NAEEMUL HASSAN
ZACHARY KEARNEY

EEDG 6306: APPLICATION SPECIFIC INTEGRATED CIRCUIT DESIGN

FINAL PROJECT REPORT

DUE DATE: 11/30/2020

Contents

MSDAP Architecture & Specification

Architecture Overview

The MSDAP is comprised of seven sub blocks. One sub block, the control unit, is dedicated for control logic and MSDAP operation mode control. One sub block, the convolution unit, is where all the mathematical computation for the convolution occurs. Three sub blocks, rj, coefficient, and data memory, serve as memory for the MSDAP. One sub block, zero checker, provides information about input data to the control unit. One module, serial to parallel converter, handle processing of input signals.

- Control Unit
- Convolution Unit
- Rj Memory
- Coefficient Memory
- Data Memory
- Zero Checker
- Serial to Parallel Converter

Control Unit (control_unit)

The control unit controls the operational flow of the MSDAP. The control unit orchestrates chip initialization, processing input and output, transitions from working and sleeping states, and handles asynchronous resets. This module will be discussed in greater detail.

Convolution Unit (convolution_unit)

The convolution unit performs the convolution. The system clock drives the addition and shifting necessary to calculate the output.

Rj Memory (rj_mem)

The rj memory block serves as register memory for the rj values needed for the convolution. A single rj memory block consists of 16 8-bit registers.

Coefficient Memory (coeff_mem)

The coeff memory sub block serves as register memory for the coefficient values needed for the convolution. A single coefficient memory block consists of 512 9-bit registers.

Data Memory (data_mem)

The data memory sub block serves as register memory for the data input values being processed in real time. A single data memory block consists of 256 16-bit registers.

Zero Checker (zero_check)

The zero-check sub module checks if the input data is equivalent to zero. This sub block provides information to the control unit.

Serial to Parallel Converter (serial_to_parallel)

The serial to parallel sub block take serial input and converts the data to parallel so that it can be stored in memory.

MSDAP Algorithm Overview

The MSDAP algorithm is implemented using a finite impulse response (FIR) digital filter. The algorithm is computation heavy and follows the equation mentioned below:

$$y(n) = \sum_{k=0}^{N} h(k) \times x(n-k)$$

where *x(n)* and *y(n)* are input and output data while *h(k)* represent a set of filter coefficients with filter order *N*. However, to make the chip affordable, the goal is to make the design and hardware less complicated; hence instead of using multiplication as mentioned in the algorithm provided above, another method is used. Each *h(k)* can be broken down to power-of-two (POT) digits to give a well-rounded approximation. This basically means that instead of a floating-point unit in the hardware, all that is needed is a shifter which can shift as per the POT digits.

$$h(k) = 0.1172 \; can\; be\; approximated\; to\; be\; as\; h(k) = 2^{-3} - 2^{-7} + 2^{-16}$$

$$h(k)x(n-k) = 2^{-3}x(n-k) - 2^{-7}x(n-k) + 2^{-16}x(n-k)$$

As it can be seen from the above equation, that a 16-bit shifter maybe necessary in some situations to come up with high accuracy results. So, a further simplification of the above idea can be expressed which employs only a one-bit shifter.

$$y(n) = 2^{-1}(\ldots 2^{-1}(2^{-1}(2^{-1}u_1 + u_2) + u_3) + \cdots) + u_{16}$$

$$\text{where } u_j = x_j(1) + x_j(2) + \cdots + x_j(r_j) \qquad 1 \le j \le 16$$

An example of the above computation is as follows:

Left Channel First input **4BC8**

Step 1 (Add #1) = 00000000000000000000000000000000000000000000

Step 2 (Shift #1) = 00000000000000000000000000000000000000000000

Step 3 (Add #2) = 00000000000000000000000000000000000000000000

Step 4 (Shift #2) = 00000000000000000000000000000000000000000000

Step 5 (Add #3) = 00000000000000000000000000000000000000000000

Step 6 (Shift #3) = 00000000000000000000000000000000000000000000

Step 7 (Add #4) = 00000000000000000000000000000000000000000

Step 8 (Shift #4) = 00000000000000000000000000000000000000000

Step 9 (Add #5) = 11111111101101000011100000000000000000000

Step 10 (Shift #5) = 11111111110110100001110000000000000000000

Step 11 (Add #6) = 11111111110110100001110000000000000000000

Step 12 (Shift #6) = 11111111111011010000111000000000000000000

Step 13 (Add #7) = 11111111111011010000111000000000000000000

Step 14 (Shift #7) = 11111111111101101000011100000000000000000

Step 15 (Add #8) = 11111111111101101000011100000000000000000

Step 16 (Shift #8) = 11111111111110110100001110000000000000000

Step 17 (Add #9) = 11111111111110110100001110000000000000000

Step 18 (Shift #9) = 11111111111111011010000111000000000000000

Step 19 (Add #10) = 11111111111111011010000111000000000000000

Step 20 (Shift #10) = 11111111111111101101000011100000000000000

Step 21 (Add #11) = 11111111111111101101000011100000000000000

Step 22 (Shift #11) = 11111111111111110110100001110000000000000

Step 23 (Add #12) = 11111111111111110110100001110000000000000

Step 24 (Shift #12) = 11111111111111111011010000111000000000000

Step 25 (Add #13) = 11111111111111111011010000111000000000000

Step 26 (Shift #13) = 11111111111111111101101000011100000000000

Step 27 (Add #14) = 11111111111111111101101000011100000000000

Step 28 (Shift #14) = 11111111111111111110110100001110000000000

Step 29 (Add #15) = 11111111111111111110110100000111000000000

Step 30 (Shift #15) = 11111111111111111111011010000111000000000

Step 31 (Add #16) = 11111111111111111111011010000111000000000

Step 32 (Shift #16) = 11111111111111111111011010000111000000000

Output: **FFFFFB4380 hex**

Pin Settings

Figure 1 shows the system connection diagram with the controller and MSDAP.

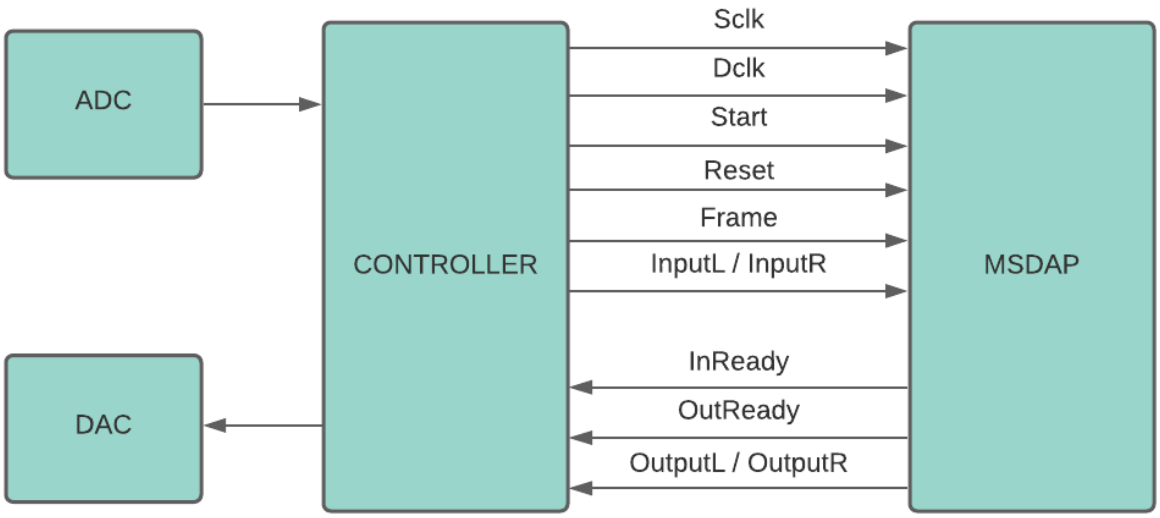

Figure 1. System Connection Diagram

Input Pins

Clock Pins

Sclk

System clock provides timing reference for control signals, internal operations, and output of data. Uses a modifiable frequency of **80.64 MHz**.

Dclk

Data clock that provides timing reference for reading in Rjs, coefficients, and input samples. Uses a fixed frequency of 768 kHz.

Asynchronous Pins

Start

Asynchronous signal that begins chip initialization when set high. After initial rising edge start is no longer monitored.

Reset

Asynchronous signal that resets MSDAP chip when a rising edge is detected. To allow enough time for the chip to reset, the reset signal may only go high in the first half of a frame period. More specifically, in a 16 dclk cycle period, reset may only go high during cycles 1-8. This is elaborated in Figure 4.

Data Pins

Frame

Used for aligning serial Rjs, coefficients, and input samples. For input values, frame is high for one Dclk cycle indicating the first bit of Rjs, coefficients, and input values. For output values, frame is set high for one Sclk cycle

InputL

Used for Rjd, coefficient, and input sample data for the left channel. All data on this pin is transmitted serially and captured on the negative edge of dclk. Bit 0, the sign bit, is transmitted first and Bit 15, the LSB, is transmitted last.

InputR

Used for Rj, coefficient, and input sample data for the right channel. All data on this pin is transmitted serially and captured on the falling edge of dclk. Bit 0, the sign bit, is transmitted first and Bit 15, the LSB, is transmitted last.

Output Pins

Control Signal Pins

InReady

Informs the controller to transmit Rj, coefficient, or input data to the MSDAP when high. If low, the MSDAP is not ready to receive data. InReady is updated on rising edge of Sclk.

OutReady

Informs the controller output data has been calculated and is being transmitted from the MSDAP. OutReady is set high with the rising edge of frame.

Data Pins

OutputL (40-bit bus)

Used for output data for the left channel. Data is transmitted over a 40-pin bus. Output is aligned with the rising edge of Sclk and Frame.

OutputR (40-bit bus)

Used for output data for the right channel. Data is transmitted over a 40-pin bus. Output is aligned with the rising edge of Sclk and Frame.

Figure 2 shows the waveform format of reading in input and Figure 3 show the waveform format of outputting data.

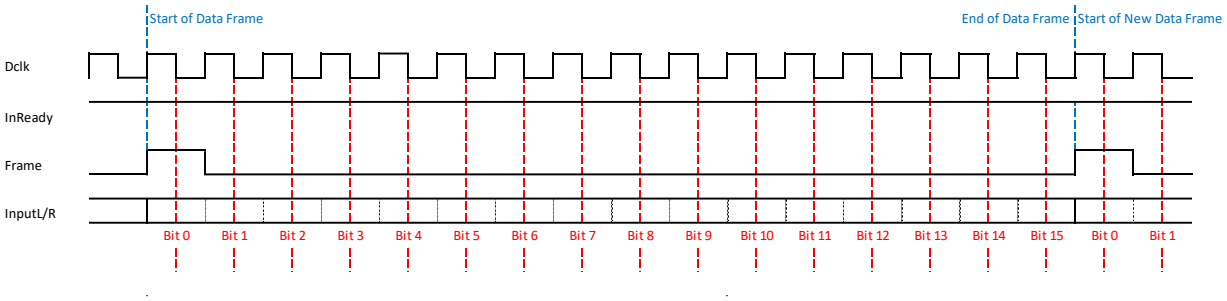

Figure 2. Receiving Input Format

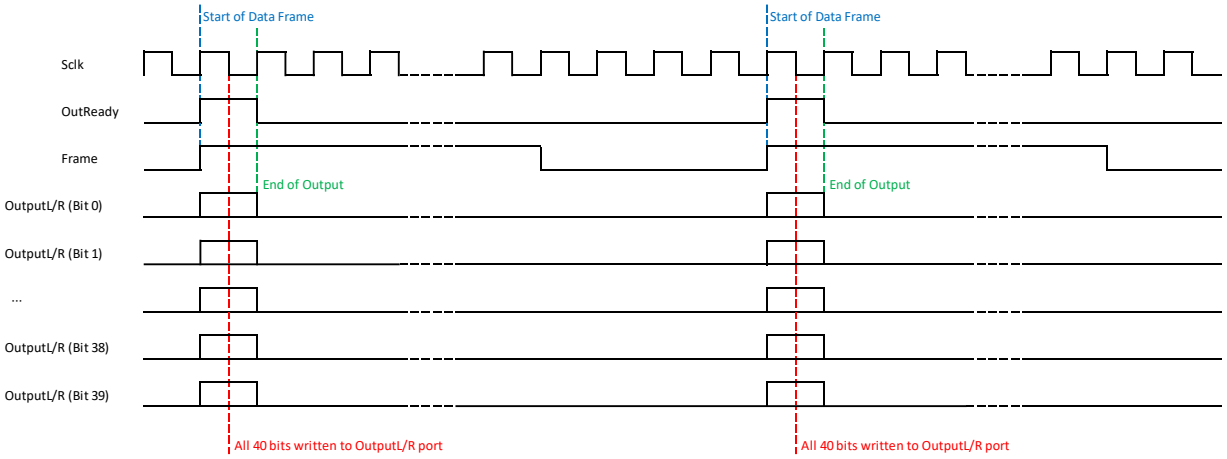

Figure 3. Transmitting Output Format

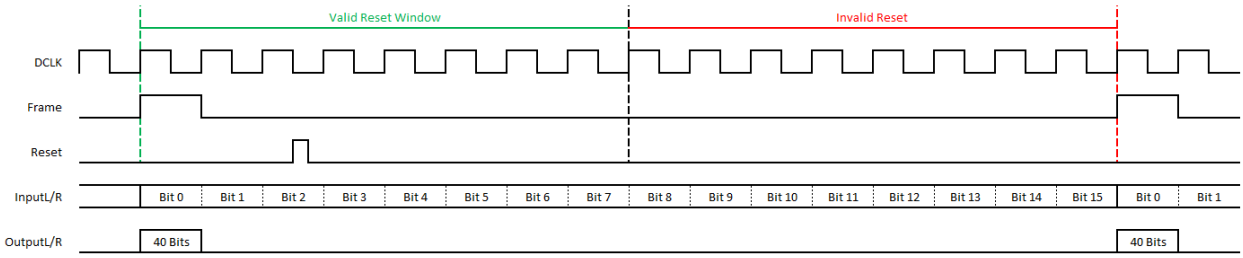

Figure 4. Reset Window

Signal Formats

All the input data x(n), coefficients and r_j values are transmitted to the chip as 16-bit, 2's complement values. Initially zeroes are padded to the r_j and coefficients to extend the signal format size to 16-bits. The output data is transmitted out of the chip as 40-bit data stream. Figure 2 shows examples of each data format.

R_j format:

Lower half of the 16-bit sized data, i.e. the lower 8-bits, holds the actual data of the R_j while the upper half of the data includes the padded zeroes.

Coefficient format:

Lower half of the 16-bit sized data, i.e. the lower 8-bits, holds the actual data of the coefficient and the 7^{th} bit holds the sign bit while the upper half of the data includes the padded zeroes.

Input data format:

Input data is 16-bit 2's complement values with the MSB being the sign bit.

Output data format:

Output data is 40-bit sized 2's complement values with the MSB being the sign bit.

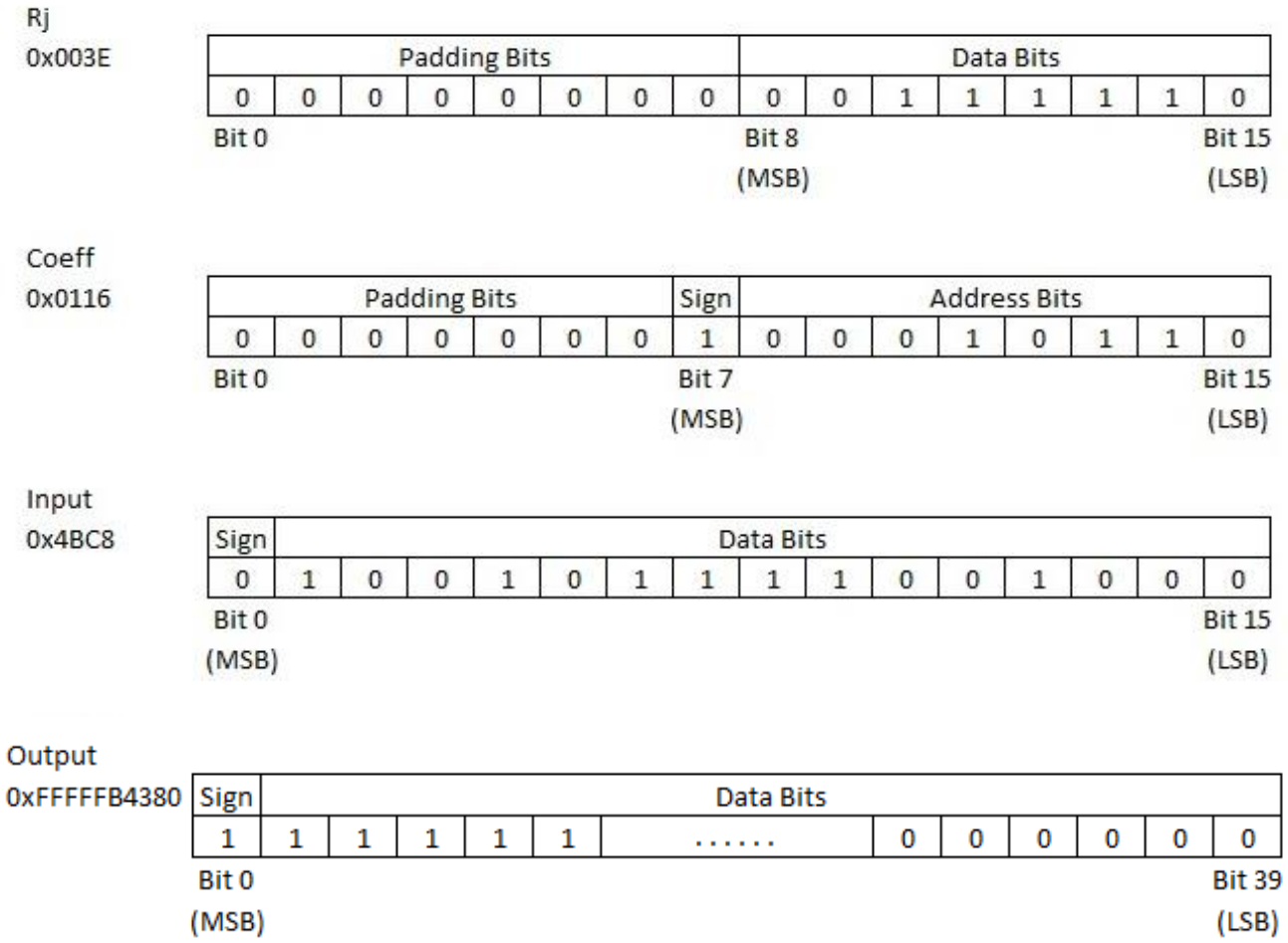

Figure 5. Data Format Examples

Operation Modes

State 0, Initialization

When Start has a rising edge the chip begins initialization. All memories and registers are cleared, control signals are set to default values. The MSDAP then transitions to State 1.

State 1, Waiting for Rj values

InReady is set to high and the MSDAP is waiting for the first Rj value. When Frame goes high, indicating the first Rj is being transmitted, the MSDAP transitions to State 2.

State 2, Reading Rj values

The MSDAP chip reads in all the Rj values. InReady remains high during the entirety of State 2. The MSDAP transitions to State 3 immediately after the last Rj value is loaded.

State 3, Waiting for coefficients

InReady is already set high from State 2. The MSDAP is waiting for the first coefficient value. When Frame goes high, indicating the first coefficient is being transmitted, the MSDAP transitions to State 4.

State 4, Reading Coefficients

The MSDAP chip reads in all the coefficient values. InReady remains high during the entirety of State 4. The MSDAP transitions to state 5 immediately after the last coefficient value is loaded in.

State 5, Waiting for channel data

InReady is set high upon transition to this state. The MSDAP is waiting for the first input data value. If Frame goes high, indicating the first input value is being transmitted, the MSDAP transitions to State 6. If a rising edge of the Reset signal is detected, the MSDAP will transition to State 7.

State 6, Working

The MSDAP repeatedly reads in inputs, performs convolution, and writes output to pins. InReady is high for the entirety of this state. If 800 consecutive zero input samples appear on either channel, the chip transitions to State 8. If a rising edge of the Reset signal is detected, the MSDAP will transition to State 7.

State 7, Clearing

Upon transitioning to the reset state all input/output streams, input memories, and registers are cleared. Rj and coefficient data is retained during this state. Once this process is completed the MSDAP will immediately transition to State 5. If a rising edge of the Reset signal is detected, the process of clearing streams, memories, and registers will restart.

State 8, Sleeping

This state puts the MSDAP in sleep mode where the convolution is not performed. InReady is set to high so a non-zero input on either channel will transition the MSDAP back to State 6. If a rising edge of the Reset signal is detected, the MSDAP will transition to State 7.

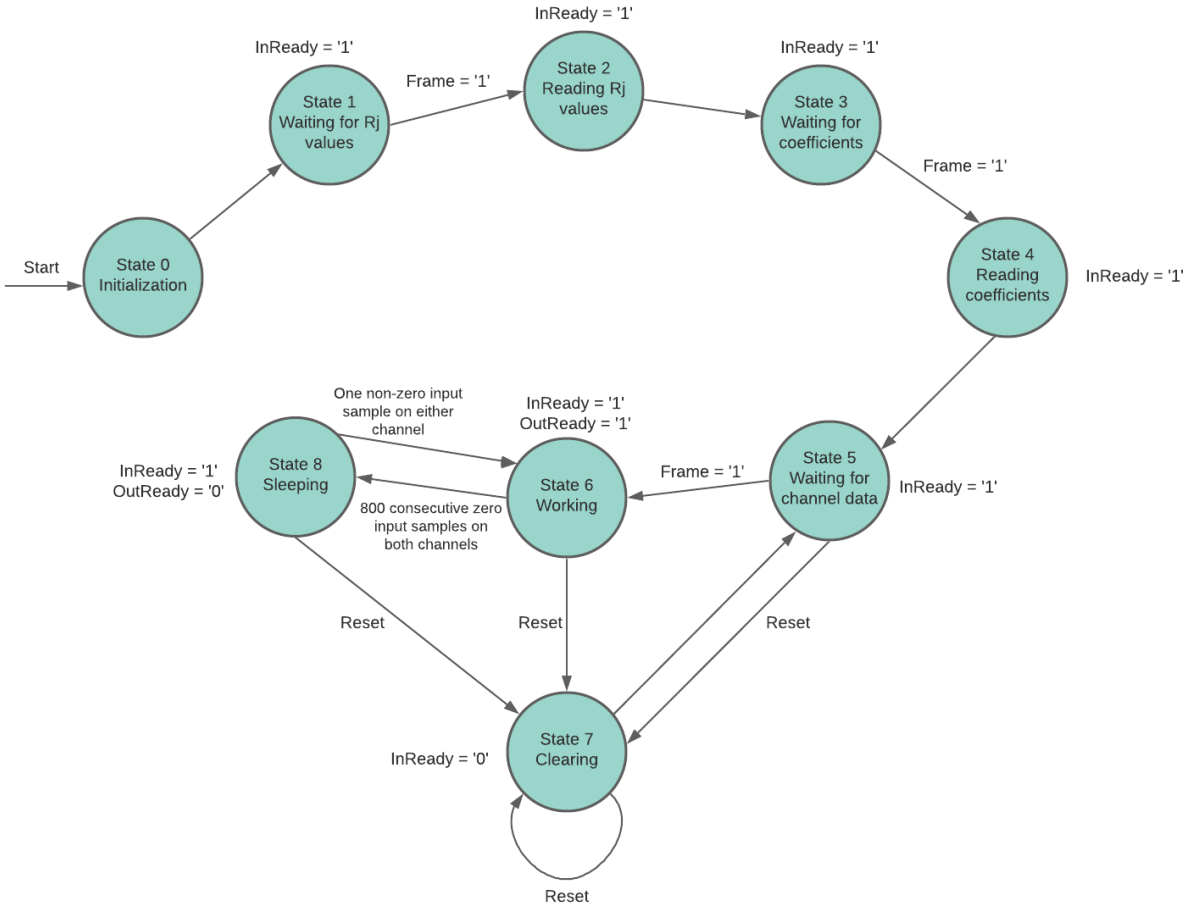

Figure 6. MSDAP Operation Modes FSM Diagram

Sub Block Interaction

The interaction of the sub blocks is shown in Figure 7.

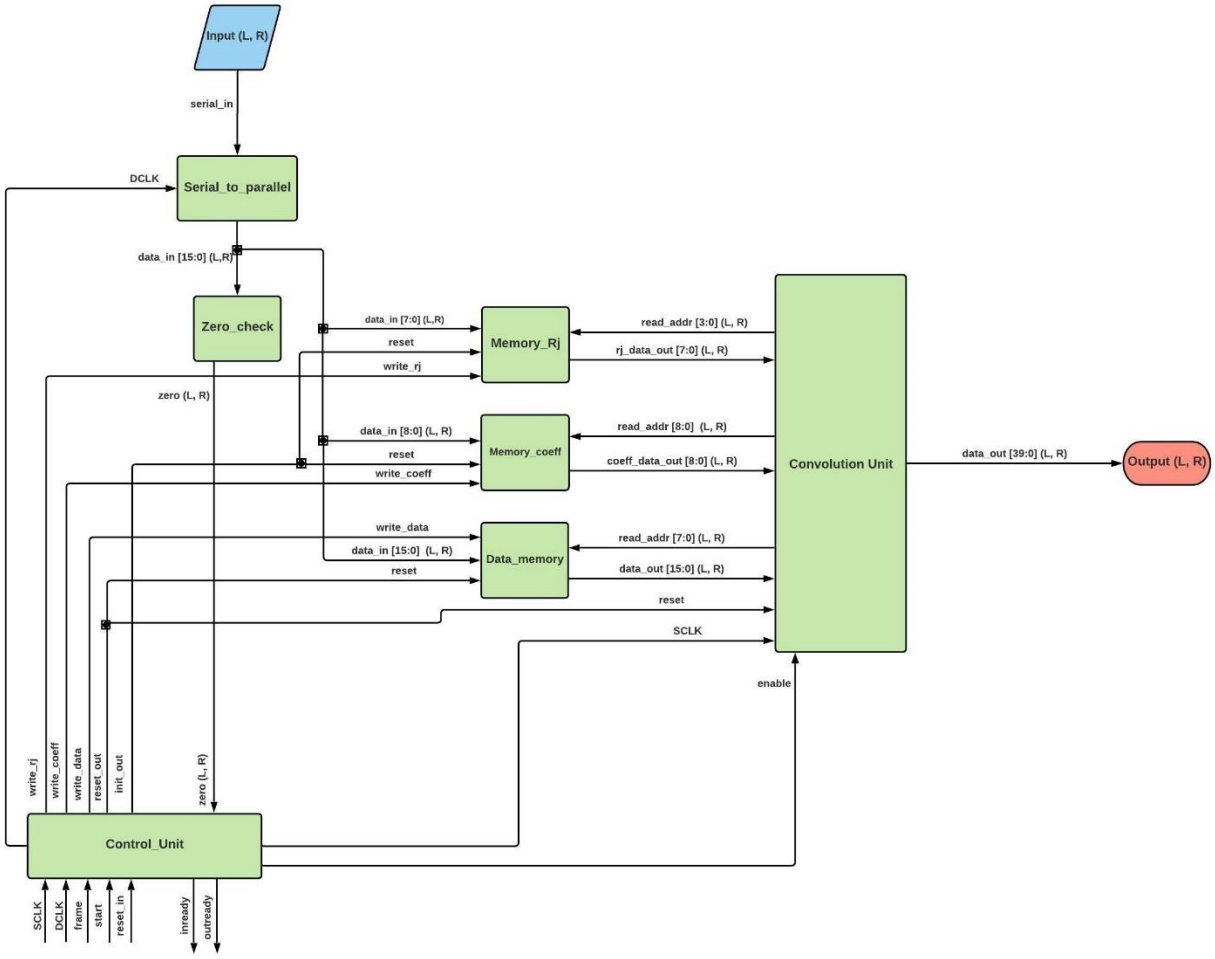

Figure 7. MSDAP Architecture

The control unit contains the initialization and reset signals that initialize the various memory units and convolution units. The reset signal will reset the convolution and data memory only. The coefficient and rj memory will not be clear at a reset. The control unit controls the write signals for all three memory unit types. The zero checker units will feed information into the control unit. An enable signal will enable the convolution unit to perform a convolution.

The serial to parallel unit will take the serial input and feed it to the zero checker and all three different memory units.

The convolution unit interacts with all three memory types to fetch rjs, coefficients, and input data from the memory sub blocks. The convolution unit sends the address of the requested values to each memory unit. The memory unit will the send the values back to the convolution unit.

Block Specification

MSDAP (msdap)

Input Ports:

- sclk – system clock that drives computation and data output
- dclk – data clock that drives data input
- start – initialization signal
- frame – signal used for aligning first value of serial input
- reset – signal that tells the MSDAP to clear memories, registers, and restart computation
- inputL – 16-bit bus for reading left channel data
- inputR – 16-bit bus for reading right channel data

Output Ports:

- inready – indicates the MSDAP is ready to receive data
- outready – indicates the MSDAP is ready to transmit data
- outputL – 40-bit output bus for sending left channel data
- outputR – 40-bit output bus for sending right channel data

Description:

The MSDAP module is the top-level module. Within the MSDAP module is the convolution unit, control unit, memories, and other supporting functional blocks. To illustrate all the modules, the flow of input data will be explained. When serial data comes in over the inputL/R ports, the data will first enter the serial to parallel converter. There are two of these units, one for the inputL and the other for InputR. Three types of data come in over the serial in: Rj values, coefficient values, and input data. Once the serial data has been converted into parallel data the data is sent to the zero-check module and the memory modules. Depending on the type of data, the controller will determine which memory the data goes to. After data has been loaded in, data gets fed to the convolution unit, where the convolution occurs and is then output from the MSDAP module at the appropriate time.

Control Unit (control_unit)

Input Ports:

- sclk – system clock that drives computation and data output
- dclk – data clock that drives data input
- start – initialization signal
- reset_in – signal that tells the MSDAP to clear memories, registers, and restart computation
- frame – signal used for aligning first value of serial input

- zeroL – signal that indicates the input value on the left channel is zero
- zeroR – signal that indicate the input value on the right channel is zero

Output Ports:

- write_rj – signal that tells the Rj memories to write the current input value
- write_coeff – signal that tells the coefficient memories to write the current input value
- write_data – signal that tells the data memories to write the current input value
- init_out – internal initialization signal that is sent to functional blocks
- reset_out – internal reset that is sent to functional blocks
- enable_cu – signal that enables the convolution unit to begin computation
- outready – indicates the MSDAP is ready to transmit data
- inready – indicates the MSDAP is ready to receive data

Description:

The control unit implements the operation modes of the MSDAP. The states have been covered extensively in the specification. Figure 6 shows all the states of the MSDAP. The waveforms of these Transitions are shown below. The state of the MSDAP is the light blue signal.

State Transitions Waveforms

State 0 Transition – When the Start signal has a rising edge, the MSDAP transitions to State 0 for initialization as shown in Figure 9.

State 0 to State 1 Transition – After initialization the MSDAP transitions to state 1. Then, on the next rising edge of sclk, inready goes high.

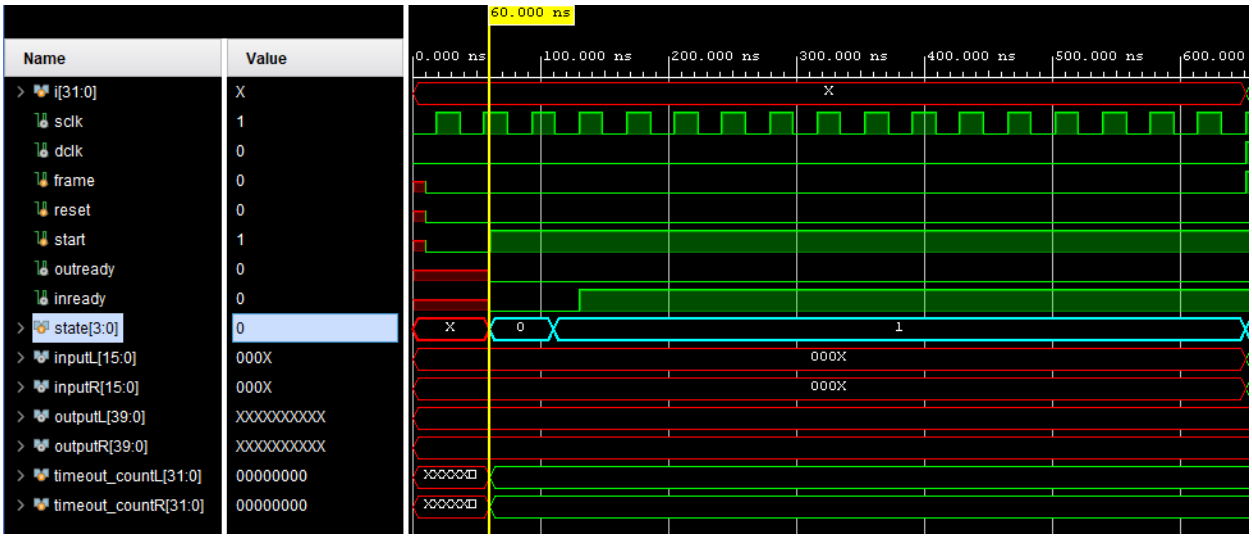

Figure 8. State 0 Transition, State 0 to State 1 Transition

State 1 to State 2 Transition – At the first rising edge of Frame the MSDAP transitions to State 2 as shown in Figure 10. Inready remains high in this state. Inready remains high throughout State 2.

Figure 9. State 1 to State 2 Transition

State 2 to State 3 Transition – Once all the Rj values are read in, the MSDAP transitions from State 2 to State 3 as shown in Figure 11. Inready remains high throughout State 3.

State 3 to State 4 Transition – At the first rising edge of Frame the MSDAP transitions to State 4 as shown in Figure 11. Inready remains high throughout State 4.

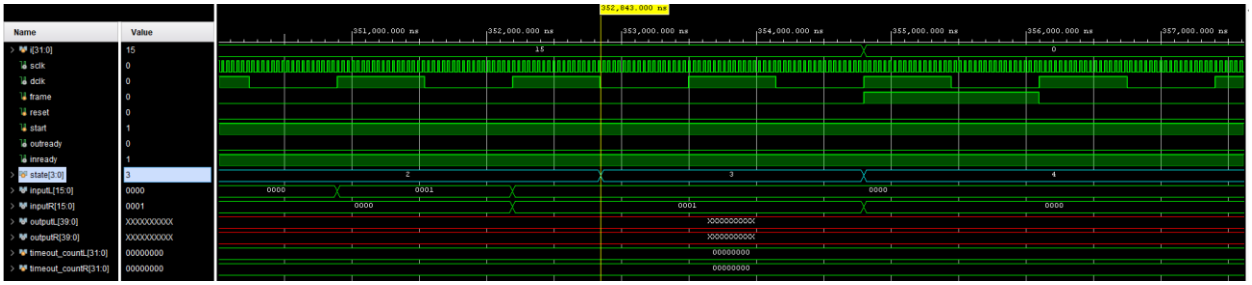

Figure 10. State 2 to State 3 Transition, State 3 to State 4 Transition

State 4 to State 5 Transition – Once all the coefficients are read in, the MSDAP transitions from State 4 to State 5 as shown in Figure 12. Inready remains high throughout State 5.

State 5 to State 6 Transition – At the first rising edge of Frame the MSDAP transitions to State 6 as shown in Figure 12. Inready remains high throughout State 6. When the output is calculated, outready goes high on the next rising edge of sclk as shown in Figure 13. Outready is the small spike to the right of the yellow marker. Inready remains high throughout State 6.

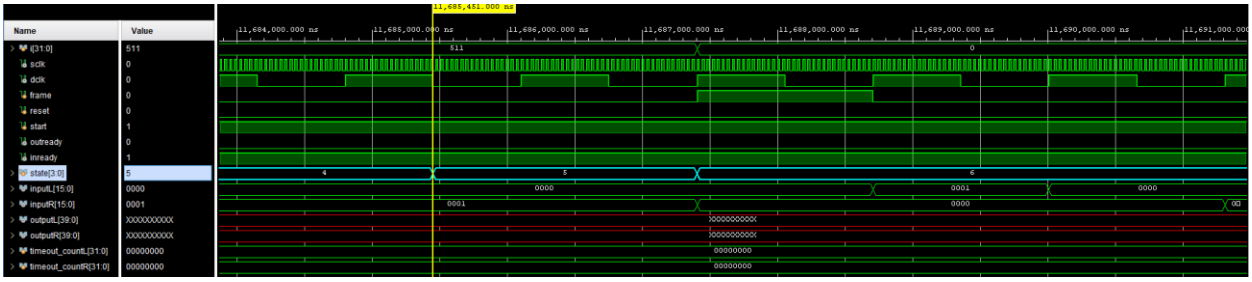

Figure 11. State 4 to State 5 Transition, State 5 to State 6 Transition

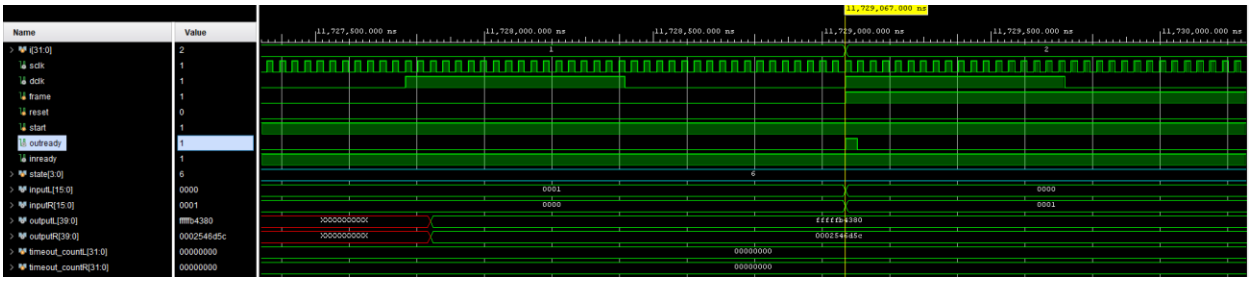

Figure 12. Outready going high once output calculated

State 6 to State 8 Transistion – After 800 consecutive 0 inputs on both channels the MSDAP transisitons to State 8 as shown in Figure 14. The bottom two signals timeout_count(L,R) read 320 hex (800 decimal) and 640 hex (1600 decimal). In State 8 inready remains high in State 8, and outready will remain low.

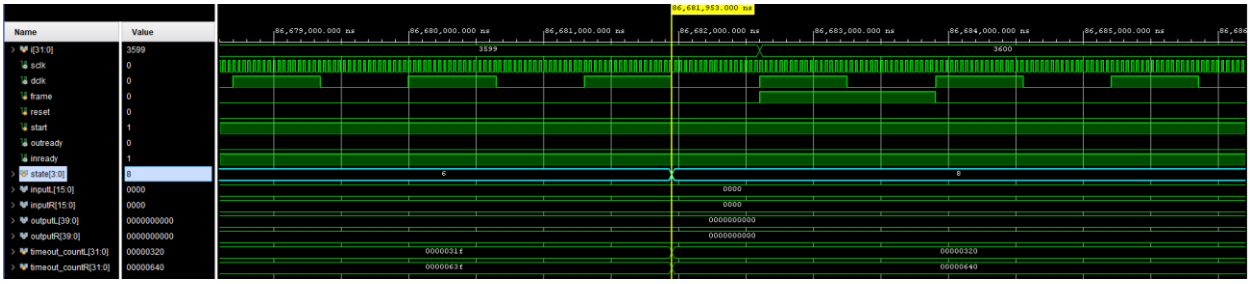

Figure 13. State 6 to State 8 Transition

State 8 to State 6 Transistion – Once a non-zero input appears on either channel, the MSDAP transistions back to State 6 as shown in Figure 15.

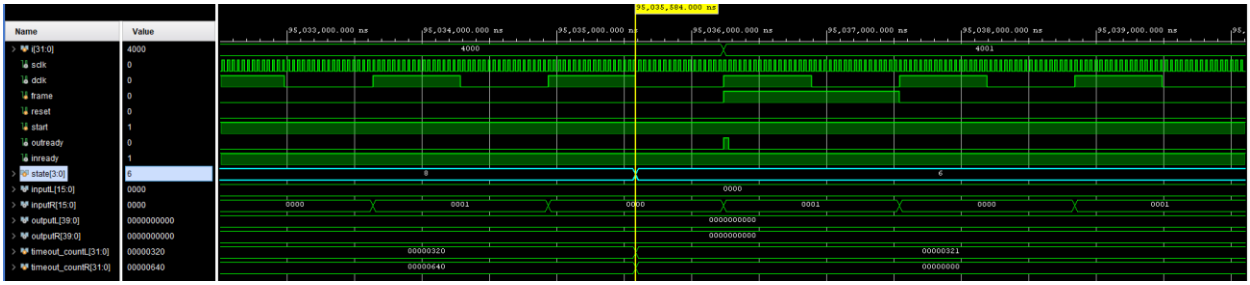

Figure 14. State 8 to State 6 Transition

State 6 to State 7 Transisiton – If a rising edge of reset is detected during State 6, the MSDAP will transisiton to State 7 as shown in Figure 16. In State 7 outready will remain low.

State 7 to State 5 Transistion – Once the reset processes is completed, the MSDAP will transistion to State 5 as shown in Figure 16. Since in simulation these events happen instatenously, an artificial delay time has been injected to simulate the clearing of registers.

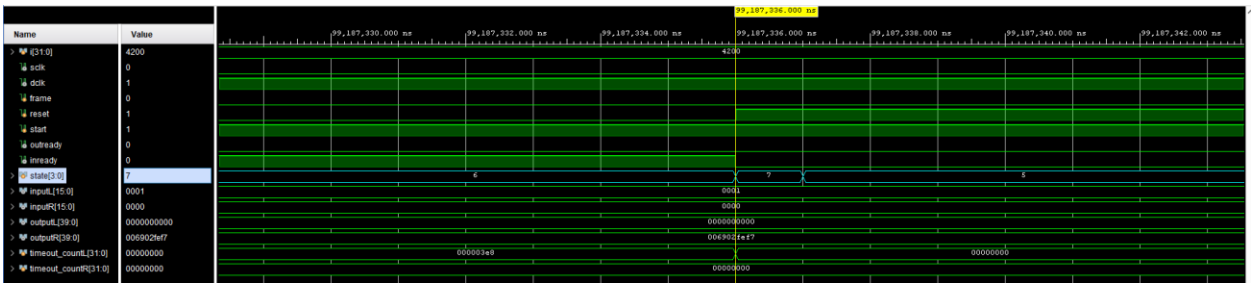

Figure 15. State 6 to State 7 Transition, State 7 to State 5 Transition

State 8 to State 7 Transistion – If a rising edge of reset is detected during State 8, the MSDAP will transisiton to State 7 as shown in Figure 17. In State 7 outready will remain low.

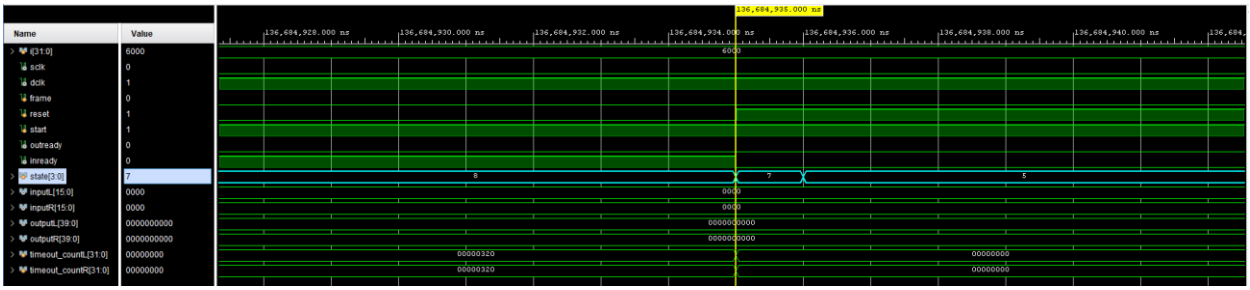

Figure 16. State 8 to State 7 Transition

State 7 to State 7 – If a rising edge of reset is detected while already reseting, the MSDAP will restart the reset process as shown in Figure 18.

State 5 to State 7 - If a rising edge of reset is detected during State 5, the MSDAP will transisiton to State 7 as shown in Figure 18.

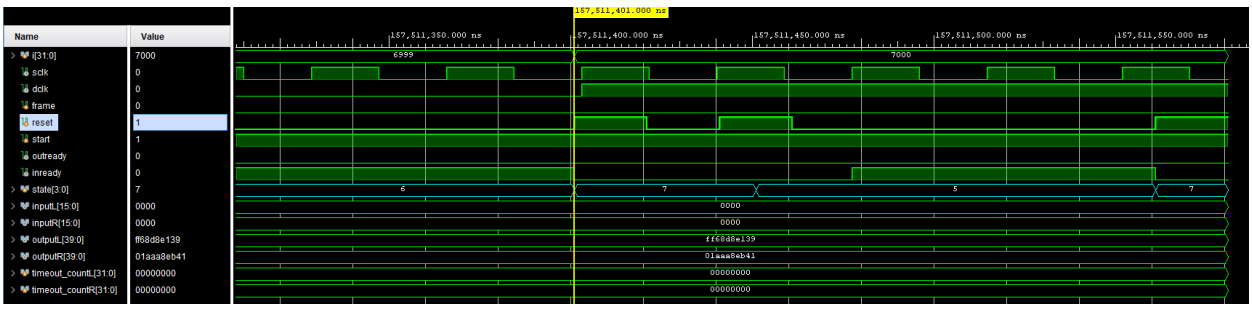

Figure 17. State 7 to State 7 Transition, State 5 to State 7 Transition

Convolution Unit (CVU)

The convolution unit (CVU) is where the computation takes place. The two main operations that occur are addition/subtraction and shifting. These operations are driven by the system clock. The convolution unit is a finite state machine. The states are listed below and shown in Figure 18.

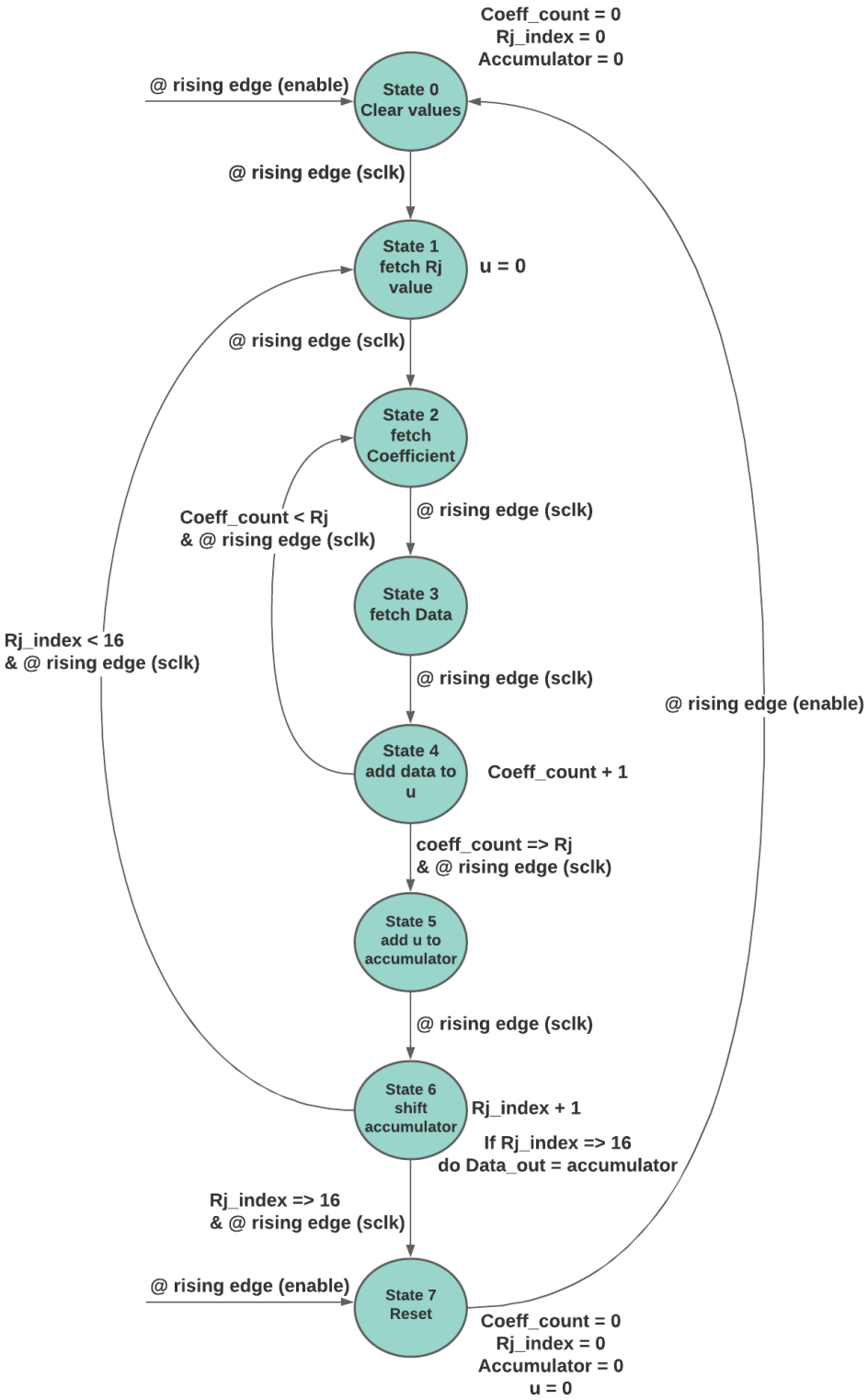

Figure 18. CVU FSM

The CVU begins in state 7. When a rising edge of enable is detect the CVU will transition to State 0.

State 0 – In this state the CVU zeros out memories and registers in preparation to run a computation. This includes clearing the accumulator register and resetting the coefficient count and Rj index to 0. At the next positive edge of the system clock, the CVU transitions to State 1.

State 1 – In this state the CVU fetches a Rj value from memory. If the CVU just transitioned from the initialization state, the Rj address is 0, as per Rj index. This state starts the calculation of a u value. The u value register is cleared in this state. Once the Rj is fetched and u value cleared, the CVU transitions to State 2 at the next rising edge of sclk.

State2 – In this state the next coefficient is fetched from memory. Once the coefficient is loaded in, the CVU transitions to State 3 at the next rising edge of sclk.

State 3 – In this state input data is fetched from memory. The coefficient from State 1 determines which memory location is fetched. Once the input data is loaded, the CVU transitions to state 4 at the next rising edge of sclk.

State 4 – In this state the data from State 3 is added to/subtracted from the u value register. State three is the main operation in calculating each individual u value. State 4 will use the coefficient from State 2 to determine whether to added or subtract the input data. The coefficient count is incremented after this step. If the coefficient count is less than the Rj value, then on the next rising edge of sclk, the CVU transitions back to State 2. If the coefficient count is greater or equal to the Rj value, then on the next rising edge of sclk, the CVU transitions to State 5.

State 5 – In this state the u value register is added to the accumulator. After this addition, on the next rising edge of sclk, the CVU transitions to state 6.

State 6 – In this state the accumulator register is right shifted once and the Rj index is incremented. If Rj index is less than 16, then the CVU transitions back to State 1 at the next rising edge of sclk. If Rj index is greater than or equal to 16, the CVU will transition to State 7 on the next rising edge of sclk.

State 7 – In this state the CVU clears all memories and registers. This state also serves as a standby state before and after computation has been completed.

If at any point in the CVU's operation, a rising edge of the reset signal is detected, the CVU will immediately transition to State 7.

RJ Memory (rj_mem)

Input Ports:

- data_in – 8-bit input bus for Rj value
- write – signal to tell the memory when to write a value
- reset – signal that clears the internal memory
- read_addr – variable width bus that tells the memory which value to output

Output Port:

- data_out – 8-bit bus that outputs the value at read_addr

Description:

This module consist of 16 memory locations to store the rj values used int the convolution computation. On a rising edge of the write signal, the value on data_in is captured and stored in the memory. The data is written to memory sequentially starting at address 0 all the way to location 15. There is a Rj memory module for both the left and right channel.

Coefficient Memory (coeff_mem)

Input Ports:

- data_in – 9-bit input bus for coefficient value
- write – signal to tell the memory when to write a value
- reset – signal that clears the internal memory
- read_addr – variable width bus that tells the memory which value to output

Output Port:

data_out – 9-bit bus that outputs the value at read_addr

Description:

This module consist of 512 memory locations to store the coefficient values used int the convolution computation. On a rising edge of the write signal, the value on data_in is captured and stored in the memory. The data is written to memory sequentially starting at address 0 all the way to location 511. There is a coefficient memory module for both the left and right channel.

Data Memory (data_mem)

Input Ports:

- data_in – 16-bit bus for input value
- read_addr – 8-bt bus that specifies the address to read from
- reset – signal that clears out the internal memory
- write – signal to tell the memory module to write value on data_in port

Output Port:

- data_out – 16-bit output bus that outputs value at read_addr

Description:

This module consists of 256 memory locations to store the input data used in the convolution computation. The 256 locations are a shift buffer with the oldest value being at the tail and the newest at the head. On a rising edge of the write signal, the value at the tail is expelled, the data is shifted towards the tail, and a new value is placed at the head. If the reset signal goes high, all 256 memory locations are zero out. The value of read_addr determines which memory location is output. Whenever read_addr is changed, the value on data_out changes. There is a data memory module for both the left and right channel.

Zero Checker (zero_check)

Input Ports:

- data_in – 16-bit bus for input data

Output Ports:

- zero – indicates if the input value is equivalent to zero

Description:

This module checks if the input value, data_in, is equivalent to zero. If the value is zero, the zero-output port is high, otherwise it is low. There is a zero checker module for both the left and right channels.

Serial to Parallel Converter (serial_to_parallel)

Input Ports:

- serial_in – port that reads serial data
- clk – clock that controls the capture of bits

Output Port:

- parallel_out – 16-bit bus that outputs converted data

Description:

This module takes serial input and converts the data to parallel. The module consists of a 16-bit wide register constantly output its value. At every falling edge of the clock, the register left shifts, and the input bit is captured and placed in the LSB of the register. There is a serial to parallel converter for both the left and right channel.

Test Bench Modules

The following modules clock generator, hold low, and parallel to serial are not used in the MSDAP module, but a necessary for implementing the testbench.

Clock Generator (clk_gen)

Output Port:

- clk – generated clock signal

Description:

A simple module that generates a clock signal. The module has the option to vary frequency.

Hold Low (hold_low)

Output Port:

- low – 15-bit bus used to hold values low

Description:

A simple module that holds the connected wires low. This is useful for the inputL and inputR bits 15-1 since the entirety of the port is not used due to the serial input specification.

Parallel to Serial (parallel_to_serial)

Input Ports:

- parallel_in – 16-bit bus for input value to be converted to serial
- clk – clock that controls the serial output
- frame – indicates when the first bit of the serial out is written
- enable – controls when the module is active

Output Port:

- serial_out – port were converted value is written to in serial format

Description:

This module converts data from parallel to serial. The module must have enable held high to function, otherwise the module stays in a standby state. The frame signal indicates when the conversion begins, the rising edge of the clock writes a single bit to the output port. This module is used to convert the values read from the input files and converting them to serial for the MSDAP.

RTL Verification

Synopsys Design Vision was used to synthesize RTL verilog code. The MSDAP RTL code, in its entirety, is shown in Appendix A1. The testbench code used for tests is shown in Appendix A2. Design Vision produced high-level block diagrams of each sub block and the top level MSDAP module. The associated SDC output is shown. The synthesized circuit schematics were also generated for all MSDAP blocks. Table 1 shows the figures for each block.

Table 1. Block Figures

Block	High Level	Circuit Schematic
MSDAP	Figure 19	Figure 20
Control Unit	Figure 21	Figure 22
Convolution Unit	Figure 23	Figure 24
Rj Memory	Figure 25	Figure 26
Coefficient Memory	Figure 27	Figure 28
Data Memory	Figure 29	Figure 30
Zero Checker	Figure 31	Figure 32
Serial to Parallel Converter	Figure 33	Figure 34

Figure 19. MSDAP Block

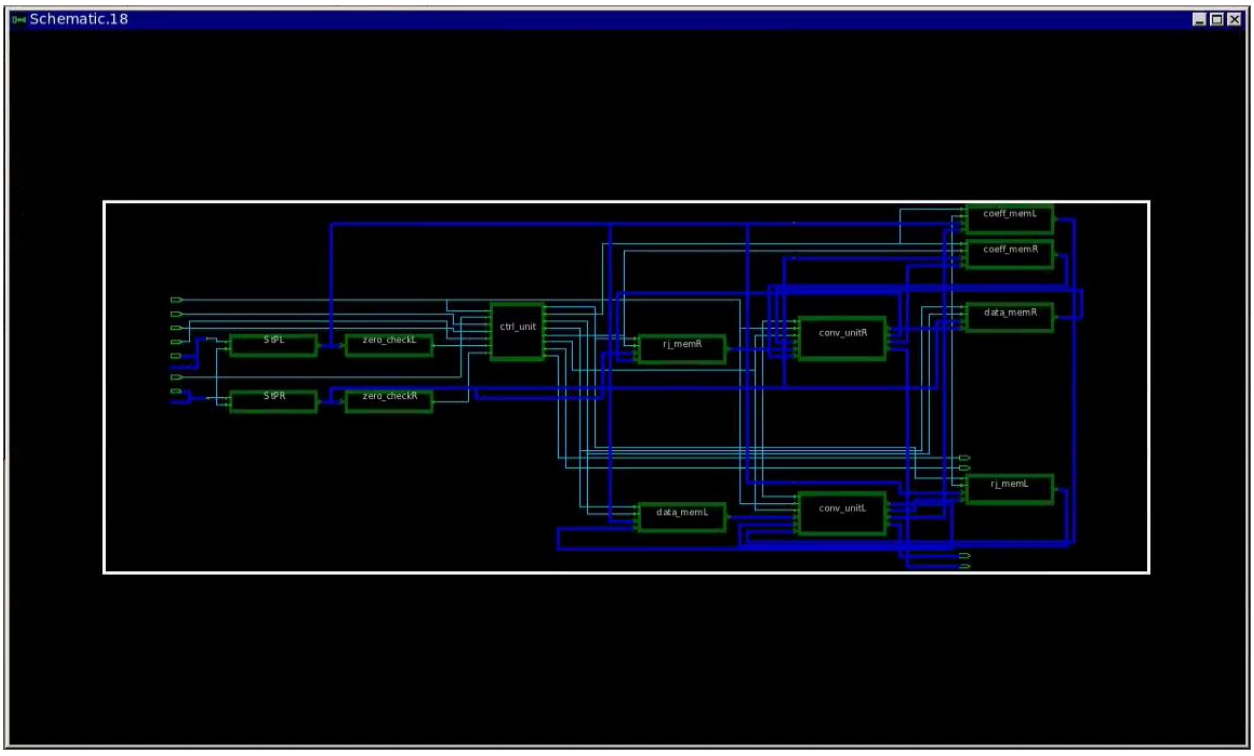

Figure 20. MSDAP Circuit Schematic

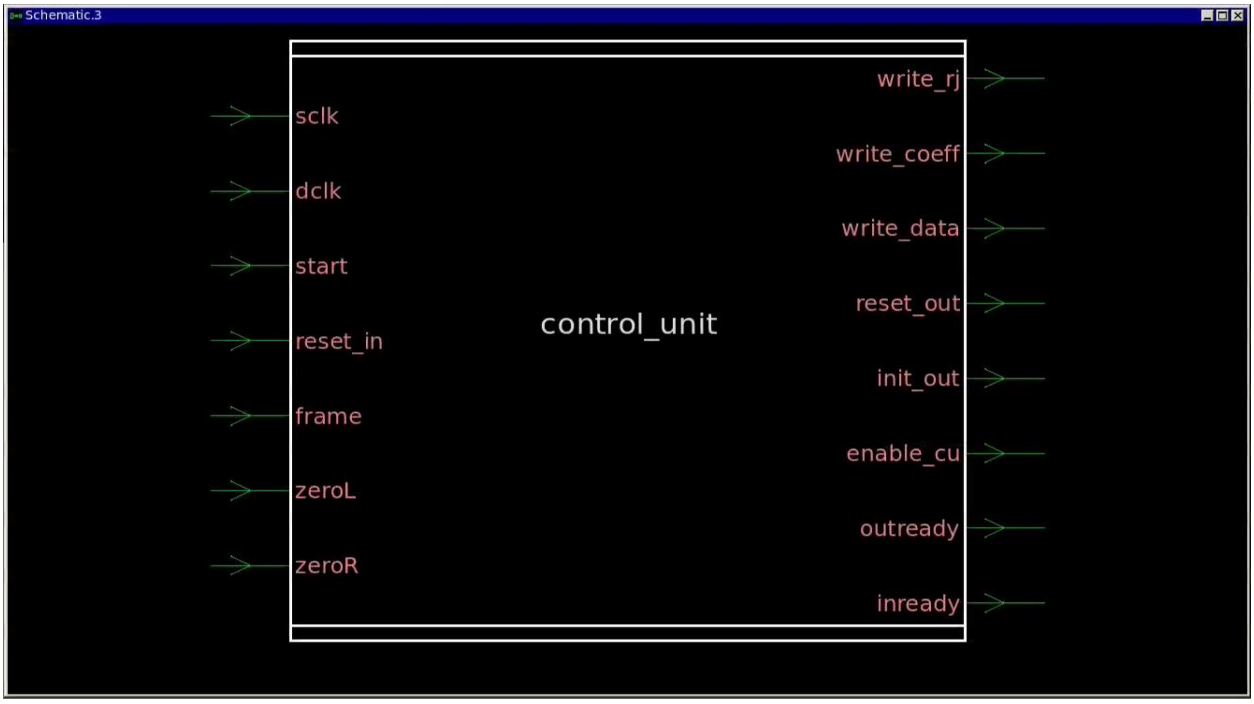

Figure 21. Control Unit Block

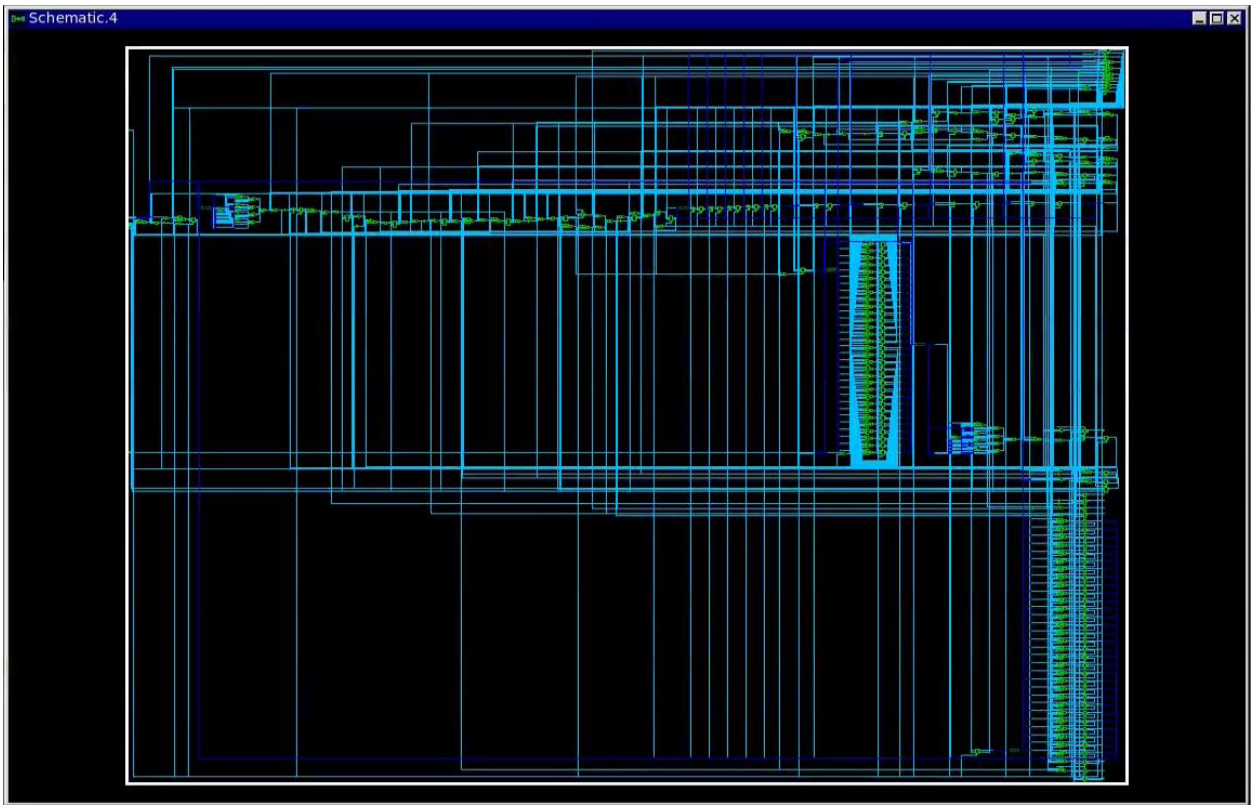

Figure 22. Control Unit Circuit Schematic

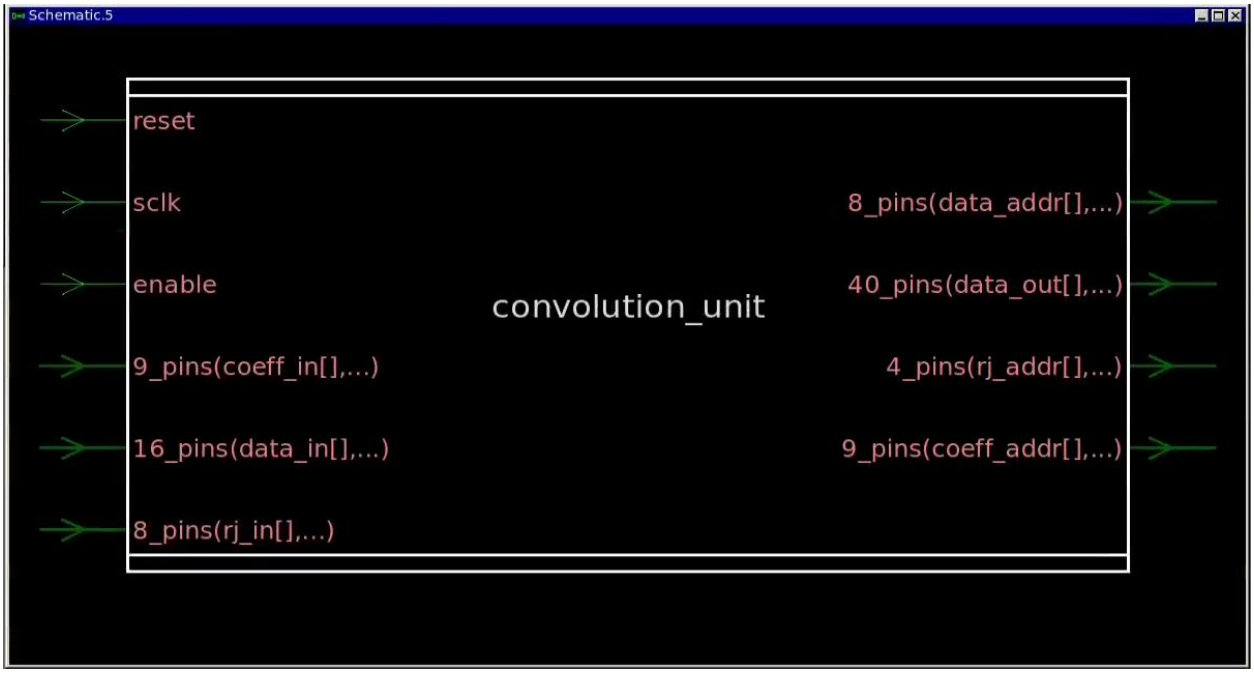

Figure 23. Convolution Unit Block

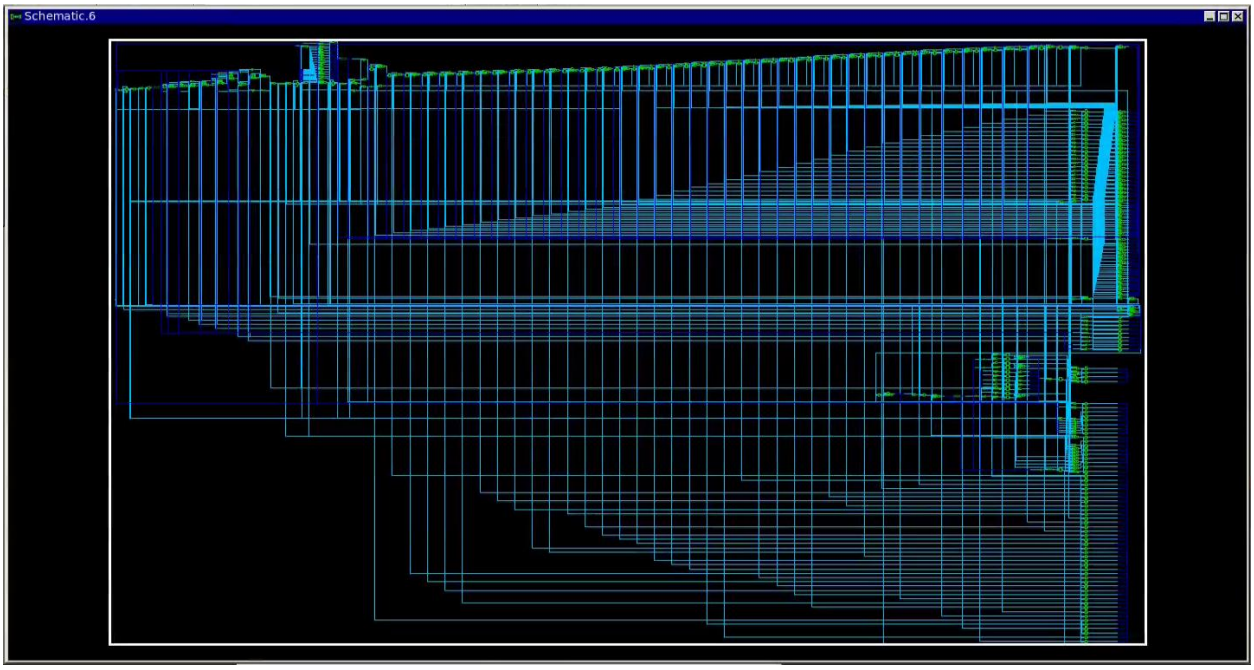

Figure 24. Convolution Unit Circuit Schematic

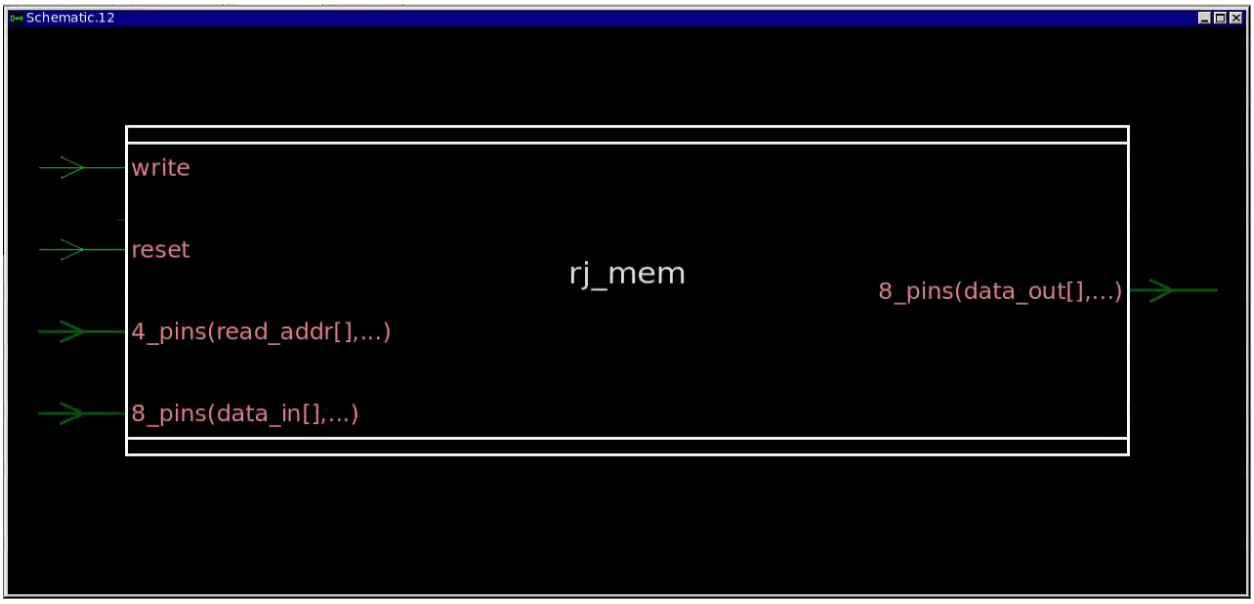

Figure 25. RJ Memory Block

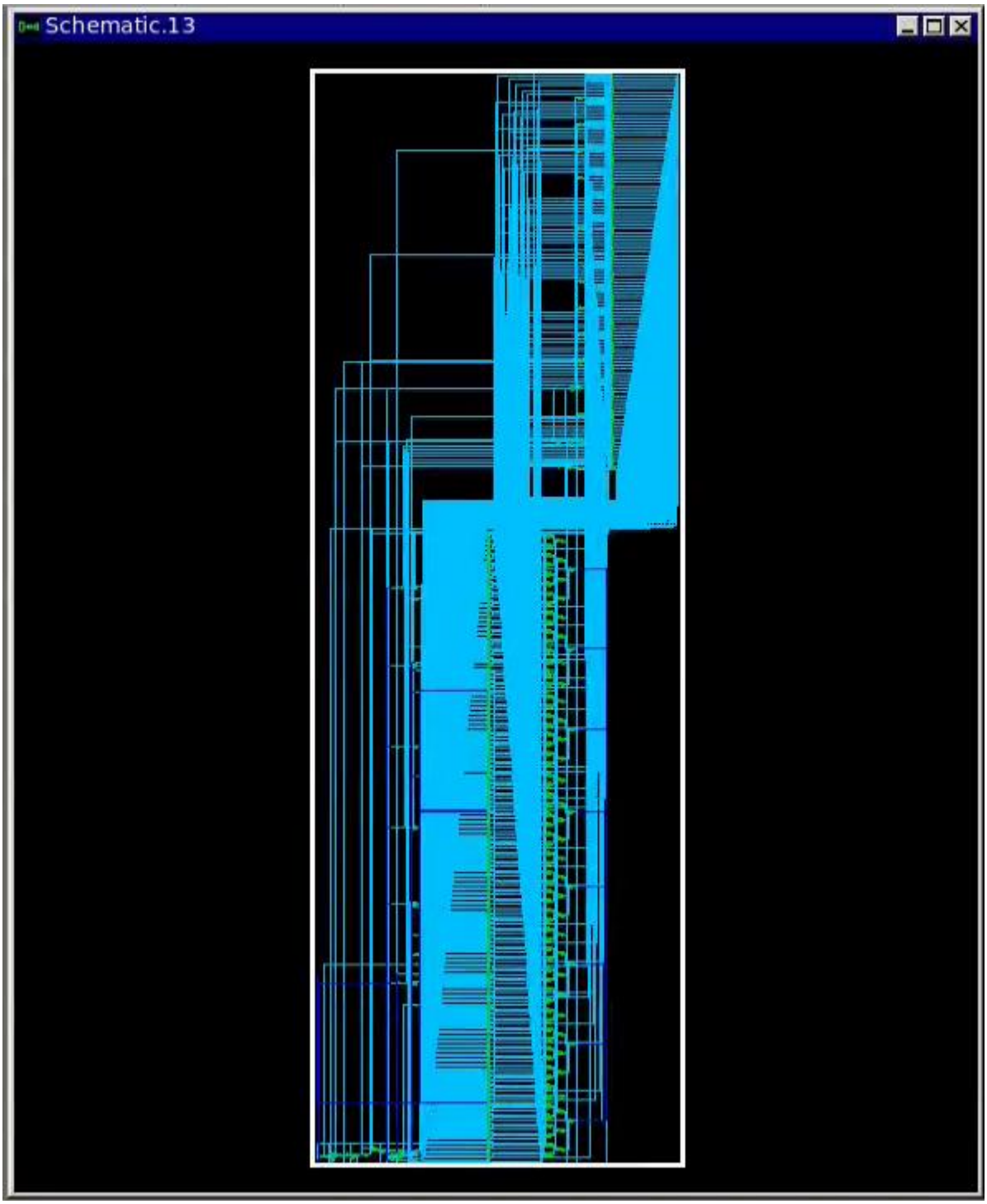

Figure 26. RJ Memory Circuit Schematic

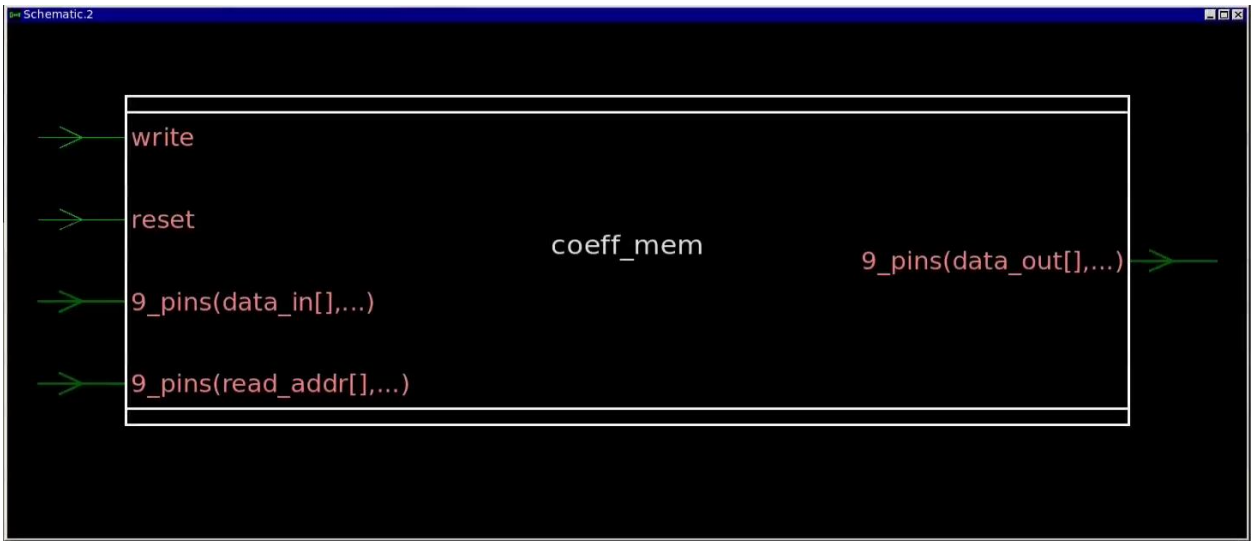

Figure 27. Coefficient Memory Block

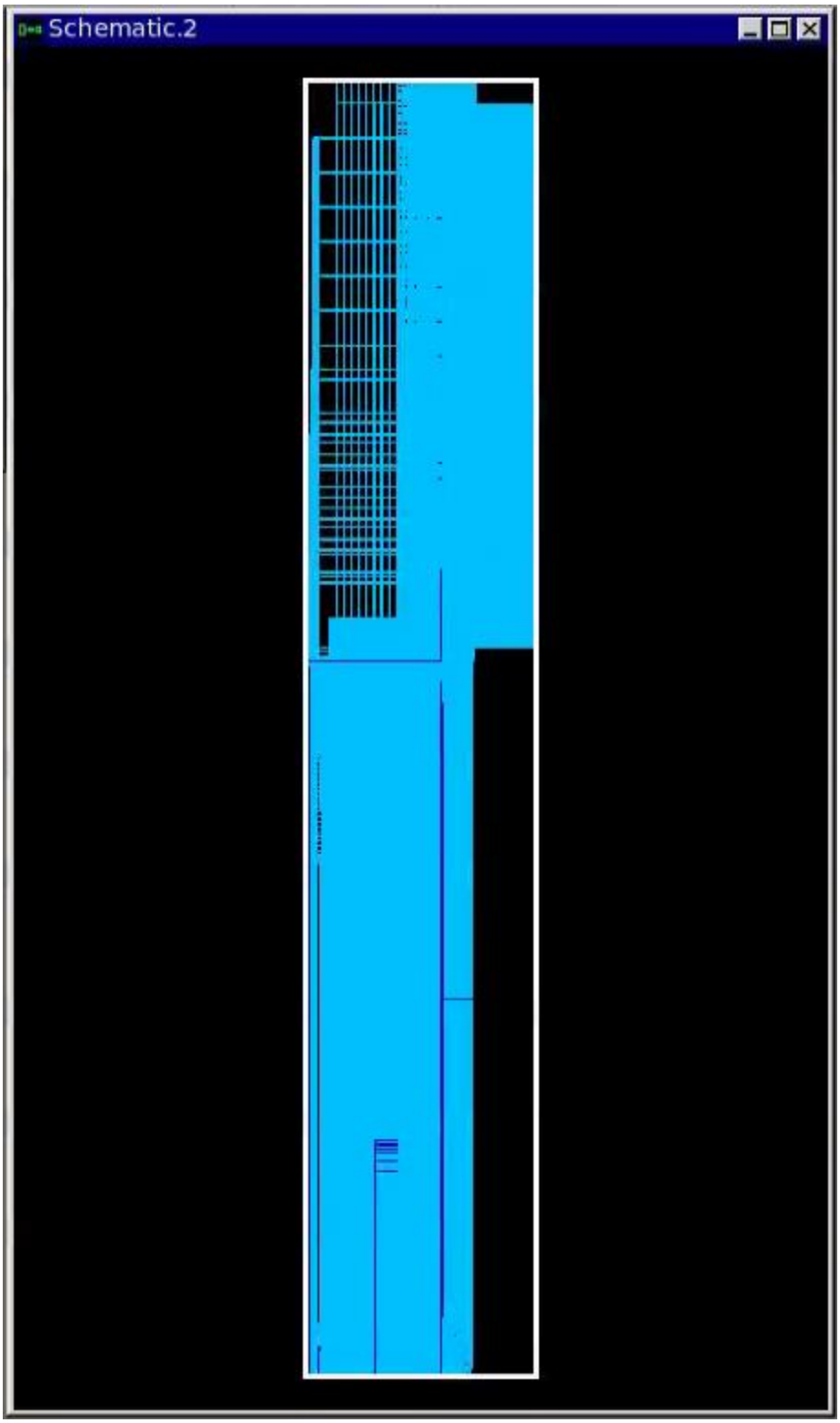

Figure 28. Coefficient Memory Circuit

Figure 29. Data Memory Block

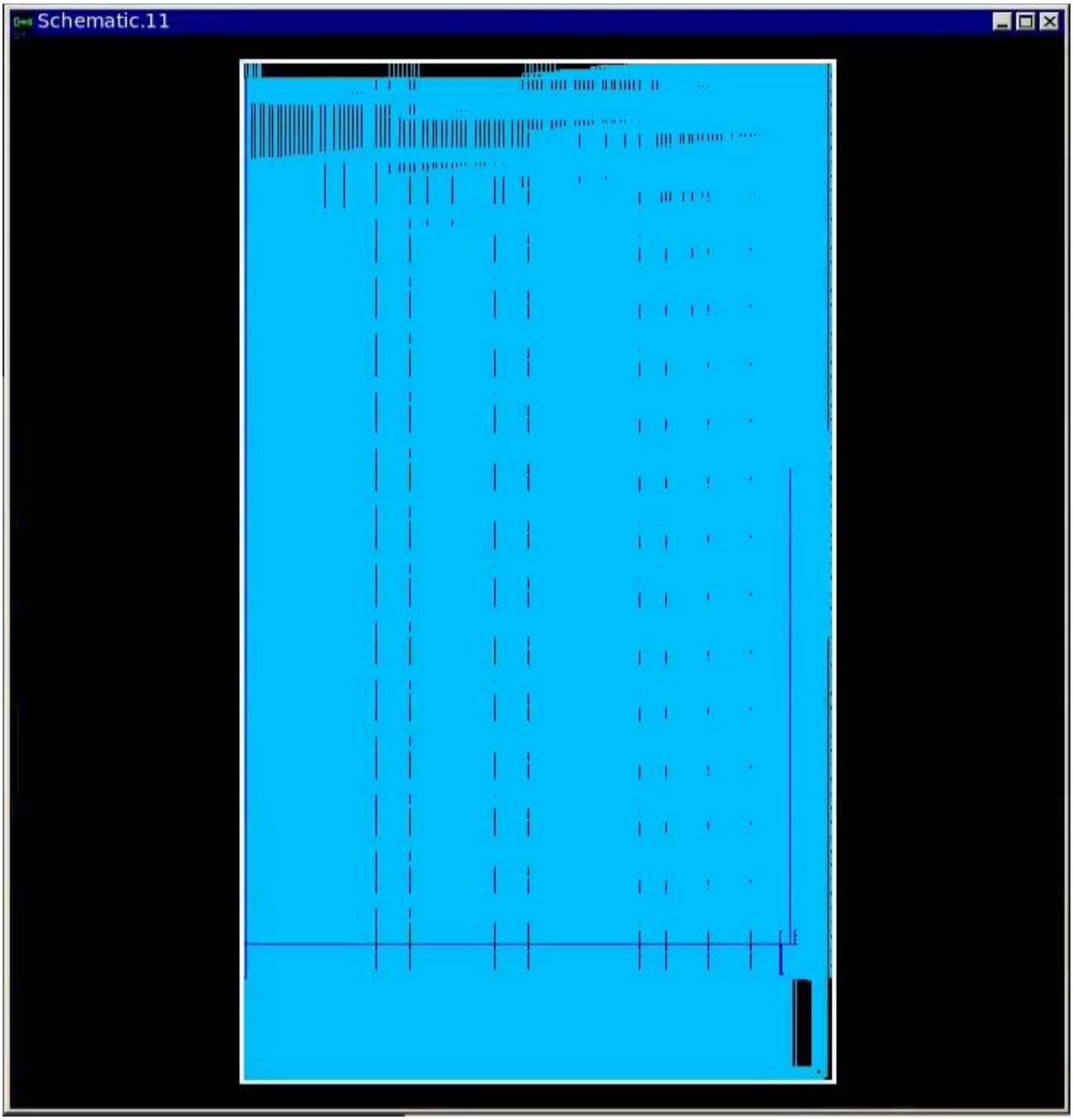

Figure 30. Data Memory Circuit Schematic

Figure 31. Zero Checker Block

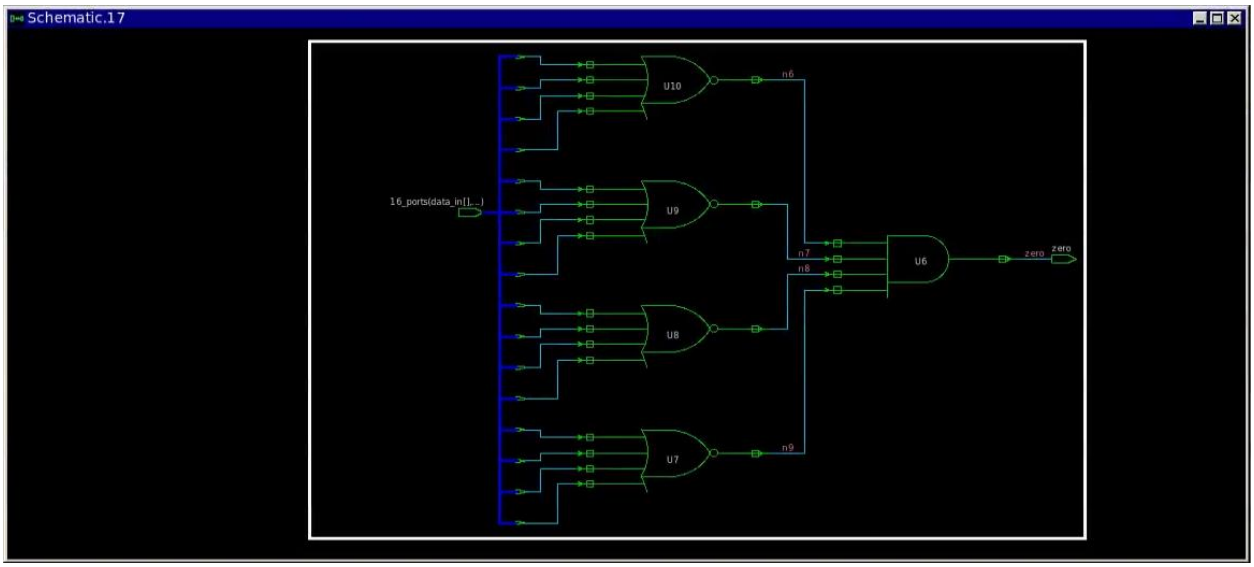

Figure 32. Zero Checker Circuit Schematic

Figure 33. Serial to Parallel Checker Block

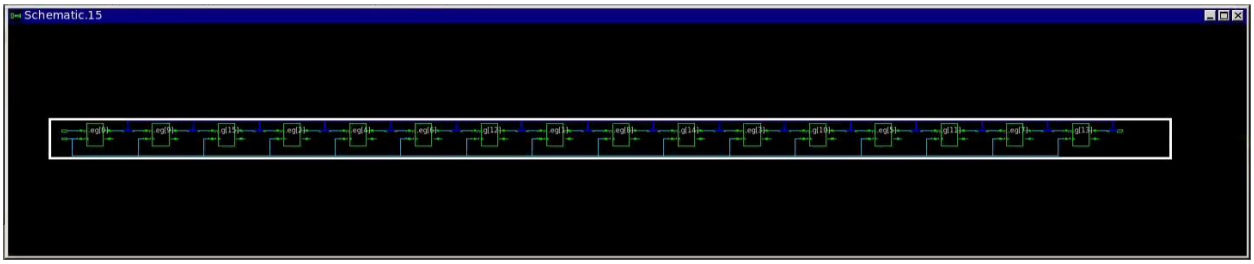

Figure 34. Serial to Parallel Circuit Schematic

Simulation Results

The MSDAP RTL verilog code was simulated to verify the correct functionality. All 7000 input values were passed to the MSDAP and the outputs were recorded. The number of outputs was slightly lower than the number of inputs at 6393. The loss of roughly 600 values is because no output is generated for when the MSDAP is in State 8, sleeping mode. Figure 35 shows the first 3 output values and Figure 36 shows the last 3 output values. All outputs were written to a file and were verified to be correct.

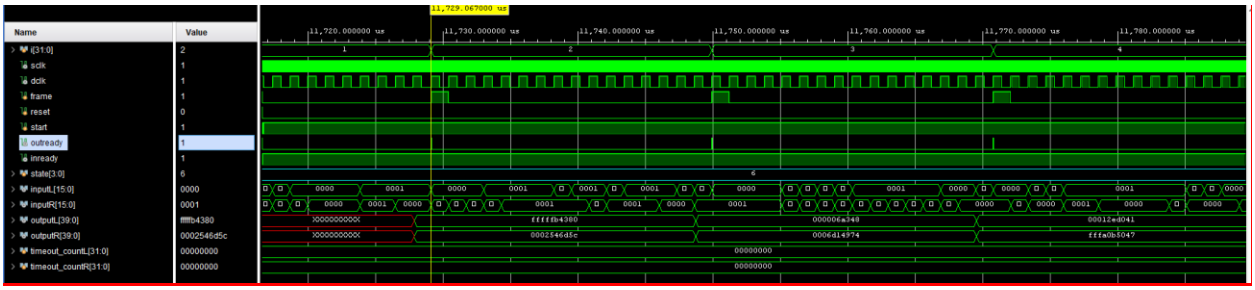

Figure 35. First 3 Output Values

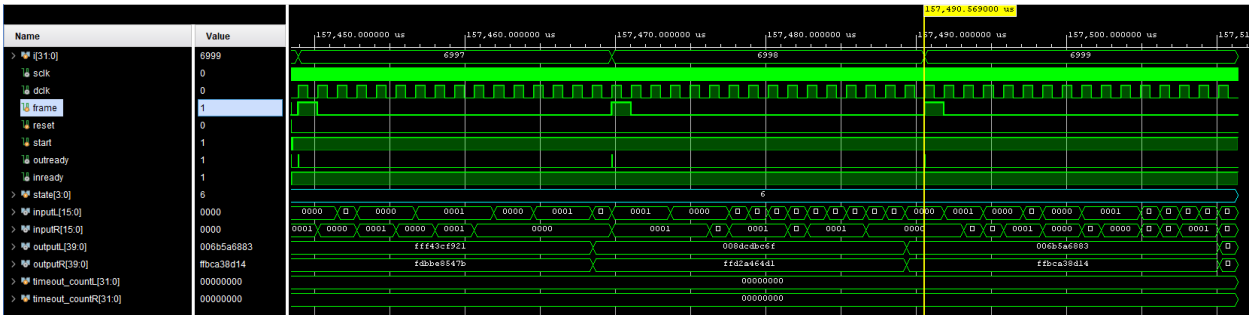

Figure 36. Last 3 Output Values

Design Vision Reports

Design Vision generated reports for area, number of cells, quality of results (qor), and timing. The full reports can be found in Appendix B. The total cell area resulted in 1239078.460320 units. The cell report listed 13 cells. This low number was due to hierarchical grouping of functional blocks. When the cell report was rerun using the ungroup all argument, the number of cells was reported to be 00000. The quality of results report found no negative slack or other violations. The timing report verified the timing of the critical path of the MSDAP for both the dclk and sclk groups. The slack was met with 0.54 and 0.87 units of time to spare.

Critical Path

Design Vision found the critical path to be control logic within the control unit. This critical path was triggered by the rising edge of the system clock at 648.00 units of time and had to reach a timeout counter register by 650.88 units of time. The data arrived at 650.34 units of time, leaving 0.54 units of time to spare. The critical path is shown in Figure 37.

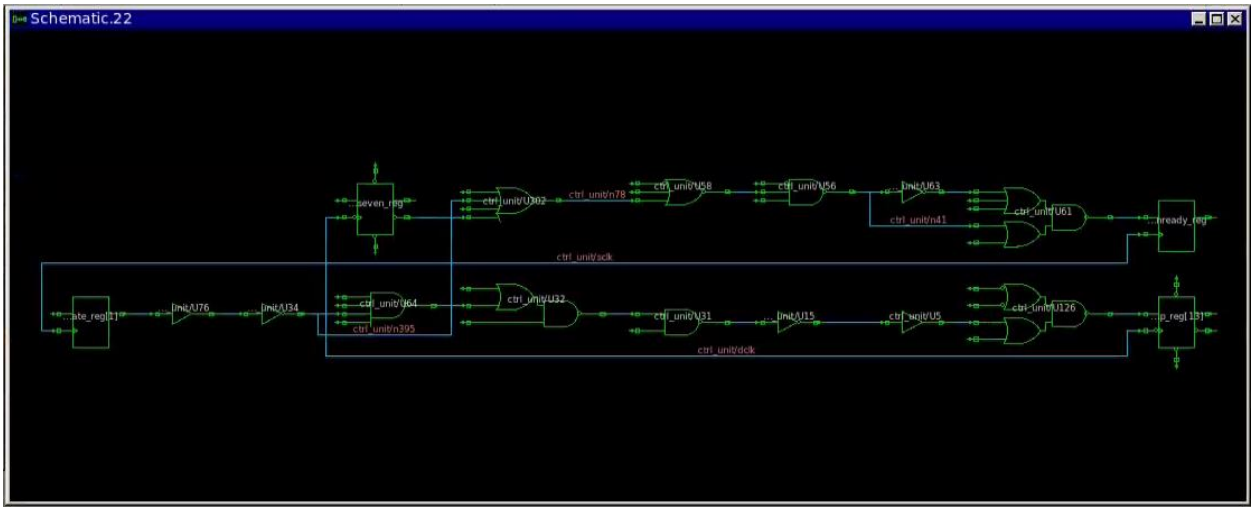

Figure 37. Critical Path

Physical Design

After utilizing Design Compiler to generate the netlist, Synopsys IC Compiler is used to generate the floorplan. Various floorplans were generated to optimize utilization, cell count, and area. Table 2 shows the floorplans generated and their evaluation parameters.

Table 2. Physical Designs

Target Utilization	Margin	Actual Utilization	sclk f (MHz)	Number of Cells	Timing (sclk group)	Timing (dclk group)	Power Consumption (mW)	Chip Area
0.5	20	48.97	80.64	44636	0.01	0.38	4.3198	1127211
0.6	20	58.74	80.64	44603	0.07	0.39	4.0446	939156
0.7	20	76.92	80.64	44358	0.16	0.39	3.9848	704374
0.8	20	76.92	80.64	44358	0.16	0.39	3.9848	704374
0.6	15	58.75	80.64	44624	0.16	0.38	3.9797	939156

In total five floorplans were generated. Four of the floorplans used margins of 20um with various target utilizations. One floorplan used 15um with a 0.6 target utilization. There was a clear winner with the 0.8 target utilization with 20um margins. Technically, the 0.7 and 0.8 target utilization resulted in the same design due to placement algorithm's limitations. The utilization after placement and clock tree synthesis resulted to 76.92% with the best positive slack values and chip area usage. The sclk frequency remained unchanged throughout the different floorplan. The 0.6 target utilization with 15 margins had slightly better power consumption, beating the chosen design by 5uW of power. However, when considering all factors, the 0.8 target utilization was a better choice. The chosen floorplan is shown in Figure 38. The other floorplans generated are shown in Figure 39 through Figure 42. As the table indicates the target utilization of 0.7 and 0.8 are identical, this is also seen in the associated floorplans Figure 38 and Figure 41. The IC compiler reports are viewable in Appendix C.

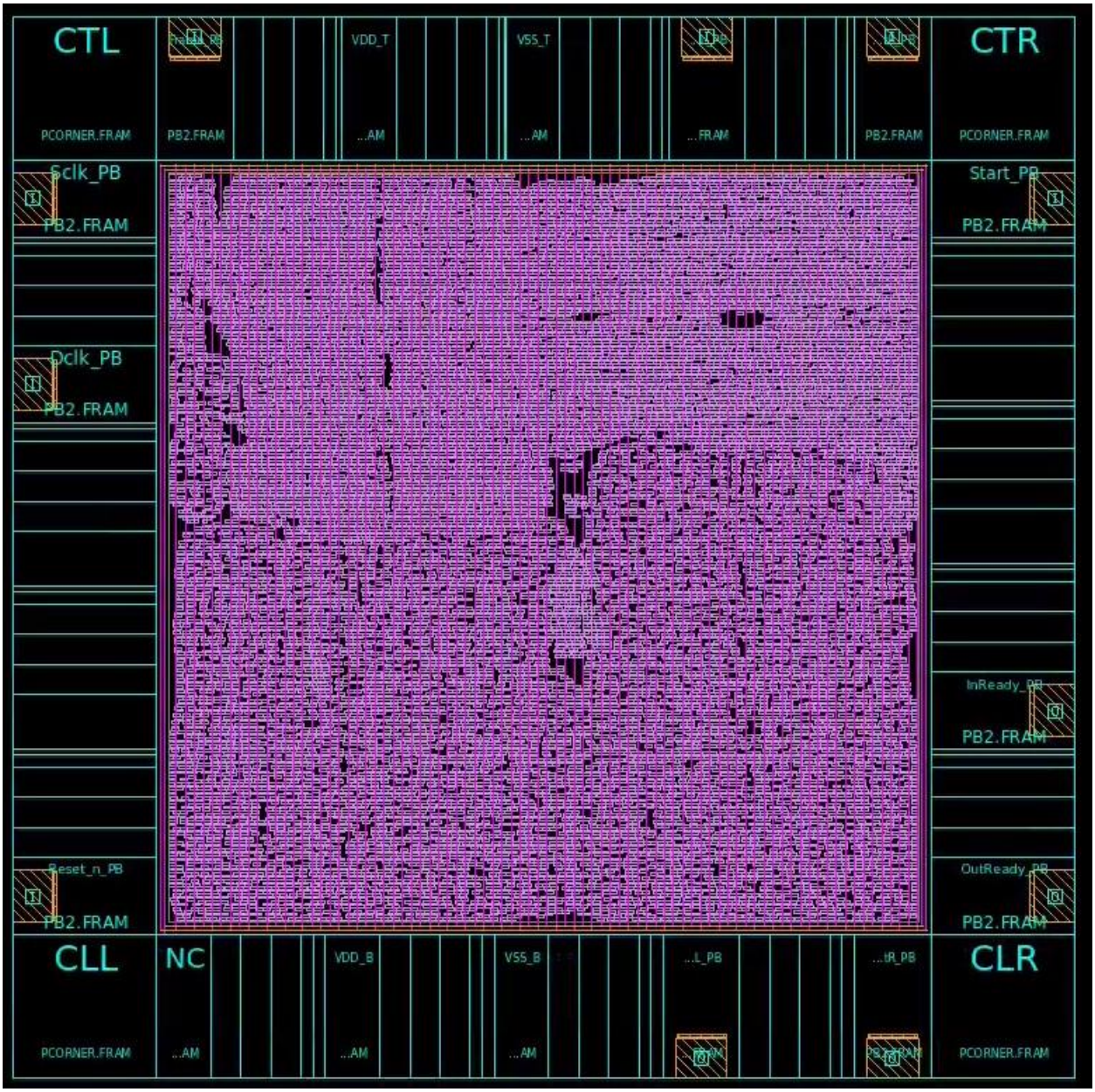

Figure 38. Floorplan for 0.8 target utilization, 20um margins (Chosen Floorplan)

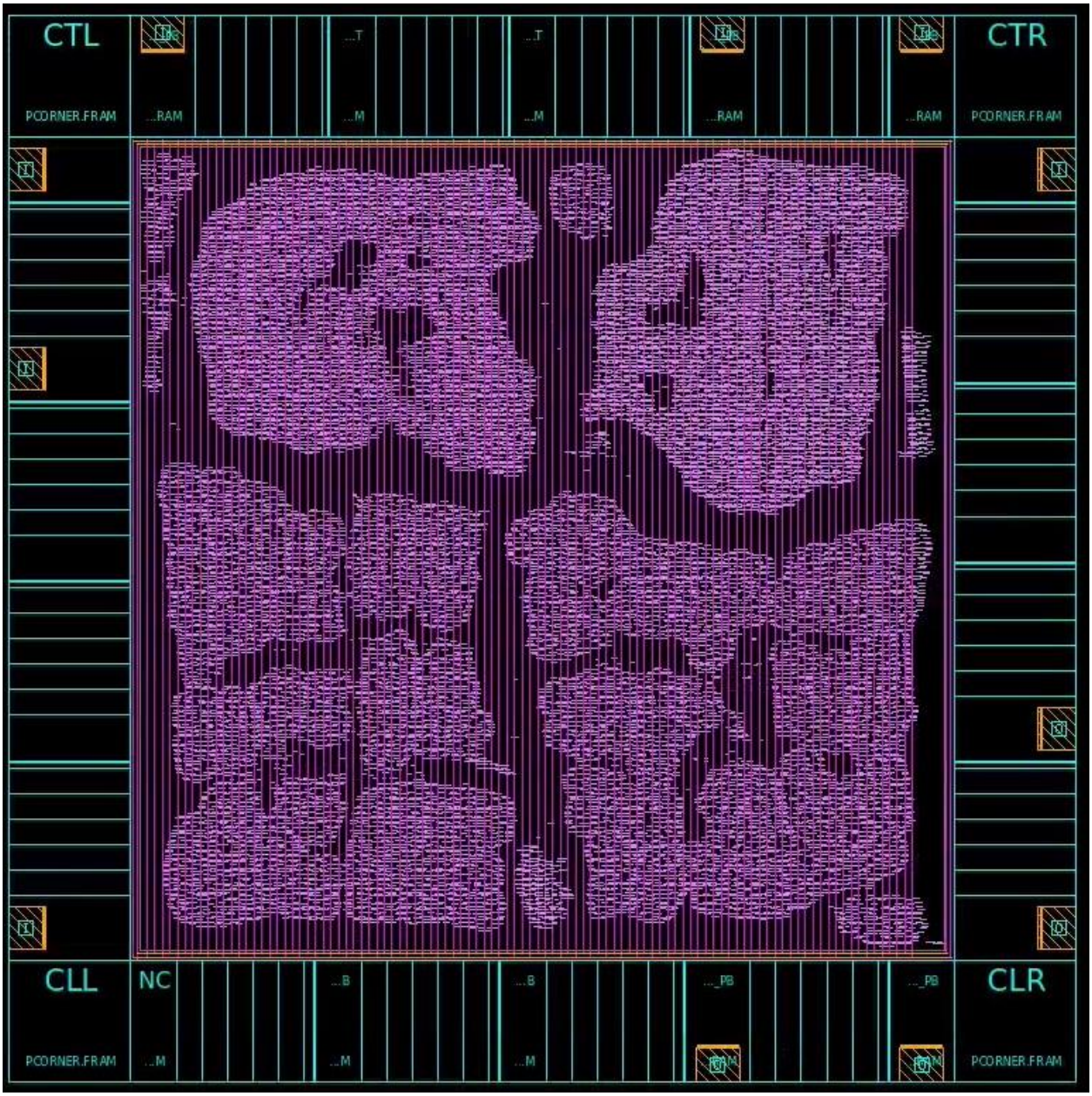

Figure 39. Floorplan for 0.5 target utilization, 20um margins

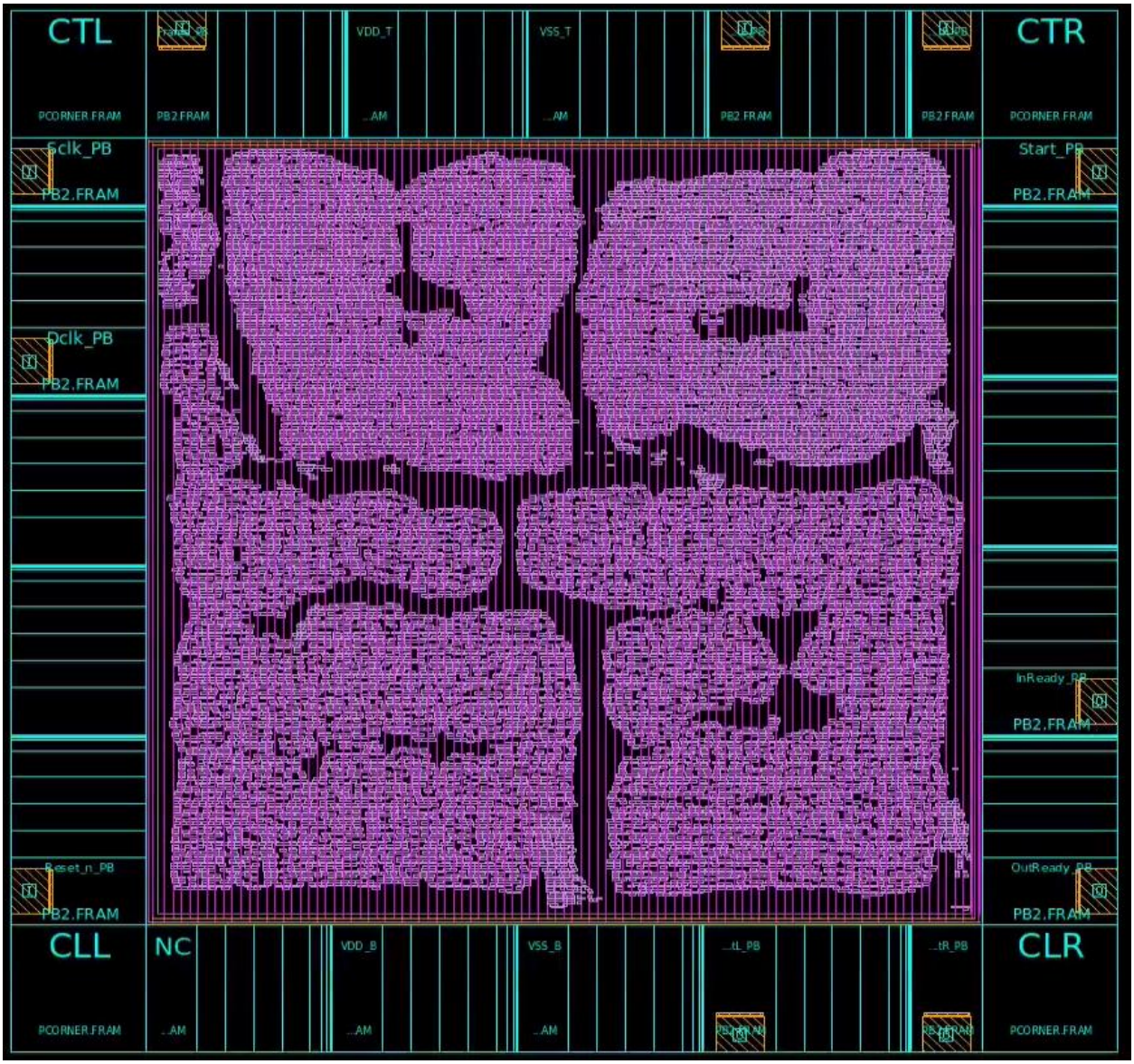

Figure 40. Floorplan for 0.6 target utilization, 20um margins

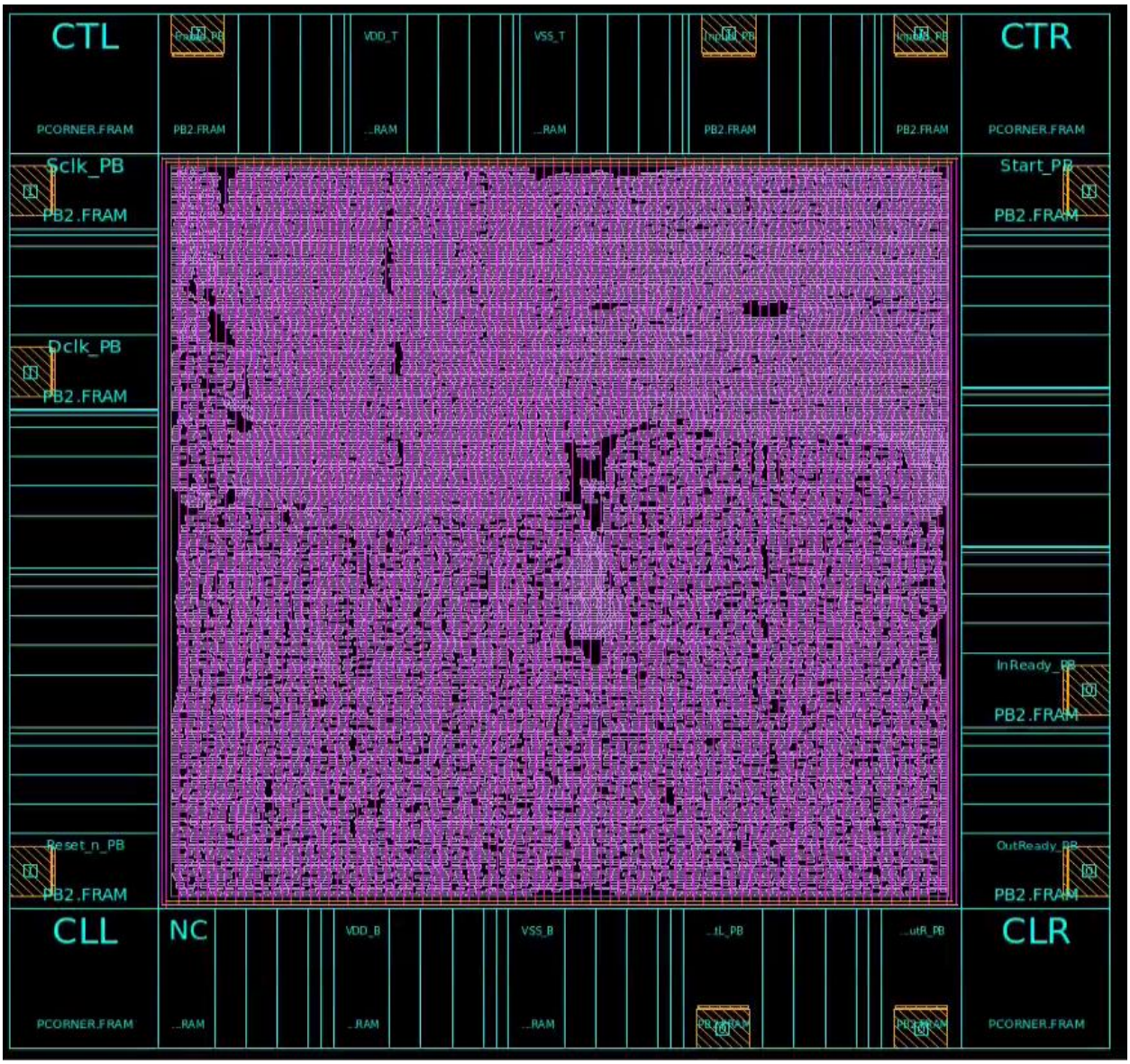

Figure 41. Floorplan for 0.7 target utilization, 20um margins

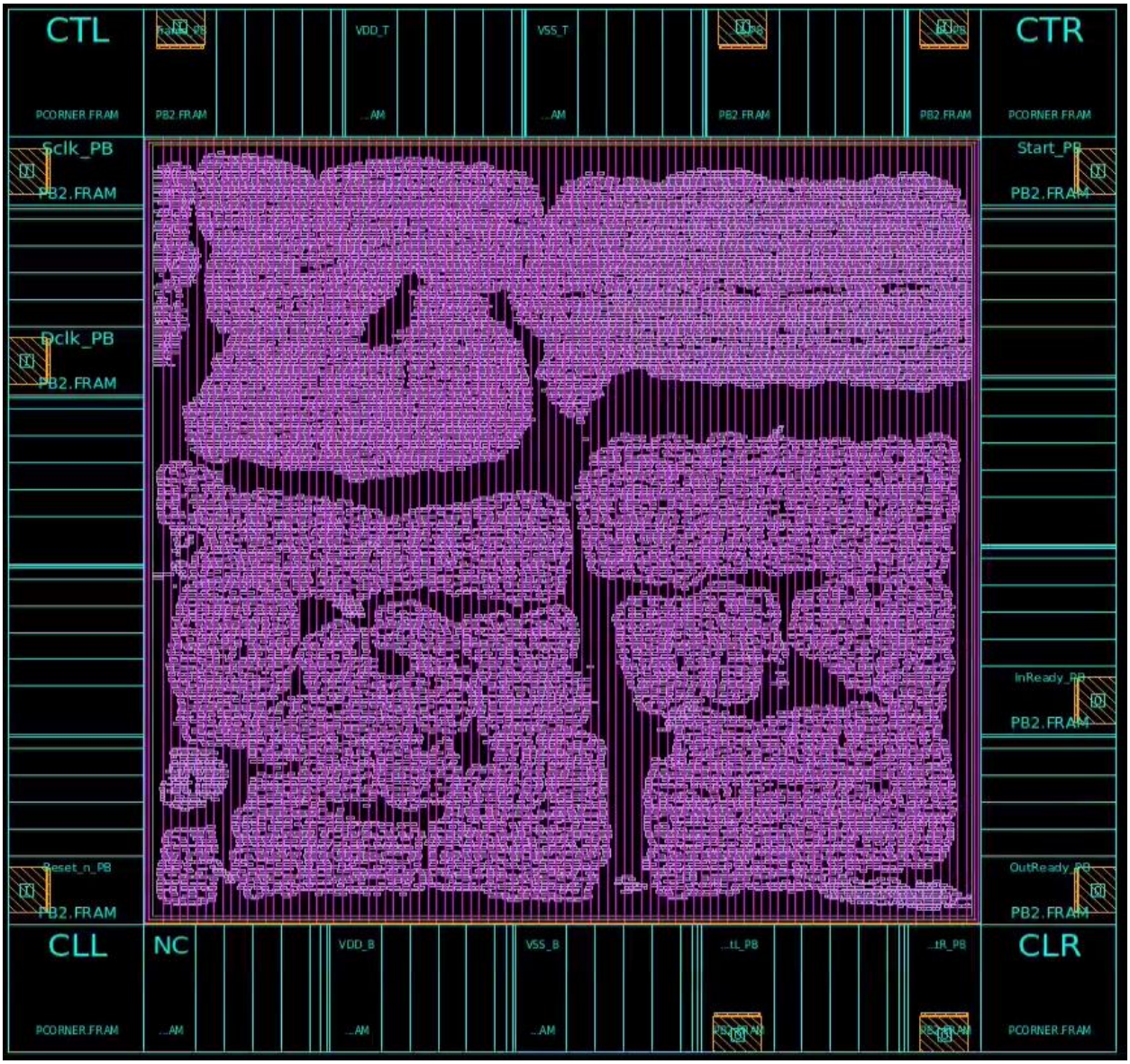

Figure 42. Floorplan for 0.6 target utilization, 15um margins

Special Topic on Verification

Physical Verification

Challenges of Physical Design in today's nanoscale VLSI design:

- Moore's law states that the number of transistors for the same area doubles every year; as a result, the cost of wiring increases every time the process is upgraded to a new finer fabrication method.
- As the number of transistors increases, the switching power also increases which needs to be handled. The devices get smaller, but many aspects of their performance deteriorate: leakage increases, gain decreases, and sensitivity to unavoidable small fluctuations in the manufacturing process rises dramatically. Power and energy becomes the key limiters on many new designs. [1]
- The IR drop problem also increases as the technology increases which results in more signal integrity issues.
- All the above issues require move vigorous efforts on the engineer's side while working with routing, floor planning, physical verification and etc.

Clock Tree Synthesis (CTS)

The placement and routing goal is always to follow the time limits of the .sdc file. If latency and uncertainty are not set for the clock at the front end, it is not possible to do the clock tree synthesis (CTS) at the back end.

Cell delay consists of transformation, timing arcs and capacitances, while net delay consists of RCs only. Cell Delays can be found in libraries.

The aim of CTS is to reduce skew and insertion delay; clock is not propagated before CTS. When CTS is implemented, the hold slack time improves.

Processes on physical verification:

1) Steps for Design Rule Check (DRC)
   DRC errors will help the user to find if there are any errors that are against the process technology rules. In IC compiler, to check the DRC errors follow the below steps:
   a) Choose verification option on the menu panel.
   b) Select DRC.
   c) Tick the relevant checks required.
2) Steps for Layout versus Schematic Check (LVS)
   LVS errors helps the engineer to identify any shorts in the design, or any mismatches in the net. In IC Compiler, follow the steps to check LVS errors:
   a) Choose verification option on the menu panel.
   b) Select LVS.
   c) Tick the relevant checks required.
3) Steps for Antenna Check
   The Antenna check is a complicated safety check that governs the area ratio of all metal layers in the design and the corresponding capacitance.

a) Click on verification menu on the ICC and select the Antenna check to perform the antenna check on the design.

4) Steps for ERC

The ERC is an electrical rule review that guarantees that the circuit is correctly manufactured. This is achieved by testing the relative location of the cells in the final layout.

a) Click on verification menu on the IC Compiler and select the ERC to perform the ERC check on the design.

ASIC Design Flow

Step 1: Specification

In the beginning stages of designing any ASIC chip, it is important to analyze the requirements set forth by the customer. These requirements are then converted to specifications for the development of ASIC design.

Step 2: High Level Design (Behavioral Description)

The goal during this process is to acquire design criteria based on the application being applied. Behavioral modeling is a high-level modeling approach where logical behavior is modelled on the basis of the learned specifications. The behavioral logic is then checked by waveform generation and testing of the application's functionality. The structural implementation is not significant at this point.

Step 3: Low level design (RTL Design)

In order to explain the logic design, Register Transfer Level (RTL) can be used. Unlike behavioral modeling, where high-level code is written, RTL is written at the transistor level. The application specification is added at register level during this point, hence the name register-transfer level. All computations being done It is the task of registers to store values in it while all other computations are done in combinational logic.

Step 4: RTL verification

The RTL code is checked in this stage by testing its functionality. It is possible to accomplish this by writing test benches; to assess the proper operation of the application, writing test benches must cover all potential instances. Testbenches are behavioral models that emulate the surrounding system environment to provide input stimuli to the design under test and process the output responses during simulation. RTL models of the detailed design are then developed and verified with the same testbenches that were used for verification of the architectural design, in addition to testbenches that target design errors in the RTL description of the design. [1] The previous step of writing RTL code may need to be visited if this verification step discover any fault in the design. Necessary reports on area, timing and power are generated in this stage which help to evaluate the verification of the RTL design. This is one of the essential steps in the execution of a proper program.

Step 5: Logic synthesis

In this stage, the RTL code is synthesized using the standard library or with any additional libraries. The schematic and the block level architecture is generated at this stage as well. The logic

designed in the high-level stage, such as fixed state machines, sequential and combinational logic will be mapped to the gates using the cells defined in the included libraries. Tools, such as Design Vision by Synopsys, perform all the aforementioned logical optimizations during this step. Certain design constrains, such as the specification of time delay and environmental restrictions (areas, power, process etc.) can also be added to the synthesis tool which later gets included in the generated netlist file. The netlist file's functionality can later be verified by running it through the testbench and should generate the same output as the high-level HDL files.

Step 6: Floor planning Automatic Place and route

The netlist file generated in the previous stage is used to place and route the cells and blocks on a chip. This includes steps such as floor planning, placement, and routing. Each stage needs to be carefully done to get the best result of the usage of resources on a chip. Chip resources utilization can be configured as per the designer's choice; some options available are core area, boundary, soft blockages, hard blockages, macros, etc. The most generic utilization metric is area utilization. Several simulations are carried out by the tools, such as Synopsys IC Compiler, to reduce the average area. Routing provides an abstract wire length and a defined channel where the actual route can take place. After this, a thorough routing repeats based on an advanced algorithm. All this is done to minimize delays spanning the whole chip.

Works Cited

[1] Y. C. X. L. K. M. L. T. P. R. A. R. K. L. S. Benton H. Calhoun, "Digital Circuit Design Challenges and Opportunities in the Era of Nanoscale CMOS," *Proceedings of the IEEE,* vol. 96, no. 2, pp. 343 - 365, 2008.

[2] L.-T. (.-T. W. Y.-W. C. Charles E. Stroud, "CHAPTER 1 - Introduction," in *Electronic Design Automation*, Morgan Kaufmann, 2009, pp. 1-38.

Appendix