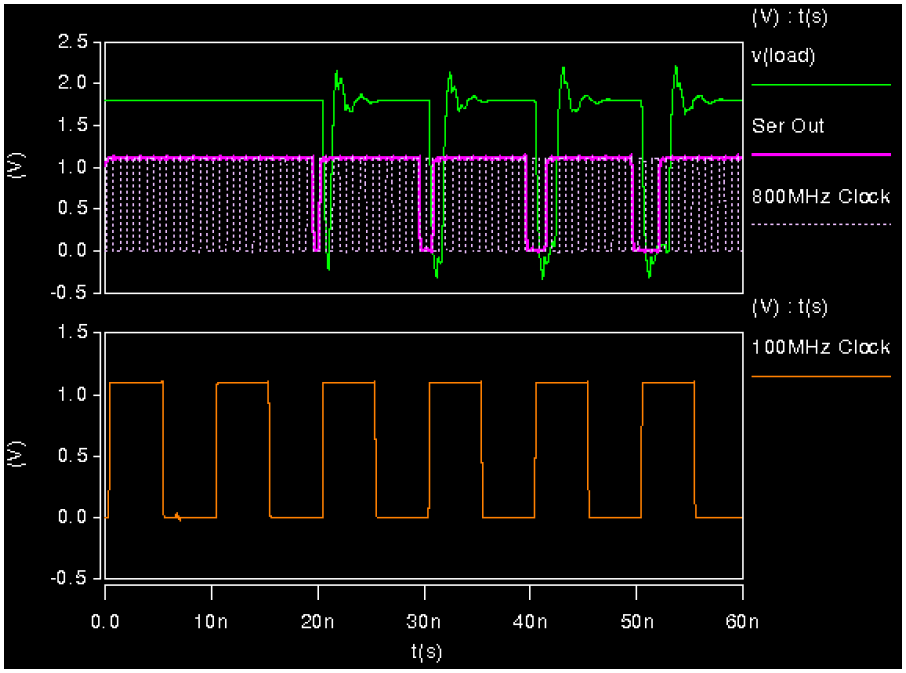
Odd - the deserializer output is being driven correctly, but the output being shown by the 16 D registers is incorrect. Unfortunately, due to time constraints, we did not have time to fully debug this issue, and are not sure what the source of the problem is. However, we believe that this simulation confirms the operation of our deserializer on the FF corner, and that it can drive an output load - however other parts of the design that interact with the registers may be incorrectly implemented. In preliminary simulation runs, we noted similar behavior on all of the partial S corners - as a result, instead of observing the 16-register-array output in our waveforms, we will only be showing the bottom 4 bits of the deserializer, which update correctly as the serializer continues its operations.

7 Simulation Results [SF Corner]

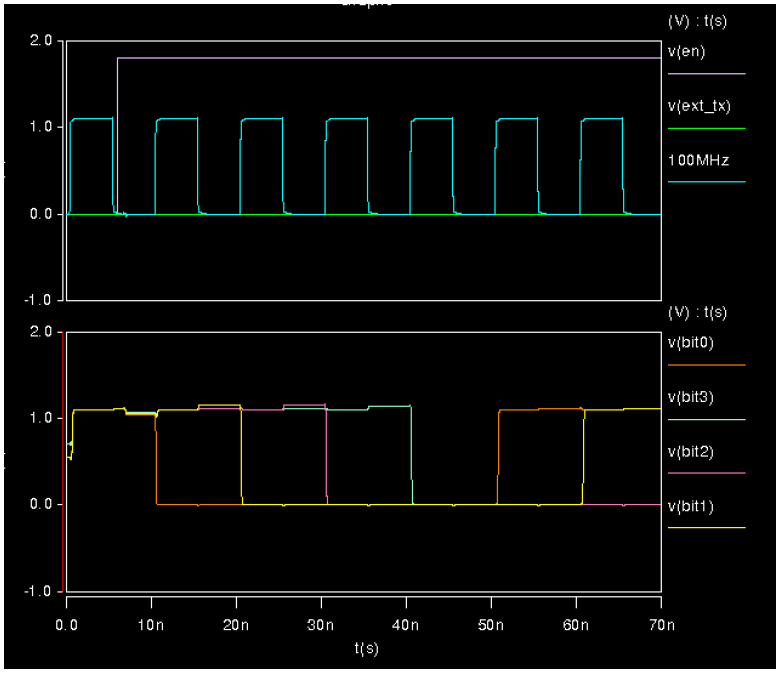
These simulations are pretty much identical to the FS corner - we succeed in driving the output signal off-chip, but run into issues driving the output D-registers connected to 20pF. We have omitted the DReg's Q ports in simulation as a result, instead opting to directly show the "out" port coming out of the Deserializer.



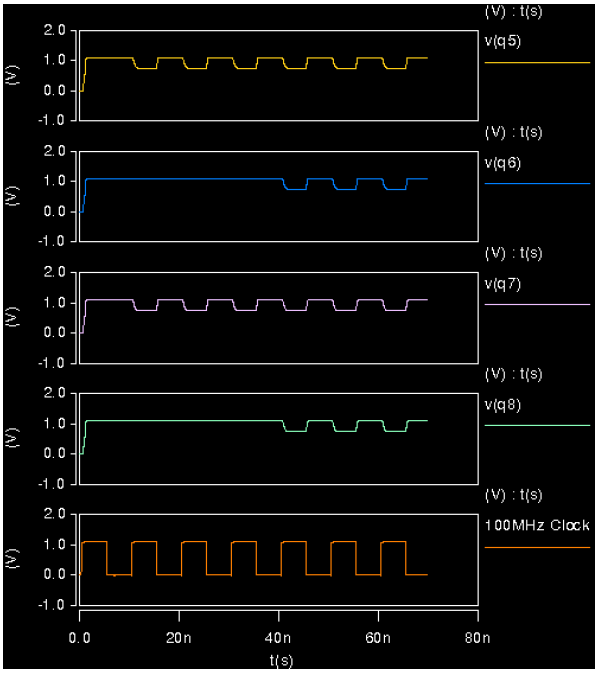
SF simulation results for Off-Chip Transmission - Input Stimulus



SF simulation results for Off-Chip Transmission - Off-Chip Driver's Values

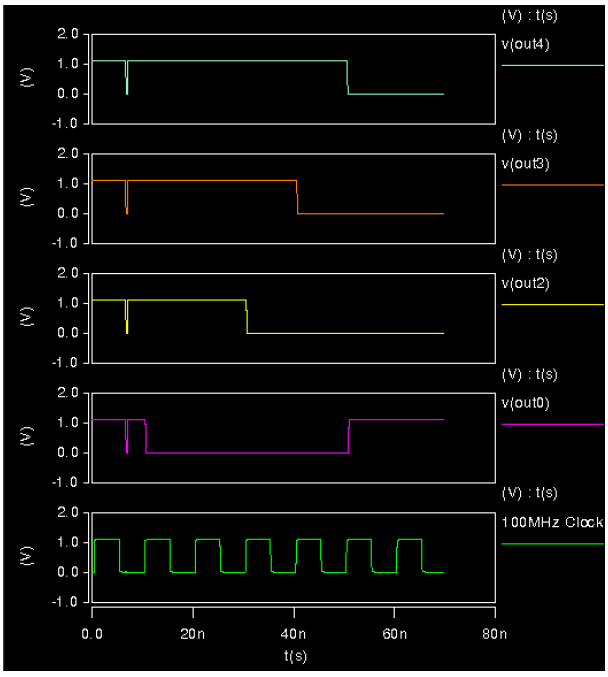


SF simulation results for Loopback Mode - Input Stimulus



SF simulation results for Loopback Mode - Weird Register Output Values

We once again notice similar register oscillations - this time however, the register (still oscillating with the clock) is always held high. We observed no out of the ordinary voltage values or shapes in the input clocks, the input data, or other control signals. We believe that this may be an issue with board skew, however we did not have time to analyze the relative transient behavior of relevant control signals. We once again, however, are able to confirm the functionality of the Deserializer by the simulation trace shown below.

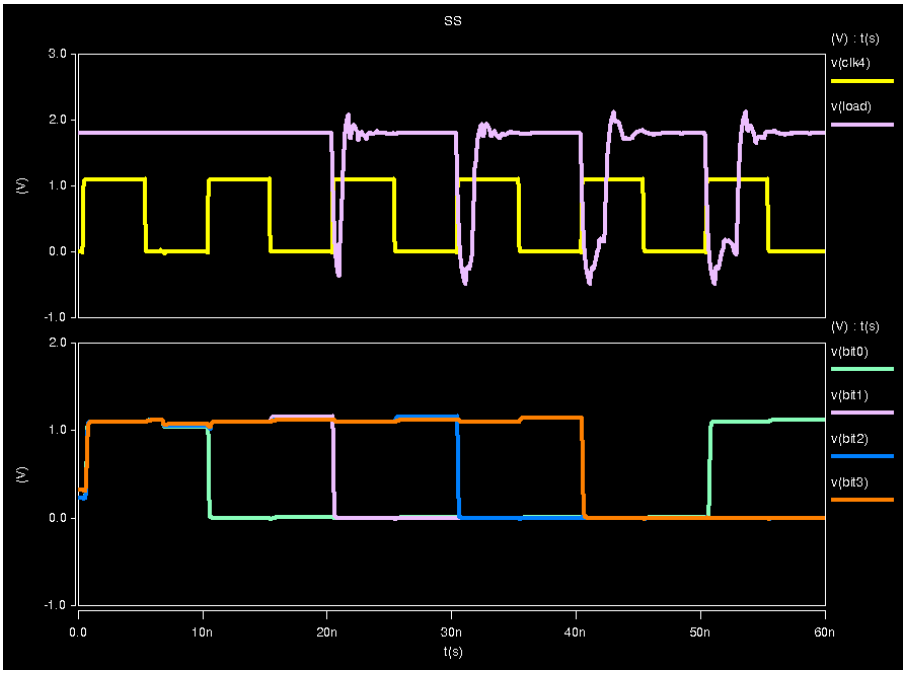


SF simulation results for Loopback Mode - Directly Observed Deserializer Ouput Values (Bottom 4 Bits)

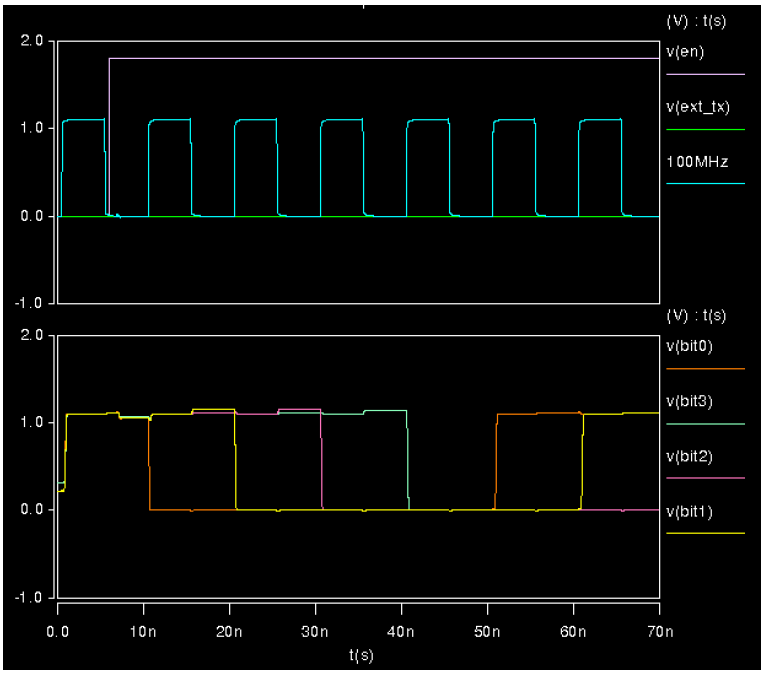
8 Simulation Results [SS Corner]

Do note that the output of the serializer pre-transmission line was omitted in the simulation - this is purely an inconsistency in our testing setup, and does nto reflect a deficiency in simulation results on the

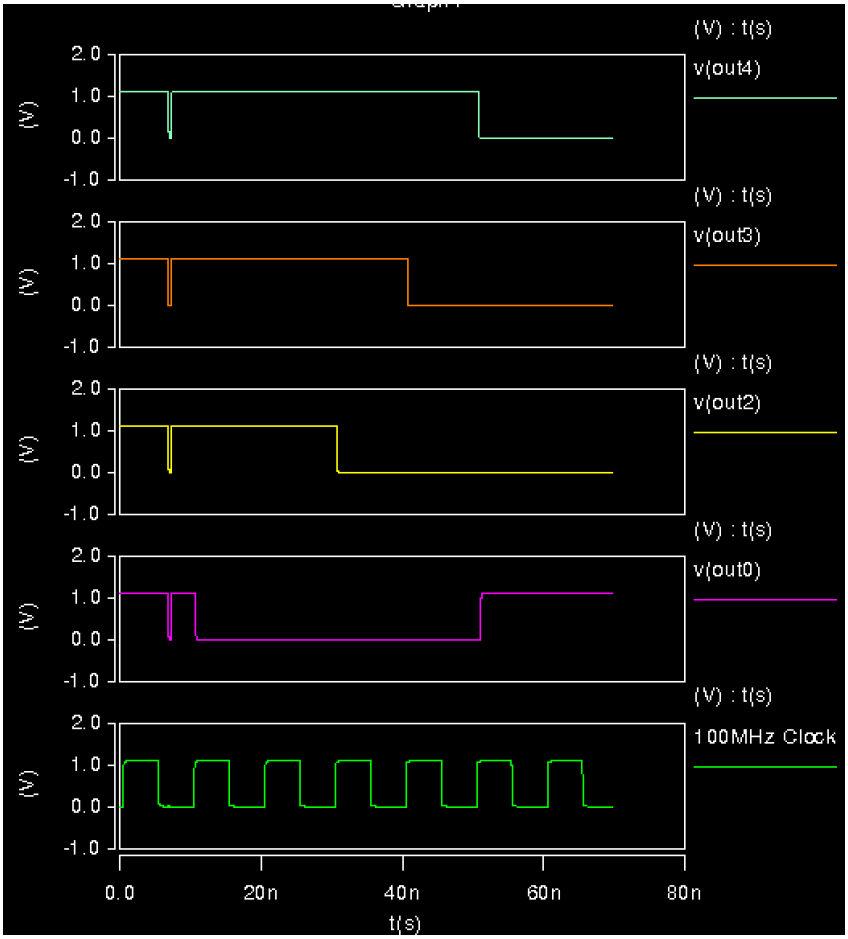
Serializer's end. The off-chip transmission results once again mirror what we saw in prior simulations, and we believe them to be self-explanatory when compared with prior results.



SS simulation results for Off-Chip - Input Stimulus and observed Transmission Line Load Value

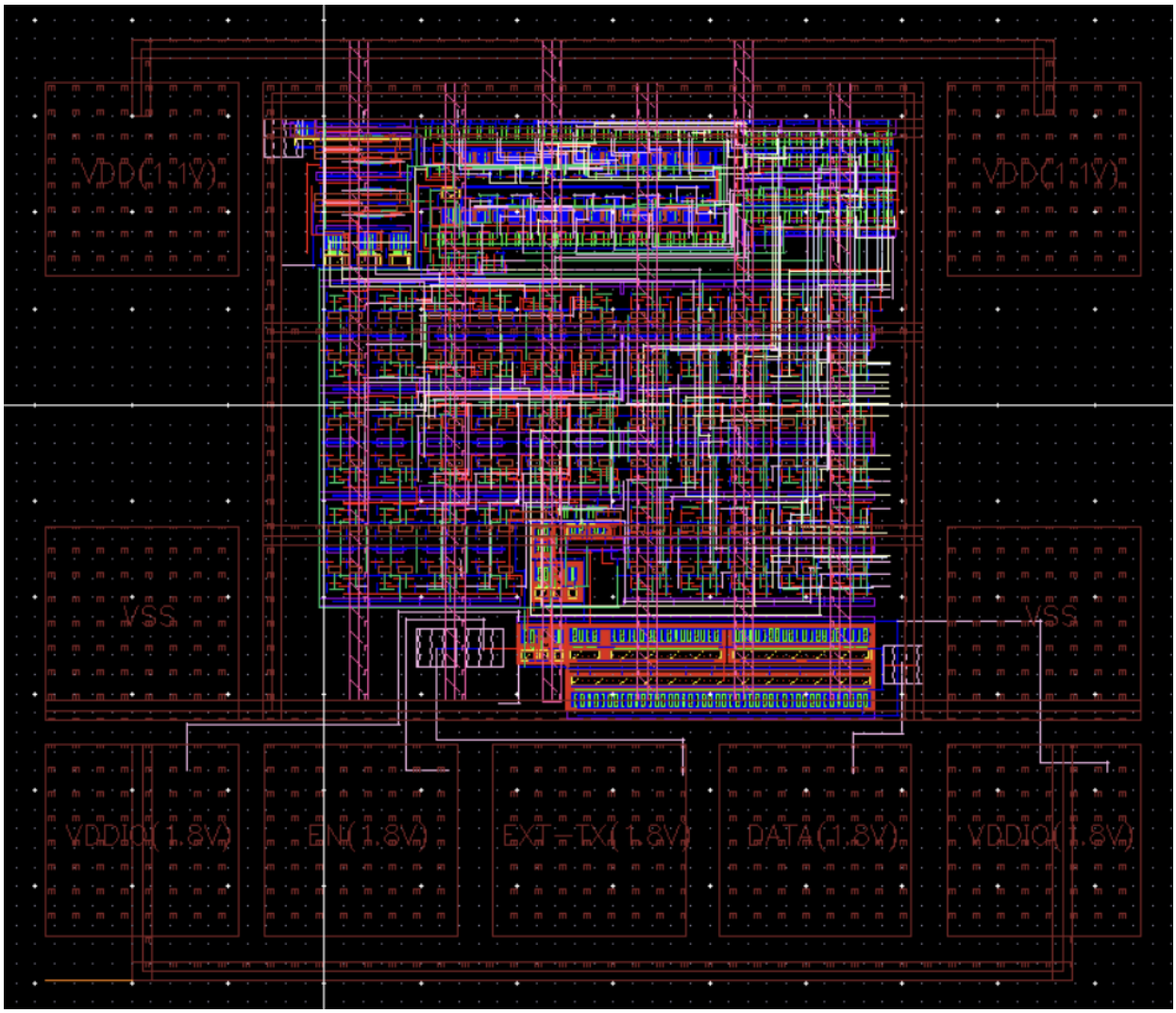


SS simulation results for Loopback Mode - Input Stimulus



SS simulation results for Loopback Mode - Directly Observed Deserializer Outputs

9 Layout



Top-Level Layout of our Design

When laying out our dsign, we started by trying to implement a unified Cell library of sorts this includes the register/additional gates detailed in the transistor-level schematics section, as well as some standard sized NOT and NAND gates - this way, we would have most of the logical functions and schmeatics needed to create complex logic in our design on hand.

From this point onwards, we tried to work on as many parts of the layout as we could in parallel - for the remainder of the final project, we essentially worked on the deserializer, serializer, and PRBS in parallel. As new generic cells were created, like the DETFF, we maintained a Google Drive "parts library" of sorts, such that other group members could download ready-made schematics as necessary.

The serializer was the first component that we completed. As the PRBS and deserializer neared completion, we also started creating layouts for some other useful components necessary for full integration, like the components mentioned in the Top-Level integration section. This parallel approach had both its pros and cons - since we ran into many more bugs at once, we were able to solve those in parallel, meaning that we also knew what to anticipate going forwards in other parts of the design. However, it also led to heterogeneity in the cell sizing - note that the row height in different parts of the design are marginally different, due to different design preferences of each group member - how long each finger should be (maximally), how long of a trough you should leave between the NMOS/PMOS transistors for routing, etc. Another caveat that we weren't immediately aware of was namescheme - since different group members used different terms to refer to similar ports, we ran into many LVS errors as a result of naming inconsistency.

However despite this heterogeneity, we were still able to achieve a relatively uniformly laid out design. Especially compared to some of the other fully integrated designs we've seen among our peers, we feel that our layout is very compact. We are especially proud of the fact that we were able to make our design this compact *despite* the fact that our serializer is more architecturally complex than many other implementations in this class, and that we required more sophisticated clock routing/circuitry in order to make our design work.

One strategy that our group used to minimize our area footprint was alternating the placement of the PMOS and NMOS transistors in vertically stacked cells - by doing this, we can separate the PMOS cells of the top and bottom cell with a row of M1-to-NWELL vias for body V_{dd} connections, and can place a row of M1-to-PSUB vias for V_{ss} connections in between the NMOS cells. This allows us to achieve a much more compacted layout than would be otherwise possible.

10 Overall Results

We are very happy with the results that we were able to achieve functionally with regards to our serializer and deserializer - apart from last minute changes that needed to be executed in order to make sure that the clock was sufficiently powerful to drive the serializer and all other pieces in our overall design. However, we're extremely disappointed by the fact that despite executing nearly our entire design correctly, including the extremely complex tree-based serializer, we failed to be able to get accurate results out of the 16-register array at the output of the deserializer. We believe that this, along with other issues throughout the design, are due to board skew - while performing final simulations close to the deadline, we began to encounter numerous errors in simulation behavior that were caused due to oursizing on our drivers being too small. Say, for example, the 16-wide 2-to-1 MUX - something that we had overlooked was that our FO1 inverter on the output of the level shifter was not strong enough to drive all 16 MUXes at the same time - as a result, the output to the transmission line was not raised fast enough in slower process corners.

Something that we really overlooked was the fact that board skew effects reared their head, not in the TT process, but in *any* of the slower processes. On the other hand, insufficient drive strength issues showed up primarily in the FF process corner. Due to how late we realized this, we were able to fix nearly all of these bugs outside of the register array. This is, from what we can tell, the biggest issue with our design right now, and we are disappointed that we were unable to fix it. We realized rather late on that many of our issues with data desynchronization were coming from the exact thing that we had been targeting the whole time - our sizing. Since nearly our entire deisgn is minimally sized, we are more prone to issues associated with insufficient drive strength. This seems sto be a common theme looking back, that we did nto give enough forethought when it came to the sizing of our registers.

Another bug that we found in our design, which manifested in numerous ways, was insufficient clock buffer sizing. Initially, we found that our clock was looking very rounded on some of the faster processes - as a result the output would not function at all. However across the board, we noticed board skew issues - the clock skew between the PRBS and the serializer meant that the serializer latched the *updated* PRBS data every cycle, not the one from the previous cycle. We believe that this is due to insufficient drive strength from the PRBS, meaning that we needed to increase the PRBS sizing by potentially a factor of 2. Time constraints were the biggest issue we ran into.

When working on top-level integration, i.e the bondpads, we ran into many, many issues with the EWS machines. At one point, all of us were unable to edit netlists or reliably run any simulations for 2-3 hours on end the day of submission. This was not an immediate phenomenon - we had been facing it for multiple days up to this point. However due to the proximity to the submission deadline, our decision to take power simulations only *after* finishing the top-level power routing and bondpad layout shot us in the foot. We not only were unable to run simulations for a while, but we soon realized that our initial integration had also run into shorts somewhere in the design. It was truly unfortunate how many things went wrong at once, but it's also something that we could not have anticipated. Considering all the time and energy that had been put into making sure we could come out with a neat, optimal design by the deadline, it was truly heartbreaking to realize that not only was our design not fully working, but that due to circumstances beyond our control we wouldn't have even been able to meet the deadline. Thankfully we were able to figure out the bondpad issues and get it to correctly run in HSPICE, but it was truly a stressful ordeal.

All in all, we all were able to gain a lot of design intuition from this experience, to the point that during the final few days we were able to predict the source of errors relatively accurately. One good example of this was our MUX propagation delay issue - our first guess, insufficient driver sizing, was

right on the money, and our first patch was correct. We believe that this project has made us all better design engineers, and know that we have grown greatly from the experience.

11 Work Division

We adopted a rotating model for the initial part of the project - everyone worked on different fundamental cells and logic gates for the final product. However, around Fall Break, we started to specialize into different components. Derek mainly did the serializer and deserializer stack, Kevin continued building additional combinational gates as needed (MUXes, clock buffers, larger cell chaining), James built the PRBS, and Pradyun helped work on any top-level integration components (register array, off-chip driver, etc.) and did most of the final report. Depending on who wasn't as occupied immediately, we all tried to pitch in with the errors other people were having (there were a lot) like LVS, DRC, or the (many) HSPICE non-convergence errors we were having.