CE 6303.001

Midterm Report

**Luke Allen** NetID: laa170930

**Yu Feng** NetID: yxf160330

11/3/2021

# Problem Description:

For homework 6, we created the calculation module which does the convolution computation. For homework 7, we created the top-level FSM which implements the behavioral model of the MSDAP chip without the computation module.

Now, the problem we were tasked with for our midterm project is to combine our work from homeworks 6 and 7 to create the behavioral model for the entire MSDAP chip as it is described in the MSDAP paper and given specification.

We also must create a testbench which acts as the controller in this scenario, and will generate the start, reset, frame, and clock signals, as well as read input data from a file, send inputs to the MSDAP module, and accept outputs from the MSDAP module.

Our assigned input and output modes are serial for input, and parallel for output. Start and frame are active high, reset\_n is active low.

# 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:

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

Data: 3C4A

|  |
| --- |
| Rj |
| 02 |
| 03 |
| … |

|  |
| --- |
| Coefficients |
| 01B |
| 00C |
| 012 |
| 002 |
| 00A |
| … |

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:

Diagram

Description automatically generated

## 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:

Diagram

Description automatically generated

**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:

Diagram

Description automatically generated

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:

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:

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

# Implementation:

A general overview of our program is as follows:

Top-Level Module

We have an always statement sensitive to the falling edge of dClk which handles the reading of all data from the controller. This statement will read one bit at a time, storing it in a temporary register at the correct bit position. When all bits are read, it stores it in its corresponding array, or if it is a data sample, sends it to the calculation module. The current state of the finite state machine will determine where this block stores the input data.

Next, we have an always statement sensitive to the rising edge of sClk and outReady. This block handles writing out the calculated data to the controller. The outReady signal indicates when output data is ready to be sent, then it simply outputs the data on the 40-bit output bus.

Next, we have two always blocks which handle the data delivery to the two calculation modules. They are sensitive to each of the calculation modules ready for data signals. These always blocks calculate the end coefficient index, obtain the coefficient needed by the calculation module from the array of coefficients, calculate the sign and value of the coefficient, calculate the data index needed, and obtain and send the data required by the calculation module. The obtained values are passed via inputs to the calculation module.

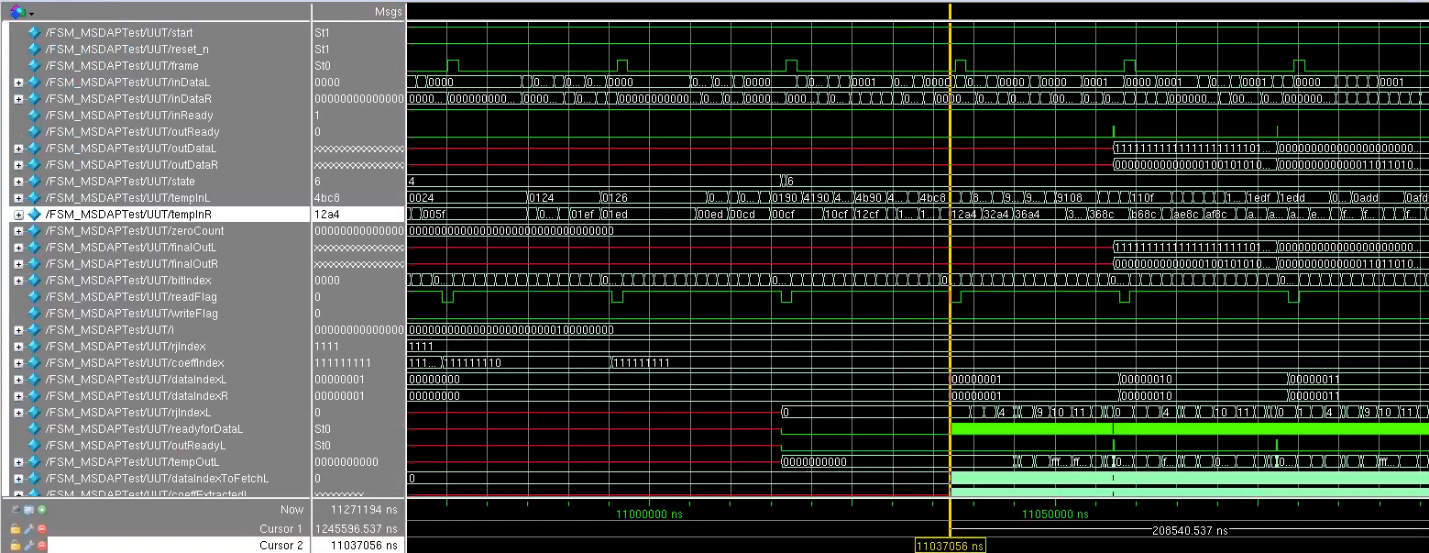
Our largest block is an always statement sensitive to the rising edge of sClk and the falling edge of reset. This block contains the logic for our finite state machine. First, if reset is detected to be low, and the FSM is in states 5 through 8, the state is set to 7 (reset state) and clears the data. Otherwise, this block performs the task of whatever state it is in. Each of the states performs its logic according to what is described in the FSM provided and determines the next state. The state is then updated on the rising edge of sClk. In our working state, we are checking to see if the calculation modules have calculated data that is ready to be output. Both channels must be synchronized, so we wait until both the left and right channels have output data ready, and then we pass it to the output of the top-level module. Also in this state, as data comes in, we are checking if the input data is all 0’s. We will keep track of the number of consecutive zeros, and if it reaches 800, we set the next state to state 8 (sleeping state).

Calculation Module

The calculation modules are almost the same modules that we created in homework 6. Since now the module is communicating with the top-level module instead of a testbench, we had to make a few changes.

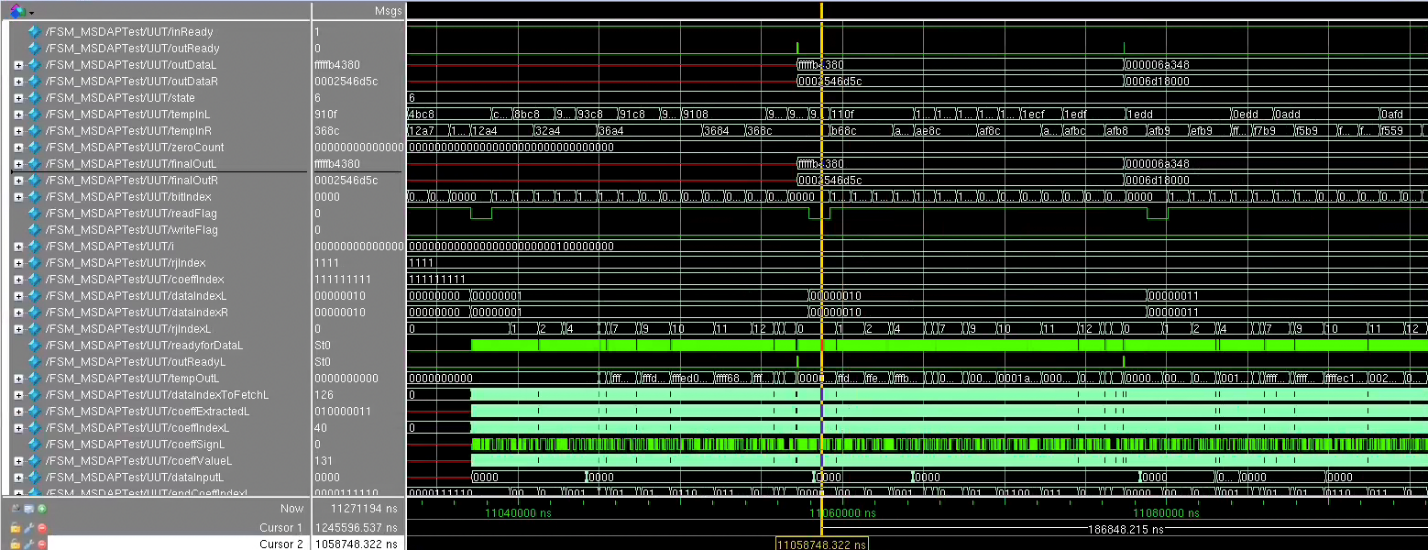
Our calculation module contains another finite state machine which handles the convolution computation. This is the only always block in our module, and it is sensitive to the rising edge of the system clock and the negative edge of resetCalc. Each state does a part of the convolution computation, and when it is done it moves to state 2. In this state, the module signals to the top-level FSM that an output is ready and stays in this state. As mentioned before, the top-level FSM must wait for both channels to finish calculating, and then it will output data for both channels. Therefore, the calculation module stays in this state until both modules are ready to output. Then, the top-level module will grab the data and reset the calculation modules. When this happens, the calculation modules will clear all data and return to state 0.

# Simulation Results:



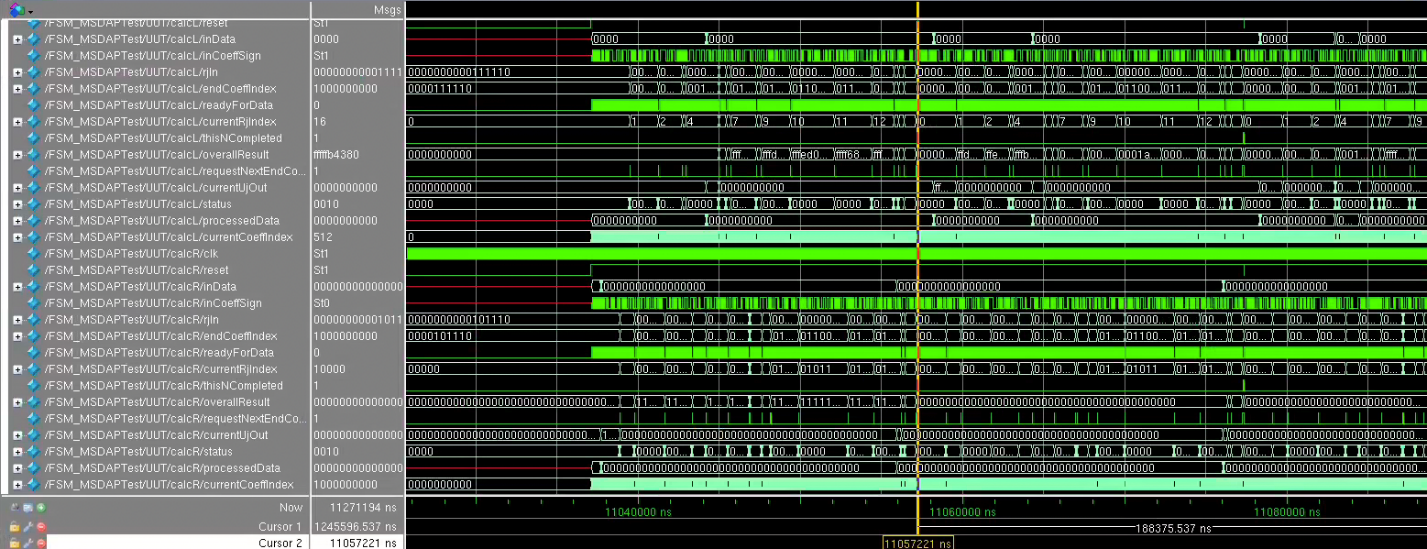
*Figure 1: Waveform demonstrating reading a data sample*

The above waveform demonstrates the input reading functionality. We can see the inDataL and inDataR channels accumulating our input data, and then being moved to temp in. At this point in time, our entire first data sample for both channels have arrived. As you can see, the tempInL is 4BC8 and the tempInR is 12A4.



*Figure 2: Waveform demonstrating outputting calculated data*

The above waveform demonstrates the output functionality. When data is finished calculating on both the left and right channels, the outReady flag is raised, and the output of the calculation modules are sent to the output of the top module. Here, you can see the results of calculating our first data sample. outDataL is FFFFFB4380 and outDataR is 0002546D5C.



*Figure 3: Waveform demonstrating the operation of the calculation module*

The above waveform demonstrates the functionality and internal signals of the calculation module.

Diagram

Description automatically generated

*Figure 4: Waveform demonstrating sleep functionality*

The above waveform shows the FSM going into sleep mode, or state 8, after the zeroCount variable, which keeps track of how many consecutive zero inputs we get on both the left and right channels, reaches 800.

Diagram, schematic

Description automatically generated

*Figure 5: Waveform demonstrating reset functionality*

Finally, the above waveform demonstrates the reset functionality. Since our reset is active low, at the negative edge of the reset\_n signal, the FSM briefly goes into state 7, all variables and memories are cleared, and then it moves to state 5.