MSDAP Final Project

EEDG 6306 ASIC Design

Mark Sears and Seoha Lee

12/15/18

SECTION 1: ASIC DESIGN SUMMARY

In this section, we explain the process of designing Application Specific Integrated Circuits (ASICs). Later sections detail the implementation of the MSDAP project.

When deciding how to implement and design a silicon chip, different design methodologies are available. Designers can use FPGAs, standard cell layouts, or ASIC chip designs. While ASICs are not the most common choice, they offer unique advantages over the other design methodologies:

- Reduced overall area.
- Improved performance in speed.
- Reduced power consumption.
- Reduced cost per chip.

Disadvantages of ASICs include:

- Increased design complexity.
- Increased time-to-market.

These factors make ASICs desirable for applications with high-volume or strict performance requirements. In this project, the Mini Stereo Digital Audio Processor (MSDAP) is taken through the ASIC design process from specification all the way through verification for tape out.

1.1 Specification

A significant amount of effort should be used for a careful specification of a chip. During this process, the initial concept is scrutinized by designers and turned into a set of inputs and outputs which perform the required function. The precise behavior of the chip should be defined, including signal formats. The more detail that is put into the specification, the easier it will be in later design stages to know exactly what the chip is intended to do. If the specification is ambiguous or incomplete, there is a greater chance that errors will be introduced.

The total number of input and output signals should be minimized without compromising the functionality of the design. For example, in the MSDAP the 40 bit output data is sent serially, which means only a single pin is required for output data for each data channel.

Designers must have an in-depth understanding of the application. The customer may be questioned to clarify the design intent. The computational aspects of the design can be simulated in a software program. This allows designers to gain insight into exactly what computation must be performed and to explore alternative implementations. In the MSDAP, understanding the convolution computation is critical to implementation with addition and shift operations.

1.2 Behavioral Model

It is usually easier to describe the behavior of the entire chip as a single module. This is a good step before moving on to a more detailed architecture. The design can be written as a large "black box" in Verilog (or VHDL) such that the outputs are correct without much thought given to implementation or how the data flows through the module. When completed, the behavioral model can be simulated with a testbench and compared with the specification for correctness. In further design steps, the behavioral model can be used as a reference to check correct functionality.

With a behavioral model description, the design could be fed into CAD tools and synthesized into a final design layout. However, doing this would most likely result in an inferior chip layout in terms of power consumption, performance, and area compared to doing a more detailed architectural design.

1.3 Architecture

With a firm understanding of what the application must do, a hierarchical chip architecture can be defined. Functional blocks are defined which perform a small part of the larger chip functionality. Each block has its own set of inputs and outputs, which must be specified separately.

Within each of these functional blocks, additional levels of hierarchy can be added until the designer is satisfied with the level of abstraction. In larger designs, there may be many levels of hierarchy. Defining these functional blocks and how they interact with each other has a number of benefits:

- Designers can be more explicit in describing the intended design.

- An intuitive understanding of each block can be developed.
- Each block can be tested and verified individually without needing to be placed in the complete system.

The flow of data and interactions between these blocks must be carefully defined and tested using RTL verilog code. This means that designers must have an understanding of exactly how and on which clock cycle data is moved through the functional blocks. After developing this structural RTL code, the design can again be simulated and compared with the operation of the behavioral model.

1.4 Synthesis

In order to be manufactured, a design must be represented in terms of transistors and metal interconnect. HDL code must be written carefully to ensure successful synthesis. The synthesis process is largely automatic using a CAD tool such as Synopsys Design Compiler. This tool takes an HDL design and a library of cells as inputs and composes a netlist of gate logic which implements the design using only the cells in that library. Synthesis may be done with some constraints, such as clock timing.

After a netlist is obtained, it must be simulated to ensure the design functionality was not compromised during synthesis. The same testbench used for the behavioral model can be used to test this netlist, and the output waveforms can be compared for correctness. The synthesized netlist uses delay estimates to more accurately simulate the logic, which can cause functionality issues when compared to the behavioral simulation. For example, a logic operation may cause setup or hold time violations on D Flip-flops. The original verilog code may need to be carefully rewritten to avoid these errors.

The synthesized netlist should have an architecture identical to the verilog code. If necessary, each block can be simulated individually. The operation of each block can be compared with its own specification to ensure proper functionality

1.5 Physical Design

Once the synthesized netlist has been verified, physical design can begin. This includes all the steps necessary to create a layout of transistors and wires which will be used to fabricate the chip. The synthesized netlist and technology rules are inputs to the physical design process. Most of the physical design process can be automated using a CAD tool such as Synopsys IC Compiler. The finished layout, often in GDSII format, is the output.

The first step is floorplanning, where each functional block is placed somewhere in the layout. The main objectives here are to minimize chip area while maintaining wire routability. A good layout will lead to a better final product, as layout is a large factor which determines chip area and wire delay. For this reason, it may be worthwhile to try many different floorplans and compare the results.

After floorplanning, the wires can be routed. This is usually split into two steps: global and detail routing. Global routing gives an approximation of where each wire will go. Detail routing determines the exact coordinates and size of each wire needed to connect the netlist.

Another important aspect of physical design is Clock Tree Synthesis. Since the clock is often needed for many gates, it can have a very large fan-out and consume a significant fraction of power. Clock skew is important to control, since logic will behave incorrectly is skew is too high. A well crafted clock tree helps to minimize skew.

Once all blocks have been placed and connected with wires, CAD tools can run verifications such as:

- **Design Rule Check (DRC)**. The technology which will be used for fabrication has many different design rules, such as minimum wire size and spacing. The DRC ensures that none of these rules are violated.
- **Layout vs Schematic (LVS)**. There must be high confidence that the physical design represents the correct system functionality. The LVS compares the gates and wire connections in the physical design with the gates and connections specified in the netlist.
- **Static Timing Analysis (STA)** and power consumption estimates are much more accurate using

the physical design. These can be used to ensure that the chip performance specifications have been met.

After the physical design has been fully verified, the design is ready to be sent out for fabrication.

1.6 System Behavioral Model (Special Topic 1)

An important step in the ASIC design process is developing a system level behavioral design based on the specification. When a customer has an application, they may only have an end goal in mind without knowing much about VLSI or what is required to produce a silicon chip. For our MSDAP project, the original idea contained only the convolution to be performed, and did not specify any inputs, outputs, or exactly how the convolution was to be implemented. After understanding the desired application, the system behavioral model can be developed by defining the following:

- Input control signals (start, reset, frame)
- Input data signals (inputL, inputR)
- Output control signals (inReady, outReady)
- Output data signals (outputL, outputR)
- Clocks (sclk, dclk)

The format for all of these signals must be carefully detailed as well. For example, new output data must be sent out after each frame, using the rising edge of sclk. However, the specific hardware used to achieve these signals can remain as abstract for the behavioral model.

The total number of input and output signals should be minimized without compromising the functionality of the design. For example, in the MSDAP the 40 bit output data is sent serially, which means only a single pin is required for output data for each data channel.

The behavioral model of the system can be understood as a "black box" where the inner workings are hidden from view. The input and output signal formats are defined, but exactly how those outputs are obtained is abstracted. For the MSDAP, all that needs to be specified is that the outputs are the result of the convolution, which is a sequence of adds and shifts. The detailed description of how those adds and shifts are realized is not part of the behavioral model.

The computation must be well understood in order to achieve a correct model. Using a software coding language such as C++ is useful to gain insight into exactly what is needed for the computation. This method makes it possible to see different perspectives on how the required operations might be accomplished. Understanding all possible implementations allows a designer to choose the one that results in the best hardware chip. For example, the MSDAP convolution could be calculated using multiplication and addition operations. However, multiplication is a very expensive operation in terms of area and power. A much better chip can be designed using the "u-term" approach, which only requires addition and shift operations. Once this C++ program is developed, it can be used at all stages of the design process as a golden standard for correct output.

1.6.1 Testing the Behavioral Model

The behavioral model can be described using Verilog. Usually a behavioral description is much simpler to describe than a complete architectural design, and so it can be developed quickly and easily.

To ensure proper functionality, the behavioral model should be simulated in a Verilog simulator. The waveforms of this simulation and the output results can be examined to determine that the behavioral description gives the correct result.

For verification, a correct and robust testbench is of utmost importance. The system inputs must be generated by the testbench with the expected format, and the output status and data signals must be interpreted correctly. For example, the MSDAP input data bits must be updated by the testbench on each dclk.

The testbench will write the outputs to a file as hex values. These values can be compared to the golden output from the C++ program to determine correct functionality of the behavioral model.

1.6.2 Using the System Level Behavior Model

The behavioral Verilog description of the system does not contain any detailed implementation. The behavioral model is excellent for understanding and defining what is expected of the system, but should remain entirely independent of the detailed architectural design. The architecture used to implement the MSDAP is detailed in Section 2.

Since the detailed architectural design will have the same top level signal formats, the signal waveforms and output results should be identical. Since the top level inputs are the identical, the same testbench can be used to test both behavioral and architectural designs. By comparing the behavioral output results with the architectural design outputs, it can be determined whether the two designs are equivalent.

1.6.3 System Verification Tutorial

The following outlines some possible steps a designer could use to verify that the top level system is functioning as expected:

1) The system spec should be scrutinized and all aspects thoroughly understood by the designers. Any ambiguities or uncertainties should be addressed with the customer.

2) A software program can be written in C++ to emulate the primary functionality of the hardware chip.

3) Describe the top level system using behavioral Verilog code. This describes only the functionality of the system and does not include any implementation details.

4) Develop a testbench which can be used to verify the system functionality. This testbench should generate input signals including clocks, control signals, and data bits. The testbench must also read the outputs from the system and write these results to a file.

5) To determine if the behavioral model has the correct output, the testbench must record the outputs in hexadecimal. These hex values can then be compared to the C++ program to determine that the system behavioral model is correct. To compare these values, another C++ program may be written or a spreadsheet can be used.

6) The resulting waveforms must be carefully compared to the system specification to ensure that all signals follow expected formats. For example, output bits must be updated on rising edges of the clock.

By following these steps, the behavioral system model can be verified for proper functionality.

SECTION 2: MSDAP SPECIFICATION

The MSDAP functions as a finite impulse response (FIR) filter for audio signals. It is intended for applications such as hearing aids, which require small physical size and very low power consumption to preserve battery life. This makes it a perfect candidate for an ASIC chip, since the design process allows for the power and area to be minimized better than other design methodologies.

The computation required to implement the FIR filter is a convolution based on a set of coefficients and data values. The convolution can be computed using the sum of products:

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

Where h(k) is a coefficient value and x(n-k) is a previous data input. This sum can be rewritten as:

$$y(n) = h(0)x(n-0) + h(1)x(n-1) + h(2)x(n-2) + \ldots\ldots + h(N)x(n-N)$$

Next, the multiplication operation can be eliminated by defining "u-terms" as a sum of previous inputs. The computation can again be rewritten as a series of addition and power of 2 operations:

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

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

Performing this mathematical transformation has a significant impact on the implementation of the ASIC chip. Multiplication is an expensive operation for hardware to perform compared to addition and shifting. Each u-term is simply a set of addition problems, and each power of 2 is simply a 1-bit shift. This will result in a final design which is smaller and consumes less power.

Figure 1a shows the system level diagram. The MSDAP must be fed audio data from the system. To simulate the system, a testbench is used to read coefficient and values from and send them to the MSDAP. It also writes the results to an output file. The testbench sends 7 input signals to the MSDAP, and receives 4 output signals. These will be discussed in detail below.

2.1 MSDAP Top Level

The MSDAP was designed using a top-down approach. The top level block diagram in Figure 1b has two data paths, each with four modules. The operation is facilitated by the State Controller module which provides control signals to other modules. Input data bits (including RJ and coefficients) are received serially by an S2P module, which parallelizes to 16-bit words. These data words are stored in memories in the Data Manager block. During computation, the Data Manager passes the data stored in memory to the ALU module in the correct order (defined by the coefficients). The ALU adds, shifts, and accumulates the data in such a way to implement the convolution computation. After a computation is completed, the result is loaded into a P2S module which shifts out the 40-bit result. Input and output pins for the top level module are shown below in Table 1.

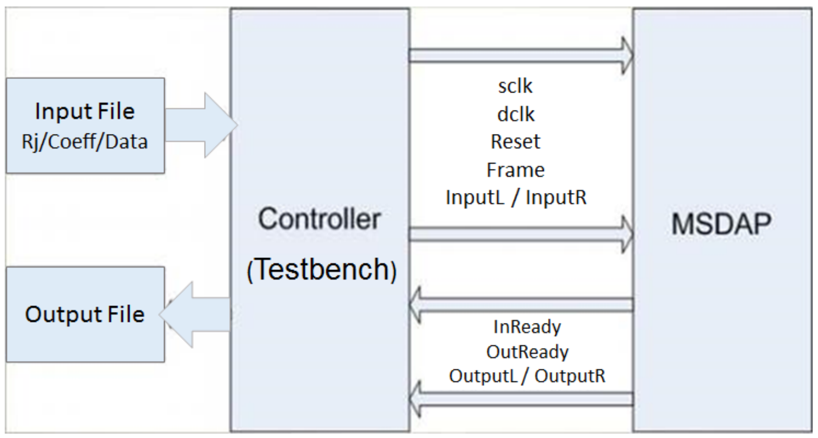


Figure 1a: MSDAP System Overview

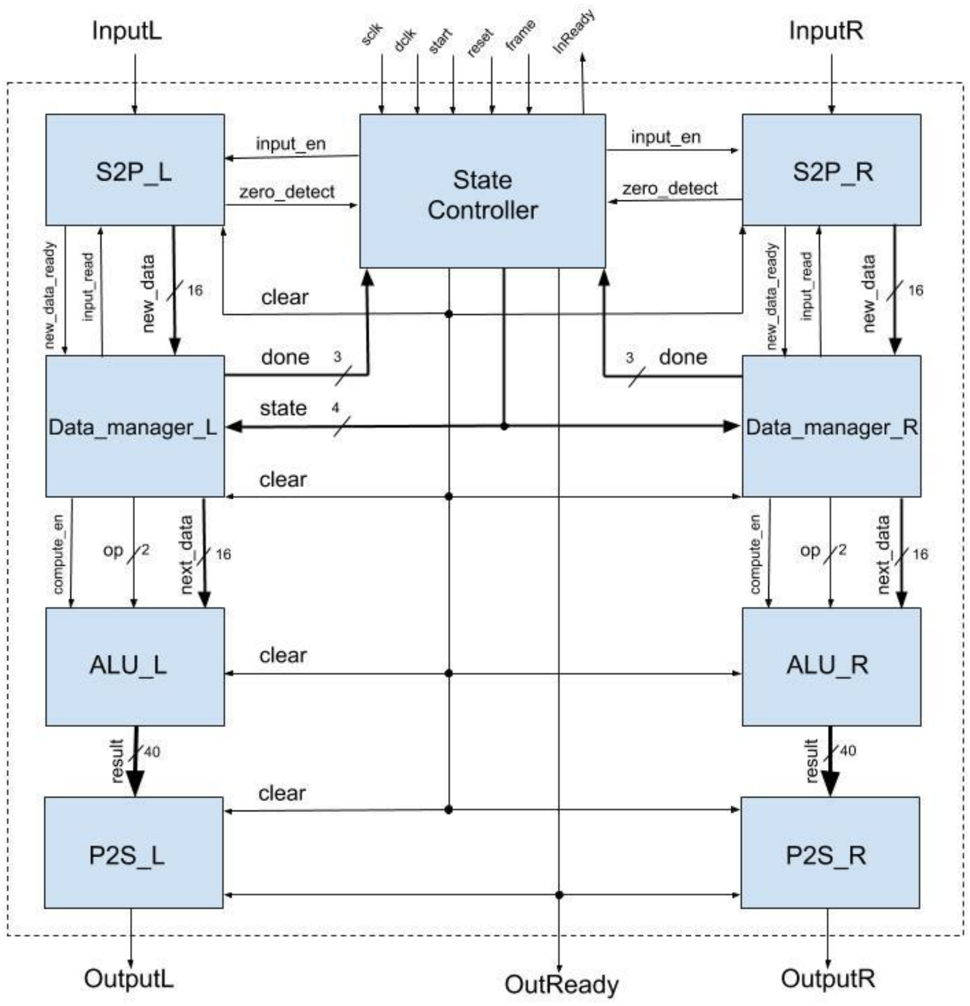


Figure 1b: MSDAP Top Level

Table 1: MSDAP Top Level Pins

Pin Name	**Type**	**Size**	**Description**
sclk	Input	1	sclk (System Clock) clock has frequency 25 MHz. It provides the timing reference for the internal control signals, and output samples. Addition and shift operations are performed on the ***falling*** edge of sclk.
dclk	Input	1	dclk (Data Clock) has frequency of 768KHz (48kHz x 16) and provides the timing reference for input data samples.
reset	Input	1	If reset is set high, chip clears previous input data and all registers (except Rj and coefficient values). Asynchronous with sclk or dclk.
start	Input	1	If start is high, the chip begins initializing. Start is asynchronous with sclk and dclk.
frame	Input	1	The frame signal is sent at 48 KHz. Frame is set high on the ***falling*** edge of dclk for one period to indicate the start of a new input data sequence. It is also used by the chip to start sending an output sequence.
inputR	Input	1	InputR carries the right channel of input data samples (16 bits), Rjs, and Coefficient in serial form. It is read on the ***rising*** edge of dclk. Bit 0 is the least significant bit (LSB) of each serial word, the bit transmitted first. Bit 15 of each serial word is the most significant bit (MSB) and is transmitted last.
inputL	Input	1	InputL carries the left channel of input data samples (16 bits), Rjs, and Coefficient in serial form. It is read on the ***rising*** edge of dclk. Bit 0 is the least significant bit (LSB) of each serial word, the bit transmitted first. Bit 15 of each serial word is the most significant bit (MSB) and is transmitted last.
outReady	Output	1	OutReady indicates the chip is sending 40-bit outputs. If the chip is ready to transmit output samples, outReady is set to 1 on the rising edge of Frame. After all 40 bits are sent, outReady is set to 0.
inReady	Output	1	InReady indicates the chip is able to accept new input data. If the chip is in a state that can receive input data samples and coefficients, inReady is set high by the chip. Otherwise it is set to 0. inReady is set on the rising edge of sclk.
outputL	Output	1	OutputL sends the outputs for the left channel serially. After frame is detected high, outputL is set on the ***rising*** edge of sclk until all 40 bits are sent. The LSB is sent first and the MSB is sent last.
outputR	Output	1	OutputR sends the outputs for the right channel serially. After frame is detected high, outputR is set on the ***rising*** edge of sclk until all 40 bits are sent. The LSB is sent first and the MSB is sent last.

2.2 MSDAP State Controller module

To facilitate proper operation, the state controller provides control signals to other modules and receives feedback to determine changes in a Finite State Machine. The FSM states, outputs, and implementation specific conditions for state changes are shown below in Figure 2a. The block diagram is shown below in Figure 2b. The module pinouts are defined below in Table 2.

The State Controller employs counters to track events and provide correct timing:

- Sleep Counter: When an input of all zeros is received, the sleep counter is clocked. The FSM goes into sleep mode after this counter reaches a threshold of 800.
- Clear Counter: While in INITIALIZE or CLEARING, this counter is clocked with sclk until the memories are cleared.
- Input Counter: While receiving a new input, this counter is clocked on each dclk. The input is finished after 16 bits.
- Output Counter: When an output is ready to be sent, this counter is clocked with sclk. The output finishes after 40 bits.

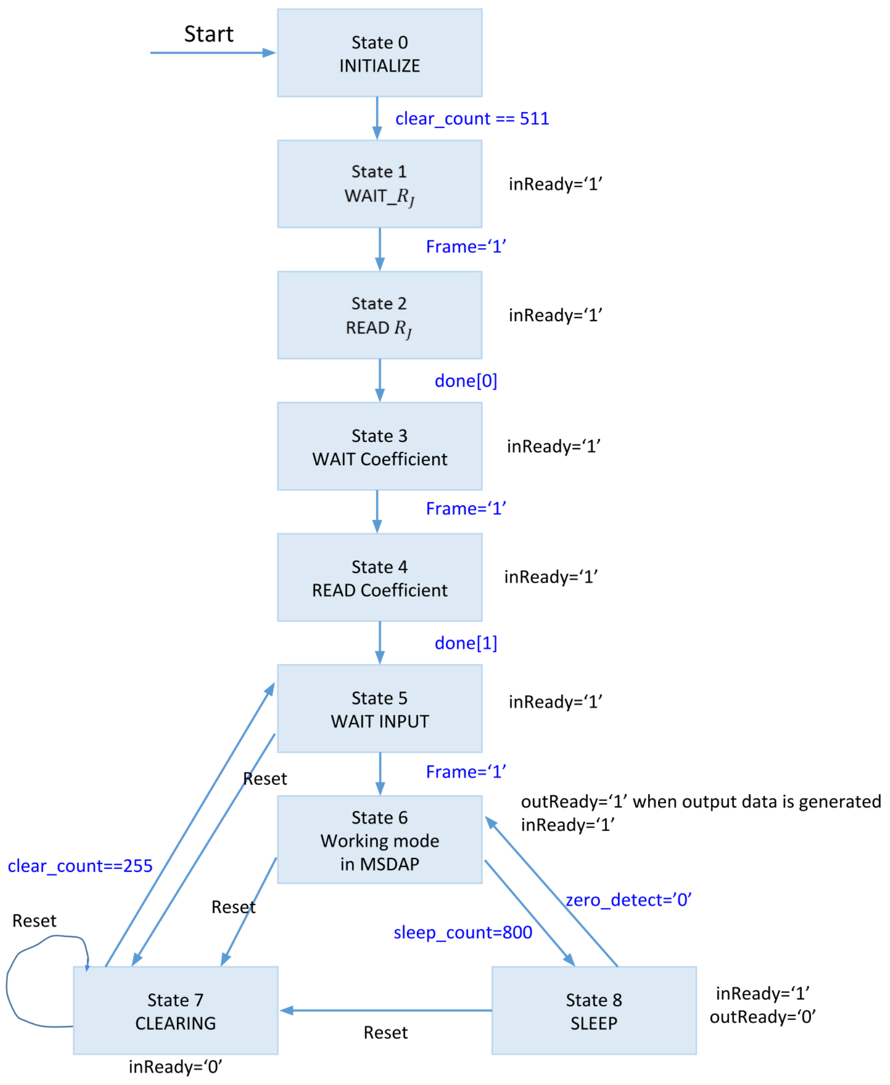


Figure 2a: Finite State Machine (FSM)

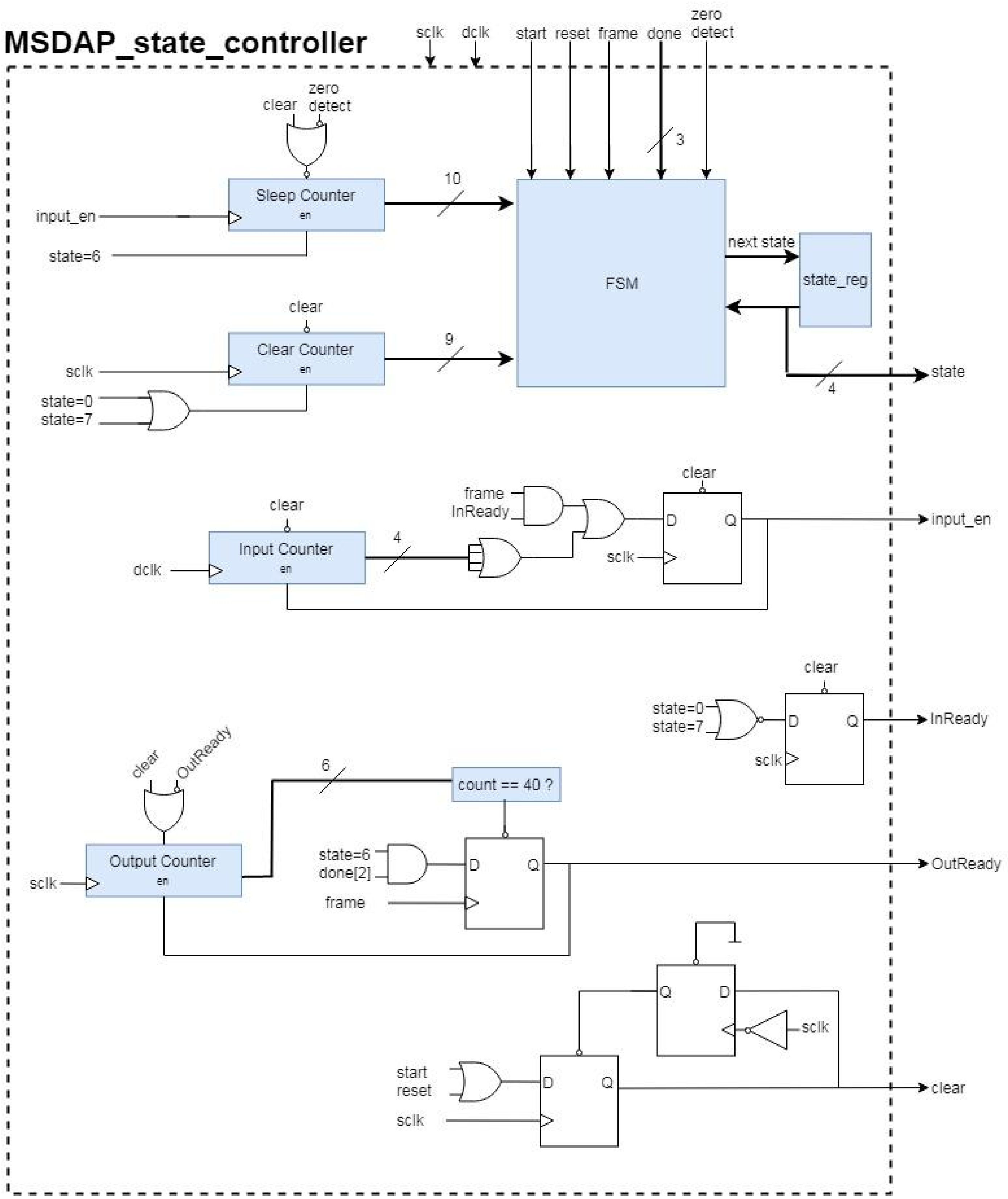


Figure 2b: State Controller

Table 2: State Controller Pins

Pin Name	Type	Size	Description
sclk	Input	1	25 MHz system clock. Used for data flow, computation, and sending output.
dclk	Input	1	768kHz data clock. Used to read in new input data.
start	Input	1	If start is high, the FSM enters INITIALIZE state. Asynchronous with sclk and dclk.
reset	Input	1	If reset is set high, FSM enters CLEARING state. Used to reset all registers and clear previous input data (except Rj and coefficient values). Asynchronous with sclk or dclk.
frame	Input	1	Indicates the start of a new input data sequence. Aligned with falling edge of dclk. On the rising edge of frame the state controller does the following: ● Sets input_en high to indicate new data should be accepted. ● If a computation has been completed (indicated by the "done[2]" signal), set outReady high to indicate that the output should be sent.
done	Input	3	Feedback signals from the data manager modules. ● done[0] indicates that all 16 rj values have been read. ● done[1] indicates that all 512 coefficient values have been read. ● done[2] indicates that a computation has been completed.
zero_detect	Input	1	Indicates the previous input was all zeros. Used to increment the sleep counter.
input_en	Output	1	Control signal to the input S2P modules. Indicates that input should be accepted. Input_en is set high when a frame is detected, and set low after 16 dclk cycles.
clear	Output	1	Control signal to indicate that all registers should be cleared. Clear is set high for 1 clock cycle when entering state 0 (INITIALIZE) or state 7 (CLEARING).
inReady	Output	1	InReady indicates the chip is able to accept new input data. The State Controller sets inReady low when in state 0 (INITIALIZE) or state 7 (CLEARING). Otherwise, it is set high.
outReady	Output	1	The State Controller sets outReady high on the rising edge of frame, but only after a computation has completed (indicated by the "done[2]" signal). After 40 sclk cycles are counted, outReady is reset to 0.
state	Output	4	Indicates the current state to the Data Manager modules. 4 bits are needed to store 9 unique states.

2.3 Serial-to-Parallel input module

To receive new input data from a single pin, the Serial-to-Parallel (S2P) module is used to parallelize into a 16-bit word. It consists of a shift register that clocks in a new bit on each dclk (LSB first) but only when the input_enable signal from the controller is set high. After 16 clocks, each bit has shifted into the correct place and is sent on to the Data Manager module as a 16-bit data word. The S2P module also provides a status signal “new_data_ready” to the Data Manager which indicates that the next 16-bit data word is ready to be written to memory. When all 16 bits in the shift register are zeroes, the “zero_detect” signal is set high as feedback to the controller, which counts all zero inputs for sleep mode.

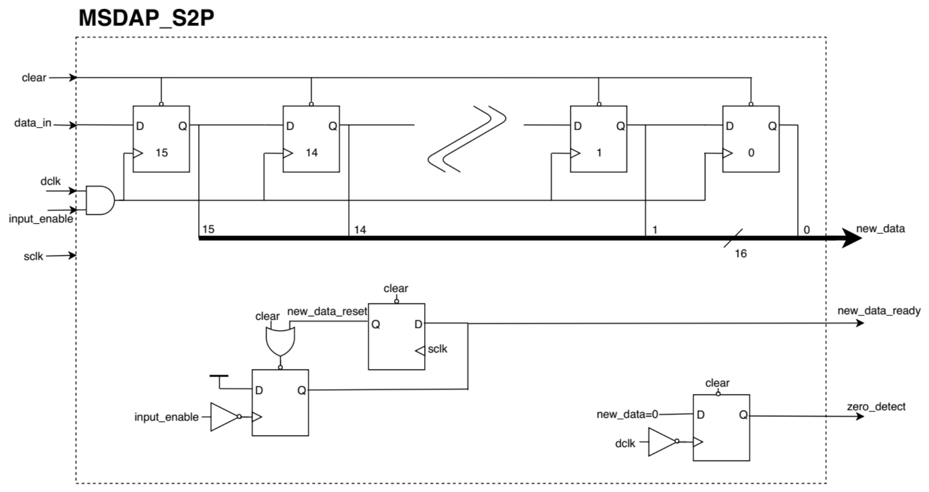


Figure 3: Input S2P Module

Table 3: Input S2P Module Pins

Pin Name	**Type**	**Size**	**Description**
sclk	Input	1	25 MHz system clock. Used for setting status signals.
dclk	Input	1	768 kHz data clock. Used to read in new input data.
input_enable	Input	1	Indicates that input should be accepted. Input_en is set high when a frame is detected, and then set low after 16 dclk cycles.
data_in	Input	1	The primary input bit for the data path. Received serially on ***rising*** edges of dclk.
clear	Input	1	Used to reset all registers.
input_read	Output	1	Status signal from the data_manager which indicates that the new input data has been read. Input_read is updated on the falling edge of sclk and stays high for 1 clock cycle.
new_data	Output	16	The parallelized 16 bit data after being fully read in to be sent to the data_manager. new_data is guaranteed to be updated when new_data_ready is high.
new_data_ready	Output	1	Control signal to Data Manager. Indicates that a new data word is ready to be stored in memory. Updated on sclk.
zero_detect	Output	1	Status signal that indicates the input data is all zeros. Updated on sclk.

2.4 Data Manager module

The Data Manager contains memories for RJ, coefficients, and input data. The memories are written to and read from depending on the current state. It uses multiple counters to move through memory addresses. During computation, previous input data stored in memory are accessed based on the coefficient values and sent to the ALU module.

- RJ memory:
    - During state 0 (INITIALIZE) the RJ memory is cleared by writing 0 to all addresses.
    - During state 2 (READ_RJ) the RJ memory is written to. After 16 RJ values are written, the “rj_done” signal (done[0]) is set high as feedback to the State Controller.
    - During state 6 (WORKING) the RJ memory is read to determine how many coefficients should be used. After the “coeffs_processed” counter reaches this RJ value, the RJ memory address is incremented and the “shift” signal (op[1]) is set high.
- Coefficient memory:
    - During state 0 (INITIALIZE) the memory is cleared by writing 0 to all addresses.
    - During state 4 (READ_COEFF) the coefficient memory is written to. After 512 coefficient values are written, the “coeff_done” signal (done[1]) is set high as feedback to the State Controller.
    - During state 6 (WORKING), the coefficient memory is read. The coefficient address is incremented on every sclk so long as “compute_en” is high.
- Data memory:
    - During state 0 (INITIALIZE) and state 7 (CLEARING) the data memory is cleared by writing 0 to all addresses.
    - During state 6 (WORKING), the data memory is read and sent to the ALU. The address to be read is determined by (n-1-k) where “n” is the count of data inputs, and “k” is the next coefficient value. All bits of the coefficient “k” are inverted to perform subtraction. With a carry-in of 0, the extra -1 is automatically included in the operation.
    - When a new input word is ready, the data memory at address “n” is written to.
- The MSB of the coefficient determines addition or subtraction. This “add/sub” bit is sent as “op[0]” to the ALU. The “shift” operation bit is sent as “op[1]”.
- Computations should only occur after an input has been received. The “input_received” signal is set high when an input data word has been written to memory.
- On the rising edge of a frame after the “input_received” signal is high, the “compute_en” signal is set high to indicate that memory should be read and that computations should occur in the ALU
- After a computation has been completed, the “compute_en” signal is reset to low and the “compute_done” signal (done[2]) is set high as feedback to the controller.

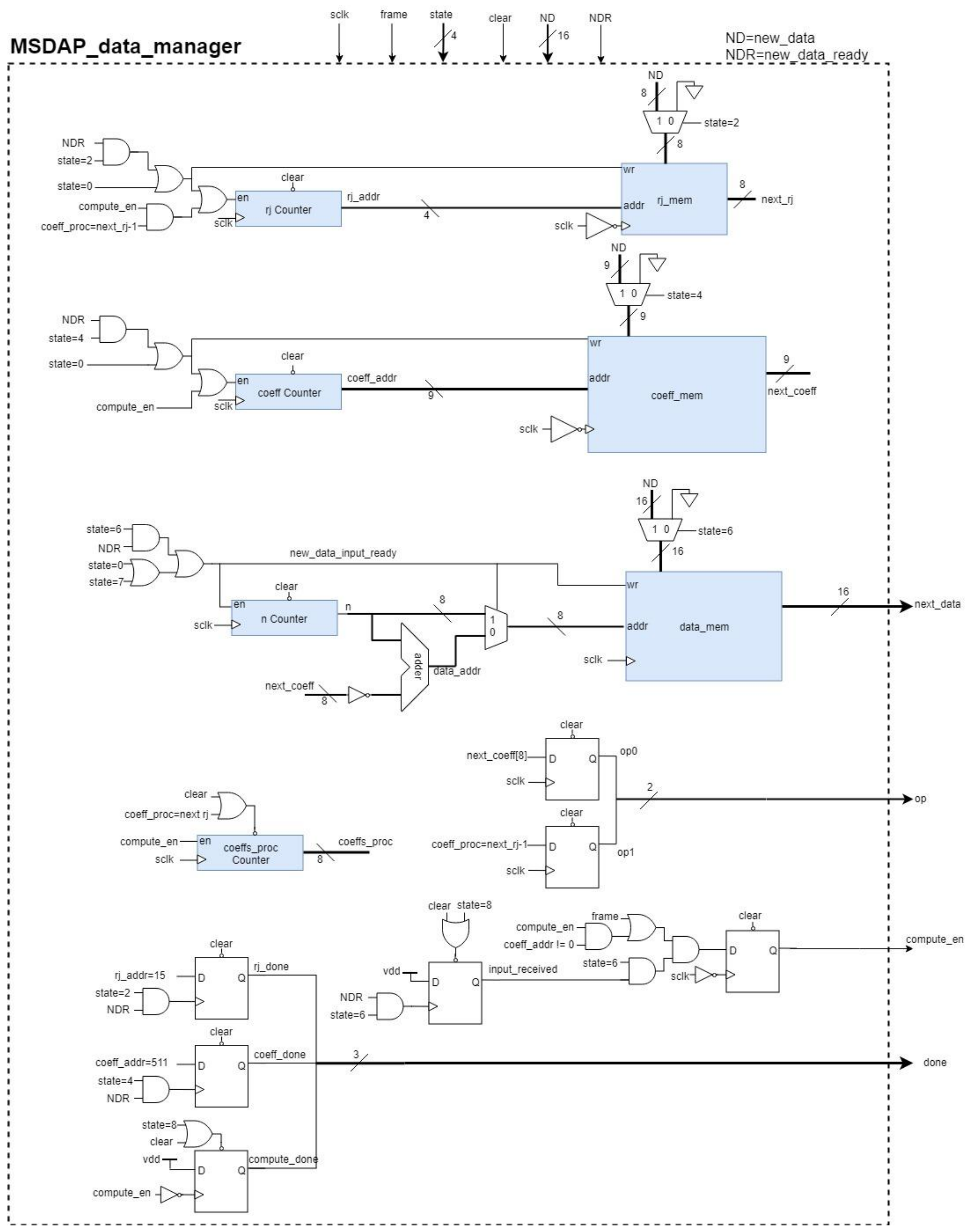


Figure 4: Data Manager

Table 4: Data Manager Pins

Pin Name	**Type**	**Size**	**Description**
sclk	Input	1	25 MHz system clock.
frame	Input	1	Indicates the start of a new data input cycle. Also used to align computations and outputs.
state	Input	4	Used by data manager to determine which memories to read or write to. ● In READ_RJ state, the input data is written to the RJ memory. ● In READ_COEFF state, the input data is written to the coeff memory. ● In WORKING state, the input data is written to the data memory. ● During WORKING state, the values stored in memories must be read and sent to the ALU in the proper sequence.
new_data	Input	16	The parallelized 16 bit data to be stored in memory.
new_data_ready	Input	1	Control signal which indicates that a new data word is ready.
clear	Input	1	Used to reset all registers.
input_read	Output	1	Status signal sent back to the S2P module which indicates that the new input data has been read. Input_read is updated on the falling edge of sclk and stays high for 1 clock cycle.
done	Output	3	Feedback signals to the state controller: ● done[0] indicates that all 16 rj values have been read. ● done[1] indicates that all 512 coefficient values have been read. ● done[2] indicates that a computation has been completed.
compute_en	Output	1	Control signal which indicates the ALU should perform the computation.
op	Output	2	Indicates the operation the ALU should perform: ● 00: add ● 01: sub ● 10: add and shift ● 11: sub and shift
next_data	Output	16	The data from memory which is to be used in the next computation step.

2.5 ALU module

The adding and shifting used to compute the convolution occurs in the ALU module. The following combinational logic is performed during each sclk cycle:

1) The 16-bit data is received from the Data Manager.
2) Each bit of data is XOR'd with the add/sub signal "op[0]". The XOR gate acts as a conditional inverter which only inverts for 2's complement subtraction when op[0] = 1.
3) The MSB of the data is used as the sign extend to create 24 bits.
4) These 24 bits are added to the 24 MSBs of the Accumulator register. Op[0] is used as the carry-in to correctly perform 2's complement subtraction.
5) If the "shift" signal op[1] is high, the sum is shifted 1 bit right. In this way, an add and a shift can be performed on the same clock cycle. Since it is an arithmetic shift, the sign bit must be maintained after the shift.
6) The Accumulator register is clocked on falling sclk, but only when the "compute_en" control signal is high.

When the "compute_en" signal goes low, it indicates that the computation is complete, and the 40-bit result in the Accumulator is clocked into the Result register.

Due to our ALU implementation, we are able to reduce the clock speed from 26.88MHz down to 25 MHz. Since the ALU is capable of performing both an addition and shift operation in the same clock cycle, only 512 clocks are necessary to complete the computation (one for each coefficient). The required 16 shifts are "folded" into the add operations and do not need a seperate clock cycle. The computation must complete within 16 dclk cycles:

$$\frac{16}{768kHz} = 20.833us\ per\ computation$$

Since we require a minimum of 512 clock cycles to complete the computation, we get a minimum clock frequency of:

$$\frac{20.833us}{512} = 40.69ns = 24.576\text{MHz}$$

Keeping a few clock cycles of safety margin, we chose sclk at 25MHz. This gives 520 sclk cycles per computation.

Pinouts for the ALU module are shown below in Table 5. The ALU block diagram is in Figure 5.

Table 5: ALU Pins

Pin Name	Type	Size	Description
sclk	Input	1	25 MHz system clock. Computation occurs on the ***falling*** edge of sclk.
compute_en	Input	1	Control signal which indicates the ALU should perform the computation.
clear	Input	1	Used to reset all registers.
op	Input	2	Control signal which indicates the operation the ALU should perform. Updated on the falling edge of sclk. ● 00: add ● 01: sub ● 10: add and shift ● 11: sub and shift
next_data	Input	16	The data from memory which is to be used in the next computation step. Updated on the falling edge of sclk.
result	Output	40	The 40-bit result of the computation. Updated on the falling edge of compute_en.

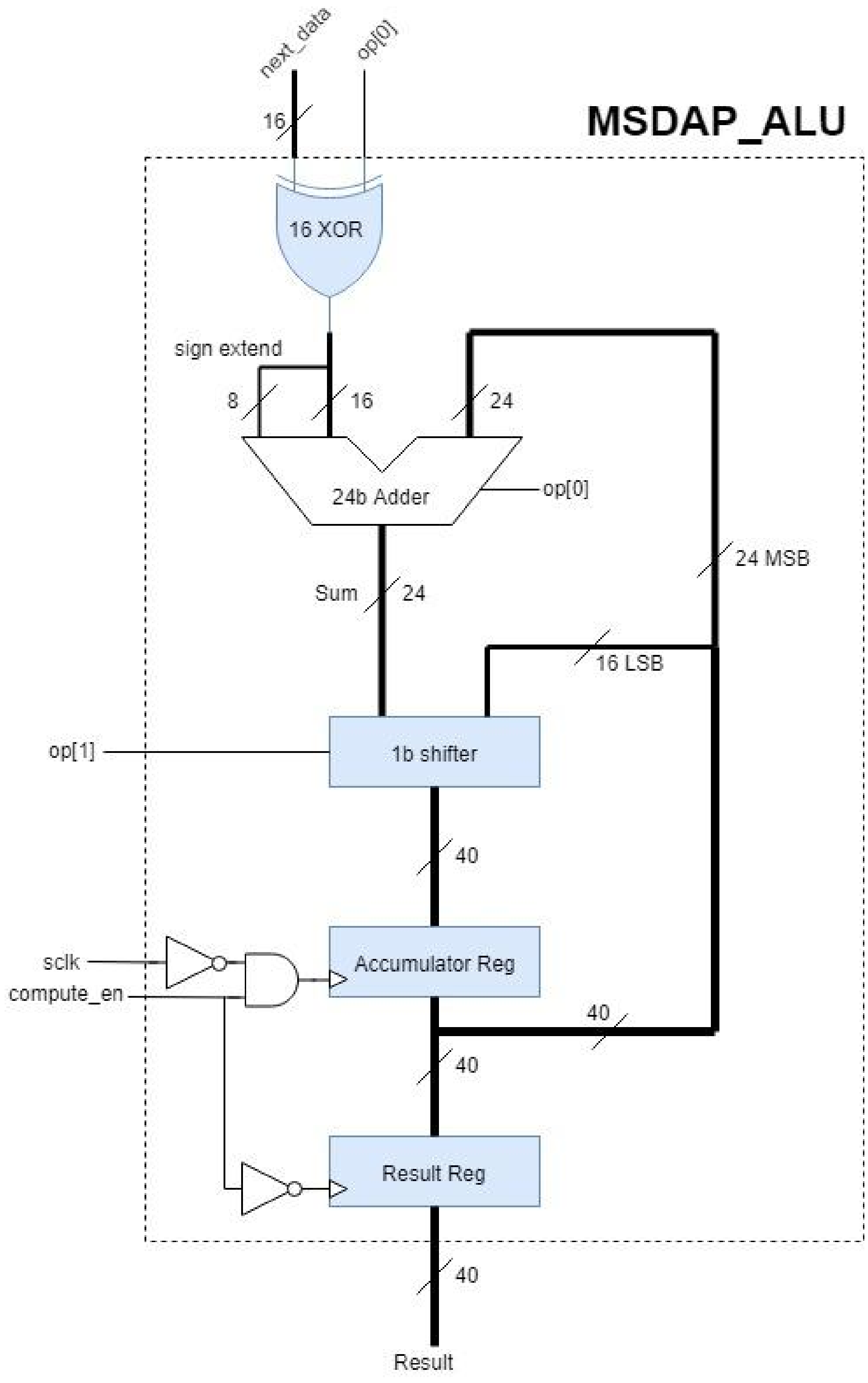


Figure 5: ALU Module

2.6 Parallel-to-Serial output module

Since the result of the computation is 40-bits, it must be serialized in the Parallel-to-Serial (P2S) module to be output on a single pin. The P2S module consists of a 40-bit shift register which can be loaded with a 40-bit word.

1) The outReady signal is set high by the State Controller to indicate that a new 40-bit data word is ready to be sent out. This triggers the "load" signal to be set high.

2) When "load" is high, the 40-bit result is loaded into the shift register on the first rising edge of sclk after outReady is set high. On the same clock, "load" is reset to low so that it will act as a normal shift register.

3) For as long as outReady stays high, a gated clock "gclk" (synchronous with sclk) will clock the shift register so that all 40 bits are sent out serially.

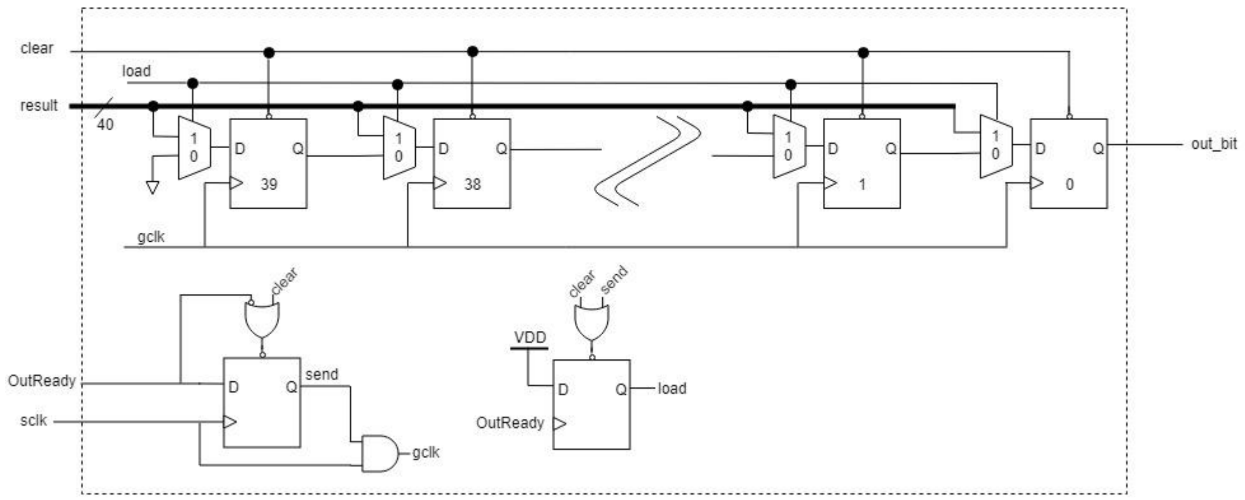


Figure 6: Output P2S Module

Table 6: Output P2S Module Pins

Pin Name	Type	Size	Description
sclk	Input	1	25 MHz system clock. Used for gated clock (gclk) to shift out bits.
result	Input	40	The 40 bit result of the computation received from the ALU. The result is updated at the end of the computation.
outReady	Input	1	Control signal from the FSM to indicate that an output should be sent. Aligned with rising edge of frame.
clear	Input	1	Used to reset all registers.
out_bit	Output	16	The primary output bit for the datapath. Updated on ***rising*** edge of sclk.

SECTION 3: RTL Verification and Synthesis

In this section, the RTL code waveforms are presented. This verilog code is synthesized using Synopsys Design Compiler, and the reports are examined. The synthesized netlist is also simulated, and waveforms are examined to show functionality. All verilog code and outputs from the Design Compiler synthesis can be found in the attachments.

3.1 RTL Simulation

The following figures illustrate the functionality of the RTL simulation:

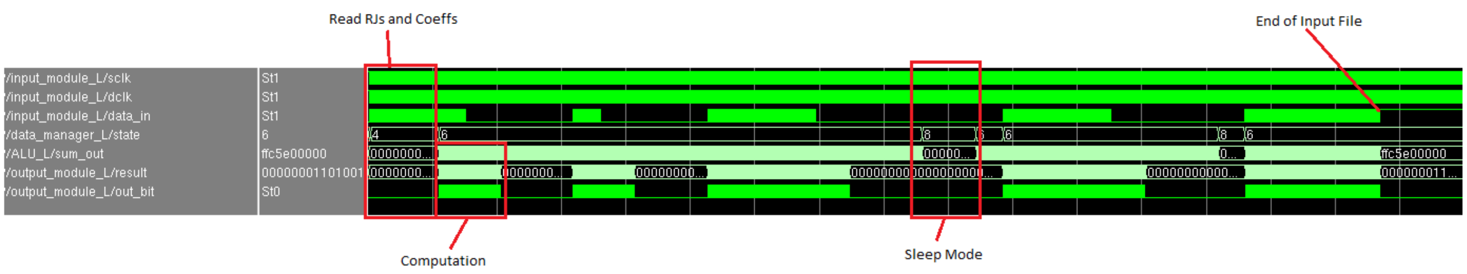
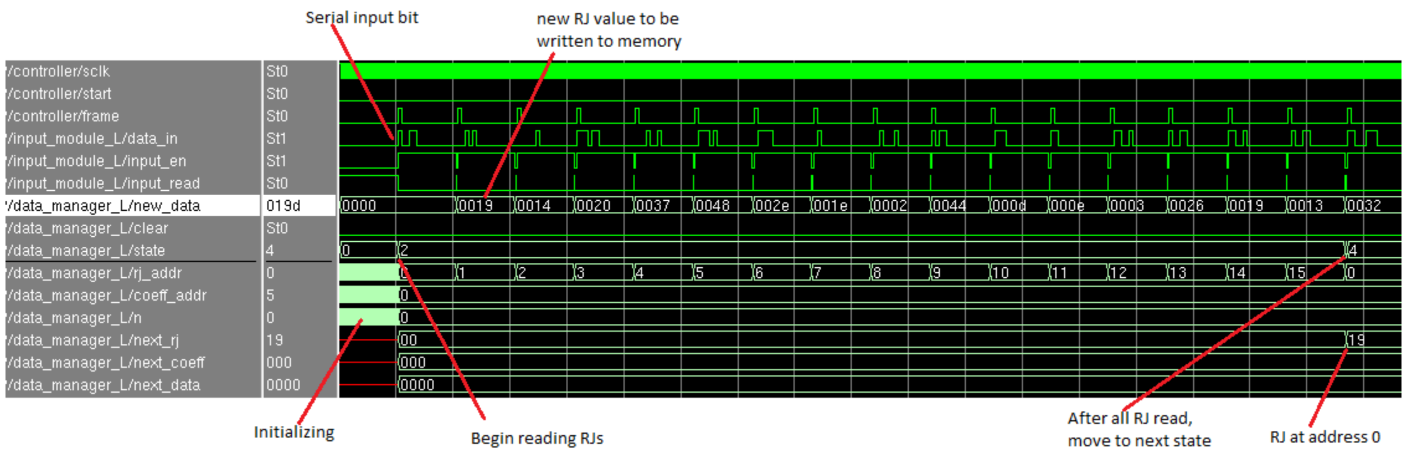


Figure 7: Overview of MSDAP Simulation

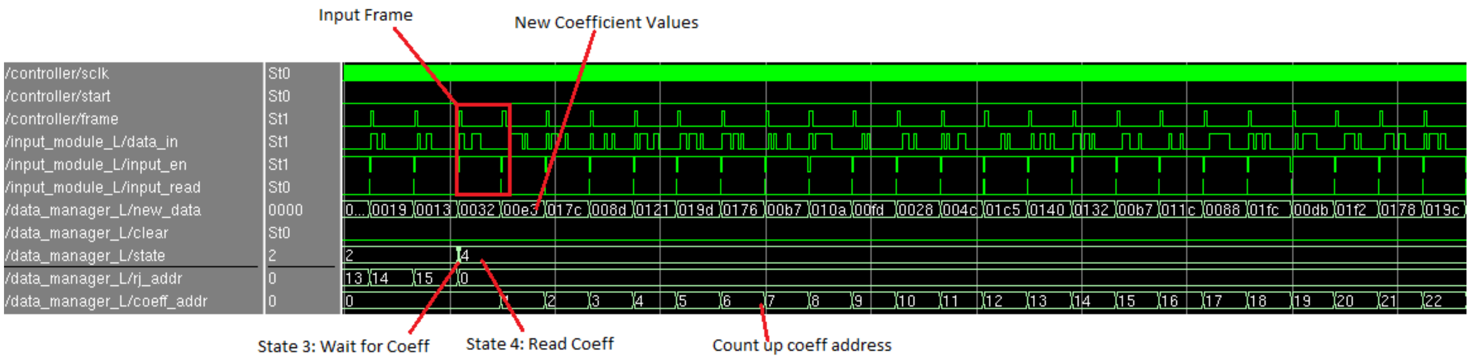
The overview of the full MSDAP operation. After reading in RJ and Coefficient values, most of the time is spent in WORKING mode (state 6). When all-zero inputs are received for 800 frames, it enters sleep mode (state 8).



Serial input bit
new RJ value to be written to memory
/controller/sclk St0
/controller/start St0
/controller/frame St0
/input_module_L/data_in St1
/input_module_L/input_en St1
/input_module_L/input_read St0
/data_manager_L/new_data 019d
/data_manager_L/clear St0
/data_manager_L/state 4
/data_manager_L/rj_addr 0
/data_manager_L/coeff_addr 5
/data_manager_L/n 0
/data_manager_L/next_rj 19
/data_manager_L/next_coeff 000
/data_manager_L/next_data 0000
Initializing
Begin reading RJs
After all RJ read, move to next state
RJ at address 0

Figure 8: Read RJ Values

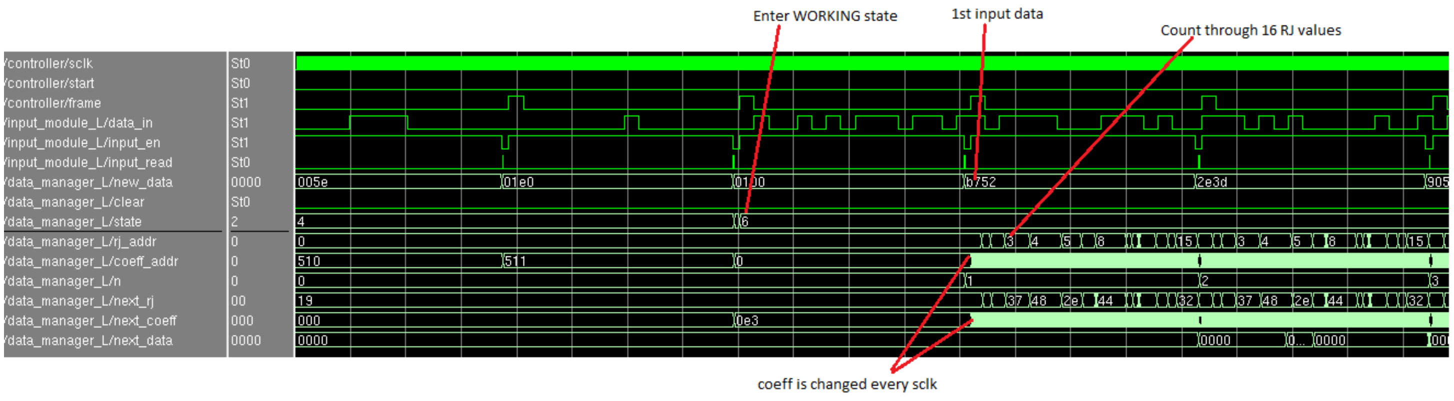
Upon receiving the "start" signal, the MSDAP initializes (state 0) by clearing all DFFs, and writing zeros to all memories. Since there are 512 coefficients, this process requires 512 sclks to clear all memories. On the first "frame" received, it begins receiving RJ values (state 2). The input bit is received by the S2P module, and after 16 bits are received the "new_data" is written to memory. After all 16 RJs have been read in, the MSDAP moves on to reading coefficient values (state 4).



Input Frame
New Coefficient Values
/controller/sclk St0
/controller/start St0
/controller/frame St1
/input_module_L/data_in St1
/input_module_L/input_en St1
/input_module_L/input_read St0
/data_manager_L/new_data 0000
/data_manager_L/clear St0
/data_manager_L/state 2
/data_manager_L/rj_addr 0
/data_manager_L/coeff_addr 0
State 3: Wait for Coeff
State 4: Read Coeff
Count up coeff address

Figure 9: Read Coefficient Values

In state 4, the coefficients are read in. These are 8-bit values with a 9th sign bit. After all 512 coefficients are read, the MSDAP moves to WORKING mode (state 6).



Enter WORKING state
1st input data
Count through 16 RJ values
/controller/sclk St0
/controller/start St0
/controller/frame St1
/input_module_L/data_in St1
/input_module_L/input_en St1
/input_module_L/input_read St0
/data_manager_L/new_data 0000
/data_manager_L/clear St0
/data_manager_L/state 2
/data_manager_L/rj_addr 0
/data_manager_L/coeff_addr 0
/data_manager_L/n 0
/data_manager_L/next_rj 00
/data_manager_L/next_coeff 000
/data_manager_L/next_data 0000
coeff is changed every sclk

Figure 10: Computation Start

After entering WORKING mode, inputs are received and written to data memory. The first computation begins only after the first 16-bit input has been received. The next input, computation, and output result all occur synchronous with “frame”.

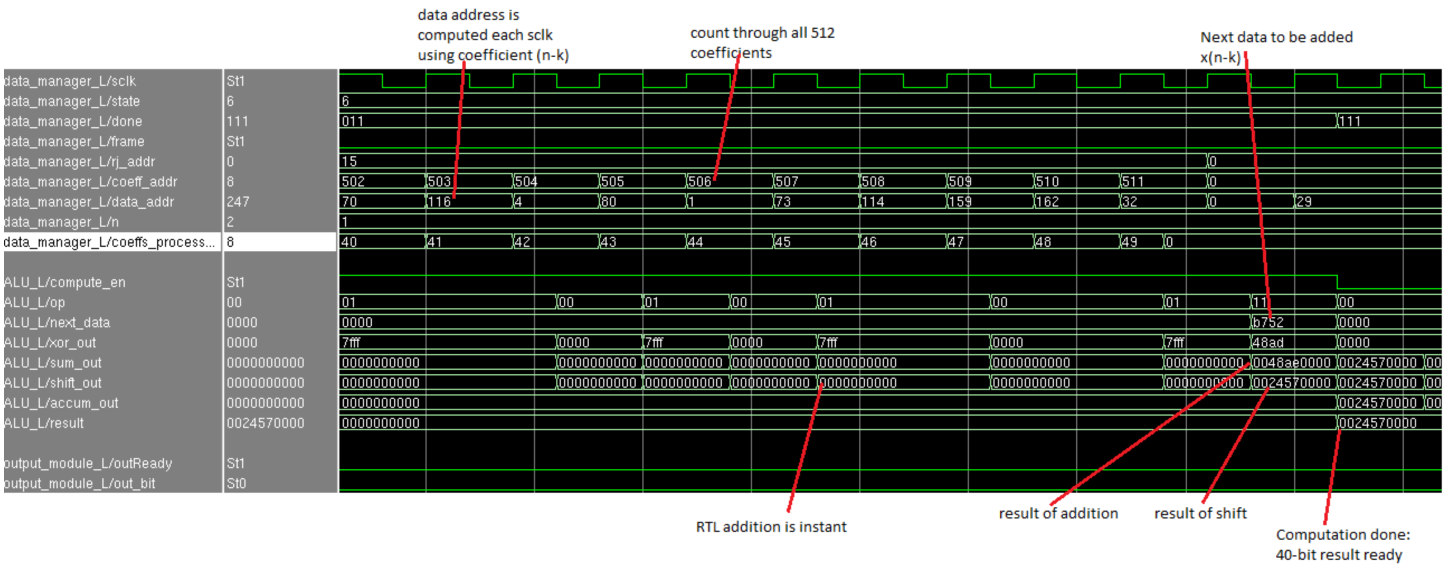


Figure 11: Computation Detail

The end of a computation cycle is shown above. The data can be seen being operated on and moving through registers. In this case, the data "b752" is subtracted from the running sum "000000" to obtain "0048ae" and then shifted to obtain the result "0024570000". Since this is the last coefficient for this computation, the accumulation is clocked into the result register and the "done" signal is set high.

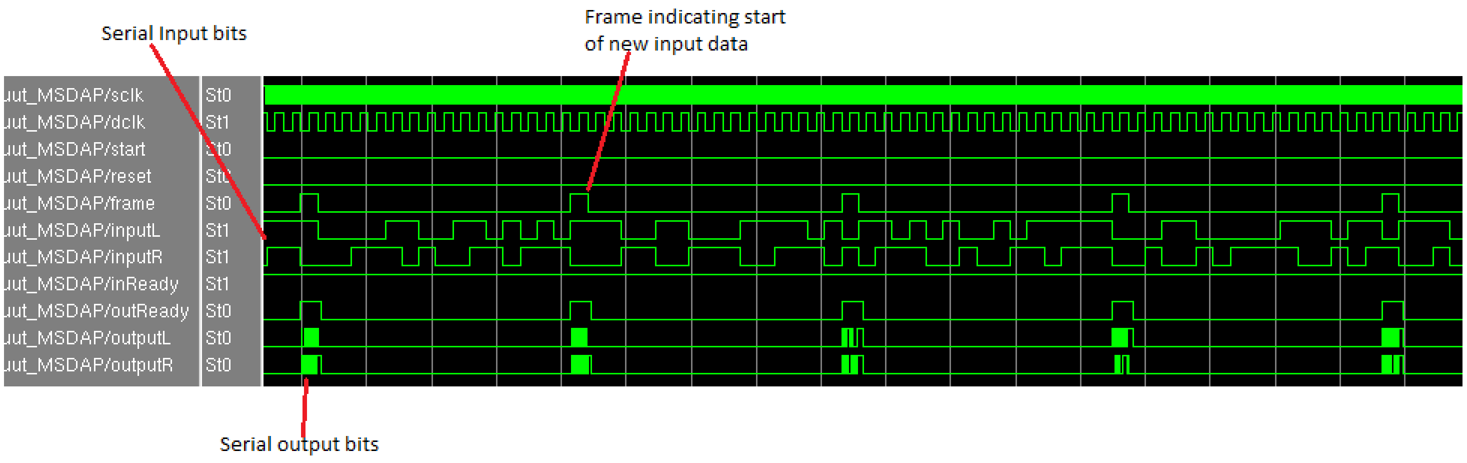


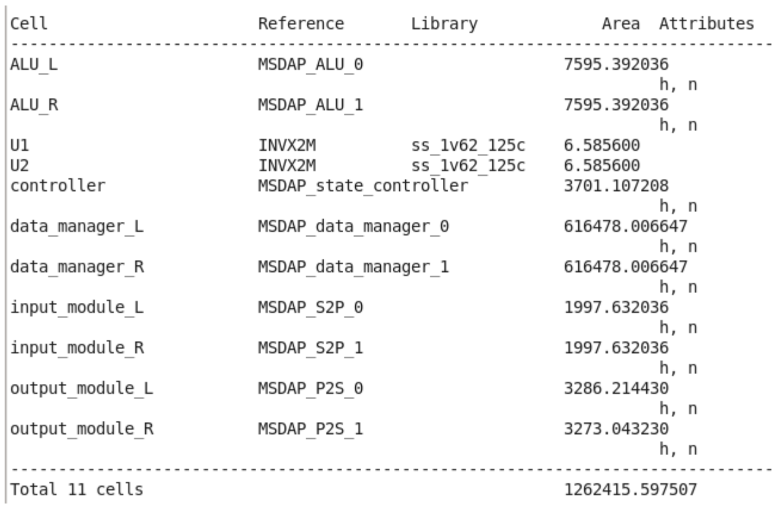
Figure 12: Computation seen from Top Level

The previous waveforms show internal signals. Figure 12 shows all the external inputs and outputs of the MSDAP during computation. On each frame, a new input is received on both the left and right data channels. InReady is held high during computation, since new inputs are being constantly accepted. At the beginning of each frame, the outReady signal goes high and 40-bits of output are sent out serially on both data out pins.

3.2 Synthesis with Design Compiler

After simulating and confirming the functionality of the RTL verilog code, the design is ready to be synthesized using a standard cell library. Synthesis realizes the design using real logic gates with physical size and characteristics. After loading the verilog RTL code and cell library in Synopsys Design Compiler, the design was successfully synthesized.

The cell report shows and block diagram shows the same block architecture after performing synthesis. The synthesis process added 2 inverters as a buffer for the “clear” signal. This is a reasonable addition because the “clear” signal has a large fanout to all DFFs. Schematics generated by Design Compiler for each sub-block can be found in Appendix A.



```
Cell                     Reference       Library           Area  Attributes
-------------------------------------------------------------------------------
ALU_L                    MSDAP_ALU_0                   7595.392036
                                                                 h, n
ALU_R                    MSDAP_ALU_1                   7595.392036
                                                                 h, n
U1                       INVX2M          ss_1v62_125c  6.585600
U2                       INVX2M          ss_1v62_125c  6.585600
controller               MSDAP_state_controller        3701.107208
                                                                 h, n
data_manager_L           MSDAP_data_manager_0          616478.006647
                                                                 h, n
data_manager_R           MSDAP_data_manager_1          616478.006647
                                                                 h, n
input_module_L           MSDAP_S2P_0                   1997.632036
                                                                 h, n
input_module_R           MSDAP_S2P_1                   1997.632036
                                                                 h, n
output_module_L          MSDAP_P2S_0                   3286.214430
                                                                 h, n
output_module_R          MSDAP_P2S_1                   3273.043230
                                                                 h, n
-------------------------------------------------------------------------------
Total 11 cells                                         1262415.597507
```

Figure 13: Synthesis cell report

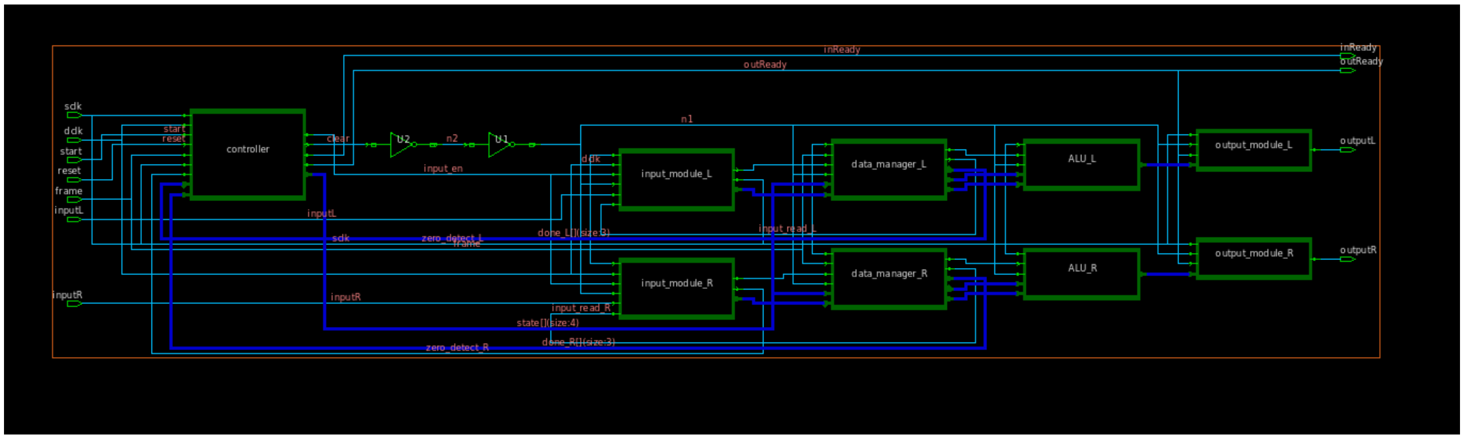
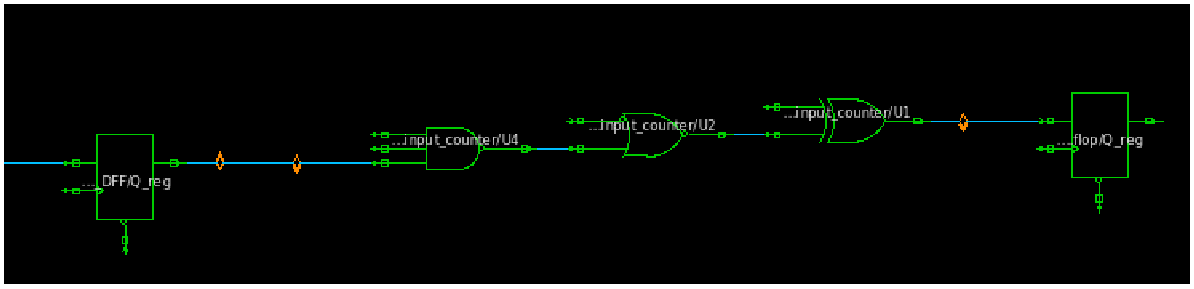
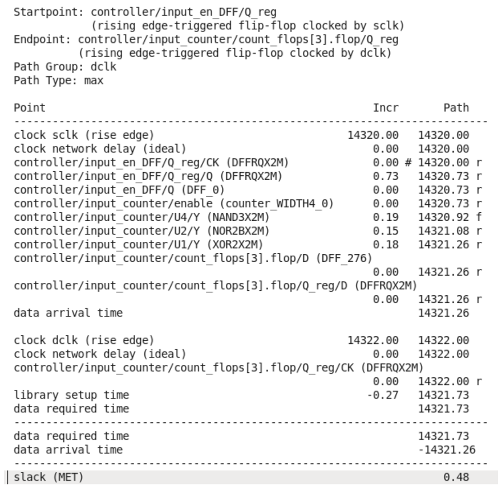
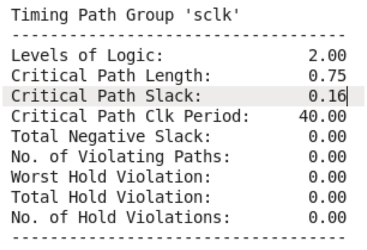


Figure 14: Synthesized block diagram

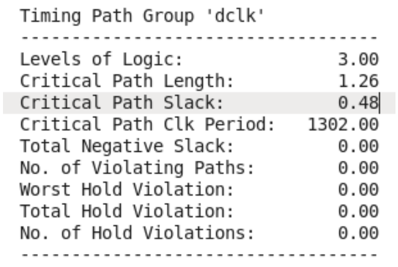




```
Startpoint: controller/input_en_DFF/Q_reg
            (rising edge-triggered flip-flop clocked by sclk)
Endpoint: controller/input_counter/count_flops[3].flop/Q_reg
          (rising edge-triggered flip-flop clocked by dclk)
Path Group: dclk
Path Type: max

Point                                                    Incr       Path
--------------------------------------------------------------------------
clock sclk (rise edge)                               14320.00   14320.00
clock network delay (ideal)                              0.00   14320.00
controller/input_en_DFF/Q_reg/CK (DFFRQX2M)              0.00 # 14320.00 r
controller/input_en_DFF/Q_reg/Q (DFFRQX2M)               0.73   14320.73 r
controller/input_en_DFF/Q (DFF_0)                        0.00   14320.73 r
controller/input_counter/enable (counter_WIDTH4_0)       0.00   14320.73 r
controller/input_counter/U4/Y (NAND3X2M)                 0.19   14320.92 f
controller/input_counter/U2/Y (NOR2BX2M)                 0.15   14321.08 r
controller/input_counter/U1/Y (XOR2X2M)                  0.18   14321.26 r
controller/input_counter/count_flops[3].flop/D (DFF_276)
                                                         0.00   14321.26 r
controller/input_counter/count_flops[3].flop/Q_reg/D (DFFRQX2M)
                                                         0.00   14321.26 r
data arrival time                                               14321.26

clock dclk (rise edge)                               14322.00   14322.00
clock network delay (ideal)                              0.00   14322.00
controller/input_counter/count_flops[3].flop/Q_reg/CK (DFFRQX2M)
                                                         0.00   14322.00 r
library setup time                                      -0.27   14321.73
data required time                                              14321.73
--------------------------------------------------------------------------
data required time                                              14321.73
data arrival time                                              -14321.26
--------------------------------------------------------------------------
slack (MET)                                                         0.48
```



```
Timing Path Group 'sclk'
-----------------------------------
Levels of Logic:               2.00
Critical Path Length:          0.75
Critical Path Slack:           0.16
Critical Path Clk Period:     40.00
Total Negative Slack:          0.00
No. of Violating Paths:        0.00
Worst Hold Violation:          0.00
Total Hold Violation:          0.00
No. of Hold Violations:        0.00
-----------------------------------
```



```
Timing Path Group 'dclk'
-----------------------------------
Levels of Logic:               3.00
Critical Path Length:          1.26
Critical Path Slack:           0.48
Critical Path Clk Period:   1302.00
Total Negative Slack:          0.00
No. of Violating Paths:        0.00
Worst Hold Violation:          0.00
Total Hold Violation:          0.00
No. of Hold Violations:        0.00
-----------------------------------
```

Figure 15: Critical Path

Timing estimates can be made after synthesis to ensure that the timing specification is met. The critical path for our synthesized design is shown above, as a signal propagates between two DFFs through three logic gates. The report shows only positive slack, indicating that no timing violations occur for either sclk or dclk critical paths. This gives us high confidence that the netlist is ready for physical design.

3.3 Gate-Level Netlist Simulation

The synthesized netlist was simulated using the same testbench. The following waveforms show that the netlist has the same functionality as the RTL verilog code.

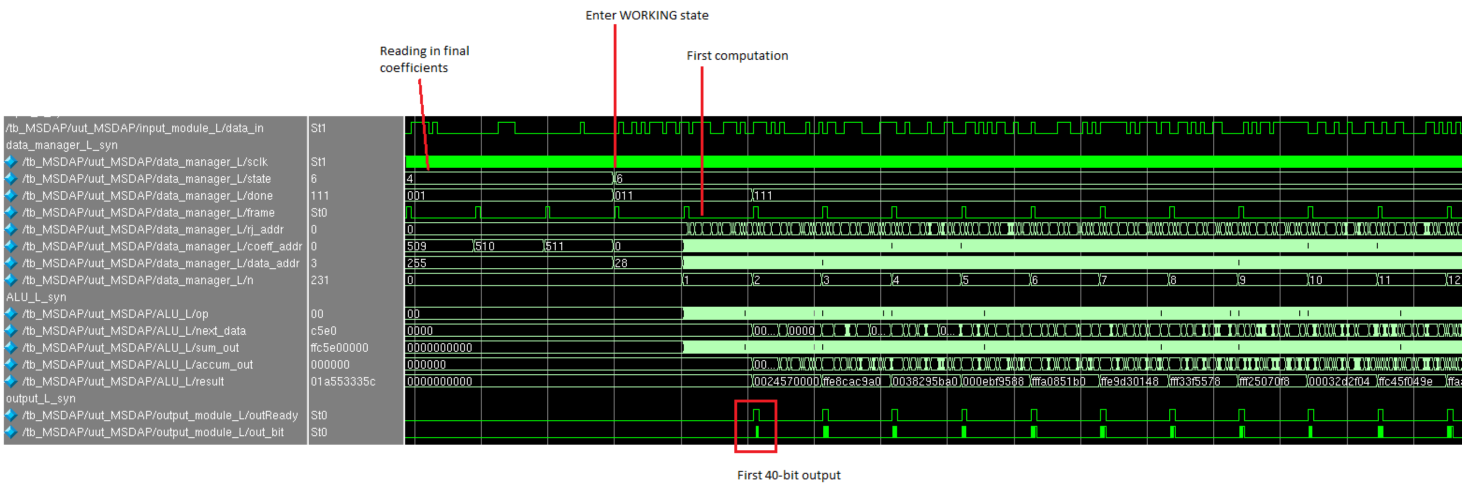

Enter WORKING state
Reading in final coefficients
First computation
First 40-bit output
/tb_MSDAP/uut_MSDAP/input_module_L/data_in
data_manager_L_syn
/tb_MSDAP/uut_MSDAP/data_manager_L/sclk
/tb_MSDAP/uut_MSDAP/data_manager_L/state
/tb_MSDAP/uut_MSDAP/data_manager_L/done
/tb_MSDAP/uut_MSDAP/data_manager_L/frame
/tb_MSDAP/uut_MSDAP/data_manager_L/rj_addr
/tb_MSDAP/uut_MSDAP/data_manager_L/coeff_addr
/tb_MSDAP/uut_MSDAP/data_manager_L/data_addr
/tb_MSDAP/uut_MSDAP/data_manager_L/n
ALU_L_syn
/tb_MSDAP/uut_MSDAP/ALU_L/op
/tb_MSDAP/uut_MSDAP/ALU_L/next_data
/tb_MSDAP/uut_MSDAP/ALU_L/sum_out
/tb_MSDAP/uut_MSDAP/ALU_L/accum_out
/tb_MSDAP/uut_MSDAP/ALU_L/result
output_L_syn
/tb_MSDAP/uut_MSDAP/output_module_L/outReady
/tb_MSDAP/uut_MSDAP/output_module_L/out_bit

Figure 16: Synthesized Computation

The synthesized design has the same signal data flow and results that were obtained using the RTL design.

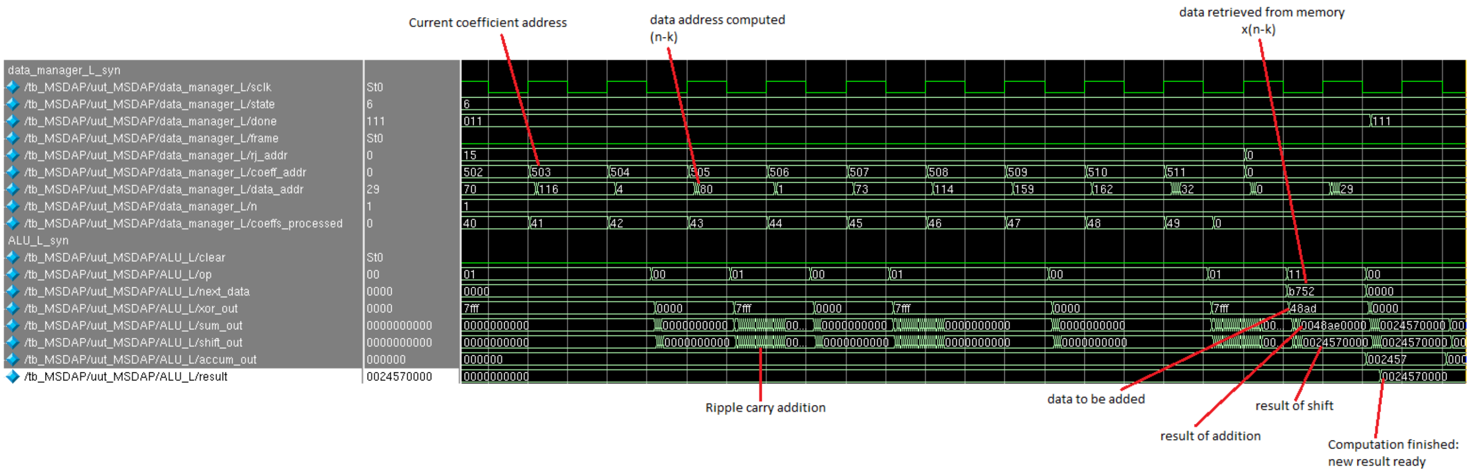

Current coefficient address
data address computed (n-k)
data retrieved from memory x(n-k)
Ripple carry addition
data to be added
result of shift
result of addition
Computation finished: new result ready
data_manager_L_syn
/tb_MSDAP/uut_MSDAP/data_manager_L/sclk
/tb_MSDAP/uut_MSDAP/data_manager_L/state
/tb_MSDAP/uut_MSDAP/data_manager_L/done
/tb_MSDAP/uut_MSDAP/data_manager_L/frame
/tb_MSDAP/uut_MSDAP/data_manager_L/rj_addr
/tb_MSDAP/uut_MSDAP/data_manager_L/coeff_addr
/tb_MSDAP/uut_MSDAP/data_manager_L/data_addr
/tb_MSDAP/uut_MSDAP/data_manager_L/n
/tb_MSDAP/uut_MSDAP/data_manager_L/coeffs_processed
ALU_L_syn
/tb_MSDAP/uut_MSDAP/ALU_L/clear
/tb_MSDAP/uut_MSDAP/ALU_L/op
/tb_MSDAP/uut_MSDAP/ALU_L/next_data
/tb_MSDAP/uut_MSDAP/ALU_L/xor_out
/tb_MSDAP/uut_MSDAP/ALU_L/sum_out
/tb_MSDAP/uut_MSDAP/ALU_L/shift_out
/tb_MSDAP/uut_MSDAP/ALU_L/accum_out
/tb_MSDAP/uut_MSDAP/ALU_L/result

Figure 17: Synthesized Computation Detail

This detailed view shows the end of the first computation cycle. Since the netlist uses delay estimates for gates, the operations do not occur instantaneously in this simulation like they do in the RTL verilog simulation. This can be seen as rapidly changing "sum_out" register as the ripple-carry adder operates on new inputs. This ripple carry addition is actually the worst case ripple, since it is adding all ones with a carry in of 1, so the carry must ripple through all 24 bits of the addition. It can be seen that this worst-case ripple still completes the addition operation within 1 cycle of sclk. Therefore, we have high confidence that all ALU operations will complete within 1 sclk.

The accumulator and result registers match the simulation of the RTL verilog, and the output files are identical. From this, we conclude that the synthesized netlist has the same correct functionality for computation.

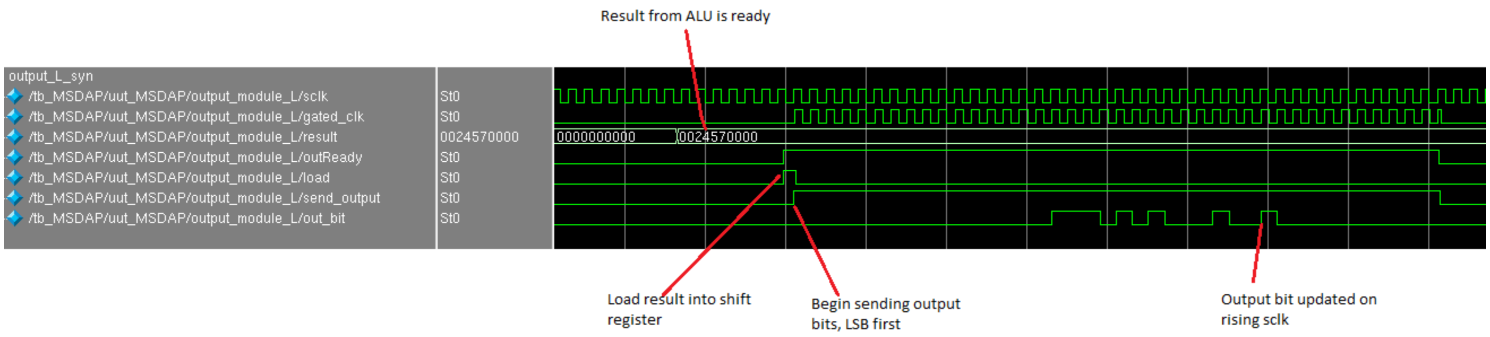

Result from ALU is ready
output_L_syn
/tb_MSDAP/uut_MSDAP/output_module_L/sclk St0
/tb_MSDAP/uut_MSDAP/output_module_L/gated_clk St0
/tb_MSDAP/uut_MSDAP/output_module_L/result 0024570000
/tb_MSDAP/uut_MSDAP/output_module_L/outReady St0
/tb_MSDAP/uut_MSDAP/output_module_L/load St0
/tb_MSDAP/uut_MSDAP/output_module_L/send_output St0
/tb_MSDAP/uut_MSDAP/output_module_L/out_bit St0
0000000000
0024570000
Load result into shift register
Begin sending output bits, LSB first
Output bit updated on rising sclk

Figure 18: Output Detail (synthesized)

The output module P2S signals are detailed here. It can be seen that the result from the ALU is shifted out serially, LSB first. At the start of a frame, the “outReady” signal goes high and it triggers the result to be loaded into the shift register. The next 40 sclks shift out 1 bit at a time until the output is complete and “outReady” is set back low.

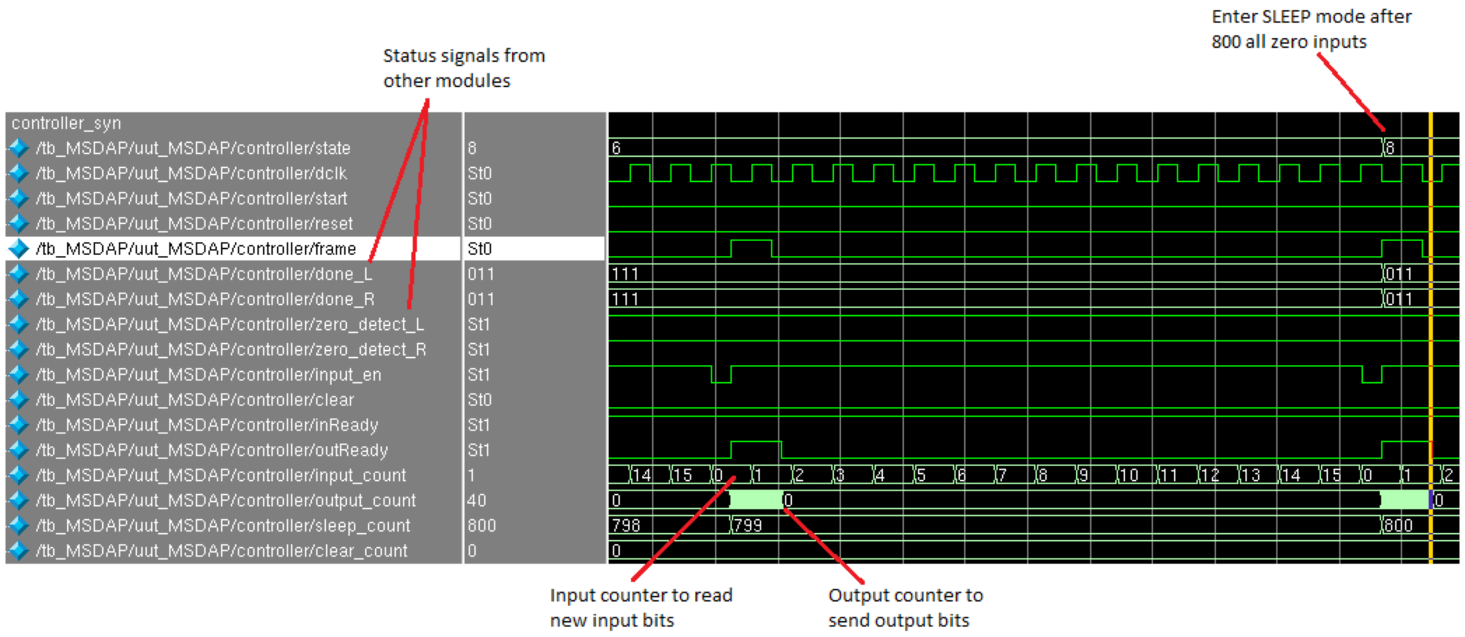
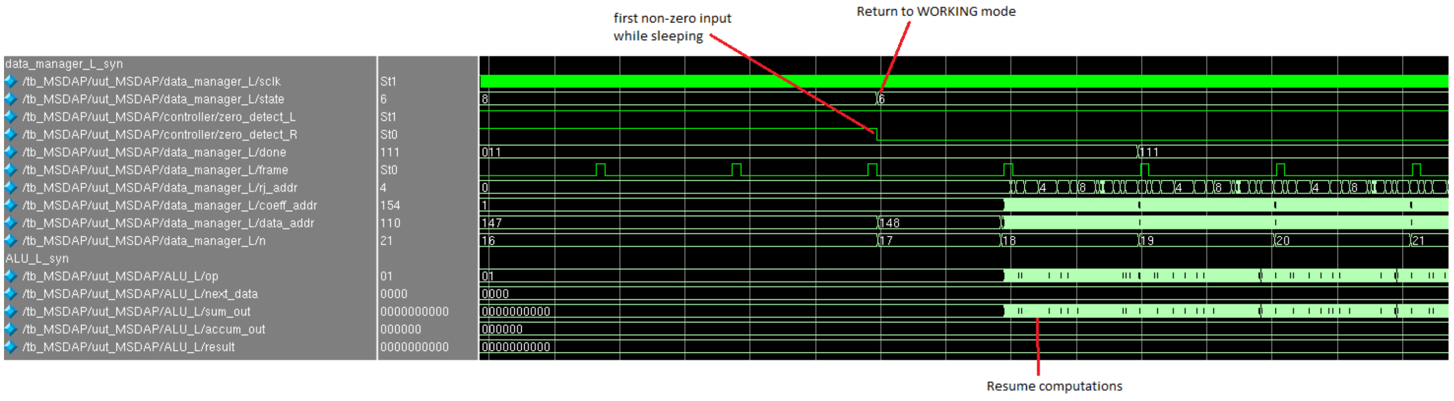

Enter SLEEP mode after 800 all zero inputs
Status signals from other modules
controller_syn
/tb_MSDAP/uut_MSDAP/controller/state 8
/tb_MSDAP/uut_MSDAP/controller/dclk St0
/tb_MSDAP/uut_MSDAP/controller/start St0
/tb_MSDAP/uut_MSDAP/controller/reset St0
/tb_MSDAP/uut_MSDAP/controller/frame St0
/tb_MSDAP/uut_MSDAP/controller/done_L 011
/tb_MSDAP/uut_MSDAP/controller/done_R 011
/tb_MSDAP/uut_MSDAP/controller/zero_detect_L St1
/tb_MSDAP/uut_MSDAP/controller/zero_detect_R St1
/tb_MSDAP/uut_MSDAP/controller/input_en St1
/tb_MSDAP/uut_MSDAP/controller/clear St0
/tb_MSDAP/uut_MSDAP/controller/inReady St1
/tb_MSDAP/uut_MSDAP/controller/outReady St1
/tb_MSDAP/uut_MSDAP/controller/input_count 1
/tb_MSDAP/uut_MSDAP/controller/output_count 40
/tb_MSDAP/uut_MSDAP/controller/sleep_count 800
/tb_MSDAP/uut_MSDAP/controller/clear_count 0
6 8
111 011
111 011
14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2
0 0 0
798 799 800
0
Input counter to read new input bits
Output counter to send output bits

Figure 19: Controller goes to sleep

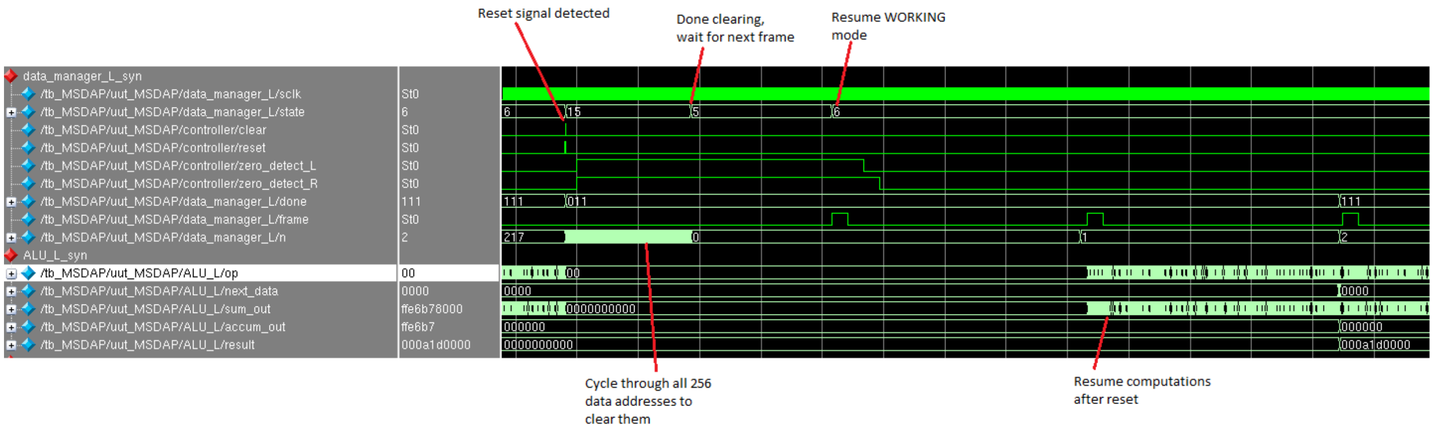
The sleep counter in the controller module is incremented whenever an input of all zeroes is received. Figure 19 shows the controller going to sleep after 800 of these inputs.



data_manager_L_syn	
/tb_MSDAP/uut_MSDAP/data_manager_L/sclk	St1
/tb_MSDAP/uut_MSDAP/data_manager_L/state	6
/tb_MSDAP/uut_MSDAP/controller/zero_detect_L	St1
/tb_MSDAP/uut_MSDAP/controller/zero_detect_R	St0
/tb_MSDAP/uut_MSDAP/data_manager_L/done	111
/tb_MSDAP/uut_MSDAP/data_manager_L/frame	St0
/tb_MSDAP/uut_MSDAP/data_manager_L/rj_addr	4
/tb_MSDAP/uut_MSDAP/data_manager_L/coeff_addr	154
/tb_MSDAP/uut_MSDAP/data_manager_L/data_addr	110
/tb_MSDAP/uut_MSDAP/data_manager_L/n	21
ALU_L_syn	
/tb_MSDAP/uut_MSDAP/ALU_L/op	01
/tb_MSDAP/uut_MSDAP/ALU_L/next_data	0000
/tb_MSDAP/uut_MSDAP/ALU_L/sum_out	0000000000
/tb_MSDAP/uut_MSDAP/ALU_L/accum_out	000000
/tb_MSDAP/uut_MSDAP/ALU_L/result	0000000000

Figure 20: Controller wakes up

When a non-zero input is detected (indicated by the "zero_detect" signals) the controller wakes up and returns to WORKING state. The computation only resumes after the first non-zero input has been fully received.


Reset signal detected
Done clearing, wait for next frame
Resume WORKING mode
Cycle through all 256 data addresses to clear them
Resume computations after reset

data_manager_L_syn	
/tb_MSDAP/uut_MSDAP/data_manager_L/sclk	St0
/tb_MSDAP/uut_MSDAP/data_manager_L/state	6
/tb_MSDAP/uut_MSDAP/controller/clear	St0
/tb_MSDAP/uut_MSDAP/controller/reset	St0
/tb_MSDAP/uut_MSDAP/controller/zero_detect_L	St0
/tb_MSDAP/uut_MSDAP/controller/zero_detect_R	St0
/tb_MSDAP/uut_MSDAP/data_manager_L/done	111
/tb_MSDAP/uut_MSDAP/data_manager_L/frame	St0
/tb_MSDAP/uut_MSDAP/data_manager_L/n	2
ALU_L_syn	
/tb_MSDAP/uut_MSDAP/ALU_L/op	00
/tb_MSDAP/uut_MSDAP/ALU_L/next_data	0000
/tb_MSDAP/uut_MSDAP/ALU_L/sum_out	ffe6b78000
/tb_MSDAP/uut_MSDAP/ALU_L/accum_out	ffe6b7
/tb_MSDAP/uut_MSDAP/ALU_L/result	000a1d0000

Figure 21: Controller Reset

When the primary input "reset" is detected, the controller moves to the CLEARING state. Originally CLEARING state was represented by state 7 (binary "0111") but this caused issues when transitioning into the state. State 15 was chosen to represent this (binary "1111") because it does not cause transition errors in the synthesized netlist simulation.

While in the CLEARING state, the 256 addresses of input data memory are cleared. RJ and coefficient values remain untouched. Therefore, only 256 sclk cycles are needed during this state before returning to state 5, and then state 6 at the next frame.

SECTION 4: PHYSICAL DESIGN

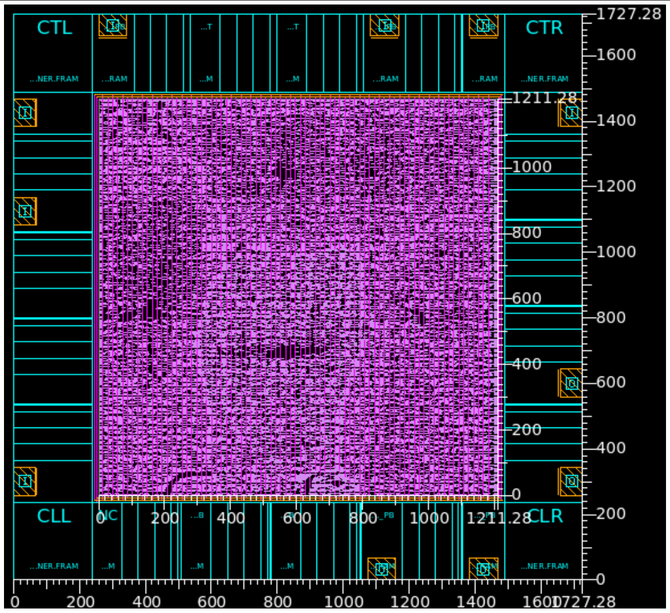
4.1 Optimum Layout Design

After successfully simulating both the RTL and synthesized netlist designs, the next step is physical design of the ASIC chip. Synopsys IC Compiler is used to perform floorplanning, routing, and verification steps.

Table 7 shows results for two unique layouts which use different sizes and block locations to implement the layouts. The best utilization ratio for the final design is Layout 1. Power consumption and the number of cells very similar for both layouts, but there is significant difference in the total area of the chip. Layout 1 chip area (2.9835mm^2) is 65% smaller than the layout 2 chip area (4.0697mm^2). Floorplan utilization is defined as the ratio of the area of standard cells, macros, and the pad cells to the area of the chip minus the area of the sub floorplan. In the Chip Summary report, Area (Macros) Area (Pad Cells), and Area (sub floorplan) are 0. Therefore, practical equation is Floorplan Utilization = Area (Standard Cells) / Area (Chip) = $\frac{545295}{668367}$*100% = 81.59%. This is the calculation used in the standard cell utilization report.

Table 7: Layout Summary for Two Floorplans

	Layout 1	Layout 2
Floorplan Utilization	81.59%	53.11%
Core Chip area	1.4672 mm^2	2.2541 mm^2
Total Chip area	2.9835 mm^2	4.0697 mm^2
Number of Cells	52324	52336
Total Dynamic Power	42.1118 mW (100%)	42.1957 mW (100%)
Leakage Power	37.3090 uW	37.4098 uW



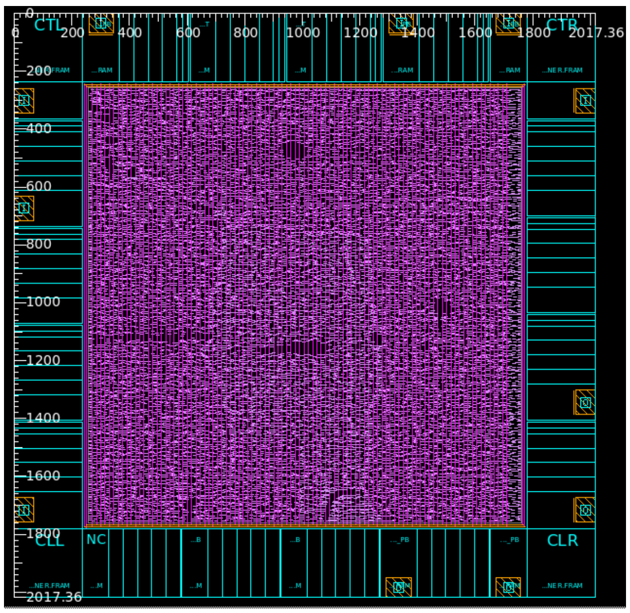
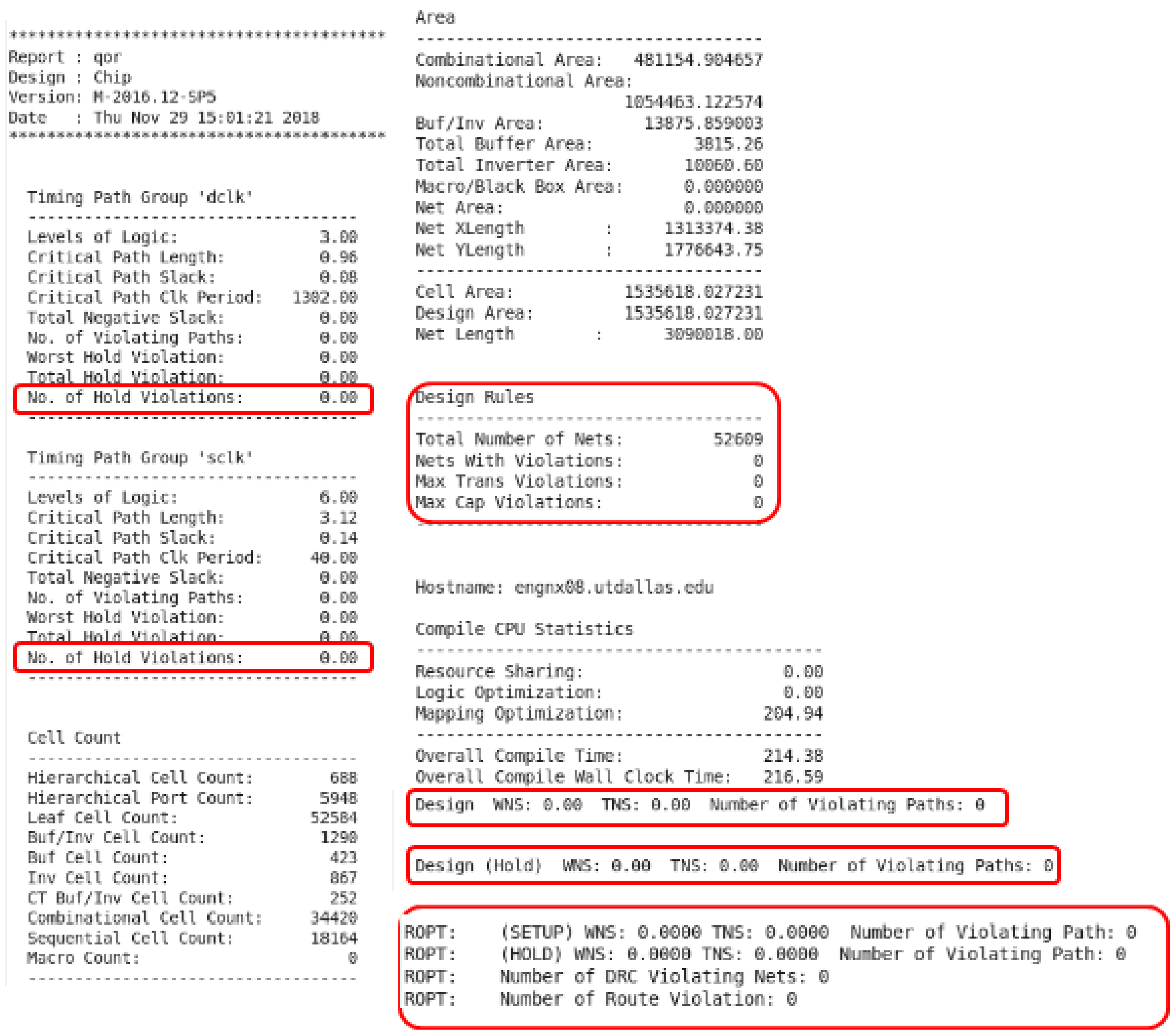


Figure 22: Layout 1 and Layout 2

4.2 Layout Violations

Figure 22 is the MSDAP layout, which was generated by IC Compiler without any violations (Report files are attached). After Placement through Core placement and Optimization, there was no congestion issues. After performing Core Clock Tree Synthesis and Optimization, there were not any negative slack (VIOLATED) in our Setup and Hold reports. In the Core Routing and Optimization, the routing is completed with no DRC violating, but one issue of slack (VIOLATED) for Hold timing. The one issue came from sclk hold time violation, which was fixed using the command "set_fix_hold sclk". After this, there were no timing, placement, and DRC violations at all.



```
****************************************
Report : qor
Design : Chip
Version: M-2016.12-SP5
Date   : Thu Nov 29 15:01:21 2018
****************************************


  Timing Path Group 'dclk'
  -----------------------------------
  Levels of Logic:               3.00
  Critical Path Length:          0.96
  Critical Path Slack:           0.08
  Critical Path Clk Period:   1302.00
  Total Negative Slack:          0.00
  No. of Violating Paths:        0.00
  Worst Hold Violation:          0.00
  Total Hold Violation:          0.00
  No. of Hold Violations:        0.00
  -----------------------------------

  Timing Path Group 'sclk'
  -----------------------------------
  Levels of Logic:               6.00
  Critical Path Length:          3.12
  Critical Path Slack:           0.14
  Critical Path Clk Period:     40.00
  Total Negative Slack:          0.00
  No. of Violating Paths:        0.00
  Worst Hold Violation:          0.00
  Total Hold Violation:          0.00
  No. of Hold Violations:        0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:        688
  Hierarchical Port Count:       5948
  Leaf Cell Count:              52584
  Buf/Inv Cell Count:            1290
  Buf Cell Count:                 423
  Inv Cell Count:                 867
  CT Buf/Inv Cell Count:          252
  Combinational Cell Count:     34420
  Sequential Cell Count:        18164
  Macro Count:                      0
  -----------------------------------
```

```
  Area
  -----------------------------------
  Combinational Area:   481154.904657
  Noncombinational Area:
                       1054463.122574
  Buf/Inv Area:          13875.859003
  Total Buffer Area:          3815.26
  Total Inverter Area:       10060.60
  Macro/Black Box Area:      0.000000
  Net Area:                  0.000000
  Net XLength        :     1313374.38
  Net YLength        :     1776643.75
  -----------------------------------
  Cell Area:           1535618.027231
  Design Area:         1535618.027231
  Net Length        :      3090018.00


  Design Rules
  -----------------------------------
  Total Number of Nets:         52609
  Nets With Violations:             0
  Max Trans Violations:             0
  Max Cap Violations:               0
  -----------------------------------


  Hostname: engnx08.utdallas.edu

  Compile CPU Statistics
  -----------------------------------------
  Resource Sharing:                    0.00
  Logic Optimization:                  0.00
  Mapping Optimization:              204.94
  -----------------------------------------
  Overall Compile Time:              214.38
  Overall Compile Wall Clock Time:   216.59

  Design  WNS: 0.00  TNS: 0.00  Number of Violating Paths: 0

  Design (Hold)  WNS: 0.00  TNS: 0.00  Number of Violating Paths: 0

ROPT:    (SETUP) WNS: 0.0000 TNS: 0.0000  Number of Violating Path: 0
ROPT:    (HOLD) WNS: 0.0000 TNS: 0.0000  Number of Violating Path: 0
ROPT:    Number of DRC Violating Nets: 0
ROPT:    Number of Route Violation: 0
```

Figure 23: Violation Reports

4.3 Analyze Performance of MSDAP

Sclk of MSDAP occurs at the frequency of 25MHz according to our spec. According to Chip Summary report attached, chip area is 1.4672 mm^2 (668367 Sites), the number of cells is 52324, and standard cell utilization is 81.59% because we setup core utilization is 0.86 for Floorplan in order to attain Aspect Ratio, 1.00, and enough high utilization on the chip. Regarding to Power Consumption, attached in Power report, total dynamic power is 42.1118 mW and leakage power is 37.3090 uW, which is really small enough by optimizing power consumption in the IC Compiler. According to Figure 27 report, there is no DRCs.

Table 8: Final Design Performance of MSDAP

Frequency of Sclk	25MHz
Number of Cells	52324
Timing Slack	No negative violation
Floorplan Utilization	81.59%
Total Dynamic Power	42.1118 mW (100%)
Leakage Power	37.3090 uW
Core Chip Area	1.4672 mm^2
Total Chip Area	2.9835 mm^2

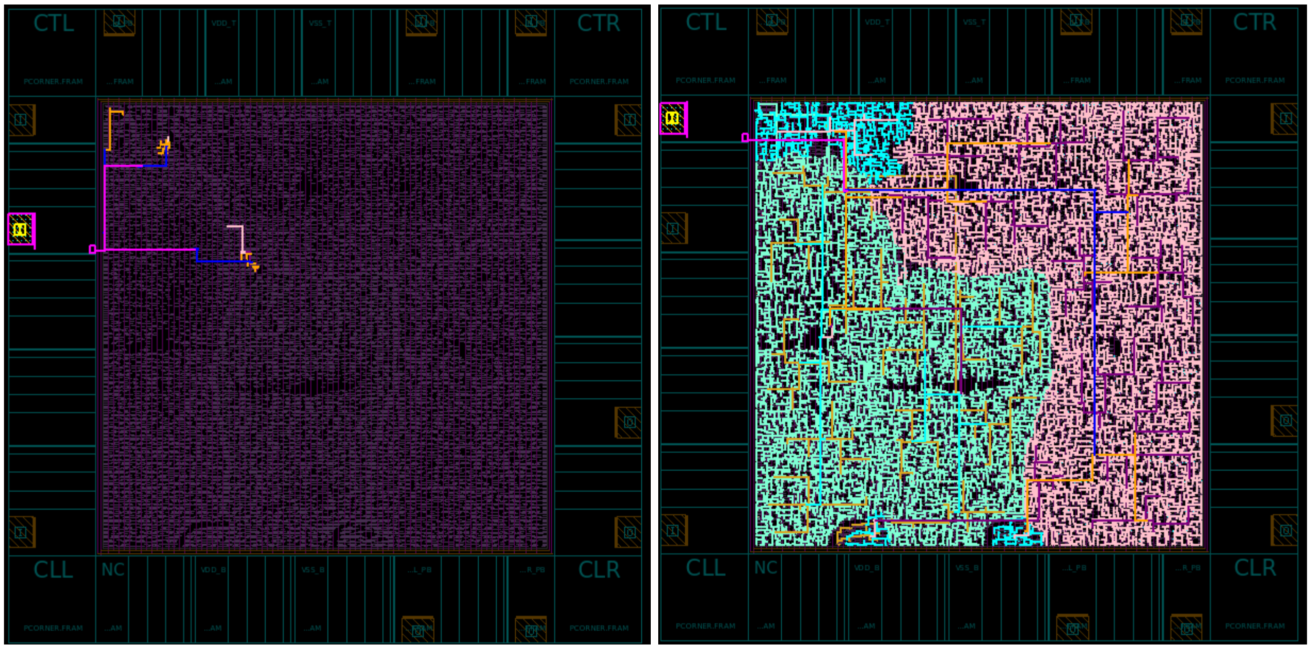
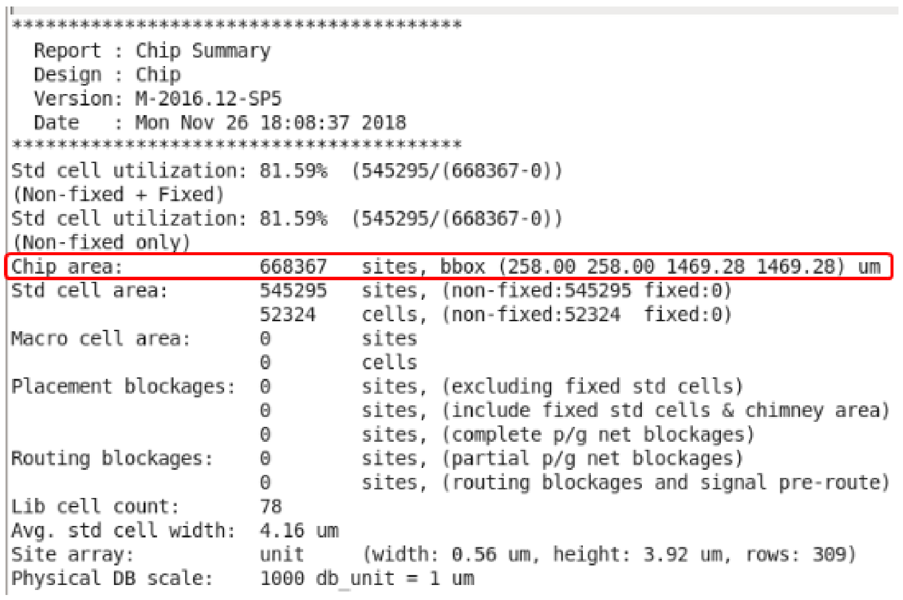
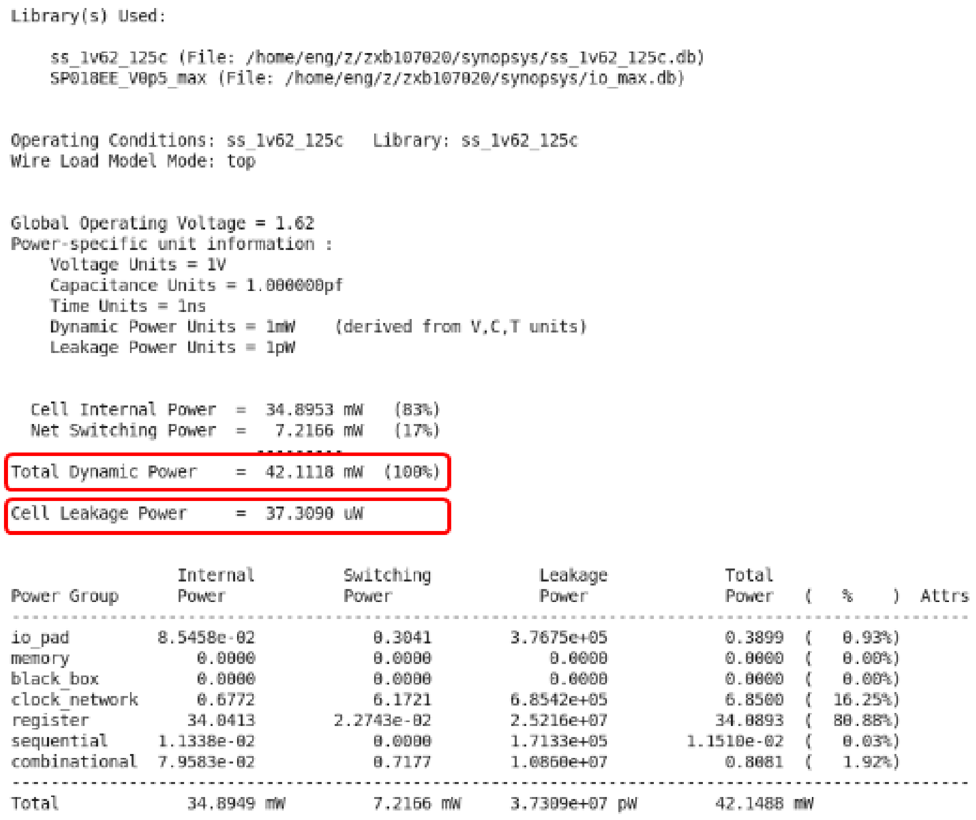


Figure 24: Dclk and Sclk clock tree



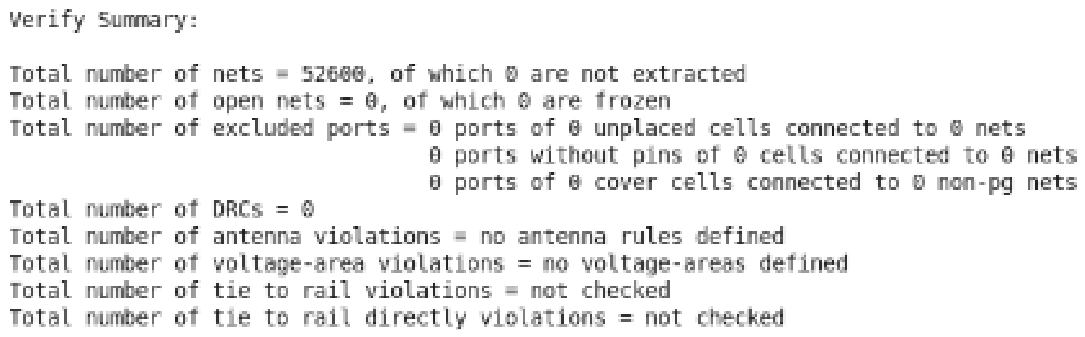
```
****************************************
  Report : Chip Summary
  Design : Chip
  Version: M-2016.12-SP5
  Date   : Mon Nov 26 18:08:37 2018
****************************************
Std cell utilization: 81.59%  (545295/(668367-0))
(Non-fixed + Fixed)
Std cell utilization: 81.59%  (545295/(668367-0))
(Non-fixed only)
Chip area:            668367   sites, bbox (258.00 258.00 1469.28 1469.28) um
Std cell area:        545295   sites, (non-fixed:545295 fixed:0)
                      52324    cells, (non-fixed:52324  fixed:0)
Macro cell area:      0        sites
                      0        cells
Placement blockages:  0        sites, (excluding fixed std cells)
                      0        sites, (include fixed std cells & chimney area)
                      0        sites, (complete p/g net blockages)
Routing blockages:    0        sites, (partial p/g net blockages)
                      0        sites, (routing blockages and signal pre-route)
Lib cell count:       78
Avg. std cell width:  4.16 um
Site array:           unit     (width: 0.56 um, height: 3.92 um, rows: 309)
Physical DB scale:    1000 db_unit = 1 um
```

Figure 25: Chip Summary



```
Library(s) Used:

    ss_1v62_125c (File: /home/eng/z/zxb107020/synopsys/ss_1v62_125c.db)
    SP018EE_V0p5_max (File: /home/eng/z/zxb107020/synopsys/io_max.db)


Operating Conditions: ss_1v62_125c   Library: ss_1v62_125c
Wire Load Model Mode: top


Global Operating Voltage = 1.62
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  34.8953 mW   (83%)
  Net Switching Power  =   7.2166 mW   (17%)
                         ---------
Total Dynamic Power    =  42.1118 mW  (100%)

Cell Leakage Power     =  37.3090 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad         8.5458e-02           0.3041        3.7675e+05           0.3899  (   0.93%)
memory             0.0000           0.0000            0.0000           0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000           0.0000  (   0.00%)
clock_network      0.6772           6.1721        6.8542e+05           6.8500  (  16.25%)
register          34.0413       2.2743e-02        2.5216e+07          34.0893  (  80.88%)
sequential     1.1338e-02           0.0000        1.7133e+05       1.1510e-02  (   0.03%)
combinational  7.9583e-02           0.7177        1.0860e+07           0.8081  (   1.92%)
--------------------------------------------------------------------------------------------------
Total             34.8949 mW        7.2166 mW     3.7309e+07 pW       42.1488 mW
```

Figure 26: Power Consumption



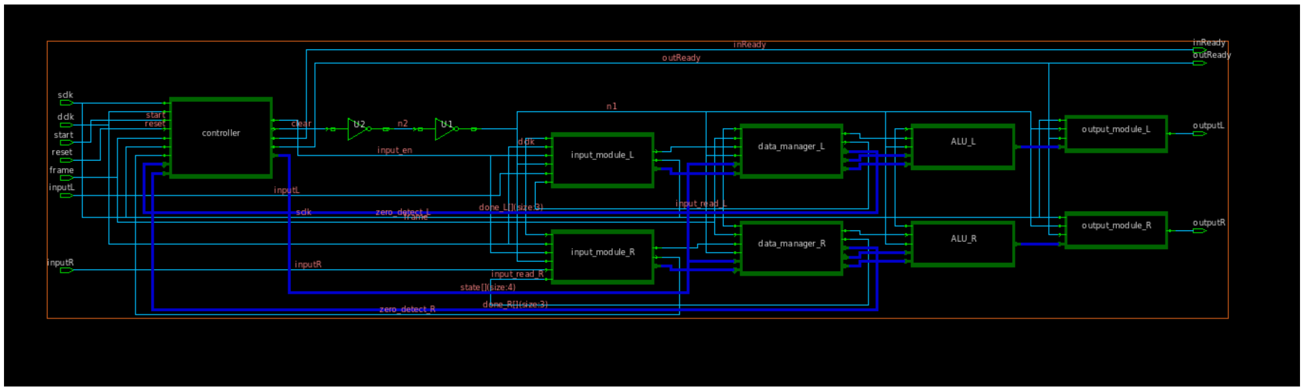
```
Verify Summary:

Total number of nets = 52600, of which 0 are not extracted
Total number of open nets = 0, of which 0 are frozen
Total number of excluded ports = 0 ports of 0 unplaced cells connected to 0 nets
                                 0 ports without pins of 0 cells connected to 0 nets
                                 0 ports of 0 cover cells connected to 0 non-pg nets
Total number of DRCs = 0
Total number of antenna violations = no antenna rules defined
Total number of voltage-area violations = no voltage-areas defined
Total number of tie to rail violations = not checked
Total number of tie to rail directly violations = not checked
```

Figure 27: Final Verification Result

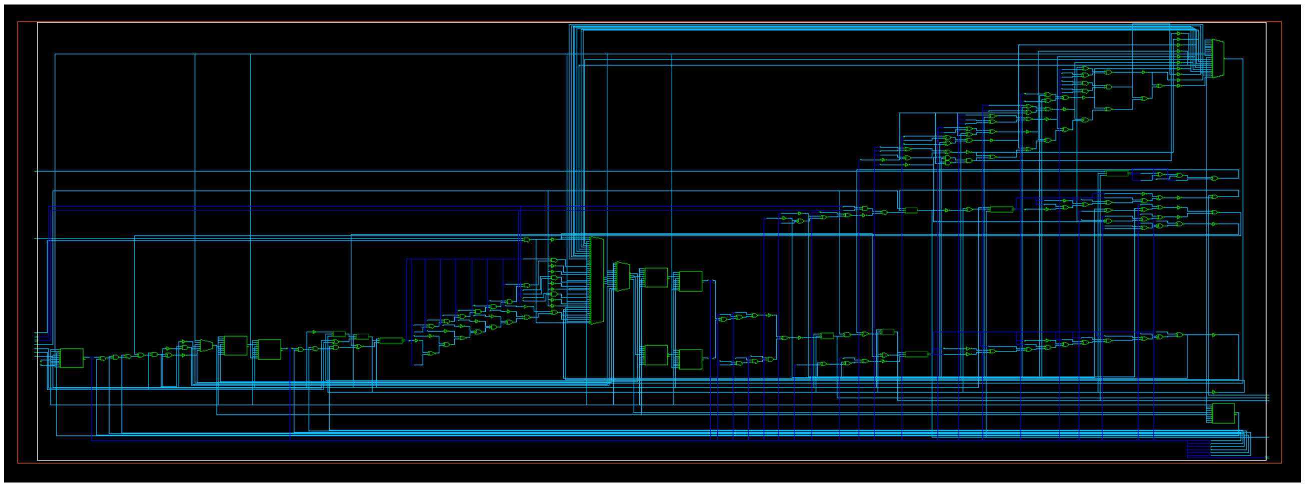
REFERENCES

- Paper on A Mini Stereo Digital Audio Processor (MSDAP), Zhangnong Jiang (Electrical Engineering Department, UCLA)
- “Modern ASIC Design”, Dian Zhou, ISBN 978-7-03-031766-7, Science Press, China, 2011
- “Modern VLSI Design”, Wayne Wolf, Prentice Hall, ISBN 0-13-061970-1

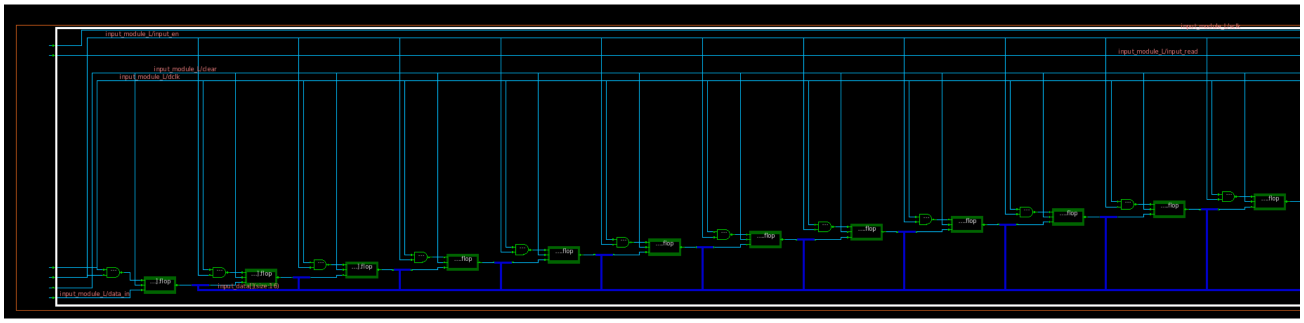
Appendix A:
Design Compiler Schematics



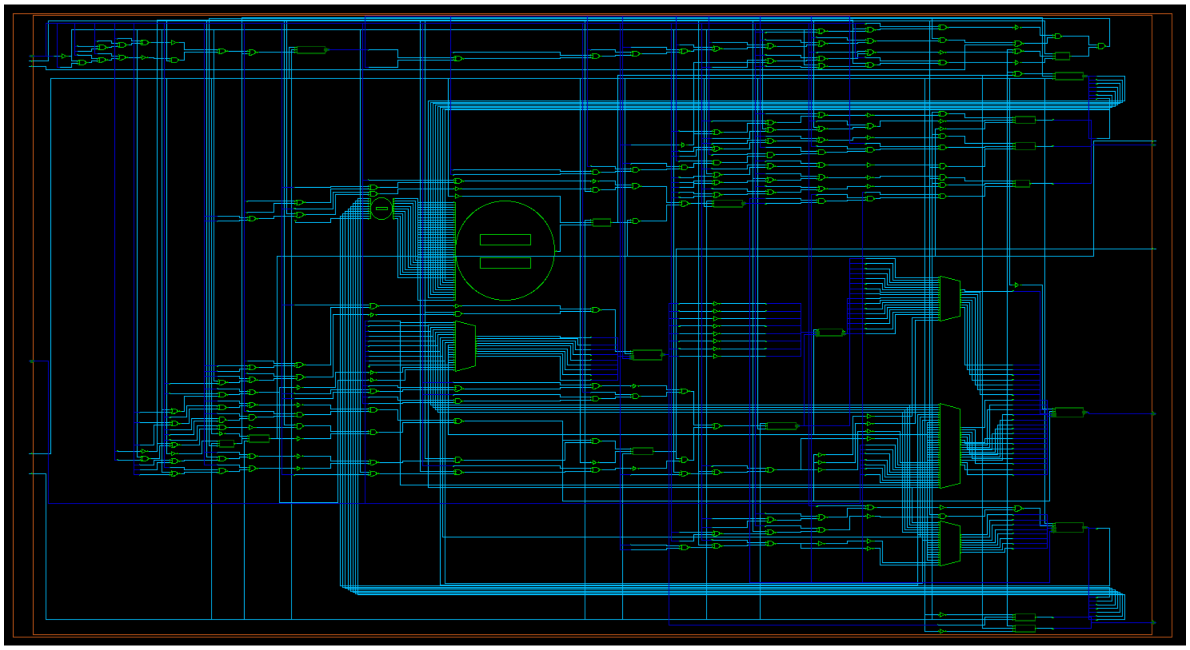
Top Level



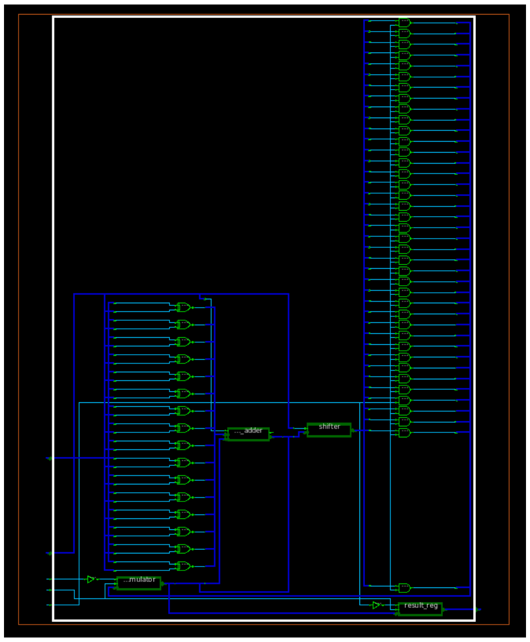
State Controller



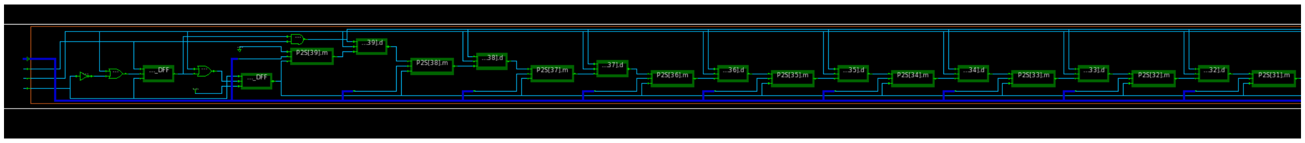
Input Module



Data Manager



ALU Module



Output Module