CE 6303.001

Final Report

Luke Allen NetID: laa170930

Yu Feng NetID: yxf160330

12/6/2021

Contents

Problem Description:

The problem that was assigned for the final project was to create an architecture design for the MSDAP code that was created in the midterm project. Said architecture design must be designed so that each functional block is stand alone and functional closure. Modifications to the code were needed so that it would conform to this architecture design. Once this is complete, we were tasked with synthesizing the code using Synopsis Design Compiler. In order to do so, the code had to be further altered so that it contained only code that is synthesizable. Finally, we were tasked with verifying the functionality of the synthesized design by simulating it in modelsim.

Specification:

Algorithm:

This device is a mini stereo digital audio processor. The device implements the following convolution algorithm on two channels using data sampled in real time:

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

The following is an example of how this algorithm is calculated, given Rj, coefficients, and an input data sample:

Rj

02
03
…

Coefficients

01B
00C
012
002
00A
…

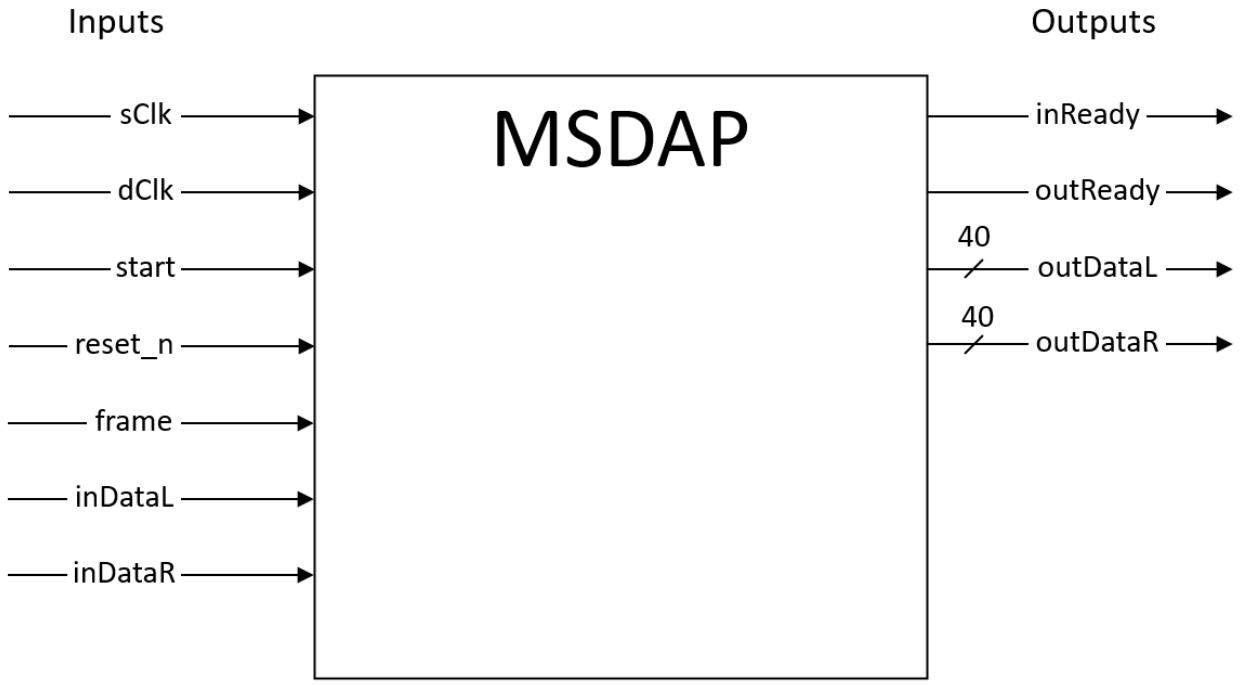
Data: 3C4A

Given the above sample variables, the convolution is as follows:

3C4A = 0011 1100 0100 1010

Extend to 24 bits: 0000 0000 0011 1100 0100 1010

Pin Settings:



Inputs and Outputs:

Inputs:

sClk: System clock, provides the timing reference for internal signals and calculation modules, as well as the reference for the output samples.

dClk: Data clock, provides the timing reference for input samples on inDataL and inDataR.

start: Asynchronous signal, active high. Tells the chip to initialize and begin working.

reset_n: Asynchronous, negative edge triggered reset signal for the main module. When reset_n is set low, the chip enters state 7 and resets.

frame: Active high, frame is used to align the serial input data. Frame is set high for one dClk cycle when the first bit of some input data is received by the chip, and then it is set low.

inDataL: Left data channel, responsible for transmitting all input data for the left channel, including Rjs, coefficients, and data samples. Data is transmitted serially, meaning one bit of data is sent at a time. The input data is read on the falling edge of dClk, and the MSB of data is transmitted first, while the LSB of data is transmitted last.

inDataR: Right data channel, responsible for transmitting all input data for the right channel, including Rjs, coefficients, and data samples. Data is transmitted serially, meaning one bit of data is sent at a time. The input data is read on the falling edge of dClk, and the MSB of data is transmitted first, while the LSB of data is transmitted last.

Outputs:

inReady: Set high on the rising edge of sClk when the chip is ready to receive input data. Otherwise, it is set low.

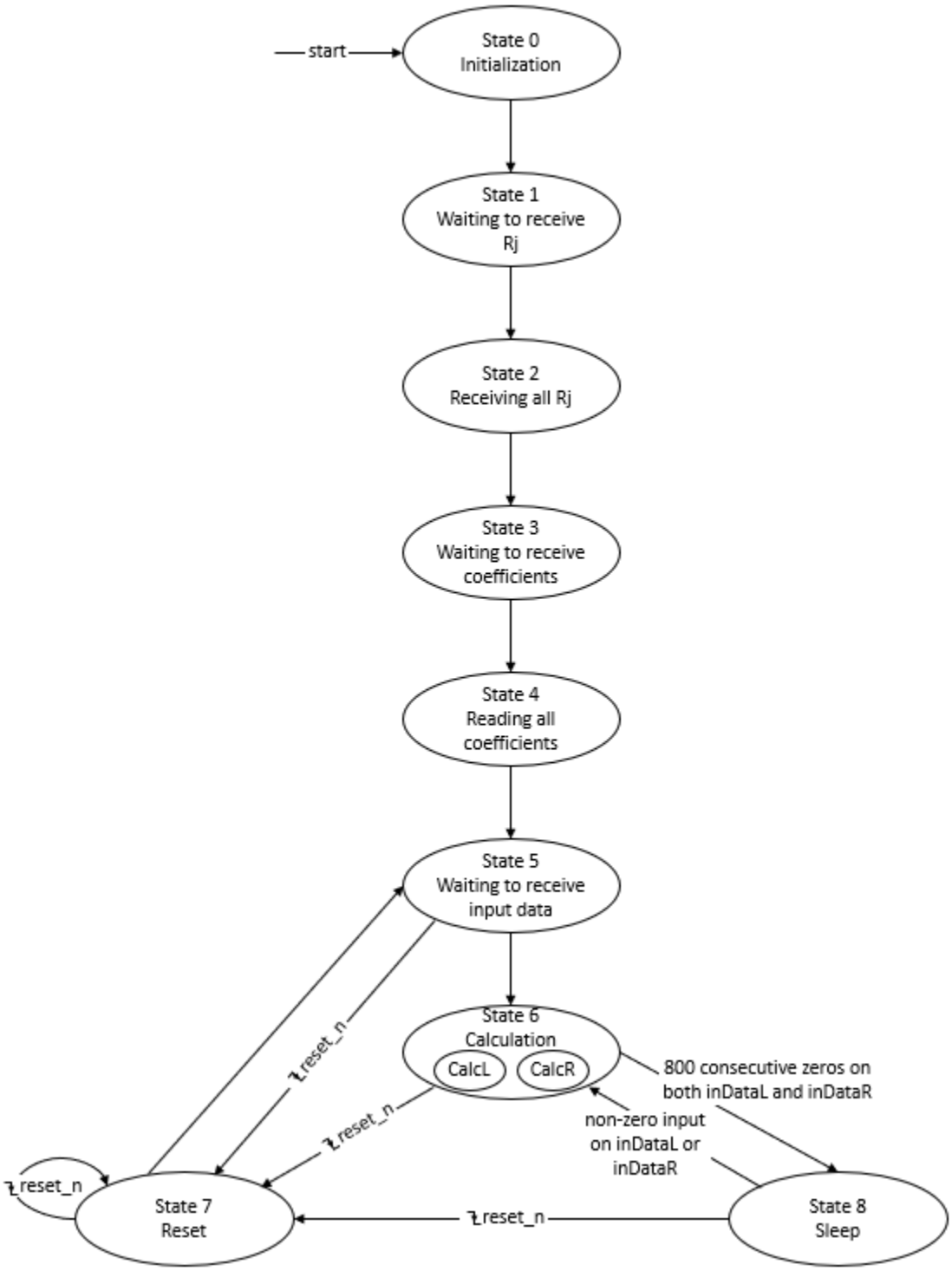
outReady: Set high at the rising edge of frame when the chip is ready to transmit output data. Otherwise, it is set low.

outDataL: Left output channel parallel data bus. Outputs all bits of the left channel output data on the rising edge of sClk. All bits are output on the 40-bit bus in one clock cycle.

outDataR: Right output channel parallel data bus. Outputs all bits of the right channel output data on the rising edge of sClk. All bits are output on the 40-bit bus in one clock cycle.

Operation Modes:

Main FSM:



State 0: Initialization stage. When start is high, the chip enters state 0 and begins initialization. Initialization entails setting all variables to their initial values and clearing all memories.

State 1: Waiting to receive Rj values. In this state, inReady is set high. Once a frame is detected from the controller, the FSM enters state 2.

State 2: Reading all Rj values. In this stage, inReady remains high. All Rj values for both channels are read and stored. Once all Rj values have been read, the FSM enters state 3.

State 3: Waiting to receive all coefficient values. Similar to state 1, In this state inReady is set high. Once a frame is detected from the controller, the FSM enters state 4.

State 4: Reading all coefficients. Similar to state 2, in this stage, inReady remains high. All coefficient values for both channels are read and stored. Once all coefficient values have been read, the FSM enters state 5.

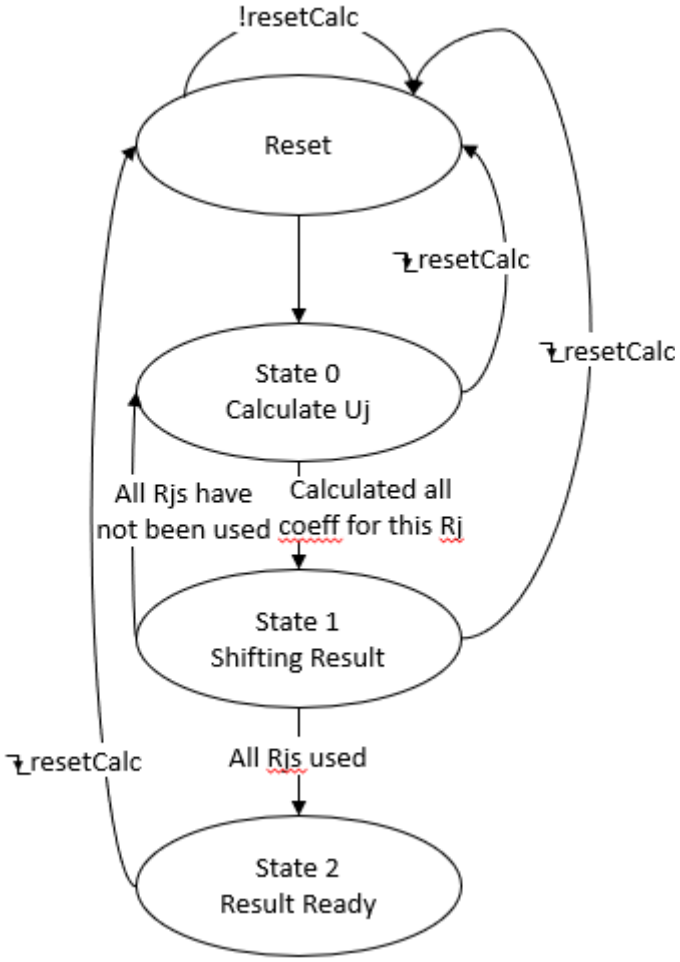
State 5: Waiting to receive data. In this state, inReady is set high. When frame is detected to be high from the controller, the chip enters state 6. If there is a falling edge of reset_n, the chip enters state 7 to reset.

State 6: This is the calculation state. In this state, inReady remains high, as the chip continually reads input samples on both channels. Additionally, the chip does the convolution computation on both channels, and outputs the computed data. If a negative edge of reset_n is detected, the FSM enters state 7 to reset. If 800 consecutive zeros are detected on both channels, the FSM enters state 8 to sleep. Otherwise, the FSM continues accepting input data and calculating results in state 6.

State 7: Reset state. In this state, inReady is set low. All variables and memories are reset to their initial values except for the saved values of Rj and coefficients. If another negative edge of reset_n is detected during this process, the chip will reenter state 7. Otherwise, the FSM will move to state 5 and await more data from the controller.

State 8: Sleeping state. In this state, the chip enters sleeping mode to consume less power when no input is detected. inReady remains high, and if any non-zero input is detected on either channel, the chip enters state 6. If a falling edge of reset_n is detected, the FSM enters state 7.

Calculation Module FSM:



Within the top-level FSM, there is another FSM which handles the convolution calculation. There are two of these FSMs present in state 6 of the top-level FSM, one for the left channel and one for the right. The following describes each state of the calculation FSM:

Reset: Reset state, clears all memory and variables to zero. If resetCalc is detected to be low while in this state, it remains in this state. Otherwise, the FSM moves to state 0.

State 0: Calculation state. The calculation module signals to the top-level FSM that it is ready to receive a data point. Then, the 16-bit input data is sign extended to 24 bits, converted to twos compliment if it is negative, and padded with zeros until it is 40 bits long. Then, this processed data is added to a variable which accumulates the calculated values for this Uj. A counter is updated to keep track of how many coefficients the chip has done the calculation for, and if all of the coefficients for this Uj have not been used, the state remains at 0 to repeat the process on the next clock cycle. If a negative edge of resetCalc is detected, the FSM moves to the reset state. Otherwise, if we have done the calculation using all of the coefficients, the state is set to 1.

State 1: Shifting state. Adds the current Uj to the overall result and shifts the overall result right 1 bit, padding a one at the MSB if it is negative. Increments the Rj index. If it has reached the last

Rj, move one to state 2. Otherwise, set the current Uj back to 0 and move to state 0. If a falling edge of resetCalc is detected, move to the reset state.

State 2: Result ready state. In this state, the calculation module signals that the calculation for the current N has been completed, and the output is ready to be sent out. The FSM does not leave this state until the top-level FSM has sent out the data and sets the resetCalc signal low. When this happens, the calculation FSM will reset and start over.

Clock Frequencies:

To determine the necessary clock frequency, you must consider how many clock cycles it will take to calculate the overall result for one iteration N. For a convolution with 512 coefficients, it will take this chip around 510 clock cycles to compute the overall result. You must also take into account the data clock frequency. This is crucial, because in order to avoid necessary data being overwritten by input data, you must ensure that your overall result is finished calculating before a new data sample is read and stored.

As an example, if a data clock frequency of 768kHz is used, we have a clock period of 1302 ns, meaning one bit of the input data is read every 1302 ns. Therefore, it will take:

$$1302\ ns\ *16\ bits = 20832\ ns$$

for one entire data sample to be read and stored. Additionally, if it takes 510 clock cycles to compute the overall result, we will need to set the system clock to a frequency fast enough where 510 clock cycles is less than 20832 ns. Therefore, we must find a clock period accordingly:

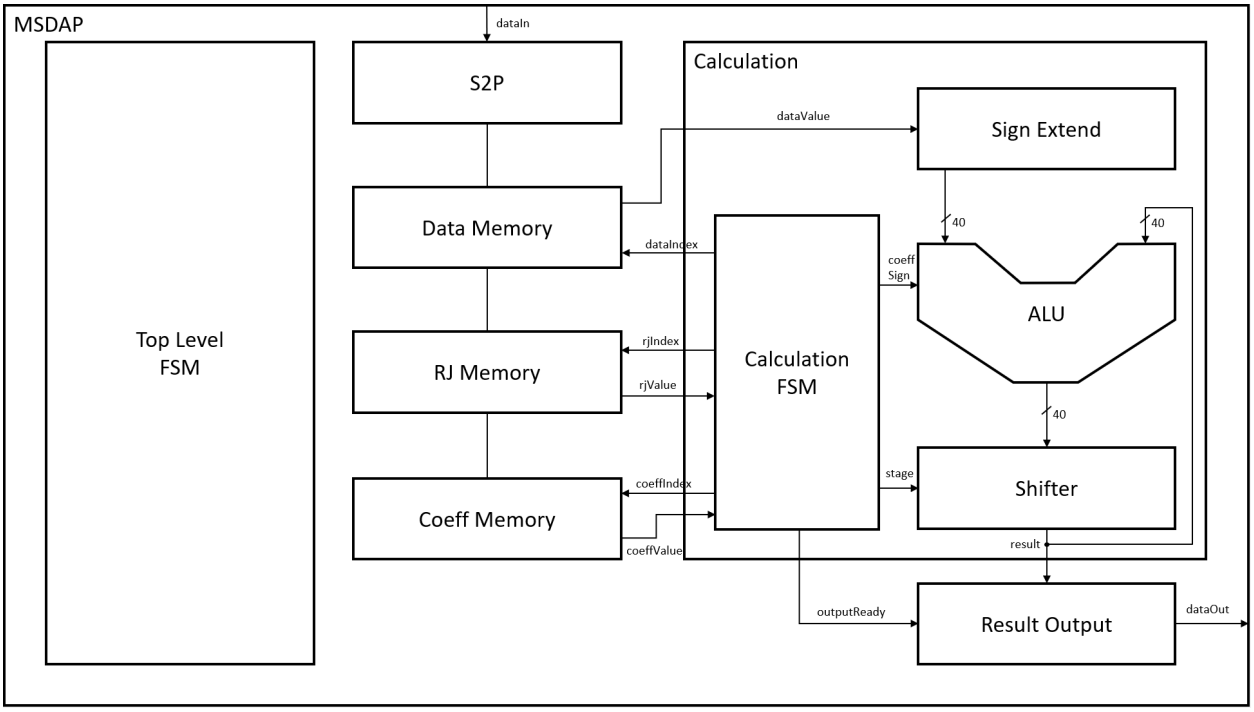
$$20832\ ns\ \div x = 510\ cycles$$

$$x = 40.847$$

Therefore, a clock period faster than 40 ns is needed, and hence a clock frequency of 26.88 MHz is selected.

Implementation:

Architecture Description



Shown above is a block diagram which illustrates the architecture design of our chip. Note that only a few signals/connections are shown with shortened names for simplicity purposes, and the comprehensive list of signals can be found in the logic block specifications section. Additionally, there are only memory and calculation modules shown for one channel in the diagram above. In order to implement both channels, duplicate instances of these modules are instantiated. The top-level module contains an always block which implements the top-level FSM. Within this module is also instances of the S2P, Data Memory, RJ Memory, Coeff Memory, Calculation, and Output modules. There are two instances of each of the memories, as well as two instances of the calculation module to implement the left and right channels of the device. The input data is read in by the serial to parallel module, and depending on what state the FSM is in, is stored into the respective memory module. Whenever a data sample is ready to be processed, the calculation module performs the necessary calculations. The calculation module contains an always block which implements the FSM that controls the calculation, as well as instances of the sign extender module, an ALU module, and a shifter module. The calculation FSM will interface with the data memory module to obtain the data sample, which will be loaded into the sign extend module. It is here where the data will be sign extended and padded to 40 bits. Then, the calculation FSM/Controller will interface with the RJ and Coefficient memory to obtain the corresponding values at the current rj and coefficient indexes. From there, the ALU will perform an addition or a subtraction depending on the sign of the current coefficient, and its result will be sent to the shifter. If the calculation has been done for all coefficients in the current rj, the shifter will shift the result. Otherwise, the shifter will output the unshifted result and it will go back to the ALU to get added. Once all rjs have been calculated, the calculation sends an output ready signal to the output module, and the output from the shifter gets sent to the output of the device. This implements the calculation described in the specification. More details can be found in the logic block specifications.

Logic Block Specifications

FSM_MSDAP:

Signal Name	Input/Output	Size	Description
sClk	input	1	System clock, provides the timing reference for internal signals and calculation modules, as well as the reference for the output samples.
dClk	Input	1	Data clock, provides the timing reference for input samples on inDataL and inDataR.
start	Input	1	Asynchronous signal, active high. Tells the chip to initialize and begin working.
reset_n	Input	1	Asynchronous, negative edge triggered reset signal for the main module. When reset_n is set low, the chip enters state 7 and resets.
frame	Input	1	Active high, frame is used to align the serial input data. Frame is set high for one dClk cycle when the first bit of some input data is received by the chip, and then it is set low.
inDataL	Input	1	Left data channel, responsible for transmitting all input data for the left channel, including Rjs, coefficients, and data samples. Data is transmitted serially, meaning one bit of data is sent at a time. The input data is read on the falling edge of dClk, and the MSB of data is transmitted first, while the LSB of data is transmitted last.
inDataR	Input	1	Right data channel, responsible for transmitting all input data for the right channel, including Rjs, coefficients, and data samples. Data is transmitted serially, meaning one bit of data is sent at a time. The input data is read on the falling edge of dClk, and the MSB of data is transmitted first, while the LSB of data is transmitted last.
inReady	Output	1	Set high on the rising edge of sClk when the chip is ready to receive input data. Otherwise, it is set low.
outReady	Output	1	Set high at the rising edge of frame when the chip is ready to transmit output data. Otherwise, it is set low.
outDataL	Output	40	Left output channel parallel data bus. Outputs all bits of the left channel output data on the rising edge of sClk. All bits are output on the 40-bit bus in one clock cycle.
outDataR	Output	40	Right output channel parallel data bus. Outputs all bits of the right channel output data on the rising edge of sClk. All bits are output on the 40-bit bus in one clock cycle.

The top level MSDAP module implements the top-level finite state machine. This FSM controls the state of the overall system, handling the reading of data, calculation and output, sleep, and reset. Further description of this FSM can be found in the specification section. The S2P, memory modules, calculation modules and result output modules are instantiated within this module. This module also

keeps track of the number of zeros input on each channel in state 6 and will enter sleep mode if it detects the appropriate number of consecutive zeros.

S2P:

Signal Name	Input/Output	Size	Description
dClk	Input	1	Data clock
coeffIndex	Output	9	Coefficient Address to write to coefficient memory
coeffWriteL	Output	1	Write enable signal for left coefficient memory
coeffWriteR	Output	1	Write enable signal for right coefficient memory
dataIndexL	Output	8	Index of current left data
dataIndexR	Output	8	Index of current right data
dataWriteL	Output	1	Write enable signal for left data memory
dataWriteR	Output	1	Write enable signal for right data memory
dataWriteIndexL	Output	1	Address to write to left data memory
dataWriteIndexR	Output	1	Address to write to right data memory
done	Output	1	Signals to FSM that all Rjs have been received, all coefficients have been received, or one full data sample has been received
frame	Input	1	Used to indicate a new serial data input
inSerialDataL	Input	16	Left input serial data bus
inSerialDataR	Input	16	Right input serial data bus
inReady	Input	1	Indicates ready to receive input data
reset_n	Input	1	External, asynchronous reset signal
resetDataDone_n	Input	1	Reset after one data sample has been received
rjIndex	Output	4	Rj address to write to Rj memory
rjWriteL	Output	1	Write enable signal for left Rj memory
rjWriteR	Output	1	Write enable signal for right Rj memory
FSMState	Input	4	State of the MSDAP FSM module
Start	Input	1	External start signal, indicates when to begin receiving
toMemL	Output	16	Input data on left channel
toMemR	Output	16	Input data on right channel
wakeupSignal	Input	1	Signal used to awaken the S2P module after sleep

This is the serial to parallel input module. This module handles the input data, reading the serial input and converting it into a parallel data bus. The module then directly interfaces with the memory modules to save the input data in the correct memory space.

DATA_MEM:

Signal Name	Input/Output	Size	Description
sClk	Input	1	System clock
dataReadAddr	Input	8	Memory address to read from
dataWriteAddr	Input	8	Memory address to write to
dataValueIn	Input	16	Input data value to write into memory
dataValueOut	Output	16	Output data value read from memory
memReset_n	Input	1	Memory reset signal, clears contents of memory
writeEnable	Input	1	Writes input data to input address when high

This is the data memory module which manages the memory space for input data samples. This module contains a memory space of 256 16-bit numbers, with and address size of 8 bits. The data memory receives parallel input data samples from the S2P module and saves them in the correct memory index. It also sends data samples that are stored in memory that have been requested by the calculation module during state 6.

RJ_MEM:

Signal Name	Input/Output	Size	Description
rjReadAddr	Input	5	Memory address to read from
rjWriteAddr	Input	4	Memory address to write to
rjDataIn	Input	16	Input rj value to write into memory
rjDataOut	Output	16	Output rj value read from memory
writeEnable	Input	1	Writes input data to input address when high

This is the Rj memory module which manages the memory space for input Rj values. This module contains a memory space of 16 16-bit numbers, with and address size of 4 bits. The Rj memory receives parallel input Rj values from the S2P module and saves them in the correct memory index. It also sends Rj values that are stored in memory that have been requested by the calculation module during state 6.

COEFF_MEM:

Signal Name	Input/Output	Size	Description
coeffReadAddr	Input	10	Memory address to read from
coeffWriteAddr	Input	9	Memory address to write to
coeffDataIn	Input	16	Input coefficient value to write into memory
coeffDataOut	Output	16	Output coefficient value read from memory
writeEnable	Input	1	Writes input data to input address when high

This is the coefficient memory module which manages the memory space for input coefficient values. This module contains a memory space of 512 16-bit numbers, with and address size of 9 bits. The coefficient memory receives parallel input coefficient values from the S2P module and saves them in the correct memory index. It also sends coefficient values that are stored in memory that have been requested by the calculation module during state 6.

Calculation:

Signal Name	Input/Output	Size	Description
sClk	Input	1	System clock
reset_n	Input	1	External, asynchronous reset signal
currentCoeffIndex	Output	10	Index of the coefficient that is being used for the current calculation
coeffExtracted	Input	16	Value of the current coefficient
dataIndex	Input	8	Index of the data sample being used by the current calculation
dataIndexToFetch	Output	8	Index of the data sample requested to fetch from memory
inData	Input	16	Data sent in from memory
currentRjIndex	Output	5	Index of the rj value being used by the current calculation
currentRjValue	Input	16	Value of rj being used by the current calculation
thisNCompleted	Output	1	Signal to indicate all iterations have been completed for the current n
overallResult	Output	40	Calculated output result

The calculation module implements the FSM for the calculation module. This FSM is mainly used to determine when the shift operation needs to be performed, as the shift and add functions cannot be executed on the same clock cycle. The FSM does this by keeping track of the calculations done for each coefficient and Rj. Additionally, the FSM determines when the result has finished calculating and when it is ready to be output. More information about this FSM can be found in the specification section. The sign extend, adder, and bit shifter modules are instantiated within this module. This module also manages control signals for the calculation, and interfaces directly with the memory modules to obtain the data sample, coefficient, and Rj values needed for each calculation.

Calc Sign Extend:

Signal Name	Input/Output	Size	Description
inData	Input	16	Data sample input to the module after being read from memory
dataBitExtended	Output	40	Output data sample that has been sign extended and padded to 40 bits

The sign extend module performs a sign extension on a requested input data sample that has been read from the data memory. This data sample is sign extended to 24 bits and then 16 zeros are padded on the right to extend it to 40 bits. The resulting data is output to the adder.

Calc Adder:

Signal Name	Input/Output	Size	Description
sClk	Input	1	System clock
inCoeffSign	Input	1	Sign of the coefficient for the current calculation, determines if the input data sample will be added or subtracted from the running total
inOperandA	Input	40	Input to the ALU from the sign extender, contains the sign extended and padded input data sample
inOperandB	Input	40	Input to the ALU from the bit shifter, contains the bit shifted or non-bit shifted result of the previous calculation
calcResult	Output	40	Output result of the ALU calculation
ALUResultReady	Output	1	Indicates that the result has been calculated
ALUReset_n	Input	1	Resets the ALU

The adder module obtains two input operands, one from the output of the sign extender module, and one from the output of the bit shifter module. Based on the sign on the current coefficient, the ALU will either add or subtract (using twos compliment) the two operands. This result is then output to the bit shifter.

Calc Bit Shifter:

Signal Name	Input/Output	Size	Description
calcStage	Input	3	Stage of the calculation FSM
ALUResult	Input	40	Input result from the ALU
shifterDataResult	Output	40	Output result after being shifted/not shifted

The bit shifter module receives the result form the ALU as an input. If the calculation FSM is in the shifting stage, this module will shift the data, padding it with the sign bit. Otherwise, if the FSM is not in the shifting stage, this module will simply pass the data through the module to the output. This output result is connected to Operand B of the ALU, as well as the result output module.

Result Output:

Signal Name	Input/Output	Size	Description
sClk	Input	1	System clock
outDataL	Output	40	Calculated result for the left channel sent to the output of the device
outDataR	Output	40	Calculated result for the right channel sent to the output of the device
outReadyFromFSM	Input	1	Signal from calculation FSM which indicated that both channels' results have been calculated and are ready to be output
outReadyToTB	Output	1	Output signal of the device which indicates that the output is ready to the controller
resultL	Input	40	Input calculated result from the left channel calculation module
resultR	Input	40	Input calculated result from the right channel calculation module

The result output module receives the calculated result from the output of the calculation module for both the left and right channels. When the outReadyFromFSM is received from the FSM indicating both the left and right calculation modules have completed the calculation, the result will be sent to the output of the device on the sClk rising edge. When this happens, the outReady signal is also sent out of the device to inform the controller that output data is ready.

Verification:

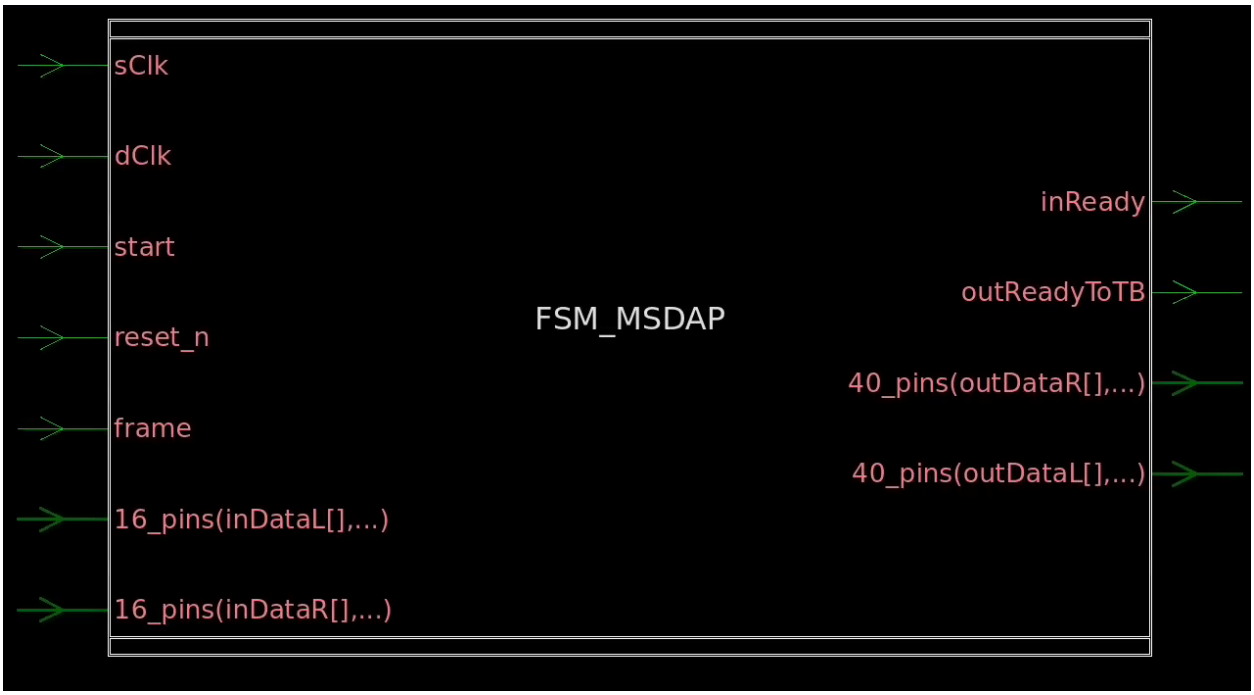
Loops

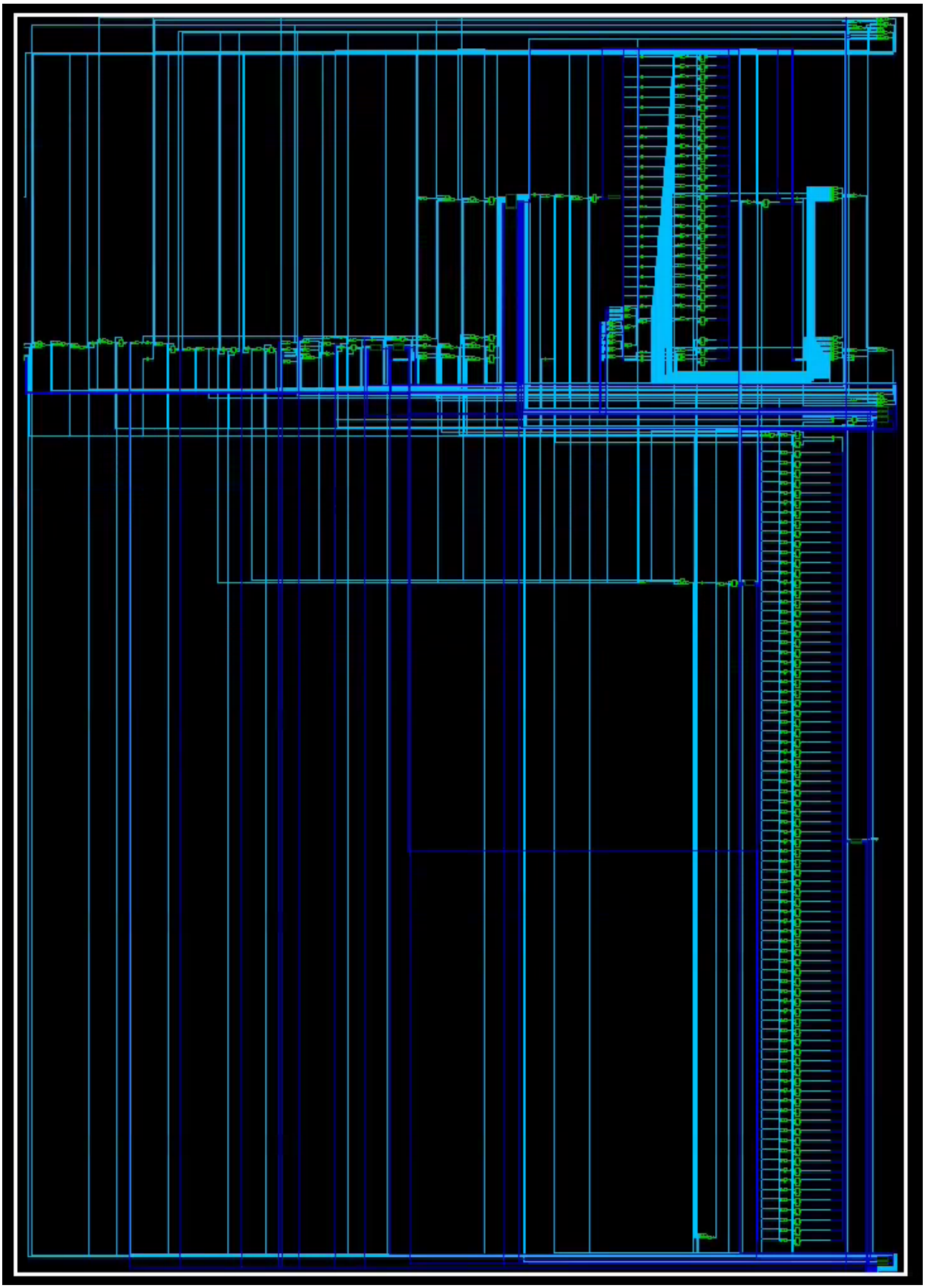
No loops were used in the final implementation of this design. Previous iterations of the design used loops, however they were removed for synthesis purposes.

Synthesis

```
engnx06a.utdallas.edu:/home/012/l/la/laa170930/ECS_6360/Final/DesignVision
File  Edit  View  Search  Terminal  Help
    0:03:03 2305003.9      1.54       5.3       0.0
    0:03:03 2304938.0      1.54       5.3       0.0
    0:03:04 2304938.0      1.54       5.3       0.0
    0:03:04 2304938.0      1.54       5.3       0.0
    0:03:04 2304938.0      1.54       5.3       0.0
    0:03:04 2304938.0      1.54       5.3       0.0
    0:03:04 2304938.0      1.54       5.3       0.0
    0:03:04 2305616.3      0.00       0.0       0.0 calcL/endCoeffIndex_reg[14]/
D
    0:03:05 2305607.5      0.00       0.0       0.0
Loading db file '/home/eng/y/yxw173630/lib/synopsys/ss_1v62_125c.db'
Loading db file '/home/eng/y/yxw173630/lib/synopsys/ff_1v98_0c.db'
Loading db file '/home/eng/y/yxw173630/lib/synopsys/tt_1v8_25c.db'


Note: Symbol # after min delay cost means estimated hold TNS across all active s
cenarios


  Optimization Complete
  ---------------------
Warning: Design 'FSM_MSDAP' contains 1 high-fanout nets. A fanout number of 1000
 will be used for delay calculations involving these nets. (TIM-134)
     Net 'calcR/adder/sClk': 8682 load(s), 1 driver(s)
```

Overall Schematic





Verilog Simulations

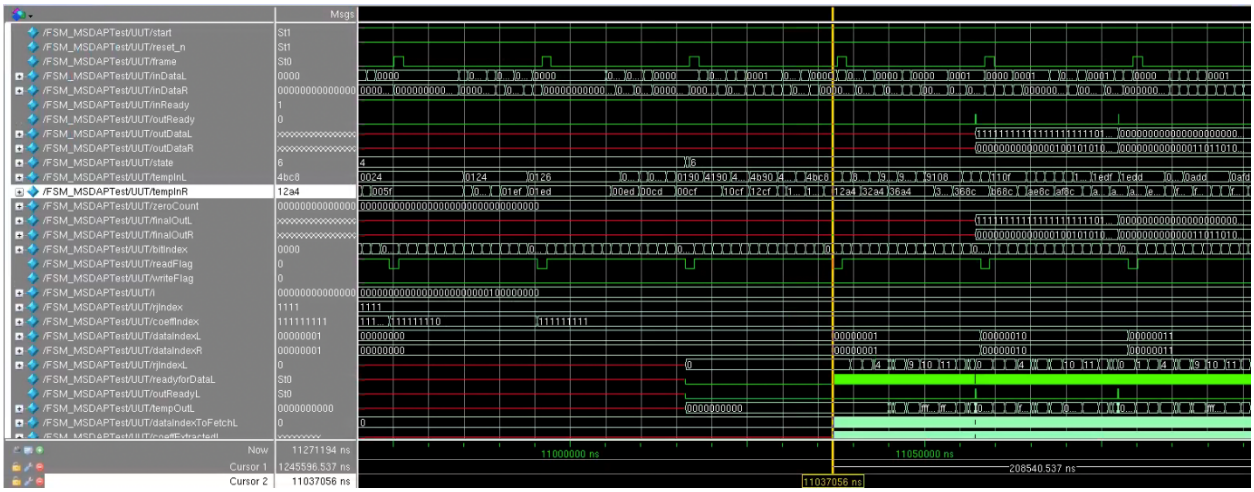


Figure 1: Waveform demonstrating reading a data sample

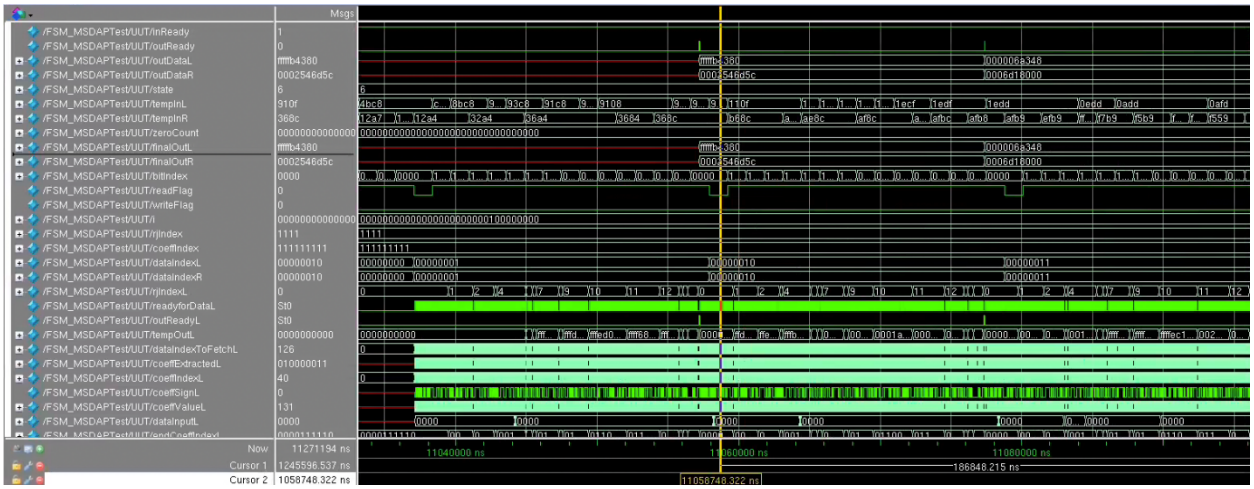


Figure 2: Waveform demonstrating outputting calculated data

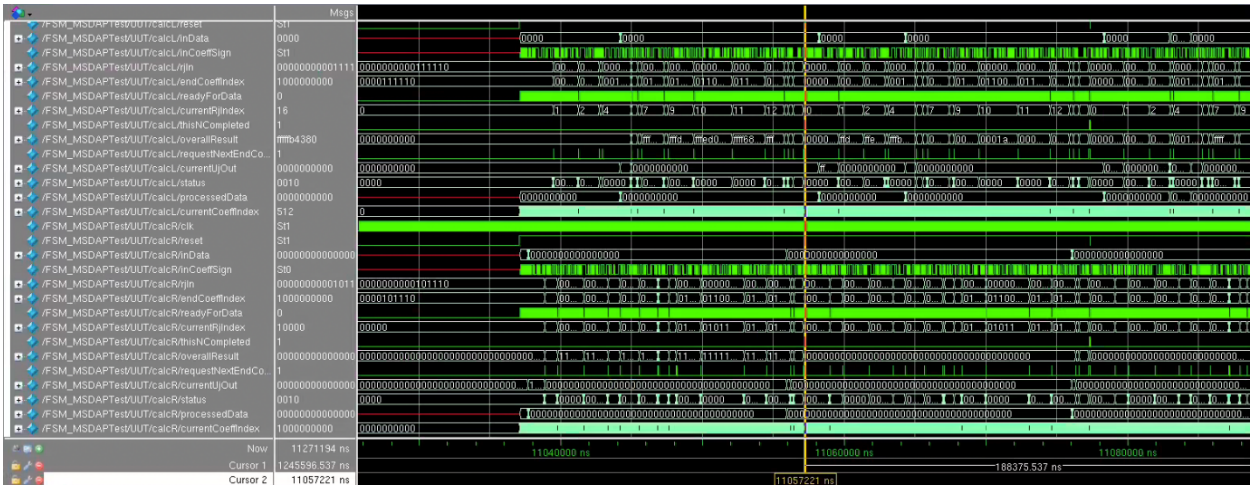


Figure 3: Waveform demonstrating the operation of the calculation module

Netlist Simulations

Timing Report

```
Information: Updating design information... (UID-85)
Warning: Design 'FSM_MSDAP' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets.
(TIM-134)
Warning: There are infeasible paths detected in your design that were ignored during optimization. Please run 'report_timing -attributes'
and/or 'create_qor_snapshot/query_qor_snapshot -infeasible_paths' to identify these paths.  (OPT-1721)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : FSM_MSDAP
Version: L-2016.03-SP3
Date   : Mon Dec  6 01:11:02 2021
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

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

  Startpoint: wakeupSignal_reg
              (rising edge-triggered flip-flop clocked by sClk)
  Endpoint: serialRead/rjIndex_reg[3]
            (falling edge-triggered flip-flop clocked by dClk)
  Path Group: dClk
  Path Type: max

  Point                                                   Incr       Path
  -----------------------------------------------------------------------------
  clock sClk (rise edge)                                648.00     648.00
  clock network delay (ideal)                             0.00     648.00
  wakeupSignal_reg/CK (EDFFX2M)                           0.00 #   648.00 r
  wakeupSignal_reg/Q (EDFFX2M)                            0.38     648.38 f
  serialRead/wakeupSignal (S2P)                           0.00     648.38 f
  serialRead/U161/Y (BUFX2M)                              0.43     648.81 f
  serialRead/U45/Y (NOR2XLM)                              0.46     649.26 r
  serialRead/U293/Y (AOI21X1M)                            0.24     649.51 f
  serialRead/U198/Y (INVX2M)                              0.10     649.60 r
  serialRead/U197/Y (NAND4BX1M)                           0.24     649.85 f
  serialRead/U196/Y (INVX2M)                              0.13     649.98 r
  serialRead/U312/Y (NAND2X2M)                            0.11     650.09 f
  serialRead/U326/Y (NOR2BX2M)                            0.20     650.29 r
  serialRead/U317/Y (NAND2X2M)                            0.10     650.39 f
  serialRead/U316/Y (XNOR2X2M)                            0.19     650.57 f
  serialRead/rjIndex_reg[3]/D (DFFNSRHX1M)                0.00     650.57 f
  data arrival time                                                650.57

  clock dClk (fall edge)                                651.00     651.00
  clock network delay (ideal)                             0.00     651.00
  serialRead/rjIndex_reg[3]/CKN (DFFNSRHX1M)              0.00     651.00 f
  library setup time                                     -0.11     650.89
  data required time                                               650.89
  -----------------------------------------------------------------------------
  data required time                                               650.89
  data arrival time                                               -650.57
  -----------------------------------------------------------------------------
  slack (MET)                                                        0.31


  Startpoint: inReady_reg
              (rising edge-triggered flip-flop clocked by sClk)
  Endpoint: inReady (output port clocked by sClk)
  Path Group: sClk
  Path Type: max
```

Cell Report

```
****************************************
Report : cell
Design : FSM_MSDAP
Version: L-2016.03-SP3
Date   : Mon Dec  6 01:11:18 2021
****************************************

Attributes:
   BO - reference allows boundary optimization
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic
```

Cell	Reference	Library	Area	Attributes
S2PReset_n_reg	EDFFX2M	ss_1v62_125c	61.465599	n
U205	INVXLM	ss_1v62_125c	6.585600	
U206	OA21XLM	ss_1v62_125c	13.171200	
U207	NOR4X2M	ss_1v62_125c	21.952000	
U208	INVXLM	ss_1v62_125c	6.585600	
U209	CLKINVX2M	ss_1v62_125c	6.585600	
U210	AOI2B1X2M	ss_1v62_125c	17.561600	
U211	CLKINVX2M	ss_1v62_125c	6.585600	
U212	MX2X1M	ss_1v62_125c	19.756800	
U213	NAND2X1M	ss_1v62_125c	8.780800	
U214	NAND2X1M	ss_1v62_125c	8.780800	
U215	MX2XLM	ss_1v62_125c	19.756800	
U216	BUFX2M	ss_1v62_125c	8.780800	
U217	BUFX2M	ss_1v62_125c	8.780800	
U218	BUFX2M	ss_1v62_125c	8.780800	
U219	BUFX2M	ss_1v62_125c	8.780800	
U220	BUFX2M	ss_1v62_125c	8.780800	
U221	BUFX2M	ss_1v62_125c	8.780800	
U222	BUFX2M	ss_1v62_125c	8.780800	
U223	NAND4X2M	ss_1v62_125c	13.171200	
U224	NAND4X2M	ss_1v62_125c	13.171200	
U225	NAND4X2M	ss_1v62_125c	13.171200	
U226	INVX2M	ss_1v62_125c	6.585600	
U227	NAND3X2M	ss_1v62_125c	10.976000	
U228	NOR4X1M	ss_1v62_125c	13.171200	
U229	NOR4X1M	ss_1v62_125c	13.171200	
U230	NOR2X2M	ss_1v62_125c	8.780800	
U231	NOR2X2M	ss_1v62_125c	8.780800	
U232	NOR2X2M	ss_1v62_125c	8.780800	
U233	NOR2X2M	ss_1v62_125c	8.780800	
U234	NOR2X2M	ss_1v62_125c	8.780800	
U235	NOR2X2M	ss_1v62_125c	8.780800	
U236	NOR2X2M	ss_1v62_125c	8.780800	
U237	NOR2X2M	ss_1v62_125c	8.780800	
U238	NOR2X2M	ss_1v62_125c	8.780800	
U239	NOR2X2M	ss_1v62_125c	8.780800	
U240	NOR2X2M	ss_1v62_125c	8.780800	
U241	NOR2X2M	ss_1v62_125c	8.780800	
U242	OAI21X1M	ss_1v62_125c	10.976000	
U243	BUFX2M	ss_1v62_125c	8.780800	
U244	BUFX2M	ss_1v62_125c	8.780800	
U245	BUFX2M	ss_1v62_125c	8.780800	
U246	BUFX2M	ss_1v62_125c	8.780800	
U247	BUFX2M	ss_1v62_125c	8.780800	
U248	BUFX2M	ss_1v62_125c	8.780800	
U249	BUFX2M	ss_1v62_125c	8.780800	
U250	NAND2X2M	ss_1v62_125c	8.780800	
U251	NOR2X2M	ss_1v62_125c	8.780800	
U252	INVX2M	ss_1v62_125c	6.585600	
U253	INVX2M	ss_1v62_125c	6.585600	
U254	NAND2X2M	ss_1v62_125c	8.780800	

Area Report

```
****************************************
Report : area
Design : FSM_MSDAP
Version: L-2016.03-SP3
Date   : Mon Dec  6 01:11:08 2021
****************************************

Library(s) Used:

    ss_1v62_125c (File: /home/eng/y/yxw173630/lib/synopsys/ss_1v62_125c.db)

Number of ports:                        2459
Number of nets:                        88068
Number of cells:                       85267
Number of combinational cells:         59330
Number of sequential cells:            25903
Number of macros/black boxes:              0
Number of buf/inv:                     12734
Number of references:                     52

Combinational area:            635484.048283
Buf/Inv area:                  110275.870314
Noncombinational area:        1668461.719595
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:              2303945.767878
Total area:                 undefined
1
```

Qor Report

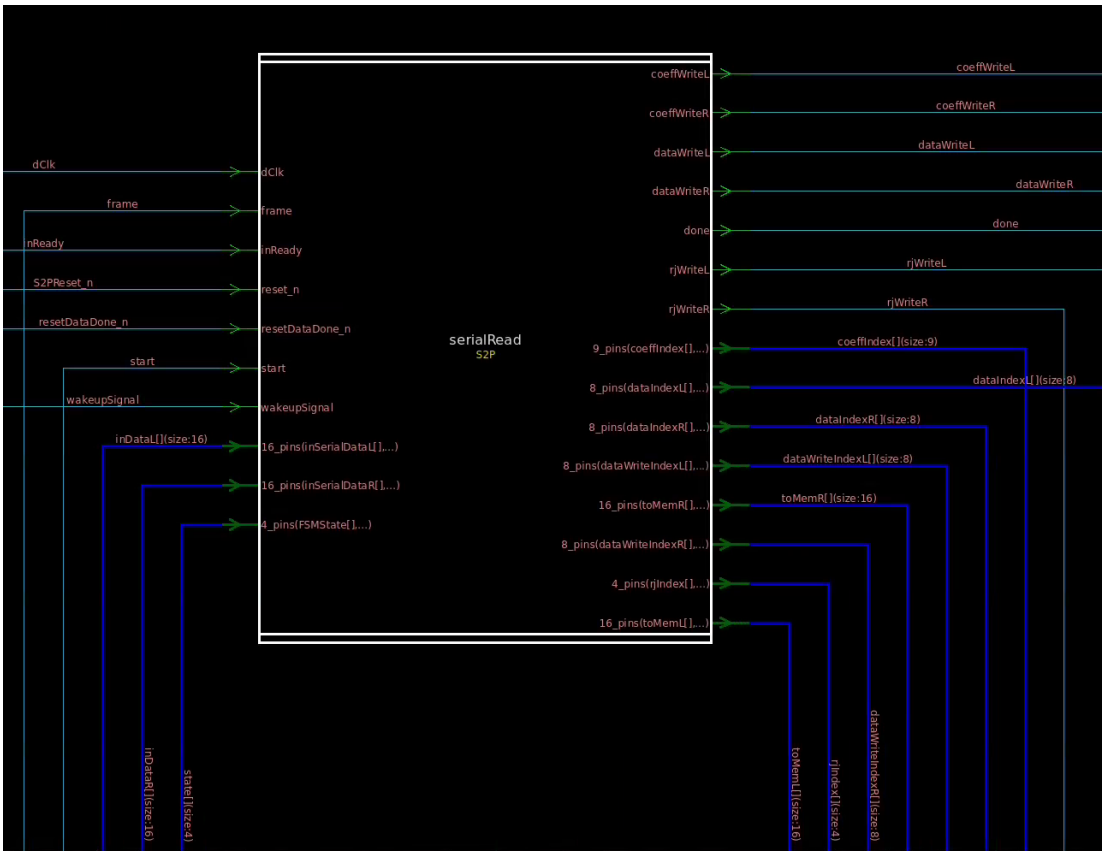
```
****************************************
Report : qor
Design : FSM_MSDAP
Version: L-2016.03-SP3
Date   : Mon Dec  6 01:11:25 2021
****************************************


  Timing Path Group 'dClk'
  -----------------------------------
  Levels of Logic:              10.00
  Critical Path Length:          2.57
  Critical Path Slack:           0.31
  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:               0.00
  Critical Path Length:          0.24
  Critical Path Slack:         -20.24
  Critical Path Clk Period:     18.00
  Total Negative Slack:        -43.73
  No. of Violating Paths:       33.00
  Worst Hold Violation:          0.00
  Total Hold Violation:          0.00
  No. of Hold Violations:        0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:         34
  Hierarchical Port Count:       2340
  Leaf Cell Count:              85170
  Buf/Inv Cell Count:           12734
  Buf Cell Count:               11973
  Inv Cell Count:                 761
  CT Buf/Inv Cell Count:            0
  Combinational Cell Count:     59363
  Sequential Cell Count:        25807
  Macro Count:                      0
  -----------------------------------


  Area
  -----------------------------------
  Combinational Area:   635484.048283
  Noncombinational Area:
                       1668461.719595
  Buf/Inv Area:         110275.870314
  Total Buffer Area:        105169.84
  Total Inverter Area:        5106.04
  Macro/Black Box Area:      0.000000
  Net Area:                  0.000000
  -----------------------------------
  Cell Area:            2303945.767878
  Design Area:          2303945.767878
```

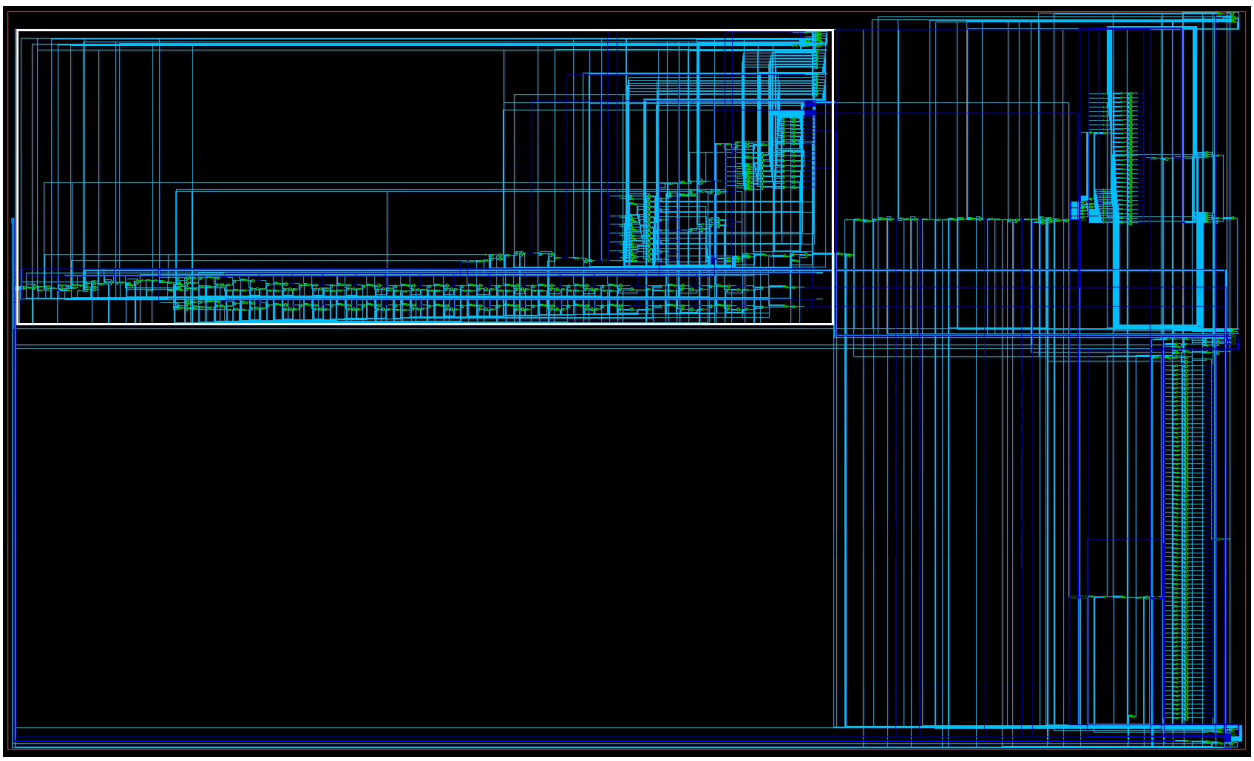
Resource Report

```
****************************************
Report : resources
Design : FSM_MSDAP
Version: L-2016.03-SP3
Date   : Mon Dec  6 01:11:32 2021
****************************************

Resource Sharing Report for design FSM_MSDAP in file
        /home/012/l/la/laa170930/ECS_6360/DCFinal/src/FSM_MSDAP.v

===============================================================================
|          |              |            | Contained     |                      |
| Resource | Module       | Parameters | Resources     | Contained Operations |
===============================================================================
| r88      | DW01_cmp2    | width=4    |               | gte_207              |
| r90      | DW01_inc     | width=4    |               | add_234              |
| r92      | DW01_inc     | width=32   |               | add_293              |
===============================================================================


Implementation Report
===============================================================================
|                    |                  | Current            | Set            |
| Cell               | Module           | Implementation     | Implementation |
===============================================================================
| add_293            | DW01_inc         | rpl                |                |
===============================================================================

1
```

Sub-Block Schematics

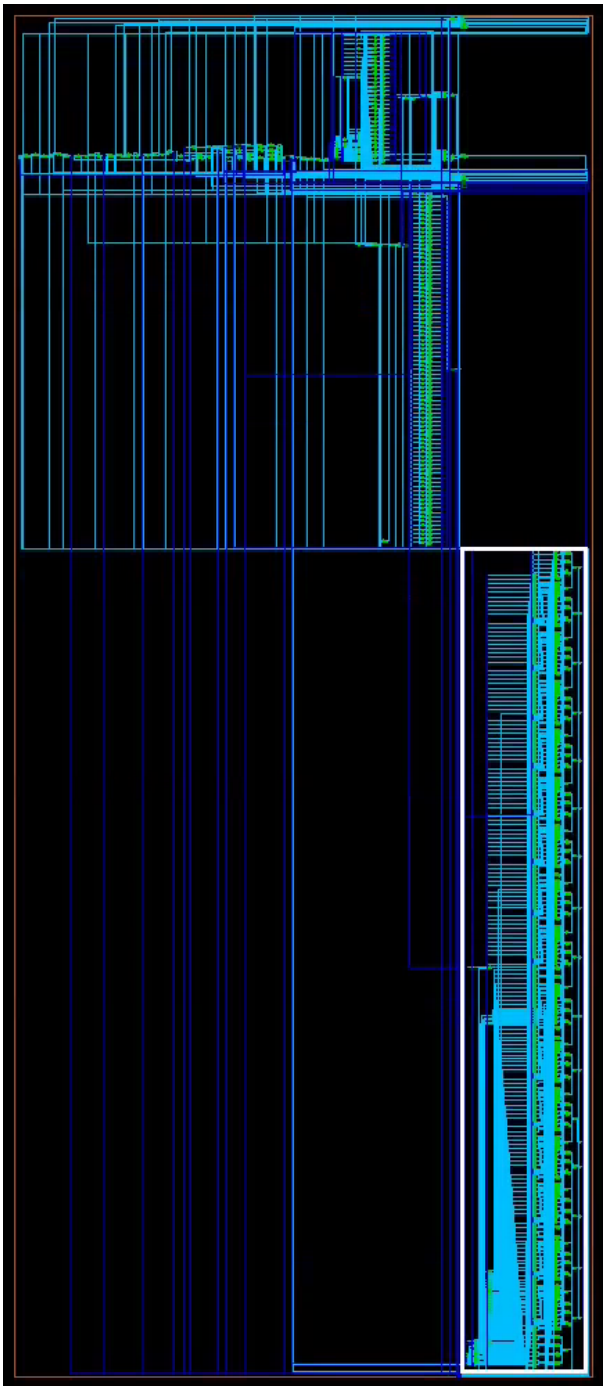
S2P



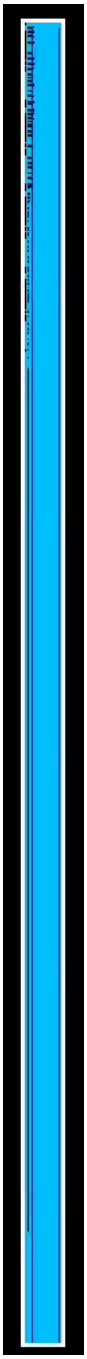


Rj Memory


rjWriteL
writeEnable
rjIndexL[](size:5)
5_pins(rjReadAddr[],...)
rjLeft
RJ_MEM_0
16_pins(rjDataOut[],...)
rjValueL[](size:16)
rjIndex[](size:4)
4_pins(rjWriteAddr[],...)
toMemL[](size:16)
16_pins(rjDataIn[],...)

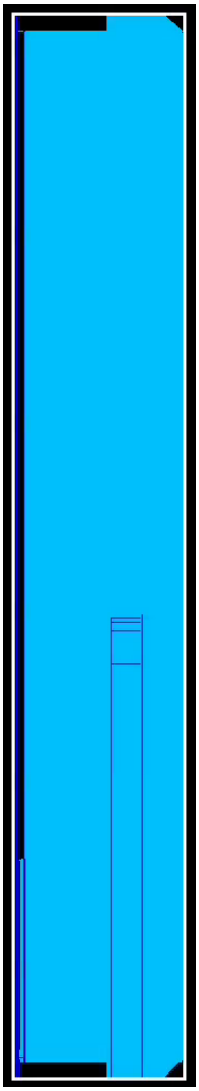


Coeff Memory

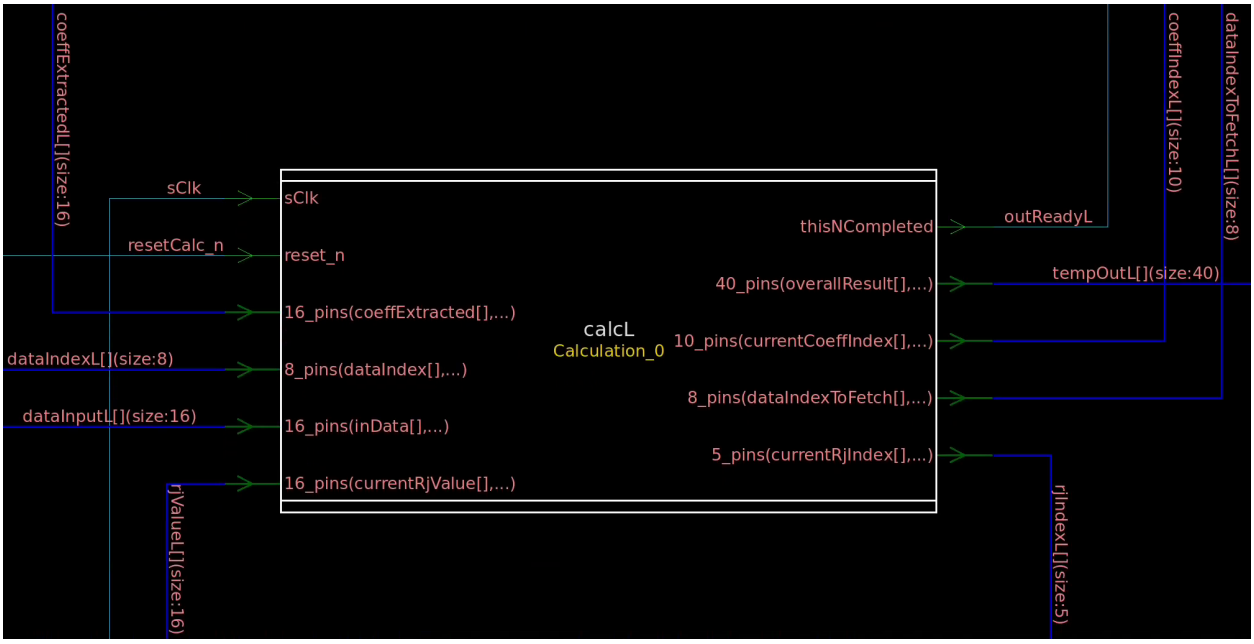


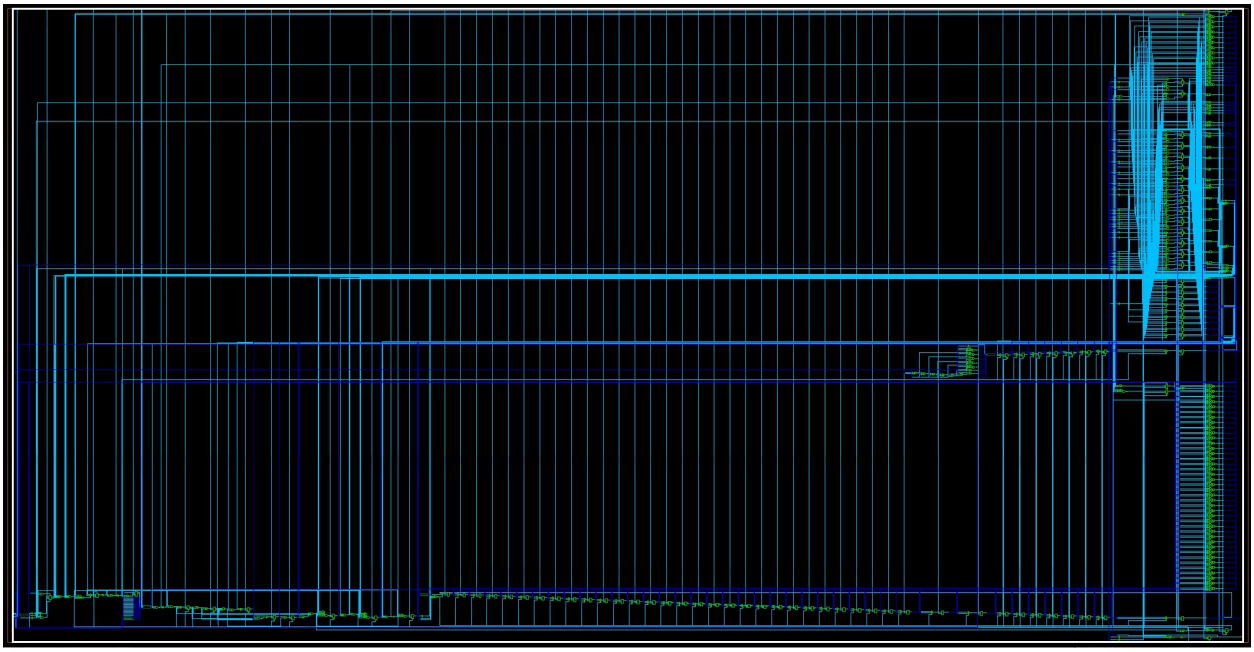
Data Memory


sClk
dataMemReset_nL
dataWriteL
toMemL[](size:16)
sClk
memReset_n
writeEnable
16_pins(dataValueIn[],...)
8_pins(dataReadAddr[],...)
8_pins(dataWriteAddr[],...)
dataLeft
DATA_MEM_0
16_pins(dataValueOut[],...)
dataInputL[](size:16)
dataIndexToFetchL[](size:8)
dataWriteIndexL[](size:8)



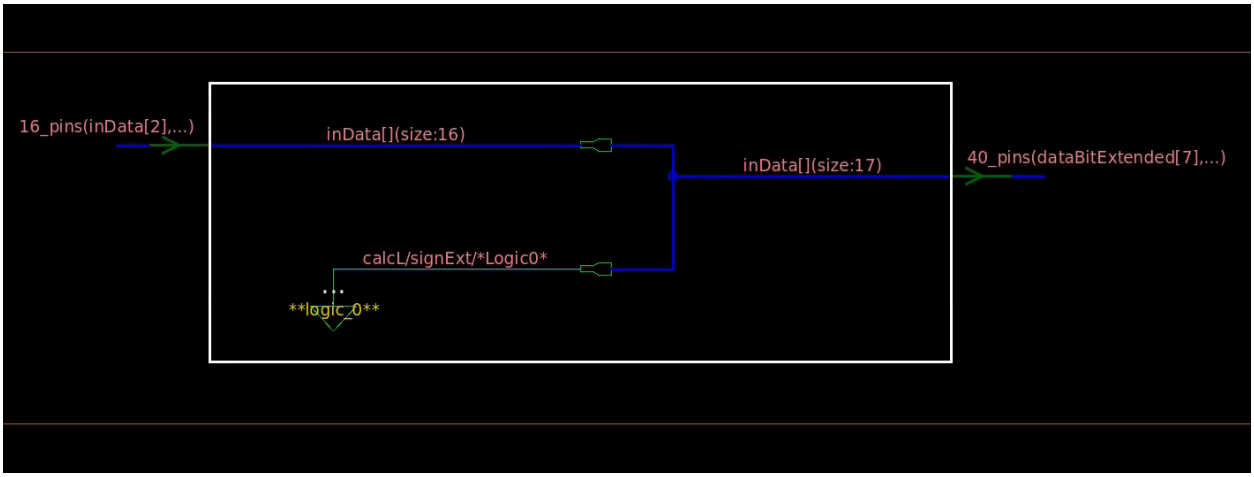
Calculation





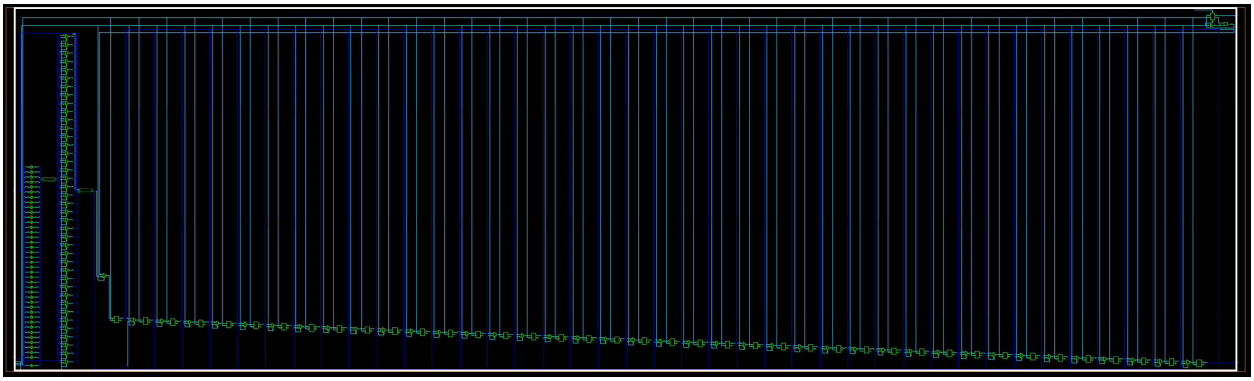
Sign Extend


inData[](size:16)
16_pins(inData[],...)
signExt
calcSignExtend_0
40_pins(dataBitExtended[],...)


16_pins(inData[2],...)
inData[](size:16)
inData[](size:17)
40_pins(dataBitExtended[7],...)
calcL/signExt/*Logic0*
...
logic_0

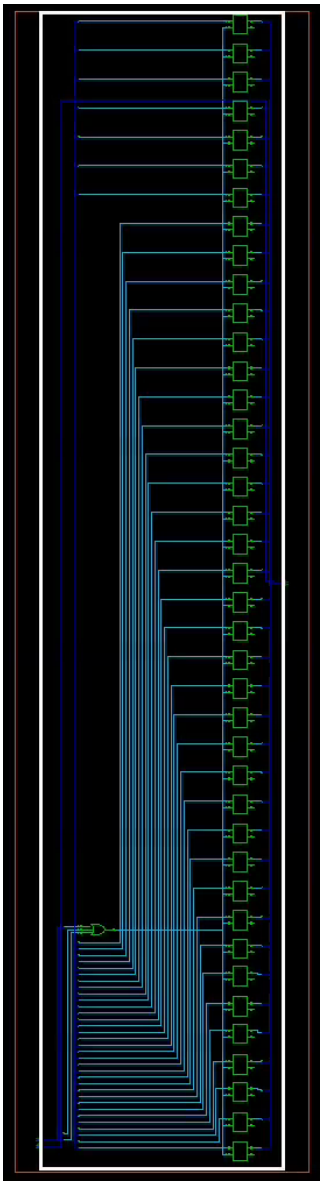
Adder


inOperandB[](size:40)
calcL/ALUReset_n
calcL/sClk
ALUResult[](size:40)
calcL/n24
sClk
calcL/inCoeffSign
inCoeffSign
ALUResultReady
adder
calcAdder_0
ALUReset_n
40_pins(calcResult[],...)
dataBitExtended[](size:40)
40_pins(inOperandA[],...)
40_pins(inOperandB[],...)

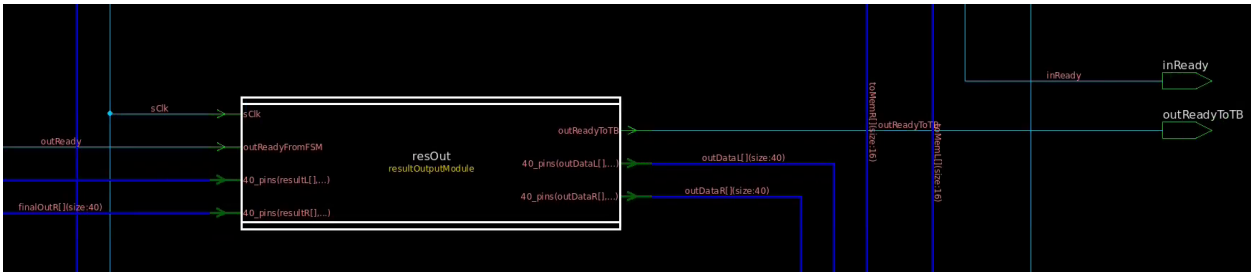


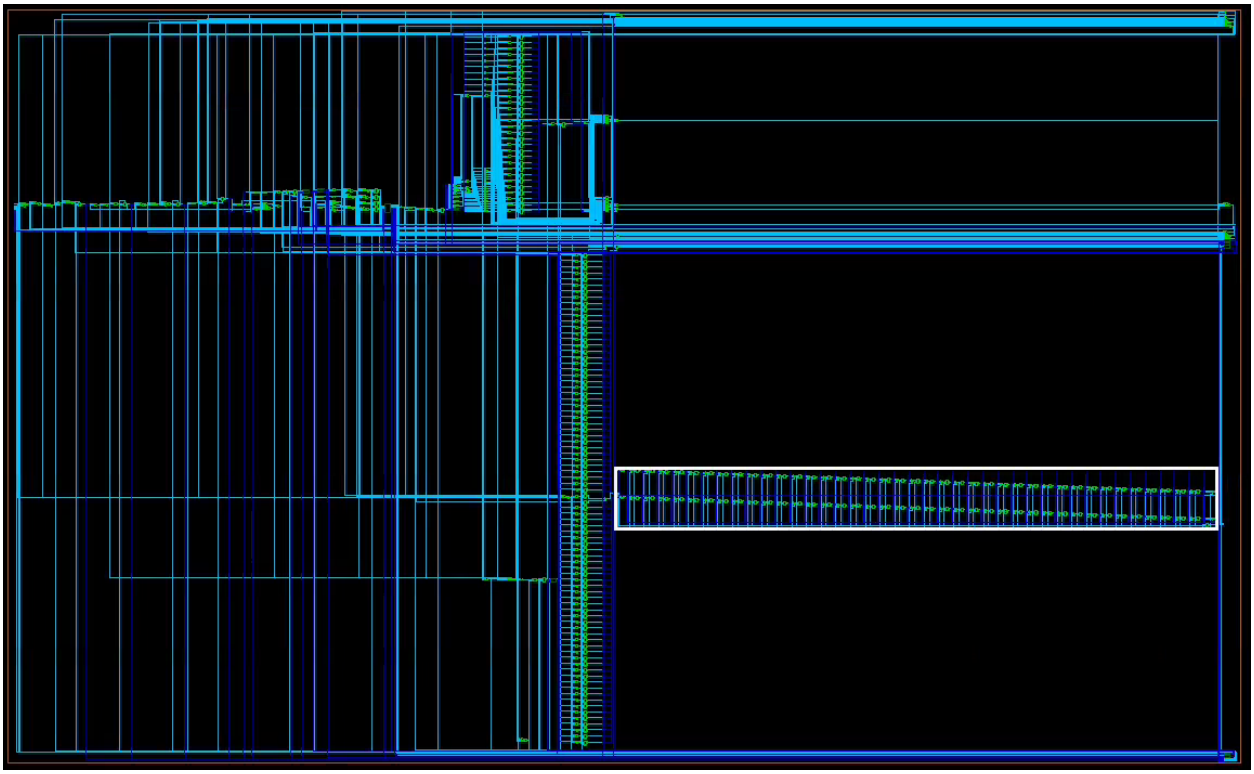
Bit Shifter


calcStage[](size:3)
ALUResult[](size:40)
shifterDataResult[](size:40)
3_pins(calcStage[],...)
40_pins(ALUResult[],...)
bitShift
calcBitShifter_0
40_pins(shifterDataResult[],...)

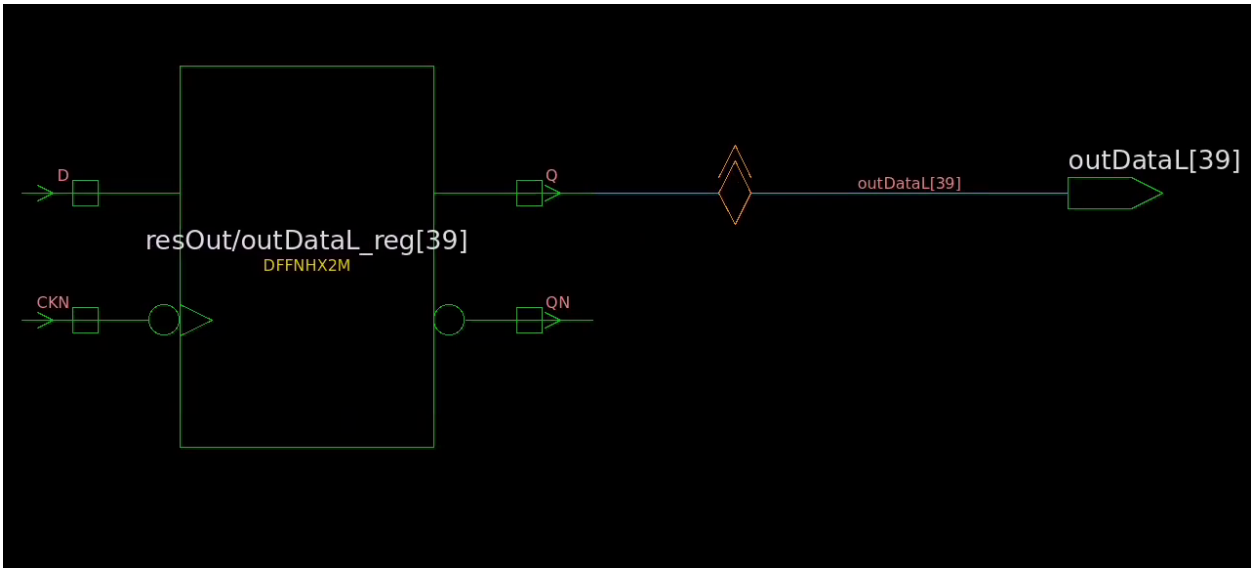


Result Output





Critical Path



Special Topic: How to Verify an ASIC Behavioral Design

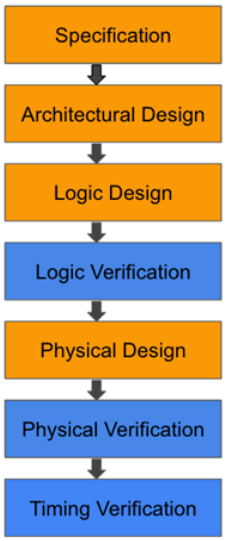
ASIC chips often contain highly complex systems made up of thousands or even millions of gates or transistors. Such a large and complex design introduces a very high likelihood that an error will occur somewhere along the design process. Verification is a very important step in the design process, and typically consumes 70-80% of the total time budget. Therefore, it is crucial to identify and correct these errors early on as they occur, in order to save time and money. The three main motivations for verification are as follows:

1) During specification: “Is what I am asking for what is really needed?”

2) During design: “Have I indeed designed what I have been asked?”

3) During testing: “Can I tell correct circuits from malfunctioning ones?”

Shown in the figure below is the ASIC design workflow. It can be seen that after every major design phase, verification is performed.



The goal of verification is to ensure that the design meets the requirements of the specification. Naturally, the first step to successfully verifying an ASIC design is to create a well-defined specification. The specification will repeatedly be referenced to ensure that the design is proceeding in such a way that ensures efficient design, as well as to check that completed modules perform as expected. Multiple stages of verification will need to be performed throughout the design process in order for this technique to be effective. The benefit of this is that errors will be found earlier in the design process, when the design is easiest to modify, as the more time goes on the more difficult modification of the design becomes. For this project, much of the specification was given to us via the project requirements. Those requirements were taken, and a specification was created based on them. For example, we knew what input and output ports were needed, so we clearly defined this in the specification, along with the size of each port and a description of their use. FSMs and timing requirements can also be determined at this point. From the functional description that we were given for this project, we were able to define our FSMs to work as described. This went a long way to ensuring the proper design whenever we began

to work in Verilog. Finally, simply documenting the convolution calculation was an important resource that was later referenced when trying to troubleshoot the calculation modules.

As discussed above, the specification can be used during the design phase in order to ensure the verification of the design as quickly and efficiently as possible. Therefore, the second step of verification is to ensure the theoretical design meets the specification during the design phase. Considering the specification during the design seems obvious but having a concrete set of guidelines helps locate and correct potential issues. During the design of the MSDAP device, the specification was referenced numerous times to ensure signals are the correct size, data was being output/read at the correct time, events are sensitive to the correct signals, FSM states perform the correct functionality, etc.

Finally, the bulk of verification happens during the testing stage. The goal of testing is to verify the design matches the spec by confirming the device functions as expected. Testing consists of proving the device with some stimuli, capturing the output, and analyzing the output to verify it is as expected. The below figure illustrates the standard testing procedure.

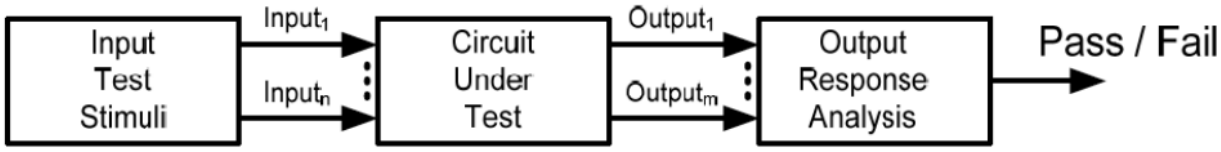


Figure 8-3 Basic testing approach (Wang, Wu, & Wen, 2006)

This testing process is straightforward for the RTL or logic level verification. The design is, at this point, in some sort of hardware verification language, and a testbench can be coded to run a comprehensive test on the design. For the MSDAP project, our testbench reads in input data for a test case, sends it to the MSDAP device in a way which simulates an actual use case, and provides some feedbakc to the user in the form of messages in the console. Once the simulation is ran, the resulting data is output to a file. To analyze our data, we created a python script which compares the result data with the expected output. If an error is found, it is easier at this stage to troubleshoot, as the testbench waveforms allow the user to visualize exactly where the issue occurred. In the MSDAP project, the majority of the time was spent in this stage, looking through the waveforms and debugging.

Once the bugs have been worked out in this stage, the design can be synthesized and converted into a representation of physical logic. At this point, several disgn choices may need to be changes in order to ensure synthesizability. For example, in our MSDAP design, we used both loops and delays, both of which are unsynthesizable. Therefore, we had to go back and modify our logic design to work properly without these. At this point in the design process, changes are the most difficult to make, however most of the issues in the design should have been worked out by now. It is possible for timing issues to arise after synthesis, and verification of the synthesized design still needs to be done. This synthesized design should be simulated using a similar testbench to that which was used to test the RTL level design. Again, it is a little more difficult to debug at this stage, as some parts of the design will be difficult to read. If an error is found, it should be located using the waveforms and corrected on the RTL level. The verification process and synthesis will have to be re-done from that point on.

References:

1. *The Ultimate Guide to ASIC verification*. AnySilicon. (2021, March 18). Retrieved December 6, 2021, from https://anysilicon.com/the-ultimate-guide-to-asic-verification/.