The purpose of this second homework assignment is to provide an introduction to different system level design concepts, which includes DataPath and Controller partitioning. The Verilog HDL is used in order to model different system constructs.

**PART 1** – **G6 System Design**

After reading the homework assignment for the G6 system design, and getting an understanding of all handshaking that needs to occur between the DataPath and Controller of the system, the following diagrams were generated, which effectively characterize all necessary components of the system:



The arrows in **blue** in the DataPath diagram above represent the different signals coming from the Controller of the G6 system. The system level diagram for the G6 controller is shown below:



A summary of the different Controller states is as follows:

**`idle:** Wait for pulse on *start* to begin (*start = 1*) circuit operation.

**`init:** Wait for pulse on *start* to end (*start = 0*) before continuing.

**`s1-s6:** On the next six pulses of *clk*, read input into the 6 byte register.

**`h1:** Send out a *request* to use *iobus*; requires 1 *clk* cycle.

**`h2:** Wait to receive a *grant* to use the *iobus* (*grant = 1)*.

**`h3:** Wait to receive notification that the data was *received*, then reset.

All Verilog code for the Controller of the G6 design, with detailed comments that go into detail regarding the definition of each state, can be found in the Pt 1 folder of the HW 2 ZIP file submitted for this assignment:

*Controller.v*

This concludes the analysis for Homework 2, Part 1.

**PART 2** – **1011 Moore Machine Sequence Detector**

In order to develop the code for the 1011 Moore Machine Sequence Detector, it is important to recall the state transitions required for this system:

**State Diagram of the 1011 Moore Machine Sequence Detector**

1

0

1

1

0

1

0

0

1

0

At this point, it is a matter of translating the diagram shown above into Verilog code. The diagram above will account for overlapping sequences or interruptions in the generic 1011 sequence.

All Verilog code for the 1011 Moore Machine Sequence Detector is listed below:

// Carlos Lazo

// ECE505

// Homework 02 - Pt2

// Moore Detector for Sequence 1011

`timescale 1ns/100ps

module MooreDetector1011 (input A, clk, R, output W);

parameter [2:0] sA = 3'b000, sB = 3'b001, sC = 3'b010, sD = 3'b011, sE = 3'b100;

reg [2:0] current;

// In order to provide an asynchronous reset variable R,

// add R to the sensitivity list and check it's state first,

// before going into the state-transition logic.

always @(posedge clk or R) begin

// If R = 1, reset the state machine back to the initial state of A.

if (R)

current <= sA;

else begin

// Implement all state transitions based on sequential logic.

case (current)

sA: if (A) current <= sB; else current <= sA;

sB: if (A) current <= sB; else current <= sC;

sC: if (A) current <= sD; else current <= sA;

sD: if (A) current <= sE; else current <= sC;

sE: if (A) current <= sB; else current <= sC;

endcase

end

end

// Assign the output W = 1 if we are at the last state of E.

assign W = (current == sE) ? 1 : 0;

endmodule

An actual copy of the Verilog code can be found in the Pt2 folder of the HW 2 ZIP file submitted for this assignment:

*MooreDetector1011.v*

This concludes the analysis for Homework 02, Part 2.

**PART 3** – **S2PC & P2SC Design**

**Subpart A – S2PC Design**

All Verilog code for the Controller & Datapath of the S2PC design, with detailed comments that go into detail regarding the definition of all datapath components and overall controller logic states, can be found in the Pt 3 🡪 Pt A folder of the HW 2 ZIP file submitted for this assignment:

*S2PC\_Datapath.v*

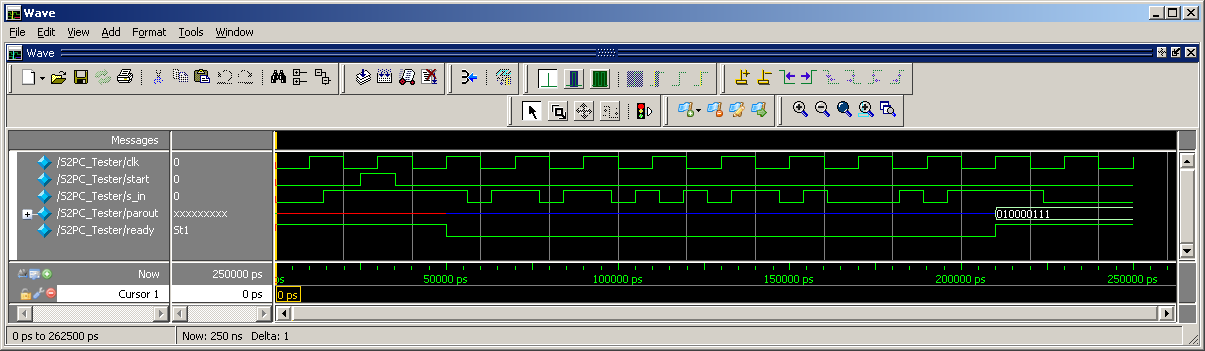
*S2PC\_Controller.v*

*S2PC.v*

*S2PC\_Tester.v*

A quick testbench was written to prove functionality of the S2PC design, given a periodic clock at 10 ns and the *start* signal pulsing 1st between 25 – 35 ns:

**Waveform Output for S2PC Testbench**



As can be seen above, once the system sees a pulse on *start*, it must wait a full *clk* cycle to ensure that the *start* signal is deasserted. This means that the reading of *clk* cycles will begin @ 70ns, and will continue for 8 total cycles. After the 8th cycle, the *done* signal is asserted within the Controller, and the 9 bit output is made available on *parout*, with the MSB being a parity bit.

This concludes the analysis for Homework 2, Part 3, Subpart A.

**Subpart B – P2SC Design**

All Verilog code for the Controller & Datapath of the P2SC design, with detailed comments that go into detail regarding the definition of all datapath components and overall controller logic states, can be found in the Pt 3 🡪 Pt B folder of the HW 2 ZIP file submitted for this assignment:

*P2SC\_Datapath.v*

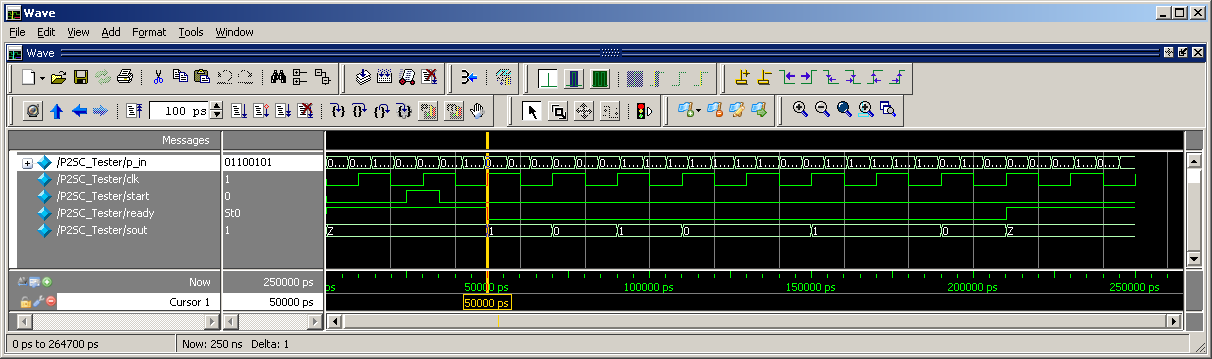
*P2SC\_Controller.v*

*P2SC.v*

*P2SC\_Tester.v*

A quick testbench was written to prove functionality of the P2SC design, given a periodic clock at 10 ns and the *start* signal pulsing 1st between 25 – 35 ns:

**Waveform Output for S2PC Testbench**



As can be seen above, once the system sees a pulse on *start*, it waits for the *clk* signal when *start = 0* to continue. Once this occurs, a snapshot of *p\_in*, the 8-bit randomly generated input, is captured. In this same clock cycle, data is transmitted, 1-bit at a time, via the *sout* signal. As shown, *p\_in* captures the data input 01100101. For the next consecutive 8 *clk* cycles, it is shown that *sout* correctly outputs the sequence, bit by bit, starting with the LSB of *p\_in*: 1 > 0 > 1 > 0 > 0 > 1 > 1 > 0

This concludes the analysis for Homework 2, Part 3, Subpart B.

**Subpart C – P2SC to S2PC Design**

All Verilog code for the Testbench of the P2SC to S2PCdesign, along with all Verilog files from Subparts A & B, can be found in the Pt 3 🡪 Pt C folder of the HW 2 ZIP file submitted for this assignment:

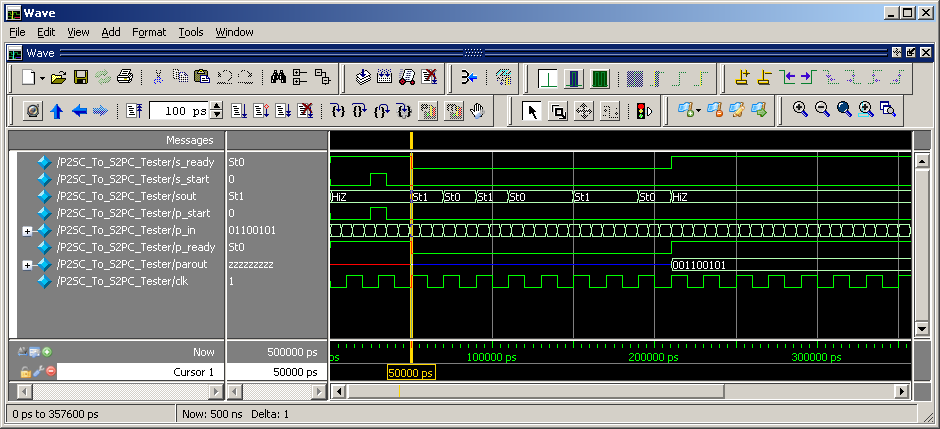
*S2PC\_Datapath.v, S2PC\_Controller.v*, *S2PC.v,*

*P2SC\_Datapath.v, P2SC\_Controller.v, P2SC.v*

*P2SC\_To\_S2PC\_Tester.v*

A testbench was written to prove functionality of the P2SC to S2PCdesign, given a periodic clock at 10 ns and both *start* signals pulsing 1st between 25 – 35 ns:

**Waveform Output for P2SC to S2PC Testbench**



The testbench above shows a working design. At first, the P2SC register stores a value of 01100101 as the parallel data received. On the *clk* pulse following *start*, it immediately begins parsing the data and sending it, LSB first, via *sout*. At this same point, S2PC begins taking serial data from *sout* and stores it in its register. Once all 8-bits from P2SC are sent, it is shown that *parout* correctly displays the original sequence of 01100101, with the added 0 in front for the parity bit. This clearly demonstrates the serial operation of the P2SC and S2PC, working together in this testbench.

This concludes the analysis for Homework 2, Part 3, Subpart C.

**Subpart D – S2PC to P2SC Design**

All Verilog code for the Testbench of the S2PC to P2SC design, along with all Verilog files from Subparts A & B, can be found in the Pt 3 🡪 Pt D folder of the HW 2 ZIP file submitted for this assignment:

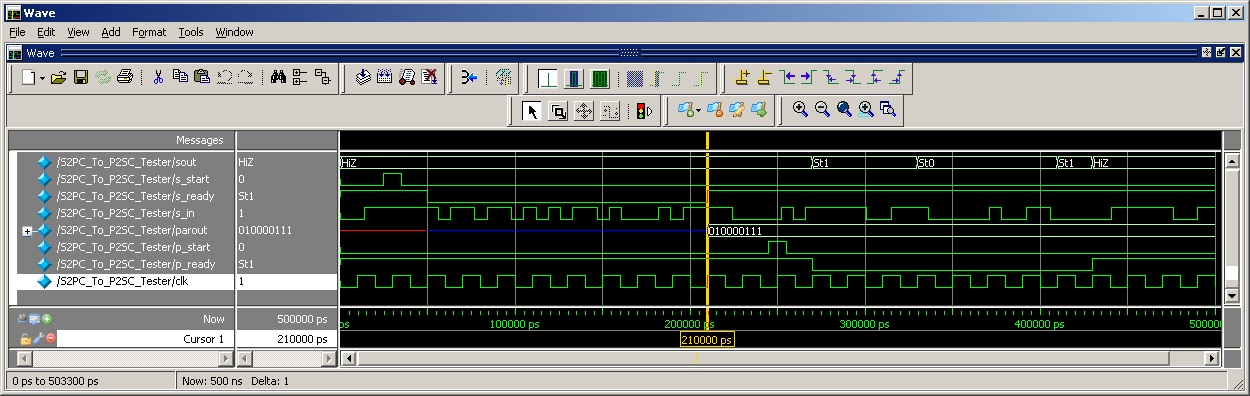
*S2PC\_Datapath.v, S2PC\_Controller.v*, *S2PC.v,*

*P2SC\_Datapath.v, P2SC\_Controller.v, P2SC.v*

*S2PC\_To\_P2SC\_Tester.v*

A testbench was written to prove functionality of the S2PCto P2SC design, given a periodic clock at 10 ns and the *start* signals pulsing at different times:

**Waveform Output for S2PC to P2SC Testbench**



The testbench above shows a working design. At first, the S2PC register stores receives random bits on the *s\_in* input line, and after 8 consecutive pulses on *clk*, it collects the sequence 10000111, and appends the parity bit of 0, leaving the full 9 bit sequence of 010000111. Shortly thereafter, the *start* signal of the P2SC is enabled, and it takes the last 8 bits of the *parout* signal and turns them into serial data for the next 8 *clk* pulses. Starting with the LSB of the 8 bit sequence, analysis of the *sout* signal shows the following sequence output: 1 > 1 > 1 > 0 > 0 > 0 > 0 > 1, which is the original input seen, starting with the LSB. This clearly demonstrates the serial operation of the S2PC and P2SC, working together in this testbench.

This concludes the analysis for Homework 2, Part 3, Subpart D.

**PART 4** – **ALU Design**

This part of the assignments asks us to develop an implementation of an Arithmetic Logic Unit (ALU) given a table of 8 complete functions. My ALU design was built around 8-bit inputs and an 8-bit output.

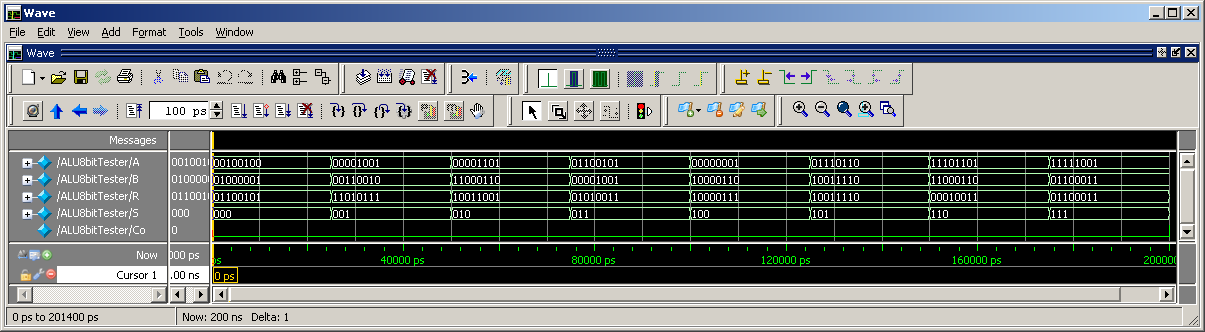
A copy of all Verilog code developed for this section of the homework can be found in the Pt4 folder of the HW 2 ZIP file submitted for this assignment:

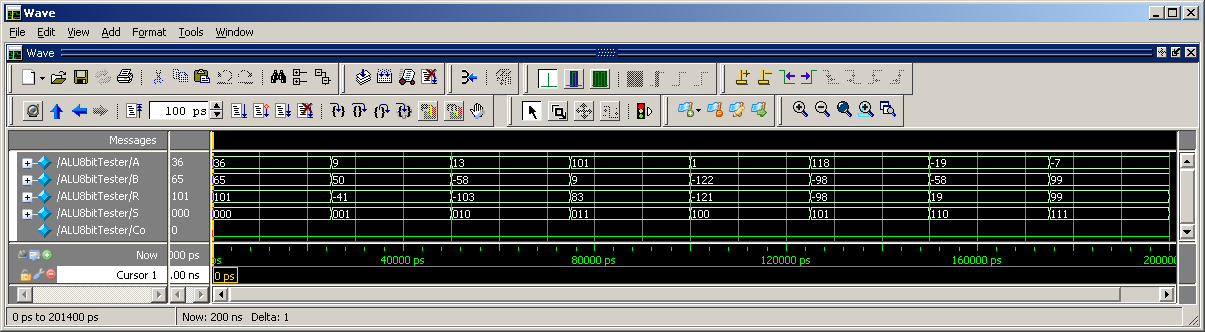
*ALU8bit.v*

*ALU8bitTester.v*

The images below represent the output from the simple ALU8bit testbench created, and will validate the operation of the ALU all together. Note that both images represent the same output, but one is in binary and the other in decimal:

**8bitALUTester Waveform Output**





Here is a more detailed analysis of the diagrams seen above, with the timings dependant on the time changes of the ALU mode variable *S*:

@ 000 🡪 A + B: **CORRECT**

A = 36 B = 65 R = 101

@ 001 🡪 A - B: **CORRECT**

A = 9 B = 50 R = -49

@ 010 🡪 A + 2\*B: **CORRECT**

A = 13 B = -58 R = -103

@ 011 🡪 A - 2\*B: **CORRECT**

A = 101 B = 9 R = 83

@ 100 🡪 A ^ B: **CORRECT**

A = 00000001

B = 10000110 R = 10000111

@ 101 🡪 min(A,B): **CORRECT**

A = 118 B = -98 R = -98

@ 110 🡪 Abs(A): **CORRECT**

A = -19 B = -58 R = 19

@ 111 🡪 B: **CORRECT**

A = -7 B = 99 R = 99

This concludes the analysis for Homework 02, Part 4.