Project Based on Automated Speed Controller

- **Objective : To control the speed of a car using distance sensor data.**

Submitted by:

Murchana Saikia

Roll No- 170810026025

Jorhat Institute of Science and Technology

Abstract:-

The objective of this project is to simulate an Automated Speed Controller. Nowadays in a fast moving world everyone is concerned about their time. People are driving vehicles in a high speed which results in frequent accidents and consequently the loss of lives and property. So in order to avoid such kind of accidents this project aims mainly at automatically controlling the speed and brakes of a vehicle which not only will prevent accidents from happening and saving lives but also minimize the loss of property. This automated speed controller uses two distance sensor as inputs, far away distance and nearby distance sensors. Whenever any obstacle is detected in running vehicle, the speed and brakes of the vehicle is automatically controlled.

The distance sensors continuously sends signals and monitors any obstacle that are in front of the car. When any obstacle or vehicle is detected by the distance sensor, it will send signals to the controller unit as inputs.

After receiving the input data, the controller unit will decide whether to control the speed or apply brake.

Introduction:-

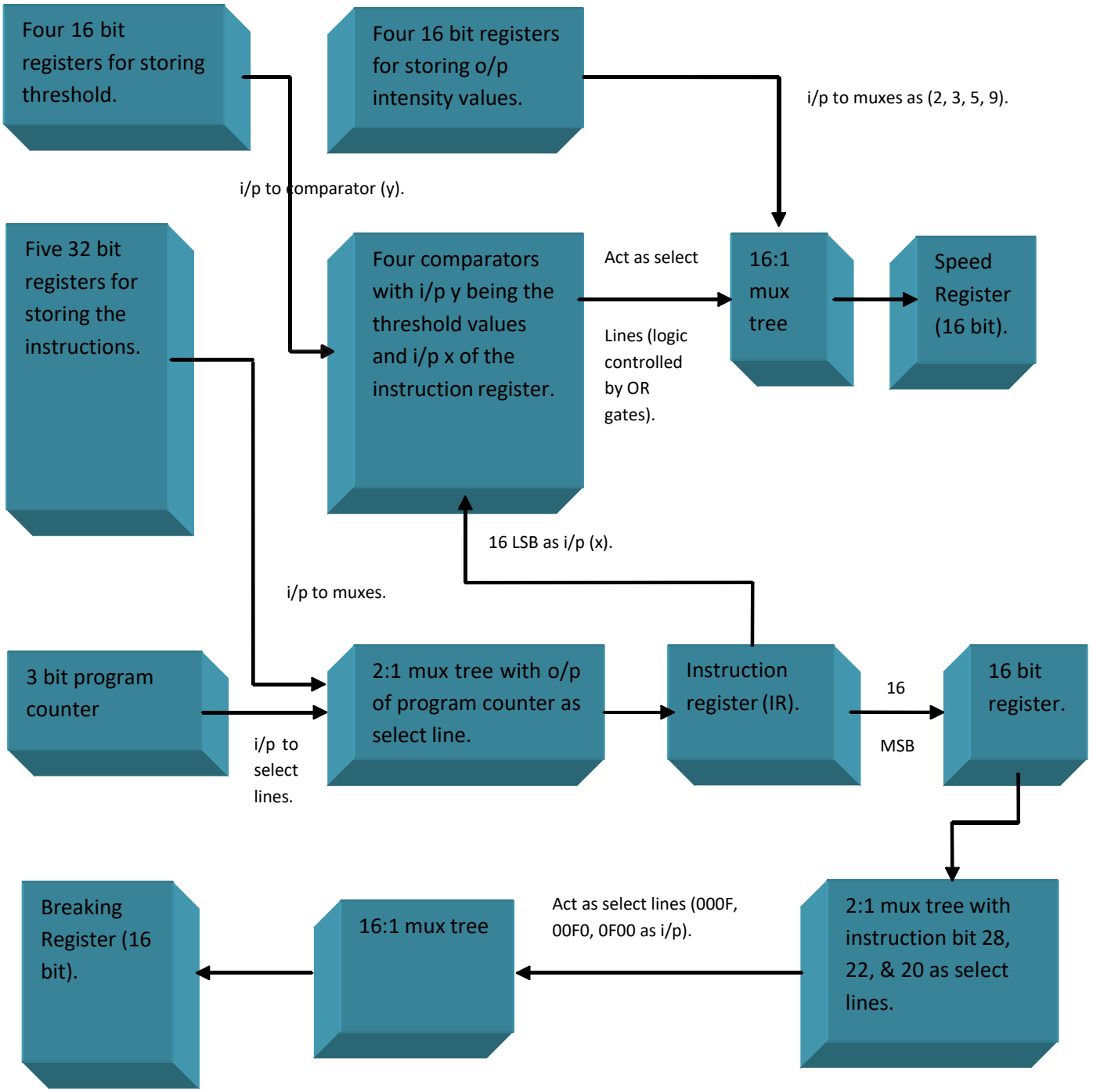
As the number of vehicles on the road are increasing day by day so is the number of deadly accidents. A number of factors contribute to the risks of collisions including vehicle designs, driving skills, road design, collisions due to obstacle, speeding and many more. For preventions of these accidents the government made some rules such as wearing helmet, seat belt compulsion, etc. But these rules are not enough to avoid an accident and moreover least number of people obey these rules. As a result automation of driving control system has become a need to avoid such hazards.

Approximately 1.35 million people meet with their death every year due to road accidents. Around 20 to 50 million people suffer non-fatal injuries and many with permanent damages resulting in disabilities. In 2013, 54 million people worldwide accounted collisions resulting in a death of 1.4 million from 1.1 million deaths in 1990.

In 2017, India alone officially reported 464,910 road accidents, of which 147,913 death and 470,975 injured, which results in 405 deaths and 1,290 injuries each day.

Only 10 percent of new vehicles include automatic braking as standard and 50 percent offer it as an option worldwide. The U.S Department of Transportation plans to require standard automatic braking systems on all vehicles by 2025. However twenty automakers accounting for U.S vehicle sales have voluntarily committed to including the standard feature by 2022.

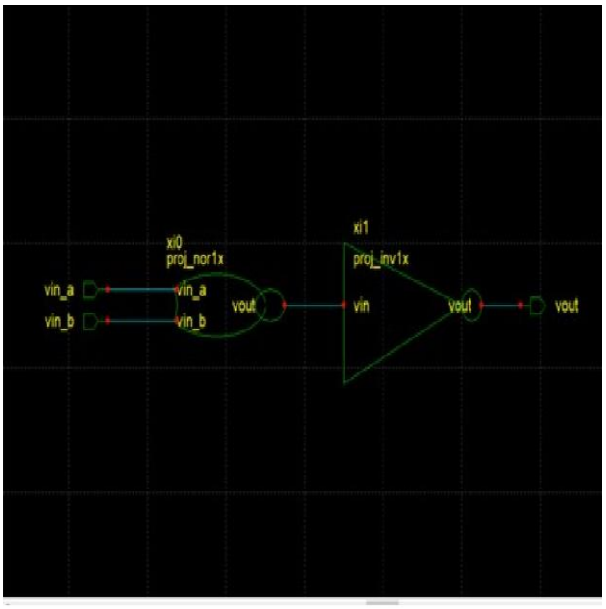
Block Diagram Of Design:-

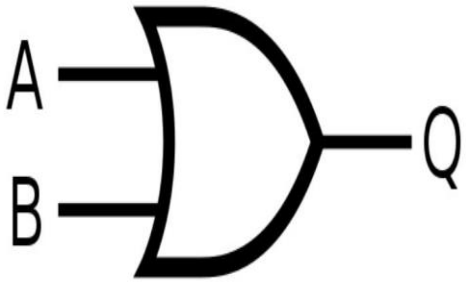


Individual Components Schematics With Explanation:-

OR GATE:-

The OR gate is a digital logic gate that implements logical disjunction – it behaves according to the adjacent truth table. A HIGH output (1) results if one or both the inputs to the gate are HIGH (1). If neither input is high, a LOW output (0) results.





INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

REGISTER:-

Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a Register. The n-bit register will consist of n number of flip-flop and it is capable of storing an n-bit word. The binary data in a register can be moved within the register from one flip-flop to another. The registers that allow such data transfers are called as shift registers. There are four modes of operations of a shift register.

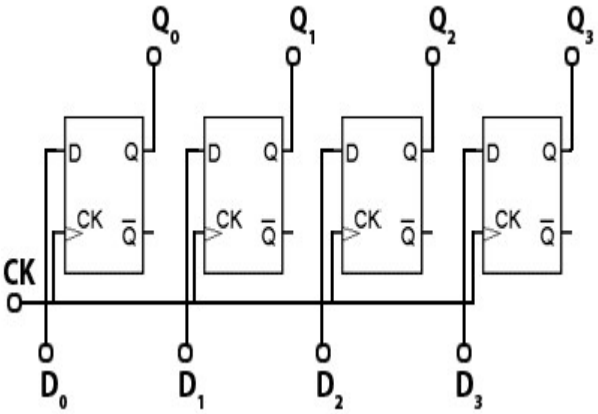
- Serial Input Serial Output
- Serial Input Parallel Output
- Parallel Input Serial Output
- Parallel Input Parallel Output

In our given project we have used parallel input parallel output register. A Parallel in Parallel out (PIPO) shift register is used as a temporary storage device

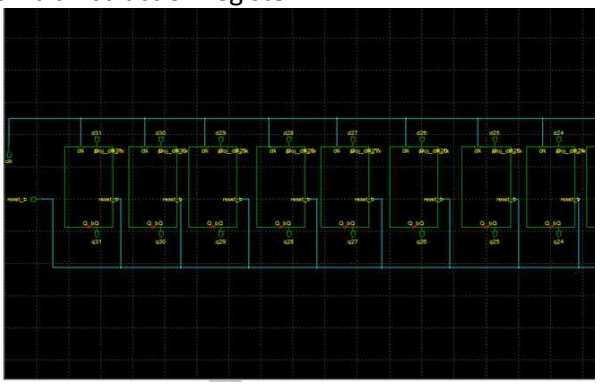
Parallel-In Parallel-Out Shift Register (PIPO) –

The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as Parallel-In parallel-Out shift register.

A simple 4-bit register is illustrated in Figure below and consists of four D type flip flop, sharing a common clock input, providing synchronous operation ensuring all bits are stored at exactly the same time. The binary word to be stored is applied to the four D inputs and is remembered by the flip-flops at the rising edge of the next clock (CK) pulse. The stored data can then be read from the Q outputs at any time, as long as power is maintained, or until a change of data on the D inputs is stored by a further clock pulse, which overwrites the previous data.

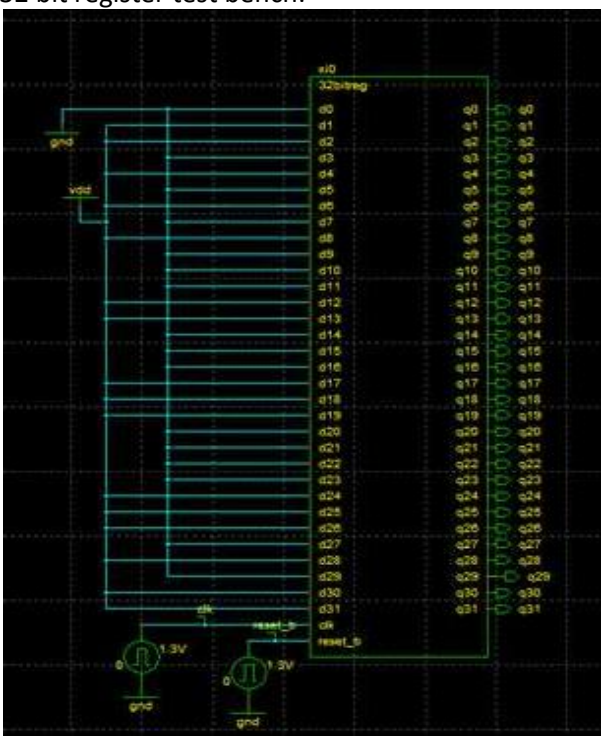


32 bit instruction register:

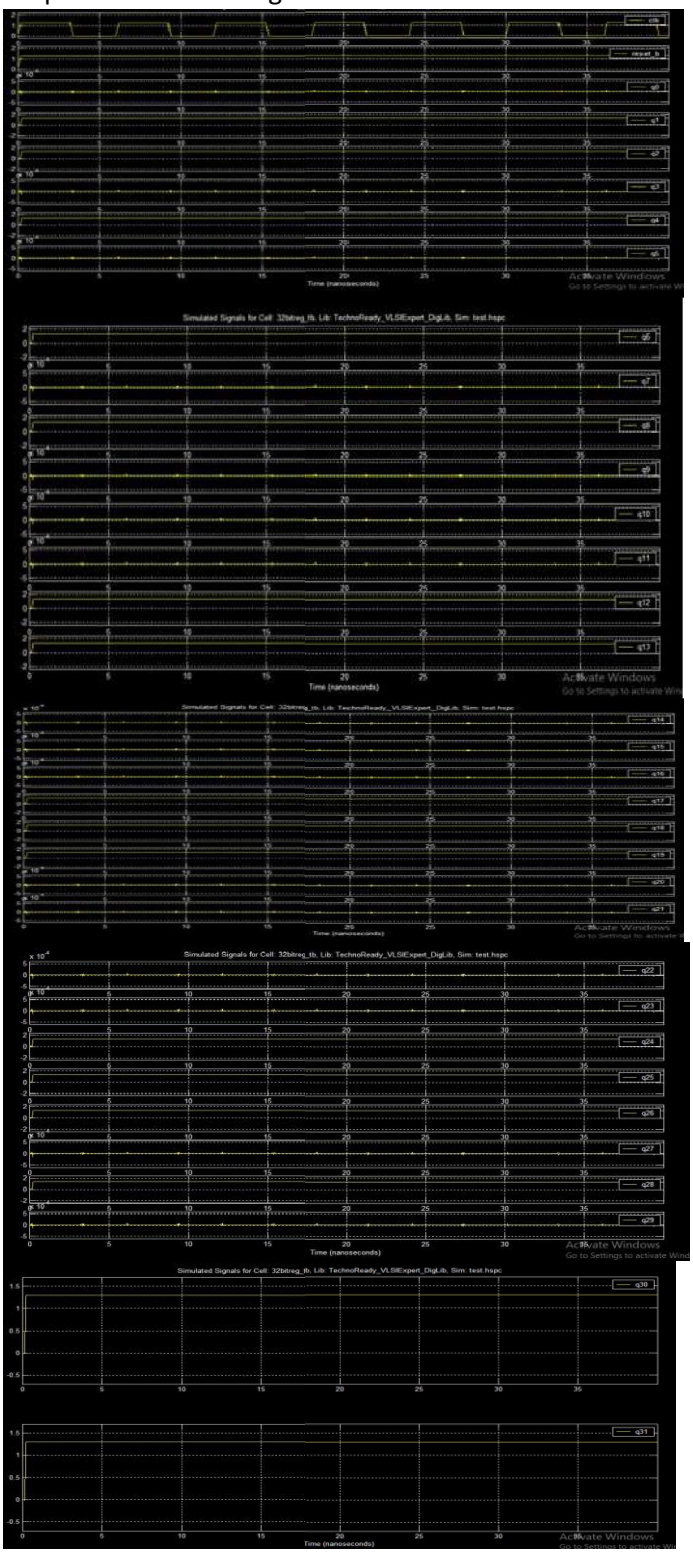


……so on…

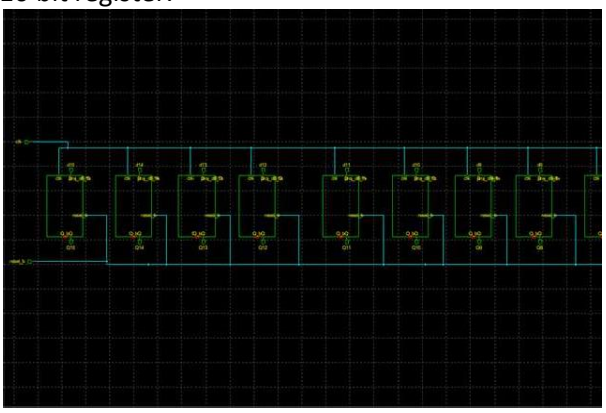
32 bit register test bench:

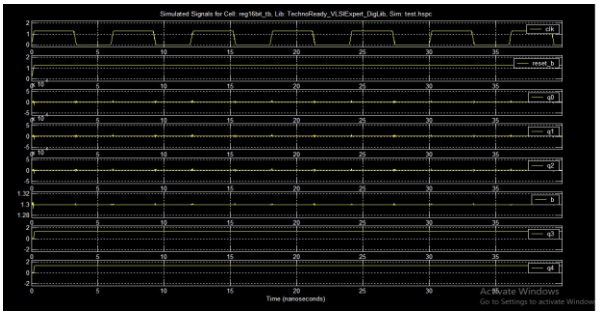


Output of the 32 bit register

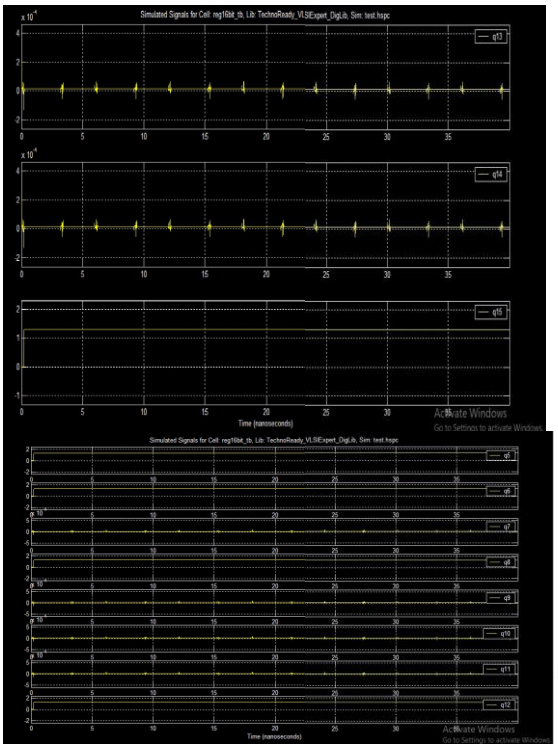


16 bit register:





16 bit register test bench:

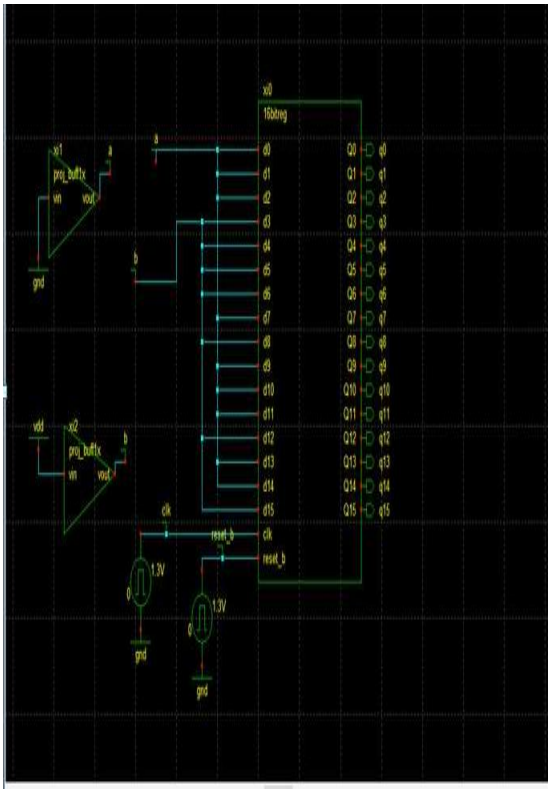


16 bit register output:

MULTIPLEXER:-

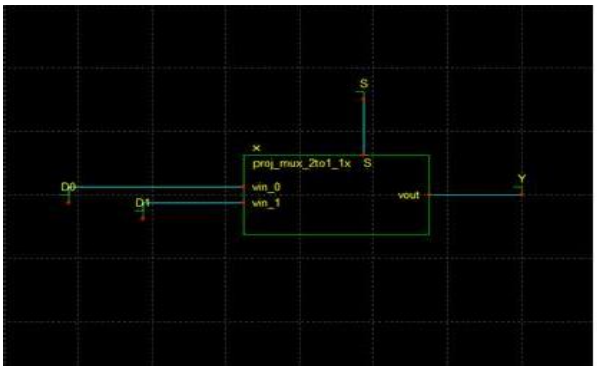
In electronics, a multiplexer also known as a data selector, is a device that selects between several digital input signals and forwards it to a single output line. A multiplexer of 2^n inputs has n select lines, which are used to select which input line to send to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth. Multiplexers can also be used to implement Boolean functions of multiple variables. An electronic multiplexer can be considered as a multiple input multiple output switch.

Generally the number of data inputs to a multiplexer is a power of two such as 2, 4, 8, 16, etc. Some of the mostly used multiplexers include 2-to-1, 4-to-1, 8-to-1 and 16-to-1 multiplexers.

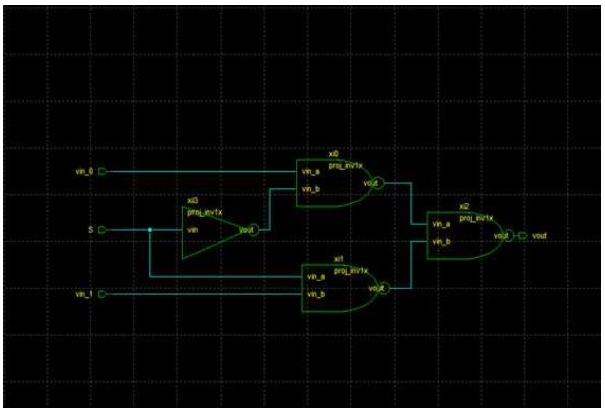


2-to-1 Multiplexer

A 2-to-1 multiplexer consists of two inputs D0 and D1, one select input S and one output Y. Depends on the select signal, the output is connected to either of the inputs. Since there are two input signals only two ways are possible to connect the inputs to the outputs, so one select is needed to do these operations. If the select line is low, then the output will be switched to D0 input, whereas if select line is high, then the output will be switched to D1 input. The figure below shows the block diagram of a 2-to-1 multiplexer which connects two 1-bit inputs to a common destination

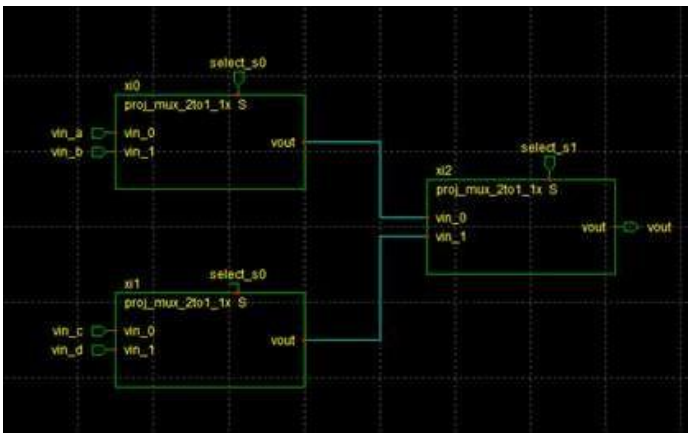


Circuit diagram of 2:1 mux:

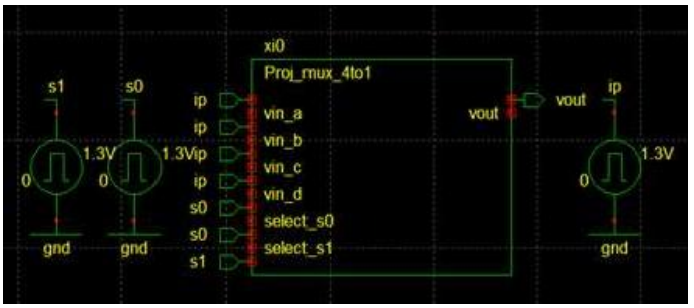


4-to-1 Multiplexer:

A 4x1 mux can be implemented using three 2x1 muxes. 2 of these multiplexers can be used as first stage to mux 2 inputs each with least significant bits of select lines (S0), resulting in 2 intermediate outputs, which, then can be muxed again using a 2:1 mux. The implementation of 4x1 mux using 2x1 muxes is shown.



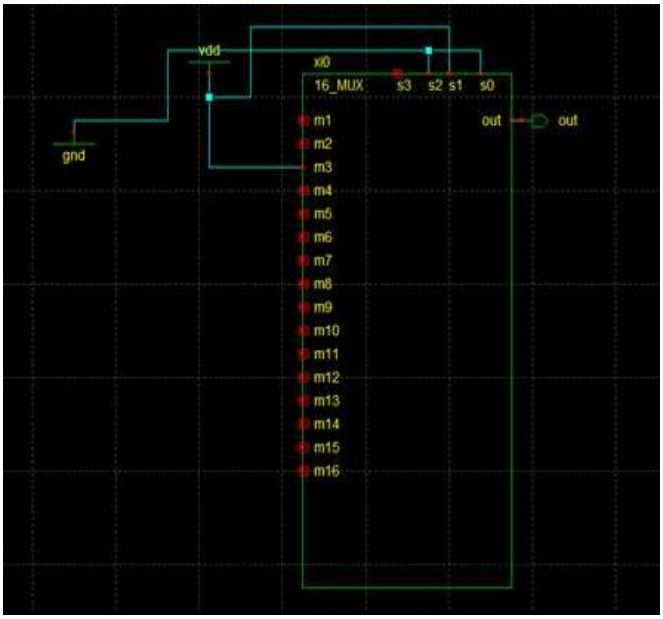
4:1 multiplexer test bench:



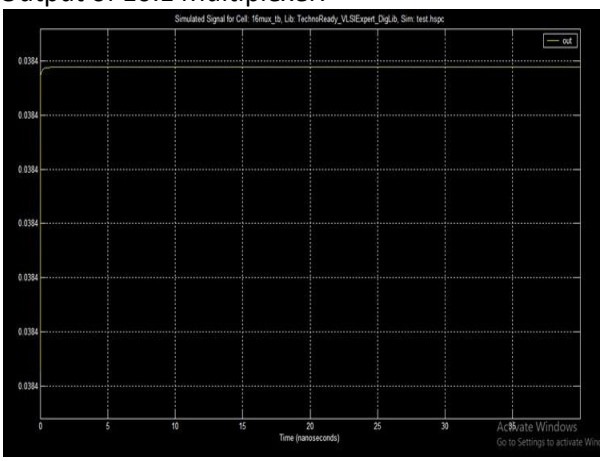
16:1 multiplexer:

A 16x1 mux can be implemented using 5 4x1 muxes. 4 of these multiplexers can be used as first stage to mux 4 inputs each with two least significant bits of select lines (S0 and S1), resulting in 4 intermediate outputs, which, then can be muxed again using a 4:1 mux. The implementation of 16x1 mux using 4x1 muxes is shown below.

16:1 multiplexer test bench:



Output of 16:1 Multiplexer:

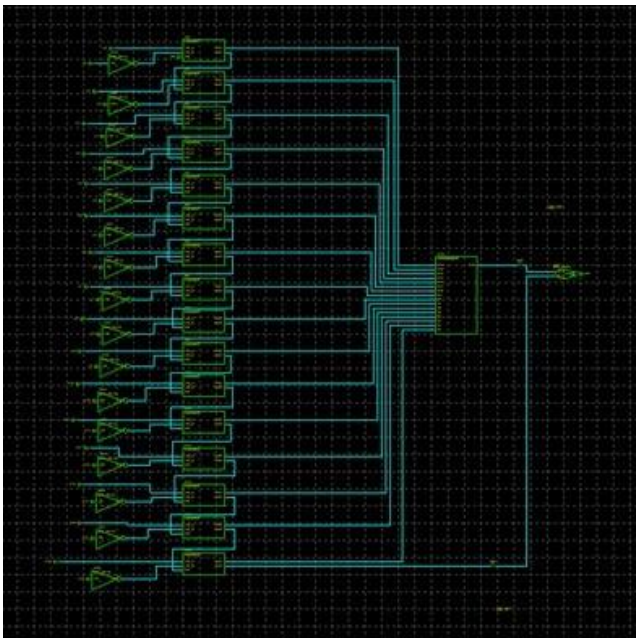


COMPARATOR:-

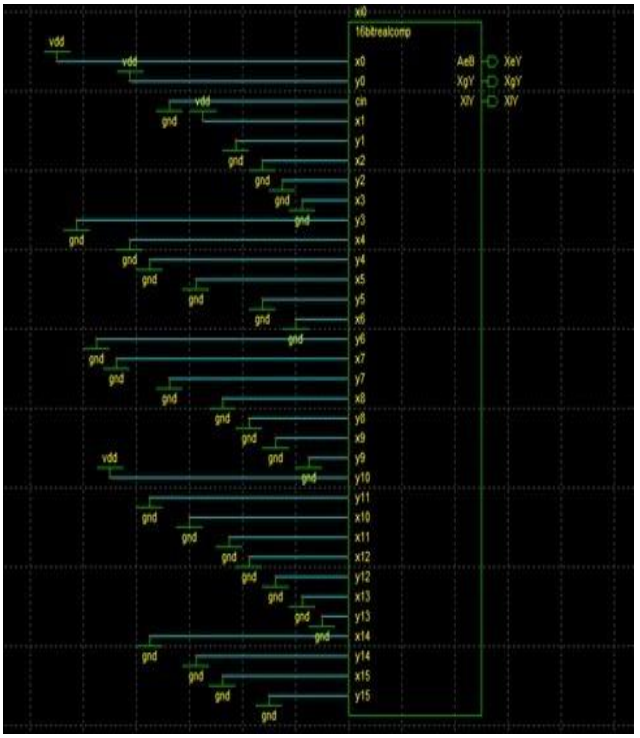
A Binary comparators, also called digital comparators or logic comparators, are combinational logic circuits that are used for testing whether the value represented by one binary word is greater than, less than, or equal to the value represented by another binary word.

16-Bit magnitude comparator using efficient full adder:

The implementation of 16-bit comparator is shown in figure below. To neither implement this we need 16 1-bit efficient Full Adders and one 16-input AND gate and a two input NOR gate. It has inputs (X16, Y16 X15,Y15 and so on up to X1,Y1).and for each Full Adder we have S and C as outputs. The output sum of each Full Adder (i.e S1 toS16) has been given to and gate the output of AND gate determines whether XeY (X equal to Y). Output carry C16 determines whether XgY(X greater than Y) and the two outputs has been given to NOR gate and output of NOR gate determines whether XlY(X less than Y).



16 bit comparator test bench:

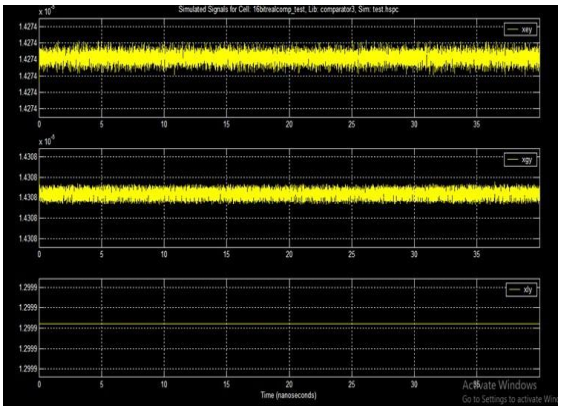


The output of the comparator for the input

X=0000000000000011

Y=0000010000000001

Output=XlY(X less than Y).



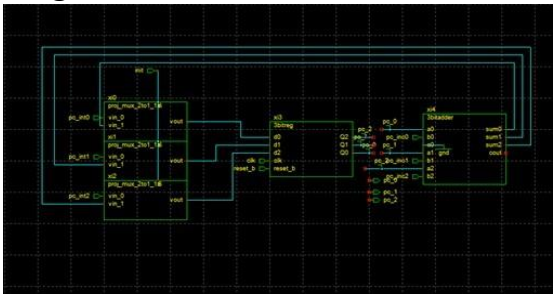
PROGRAM COUNTER:-

A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence.

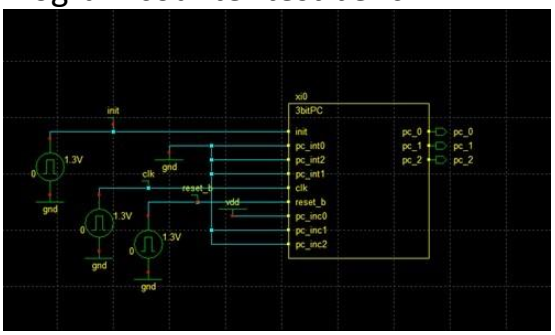
In our project we have used 3-bit program counter consisting of 2:1 mux tree, one 3-bit register and one 3-bit adder.

All the input to the mux's are grounded so when init is 0 then the register output i.e. pc_0 ,pc_1 and pc_2 would be 000.The input to the adder are 001 and pc_0, pc_1, pc_2. So the output sum0 will be high and all the others low. The output of the adder are provided as the second input to the muxes. So when init is high the adder value will be passed i.e. in the next clock cycle the register output would be 001 this 001 would act as a input to the adder and hence it will be added with the 001 and the output would be 010. In this way IC will continue to increase its value by 1 in each fetch cycle (clock cycle).

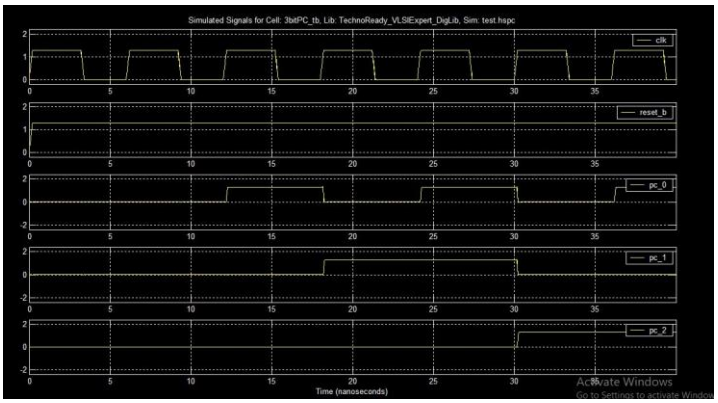
Program Counter circuit:



Program counter test bench:



Program counter output:



Detailed Test Bench Description:-

Working of the circuit:-

We have prepared the circuit to basically simulate an automatic speed controller. The complete circuit diagram is shown on the next page.

For this we used registers, multiplexers, program counter, comparators and the other small components. We were provided with four hexadecimal instructions which will represent the operations such as FAR DISTANCE, NEAR DISTANCE HIGH, NEAR DISTANCE MEDIUM and NEAR DISTANCE LOW. These four instructions act as op-codes. With these four op-code including the operand part we were given total five 8 bit hexadecimal instructions.

At first, for simplicity purpose we converted the hex codes to binary 1's and 0's. So the 5 instructions with their binary forms are listed below.

Instruction Count	Instructions	Binary Conversion
1	0FB10000	00001111101100010000000000000000
2	0FB10FFF	00001111101100010000111111111111
3	1FD200C0	00011111110100100000000011000000
4	1FC30001	00011111110000110000000000000001
5	1FA100F0	00011111010000100000000011110000

So, we have used five 32 bit registers to store the above five, 32 bit instructions. During one clock, only one instruction will pass to the next stage, but need to have all the instructions fetched to the next part. So, here we have used one 3 bit program counter. It will basically fetch all the instructions one by one in different clock cycles. For better understanding, let during the first clock cycle the instruction 1 will be fetched, then for the second clock cycle program counter will increase the value by one and then instruction 2 will be fetched and so on. Then we have used multiplexer tree as shown in the figure. The output of the program counter is used as the select lines to the multiplexers. We have used total 192 (2:1) multiplexers divided them, as per we required to get the desired instruction one by one as shown in the figure. The three columns of multiplexers depending upon the three select lines provides us the outputs as per required.

Let us consider for the first 6 multiplexers. In the first MUX we have given first instruction of instruction 1 and first instruction of instruction 2. For the second MUX in its two inputs we have provided the first instruction of instruction 3 and first instruction of instruction 4 and in the next MUX we have provided the first instruction of instruction 5 and other terminal is grounded. For these 3 MUXES we use one common select line named as PC_0. If PC_0 is low then all Vin_0th input will pass if PC_0 is high then all Vin_1th input will pass. The output of these 3 MUXES are passed through 2 more multiplexers in the same way above mentioned and here the select line is PC_1, depending upon high and low of this select line the output of the MUX will be obtained and then is passed through one more MUX with select line PC_2. And finally we can obtain the first bit of the instruction. All the other multiplexers are working in the same manner. The output of the MUXES will be stored on the instruction register and then it will be divided into 2 parts i.e. op-code and operand. The op-code i.e. the first 16 most significant bits are stored in a 16 bit register and the operand i.e. the 16 least significant bits are provided as inputs (x) to the comparators. And the values stored in the registers.

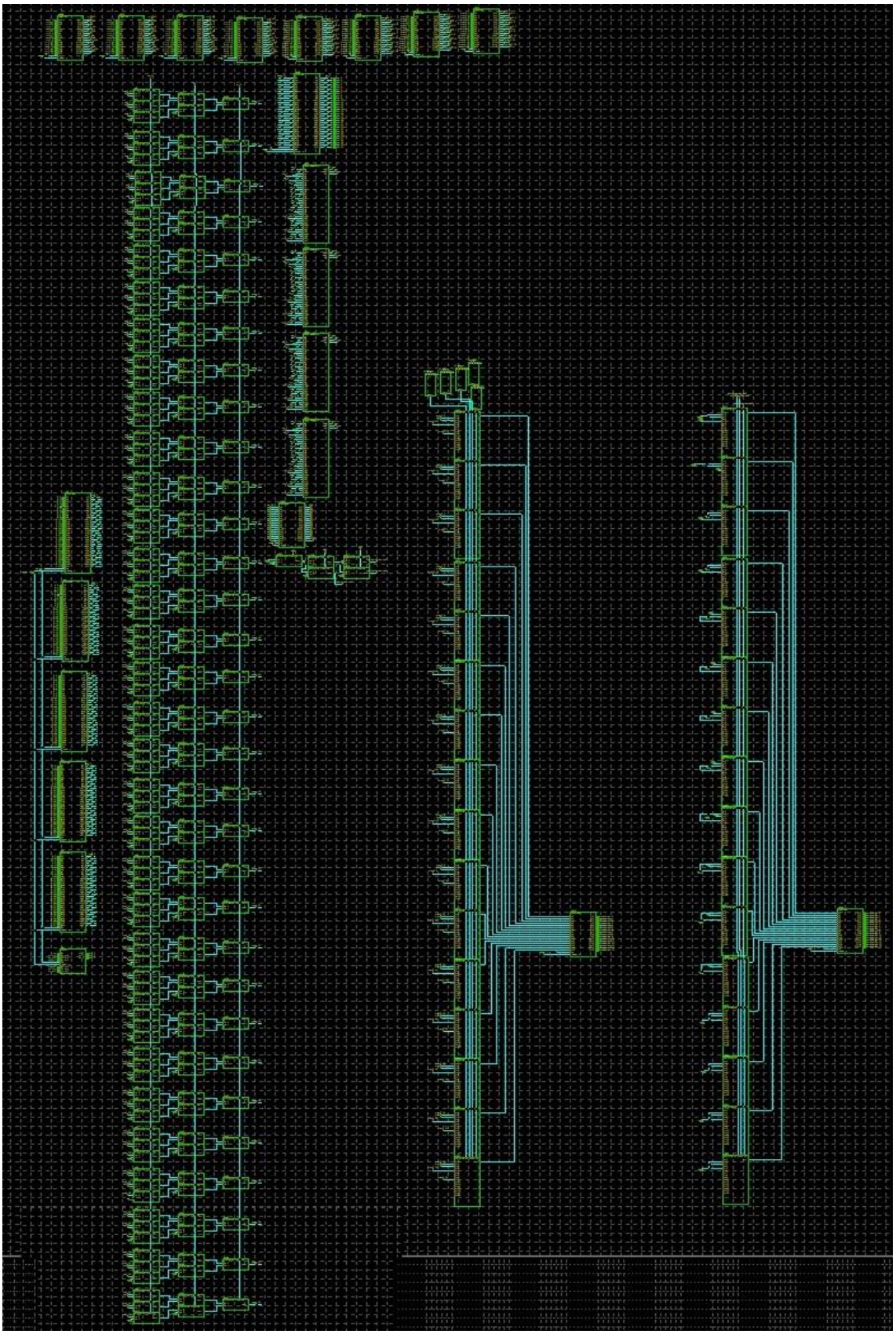


Fig:- Automated Speed Controller.

Here, we have used eight 16 bit registers to store the threshold value and its output intensity. We have 4 comparators each one filled with one threshold value. The input and output threshold intensity with their binary form is listed below.

i/p threshold intensity	Binary	o/p threshold intensity	Binary
000D	0000000000001101	FFFF	1111111111111111
00DD	0000000011011101	0FFF	0000111111111111
0DDD	0000110111011101	00FF	0000000011111111
DDDD	1101110111011101	000F	0000000000001111

During the first clock cycle instruction 1 will be fetched and operand of the instruction will go to all the comparators. If the operand is less than, or equal to 000D or less than 00DD then also in the speed register FFFF instruction will be stored. If the given operand is equal to 00DD or less than 0DDD then in the speed register 0FFF will be stored. Again if the operand is equal to 0DDD or less than DDDD then 00FF will be stored and if the operand is equal to DDDD or greater than DDDD then in the speed register, 000F will be stored. We have designed this logic using 16:1 multiplexer and the output of the comparators acts as the select lines. We have provided the inputs to only that terminals which can give us the output as we required as shown the circuit above.

Now for the breaking part we have stored all the op-codes in a register. Here we need only 3 last instructions. When we observed the instructions we saw that all three instructions are varied in the 22^{th}, 28^{th} and 20^{th} bit. So using MUX circuit we have made a logic that if 28^{th} bit is high and 22^{th} bit is low then we get instruction 5. Similarly with 22^{th} and 28^{th} bit high, if 20^{th} bit is high then it will be instruction 3 and if 20^{th} bit is low then it will be instruction 4. We have used the output of these MUXES as select lines to a another 16:1 MUX tree where we have directly provided the input values as we required.

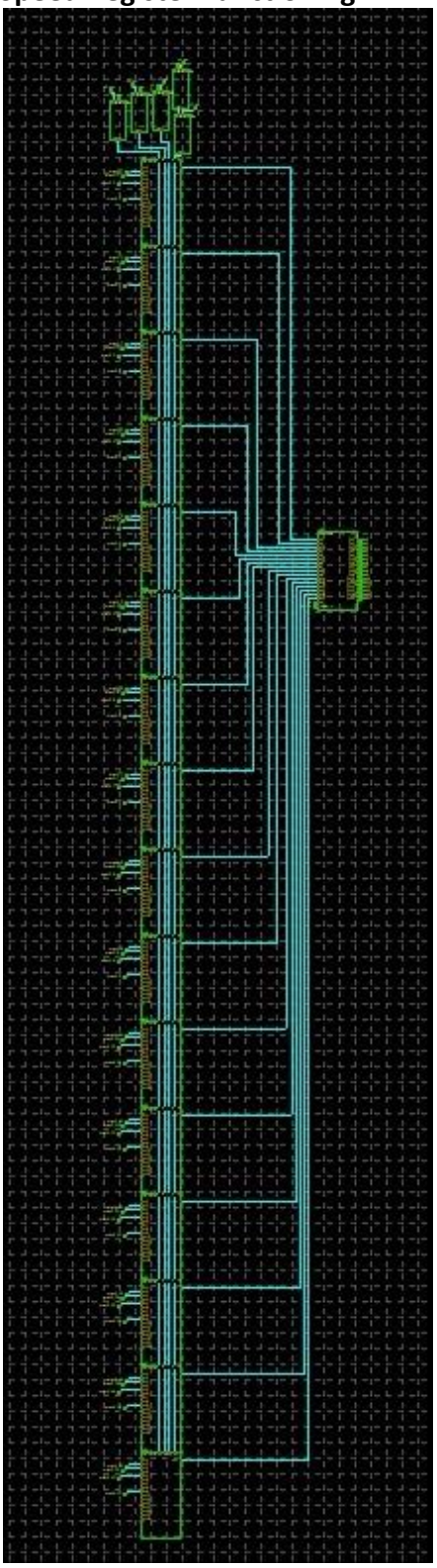
So, on the basis of the select lines we obtain the output of the breaking register i.e. No breaking (000F), Partial breaking (00F0) and Complete break (0F00).

As the circuit is too large, so we are unable to simulate such a huge circuit with too many of individual components. So we divided the circuit into parts. We have basically divided the whole circuit into three parts.

1. Instruction register part,
2. Speed register functioning and
3. Breaking register functioning.

So, after we simulate the three individual parts it works perfectly fine. So, these parts are also shown below.

Speed Register Functioning:-



As we can see we have used sixteen 16:1 multiplexers, with the OR gates. The input of the OR gates are the output of the comparators. The output of the MUXES will be stored on the speed register itself.

The test bench of the speed register is shown below.

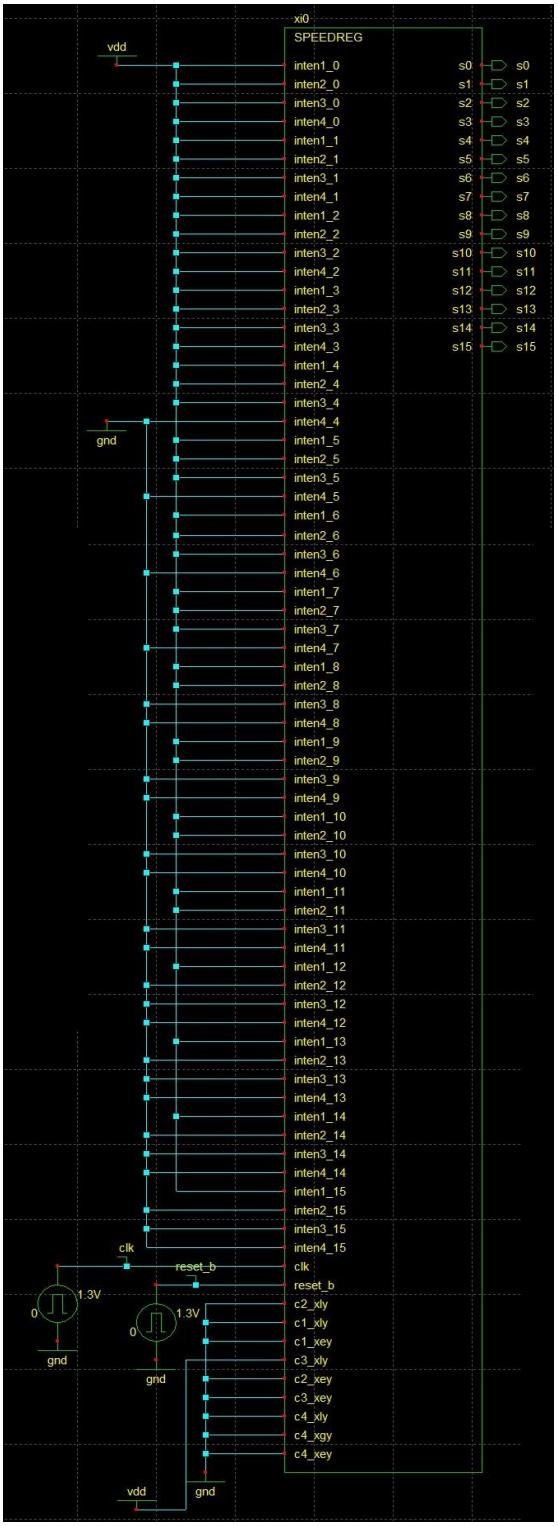
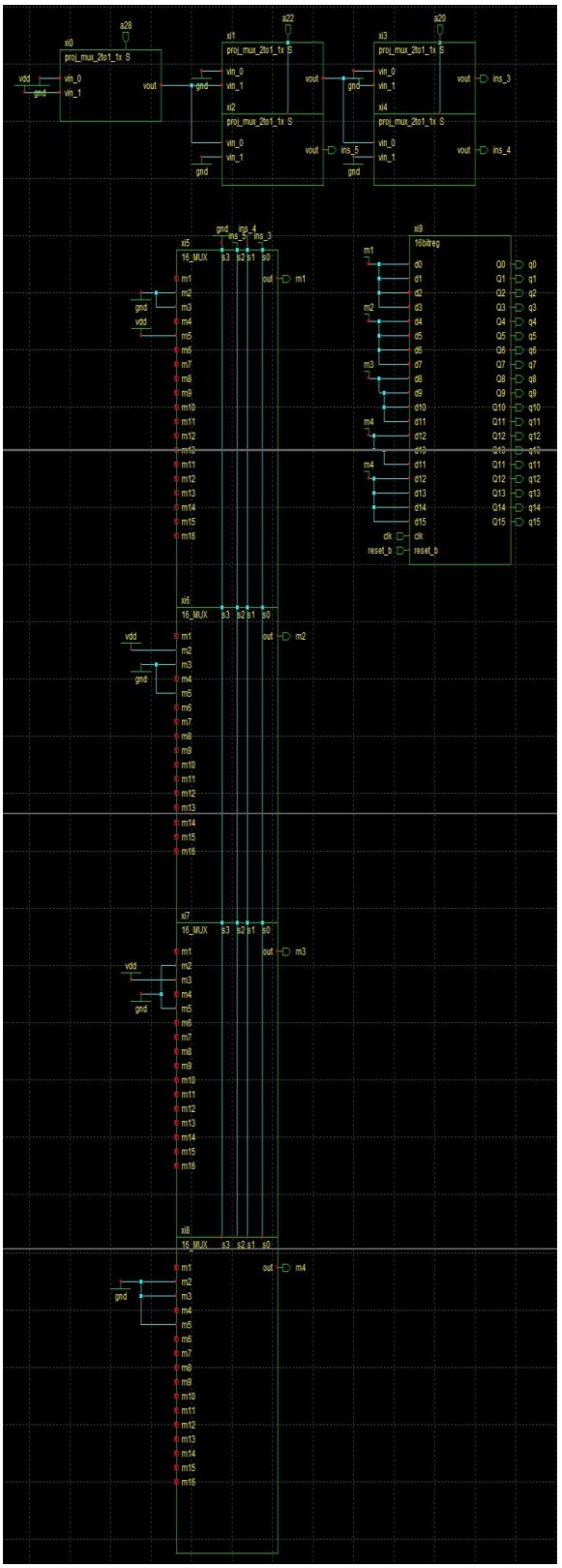


Fig:- Test bench of speed register

As we can see all the inputs are provided as per required and are obtaining the desired outputs as per required. The input and output graphs are shown in the next part.

Breaking Register Functioning:-



As per the explanation given above, here the outputs of the upper MUXES are given as the select lines to the 16:1 MUX tree. The inputs of the 16:1 MUX tree is given as per required and the output of the MUX tree will be stored on a register called Breaking Register.

The test bench of the circuit is shown below.

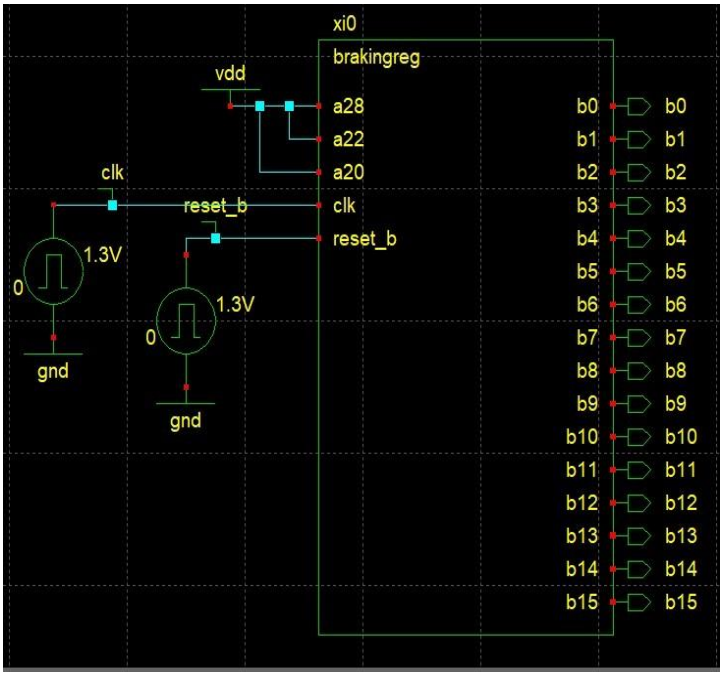


Fig:- Test bench of the breaking register.

As we can see the clock and reset are provided with pulse signal and the other inputs are provided with 1's and 0's. As per we provided the inputs, we can obtain the desired outputs. Also, the output graphs are shown in the next part.

Simulation Plots Of Input, Output and Important Intermediate Signals:-

As our speed register, breaking register and instruction register are working completely fine so, we are able to share the output graphs of these three registers as given below.

- **Output of instruction register.**

The output of the instruction register is available to control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction. The instruction is loaded into the instruction register after the processor fetches it from the memory location pointed to by the program counter.

- **Output graphs of the speed register.**

The first output can be obtained by supplying C3_xdy input as high. The second output is the intensity of the threshold.

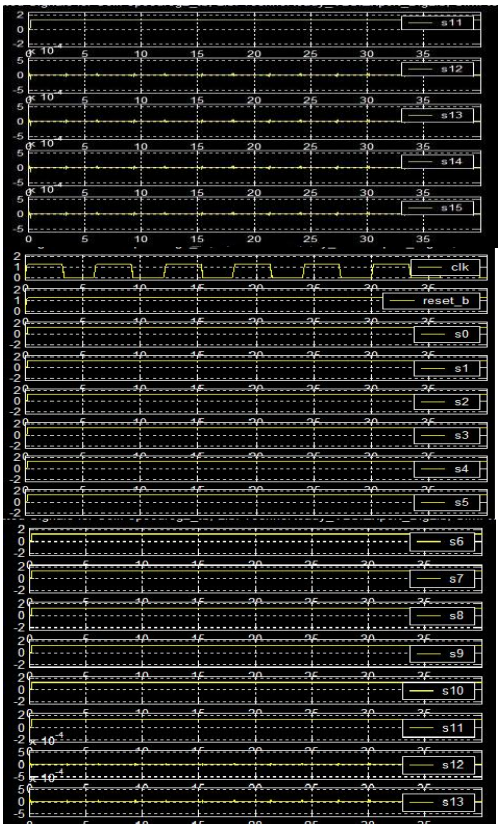


Fig:- Output Speed Register.

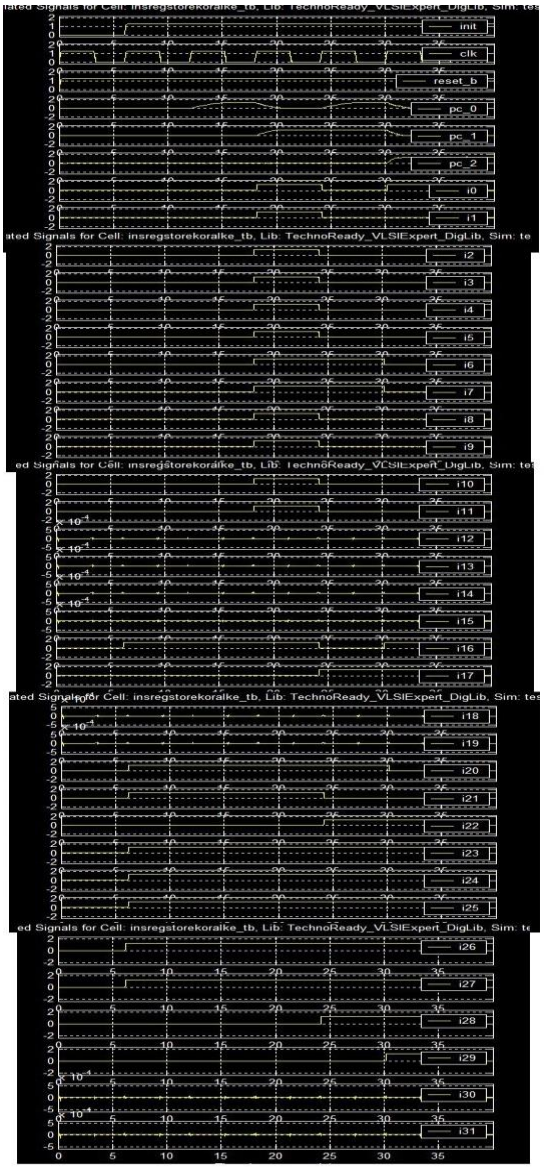


Fig:- Output of Instruction Register.

- **Output of breaking register.**

As we gave a22, a28, and a20 all bits as 1 so, the instruction 3 will appear and hence the output of the breaking register will show partial breaking instruction as 00F0. The output graph is shown beside.

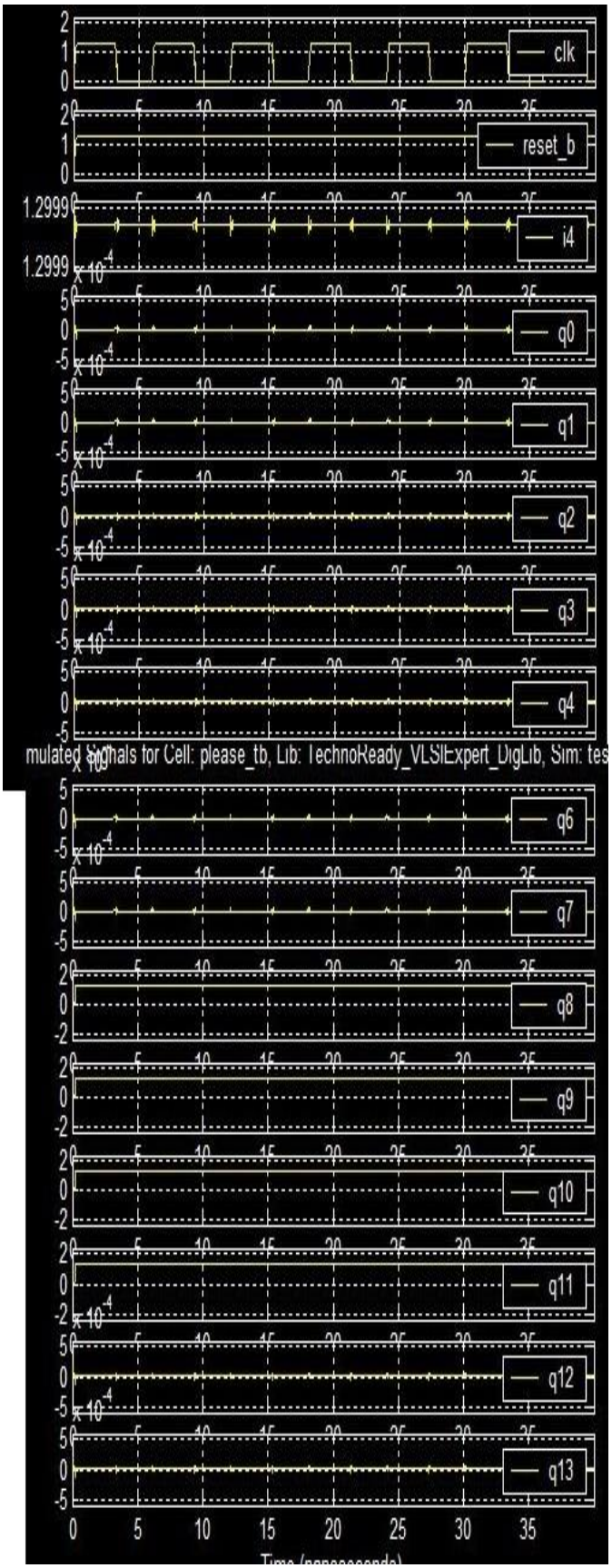


Fig:- Output of Breaking Register.

Conclusion and Scope of Development:-

This project presents the implementation of an Automatic Breaking System for Forward Collision Avoidance, intended to use in vehicles where the drivers may not brake manually, but the speed of the vehicle can be reduced automatically due to the sensing of the obstacles. With this future study and research, we hope to develop the system into an even more advanced speed control system for automobile safety also addition of wi-fi device to communicate with other cars around to avoid collisions and possibility to alter the direction of collision, while realizing that this certainly requires tons of work and learning, like the programming and operation of microcontrollers and the automobile structure. We believe that the incorporation of all components in Automatic Breaking System will maximize safety and also give such system a bigger market space and a competitive edge in the market.