CE 6303.001

HW 7 Report

**Luke Allen** NetID: laa170930

**Yu Feng** NetID: yxf160330

10/20/2021

# Problem Description:

The problem we were tasked with for Homework 7 was to implement the finite state machine to implement the MSDAP chip as described in class, save for a few minor differences.

For this homework, we will not be implementing the algorithm to calculate the convolution that we created in homeworks 5 and 6. Instead, we are only creating the finite state machine that provides the framework for this algorithm, and we will be simply sign extending the input data rather than performing the convolution. Additionally, we will only be considering one input channel, and we will go into sleep mode after 10 consecutive 0 inputs.

The testbench for this homework will generate start, reset, frame, and clock signals, as well as read input data from a file similar to the files used in homework 5, send inputs to the MSDAP module, and accept outputs from the MSDAP module.

# Algorithm Implementation:

A general overview of our program is as follows:

We have an always statement sensitive to the falling edge of dClk which handles the reading of all data from the controller. 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 one bit at a time on each positive edge of sClk.

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 do the sign extension and set outReady to 1 to indicate data is ready to be sent out. 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 10, we set the next state to state 8 (sleeping state).

# Simulation Results:

The following images are screenshots of out output waveform results from our testbench. All of the inputs and outputs are shown in the waveforms, as well as some internal registers and signals for reference.

Figure 1 shows the beginning of the simulation. You can see that some time into the simulation, the start signal goes high. At this point, all of the registers are initialized to zero, as shown. At the next rising edge of the sClck, the state proceeds to state 1 (which can be seen in the state register), where it will raise the inReady signal and wait for the frame signal to go high.

Diagram

Description automatically generated

*Figure 1: Output waveform at the start of the testbench*

Next, Figure 2 shows the transition between states 2, 3, and 4. Once all 16 values of Rj are read in (which can be seen in the tempIn register) the done signal is set to 1, which tells the FSM to move on to the next state. The done signal is set back to zero after a very short time, and the FSM proceeds to state 3, where it waits for a frame signal. Once the frame signal is high, it then moves on to state 4.

Diagram

Description automatically generated

*Figure 2: Transition between states 2, 3, and 4*

Next, figure 3 shows the transition between states 4, 5, and 6, and demonstrates how the program gets the input data. Similar to Figure 2 with states 2 and 3, the done signal is raised whenever all 159 coefficients are read in. Done is set back to zero after a very short time and signals the FSM to move on to state 5. State 5 waits for the frame signal to go high and then moves on to state 6, where it begins reading in data values, since inReady is high. The data value being read in Figure 3 is C48A. You can see the value in tempIn slowly fill into C48A at every falling edge of dClk as each individual bit is read in. We used the bitIndex register to keep track of which bit of the data is being read in.

A picture containing graphical user interface

Description automatically generated

*Figure 3: Transition between states 4, 5, and 6, and data input*

Next, Figure 4 shows how the data is output. In contrast to reading the coefficients and Rjs, when we are reading data, we used the done signal to indicate when all 16 bits of one data value has been received. This allows state 6 to perform the sign extension on the input data. As shown below, the done signal is set back to zero after a very short amount of time, and the tempOut and finalOut registers are updated to the sign extended input. Since our input data was C48B, our sign extended result is FFFFFFC48B, which is shown in our output registers. Once this is done, the outReady signal is raised and the 40 bits of output data are sent at the rising edge of each sClk. We use the outCount register to keep track of which bit of the output data is being sent.

Graphical user interface

Description automatically generated

*Figure 4: Writing the output data*

Next, Figure 5 shows the sleep functionality. We used the zeroCount register to keep track of the number of consecutive all zero inputs we have received. As shown below, the zero count is at 9, and the temp in is 0000. You can see the zeroCount briefly get set to 10, and then on the next rising edge of sClk, the FSM goes into state 8 (sleeping mode). When any nonzero input is detected, the FSM will awaken and resume in state 6. This is shown in Figure 6. The state is at 8, and then a new nonzero input is received, which is shown in tempIn as 20B6. As soon as the first nonzero bit is received, it goes back into state 6 and awaits the remaining bits to be read in so it can calculate the sign extension.

Graphical user interface

Description automatically generated

*Figure 5: Going into sleep mode*

Diagram

Description automatically generated

*Figure 6: Awakening from sleep mode*

Finally, Figure 7 demonstrates the reset functionality. The reset\_n signal is set to zero, and on the falling edge of the signal, the state is changed very briefly to state 7, where it clears all of the registers to zero, and then proceeds to state 5.

Graphical user interface, diagram

Description automatically generated

*Figure 7: Resetting the system*