Summer Fellowship Report

On

Implementation of GUI Interface and Simulation of Multiple Instance for Attiny Microcontroller

Submitted by

Vatsal Patel

Under the guidance of

Prof. Kannan M. Moudgalya

Chemical Engineering Department

IIT Bombay

September 8, 2022

Acknowledgement

I would like to express my very gratefulness to Prof. Kannan M. Moudgalya for his valuable and constructive suggestions. His willingness to give his time so generously and encouraging fellows have been very much appreciated.

I would also like to thank the eSim team for giving me such a great learning opportunity, being a part of such a wonderful project, and their help in offering me the resources and guiding me throughout the project.

I would also like to thank my Project managers, Usha Viswanathan and Vineeta Ghavri, for their guidance and support throughout the fellowship.

A special thanks to my mentors, Sumanto Kar and Rahul Paknikar, for helping me throughout the fellowship, sharing a lot of knowledge with me, guiding me, and giving me a wonderful fellowship experience.

Finally, I wish to thank my mother for her support and encouragement throughout my study.

Contents

1 Introduction

1.1 HEX Files

A HEX file is a hexadecimal source file. These files are used mainly for programmable logic devices like microcontrollers. The HEX file contains all the settings regarding the information about the configuration of I/Os, the logic behind controlling the process/ task, and the other data saved in hexadecimal format. These files are stored in either binary or text format.[1]

1.2 Importance of HEX files in microcontrollers

The HEX files store the machine code in hexadecimal form. It is widely used to store programs, to be transferred to microcontrollers, ROMs, EEPROMs, etc. The corresponding compilers convert the programs, which are written in C or the assembly language, etc into the respective hex files. Now, these files are flashed/ dumped into the microcontrollers using burners or respective programmers.

While simulating the circuits with microcontrollers, the same process of uploading/flashing up the machine code onto the microcontrollers is needed. Hence, HEX file content must be uploaded onto the microcontroller.

Since the microcontroller understands only the machine language consisting of zeroes and ones, it is practically difficult for humans to write codes in such a manner. Hence, by using high-level languages, we write the code, and then using a compiler, the high-level language code gets converted into machine language that gets stored in the hex file format. A HEX file is a text file with a .hex extension. eSim accepts and works with both Text(.txt) and HEX(.hex) formats of HEX file as the content would be same in either format. A HEX file contents may look like as shown in fig. 1.1 as below.

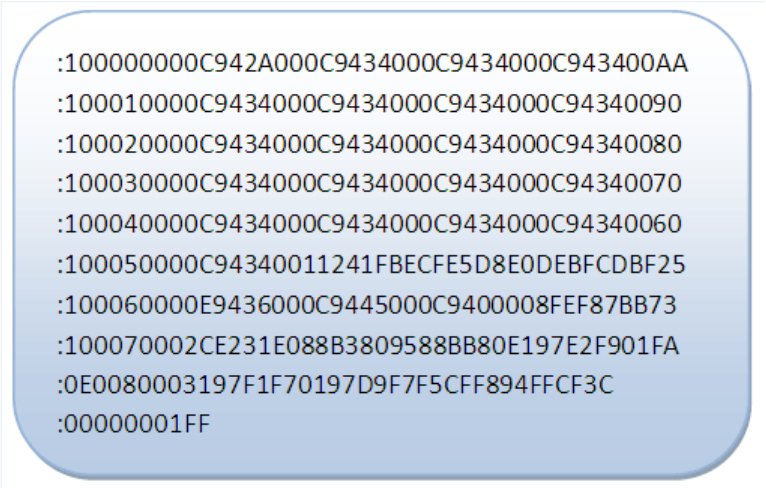


Figure 1.1: HEX file Format

1.3 eSim Microcontrollers

At present eSim provides the Attiny series microcontrollers[2]. Using these microcontrollers, one can create their required circuits and simulate them. Since the microcontrollers require the HEX file to be uploaded, from which the microcontroller will configure its I/Os and process as per the specified code, there is a need of uploading the HEX files onto the microcontroller. Once the HEX file is uploaded, the circuit can be simulated successfully.

2 Microcontroller Simulation in eSim

2.1 Uploading HEX file onto microcontroller in eSim

eSim provides the Attiny series of microcontrollers for simulation of the circuits. The user has to upload the hex file onto the microcontroller for the proper working of the circuit as intended. For simulating the microcontrollers in eSim, one has to set up the NGHDL server[3] for the respective Attiny microcontroller instance which may be Attiny25/45/85. Once the server is set up, the files associated with the microcontroller can be found in the DUTghdl folder. This folder contains all the necessary required files like the microcontroller VHDL code file, start server batch file, hex.txt file, etc. The hex.txt file is the file that contains the code that needs to be loaded. On simulating the circuit on eSim, the contents of the hex.txt are fetched, and accordingly, the microcontroller present in the circuit is configured i.e., the configuration of the microcontroller is done accordingly.

2.2 Workflow

As mentioned in section 2.1, the HEX file contents must be uploaded in the hex.txt file present in the respective DUTghdl folder of the particular instance of the microcontroller. The current procedure has the same steps i.e., uploading the HEX file or the contents of the HEX file onto the microcontroller by manually copying the contents of the HEX file and pasting it into the hex.txt file present in the DUTghdl folder. The current workflow procedure carried out manually is shown below in figure 2.1.

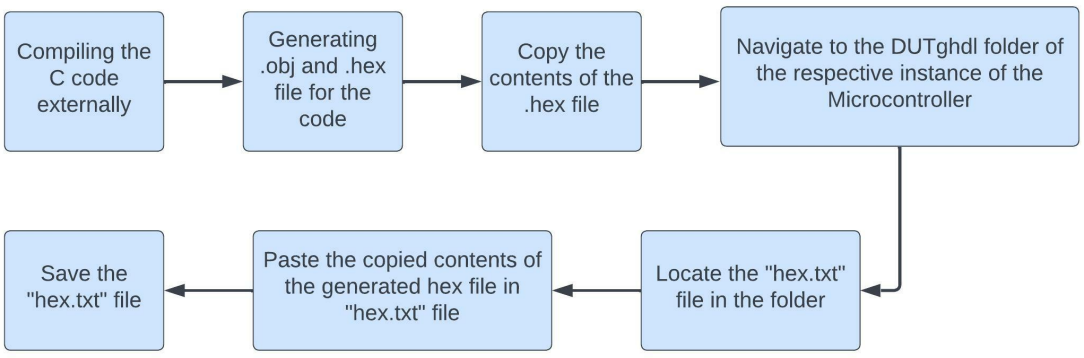


Figure 2.1: Current workflow

2.2.1 DUTghdl Folder

The path for the DUTghdl folder for the respective instance of the microcontroller is very long. One has to navigate through many folders, and after this folder can be located. Hence, practically it is not a feasible process to be followed.

For eg.: The path for the Attiny85 instance of microcontroller in Linux OS is: As mentioned above, the process is tedious and carried out manually. Also, the different instances of microcontrollers like Attiny25, Attiny45, or Attiny85 will have different NGHDL servers, and hence, it will have different folders present in the ghdl folder[4]. The path for these three microcontrollers is as follows:

3 Problem Statement

The current workflow for uploading the HEX file onto a microcontroller is tedious and has to be manually carried out. Every time user changes something in the circuit and wants to check output the hex.txt file needs to be updated manually, costing much user time and effort.

3.1 GUI

There should be a separate section provided for microcontrollers. Currently all the Ngspice models of the circuit in eSim are listed under Nspice Models tab. It creates difficulty and confusion for user to search the Microcontroller instances everytime when user wants to chage the model parameters.

3.2 Hex File Upload

There is a need for a feature in the eSim, which lets the user upload the HEX file from the software itself. If any circuit contains microcontroller in the schematic user needs to update the parameters of microcontroller by a long and tedious process described in eSim. If there are n microcontrollers in the circuit user needs to repeat same process n number of times.

Hence there is need of a button to select .hex file path for respective microcontroller instance while the software taking care of all the process.

3.3 Multiple Instances of Microcontroller

eSim doesn't support the simulation of circuits with multiple instance of a specific microcontroller in one schematic.

Ngspice is capable of simulating such circuits but the problem lies in the interface between eSim and Ngspice. It is difficult to differentiate between the microcontroller instances and also to have separate .hex files for all the microcontrollers

4 Solution

Solutions to the problems stated in section 3 are mentioned below

4.1 GUI

As the microcontrollers as currently listed in the Ngspice Models tab it creates confusion and difficult for the user to find the microcontroller in other components. So a new tab **Microcontroller** has been created as in image below.

This tab will contain all the microcontrollers(Attiny25/45/85) used in the circuit within this tab creating it easy for the user to modify the parameters. Also, a parameter **Instance Id** has been removed from GUI and is being used for internal working.

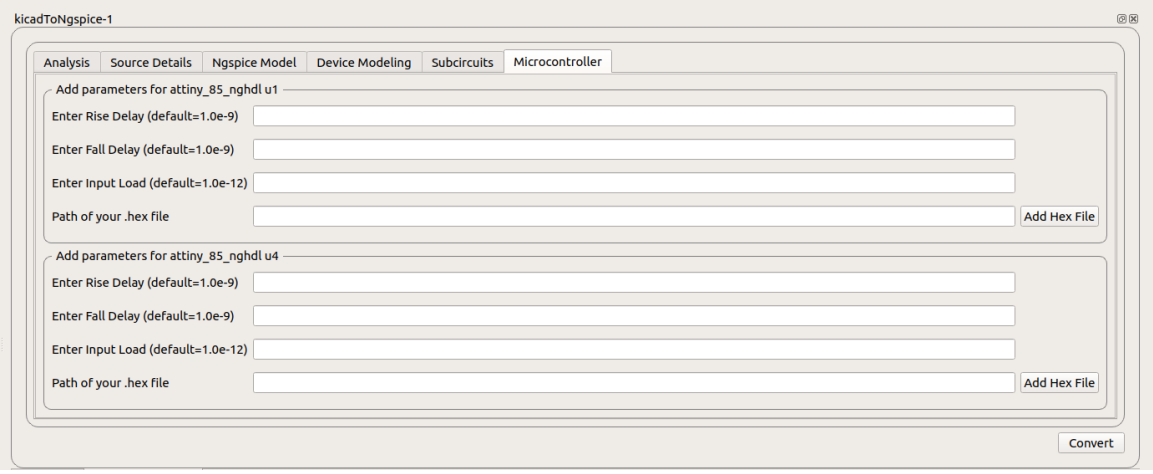


Figure 4.1: New Microcontroller Tab

4.2 Hex File Upload

4.2.1 Introduction

If any circuit comprises any of the instances of the microcontroller, then for the respective microcontroller there will be an option to add the HEX file. This feature is provided in the KicadtoNgpice module, under the Microcontroller tab. Once the user finishes up with the circuit designing, on converting the current circuit to Ngpsice Model, for setting up the various transient parameters, or the AC/DC parameters, the details of the sources, the user has to execute the KicadtoNgpice module. In this

if the circuit contains any of the components like ADC or DAC, then the different parameters associated with them like rise time, delay time, etc. need to be entered by the user if required, or else the default values are selected. In this section, the button to add the HEX file is placed. If the circuit will contain any instance of the microcontroller, (which may be any Attiny25/45/85), one more additional parameter Path of your .hex file will be enabled.

4.2.2 Workflow

The user will have to follow simple 3 steps:
1. Press Add Hex File button
2. Browse the HEX file

Step 1: Press Add Hex File button
On clicking the Add Hex file button as shown in figure 4.2, a window will be prompted allowing the user to navigate through the computer to upload the HEX file for the microcontroller circuit. This will only show .hex files and .txt files present in any of the directories.

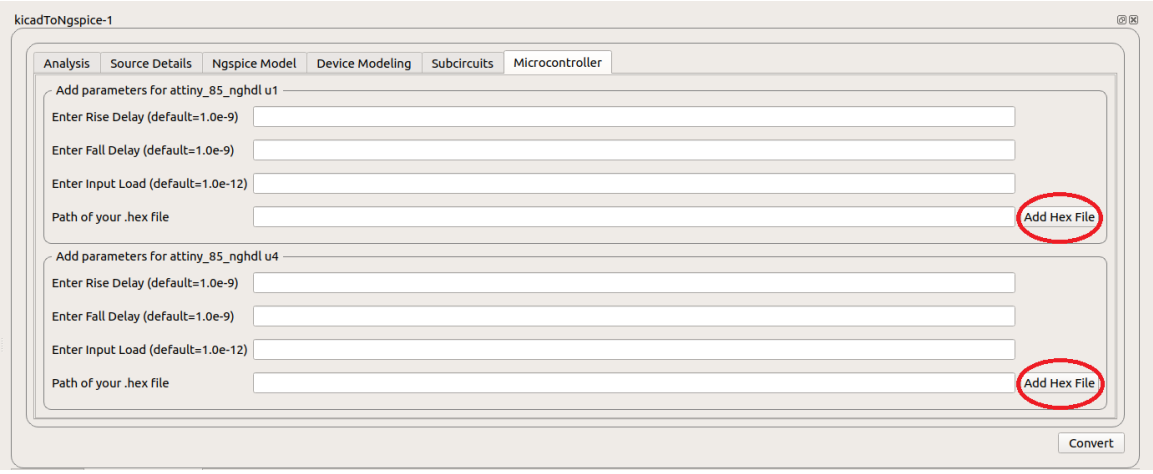


Figure 4.2: Hex File Selection Button

Step 2: Browse the Hex File

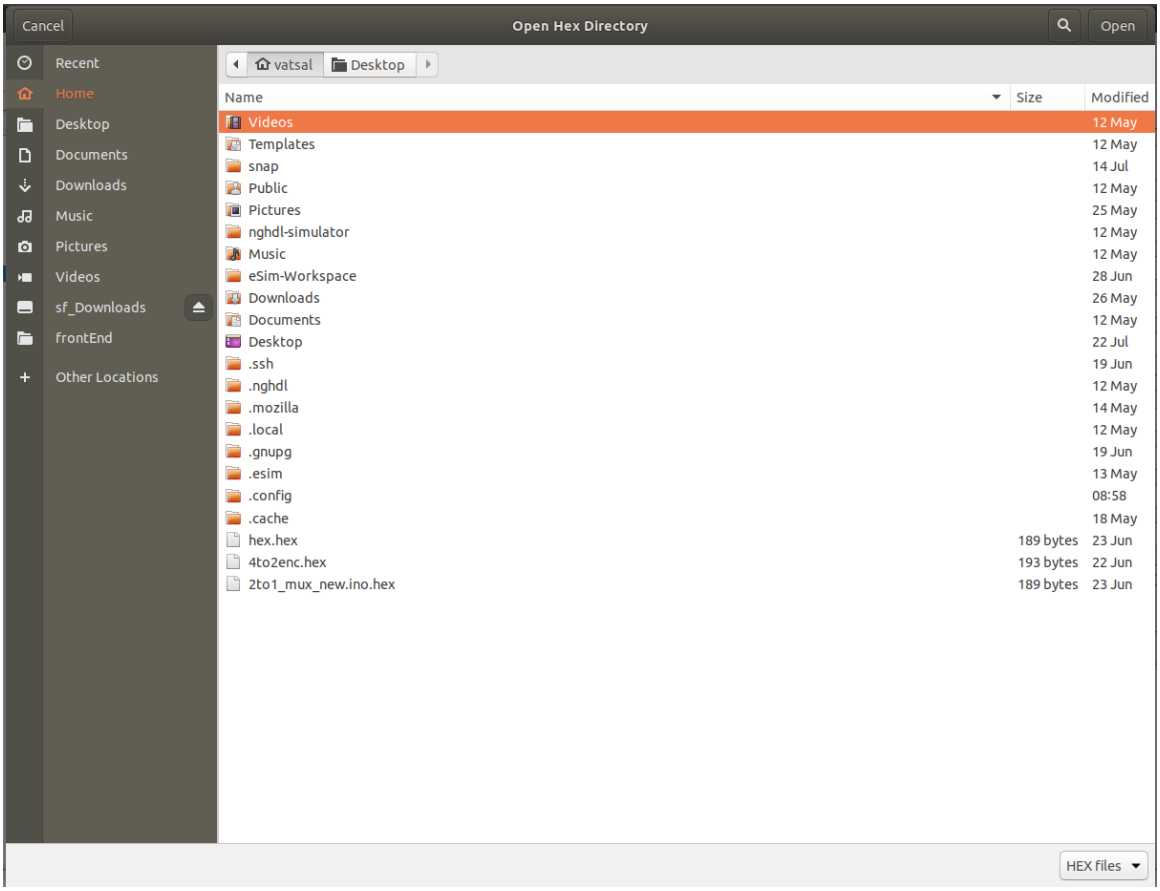


Figure 4.3: Hex File Selection Dialog

The user needs to select the appropriate .hex file for the microcontroller located on the computer. Once the user has selected the .hex file, the user will click on the **Open** button. After the user presses the **Open** button dialogue disappears and the selected file path will be displayed in the respective text holder on the left to it.

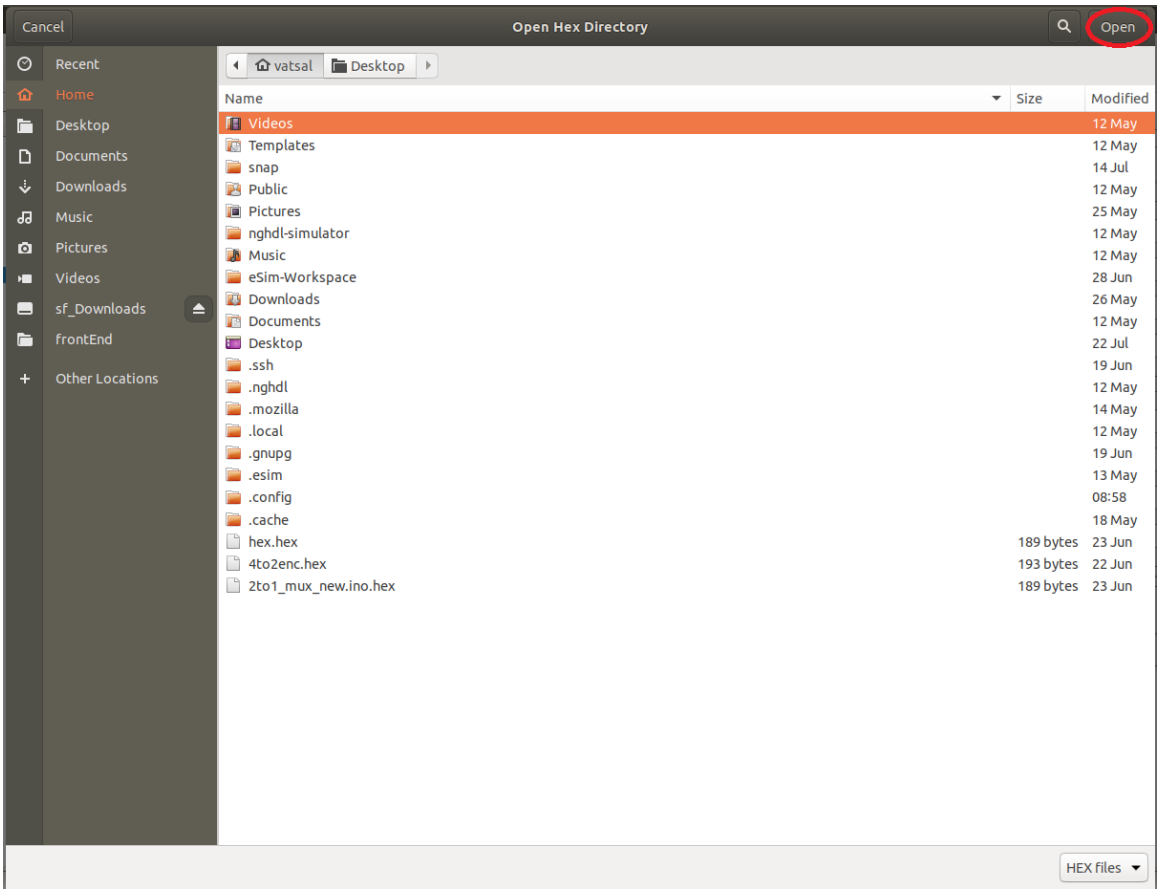


Figure 4.4: Hex File Selection Button

Once the user selects the .hex file and clicks on Open button, the file path appears in the Microcontroller tab as shown below in figure 4.5

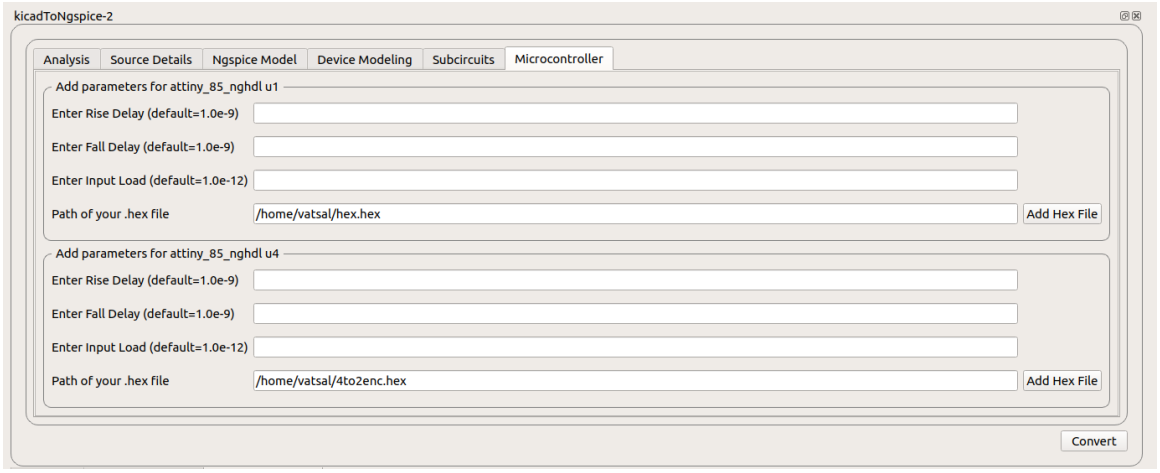


Figure 4.5: Hex Files Selected

4.2.3 Working

The user needs to click on the Convert button to perform the Kicad-to-Ngspice operation. On clicking the button the hex file selected is passed through various parts of eSim and gets fully lowercase in cfunc.mod file. It does not create any problem in Windows as it is case insensitive, but it creates severe problems in Linux.

For eg. the path **"/home/Desktop/Folder"** gets converted to **"/home/desktop/-folder"** so Linux is not able to locate desktop as it has Desktop in it's definition. Hence a special provision is made in Convert.py and cfunc.mod to encode and decode the file path.

The path sent to cfunc.mod is encoded with an asterisk(*) such that the capital letter in the path has one asterisk(*) ahead it and 2 asterisks (*) behind it.

For eg. the path path **"/home/Desktop/Folder"** gets encoded to path **"/home/*D**esktop/*F**older"** while sending. The cfunc.mod file receives the path as path **"/home/*d**esktop/*f**older"** now it will convert all the original capital letters to their initial format and remove all the asterisk(*). Hence the filtered path would be again **"/home/Desktop/Folder"**. This path is then passed to the controller where it reads the file during the simulation.

The whole process of path processing and passing is described in figure 4.1 below

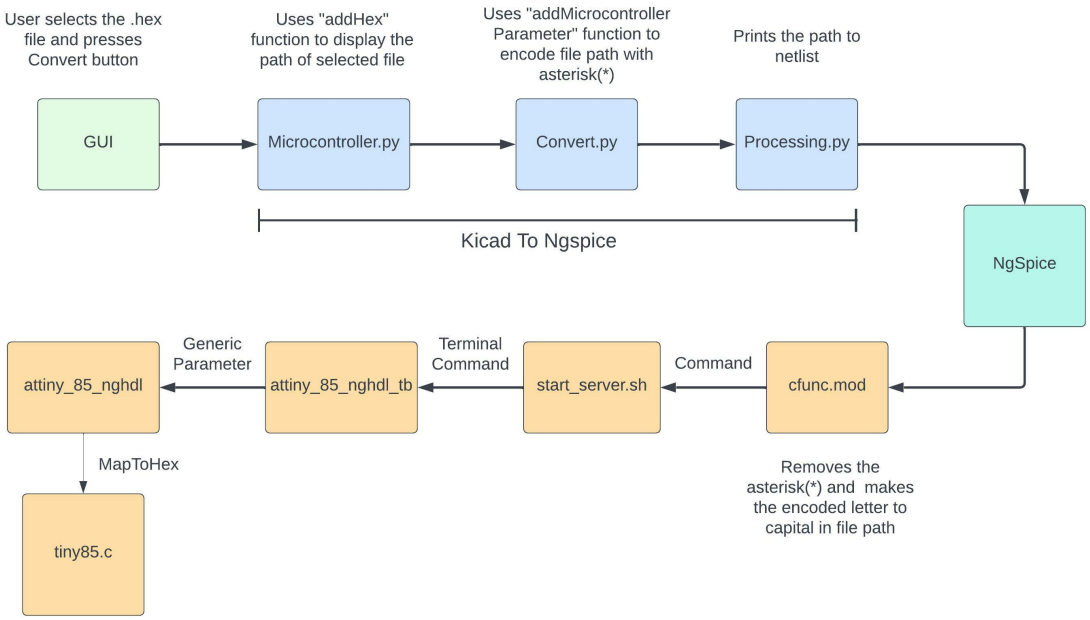


Figure 4.6: Path Processing and Passing

The GUI flow of the HEX gile path has been expained here. The Flow from Ngspice to tiny85.c has been explained in detail in Section 4.3.1 Multiple Instances Working.

The files updated or created in eSim **frontend** folder are mainly:

- Microcontroller.py
- Convert.py
- KicadtoNgspice.py

Microcontroller.py

```
def addHex(self):
        """
        This function is use to add the file path of the selected .hex file
            to the respective QLineEdit
        """
        if os.name == 'nt':
            self.home = os.path.join('library', 'config')
        else:
            self.home = os.path.expanduser('~')
```

```
        self.parser = ConfigParser()
        self.parser.read(os.path.join(
            self.home, os.path.join('.nghdl', 'config.ini')))
        self.nghdl_home = self.parser.get('NGHDL', 'NGHDL_HOME')

        self.hexfile = QtCore.QDir.toNativeSeparators(
            QtWidgets.QFileDialog.getOpenFileName(
                self, "Open Hex Directory", os.path.expanduser('~'),
                "HEX files (*.hex);;Text files (*.txt)"
            )[0]
        )
        if not self.hexfile:
            """If no path is selected by user function returns"""
            return
        chosen_file_path = os.path.abspath(self.hexfile)
        btn = self.sender()

        """ If path is selected the clicked button is stored in btn
            variable and checked from list of buttons to add the file path
            to correct QLineEdit"""

        if btn in self.hex_btns:
            if "Add Hex File" in self.sender().text():
                self.obj_trac.microcontroller_var[4 +
                    (5*self.hex_btns.index(btn))].setText(chosen_file_path)
```

Microcontroller.py has many similar functions to Model.py. But the **addHex** function is created to solve the problem of HEX file path. It handles and operates on user response to HEX file path selection. It identifies file path selected for the respective instance of microcontroller and displays the file path in that particular instance of the microcontroller.

A list of buttons **hex_btns** is created that contains all the instantiated **Add Hex File** buttons. Also all the buttons are connected with a listener that call addHex function when any of the button is clicked. This **hex_btns** helps in checking which button was clicked and where to enter the selected HEX file path.

Convert.py(addMicrocontrollerParameter() Function)

```
def addMicrocontrollerParameter(self, schematicInfo):
   z=0
```

```
for key, value in line[9].items():
    # Checking for default value and accordingly assign
    # param and default.
    if ':' in key:
        key = key.split(':')
        param = key[0]
        default = key[1]
    else:
        param = key
        default = 0
    # Checking if value is iterable.its for vector
    if (
        not isinstance(value, str) and
        hasattr(value, '__iter__')
    ):
        addmodelLine += param + "=["
        for lineVar in value:
            if str(
                    self.obj_track.microcontroller_var
                    [lineVar].text()) == "":
                paramVal = default
            else:
                paramVal = str(
                    self.obj_track.microcontroller_var
                    [lineVar].text())
            # Checks For 5th Parameter(Hex File Path)
            if z == 4:
                chosen_file_path = paramVal
                star_file_path = chosen_file_path
                star_count = 0
                for c in chosen_file_path:
                    # If character is uppercase
                    if c.isupper():
                        c_index = chosen_file_path.index(c)
                        c_index += star_count
                        # Adding asterisks(*) to the path around the
                            character
                        star_file_path = star_file_path[:c_index] + "*" +
                            star_file_path[c_index] + "**" +
                            star_file_path[c_index+1:]
                        star_count += 3
```

```
            paramVal = "\"" + dollar_file_path + "\""

        addmodelLine += paramVal + " "
        z = z+1
    addmodelLine += "] "
```

Here Convert.py encodes the file path as described in the beginning. Every instance of microcontrolle has 5 parameters namely instance_id, input_load, rise_delay, fall_delay, hex_path. But we need to encode only **hex_path** so we use **z** as a counter to iterate on parameters. As it reaches 5th item (z=4) we will take the **paramVal** and process and encode it and append it at the end.

4.2.4 Code Snippets

To see the code changes in detail please click here.

4.3 Multiple Instances of Microcontroller

Currently, eSim is not able to simulate the circuits with two or more instance of a specific microcontroller. Ngspice is capable of simulating such circuits but the problem lies in the interface between eSim and Ngspice. It is difficult to differentiate between the microcontroller instances and also to have separate .hex files for all the microcontroller instances.

Both the problems have been solved as **Instance ID** is now used to differentiate between the microcontroller instances and the **hex_path** parameter has been added to the model parameters of the microcontroller.

Hence the Instance ID and hex_path are used to solve the problem of multiple instances of the microcontroller.

4.3.1 Working

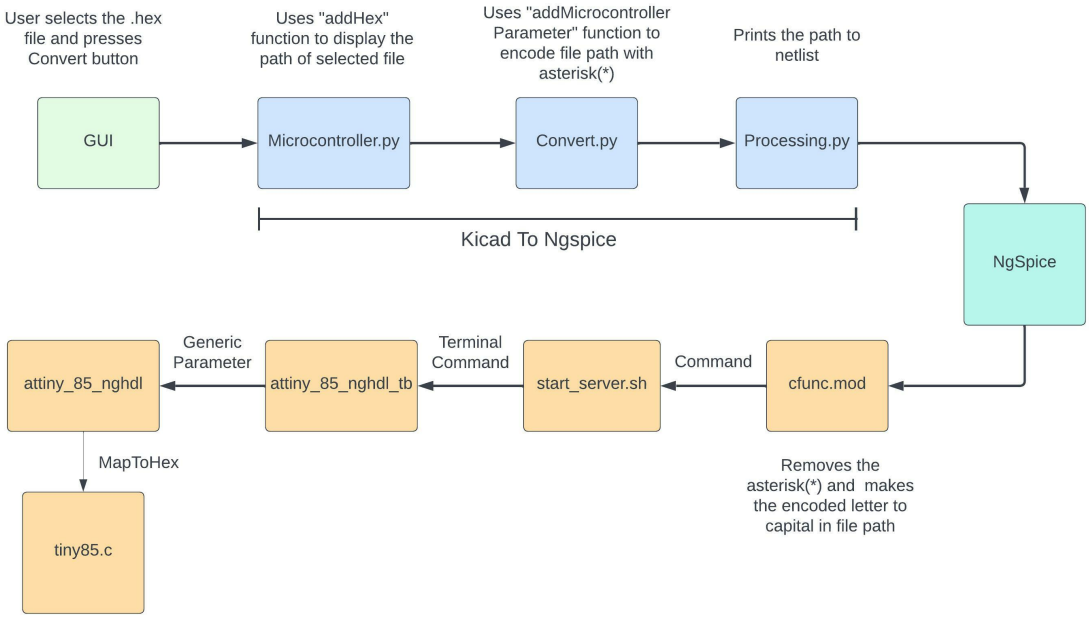


Figure 4.7: Path Processing and Passing

The above image shows complete flow through which hex file path passes at the time of simulation. Here the files updated in **nghdl** folder of eSim are mainly :-

- cfunc.mod
- start_server.sh
- attiny_85_ngdhl_tb
- attiny_85_nghdl
- tiny85.c

Some other files have been edited to support the hex file path passing.

cfunc.mod

```
char* remove_star(char * str, char char1, char char2){
   printf(".......Remove Star.................");
    int i, j;
    int len = strlen(str);
```

```
    for(i=0; i<len; i++){
      if(str[i] == char1){
         if(str[i+2] == char2 && str[i+3] == char2){
            str[i+1] = str[i+1] - 32;
            for(j=i; j<len; j++)
            {
               str[j] = str[j+1];
            }
            for(j=i+1; j<len; j++)
            {
               str[j] = str[j+2];
            }
            len--;
            i--;
         }
      }
    }
    return str;
}
```

In cfunc.mod a function **remove_star** has been created that takes input as string along with two parameters char1 and char2. Here string is checked for character matching to char1 and char2. If matching characters are found then those characters are removed and the character in between them is converted to uppercase. Resulting in the original HEX file path sent by the GUI side.

start_server.sh

```
./attiny_85_nghdl_tb -ghex_path_tb=$3
```

The HEX file path is passed to attiny_85_ngdhl_tb via start_server.sh. For this a parameter **hex_path_tb** is used.

attiny_85_ngdhl_tb

```
entity attiny_85_nghdl_tb is
       generic( hex_path_tb : string );
end entity;

generic( hex_path : string );
generic map(hex_path => hex_path_tb)
```

To pass the parameter to attiny_85_ngdhl a new paramters **hex_path** is created and mapped from **hex_path_tb**.

attiny_85_nghdl

```
generic( hex_path : string );

process
begin
report "Hex path in attiny 85 in VHDL" & integer'image(hex_path'length);
for item in 1 to hex_path'length loop
   MapToHex(hex_path(item));
   end loop;
wait;
end process;
```

Once attiny_85_ngdhl receives the HEX file path, it sends it to **tiny85.c** with the help on newly created function MapToHex. This function transfer file path to tiny85.c with help of ghdl_access.vhdl file.

tiny85.c

```
void MapToHex(char hex_temp)
{  printf("%s",hex_path);
   hex_path[hex_count++]=hex_temp;
}

void MapToRam(int flag)
//Function to map the external hex file contents into this C code
{
   int i=0,filesize,j,s,line,adr=0,lineCount=0;
   unsigned char c,temp;
   hex_path[++hex_count]='\0';
   SetRam(0,size,0x0);
   if(flag==1)
   {
      FILE *fptr;
      char * line = NULL;
      size_t len = 0;
      ssize_t read;
        printf("\nHex path in tiny85 %s\n", hex_path);
      fptr = fopen(hex_path, "r");
```

```
while ((read = getline(&line, &len, fptr)) != -1)
{
     printf("\nLine: %s\n", line);
```

In tiny85.c a parameter **static char hex_path** is created that saves the HEX file path of the respective instance of microcontroller. **hex_path** parameter is used by **MapToHex** to store the value passed by attiny_85_ngdhl. This value is used by another function **MapToRam** during simulation to open correct HEX file and read its content.

The HEX file path is passed through the above listed files during simulation. This helps enable multiple instance simulation in eSim as every instance of microcontroller will have its own HEX file path so it won't cause any conflicts and Ngspice can easily recognize them during the simulation.

4.3.2 Code Snippets

To see the code changes in detail please click here.

5 Mixed Signal Circuit Examples

5.1 Counter with Astable Multivibrator

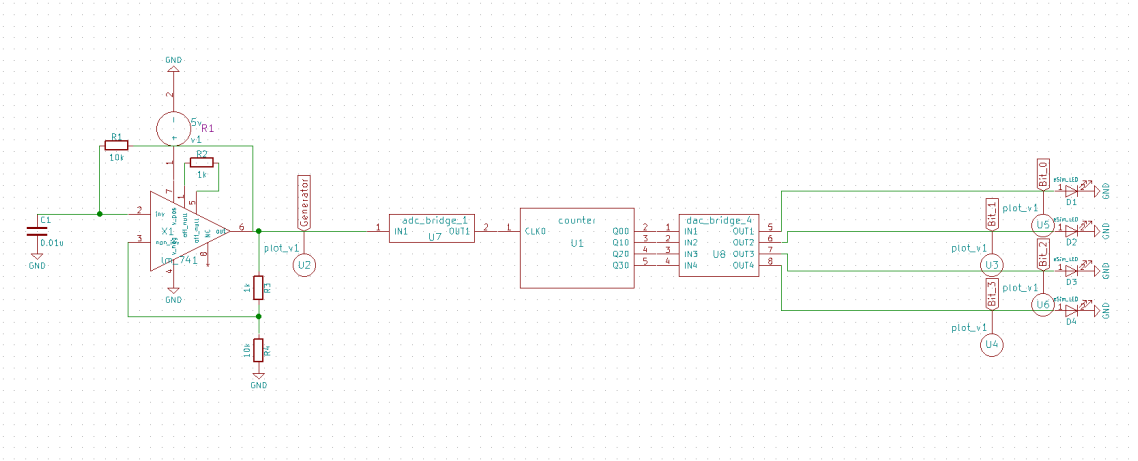


Figure 5.1: Counter with Astable Multivibrator Schematic

The Astable Multivibrator is also called a free-running multivibrator. It has two quasi-stable states and no external signal is required to produce the changes in state. The component values are used to decide the time for which circuit remains in each state. Usually, as the astable multivibrator oscillates between two states, is used to produce a square wave. In this circuit, the time period is dependent upon the value of the resistor and capacitor. It also depends upon the upper and lower threshold voltage of the op-amp[5].

The counter is a sequential circuit. A digital circuit that is used for counting pulses is known as a counter. The counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Not only counting, but a counter can also follow a certain sequence based on our design like any random sequence 0,1,3,2.They can also be designed with the help of flip-flops.

Here Astable Multivibrator is connected to the counter such that the clock signal generated by the Astable Multivibrator is used as input for the counter. The frequency of the clock signal generated by the Astable Multivibrator is 2.6kHz. The counter here is made of 4 JK Flip Flops with J and K as 1 while input as the clock signal. The output of the counter is in form of 4 signals representing respective bits of a binary number. By combining the bits and converting them from binary to decimal we get the exact count of the counter.

5.1.1 Code Snippets

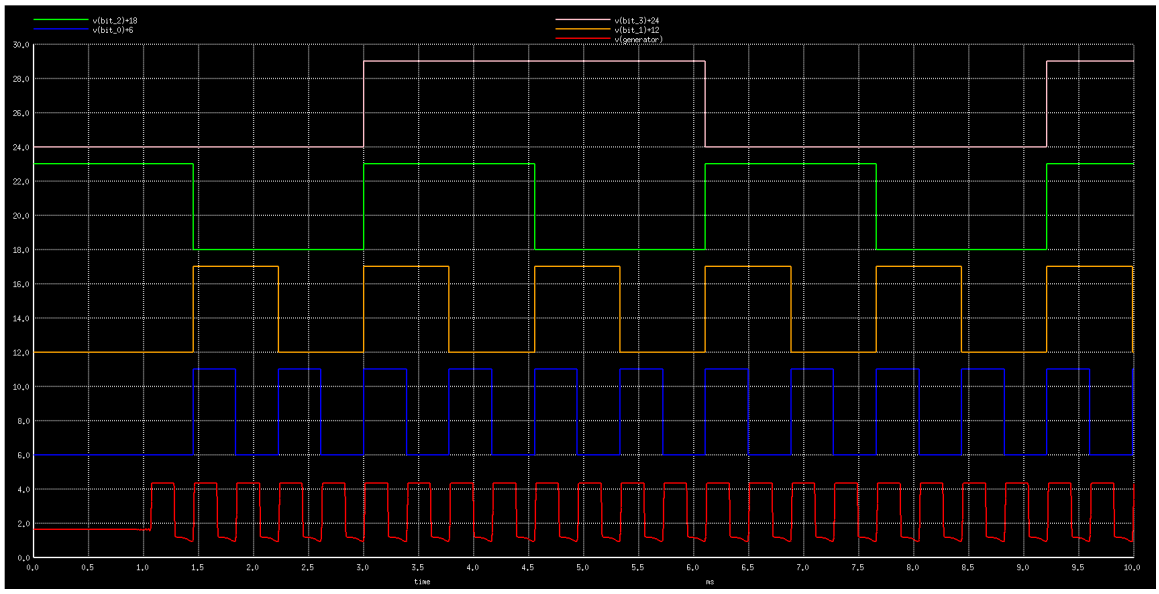
Counter

```
‘include "jk.v"
module counter(input CLK, output Q0, output Q1, output Q2, output Q3);

   jk jk1(.j(1), .k(1), .clk(CLK), .qbar(Q0));
   jk jk2(.j(1), .k(1), .clk(Q0), .qbar(Q1));
   jk jk3(.j(1), .k(1), .clk(Q1), .qbar(Q2));
   jk jk4(.j(1), .k(1), .clk(Q2), .qbar(Q3));


endmodule
```

JK Flip Flop

```
module jk ( input j, input k, input clk, output reg qbar);
   reg q;

   always @ (posedge clk)
      case ({j,k})
         2’b00 : q <= q;
         2’b01 : q <= 0;
         2’b10 : q <= 1;
         2’b11 : q <= ~q;
      endcase

   assign qbar = ~q;
endmodule
```

5.1.2 Output


v(bit_2)+18
v(bit_0)+6
v(bit_3)+24
v(bit_1)+12
v(generator)
30.0
28.0
26.0
24.0
22.0
20.0
18.0
16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0
time
ns

5.2 PISO Register with Astable Multivibrator

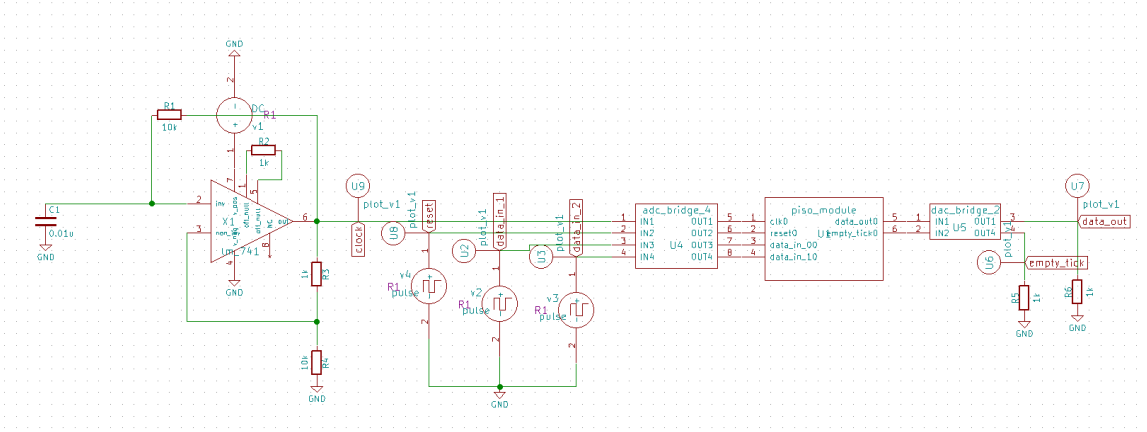


Figure 5.2: PISO Register with Astable Multivibrator

The Astable Multivibrator is also called a free-running multivibrator. It has two quasi-stable states and no external signal is required to produce the changes in state. The component values are used to decide the time for which circuit remains in each state. Usually, as the astable multivibrator oscillates between two states, is used to produce a square wave. In this circuit, the time period is dependent upon the value of the resistor and capacitor. It also depends upon the upper and lower threshold voltage of the op-amp[5].

In Parallel In Serial Out (PISO) shift registers, the data is loaded onto the register in the parallel format while it is retrieved from it serially. The parallel-in serial-out shift register stores data shifts on a clock-by-clock basis, and delays it by the number of stages times the clock period. In addition, parallel-in serial-out really means that we can load data in parallel into all stages before any shifting ever begins. This is a way to convert data from a parallel format to a serial format. By parallel format, we mean that the data bits are present simultaneously on individual wires, one for each data bit. By serial format, we mean that the data bits are presented sequentially in time on a single wire.

Here Astable Multivibrator is connected to the PISO register such that the clock signal generated by the Astable Multivibrator is used as the clock for the PISO register. The frequency of the clock signal generated by the Astable Multivibrator is 2.6kHz. The PISO register consists of 4 signals CLK, reset, data_in_0 and data_in_1. The reset signal is used to empty the register and reset it. data_in_0 and data_in_0 are the two data signals which take input parallelly as 2-bit input and convert them to serial data with the help of a clock. The PISO register consists of two outputs

data_out and empty_tick. The data_out outputs the data converted serially. While the empty tick is used as a signal that notifies that register has been emptied. It is triggered either when data is sent out or while the register gets a reset signal.

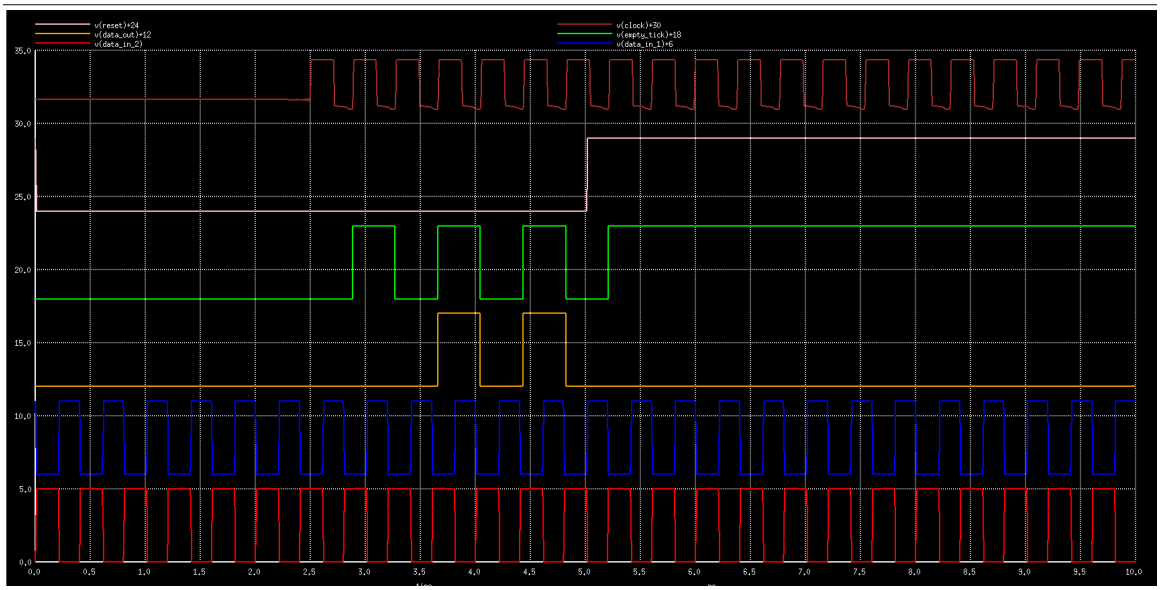
5.2.1 Code Snippets

PISO Module

```
module piso_module(
    input wire clk, reset, data_in_0, data_in_1,
    output reg data_out,
    output reg empty_tick
);

reg [1:0] data_reg, data_next;
reg [1:0] count_reg, count_next;
reg empty_reg, empty_next;

always @(posedge clk)
    empty_tick = empty_reg;

always @(posedge clk, posedge reset) begin
    if(reset) begin
        count_reg = 0;
        empty_reg = 1;
        data_reg = 0;
    end
    else begin
        count_reg = count_next;
        empty_reg = empty_next;
        data_reg = data_next;
    end
end

always @(*) begin
    count_next = count_reg;
    empty_next = empty_reg;
    data_next = data_reg;
    data_out = data_reg[count_reg];

    if (count_reg == 1) begin
        count_next = 0;
```

```
            empty_next = 1;
            data_next = {data_in_1, data_in_0};

        end
        else begin
            count_next = count_reg + 1;
            empty_next = 0;
        end
    end

    endmodule
```

5.2.2 Output



6 Attiny Circuit Examples

6.1 2:1 Multiplexer

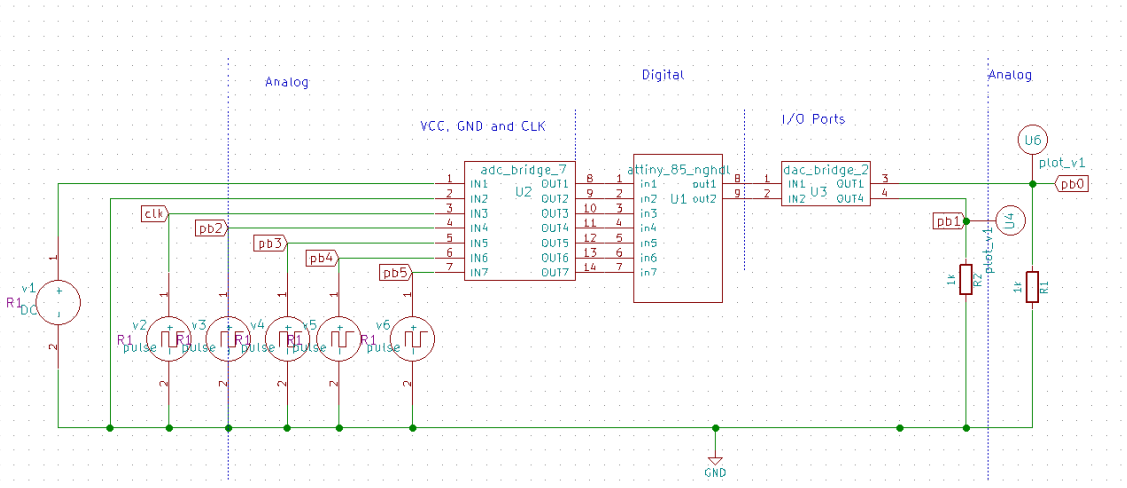


Figure 6.1: 2:1 Multiplexer Schematic

In electronics, a multiplexer also known as a data selector is a device that selects between several analogue or digital input signals and forwards the selected input to a single output line. The selection is directed by a separate set of digital inputs known as select lines. A multiplexer of 2^n inputs has n select lines, which are used to select which input line to send to the output.

Multiplexer makes it possible for several input signals to share one device or resource, for example, one analog-to-digital converter or one communications transmission medium, instead of having one device per input signal. Multiplexers can also be used to implement Boolean functions of multiple variables[6].

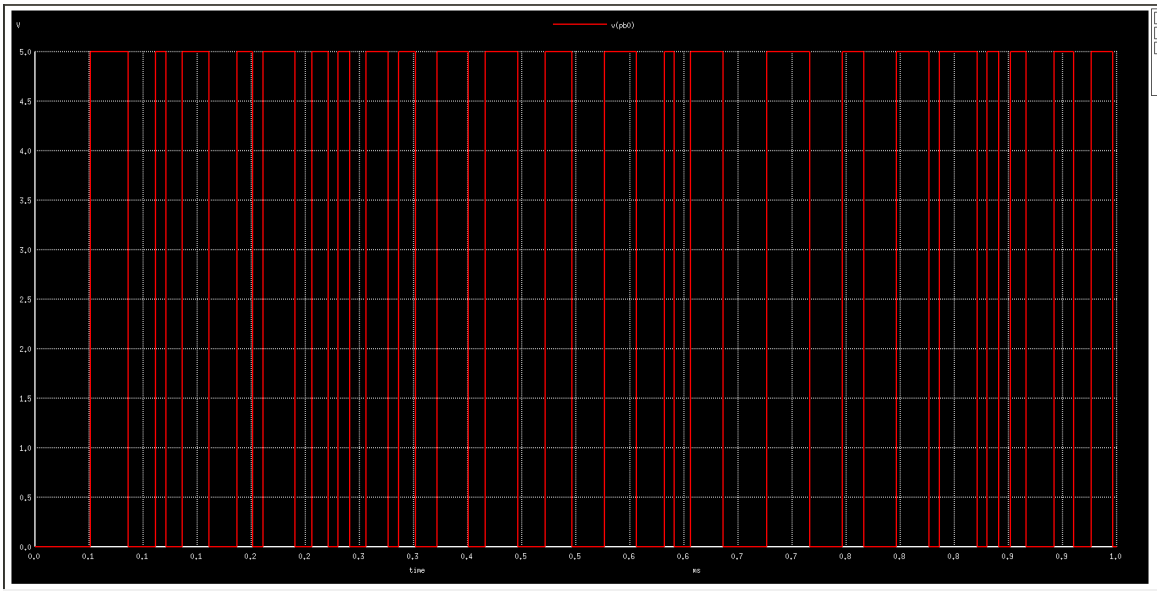
In this project, a 2:1 multiplexer is designed in C language. The written code is then compiled and a .hex file is obtained. This file contains all the necessary information needed by the microcontroller for appropriate functioning. This hex file is then uploaded onto the Attiny 85 microcontroller and then simulated using the eSim software.

6.1.1 Code Snippets

```
#include <avr/io.h>
```

```
#include <avr/sleep.h>
#include <math.h>
#define F_CPU 2.0E6
#include<util/delay.h>

int main() {

  DDRB |= 0x03;
  PORTB = 0x00;

  while(1){
    if(PINB2){
      if(PINB4){
        PORTB ^= 0x01;
      }else{
        PORTB ^= 0x01;
      }
    }else {
      if(PINB3){
        PORTB ^= 0x01;
      }else{
        PORTB ^= 0x01;
      }
    }
  }
}
```

6.1.2 Output


v(pb0)
V
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.1
0.1
0.1
0.2
0.2
0.3
0.3
0.4
0.5
0.5
0.6
0.6
0.7
0.7
0.8
0.8
0.8
0.9
0.9
1.0
time
ms

6.2 4-to-2 Priority Encoder

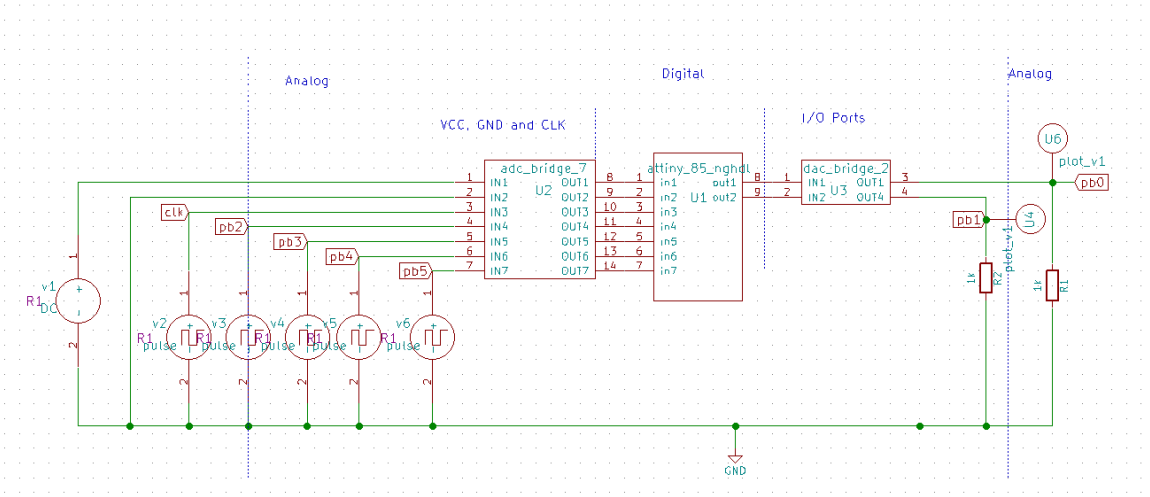


Figure 6.2: 4-to-2 Priority Encoder Schematic

The priority encoder is a combinational logic circuit that contains 2^n input lines and n output lines and represents the highest priority input among all the input lines. When multiple input lines are active high at the same time, then the input that has the highest priority is considered first to generate the output.

The output of this encoder corresponds to the input that has the highest priority. To obtain the output, only the input with the highest priority is considered by ignoring all other input lines. This is a type of binary encoder or an ordinary encoder with a priority function. The input that has the larger magnitude or highest priority is encoded first rather than other input lines. Hence, the generated output is based on the priority assigned to the inputs[7].

Truth table of the 4 Input Priority Encoder is below

Inputs				Outputs	
D3	D2	D1	D0	Q1	Q0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

From the above truth table, we can observe that D3, D2, D1, D0 are the inputs; Q1 and Q0 are the outputs. Here D3 input is the highest priority input and D0 is the lowest priority input.

When the input D3 is active high (1), which has the highest priority irrespective of all other input lines, then the output of the 4-bit priority encoder is 11.

When the D3 input is active low and the D2 is active high that has the next highest priority irrespective of all other input lines, then the output is Q1Q2=10.

When D3, D2 inputs are active low, and the D1 is active high and has the next highest priority regardless of the remaining input line, then the output will be Q1Q2 = 01

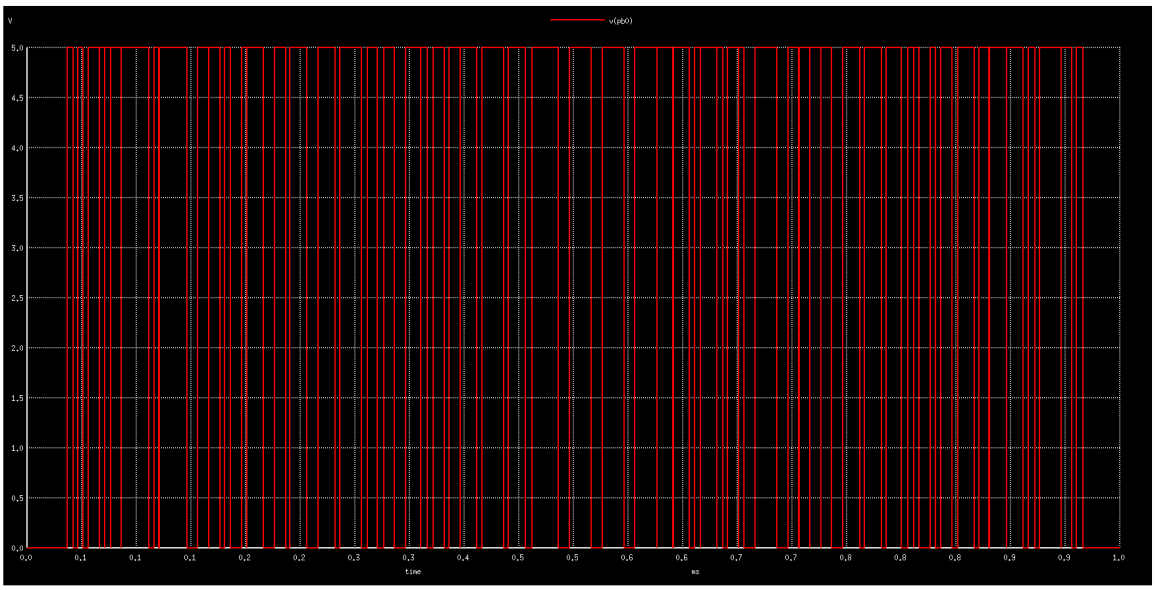
6.2.1 Code Snippets

```
#include <avr/io.h>
#include <avr/sleep.h>
#include <math.h>
#define F_CPU 2.0E6
#include<util/delay.h>

int main() {
  DDRB |= 0x03;
  PORTB = 0x00;

  while(1){
    PORTB = 0x00;
    if(PINB5){
        PORTB |= 0x01;
        PORTB |= 0x02;
    }else if(PINB4){
        PORTB &= ~0x01;
        PORTB |= 0x02;
    }else if(PINB3){
        PORTB |= 0x01;
        PORTB &= ~0x02;
    }else if(PINB2){
        PORTB &= ~0x01;
        PORTB &= ~0x02;
    }else{
```

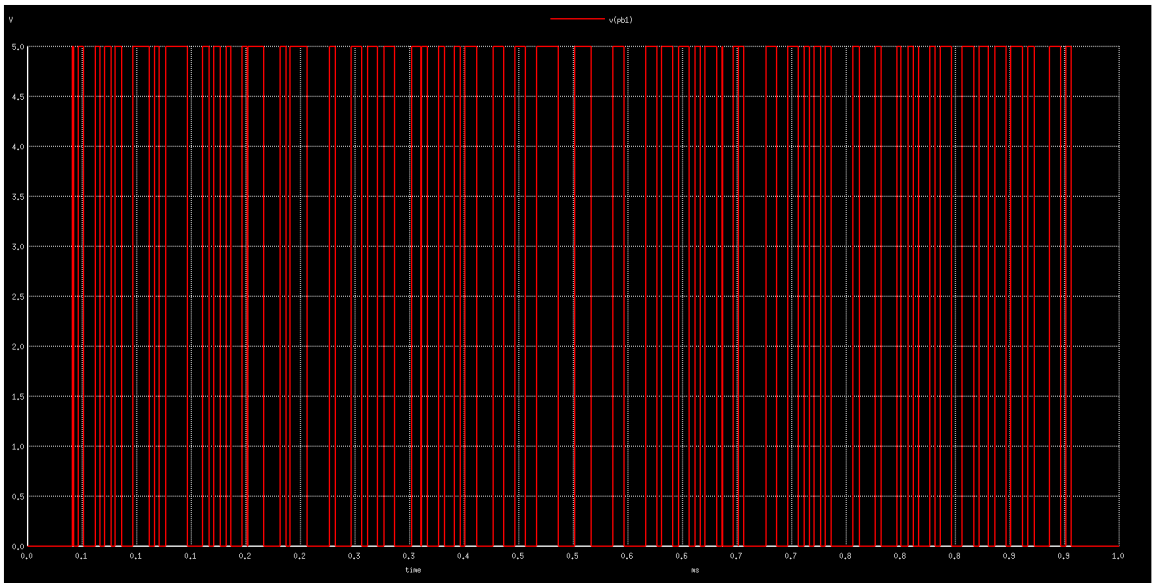
```
        PORTB &= ~0x01;
        PORTB &= ~0x02;
    }
  }
}
```

6.2.2 Output

Q0



Q1


V
v(pb1)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.1
0.1
0.1
0.2
0.2
0.3
0.3
0.4
0.5
0.5
0.6
0.6
0.7
0.7
0.8
0.8
0.8
0.9
0.9
1.0
time
ms

7 Other Tasks

7.1 Project Explorer

eSim has a project explorer on the left side of the main window comprising all the project folders and sub files in a tree format. It displays all the files and user can also open files from there. But the project explorer doesn't updates the project status automatically. If user deletes or edits a file outside the eSim i.e. from Operating System then eSim doesn't update the status of file until restart, else the user needs to refresh from the menu appearing on right click.

This problem has been solved by adding a function named **refreshInstant**. The refreshInstant() is triggered when QTreeWidget is expanded and then calls refreshProject() function.

When user clicks on any project, **refreshInstant** function is called that checks for the expanded item in the whole tree. It then calls **refresProject** function that checks for valid options and if filePath is available. It then deletes old values of that specific project and adds new and updated file names only. Here **indexItem** parameter is used to check if project folder is containing sub folders. Hence helping in deleting files names of subfolders too.

To see the code changes in details please click here.

7.2 Kicad To Ngspice GUI Changes

In the KicadToNgspice tab of eSim various sections are available for different type of circuits. In such sections there are various parameter field that can be edited by user before simulation. One such field is file path parameter. If user clicks on **Add File** button of this parameter, a file selection dialog box appears. On selecting file successfully the respective file path appears in the text field left to the button. But as the text field is editable, file path might be edited unintendedly by the user resulting in a simulation error due to wrong file path.

To resolve this issue text fields of file path **Subcircuit** and **Device Modelling** tabs have been made read-only. Hence user can select file but can't type it in the text field.

To see the code changes in details please click here.

Bibliography

[1] HEX File.
URL: https://www.engineersgarage.com/hex-file-format/

[2] Attiny Microcontrollers.
URL: https://www.microchip.com/wwwproducts/en/ATtiny85/

[3] Research Paper.
eSim: An Open Source EDA Tool for Mixed-Signal and Microcontroller Simulations

[4] GitHub Official Website.
URL: https://github.com/FOSSEE/nghdl

[5] Astable Multivibrator
URL: https://www.electronics-tutorials.ws/waveforms/astable.html

[6] Multiplexer
URL: https://en.wikipedia.org/wiki/Multiplexer

[7] Priority Encoder
URL: https://www.elprocus.com/priority-encoder