Automated Chess

Group 52

- Aarabdh Tiwari 2019AAPS0241G
- Aayush kabra 2019AAPS0222G
- Aditya Agarwal 2019AAPS0243G
- Ashutosh Gupta 2019AAPS0223G
- Sarvesh Garge 2019AAPS0233G
- Siddhant Sharma 2019AAPS0226G

Question 9

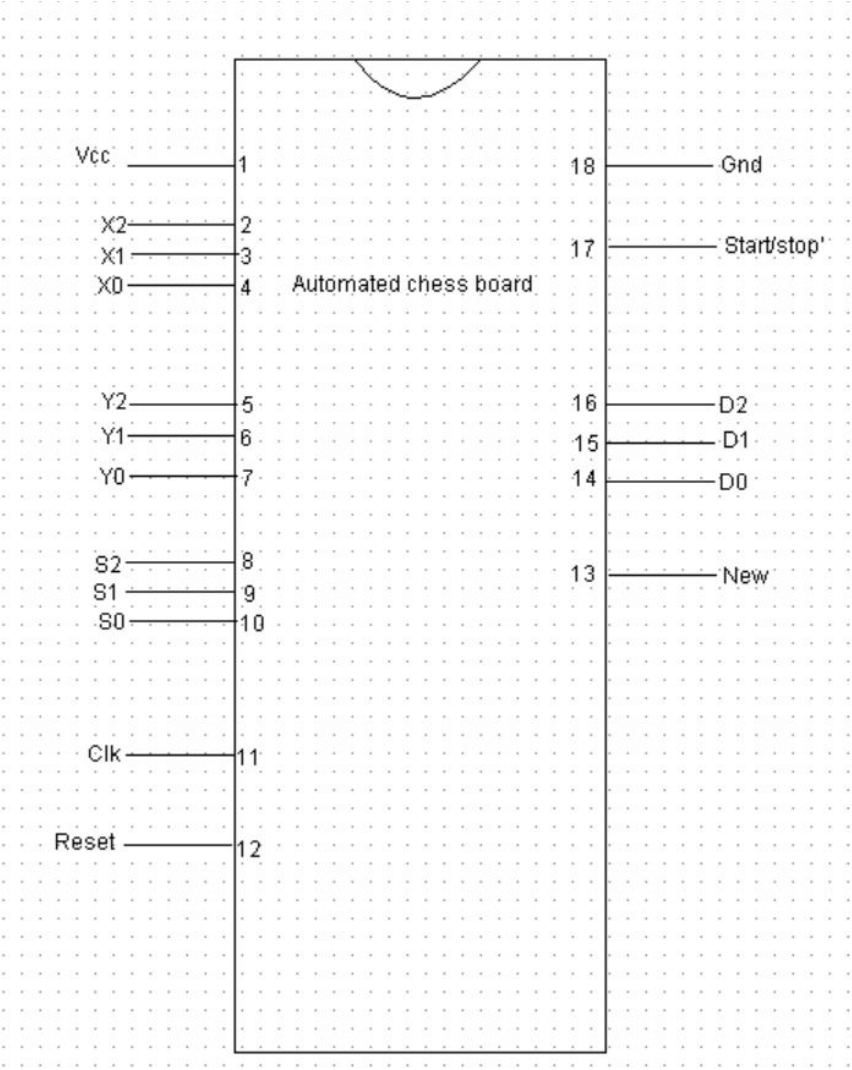
Design an automated chessboard where the user selects the chess piece and inputs the final position. Based on user input design a system that moves the chess pieces. Assume that your chess board is only automated for the bishop and the knight pieces only. Assume that these pieces are automated by servo motors that move one box at a time in any direction.

Top level block diagram

➔ Inputs

- ◆ X - This is a 3 bit input allowed from $(000)_2$ to $(111)_2$. Input X depicts the x-coordinate the user wishes the selected chess piece to move to.
- ◆ Y - This is a 3 bit input allowed from $(000)_2$ to $(111)_2$. Input Y depicts the y-coordinate the user wishes the selected chess piece to move to.
- ◆ Select - This 3 bit input allows the user to select the chess piece to move. The encoding of select input is as follows: $(0XX)_2$ - Knight, $(100)_2$ - Bishop, $(101)_2$ - Queen, $(110)_2$ - King, $(111)_2$ - Rook.
- ◆ Reset - Resets all the pieces to their respective initial positions. (Here initial positions of white pieces have been coded)

◆ Clk - Clock signal of 0.5Hz frequency (at maximum).



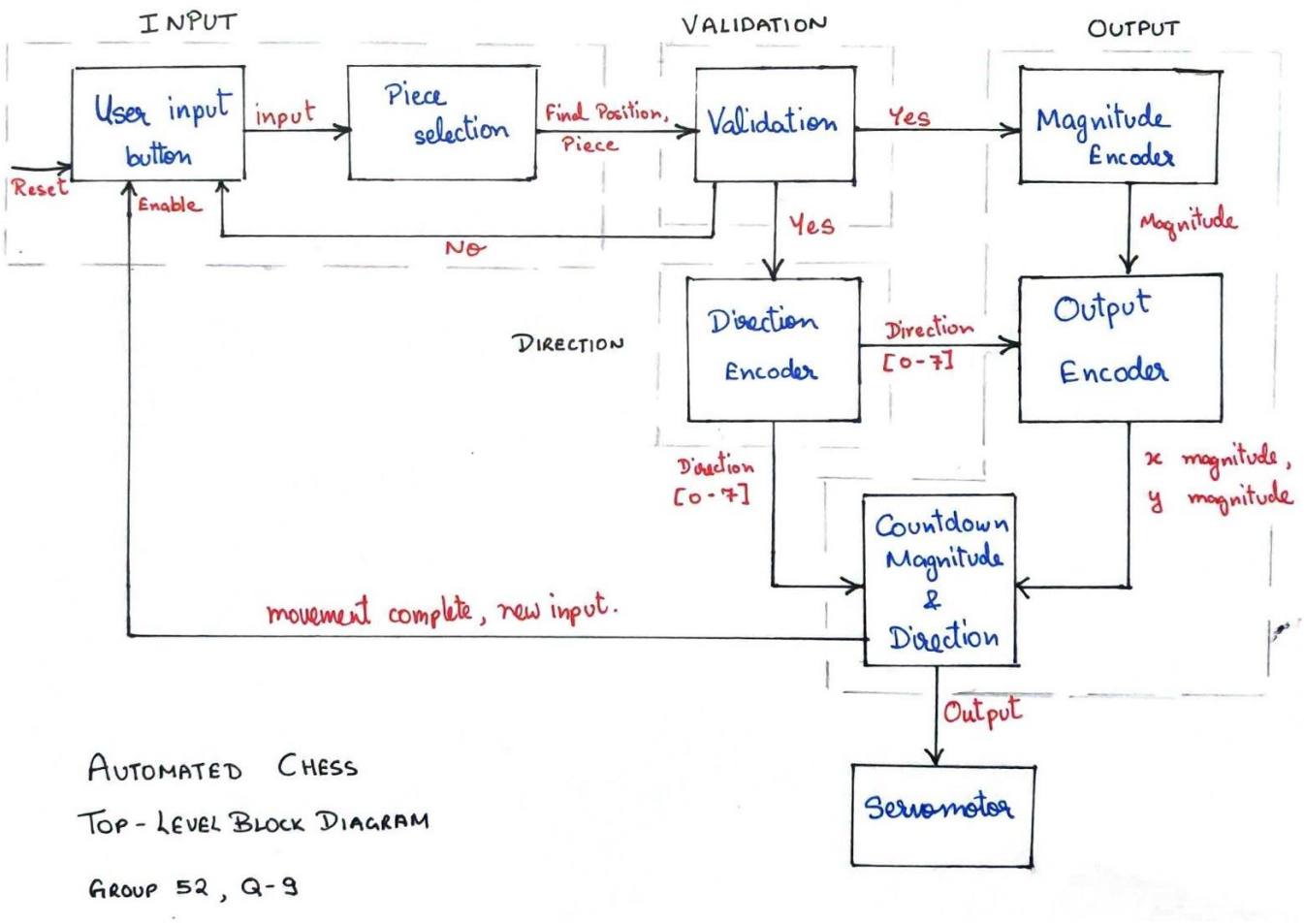
(Top level IC of the circuit)

➔ Outputs

◆ Start/Stop' (in short 'Go') - This output when High, tells the motor to move a box in the specified direction and when it is Low the motor stops irrespective of the direction output.

◆ Dir - This 3 bit output specifies the direction for the movement to the motor. The direction encoding is as follows: $(000)_2$ - East, $(001)_2$ - North East and so on in an anti-clockwise direction until $(111)_2$ - South East.

◆ New - This output when High signifies that the user needs to give a new input.



(Block diagram)

Assumptions

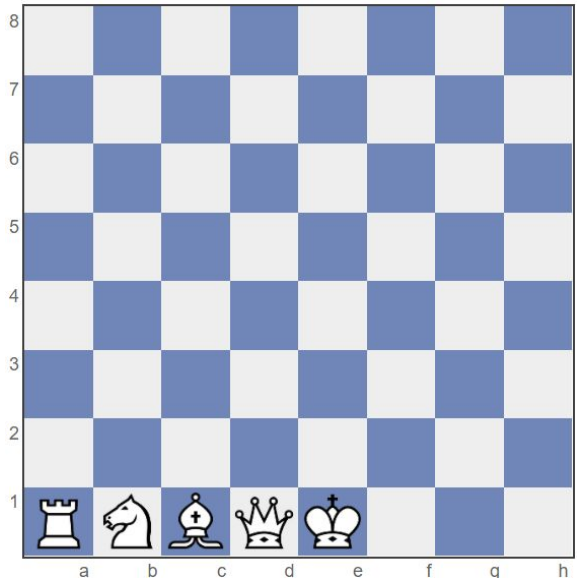
1. Structural Overview:
    - The entire Chess-board is pictured in the form of a coordinate-system grid, with the bottom left corner as (0,0) and the top right corner as (7,7).
    - The chessboard is functional for White King, Queen and one (*for complete just repeat the circuit*) Rook, Bishop and Knight who will be present at their initial positions.
    - The circuit also currently assumes that the chess board only has these pieces and currently is not capable of obstacle avoidance in the path.

2. Input Assumptions:
    - The user inputs 2 numbers, the first will be the final position of the piece on the X-axis, and the second will be the final position on the Y-axis.
    - The final values entered by the user must be within the 8x8 grid. For example, the user cannot enter (9,9).
    - Users also enter one number to select which piece they want to move. The coding will be mentioned.(example 0-3=Knight, 4=Bishop, 5=Queen etc)
    - If the final coordinates entered by the user are found to be an invalid chess-move for the selected piece, the user will have to input new coordinates.

3. Output Assumptions:
    - The circuit gives outputs in the form of an Encoded Direction and Magnitude.
    - The servo-motor takes in inputs of Direction and Magnitude and a down counter is used to allow the motor to know how many more blocks it must move.
    - The motor moves from the center of one block to the next in exactly 2 seconds. The internal speed of the motor is controlled such that for diagonal moves, it moves at $speed_{diag}$= $speed_{ver/hor}$*$\sqrt{2}$, in order to keep the time taken fixed at 2 seconds.
    - The motor is assumed to have the same clock signal and sample the inputs given to it at a falling edge of clock, as the outputs from the counter are triggered by the rising edge of clock.

4. Direction and Magnitude Control:
    - Direction is encoded counter-clockwise from the east.That is, East=000, North-East=001, North = 010 and so on till South-East=111.
    - The magnitude for the knight is fixed at 3, but its direction changes after 2 steps and thus the motor remains functional.

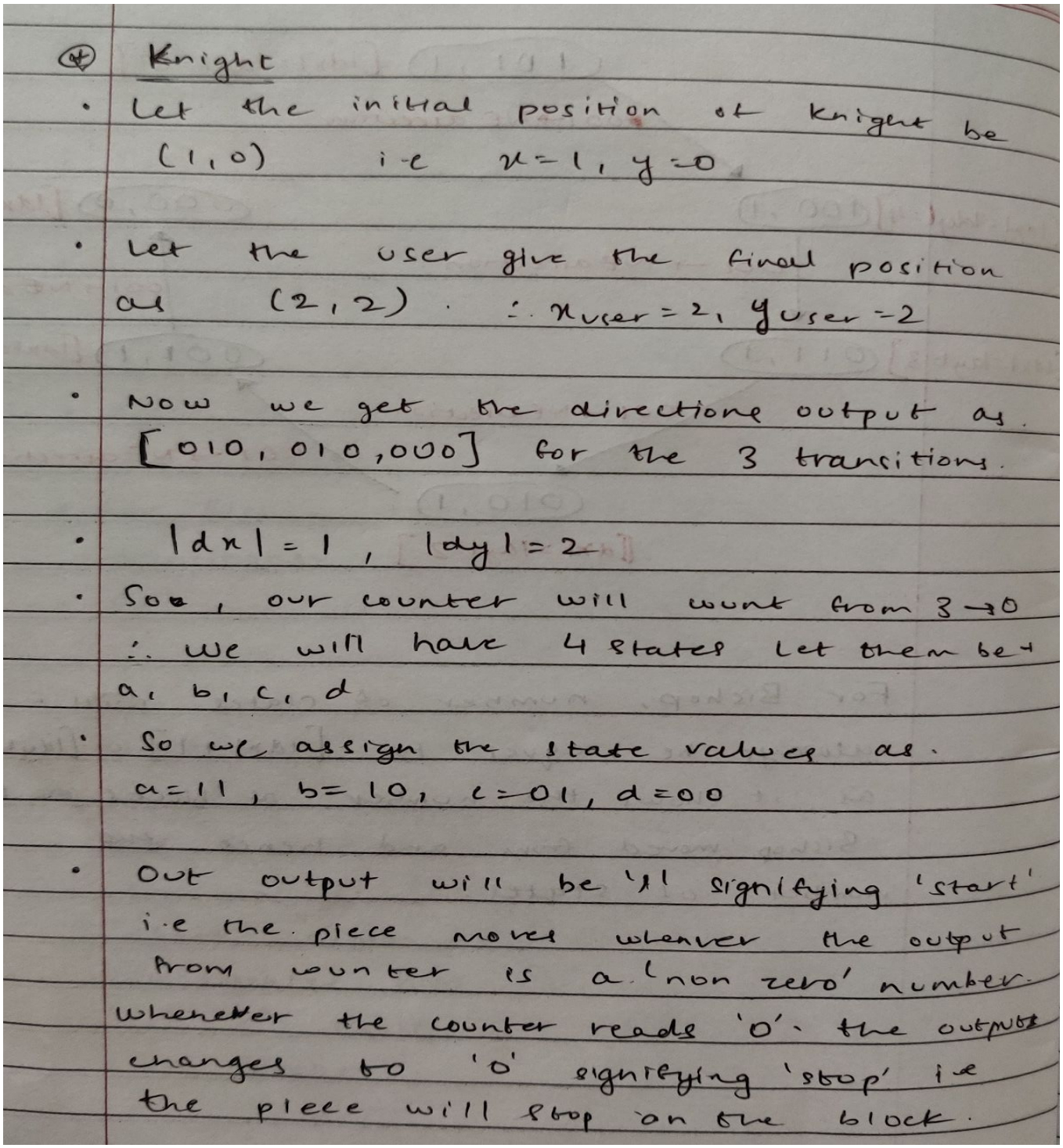
5. Initial Positions:
    - The board's initial setup is assumed to be as shown below, with the Rook piece at (0,0), Knight at (1,0) and so on.



(Initial setup)

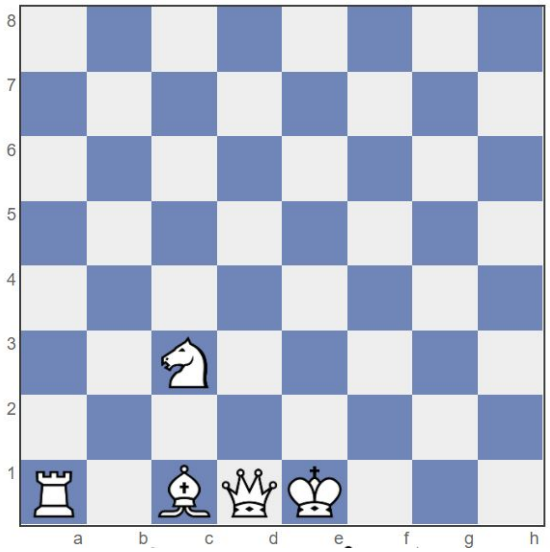
State Diagram *(for one input/output combination)*

1. For Knight

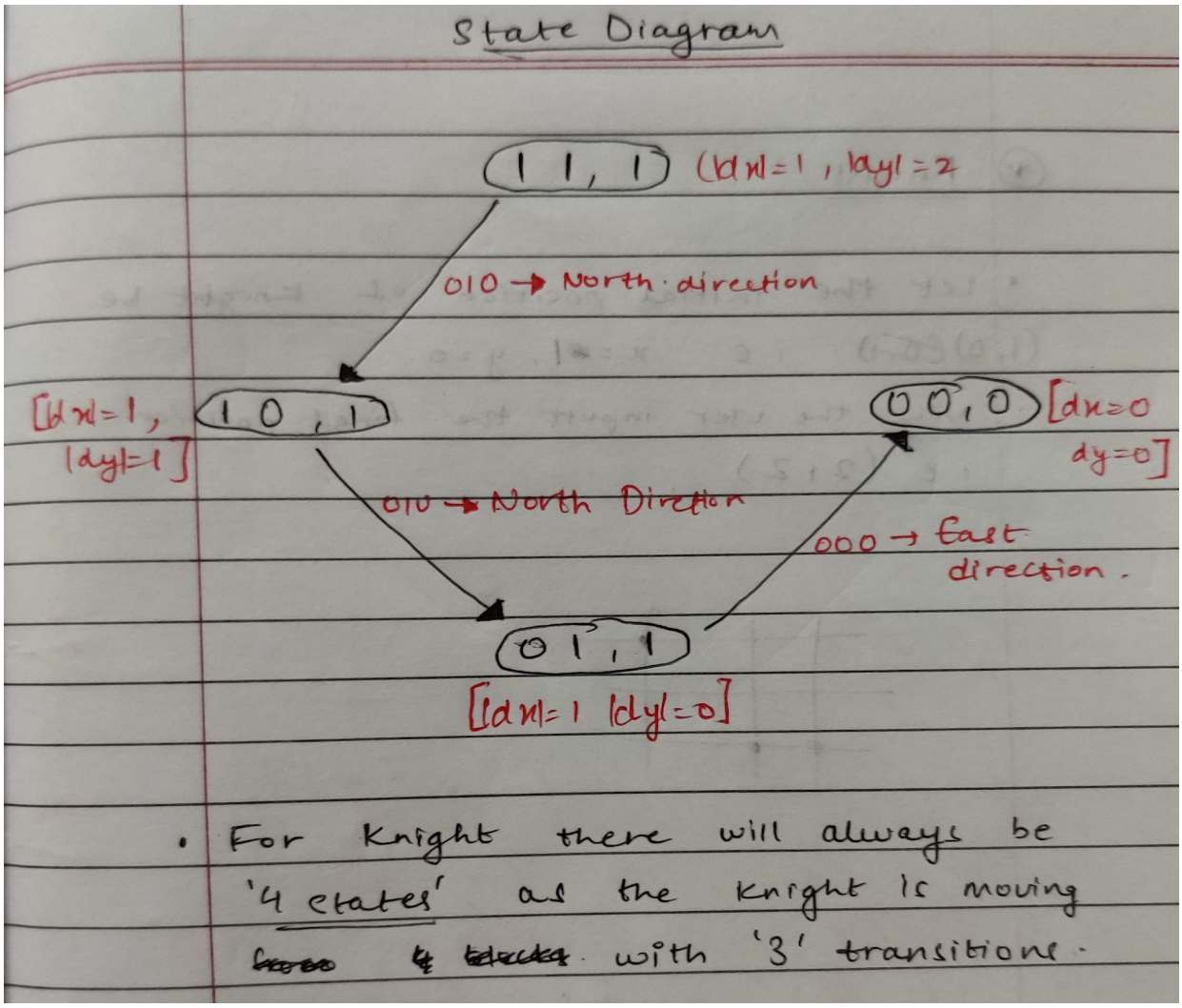


④ Knight

- Let the initial position of knight be (1,0) i.e $x=1, y=0$
- Let the user give the final position as (2,2). $\therefore x_{user}=2, y_{user}=2$
- Now we get the directions output as. [010, 010, 000] for the 3 transitions.
- $|dx|=1$, $|dy|=2$
- So, our counter will count from 3→0 ∴ we will have 4 states Let them be a, b, c, d
- So we assign the state values as. a=11, b=10, c=01, d=00
- Our output will be '1' signifying 'start' i.e the piece moves whenever the output from counter is a 'non zero' number. Whenever the counter reads '0'. the output changes to '0' signifying 'stop' i.e the piece will stop on the block.



(The movement knight will follow)

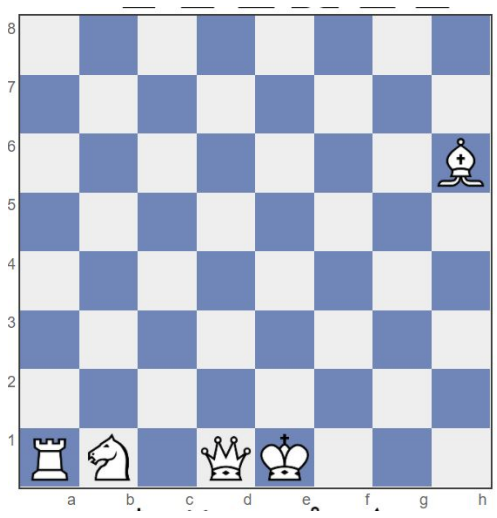


X	Y	S	clk	Go	Dir
2	2	000	0	0	xxx
2	2	000	0	0	000
2	2	000	1	1	010
2	2	000	0	1	010
2	2	000	1	1	010
2	2	000	0	1	010
2	2	000	1	1	000
2	2	000	0	1	000
2	2	000	1	0	xxx

(Logging table for the Knight demo)

Circuit demo video for Knight is included in the zip file by the name “Knight demo.mp4”. The video is even linked here.

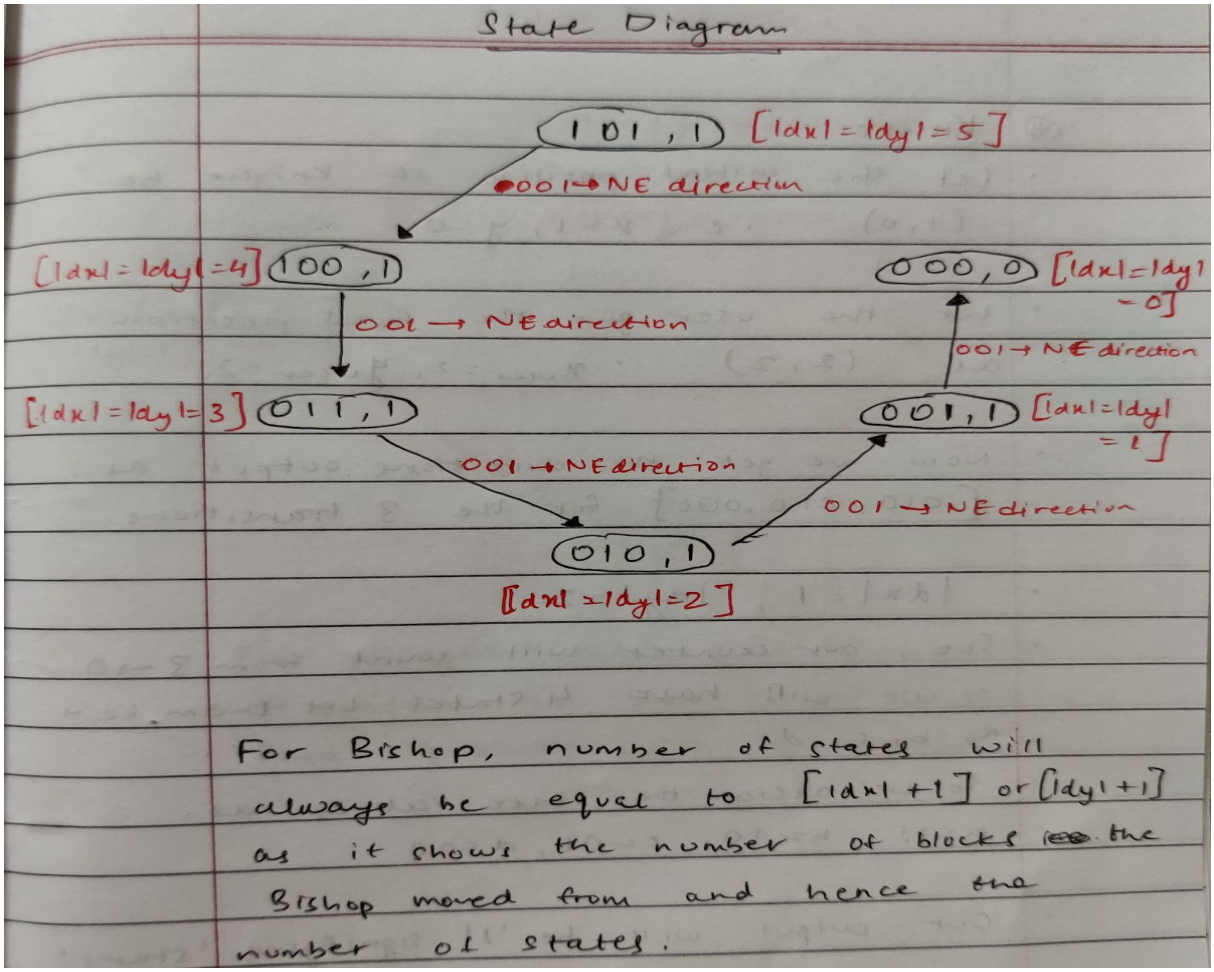
2. For Bishop



(The movement bishop will follow)

④ Bishop :→

- Let's say intial position of Bishop is (2,0) i.e $x=2$, $y=0$
- Now, the user inputs the output as (7,5) ie $x_{user}=7$, $y_{user}=5$
- $|dx|=5$, $|dy|=5$
- Now, we will get the ~~output~~ direction output as $(001)_2$ i.e 'North East'
- For Bishop direction is same for all transitions.
- So our counter will count from '5 to 0'
  ∴ We will have 6 states →
  a, b, c, d, e, f.
  Let a= 101, b= 100, c= 011, d= 010, e= 001
  f= 000
- Our output will be 1 whenever counter is 'non-zero' number signifying 'start' so our piece moves. and when our counter hits '0' output changes to '0' signifying 'stop' so our piece stops.



X	Y	S	clk	Go	Dir
7	5	100	0	0	001
7	5	100	0	0	xxx
7	5	100	0	0	001
7	5	100	1	1	001
7	5	100	0	1	001
7	5	100	1	1	001
7	5	100	0	1	001
7	5	100	1	1	001
7	5	100	0	1	001
7	5	100	1	1	001
7	5	100	0	1	001
7	5	100	1	1	001
7	5	100	0	1	001
7	5	100	1	0	xxx

(Logging table for the Bishop demo)

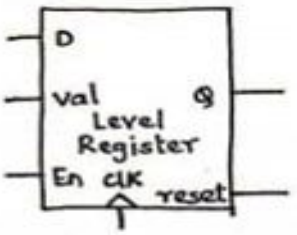
Circuit demo video for Bishop is included in the zip file by the name “Bishop demo.mp4”. The video is even linked here.

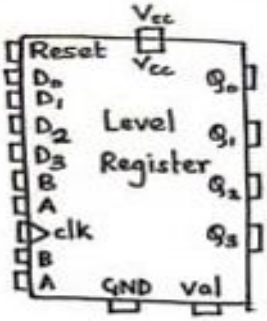
Pin out diagram

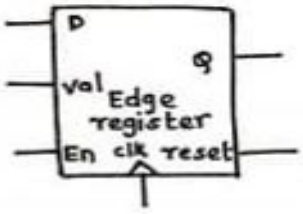
KEY:

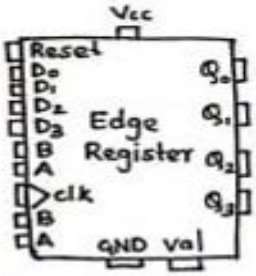
Please note that some of the ICs used in the circuit are customized to ensuring simpler design, as well as provide additional functionalities to the circuit. The chips are as follows:

1) Level and Edge Register

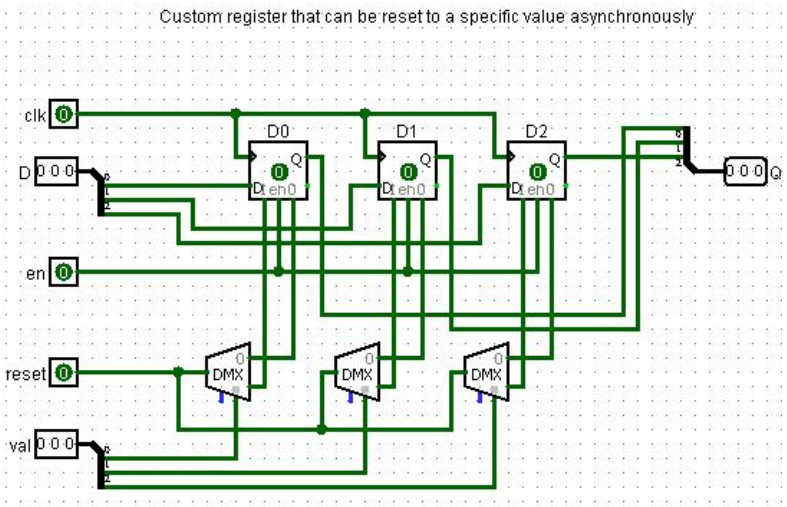








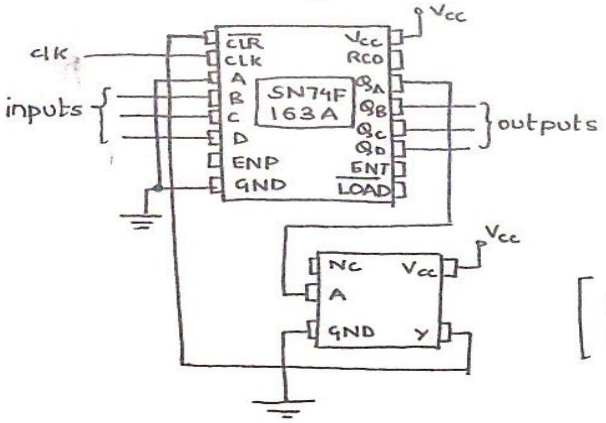
Symbols	Explanation
D	Inputs to the register
val	The value to which the register is set after using reset pin
En	When En = 1, register is active
Q	Output of register
Reset	When reset = 1, register is asynchronously set to 'val'



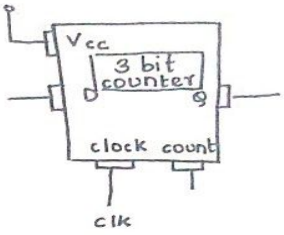
(Custom register module logisim implementation)

2) 3 bit counter

Pinout diagram of 3 bit counter using a 4 bit counter



≡



[Please note that count acts as an enable signal]

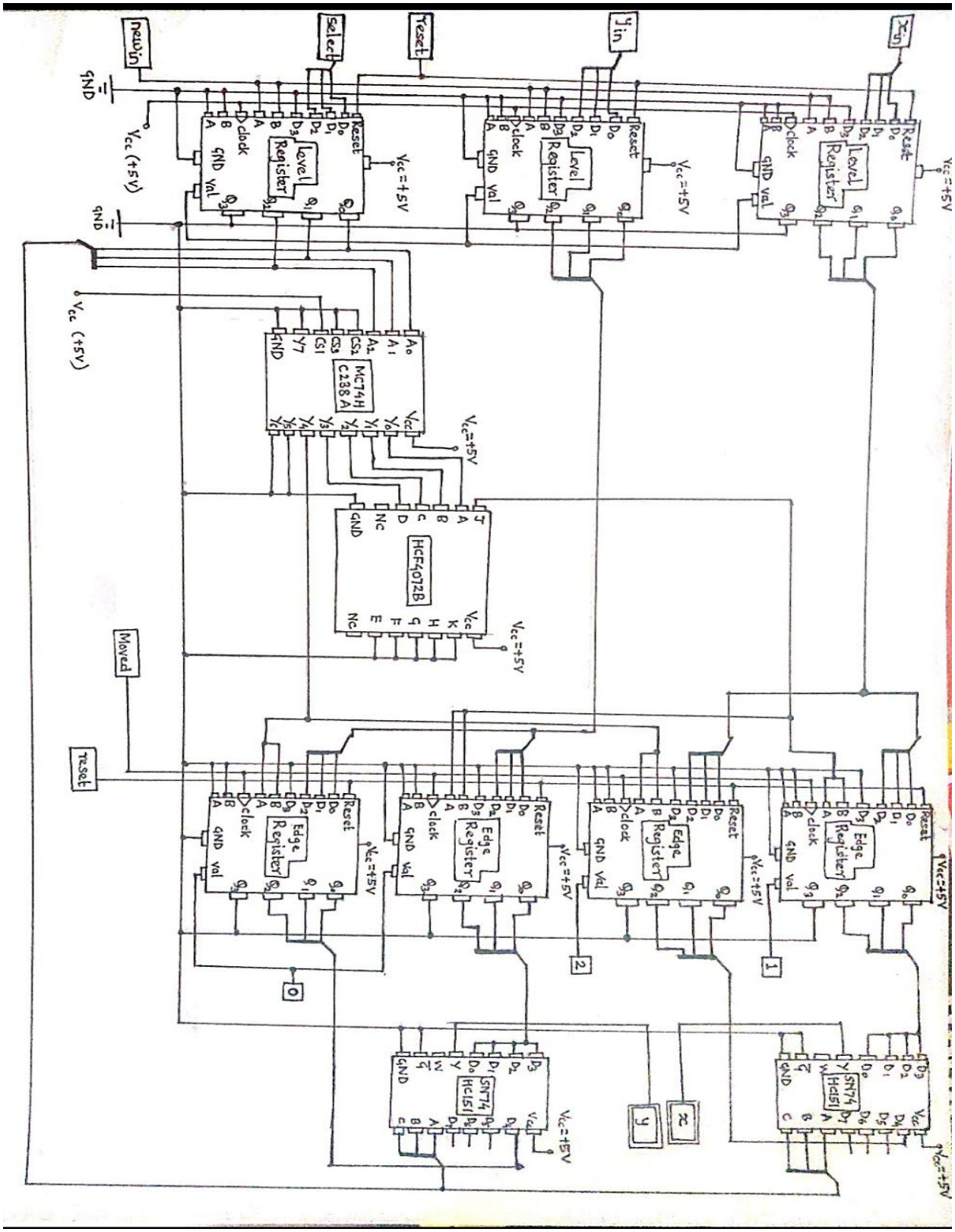
3) The input signals to the given part of the circuit (which may be provided by the user or are coming from any other segment of the circuit as input to the given circuit) are shown as:

input

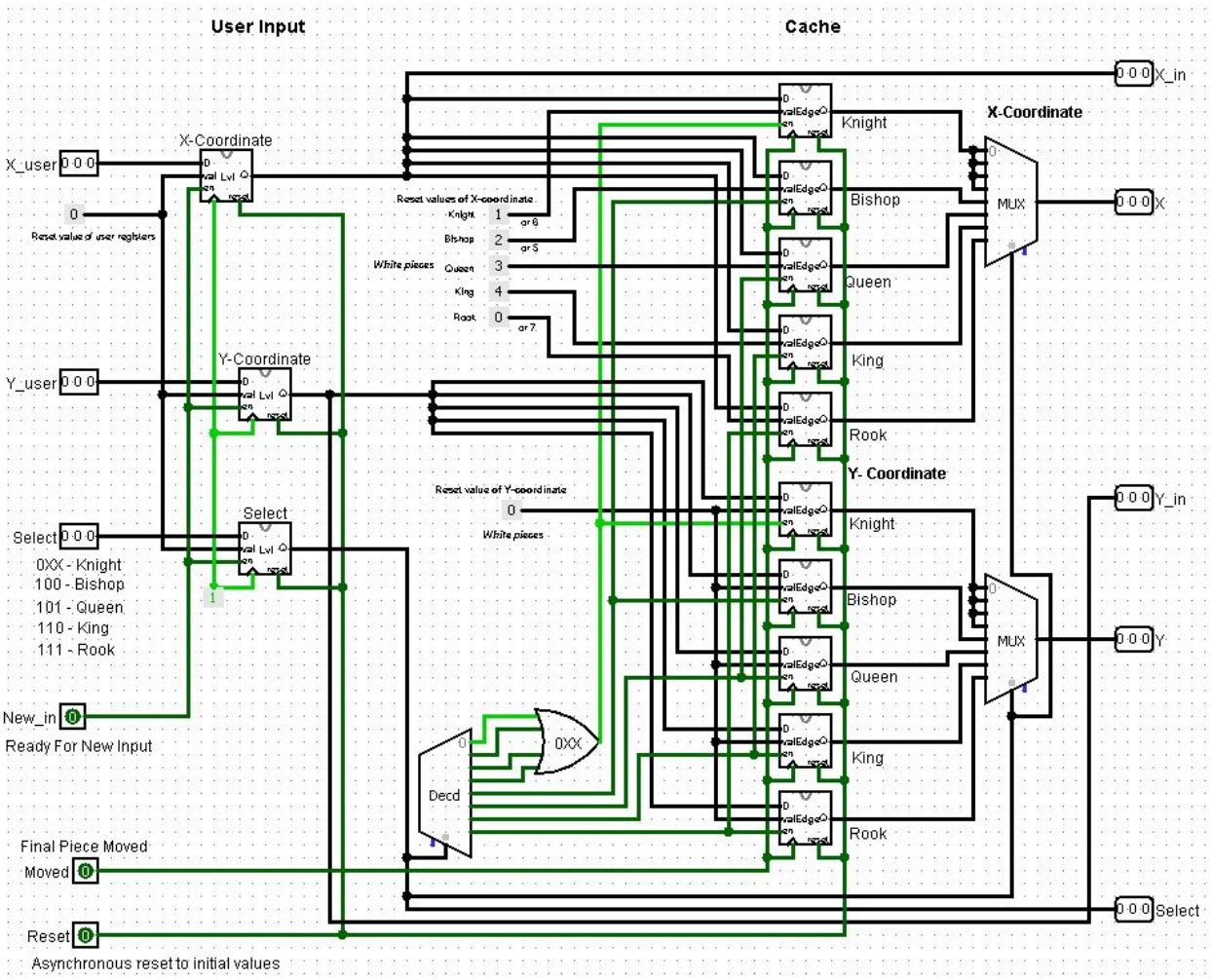
Similarly, the outputs of the circuit segment are shown as:

output

4) Splitters and mergers are used to ensure that the pins of the IC are connected to compatible bit lines. In some instances, decimal numbers are shown as input wherein these numbers can be formed using a merger.



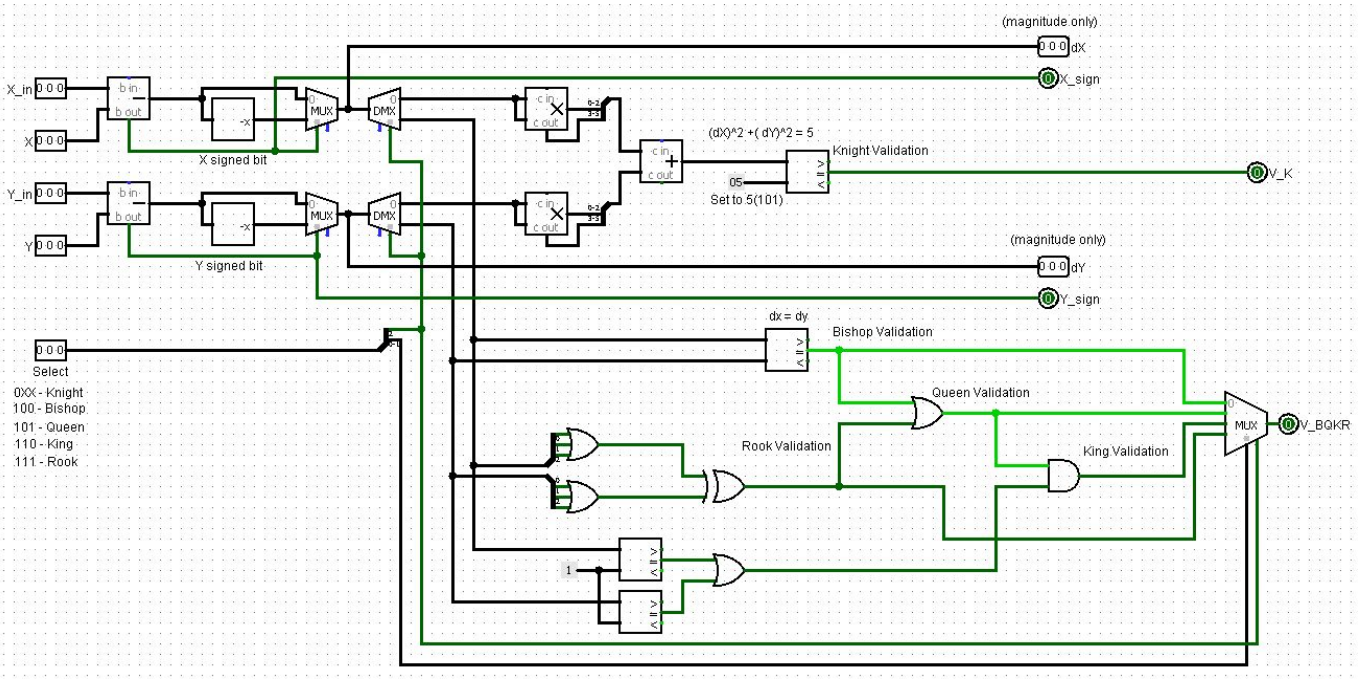
(Input module pin out diagram)



(Input module logisim implementation)

1. **Input** module is the first module that interacts with the user. When ‘New’ input goes high it will accept and store the user input in the initial (level sensitive) registers and then disable them, to reject noise affecting the values between move cycles.

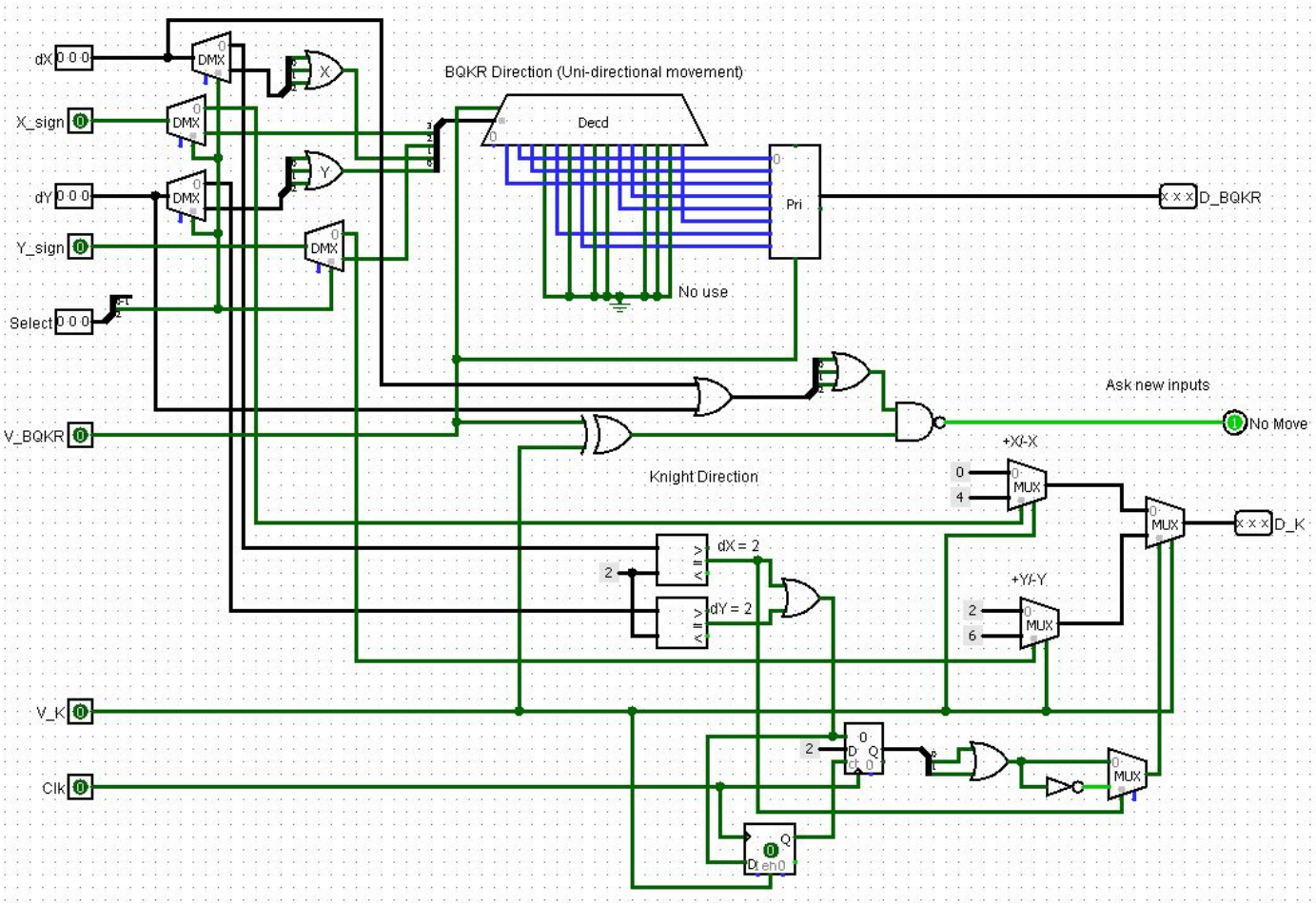
   Then when the ‘Moved’ input has falling edge the particular position inputs are stored on respective (edge triggered) registers to keep a track of positions of each piece.



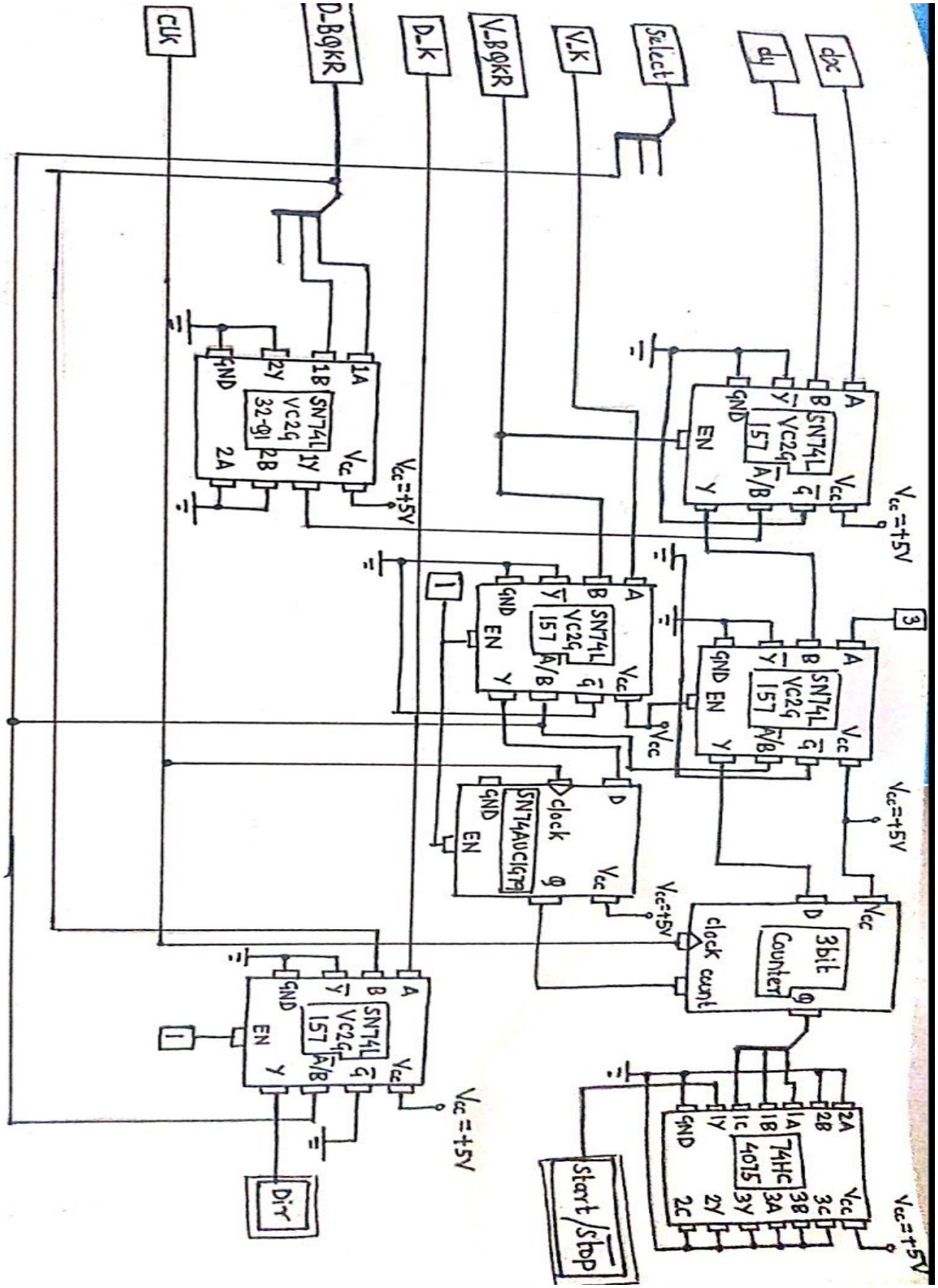
(Path validation module logisim implementation)

2. **Path validation** module takes in user inputs of ‘X’, ’Y’, ‘Select’ and the stored values of the current position of the chess piece. Then this module checks for the validity of the path for the specified piece, and gives the magnitude and signed bit for both directions as output. If the path is valid then the circuit moves further otherwise the ‘New’ input goes high.

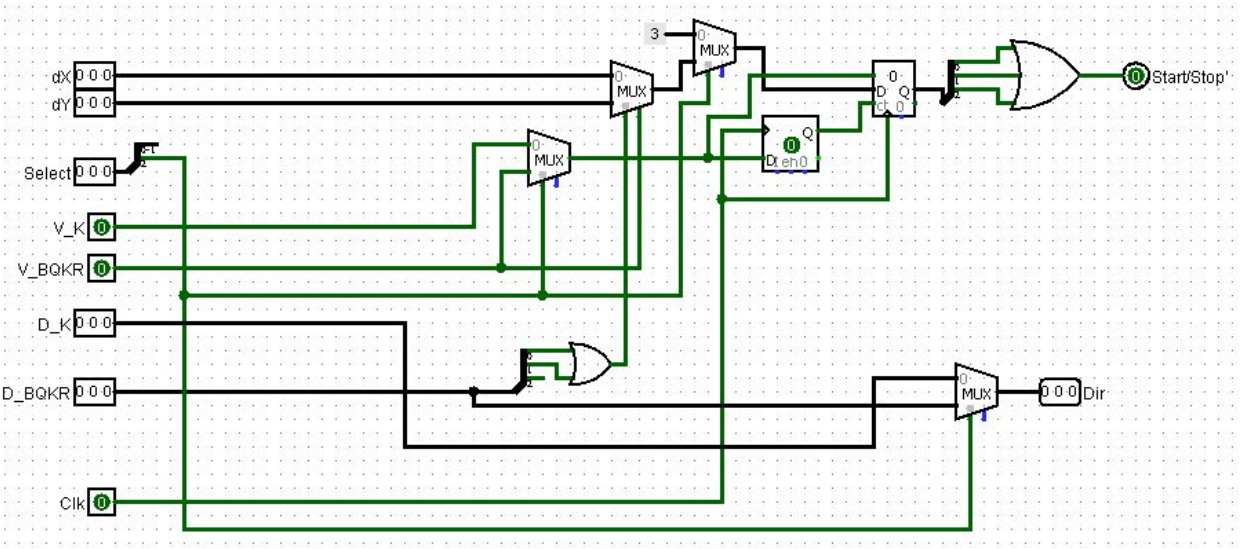
3. **Direction encoding** module takes the magnitude, signed bit of each direction, validity outputs and ‘Select’ as input. Based on the piece selected, validity of path and the signed bit this module produces an encoded sequence of direction for the motor to follow.



(Direction encoding module logisim implementation)



(Input module pin out diagram)



(Output module logisim implementation)

4. **Output** module takes in the magnitude, encoded direction and ‘Select’ and then just produces a timed signal for the motor to interpret.

Additional Functionalities

1. **Additional Pieces** → Our chess board is functional not only for the Knight and Bishop Pieces, but also for the King, Rook and Queen pieces. Also validity checker for Pawn is included but not integrated in the circuit.
2. **Validity Checker** → Before moving the pieces from their current position to final position, our circuit checks the validity of the move and confirms whether the move is a legal chess move or not. If the move is illegal, the user gets to input their final position values again.
3. **Modular** → The circuit makes use of multiple modules that we have designed, brought together to carry out the chess gameplay.
4. **Savings** → The reusability of modules in our circuit enables us to reduce the number of required MSI components and gates, thus reducing costs. The circuit also uses well controlled Enables for all its components, which only execute their required functions when asked to, thus reducing power consumption as well.

5. **Extendable** → Using our existing modules and chips, the board can easily be extended to both sides of the chess board. In our circuit we have also shown a RAM component as a subpart, which can be used for obstacle mapping, which would be useful for the “killing” move in chess and to avoid the obstacles in the path.
6. **Reset** → The initial reset allows the game to be reset, with all pieces back at their starting positions, this enables the game to be played again from scratch.
7. **Custom Registers** → To carry out the above function, we have designed registers that reset the piece location values to their initial positions.

Bill of Materials

Sr. No.	IC code	Common name	Manufacturer	Quantity
1	SN54/74LS83A	Four Bit Binary **Adder**	Motorola	5
2	DM74LS85	4-Bit **Magnitude Comparator**	Fairchild Semiconductor	8
3	CD54/74HC157	Quad 2-Input **Multiplexers**	Texas Instruments	15
4				
5	SN 54284	4-Bit **multiplier**	Texas Instruments	2
6	CD4070BC	Quad 2-Input **EXCLUSIVE-OR** Gate	Fairchild Semiconductor	
7	SN74LVC2G32-Q1	DUAL TWO-INPUT POSITIVE-**OR** GATE	Texas Instruments	3
8	DM74LS153	Dual 1-of-4 Line Data **Selectors/Multiplexers**	Fairchild Semiconductor	1
9	SN74LVC2G08-EP	DUAL 2-INPUT POSITIVE-**AND** GATE	Texas Instruments	1
10	SN54/74LS138	1-OF-8 **DECODER/ DEMULTIPLEXER**	Motorola	3

11	M54HC4072	DUAL **4 INPUT OR** GATE	SGS-Thomson Microelectronics	1
12	MC14515B	4-to-16 Line **Decoder**	On Semiconductor	1
13	MC14532B	8-Bit **Priority Encoder**	On Semiconductor	1
14	SN54284, SN74294	4 Bit by 4 Bit parallel binary **multiplier**	Texas Instruments	2
15		Counter		
16	SN74AUC1G79	Single Positive-Edge-Triggered **D-type Flip-Flop**	Texas Instruments	2
17	SN74LVC2G02	Dual 2-Input Positive-**NOR** Gate	Texas Instruments	1
18	74HC4075; 74HCT4075	Triple **3-input OR** gate	nexperia	1
19	SN74LVC2G08-EP	DUAL **2-INPUT POSITIVE-AND** GATE	Texas Instruments	1
20	SN74F163A	4 bit Counter	Texas Instruments	1
21	SN74LVC1G04	Single NOT gate	Texas Instruments	1
22	SN74HCS72	Dual D-Type Negative-Edge-Triggered Flip-Flops	Texas Instruments	15
23	SN74LVC1G373	Single D-Type Latch	Texas Instruments	9

Appendix

All the data sheets for the ICs used in the design are listed here:

https://drive.google.com/drive/folders/182CY4fGHlwm83EDcNP0AI_M9-RHQEYUB?usp=sharing

(This report was made as a submission for the project of the 'Digital Design' course for AY 2020-21, in BITS Pilani KK Birla Goa Campus)

--The End--