Lucas Elroy
Date 2/21/24

Conway's Game of Life Hardware Accelerator

The goal of this project is to test my ability to create hardware-minded RTL design. Instead of going over Conway's Game of Life this link will describe it better than I can https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.

Accelerator Architecture:
The computation begins by providing the accelerator with an initial game board and the number of frames to run. Subsequently, the accelerator processes the game for the given number of frames and returns the final board state.

Data Path:
The primary computation takes place within the "Cell Array". This array emulates the game-of-life board, where each cell represents one square in the main playing area, and the border consists of dummy cells that are always dead. The array size is determined by a parameter, board_width_p, which sets the side length of the square array. Each cell is connected to its eight adjacent neighbor's state. In each compute cycle, every cell examines its neighbor's states and updates its state. There are data paths from the accelerator's input to each cell to initialize the board state and from each cell to the accelerator's output to provide the final board state. These data paths are represented by a single wire bus.

Control Path:
The control logic for the accelerator is in a module called"controller". This module is responsible for managing all control signals in the accelerator, including the IO handshake signals (ready-valid) and the Cell control signals. Additionally, the controller keeps track of the number of frames to compute using the parameter max_game_length_p, which determines the maximum number of frames that can be calculated at a time.

IO Logic
For input and output data communication, a single-channel BSG Link is used with the same FSB packet for communication The FSB data packet is 80-bits long (the first 4 bits denote the destination ID, the next bit is 0 when sending data, 1 when sending data, and the following 75 bits are for the data). The BSG Link system splits this 80-bit packet into 8-bit chunks (called "flits") as it transfers data from off-chip (the testbench) into our accelerator SoC and then reconstructs the 80-bit packet on the other side, using a DDR clock controlled by the IO clock in our timing constraints and testbench. This means that every IO clock cycle, two 8-bit flits are sent. These are all systems that are in place through the BSG setup and were not developed by me. I decided to use these in order to challenge myself to work within the constraints of these already developed communication protocols.

In the current design, we utilize only the lower 64 bits of the available 75 bits (to simplify interfacing with the testbench with the traces). Therefore, for every 64 bits we wish to send to the accelerator, a full 80-bit FSB packet is required. However, if the board size exceeds 64 bits, an additional layer is added atop the FSB packet. The "Input Data Channel" and "Output Data Channel" blocks in the accelerator architecture perform this task, splitting every 64 bits into individual 80-bit FSB packets, transmitting the data over BSG Links, and reconstructing the message on the other end as a single message. This proved to be far from a trivial exercise.

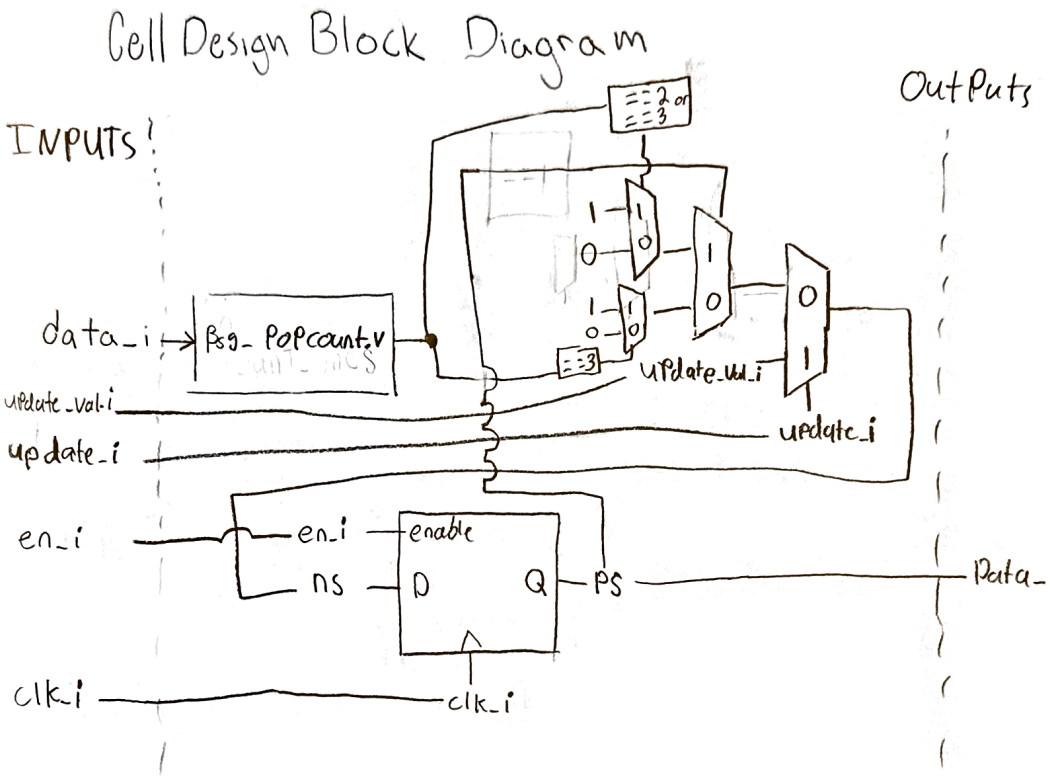


Figure 1: General Block Diagram for Single Cell, Represented by bsg_cgol_cell

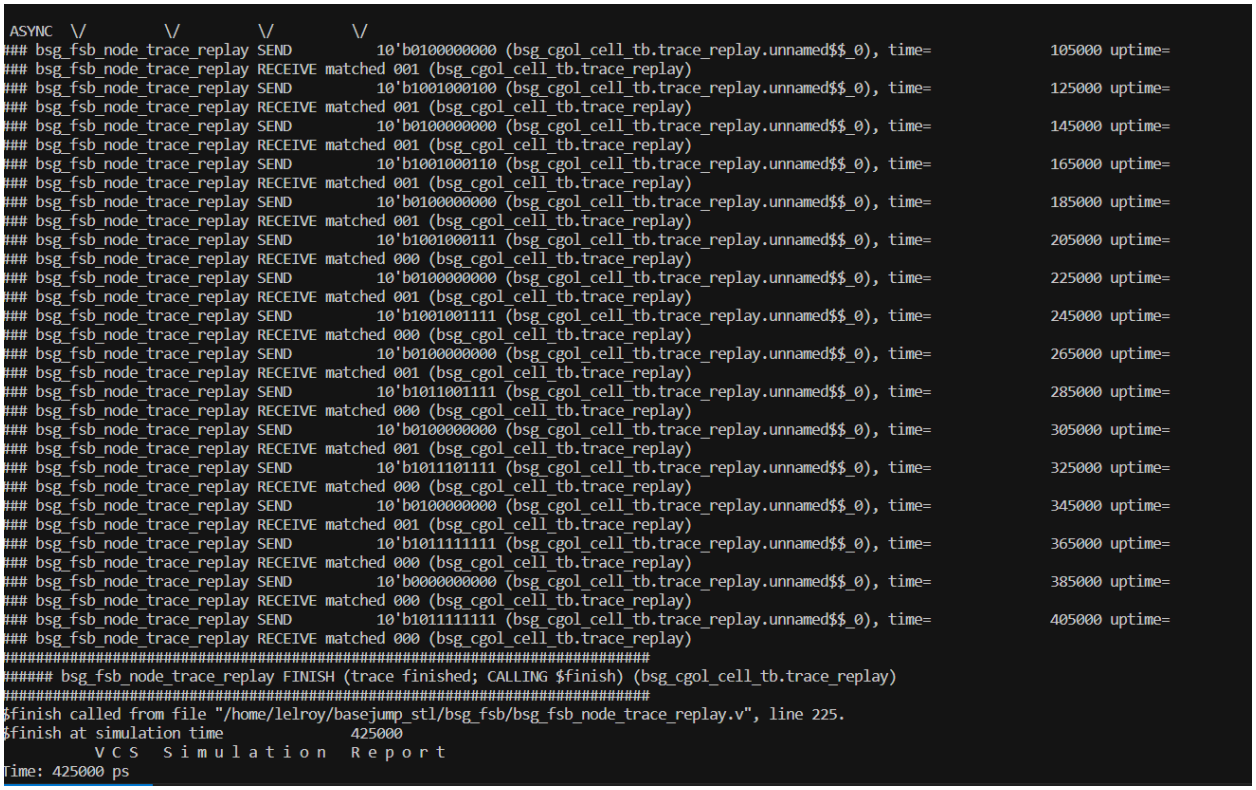


Figure 2: Simulation Traces for the cells Passing!

PPA Analysis for cells:

The best performance I could get out of the circuit is a clock speed of 4.2 ns. This is fairly fast, but might not be fast enough for later uses.

The power of the module is as follows:
Total Internal Power: 0.05578210 mW 52.4784%
Total Switching Power: 0.05010155 mW 47.1343%
Total Leakage Power: 0.00041177 mW 0.3874%
Total Power: 0.10629542 mW
These are around the numbers we expected.

The area for this module is 668.141 um^2. This is also about what was expected.

Control Logic:

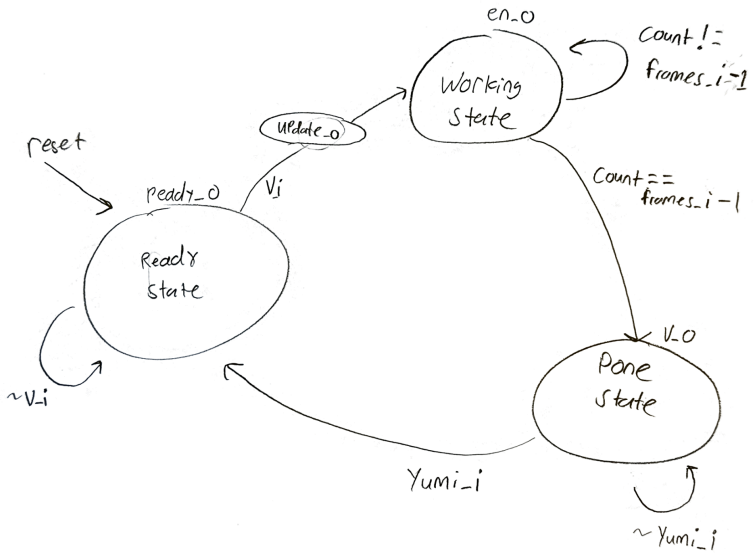


Figure 4: State Diagram of Control Module

PPA ANALYSIS:

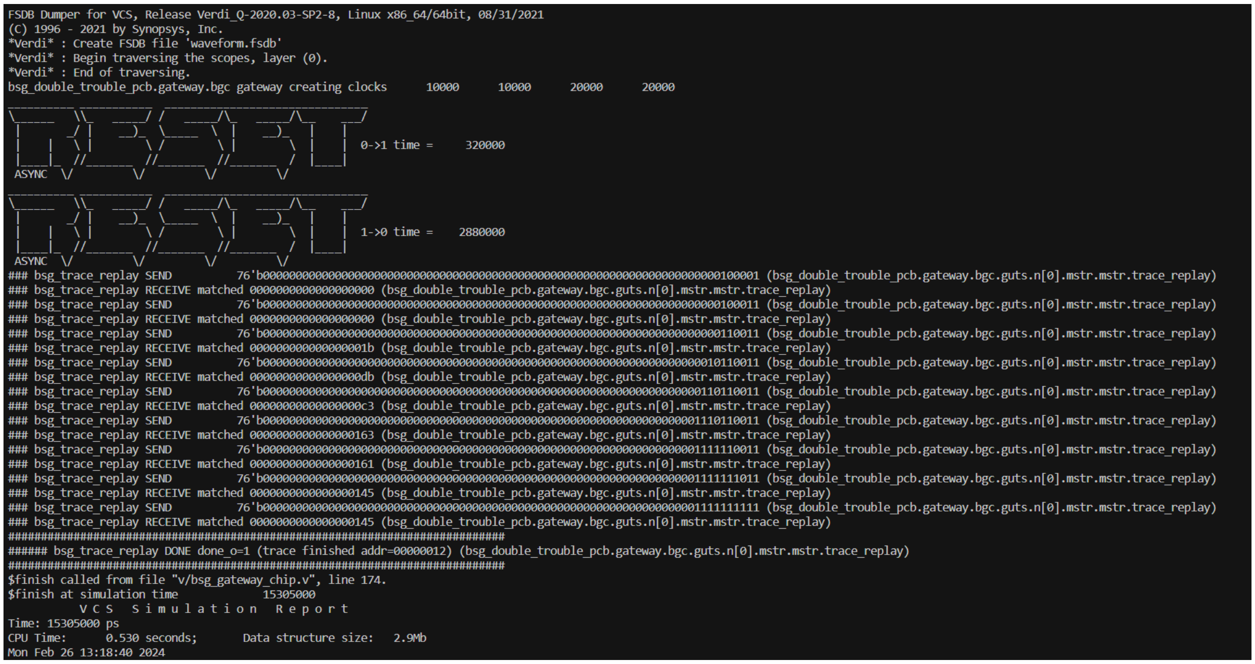
My design had a core clock speed of 4.8 ns. This is a relatively fast clock and is suitable for this size of a design. I may have to come back to this to further optimize at a later date

Our total design had an area of 73467.962 um^2. This suits the expectations for how large the design would be.

The power characteristics of the design are as follows:

Total Internal Power:	8.28142960 mW	69.6143%
Total Switching Power:	3.57247260 mW	30.0305%
Total Leakage Power:	0.04225598 mW	0.3552%
Total Power:	11.89615819 mW	

11.89 mW is higher than I would’ve expected for this sort of a design, but it will be readily apparent that the power is too high in a later phase. So, at this point this is fine.



```
FSDB Dumper for VCS, Release Verdi_Q-2020.03-SP2-8, Linux x86_64/64bit, 08/31/2021
(C) 1996 - 2021 by Synopsys, Inc.
*Verdi* : Create FSDB file 'waveform.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
bsg_double_trouble_pcb.gateway.bgc gateway creating clocks      10000      10000      20000      20000
__________ ___________  _____________________________
\______   \\_   _____/ /   _____/\_   _____/\__    ___/
 |       _/ |    __)_  \_____  \  |    __)_   |    |
 |    |   \ |        \ /        \ |        \  |    |   0->1 time =     320000
 |____|_  //_______  //_______  //_______  /  |____|
 ASYNC  \/         \/         \/         \/
__________ ___________  _____________________________
\______   \\_   _____/ /   _____/\_   _____/\__    ___/
 |       _/ |    __)_  \_____  \  |    __)_   |    |
 |    |   \ |        \ /        \ |        \  |    |   1->0 time =    2880000
 |____|_  //_______  //_______  //_______  /  |____|
 ASYNC  \/         \/         \/         \/
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000000000100001 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 000000000000000000 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000000000100011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 000000000000000000 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000000000110011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 00000000000000001b (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000000010110011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 0000000000000000db (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000000110110011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 0000000000000000c3 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000001110110011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 000000000000000163 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000001111110011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 000000000000000161 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000001111111011 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 000000000000000145 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay SEND         76'b0000000000000000000000000000000000000000000000000000000000000000001111111111 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
### bsg_trace_replay RECEIVE matched 000000000000000145 (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
###########################################################################
###### bsg_trace_replay DONE done_o=1 (trace finished addr=00000012) (bsg_double_trouble_pcb.gateway.bgc.guts.n[0].mstr.mstr.trace_replay)
###########################################################################
$finish called from file "v/bsg_gateway_chip.v", line 174.
$finish at simulation time            15305000
           V C S   S i m u l a t i o n   R e p o r t
Time: 15305000 ps
CPU Time:      0.530 seconds;       Data structure size:   2.9Mb
Mon Feb 26 13:18:40 2024
```

Figure 5: Traces for control passing

Total Design:

After many challenges I was successfully able to place and route the entirety of the game of life chip. In the following diagram pink is the BSG tag communication module, the green is all the instantiations of cells, and the blue is the cell controller. There are DRC errors located at the bottom of the chip that can be ignored. This is an example of the DRC system not realizing that the hanging connections are meant to be I/O ports.

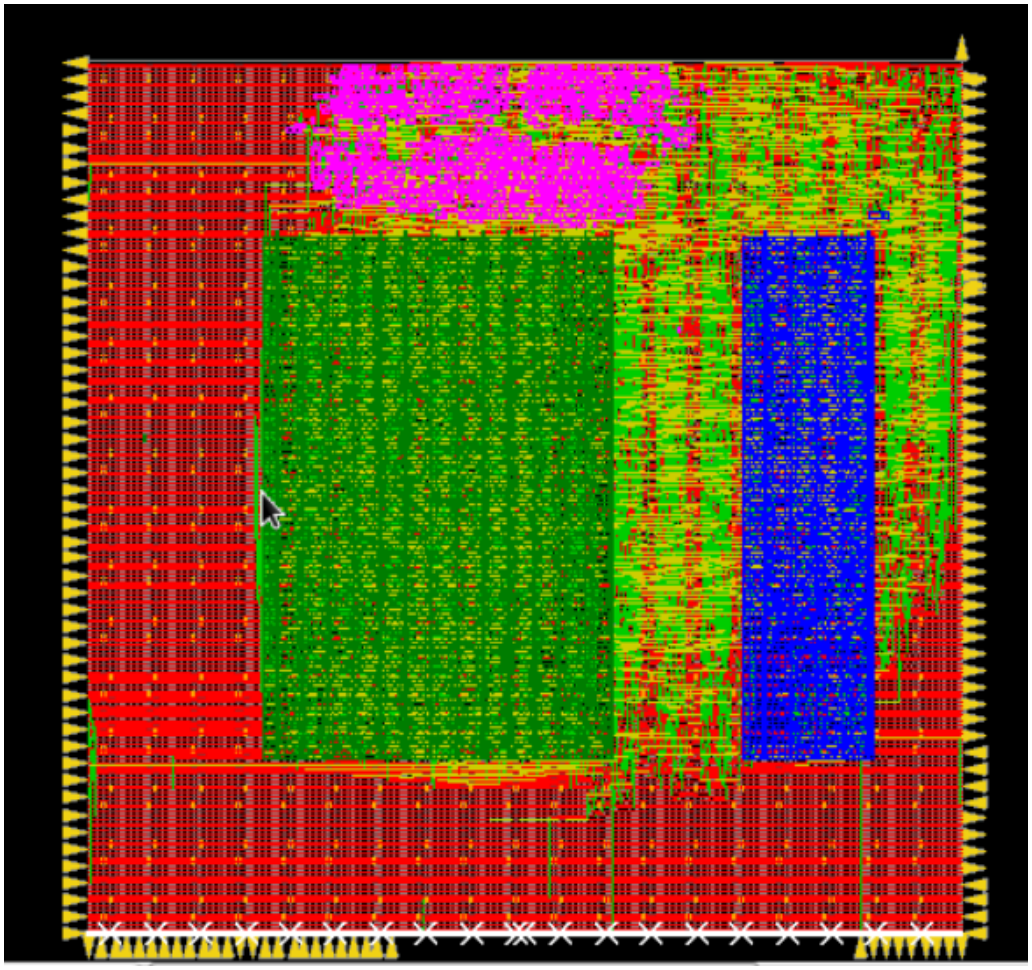


Figure 6: Complete place and routed game of life

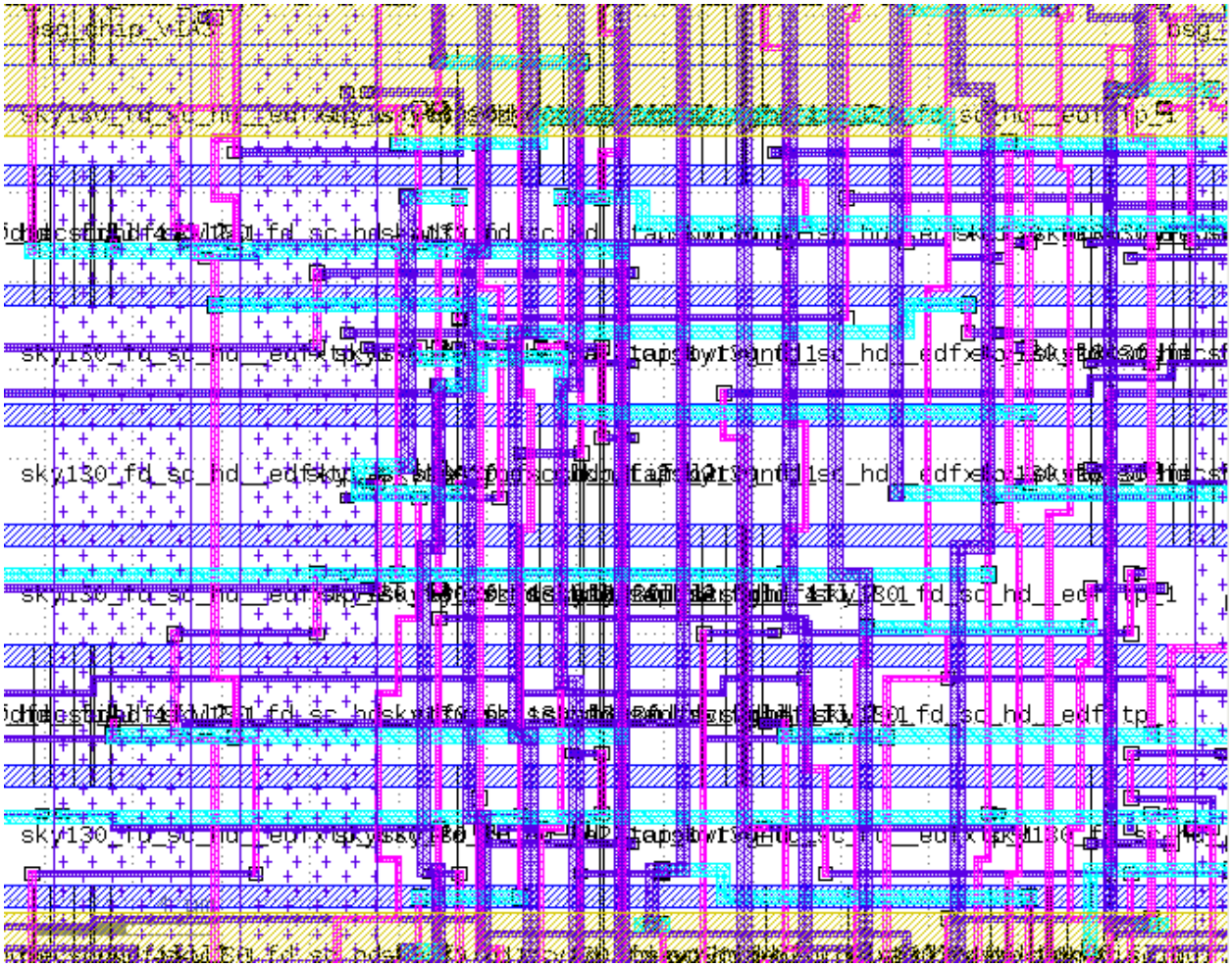


Figure 7: example of some extremely close routing and overlapping

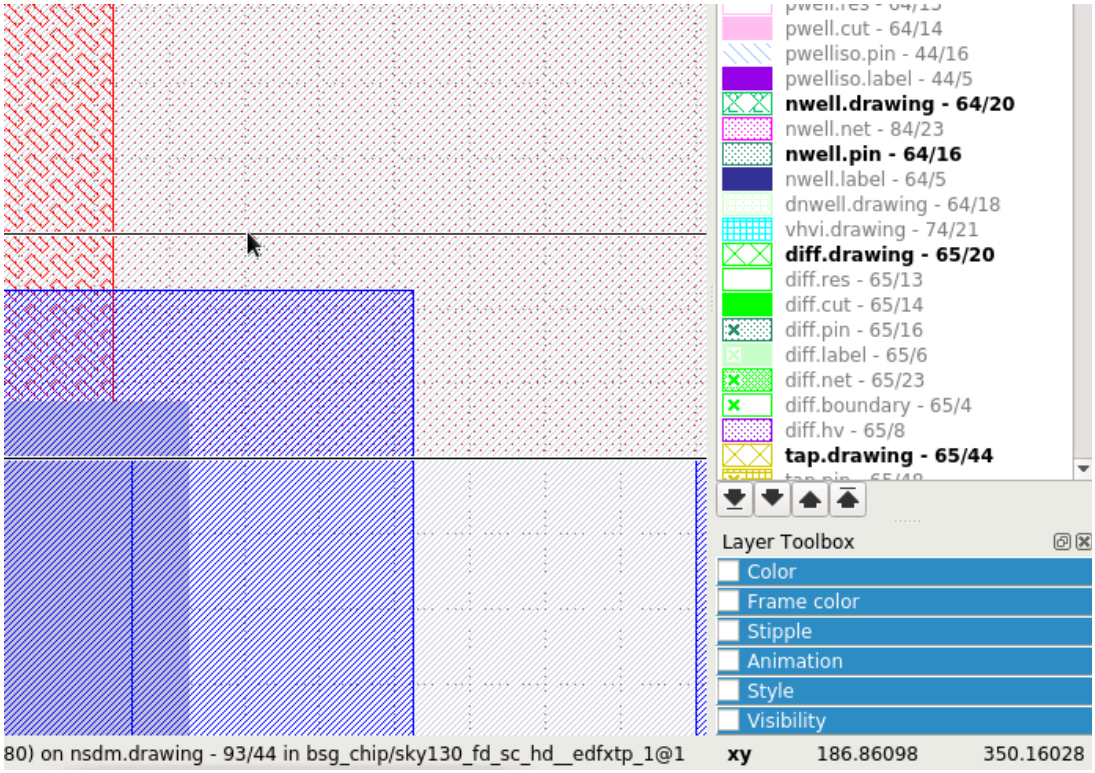


Figure 8: Example DRC errors on layout geometry due to overlapping