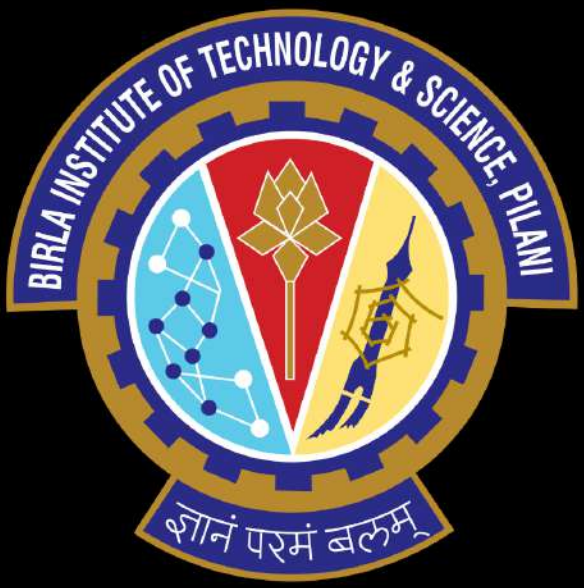


F215 - DIGITAL DESIGN

DESIGN ASSIGNMENT

GROUP 62

TOPIC 18 - Digital taxi fare meter

Group members-

SOUMYA UPADHYAY -2019A8PS0520G.

KARAN PESHWANI-2019A8PS0533G

RAGHAV KHANDELWAL-2019A8PS0541G.

PRANAV GOYAL-2019A8PS0548G .

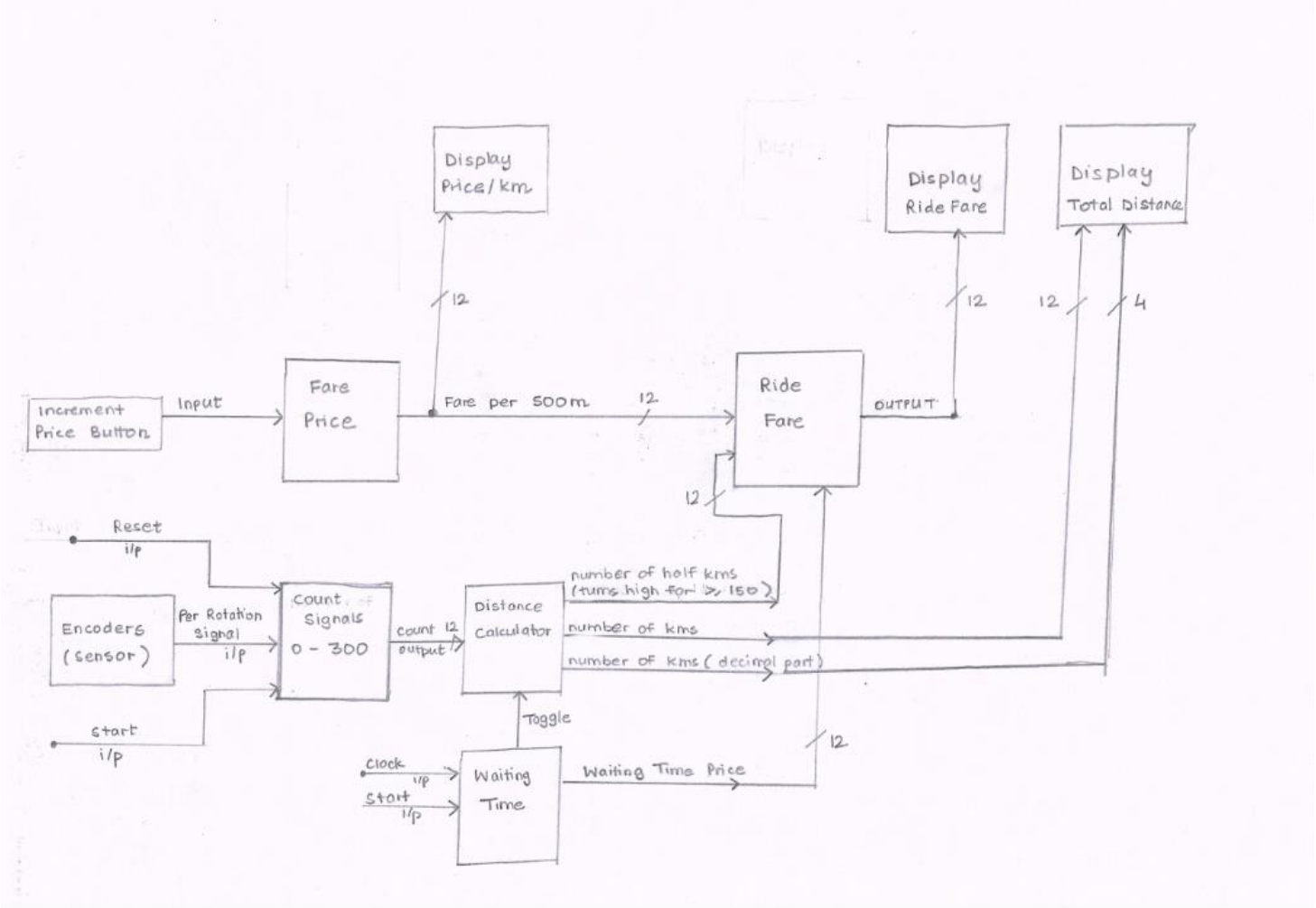
PATIL NIRUPAM ATUL-2019A8PS0566G .

SVADHI JAIN- 2019A8PS0573G.

The Problem Statement:

Design a digital taxi fare meter. 300 full turns of the wheel equal a 500m and a fare change. The driver can update the fare/km is multiples of Rs.10 from Rs 10 to Rs 50. The final fare will be calculated for a distance rounded up to the nearest 0.5km.

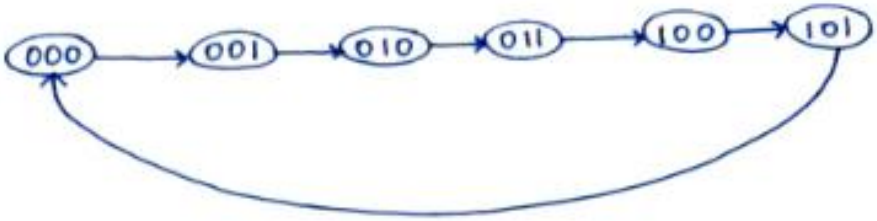
Top-Level Diagram:



State Diagram:

State diagram :

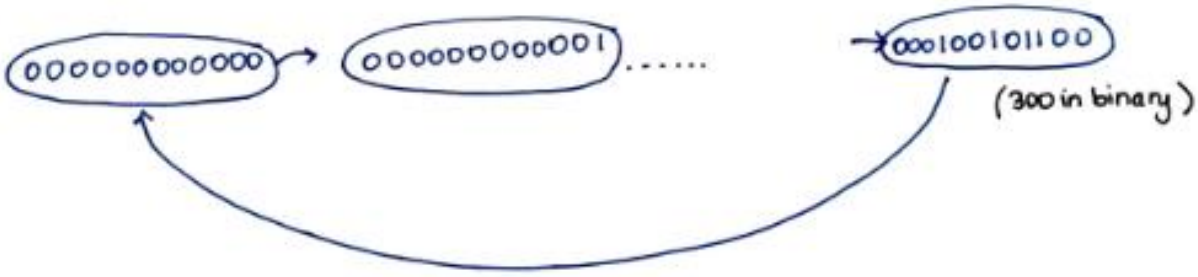
To count from 0 to 5 and again depending on input from user via a button.
This counter is used for fare per km.



Since we have used 12 bits in our circuit, the other bits are all zeroes.

[000 → 000000000 000]

This counter is used to count from 0 to 300 and then again from 0.
Output of this goes to distance calculator.



Assumptions:

1. The circuit is made using a maximum limit of 12 bit output. Therefore, to have the final fare displayed, that is to have a maximum ride fare of Rs. 4096 without error (excluding the waiting time price) the limits of the system are:

FARE PER KM	MAXIMUM DISTANCE
Rs 10	409.5 kms
Rs 20	204.5 kms
Rs 30	136.5 kms
Rs 40	102.0 kms
Rs 50	81.5 kms

2. The circuit gets a High pulse for every rotation of the wheel from a wheel encoder for all four wheels, for the sake of simulation on Logisim we have used a clock to substitute the signals from the sensors.

3. We have used custom multipliers and dividers in the pin-out diagram. We can make multipliers using adders and shift registers and dividers using subtractors and shift registers.

Additional Functionality and Methods

1. We have added a logic block to calculate the waiting time of the taxi i.e when the driver presses the wait button then the waiting circuit starts and for every passing clock pulse, a fixed price is multiplied to calculate the total waiting price which then gets added to the distance fare.

   We have considered Rs 1 as the waiting time cost for each clock pulse. The clock pulse and the waiting time cost per clock pulse can be adjusted.

   For the sake of simulation on Logisim, we have set the number of ticks of the clock of the waiting time clock to 1000 ticks.

   Since the waiting time element only makes sense when the taxi is not moving, at a particular instant of time either the distance fare or the waiting time fare is active in the circuit. To implement this in our circuit we have a toggle switch between these 2 modules.

2. To display the total fare and total distance covered, we have used the double dabble method to convert a 12-bit binary to decimal and display it using the hex display component.

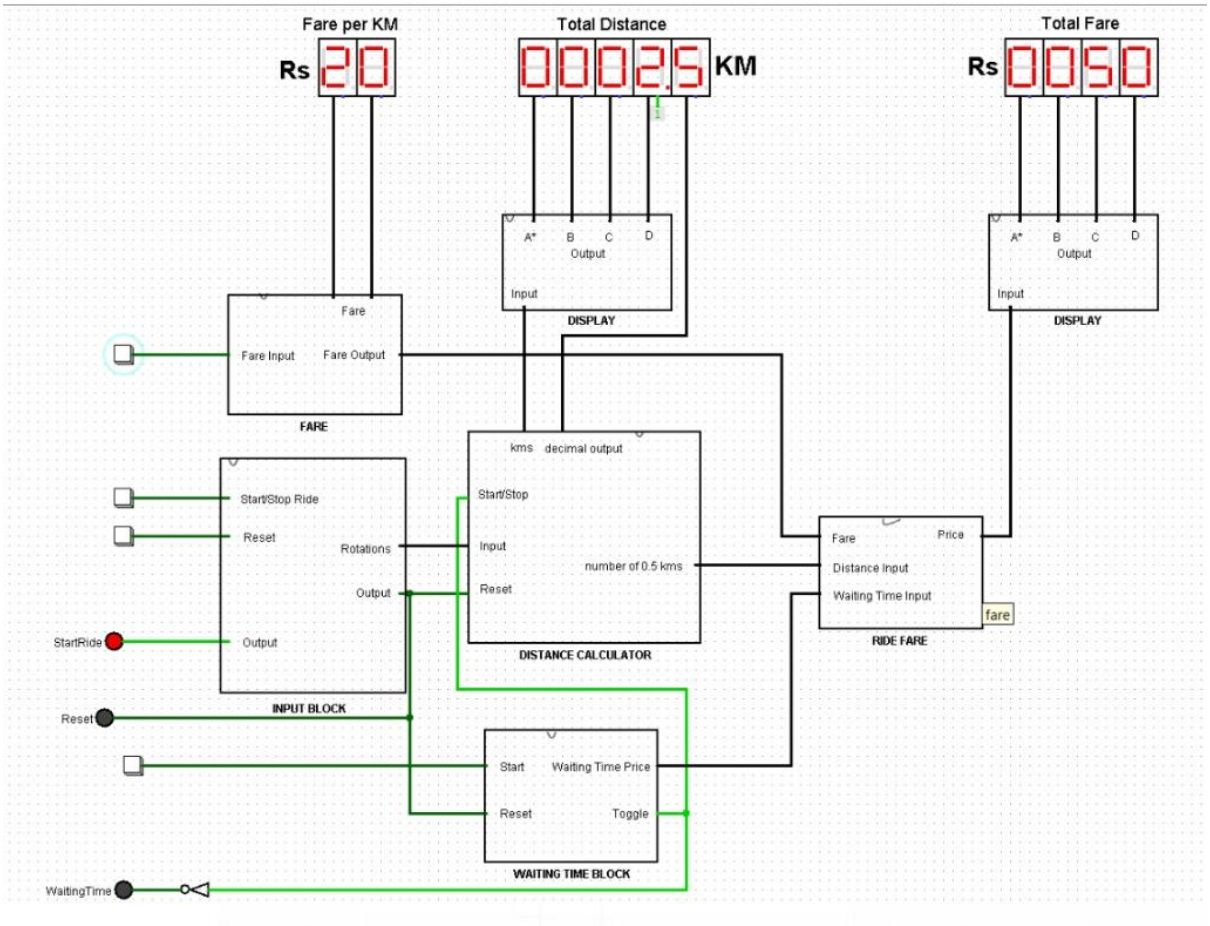
   The double dabble algorithm can be extended to any bit. Thus, the circuit can be easily extended to more bits provided the availability of corresponding ICs.

3. Circuit is independent of magnitude of frequency of the clock i.e., we did not require to assume a particular speed of the taxi for the circuit to works it works for all speed

4. For calculation fare instead of continuous addition method we are directly multiplying the total kms with the fare price, so any delay in the arrival of the total km value will not affect the accuracy of the circuit.

5. For dividing by 2 to get the decimal part and whole part of kms travelled separately in the distance calculator, we have used a divider instead of a bit shift. This allows us to reduce noise as well as extend the circuit if needed for a system with different configurations. (eg. 0.25 kms for 300 rotations)

Main Circuit (Logisim)



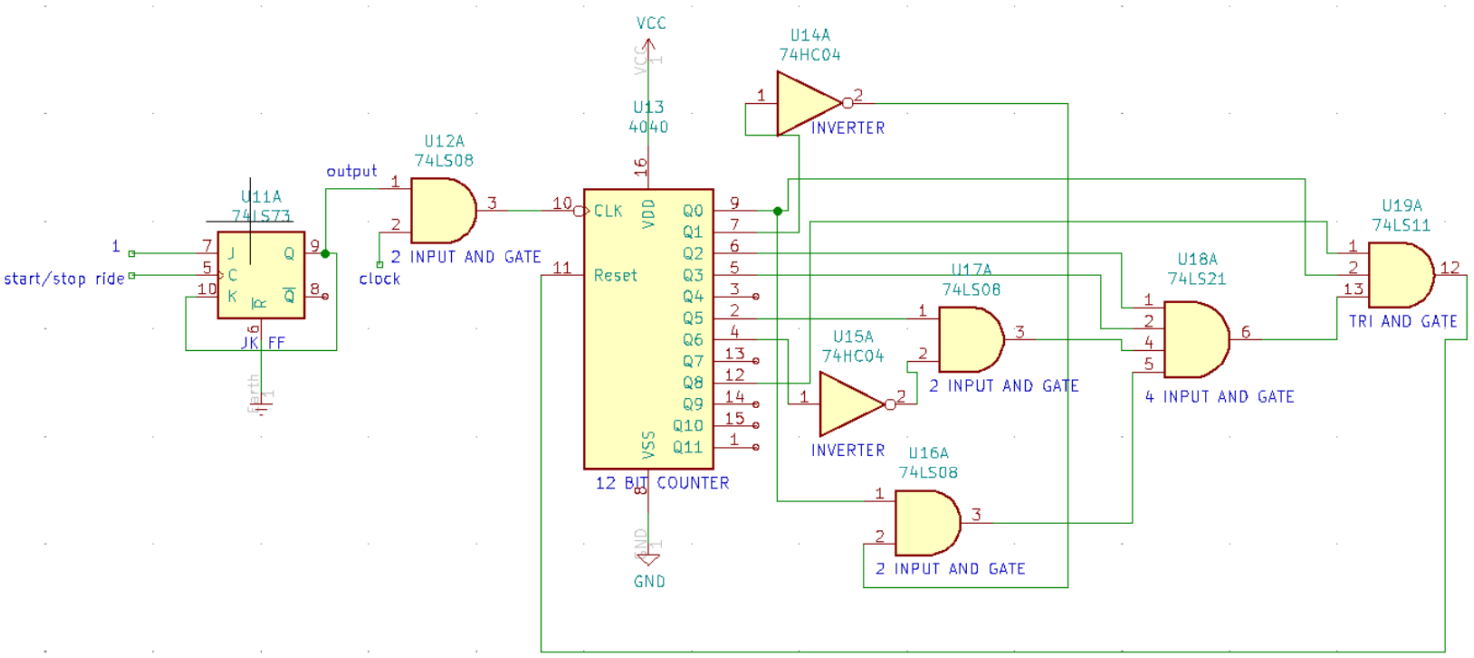
SIMULATION

In order to show the simulation of the circuit in an efficient and understandable way we have included a video in the zip file. The video contains the simulation and demonstration of different parts of the circuit.

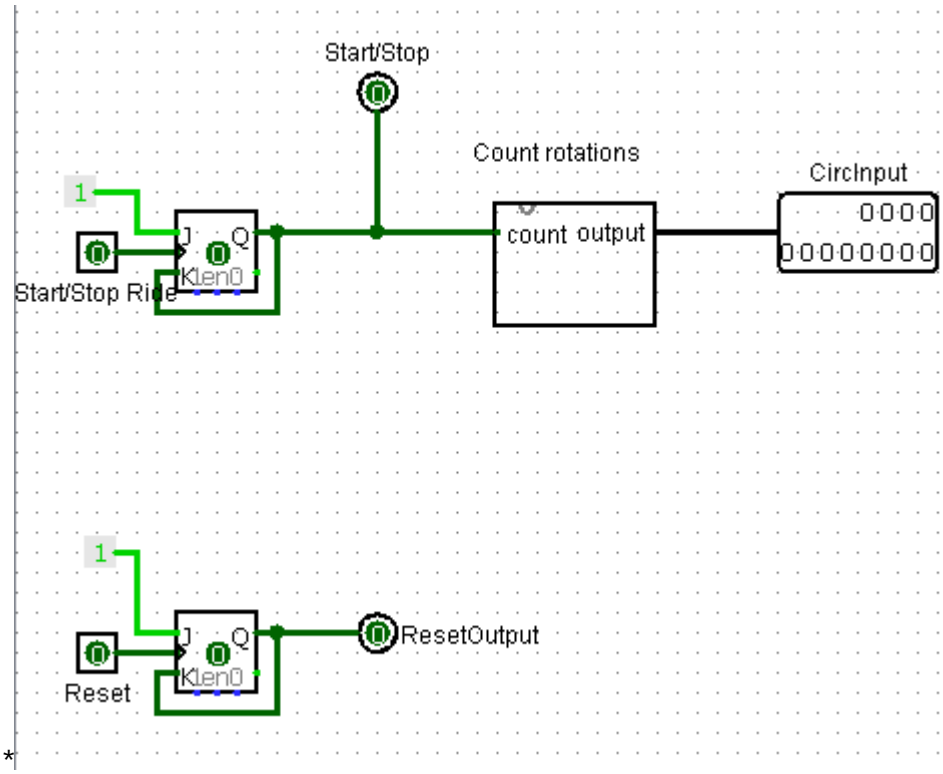
Input block:

MOD 300 counter is made to count up to 300 turns and pass it to the next counter. This counter is made by using 12 stage ripple binary counters, reset and AND and OR logic gates. We have also used a JK Flip Flop to store the state of the start/stop button with each click, thus this is then used to give start/ stop signal to the counter.

Pinout:



Logisim:

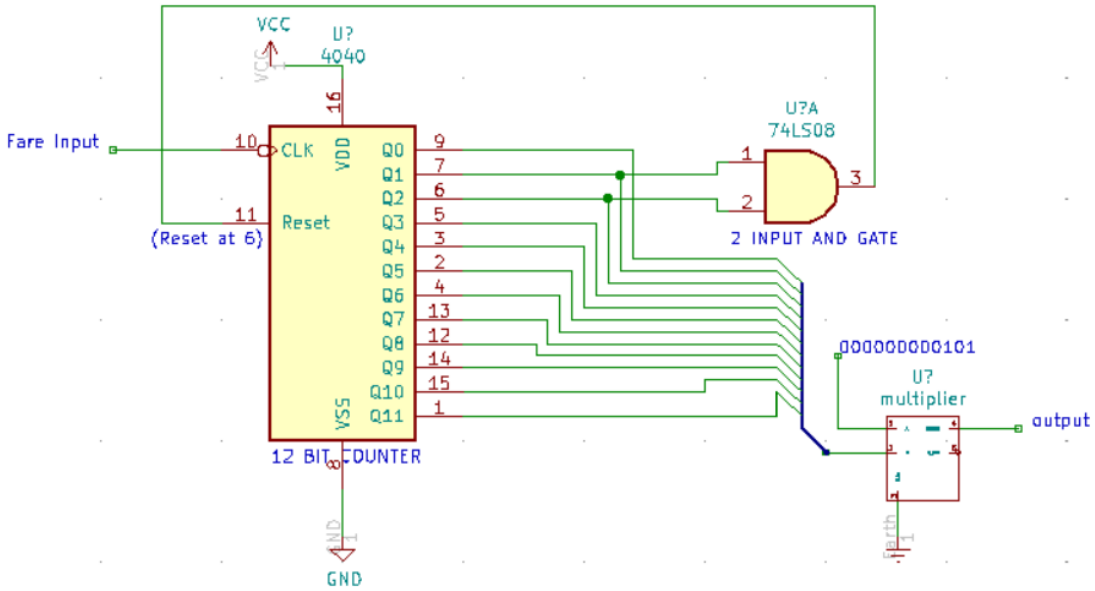


1. **<u>start /stop ride input</u>** pin is used to start or stop the circuit.
2. The **<u>start/stop output</u>** pin shows whether or not the circuit is running at the moment via a LED output.
3. The **<u>count rotations</u>** counter is used to count the number of rotations according to which the distance travelled and the fare is calculated. 300 turns represent 0.5km.
4. The **<u>circinput</u>** is the output of the counter mentioned above is given as an input to the rest of the circuit and is required for the calculation of the distance and fare.
5. The **<u>second module</u>** is used to take input to reset the circuit.

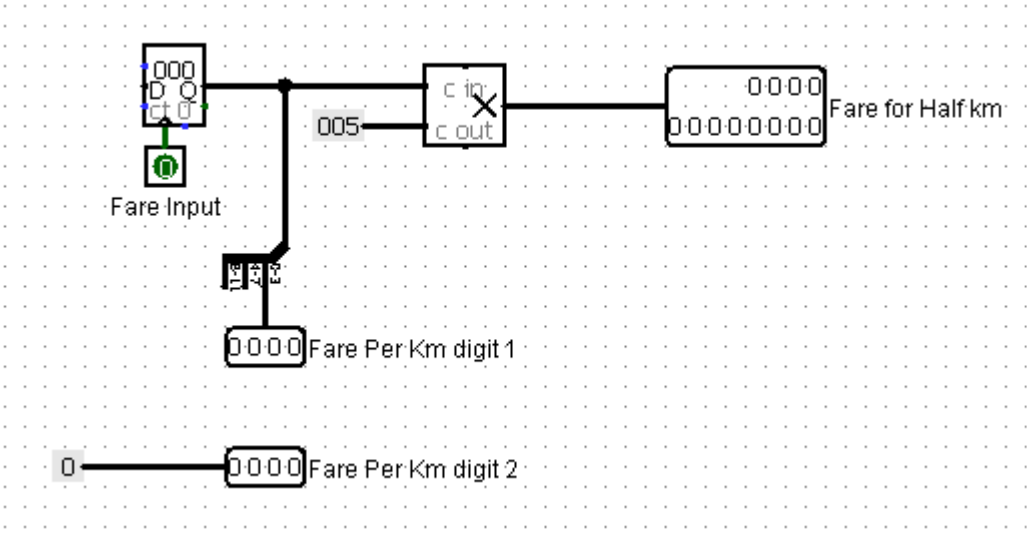
Fare Input Block:

This block takes the fare input via a button in multiples of 10 i.e. from Rs 10 to Rs 50 and passes the output to the display block and the fare calculator. Rs 0 is the default reset state.

Pinout:



Logisim:

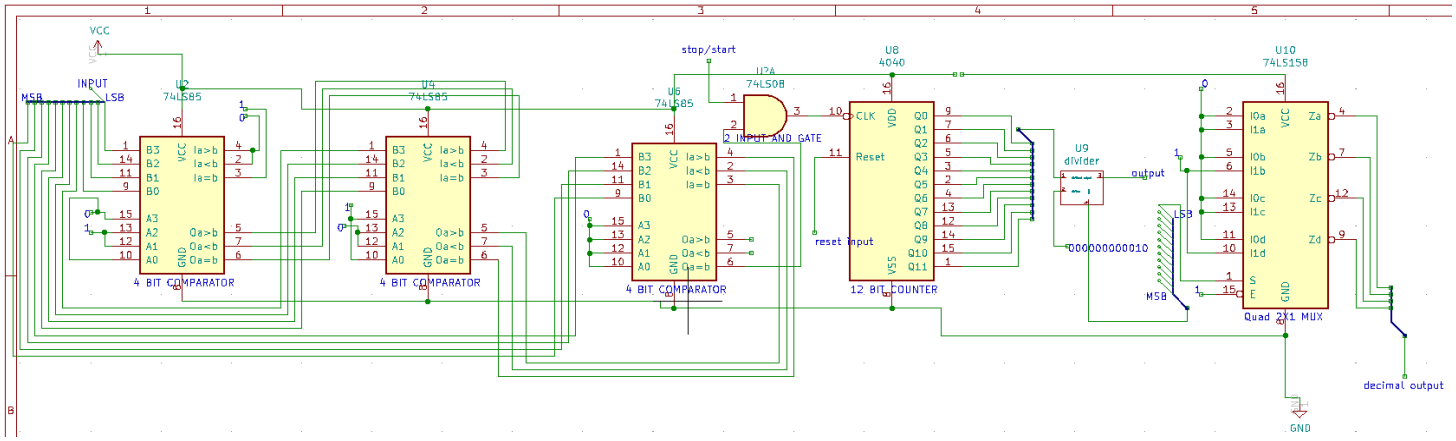


1. The **first counter** is used to take the fare input. Its values range from 0 to 5.
2. The second digit of fare is always zero.
3. The **fare per km digits** is used to display the rate per km.
4. Since we are using half-kms to calculate the total fare instead of multiplying by 10 we multiply by 5 to find the fare per half-km.

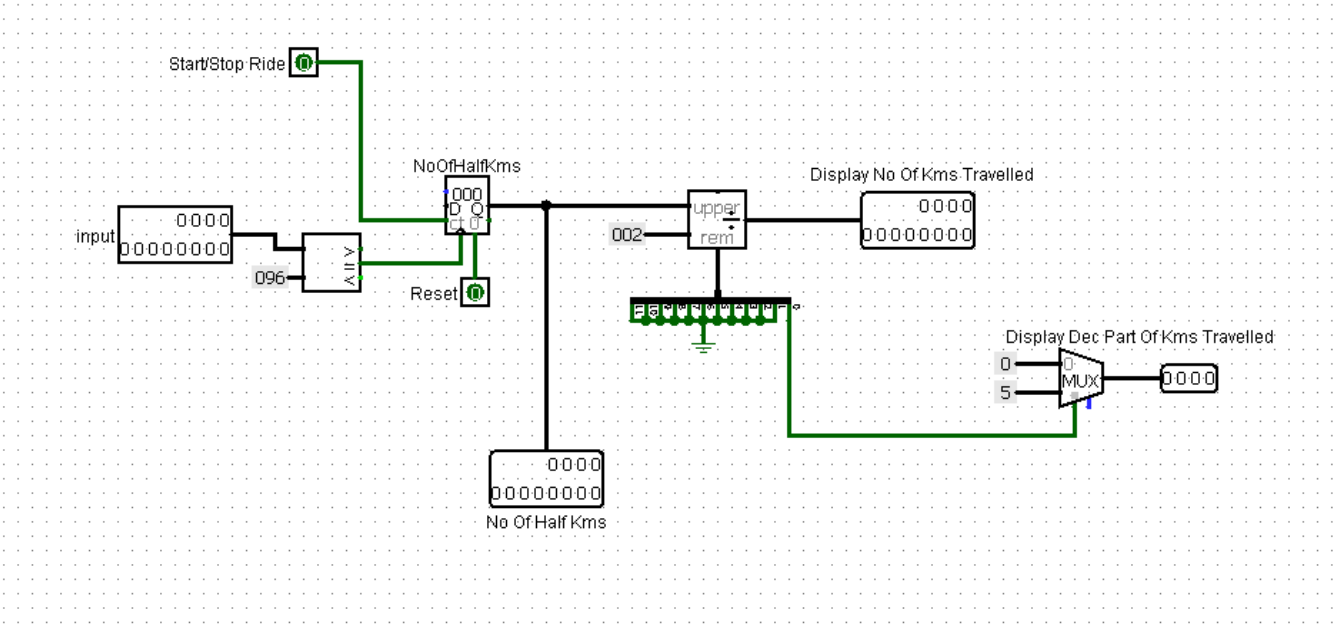
Distance Calculator Block:

This block gets input from the input block. Every time the input reaches 150, we count it as a half km. We then pass the total number of half kms to the fare calculator block. We also divide the number of half kms to get outputs in terms of kms and metres travelled to pass it to the display block. To make a 12-Bit comparator we have cascaded 3 4-Bit comparators (74LS85) using cascade inputs. We have implemented the count enable of the 12-Bit counter using an AND gate. To implement the decimal part of the total KMs we have used a Quad 2:1 MUX which gives output 5 (corresponding to 0.5 kms) when the total number of half kms are odd and gives 0 if the total number of half KMs are even.

Pinout:



Logisim:

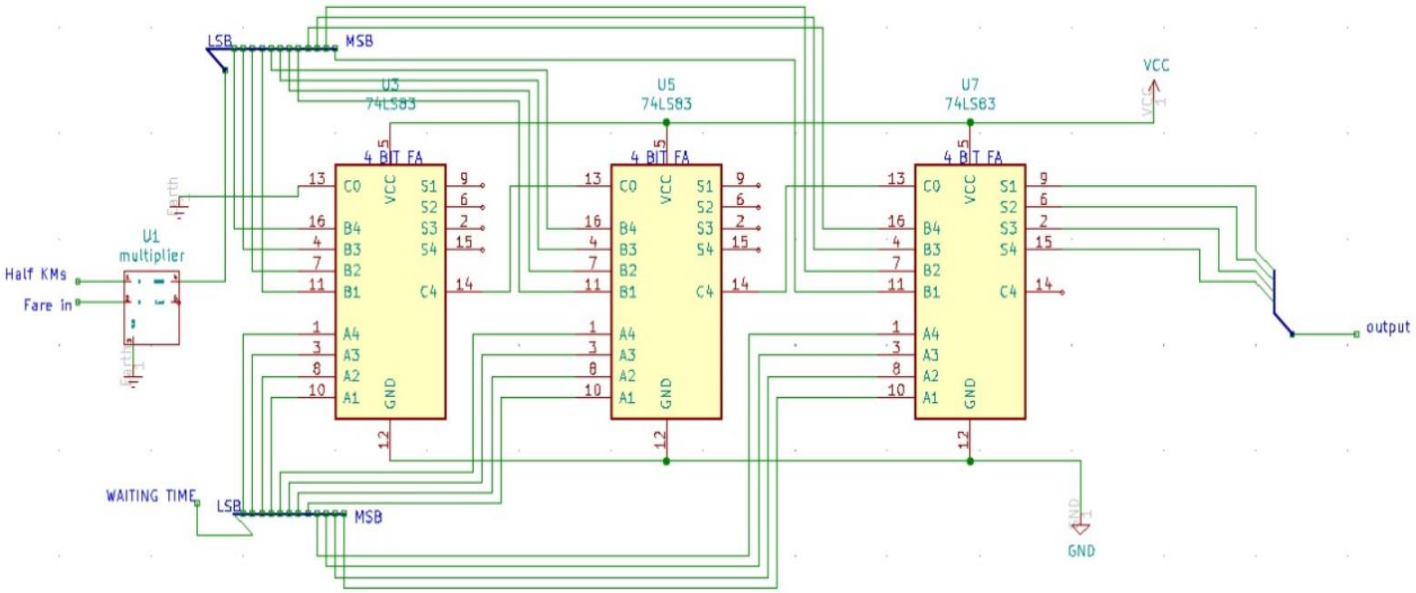


1. The **Start/Stop Ride** button is used to start or stop the meter. When the meter is stopped, regardless of the inputs, the outputs are not affected and do not change.
2. The **Input** is the number of turns of the meter which are taken as an input for the meter. It resets after every 300 turns.
3. The **No of Half Kms** counter counts the number of half kilometres that have been travelled. According to the problem, 300 turns have been assumed to represent 0.5km. So, we use the **comparator** to compare the number of turns. Whenever the number of turns in the input are equal to 150 the number of half kms is incremented by 1 as, 150 turns=0.25 km.
4. The **Reset** button is used to reset the meter at the end of a journey i.e., all the outputs and inputs become 0.
5. The **Number of Half Kms** display displays the number of half kilometres that have been travelled by the taxi.
6. **Display Number of Kms Travelled** displays the number of kilometres that have been travelled by the taxi by dividing the number of half-kilometres by 2 [half-kms=2*kms]. It displays the quotient after dividing the number of half-kms by 2.
7. **Display decimal part of kms travelled** part displays the part after the decimal point of the distance travelled. Since the part after the decimal point can only be a 0 (for an even number) or a 5 (for an odd number), a multiplexer which uses the remainder (0 or 1) of the divider as an input is used to select the decimal part from 0 and 5.

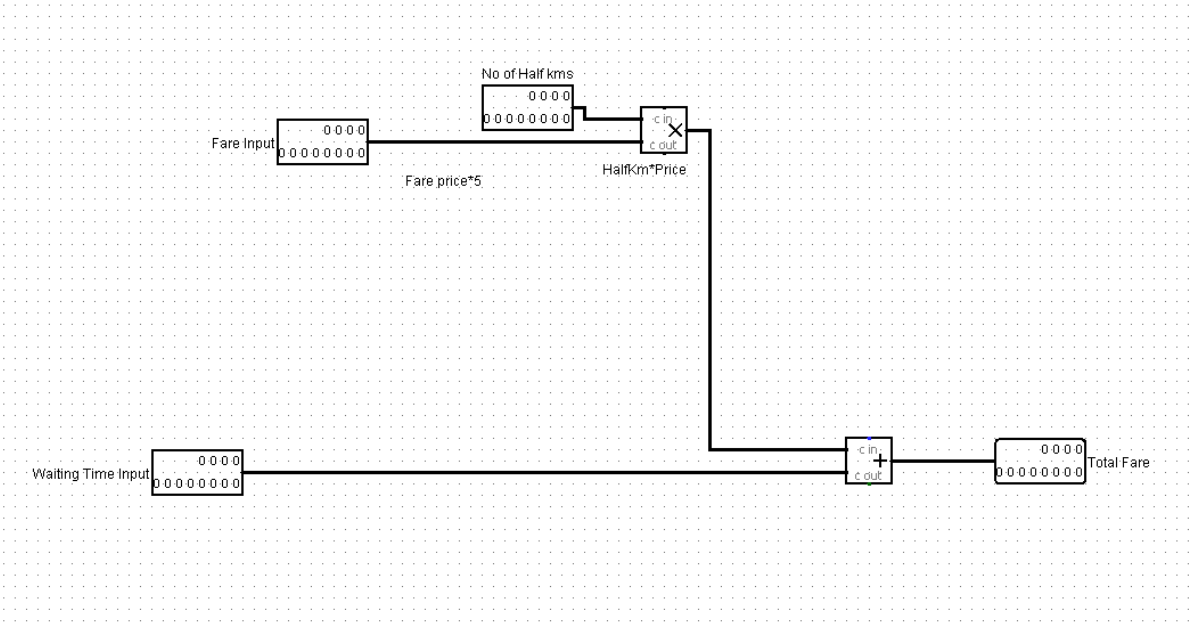
Fare Calculator Block:

Total Fare is calculated by multiplying the total number of half KMs with fare per half KMs using a multiplier. To add the waiting time fare to this fare we have used 3 4-BIT adders in cascade to add 12-bit binary numbers. To cascade the adders we have connected the Cout of one adder to Cin of another.

Pinout:



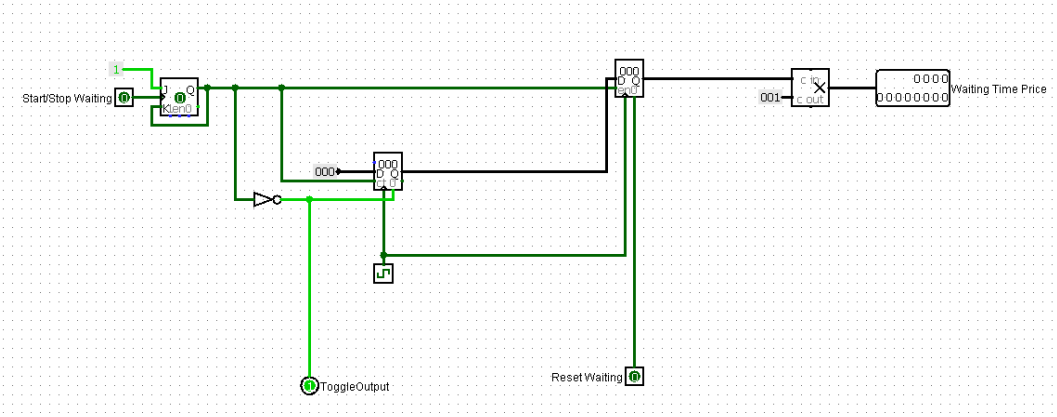
Logisim:



1. **Fare input** block is the input from the fare block meter. (Fare per half a km)
2. **No of half-kms** takes the number of half-kms as input from the distance block.
3. **Half-km*Price** multiplier multiplies the price by the number of half kilometres travelled.
4. **Waiting time input** adds the additional fare if the taxi is made to wait or has to wait due to unavoidable circumstances. (this is calculated by another block)
5. The **Adder** adds the fare due to distance travelled and the fare due to the waiting time.
6. The **Total Fare** is the output final fare after adding the respective components and is passed to the display block.

Waiting Time:

Logisim:

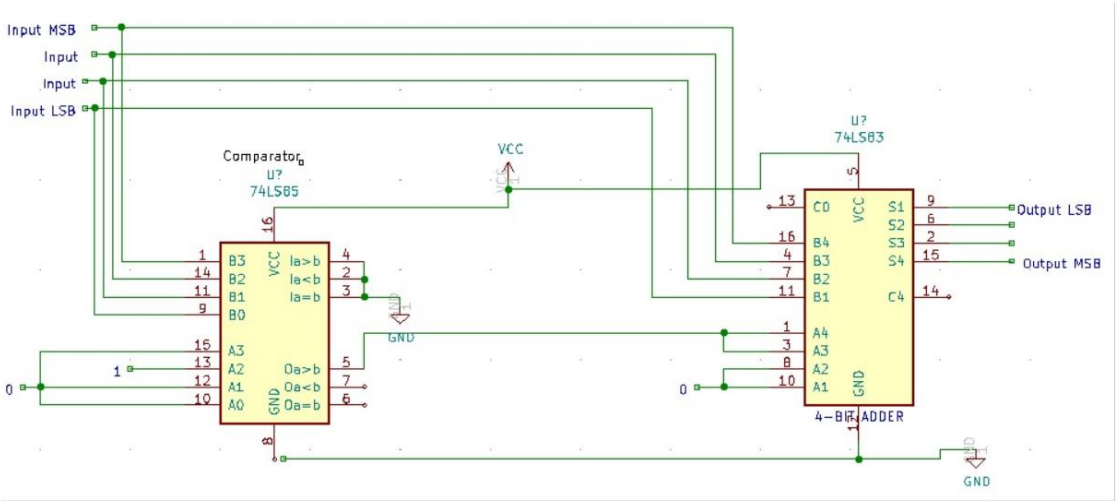


1. **<u>Start/Stop waiting</u>** is used to start or stop the waiting fare mechanism. This button helps in synchronising the waiting time module with the distance calculator as only one of the modules (ie the waiting calculator or the distance calculator) can be active at a time.
2. The **<u>counter connected to the start stop button</u>** is used to turn the circuit on or off.
3. The **<u>second counter</u>** is used to count the duration of time for which the taxi has been waiting.
4. The **<u>register which is connected to the waiting time price output</u>** is used to store the total duration for which the taxi has been waiting.
5. The **<u>multiplier</u>** multiplies the waiting fare with the waiting time to give the final waiting fare. (we have assumed the waiting fare to be Rs.1)
6. The **<u>waiting time price output</u>** is used to display the additional fare charged due to the waiting time and this is then passed to the fare calculator.

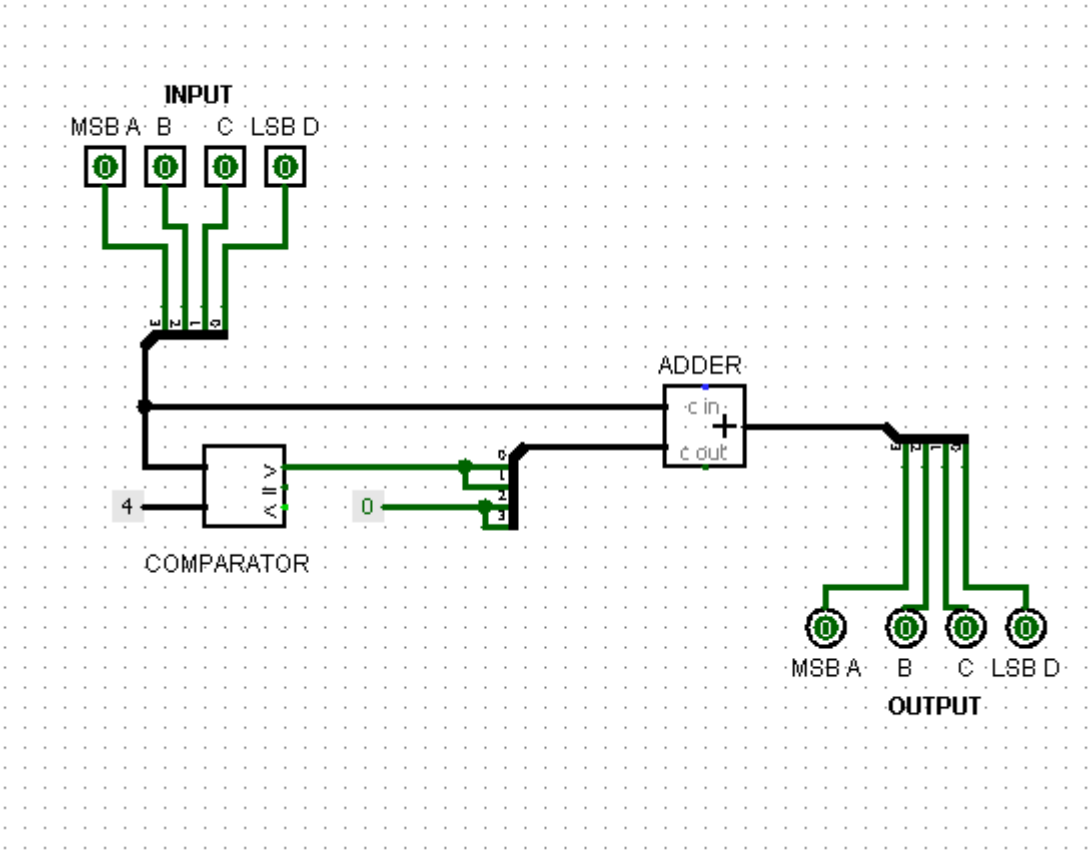
Add 3:

This method checks whether the binary input given to it is greater than 4 via a 4 bit comparator and if the input is greater than 4 it adds 3 to the input using a 4 bit-adder.

Pinout:



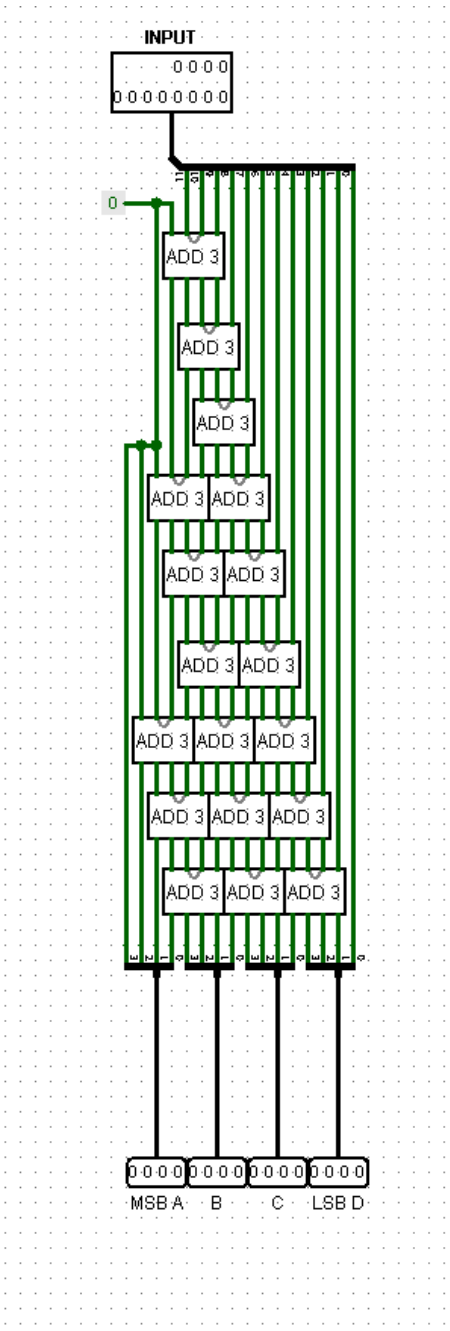
Logisim:



1. This method checks whether the binary input given to it is greater than 4, and if the input is greater than 4 it adds 3 to the input. This is a repeating unit of the double dabble method

Binary To Decimal(Double Dabble):

This takes in a binary number as an input and using multiple connected add3 methods, it converts the binary input to a decimal output and hence can be passed to hex displays. We have used this to display the total distance and total ride fare.



Bill Of Material:

Below is the list of integrated circuits that we used to design the circuit. First column contains name of the IC, followed by its functionality and finally number of such ICs used in the circuit

IC	Description	Quantity
74LS85	4-Bit comparator	4
74LS83	4-Bit Adder	3
74HC4040	12-Stage Binary Ripple Counter	2
74LS158	Quad 2 to 1 multiplexer	1
74LS08	QUADRUPLE 2-INPUT AND GATE	4
74LS21	Dual 4-input AND	1
74LS11	Triple 3-input AND	1
74HC04	Inverter	2
74LS73	Dual JK Flip-Flop, reset	1

Appendix:

IC	DESCRIPTION	DATASHEET
74LS85	4-Bit comparator	https://www.alldatasheet.com/datasheet-pdf/pdf/12664/ONSEMI/74LS85.html
74LS83	4-Bit Adder	https://www.alldatasheet.com/datasheet-pdf/pdf/5744/MOTOROLA/74LS83.html
74HC4040	12-Stage Binary Ripple Counter	https://www.alldatasheet.com/datasheet-pdf/pdf/15607/PHILIPS/74HC4040.html
74LS158	Quad 2 to 1 multiplexer	https://datasheetspdf.com/pdf-file/817764/Motorola/74LS158/1
74LS08	QUADRUPLE 2 INPUT AND GATE	https://www.alldatasheet.com/datasheet-pdf/pdf/12619/ONSEMI/74LS08.html
74LS21	Dual 4-Input AND	https://www.alldatasheet.com/datasheet-pdf/pdf/64020/HITACHI/74LS21.html
74LS11	Triple 3-Input AND	https://www.alldatasheet.com/datasheet-pdf/pdf/64020/HITACHI/74LS21.html
74HC04	Inverter	https://pdf1.alldatasheet.com/datasheet-pdf/view/15523/PHILIPS/74HC04.html
74LS73	Dual JK Flip Flop, reset	https://pdf1.alldatasheet.com/datasheet-pdf/view/5740/MOTOROLA/74LS73.html

Note: If any of the above Pinout diagrams are not visible properly in the report. Please refer to the Pinout pdf in the folder as it contains a much clear picture of all the Pinout Diagrams