Final Project Report

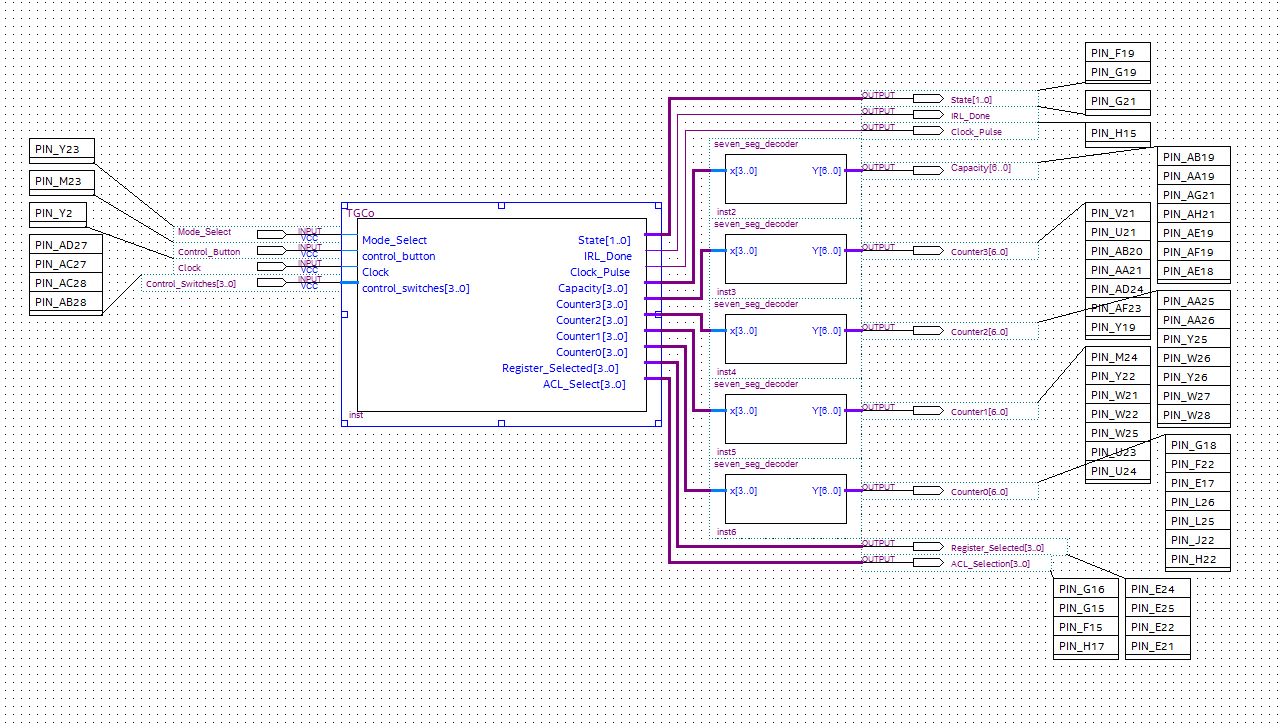
Sean Griffen

CprE 281 Fall 2018

Section V

For my final project, I followed the specifications for the provided project, Project #3: Traffic Light Fairness System. The following report contains all block diagram files and Verilog code used to complete this project, starting with the top-level entity of every module and ending with the lowest-level entity.

**Module FPM:**

**

This is the diagram of the module Final\_Project\_Machine or FPM, as it is named in the project files and .bsf’s. It is the module that connects the desired switch and button inputs and outputs of the TGCo module, which will be described in detail later in this report. Here is a description of the inputs and outputs.

Inputs:

Mode\_Select:

This is controlled by switch 17on the board and controls the mode of the system; switch = 0 being mode A (mode specified in the project description) and switch = 1 being mode B (mode specified by the project description).

Control\_Button:

This is controlled by key 0on the board and is only used to select the capacity for the lane, starting from lane 0 to lane 3, during the initial register file loading mode (mode specified in the project description). When the button is pushed, the program assumes that you are submitting the desired capacity for that lane and updates the register file on the next clock cycle.

Clock:

This is the provided 50MHz clock on the board. It is used to provide clocks to the finite state machine, 4\_4bit\_Register\_File module, TLCF module, IRL\_Machine module, and ACL\_Machine module, all of which will be described later in this report.

Control\_Switches:

These switches are mapped to switches 0, 2, 3, and 3 on the board and, while in the initial register file load mode, control the capacity for the lanes, and, while in Mode A or Mode B, control which lane gets cars added to it, all switches off being no lane selected, switch 0 on being lane 0 selected, switch 1 on being lane 1 selected, switch 2 on being lane 2 selected, and switch 3 being lane 3 selected.

Outputs:

State:

This is mapped to LED’s RLED0, and RLED1 on the board and correspond to the state TGCo is in both LEDs off being the capacity load state, LED RLED0on signifying that the machine is in Mode A, and LED RLED1on signifying that the machine is in Mode B.

IRL\_Done

This is mapped to LED GLED7 on the board and, when on, signifies that the initial capacity loading is done, and TGCo is free to switch between Mode A and Mode B using the Mode\_Select switches.

Capacity:

This is provided by a seven segment decoder and outputs to seven segment display HEX4. This corresponds to the capacity that the currently selected lane is bound to.

Counter 3:

This is provided by a seven segment decoder, the same design used by output Capacity, and outputs to seven segment display HEX3 on the board. This displays the current number of cars in lane 3.

Counter 2:

This is provided by a seven segment decoder, the same design used by output Capacity, and outputs to seven segment display number HEX2 on the board. This displays the current number of cars in lane 2.

Counter 1:

This is provided by a seven segment decoder, the same design used by output Capacity, and outputs to seven segment display number HEX1 on the board. This displays the current number of cars in lane 1.

Counter 0:

This is provided by a seven segment decoder, the same design used by output Capacity, and outputs to seven segment display number HEX0 on the board. This displays the current number of cars in lane 0.

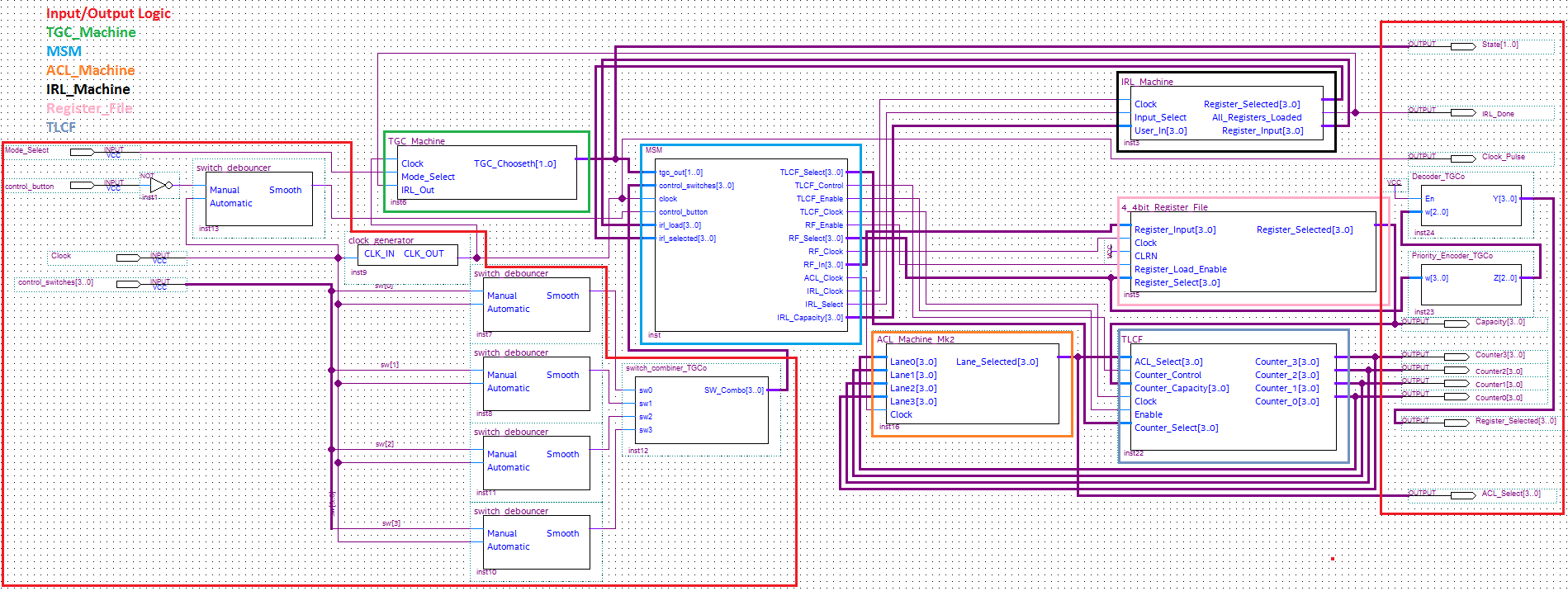
Register\_Selected:

This is mapped to LED’s GLED 0, GLED1, GLED2, and GLED 3 on the board and correspond to the current register and lane currently selected by the input Control\_Switches. All LED’s off being no register or lane selected, LED GLED 0 on being register 0 and lane 0 is selected, LED GLED 1 on being register 1 and lane 1 is selected, LED GLED 2 on being register 2 and lane 2 is selected, and LED GLED 3 on being register 3 and lane 3 is selected.

ACL\_Selection:

This is mapped to LED’s RLED13, RLED14, RLED15, and RLED16 on the board and corresponds to which lane currently has a green light during Mode B, LED RLED13, on being lane 0 has a green light, LED RLED14, on being lane 1 has a green light, LED RLED15, on being lane 2 has a green light, and LED RLED16, on being lane 3 has a green light. The lights are only accurate in Mode B and do not reflect which lane has a green light in Mode A, as no lane has a green light in Mode A.

**Module TGCo:**

**

The\_Grand\_Combo or TGCo, as it is named in the project files and .bsf’s, is where all of the segments of the project get wired together. The inputs from the FPM module correspond directly to these inputs:

FPM Input TGCo Input

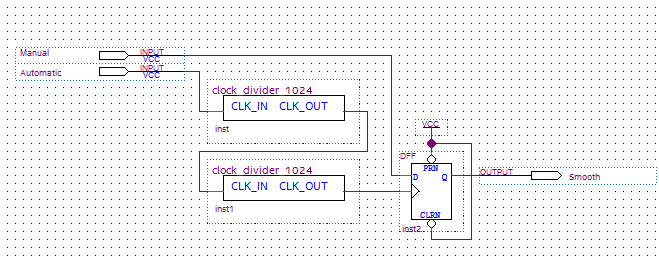
Mode\_Select → Mode\_Select

Control\_Button → control\_button

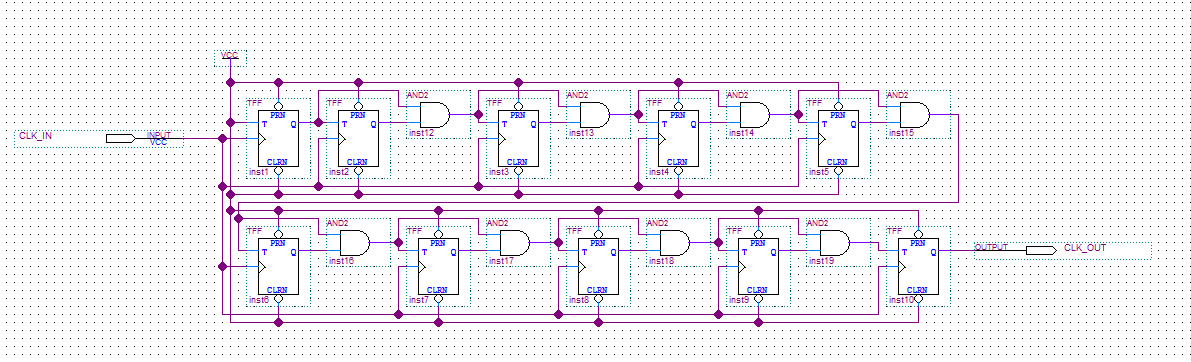
Clock → Clock

Control\_Switches → control\_switches

These inputs also share the same description and functions as those in the FPM module. The inputs control\_switches and control\_button are passed through the switch\_debouncer module made in Lab 11. Here is the diagram for that:

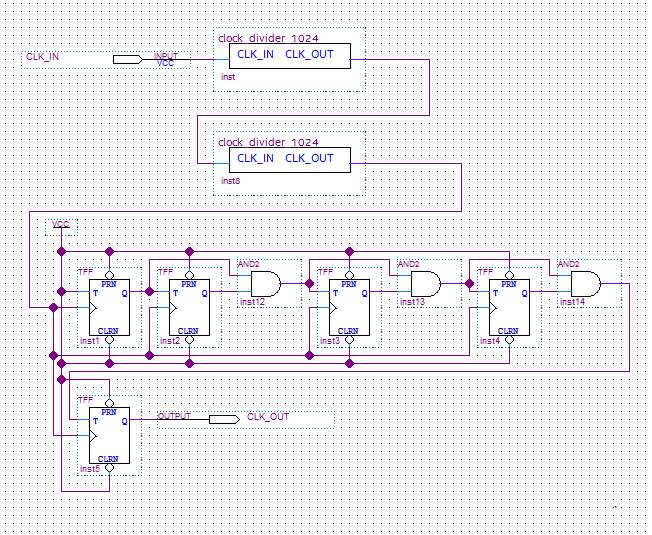
**

The switch debouncer takes in a switch or button input, and a 50MHz clock that is divided by 2048 by the modules clock\_divider\_1024 shown here:

**

This smooths out the input by getting rid of any “bounce” between 1 and 0 that a switch or button may have when set to 1 or 0, ensuring a clear “1” or “0” output.

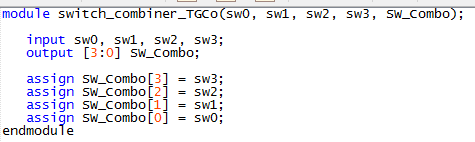
The Clock input is passed through the clock\_generator module, provided to us in Lab 11. Here is the diagram for that:

**

This uses the same clock\_divider\_1024 module as in the switch\_debouncer and some extra T Flip-Flops to slow the clock input from 50MHz to something more manageable like a little over 1 Hz. The T Flip-Flops act like an input divider, dividing the input speed by 2. Stacking these together slows down the input wanted to be slowed down by .

This makes all of the switch flips and button presses in the machine work and make everything readable.

After the input switches and button are debounced and the clock is slowed down to something manageable, the switch inputs are put back into a bus by the module switch\_combiner\_TGCo shown here:

**

This 4-bit bus along with the button and clock, are then fed into the control\_switches input to the module MSM. This module is described later. The Mode\_Select input is fed into the module TGC\_Machine, described later in this report.

The outputs of the TGCo module also correspond directly to the outputs of the FPM module:

FPM Input TGCo Input

State → State

IRL\_Done → IRL\_Done

Clock\_Pulse → Clock\_Pulse

Capacity → Capacity

Counter3 → Counter3

Counter2 → Counter2

Counter1 → Counter1

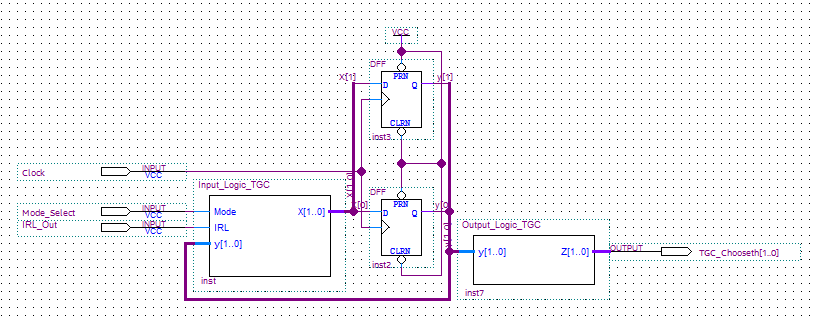
Counter0 → Counter0

Register\_Selected → Register\_Selected

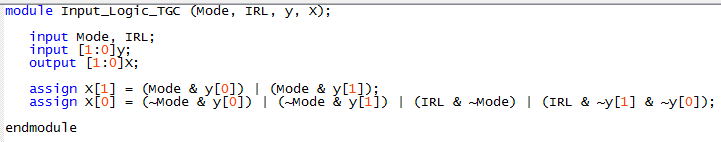
ACL\_Selection → ACL\_Select

These, like the inputs, share the same function as the outputs in module FPM. The only differences being in the Counter outputs. In this module, the counter outputs have not been through a seven\_seg\_decoder module, so they are passed as a 4-bit bus instead of a 7-bit bus.

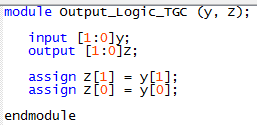
**Module TCG\_Machine:**

**

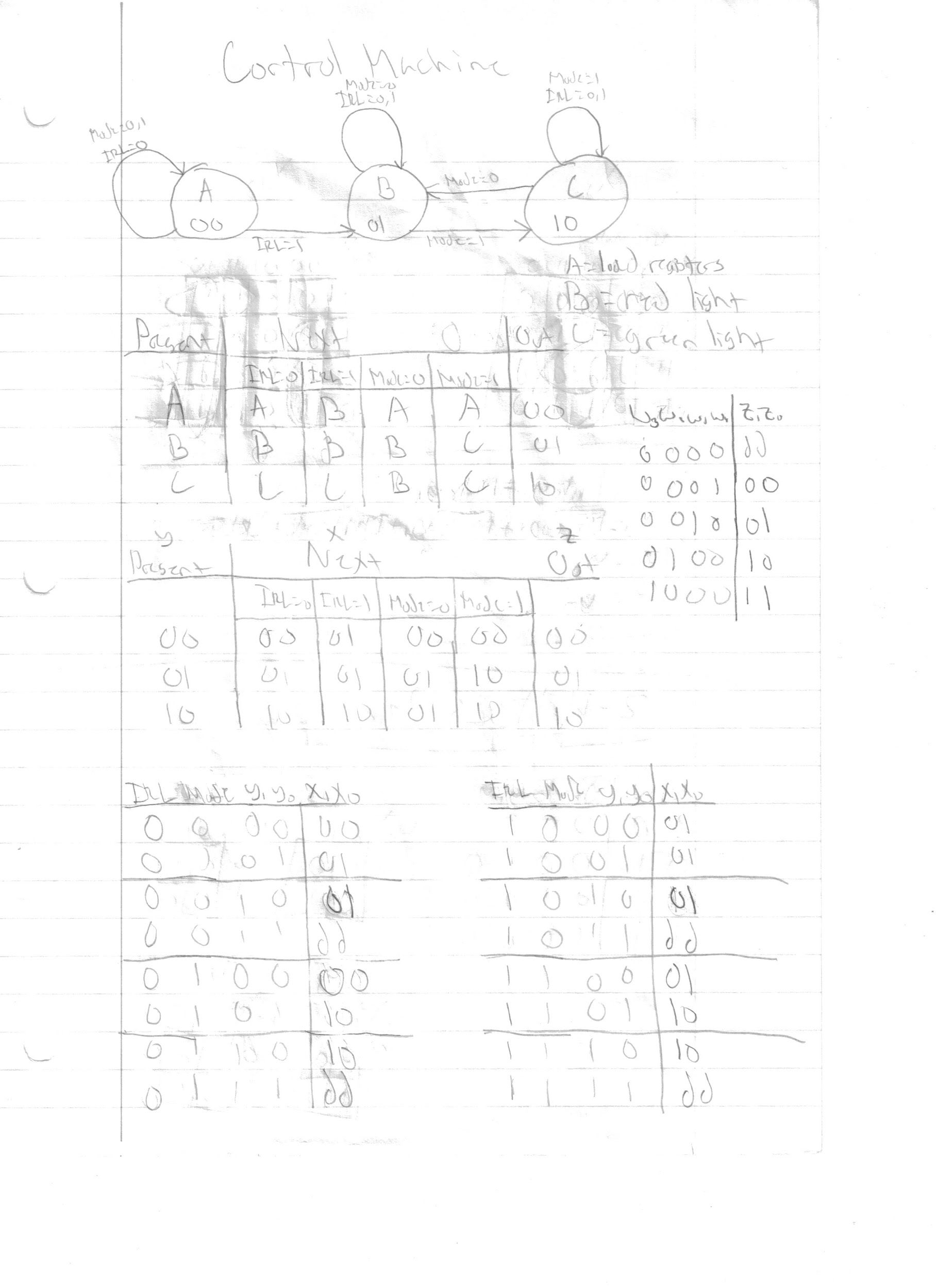
Module The\_Grand\_Control or TGC\_Machine, as it is named in the project files and .bsf’s, is the finite state machine that switches between loading capacities into the registers, and, once that is completed, switches between Mode A and Mode B based on a switch input, named Mode\_Select. The switch input along with an input from module IRL, described later, are put into the block Input\_Logic\_TGC. The Verilog code for this module is shown here:

**

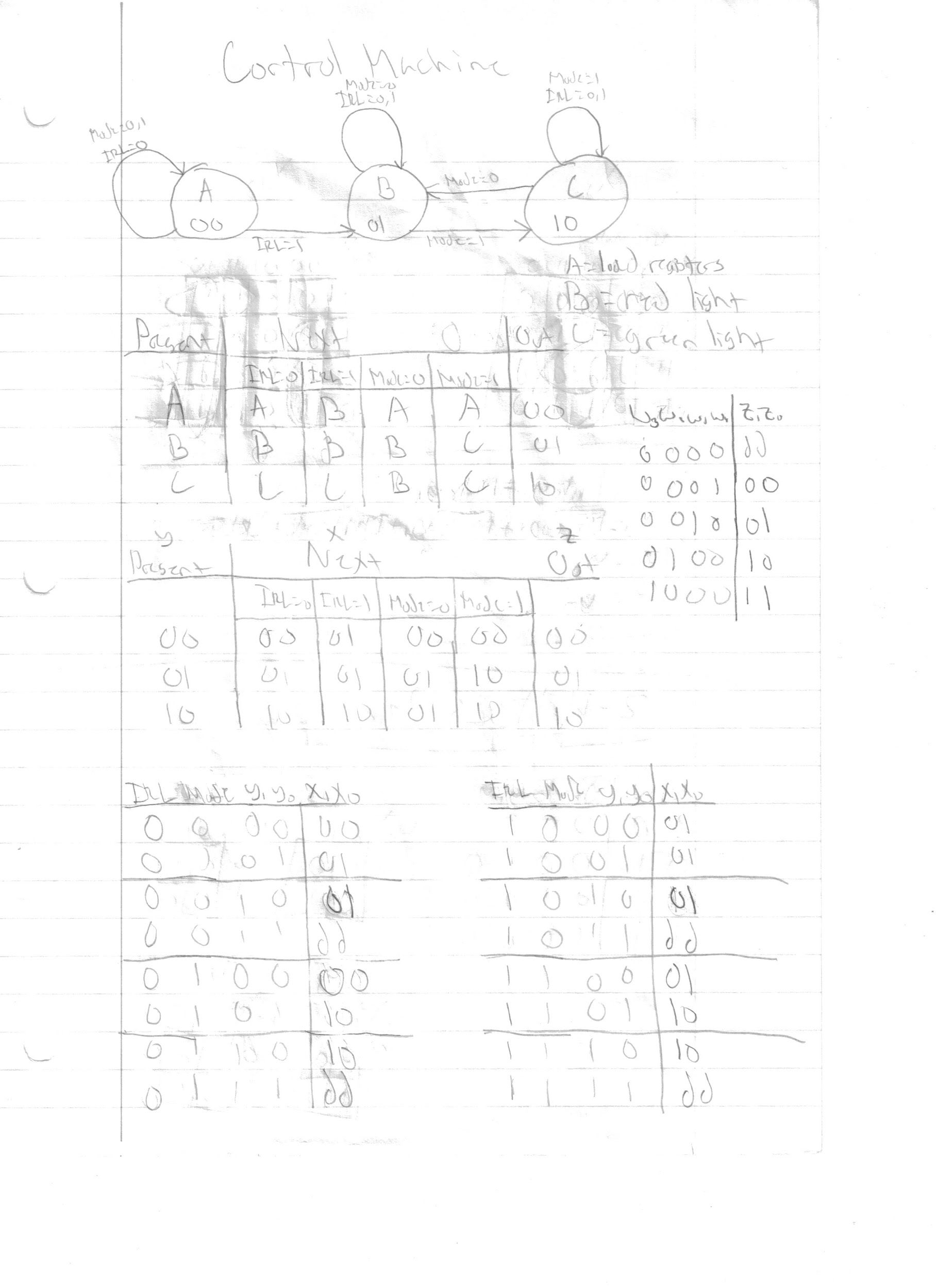
This block outputs a 2-bit bus that corresponds to the next state the machine going to be in, 00, 01, or 10. The block never outputs 11. These states map to which mode the project is in, 00 is the initial capacity loading, 01 is Mode A, and10 is Mode B. This 2-bit number is then distributed into two different D Flip-Flops which update every clock cycle, and then put into the block Output\_Logic\_TGC:

**

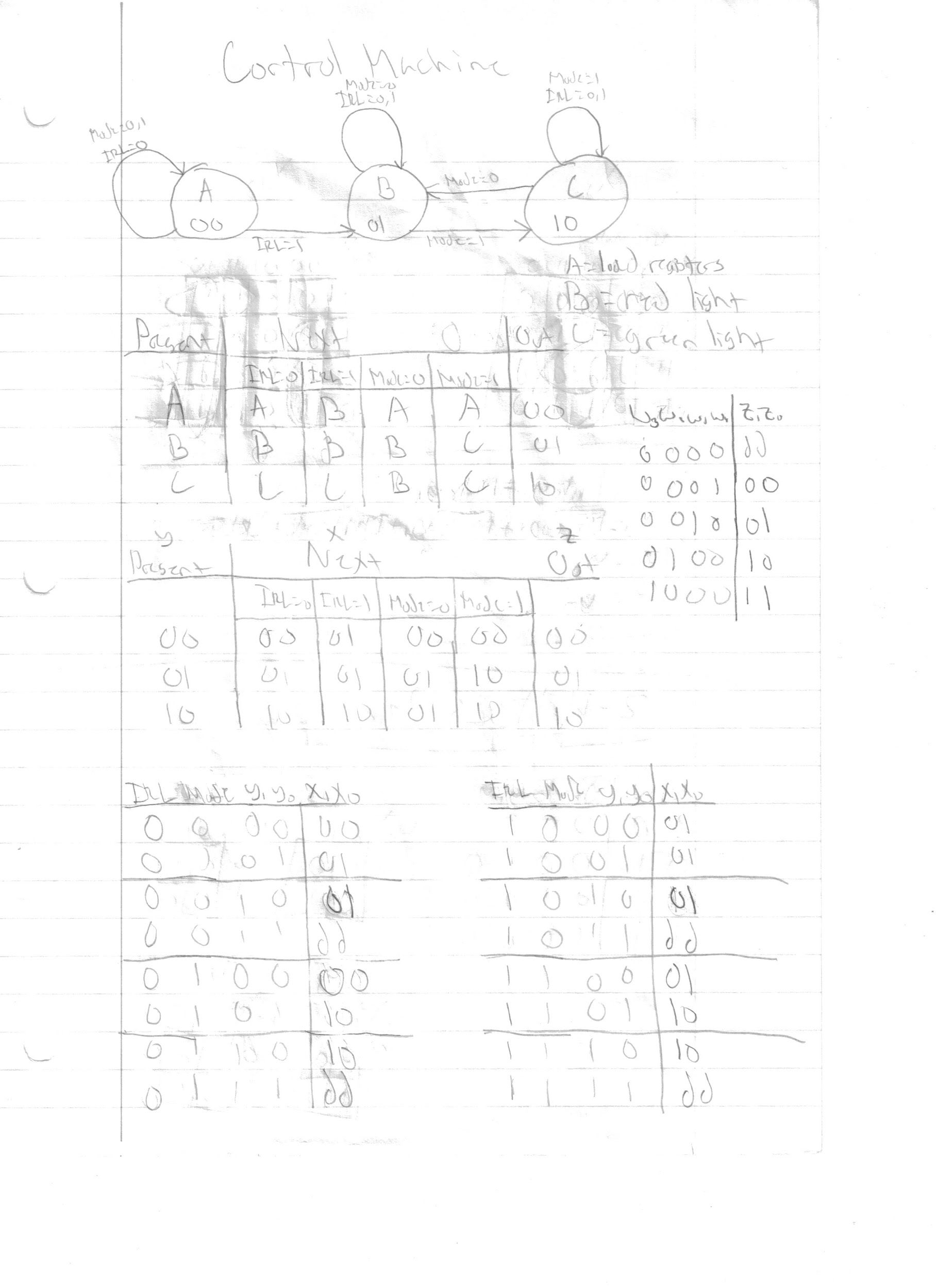
This outputs a 2-bit number which is interpreted as the state TGCo is to be in. This output corresponds directly to what is outputted by the Input\_Logic\_TGC.   
The input and output logic of this module were derived from the following state diagram:

**

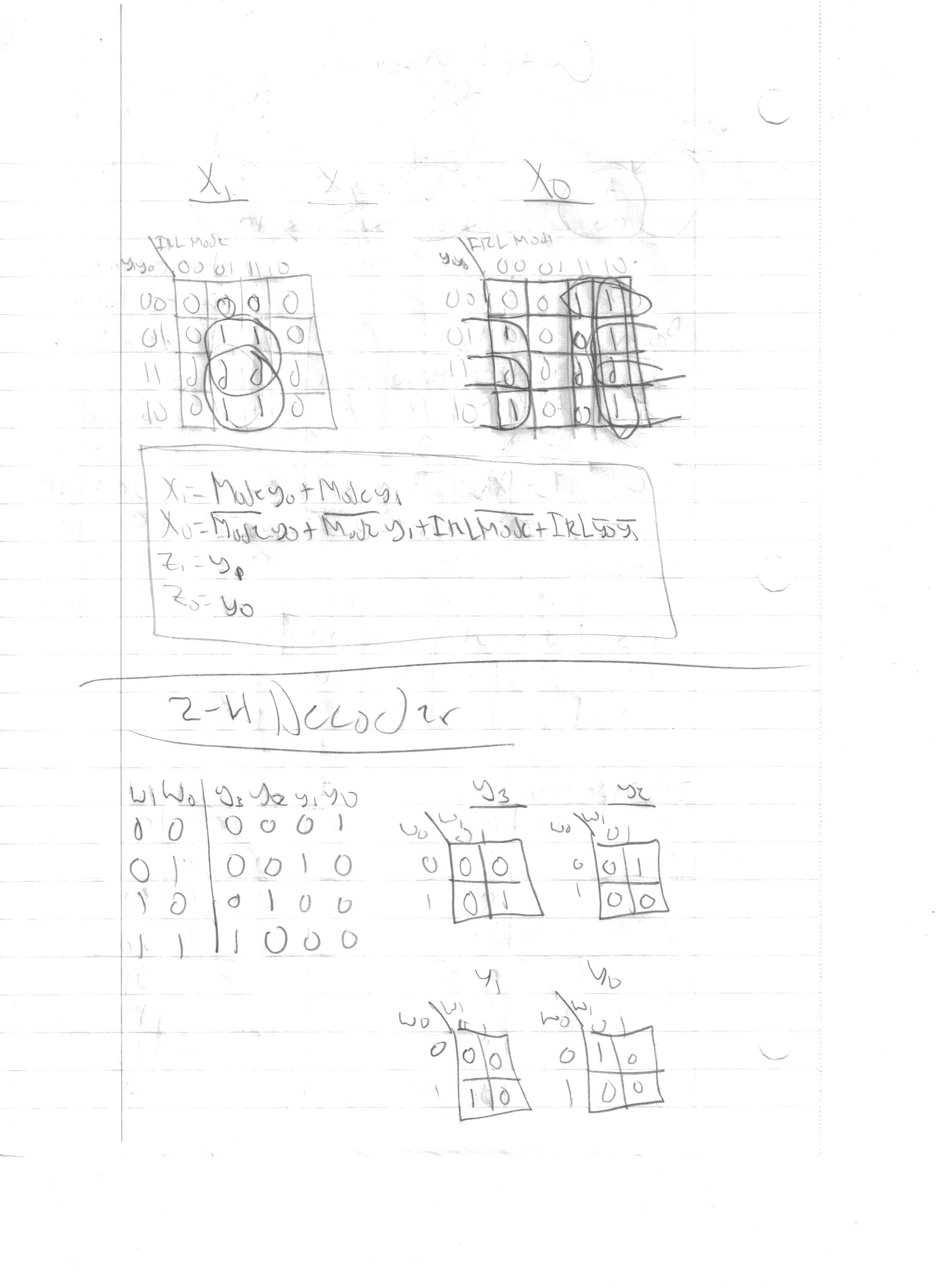
This diagram was then put into a state table

**

which was then put into a truth table

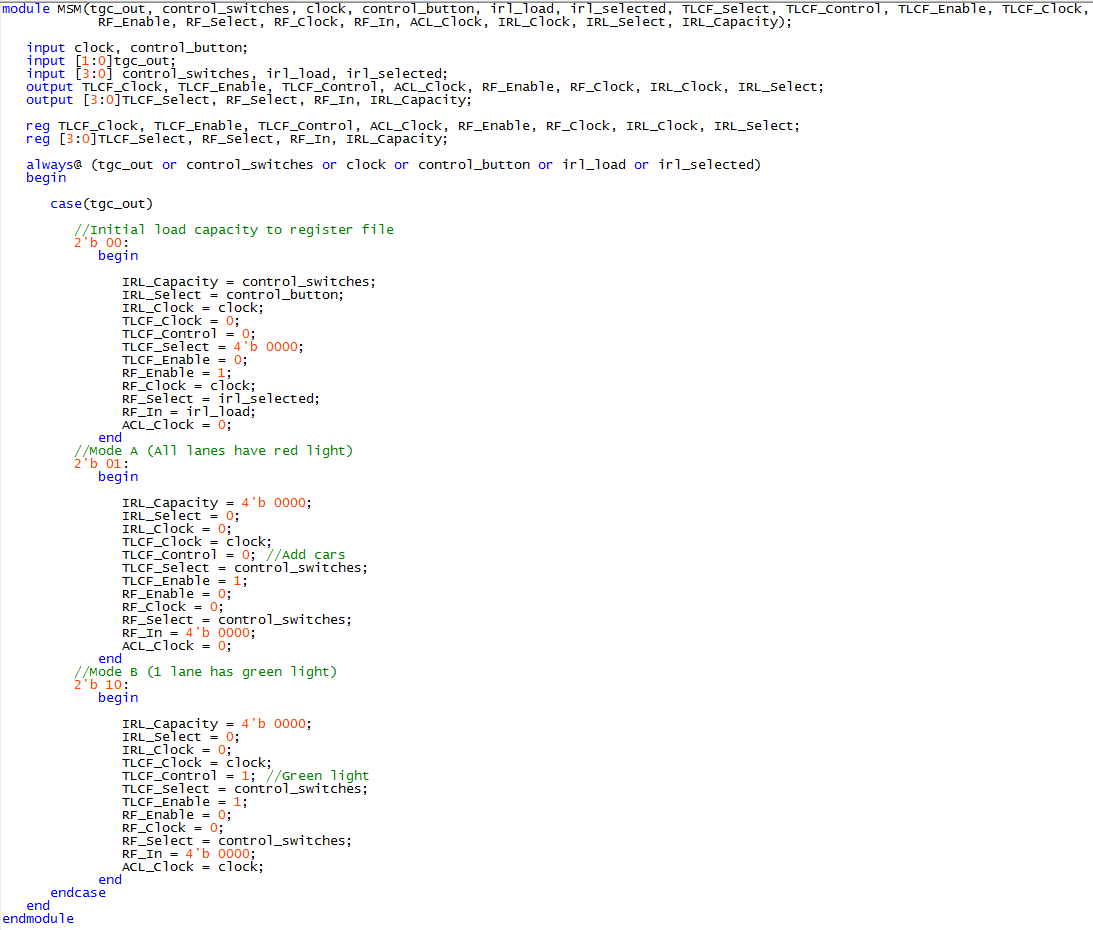
**

To derive the next state logic, K-maps were used for each bit

**

I then had full equations for each next state bit, which was written in Verilog and compiled as the block Input\_Logic\_TGC. The outputs were the same as the current state, so K-maps were not needed to derive equations for each bit. This is outputted as the 2-bit bus TGC\_Chooseth.

**Module MSM:**

**

Module Mode\_Select\_Mux or MSM, as it is named in the project files and .bsf’s, takes in inputs and decides, based on the state chosen by the module TGC\_Machine, where those inputs go to the rest of the module TGCo.

Here is a list of those inputs and outputs:

Inputs:

clock

The board clock, as described in module TGCo.

control\_button

The button on the board, as described in module TGCo.

tgc\_out

The output from module TGC\_Machine. This is the “select line” for this mux.

control\_switches

The switches on the board, as described in module TGCo.

irl\_load

The data from module IRL\_Machine to be loaded in to module 4\_4bit\_Register\_File.

irl\_selected

The register to load data into, provided by IRL\_Machine.

Outputs:

TLCF\_Clock

The clock input that goes to module TLCF.

TLCF\_Enable

Enables the selection of a lane. Goes to module TLCF.

TLCF\_Control

If the project machine is in Mode B, this is set to 1 so a lane can have a green light.

TLCF\_Select

Inputted to TLCF to select a lane to add cars to.

ACL\_Clock

The clock input that goes to module ACL\_Machine\_Mk2

RF\_Enable

Enables the module 4\_4bit\_Register\_File to load data.

RF\_Clock

The clock input that goes to the register file.

RF\_Select

Goes to the register file to select which register to read or write to.

RF\_In

The data to write to a register in the register file.

IRL\_Clock

The clock input that goes to the module IRL\_Machine

IRL\_Select

The input into IRL\_Machine that, when 1, selects the data from the input control\_switches as the capacity for a lane.

IRL\_Capacity

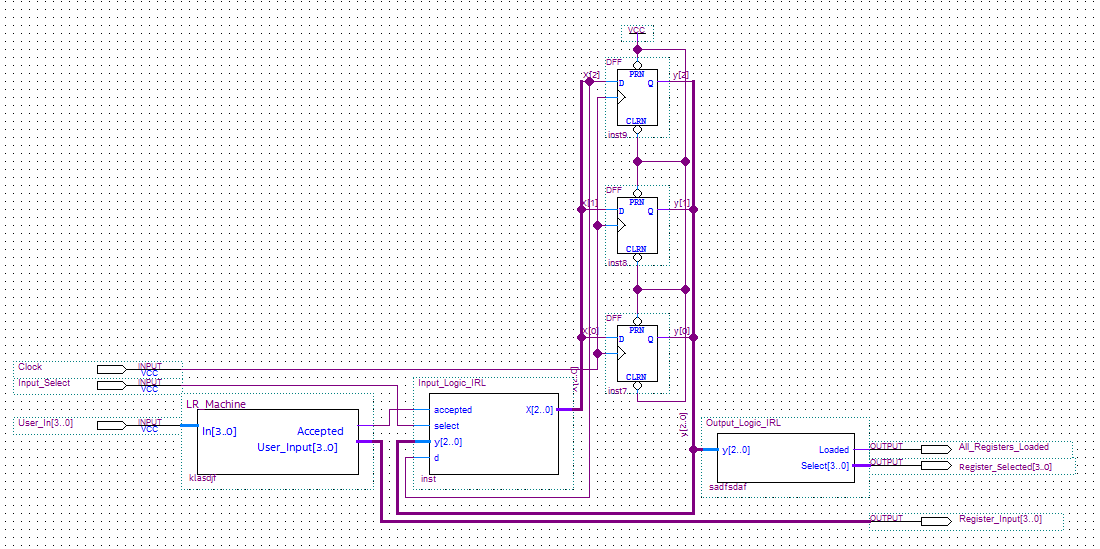
The desired capacity for a lane, provided by the input\_control switches.

If the machine is in the initial capacity loading state, TGC\_Machine is outputting a 00, then the machine is in the initial capacity loading stage. Because of this, IRL\_Capacity is set to what is on control\_switches, IRL\_Select is set to what control\_button is, TLCF\_Clock is set to 0 to ensure that no lane counter can be selected, TLCF\_Control is set to 0 to ensure that no lane has a green light, TLCF\_Select is set to the 4 bit number 0000 to ensure that no lane counter is selected, RF\_Enable is set to 1 because in this stage loading data into a register is desired, RF\_Clock is set to the clock input, RF\_Select is set to what is selected by the IRL\_Machine to load data into a specific register, RF\_In is ultimately set to what the control\_switches are, and ACL\_Clock is set to 0 to ensure that no lane is selected to have a green light.

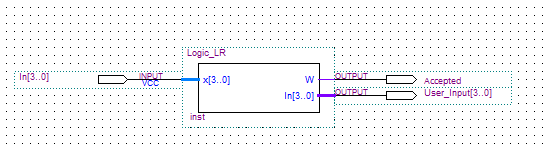
If the machine is in Mode A, TGC\_Machine is outputting a 01, then every lane has a red light no register loading is being done. Because of this, IRL\_Capacity is set to all 0s, IRL\_Select is set to 0 to ensure TGC\_Machine never goes back to the initial capacity loading state, IRL\_Clock is set to 0, TLCF\_Clock is set to the input clock because a lane is to be selected, TLCF\_Select is set to what the input control\_switches is to select a lane, TLCF\_Enable is set to 1, RF\_Enable is set to 0 because no loading is being done to the register file, RF\_Clock is set to 0 for that same reason, RF\_Select is set to what the input control\_switches is to ensure that correct capacity for the selected lane is being read from the register file, RF\_In is set to all 0s, and ACL\_Clock is set to 0 for the same reason as in the last state.

If the machine is n Mode B, TGC\_Machine is outputting a 10, then 1 lane has a green light. Because of this all of the IRL and RF outputs stay the same, TLCF\_Clock is the same, TLCF\_Control is set to 1 because one lane is going to have a green light, TLCF\_Select says on the control\_switches, TLCF\_Enable stays 1, and ACL\_Clock is set to 1 to start counting clock cycles and choose which lane has a green light.

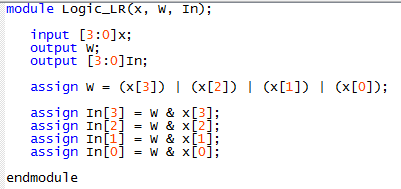
**Module IRL\_Machine:**

**

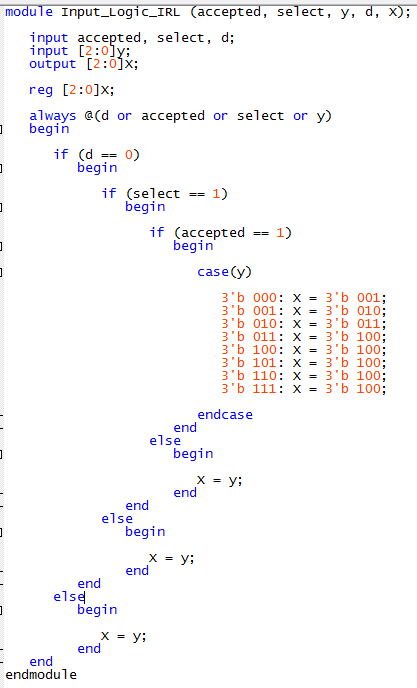
Module Initial\_Register\_Load or IRL\_Machine as it is named in the project files and .bsf’s is used for, as the name suggests, loading the register file with capacities for the lanes during the initial capacity loading stage of the project machine, and then outputs a 1 once it is completed with its task. Loading starts with register 0 and ends with register 3. Per the project requirements, the capacity cannot be 0 and this is checked using the module Load\_Register or LR\_Machine shown here:

**

This module takes in a 4 bit input and then either outputs a 1 if that input is anything other than 0, or a 0 if that input is 0. It also outputs a 4-bit number that is the inputted 4-bit number, just modified with either a 1 or a 0. The Verilog code for this module is shown here:

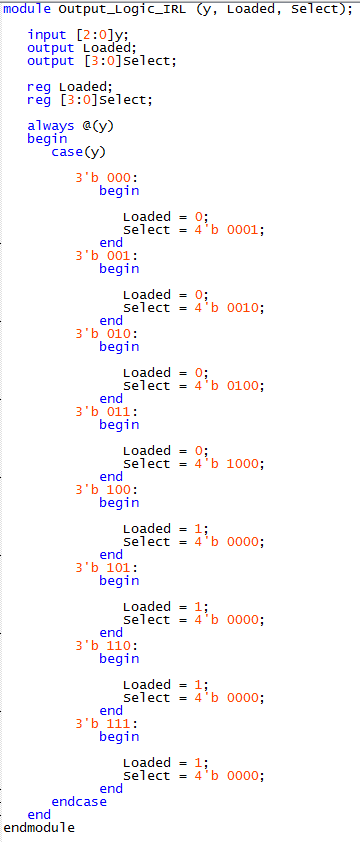
**

This modified 4 bit input and “validness” is then inputted into the block Input\_Logic\_IRL:

**

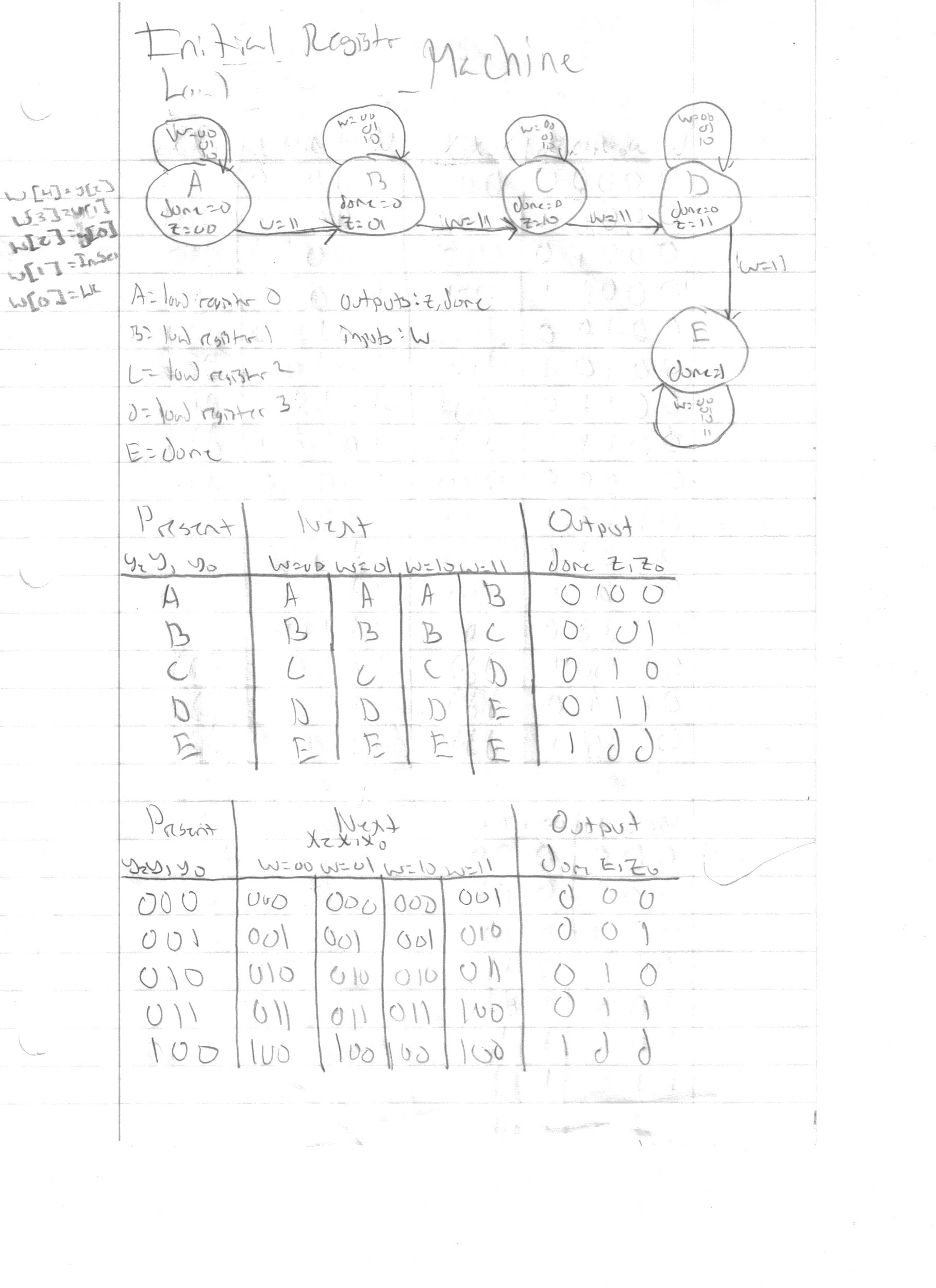
This block outputs a 2-bit number that corresponds to the next state that the module will be in, similar to what was done in TGC\_Machine. The state corresponds to a register to load the validated capacity 000 meaning load into register 0, 001 meaning load into register 1, 010 meaning load into register 2, and finally 011 meaning load into register 3. There is 1 more bit though, one may notice. This last bit, or most significant I should say, represents whether or not the module is done loading or not, 0 if not done, 1 if done. Once this 1 is outputted it will forever stay 1 to ensure that the user cannot go back into loading capacities per the project requirements.

The module then, after updating the D Flip-Flops with the new state outputs a 3-bit number Select which corresponds to the register to load data into, 0001 meaning register 0, 0010 meaning register 1, 0100 meaning register 2, and 1000 meaning register 3. This logic is shown here:

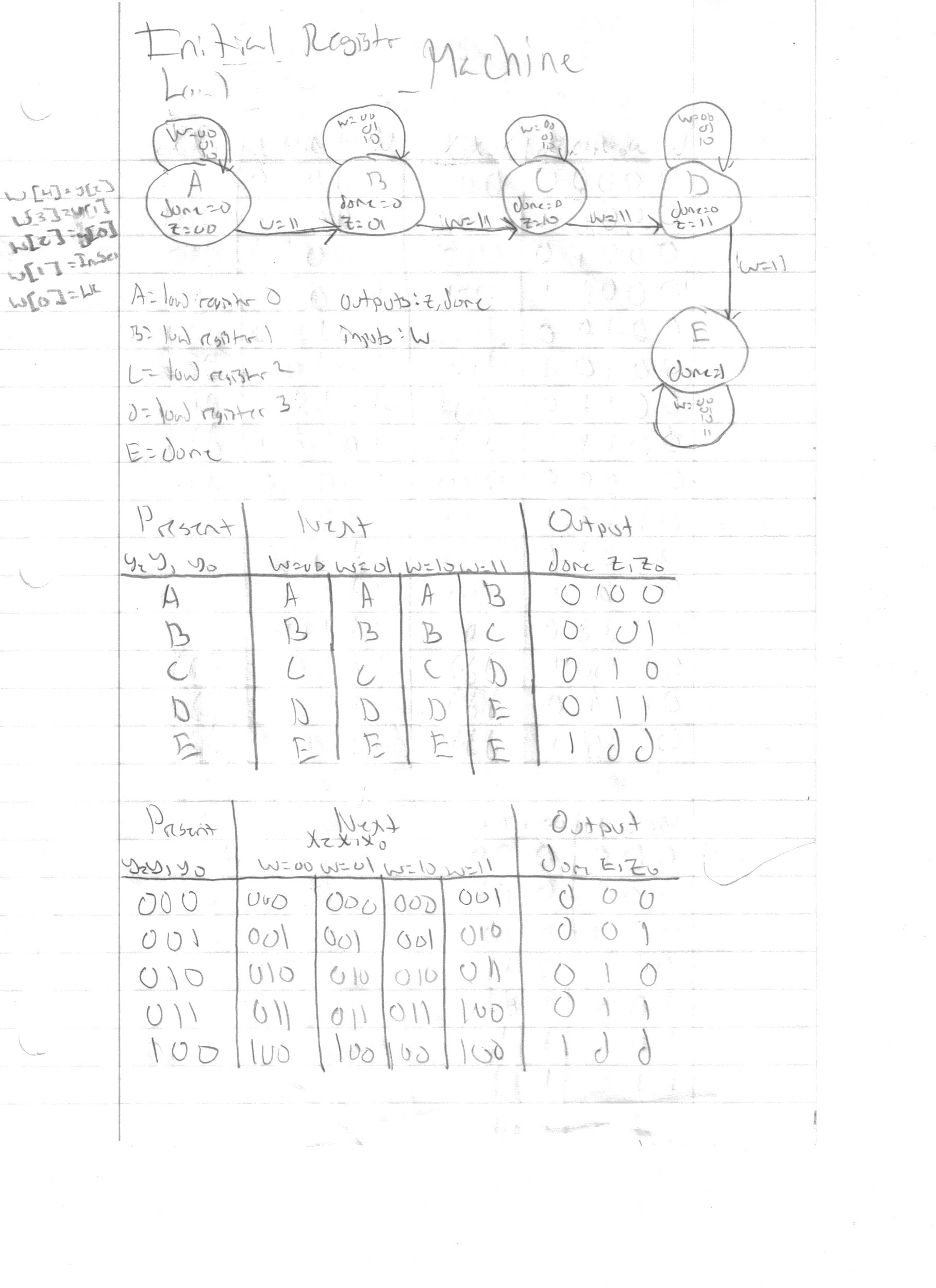
**

Once the most significant bit of the input is a 1, the variable Loaded is set to 1, signifying that the IRL\_Machine module is done.

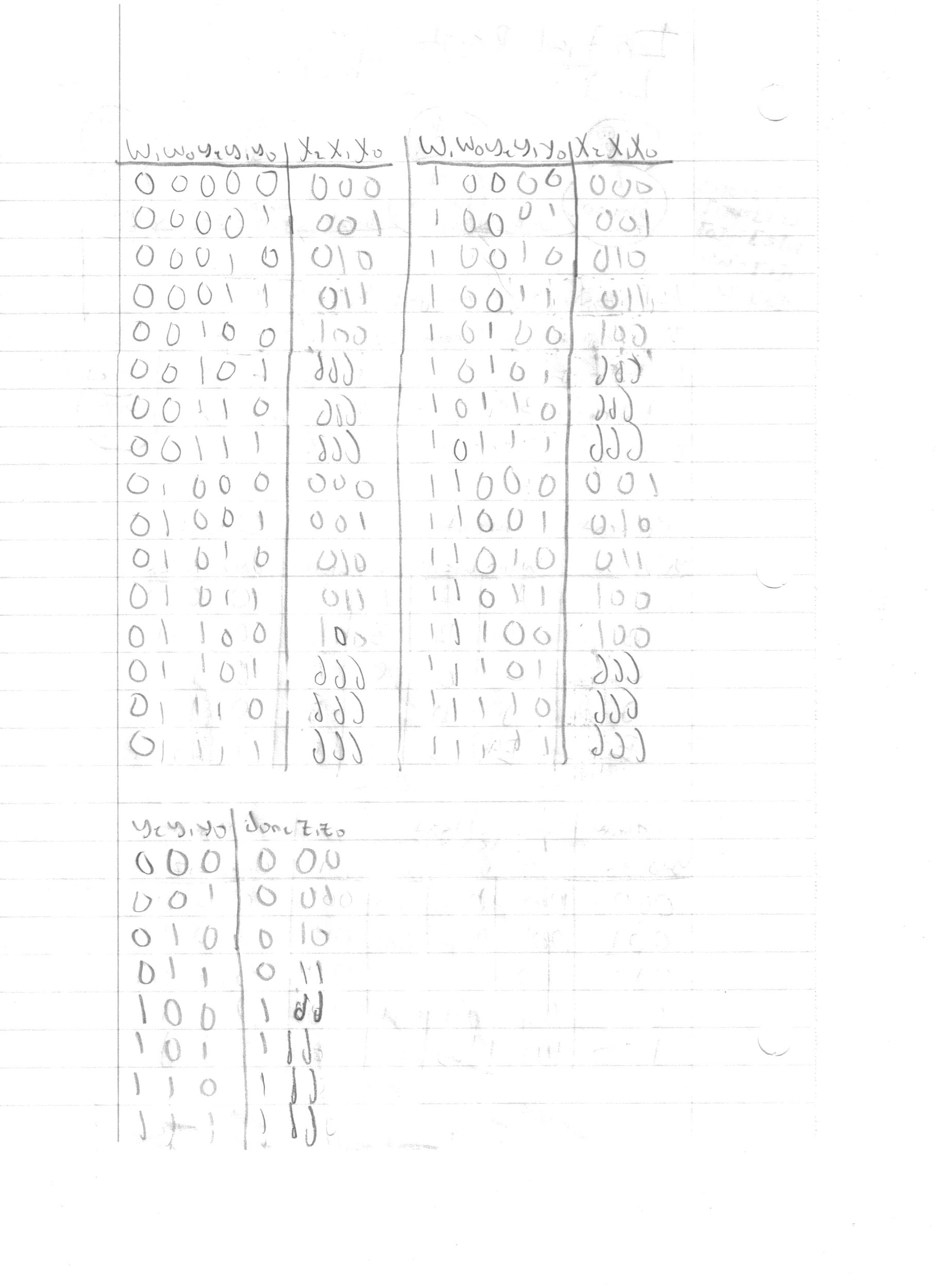
This input and output logic was derived from the following state diagram:

**

This was then put into a state table:

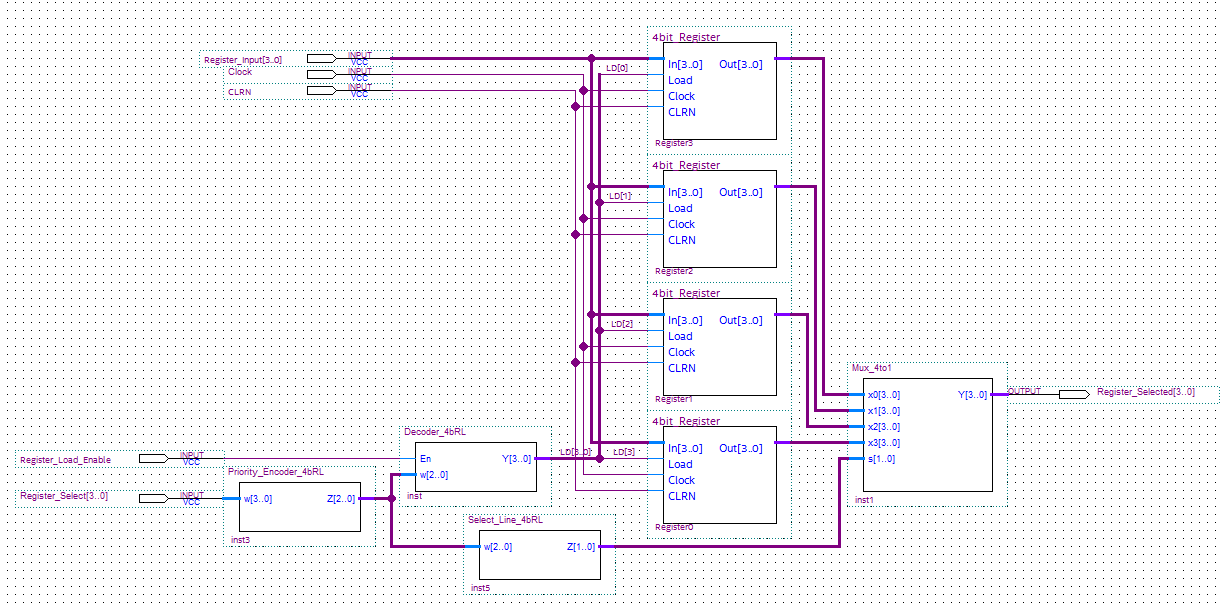
**

Which was translated into a truth table:

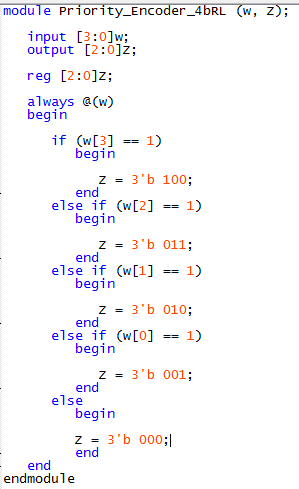
**

Because both the input and output logic were written in Verilog, and the module outputs correspond directly to the current state, no K-maps were used.

**Module 4\_4bit\_Register\_File**

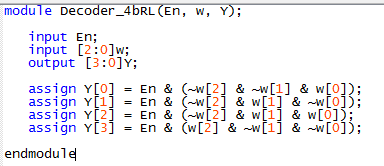
**

This is the register file for this project, as the name suggests. It holds the specific capacities for a lane. As stated in the IRL\_Machine description, it is loaded during the initial capacity loading stage of the project machine, per the requirements and are read from when a lane is chosen. The lane selection and a register selection is one-to-one. For example, if cars are being added to lane 3, then register 3 is read from. Inputs are given by module MSM. The input Register\_Select are put into the priority encoder Priority\_Encoder\_4bRL to endure no two registers are loaded to at one shown here:

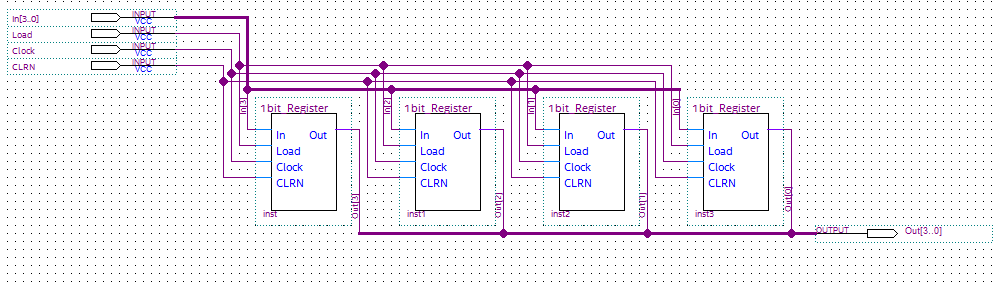
**

Priority is given in increasing register name: register 3 has priority over all other registers, register 2 has priority over registers 1 and 0, register 1 has priority over register 0 and register 0 has priority over no other registers. There is also the case where if the input line contains all 0s, all the switches are 0, then no register is to be chosen. This is taken advantage of in the module TCLF for user convenience.

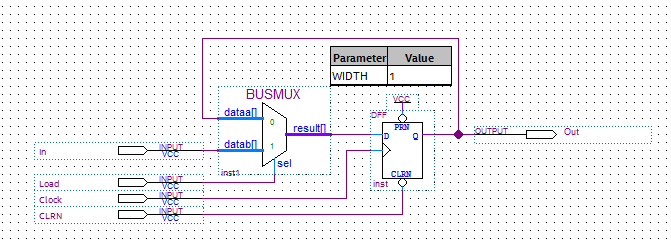
The output of the priority encoder is put into two different blocks. The first being a decoder shown here:

**

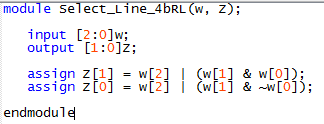
This takes advantage of the one hot encoded nature of a decoder and selects which register to enable, loading whatever data is supplied through the input Register\_Input into that register on the next clock cycle. The register being loaded is shown here:

**

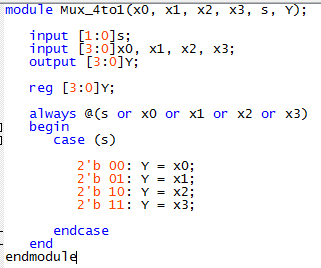
This is a parallel load register similar to ones found in the class presentation slides. Each of these 4 bit registers are comprised of 4 1 bit registers shown here:

**

Back to the 4\_4bit\_Register\_File, the second block the priority encoded line goes to is Select\_Line\_4bRL:

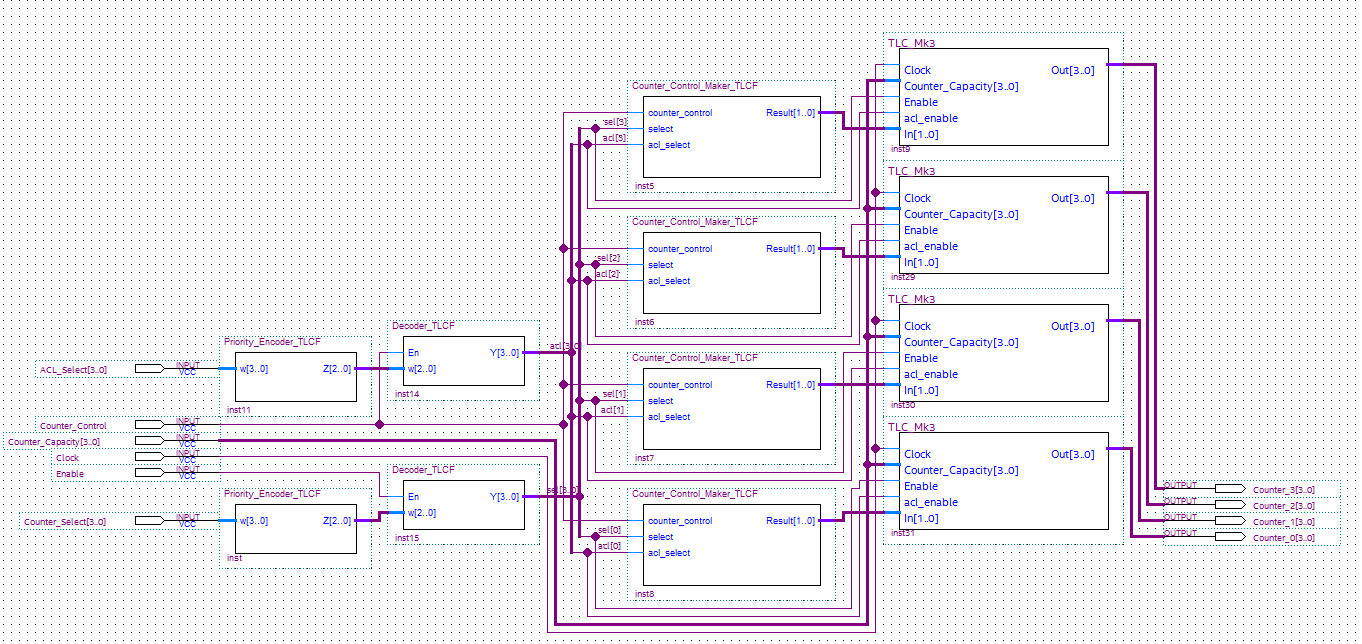
**

This block takes in the priority encoded output and turns that into a 2 bit select line for the 4\_4bit\_Register\_File 4-1 mux shown here:

**

This takes in all of the registers contents, and then outputs 1 of them based on a select line. This ensures that whatever register is being written to, if in the loading stage, the contents of that register are being read.

**Module TLCF and TLC\_Mk3**

**

This module acts similar to module 4\_4bit\_Register\_File, except with counters instead of registers. This takes in numerous inputs:

ACL\_Select:

This is a 4 bit input which is determined by the module ACL\_Mk2\_Machine, described later. This goes through a priority encoder, identical in spec to the one used in the module 4\_4bit\_Register\_File and then through a decoder, again identical to the one used in the register file module, to select a counter to decrement, or have a green light.

Counter\_Control:

This input is 1 if the project machine is in Mode B and 0 if not. This is the enable line for the decoder that selects a line to have a green light.

Counter\_Capacity:

This is a 4 bit input that comes from the module 4\_4bit\_Register\_File. This is the capacity that a counter, when selected cannot increment over.

Clock:

This is the clock for the counters.

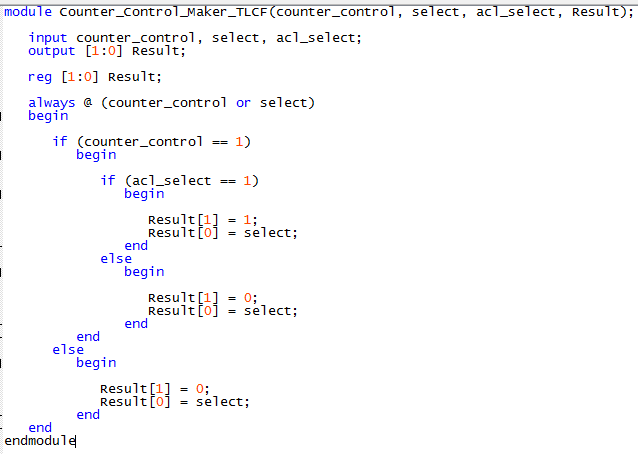
Enable:

This input enables this module to work and select a counter to me modified.

Counter\_Select:

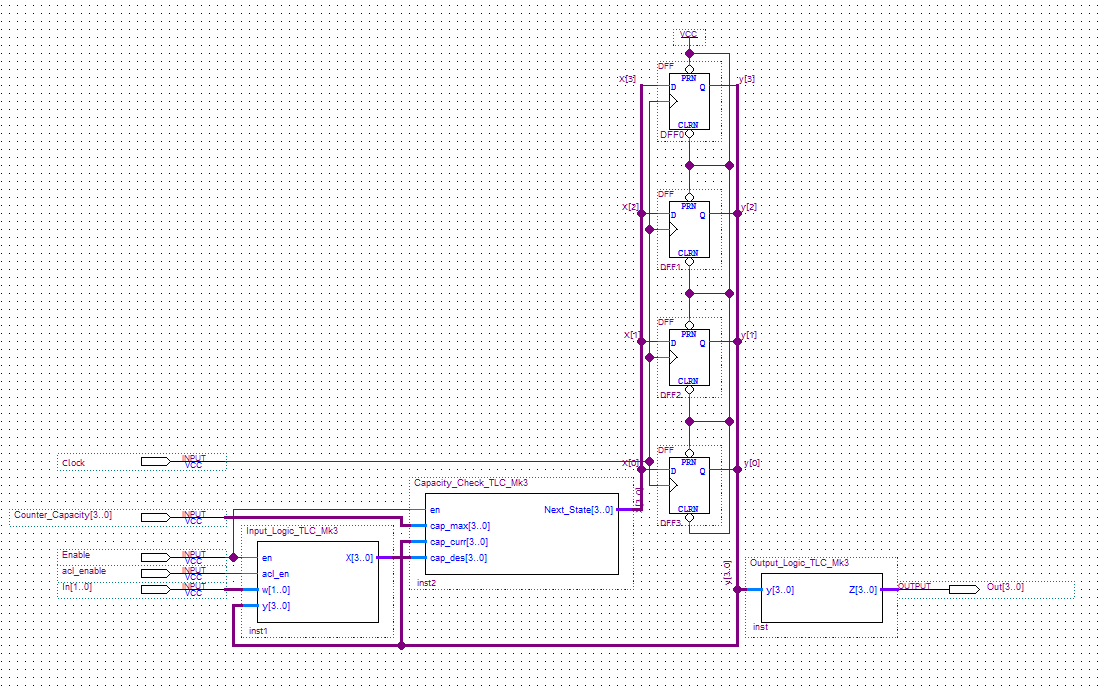
This is a 4 bit input that goes through an identical priority encoder and decoder used for the input ACL\_Select. This selects a counter to be modified.

While in Mode A or Mode B, a counter is selected by the input Counter\_Select to either increment in Mode A’s case, or decrement or increment in Mode B’s case. This selection, after choosing priority and decoding it to be one-hot encoded, is split into 4 different data lines and enters the block Counter\_Control\_Maker\_TLCF shown here:

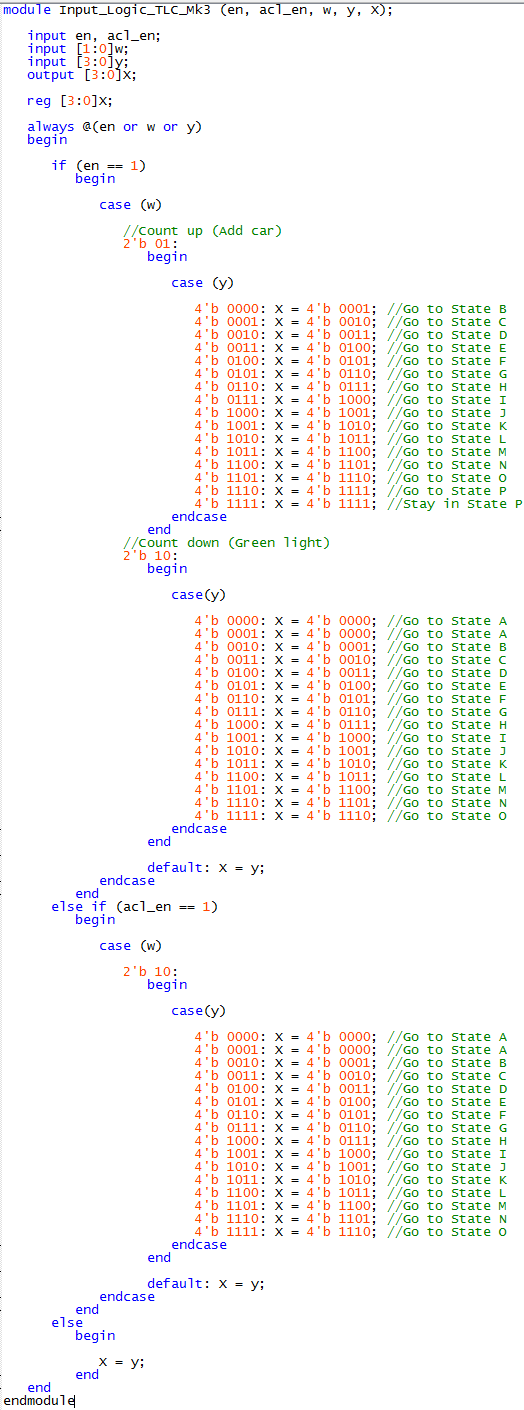
**

This block, written in Verilog, outputs a 2-bit bus where the most significant bit is 1 if the counter is to decrement, the lane has a green light, and the least significant bit is 1 if the counter is to increment, the lane gets another car. If the counter is supposed to increment and decrement at the same time, the counter stays at the same value it is at per the specifications, this block outputs a 0 and whatever the select line is. This select line comes from the higher level input Counter\_Select, which determines if the counter should increment or not.

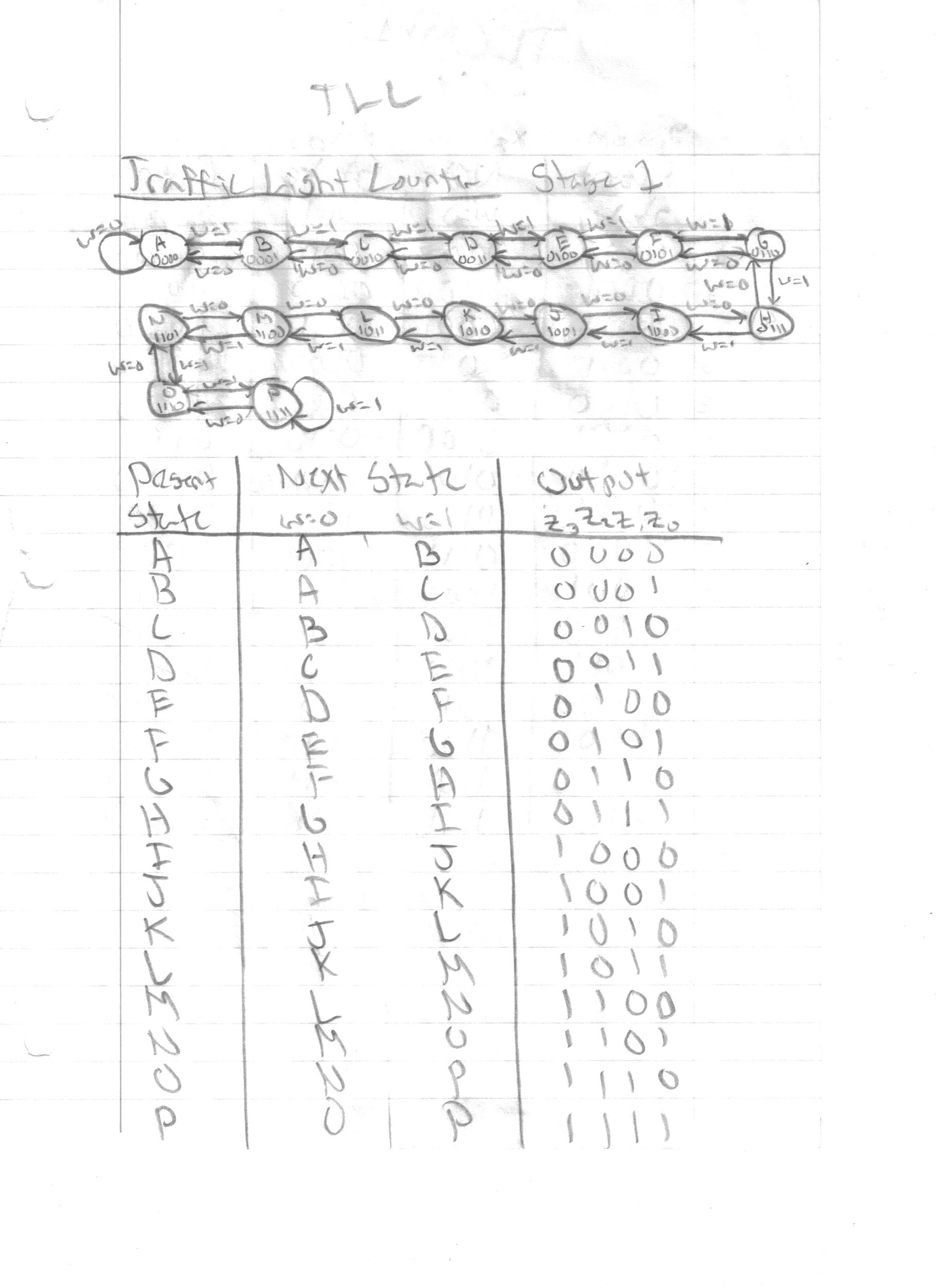
The output of the block Counter\_Control\_Maker goes into the actual counter itself along with a multitude of other variables. The specialized counter module TCL\_Mk3 is shown here:

**

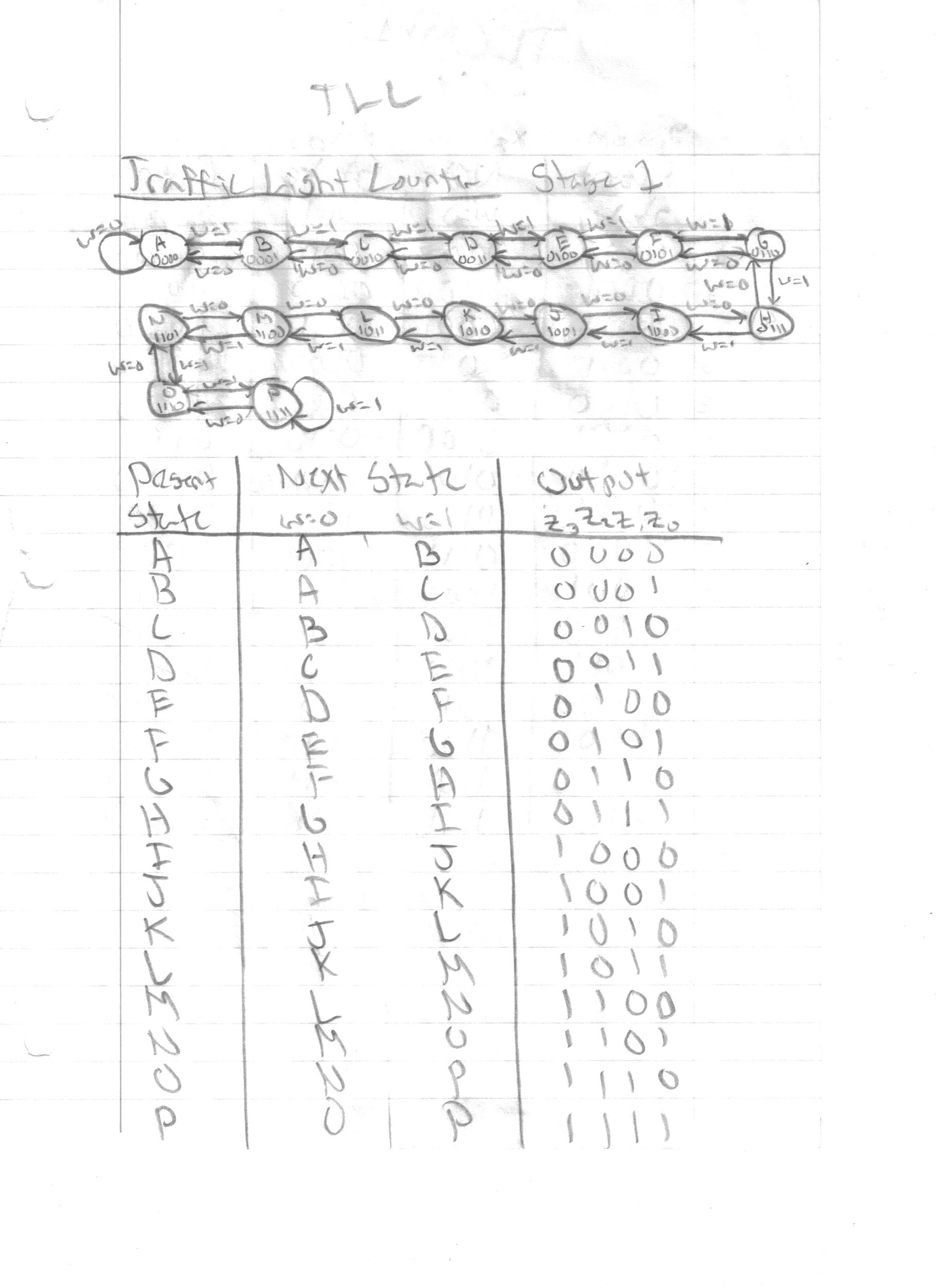
Traffic\_Light\_Counter\_Mk3 or TLC\_Mk3, as it is named in the project files and .bsf’s, is the specialized counter for this project. It takes in a clock input, a 4 bit input Counter\_Capacity which is the limit to what the counter can increment to, an enable line Enable which enables this counter to increment or decrement or not, an input named acl\_enable which determines if this counter is selected to decrement or not and a 2 bit input In which controls if the counter is allowed to increment or decrement or not. The inputs Enable, acl\_enable, and In are passed through the block Input\_Logic\_TLC\_Mk3 shown here:

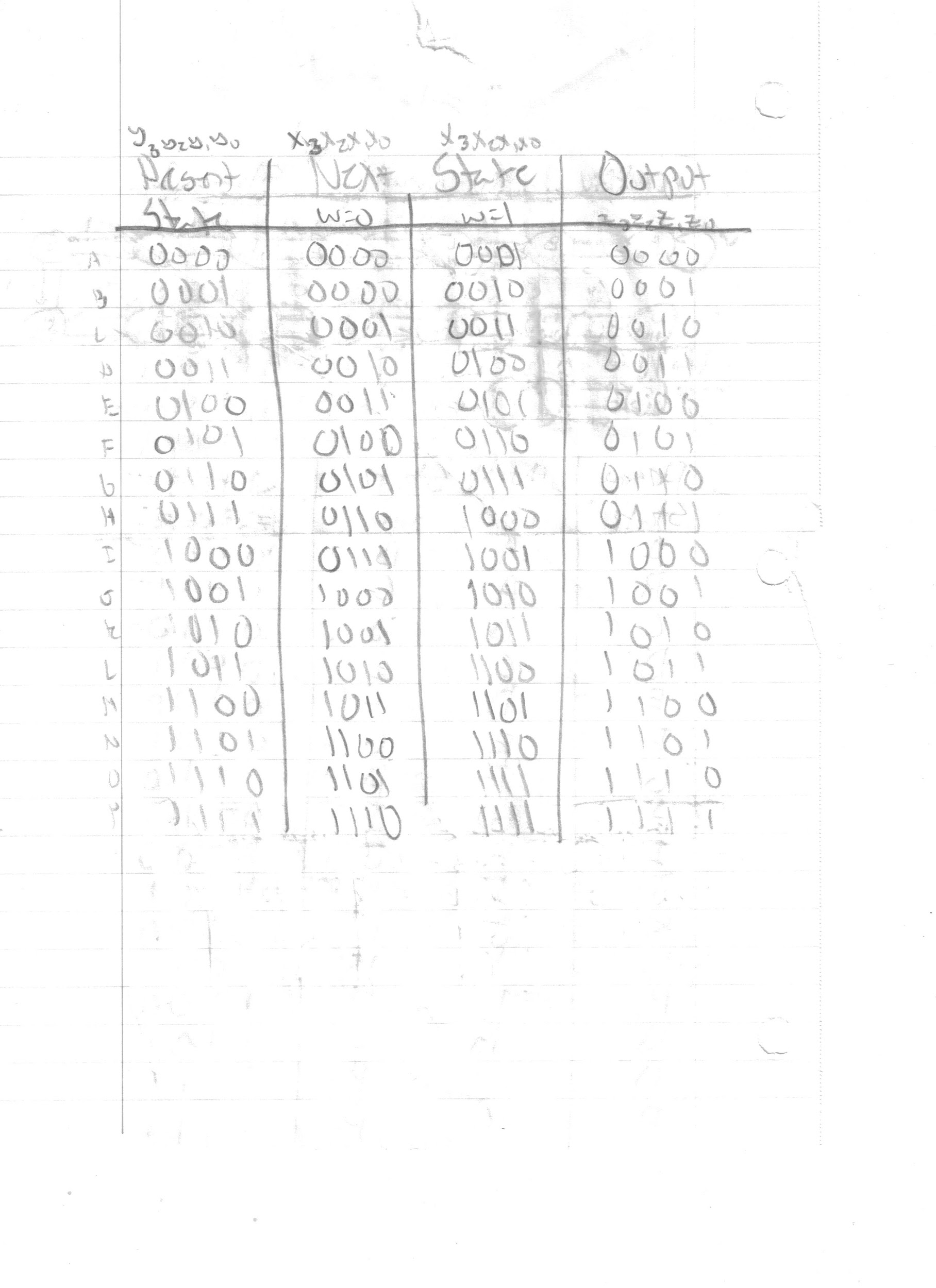
**

This determines if the counter is supposed to change states or not. Using Verilog, if the counter is enabled, input en is 1, the counter does whatever is on the control input w. If w is a 01, then the counter increments states, if w is a 10, the counter decrements states, and if w is neither, then the counter stays in the same state. If the counter is not enabled, but is supposed to decrement states, based on the input acl\_en, then the counter decrements. In other words, input en enables the counter to increment, and acl\_en enables the counter to decrement. If neither of these inputs are 1, the counter stays in the same state. All of this was derived from the following state diagram, and then modified to as needed:

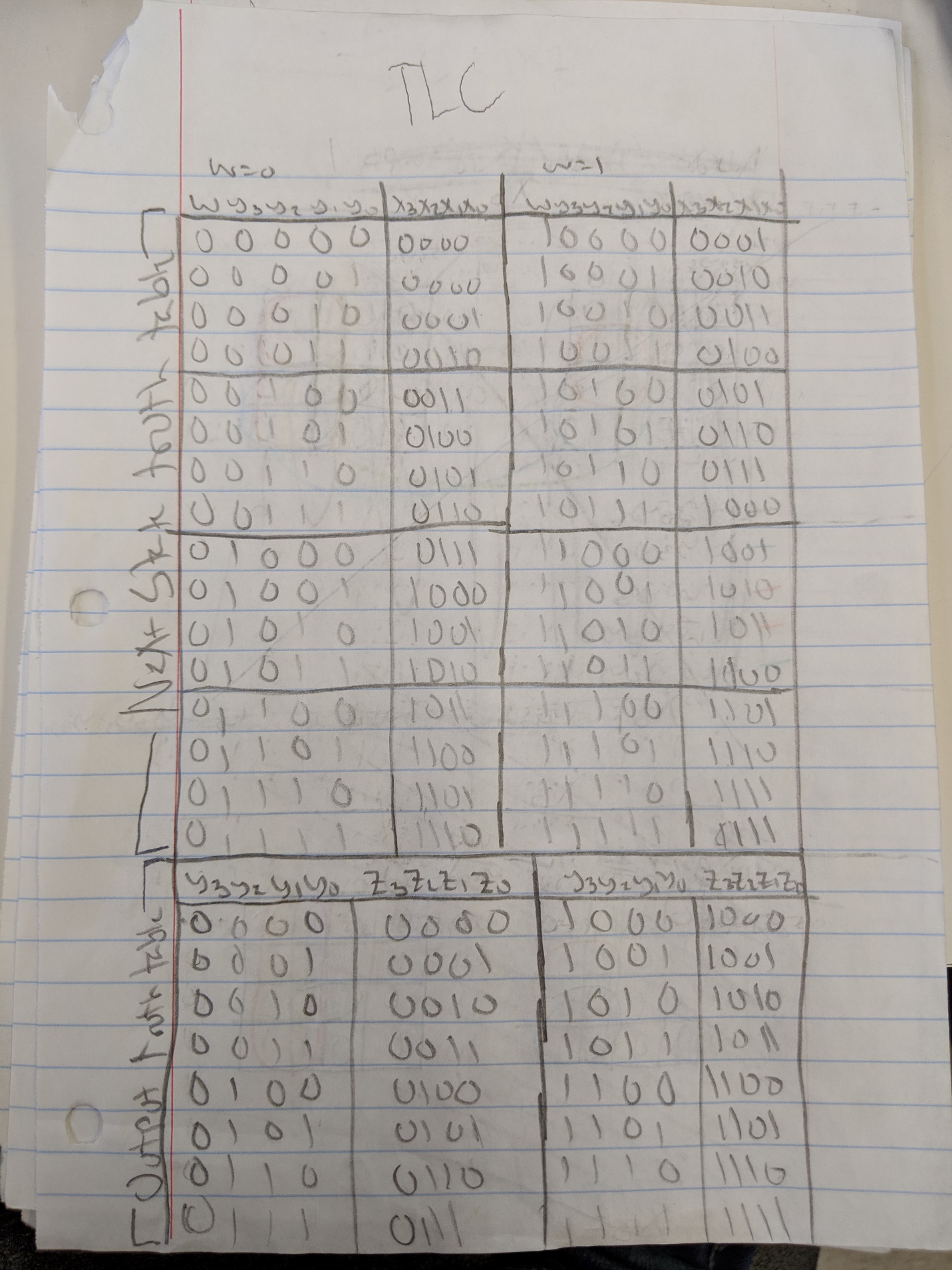
**

This was then put into a state table:

**

**

and then put into a truth table:



Because I used truth table coding in Verilog, I used no K-maps to derive equations for variables.

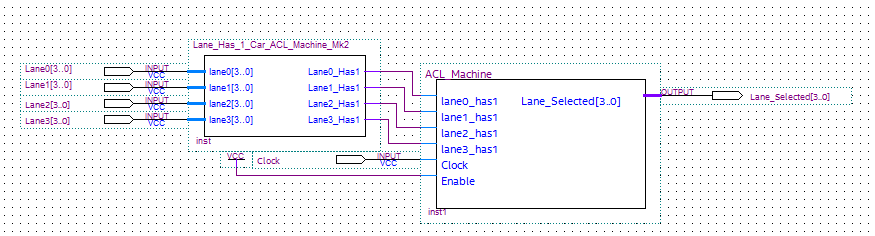
The output of the module Input\_Logic\_TLC\_Mk3 is then passed through another block Capacicity\_Check\_TLC\_Mk3 shown here

**

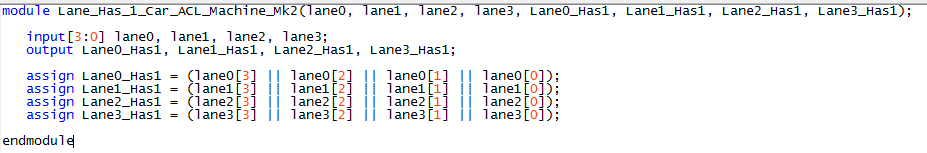
This, as the name suggests, checks whether or not the state outputted by Input\_Logic\_TLC\_Mk3 is over the capacity the counter is bound to. If the next state is less than or equal to the capacity, then the counter increments to that state. If it is greater than the capacity, then the counter stays in the current state.

This is then fed to D Flip-Flops which ultimately go to the block Output\_Logic\_TLC\_Mk3 which outputs a 4-bit bus that is one to one to the counter’s state.

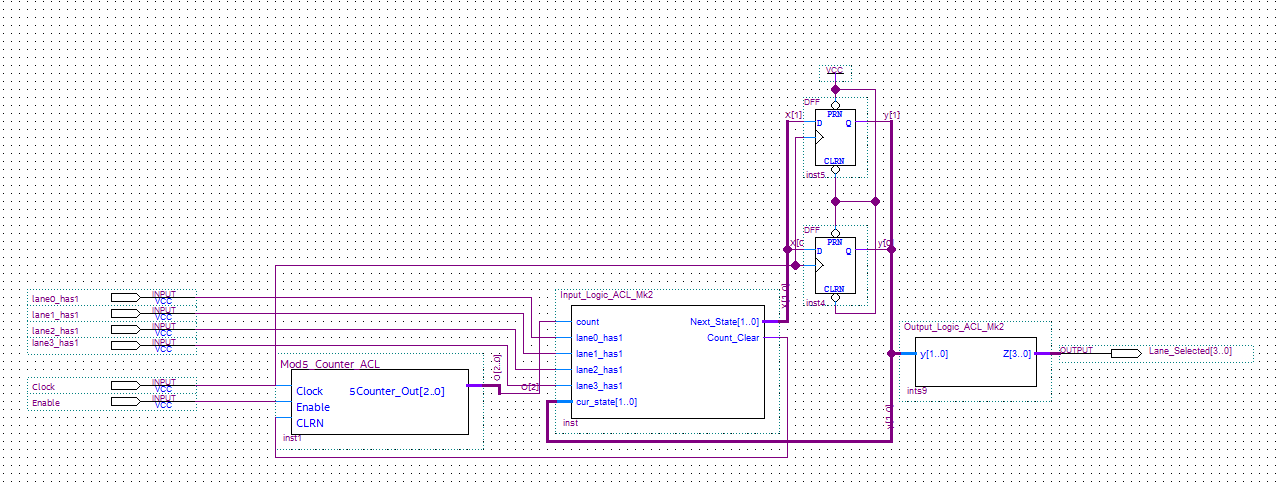
**Module ACL\_Machine\_Mk2**

**

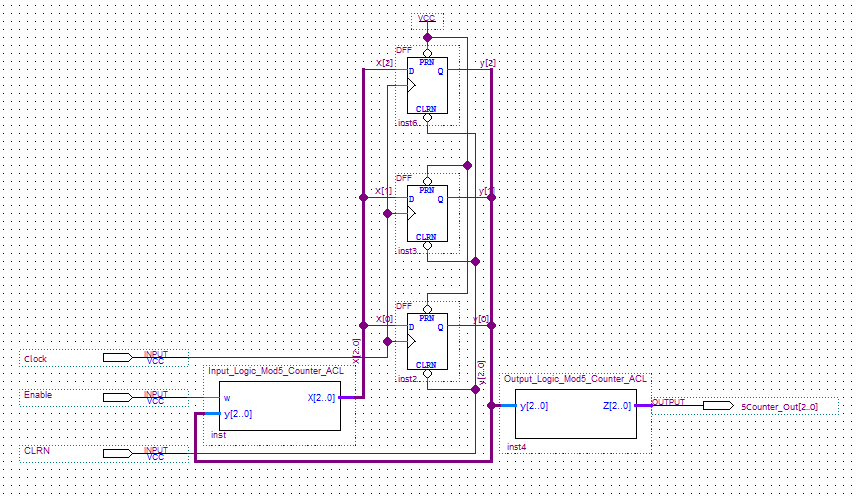
Module Auto\_Change\_Lanes\_Mk2 or ACL\_Mk2, as it is named in the project files and .bsf’s is the state machine that selects a counter to decrement or not. It first determines if a lane has a car in it using the block Lane\_Has\_1\_Car\_ACL\_Machine\_Mk2 shown here:

**

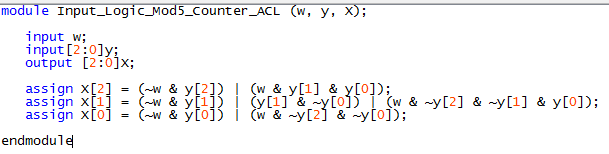
It takes inputs that are the current counter capacities from module TLCF and then outputs a 1 corresponding to each lane if that lane has at least 1 car in it. This is then outputted to the machine ACL\_Machine:

**

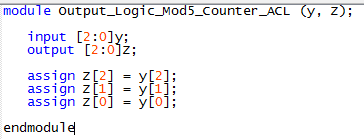
This finite state machine takes in the outputs from the block Lane\_Has\_1\_Car\_ACL\_Machine\_Mk2, and then determines which lane is supposed to have a green light. Per the project specifications, the green light moves to a different lane that has cars in it after 5 clock cycles. The clock cycles are counted by the module Mod5\_Counter\_ACL shown here:

**

This counts to 5, incrementing every clock cycle, and then resetting only if told to, or if it reaches 5. This is similar to counters derived in class presentation slides. The next state is determined by the block Input\_Logic\_Mod5\_Counter\_ACL shown here:

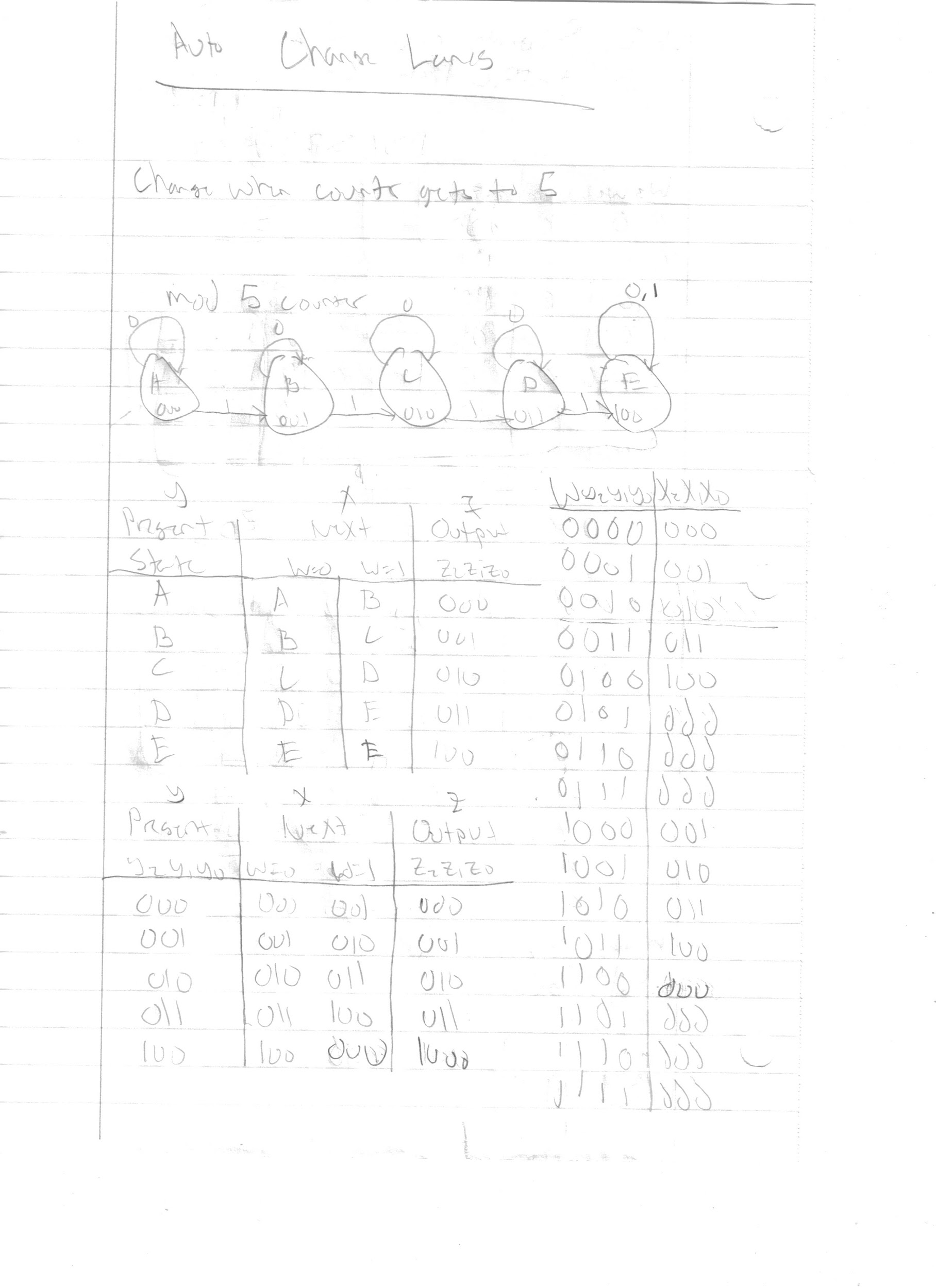
**

This new state is then fed into d Flip-Flops and ultimately outputted by the block Output\_Logic\_Mod5\_Counter\_ACL:

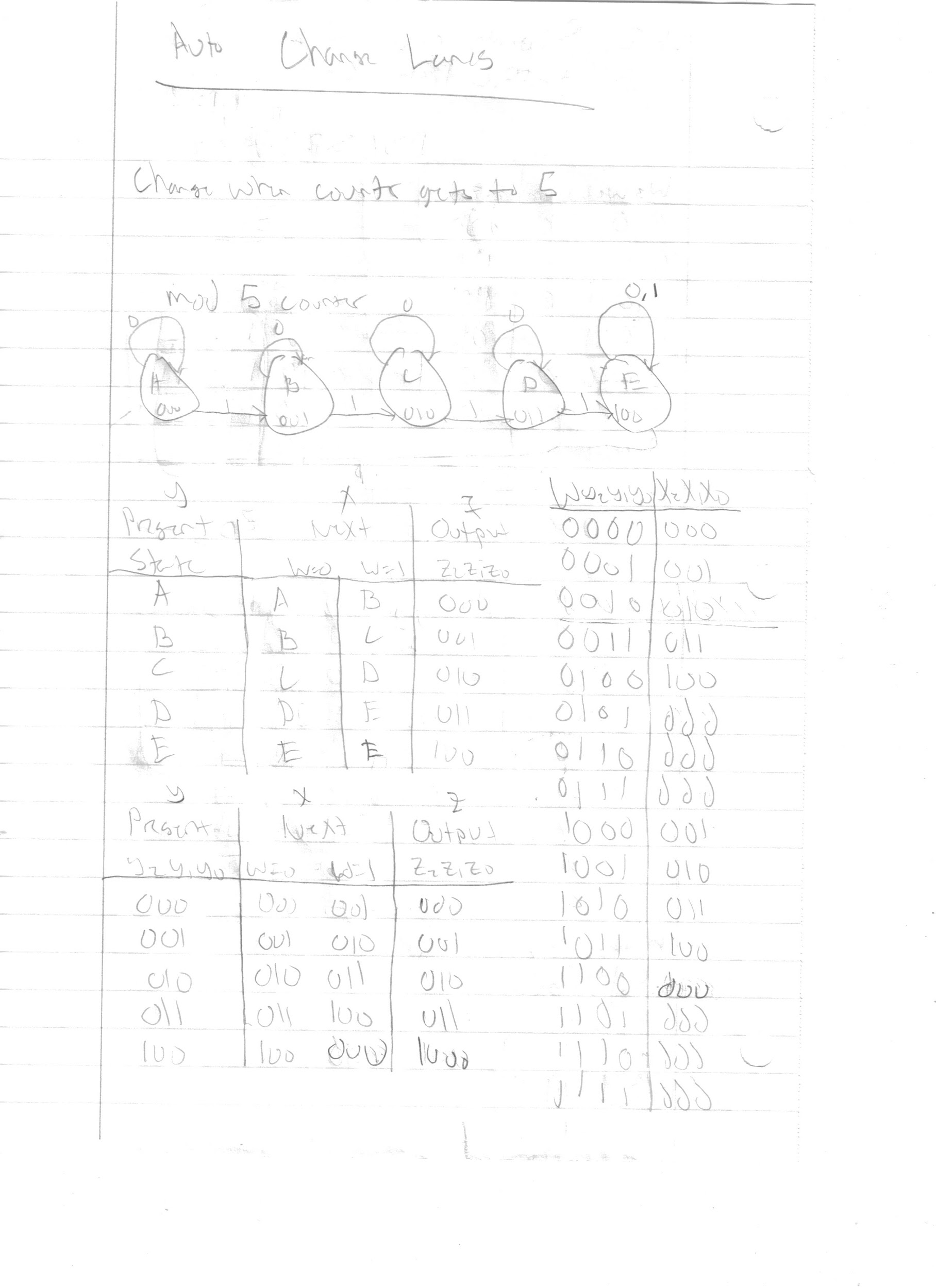
**

The outputs are one to one with the current state.

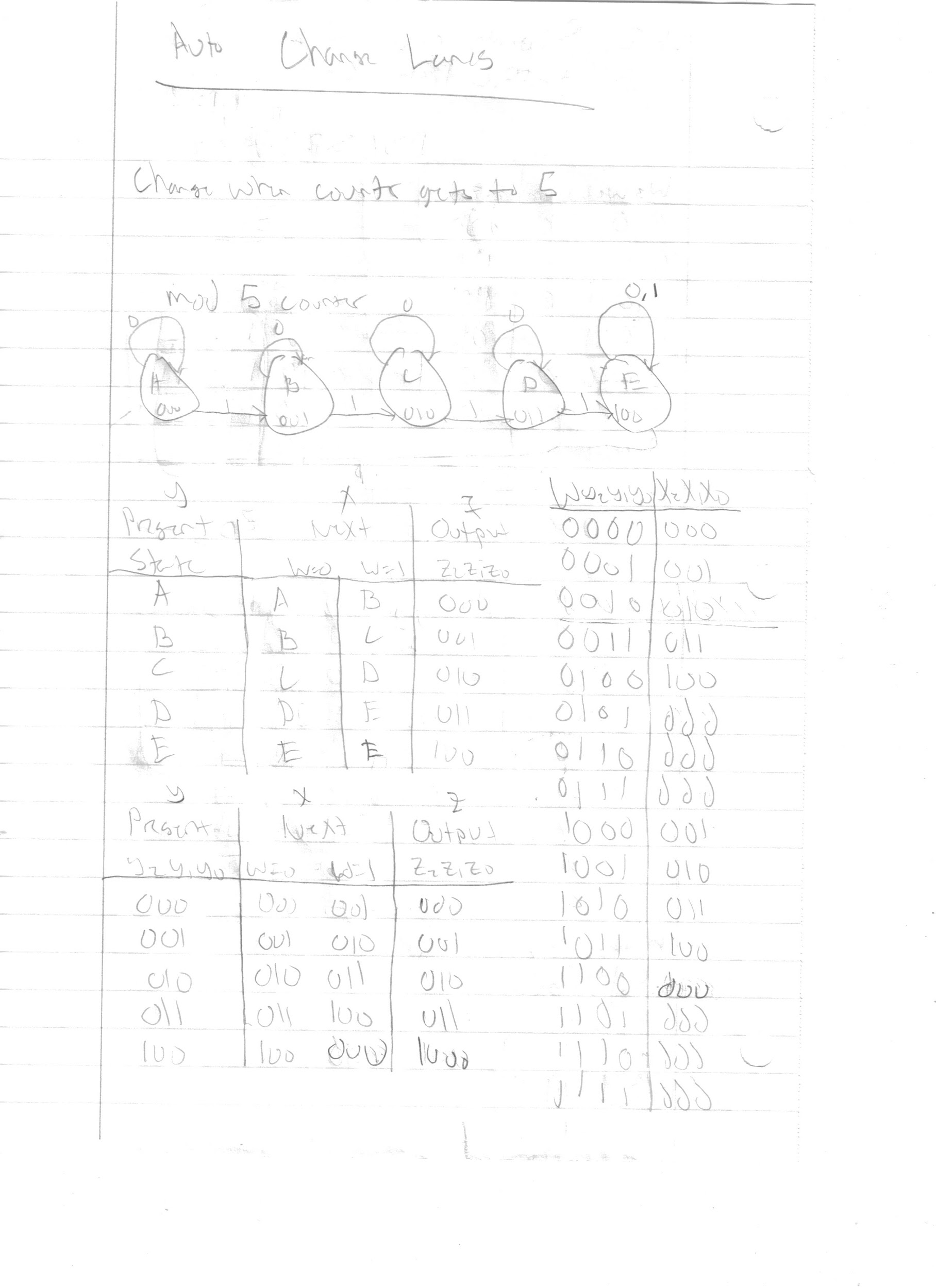
The inputs and outputs were derived from the following state diagram

**

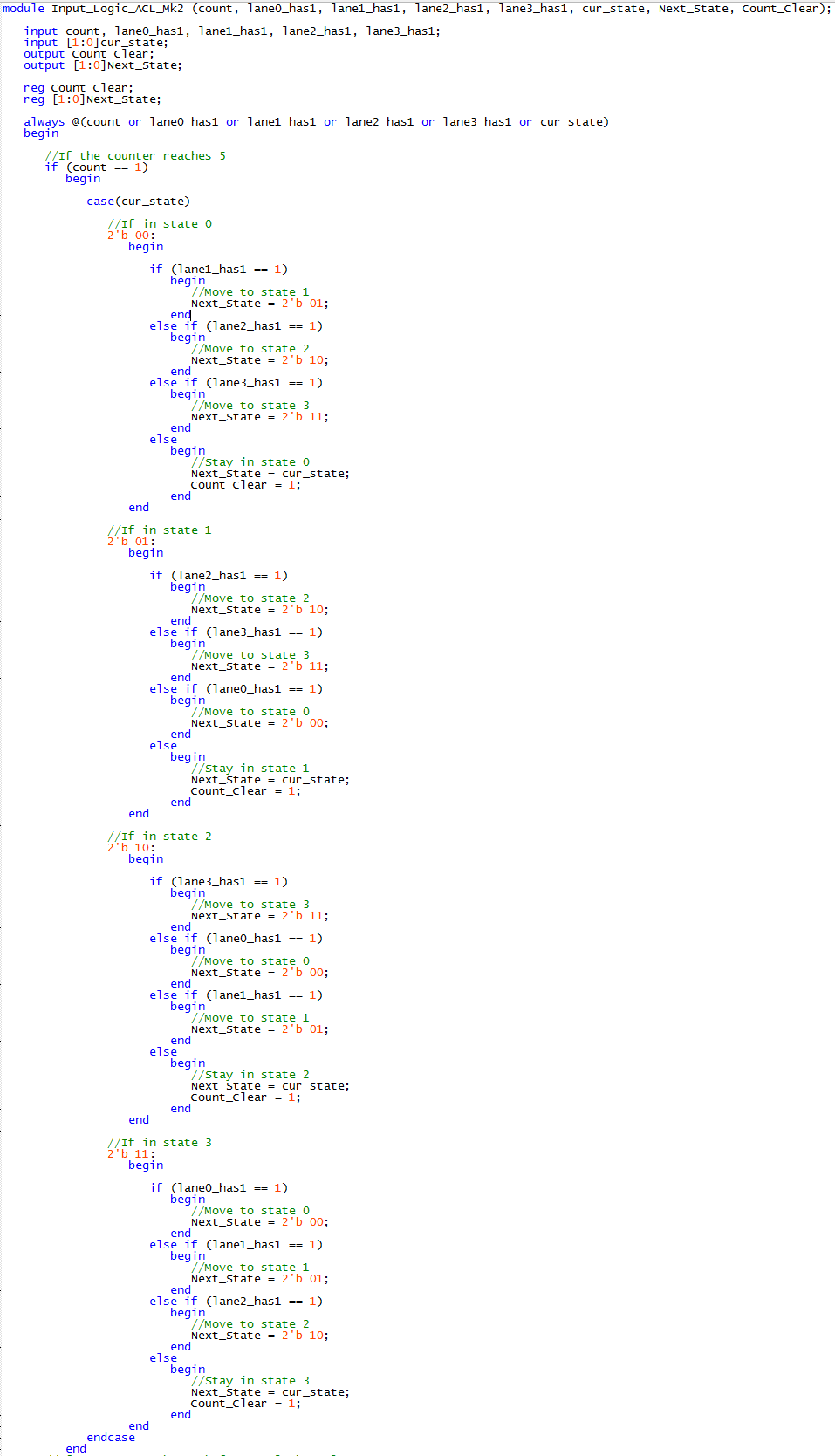
Which was then translated into this state table:

**

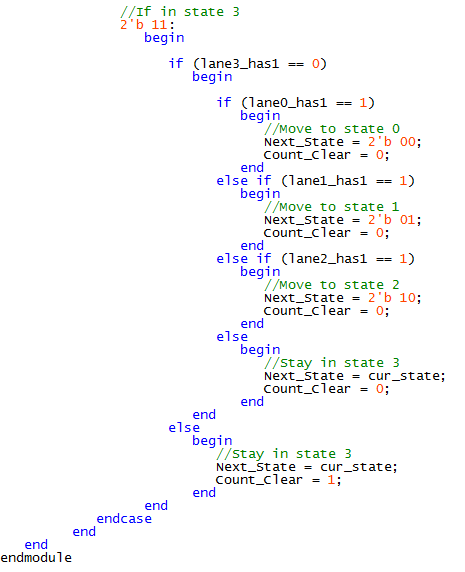
and then put into this truth table:

**

The output of the module Mod5\_Counter\_ACL is then inputted to the block Input\_Logic\_ACL\_Mk3 in the module ACL\_Machine shown here:

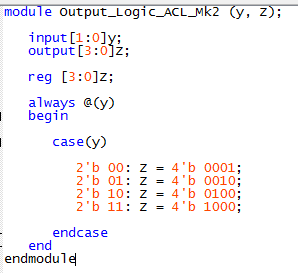
**





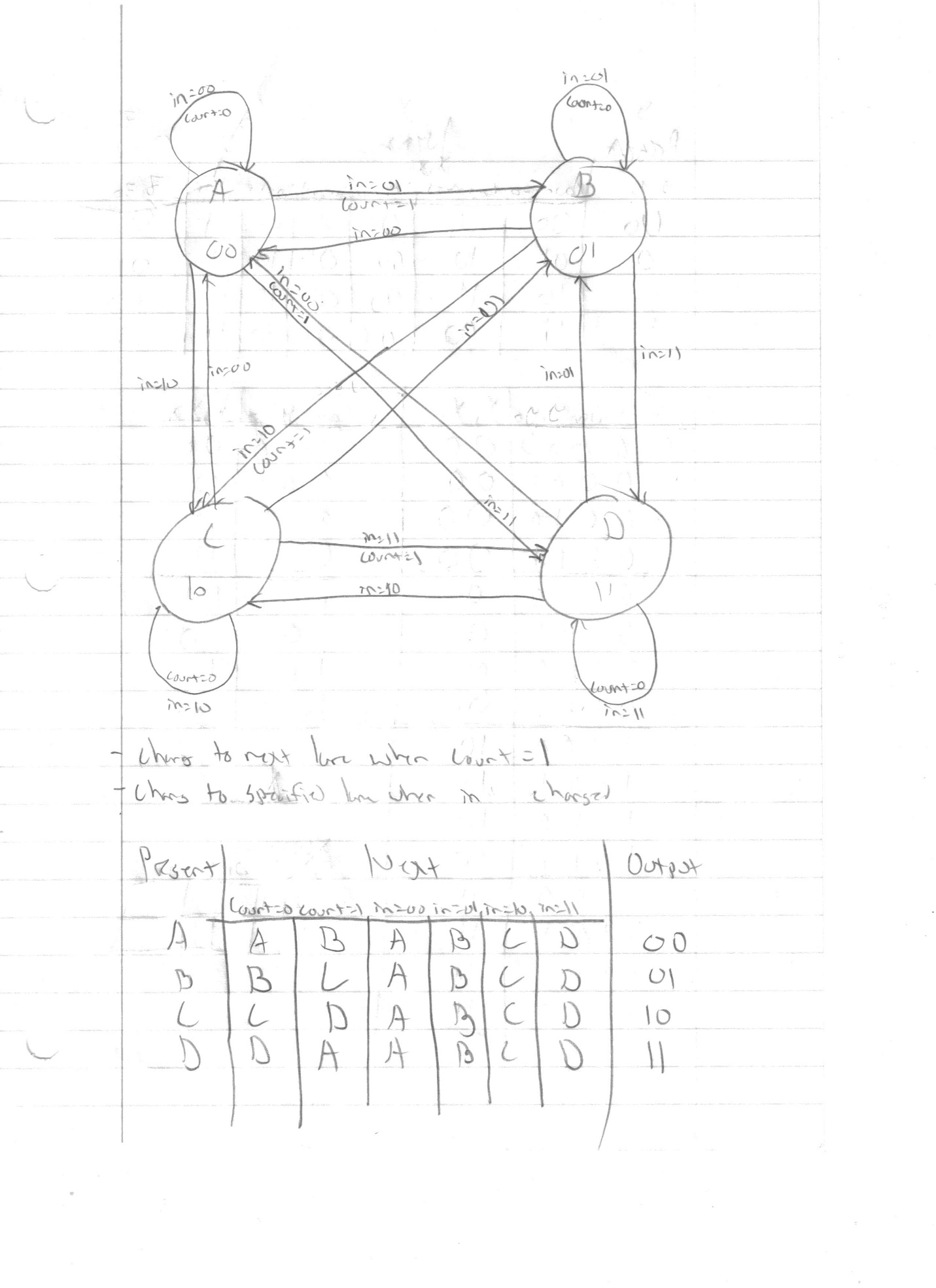
This module selects a next state for the ACL\_Machine based on if the counter has reached 5, the current state’s counter has reached 0, and or there is another counter that has at least one car in it. Contrary to the priority found in modules 4\_4bit\_Register\_File and TLCF, this always gives the green light to lane 0 first while starting out, and then checks in increasing order moving from 0 to 1 to 2 and so forth circling back to 0 after 3. If the current lane has reached 0 before the counter has reached 5, this then sends a signal, named Count\_Clear, that resets the mod5 counter back to 0.

This next state is fed into D Flip-Flops and then into the block Output\_Logic\_ACL\_Mk2 shown here:

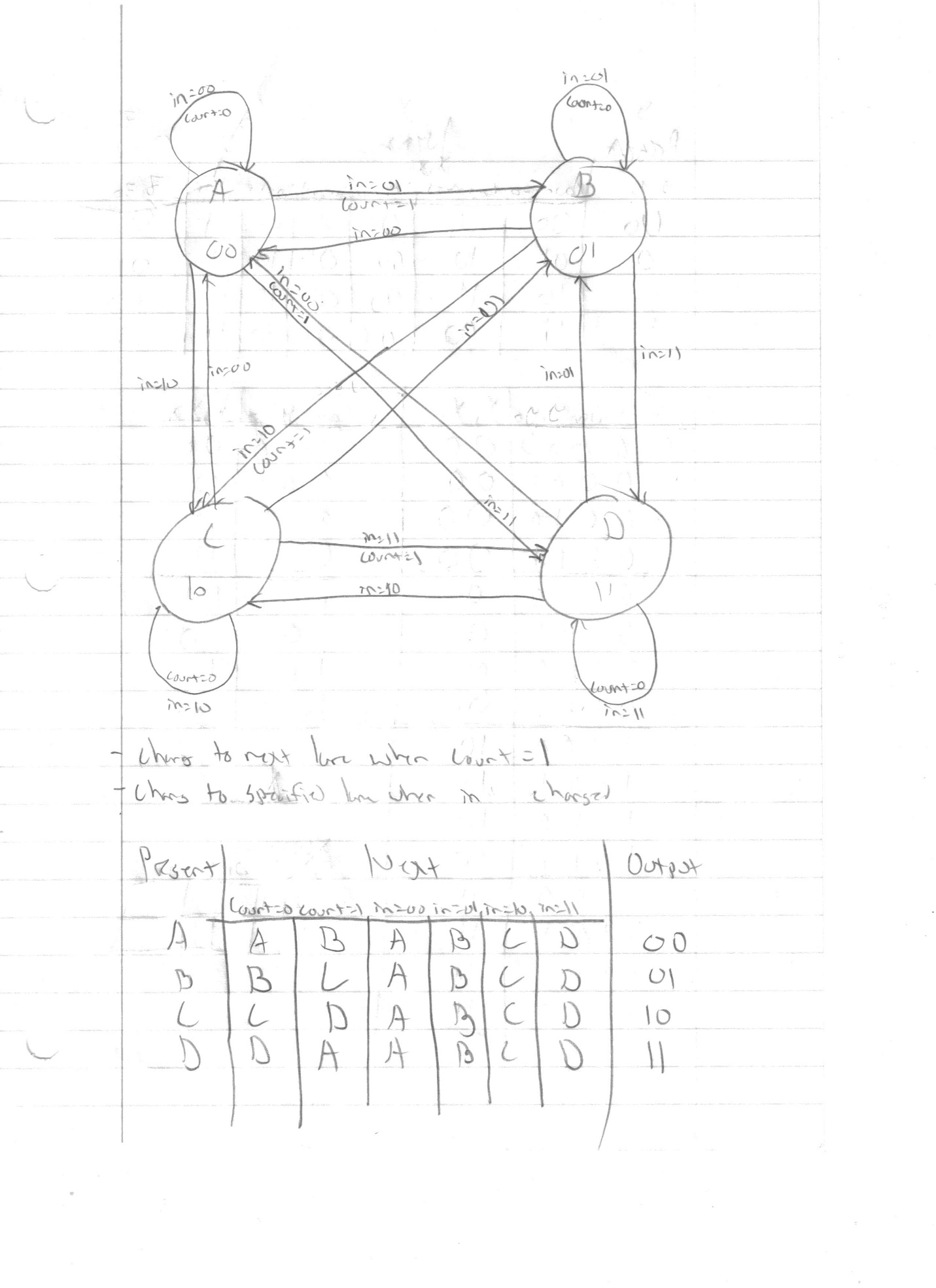
**

This, similar to how a decoder is one hot encoded, selects only 1 lane to decrement based on the current state.

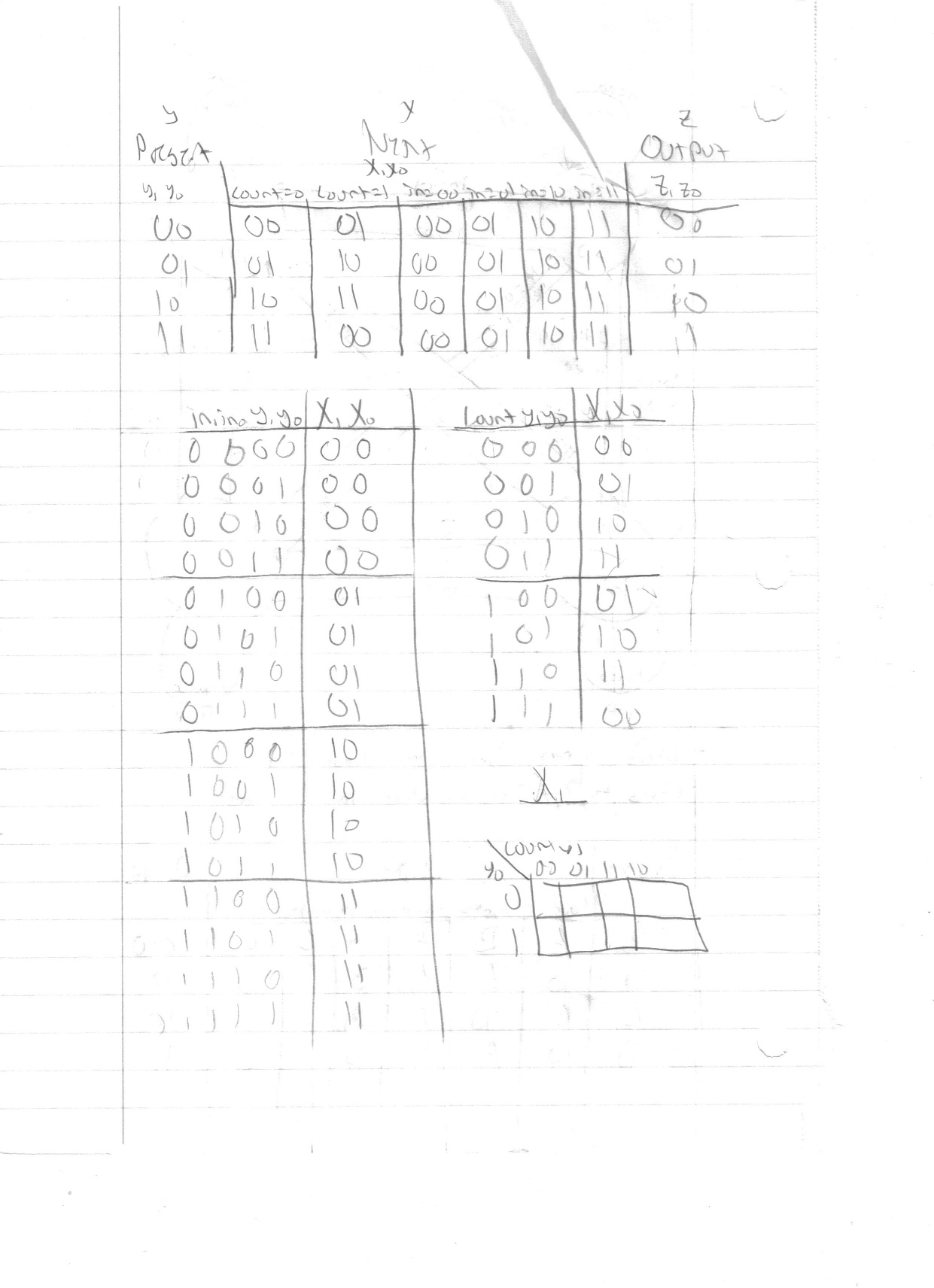
The input and output logic were derived from the following state diagram:



Which was then translated into the following state table:



Which was then put into a truth table:



This concludes this project report. As a final note, here is a list of the different inputs and outputs used in this project and some test cases to follow if needed:

Inputs:

**control\_switches** - switches SW0 through SW3 on the board, while initially loading lane capacities, used to select capacity for lanes. While in Mode A or B, setting switch 0 to 1 adds car lane 0, switch 1 to 1 adds cars to lane 1, and so on. Lane 3 has priority over lane 2, lane 2 has priority over lane 1 and so forth.

**mode\_switch** - switch 17 on the board, switches between the two modes specified by the project description.

**control\_button** - key 0 on the board. Used only for initial capacity loading.

Outputs:

**HEX0** - corresponds to current amount of cars in lane 0.

**HEX1** - corresponds to current amount of cars in lane 1.

**HEX2** - corresponds to current amount of cars in lane 2.

**HEX3** - corresponds to current amount of cars in lane 3.

**HEX4** - corresponds to maximum capacity of selected lane.

**LEDR0** - when lit, machine is in Mode A

**LEDR1** - when lit, machine is in Mode B

**LEDR17** - when lit, clock is 1

**LEDR13** - when lit and in Mode B, lane 0 has a green light

**LEDR14** - when lit and in Mode B, lane 1 has a green light

**LEDR13** - when lit and in Mode B, lane 2 has a green light

**LEDR16** - when lit and in Mode B, lane 3 has a green light

**LEDG0** - when lit, loading capacity to register 0

**LEDG1** - when lit, loading capacity to register 1

**LEDG2** - when lit, loading capacity to register 2

**LEDG3** - when lit, loading capacity to register 3

**LEDG7** - when lit, loading is done, and machine can proceed to Mode A or B

Test 1:

1. Set register 0 capacity to 4 using control\_switches and confirming capacity by pressing control\_button

2. Set register 1 capacity to 3 using control\_switches confirming capacity by pressing control\_button

3. Set register 2 capacity to 2 using control\_switches confirming capacity by pressing control\_button

4. Set register 3 capacity to 1 using control\_switches confirming capacity by pressing control\_button

5. While in Mode A, fill lanes to capacity using control\_switches

6. Switch to Mode B using mode\_switch

7. Let lanes empty, noticing how if a lane reaches 0, the green light moves to the next lane that has cars

Test 2:

1. Set register 0 capacity to 10 (or A in hex), using control\_switches

2. Set register 1 capacity to 8, using control\_switches

3. Set register 2 capacity to 12 (or C in hex), using control\_switches

4. Set register 3 capacity to 5, using control\_switches

5. While in Mode A, fill lanes to any capacity using, control\_switches

6. Switch to Mode B, using mode\_switch

7. Add cars to any lane if possible, noticing how if the amount of clock cycles on a lane reaches 5, the green light moves to the next full lane, and how if a lane has a green light and user is trying to add cars to it, the lane does not increment, and the green light moves on to the next full lane after 5 clock cycles.

Test 3:

1. Set register 0 capacity to any 4-bitnumber, using control\_switches

2. Set register 1 capacity to any 4-bit number, using control\_switches

3. Set register 2 capacity to any 4-bit number, using control\_switches

4. Set register 3 capacity to any 4-bit number, using control\_switches

5. While in Mode A, fill lanes to any capacity using, control\_switches

6. Switch to Mode B, using mode\_switch

7. Add cars to any lane if possible, noticing how if the amount of clock cycles on a lane reaches 5, the green light moves to the next full lane, and how if a lane has a green light and user is trying to add cars to it, the lane does not increment, and the green light moves on to the next full lane after 5 clock cycles.

8. Switch between Mode A and Mode B until satisfied with the machine.