



# Summer Fellowship Report

On

**Implementation of GUI Interface for uploading hex files**

Submitted by

**Jay Mistry**

Under the guidance of

**Prof.Kannan M. Moudgalya**  
Chemical Engineering Department  
IIT Bombay

August 18, 2021

# Acknowledgment

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 opportunity of learning, being a part of such a wonderful project in such difficult pandemic situations, and also for enabling me for their help in offering me the resources and guiding me throughout the project.

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

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

Finally, I wish to thank my parents for their support and encouragement throughout my study.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	HEX files . . . . .	4
1.2	Importance of HEX files in microcontrollers . . . . .	4
1.3	eSim Microcontrollers . . . . .	5
<b>2</b>	<b>Current Scenario</b>	<b>6</b>
2.1	Uploading HEX files onto microcontroller in eSim . . . . .	6
2.2	Workflow . . . . .	6
2.2.1	DUTghdl folder . . . . .	7
<b>3</b>	<b>Problem Statement</b>	<b>8</b>
3.1	Problem . . . . .	8
<b>4</b>	<b>Solutions</b>	<b>9</b>
4.1	Solution - 1: User uploads directly the HEX file . . . . .	9
4.1.1	Introduction . . . . .	9
4.1.2	Workflow . . . . .	10
4.1.3	GUI Interface . . . . .	11
4.1.4	Code Snippets . . . . .	11
4.1.5	Advantages of this solution . . . . .	14
4.1.6	Disadvantages of this solution . . . . .	14
4.2	Solution - 2: User uploads/ creates .C file, compiles and uploads the HEX file . . . . .	14
4.2.1	Introduction . . . . .	14
4.2.2	Workflow . . . . .	15
4.2.3	GUI Interface . . . . .	16
4.2.4	Code Snippets . . . . .	17
4.2.5	Advantages of this solution . . . . .	18
4.2.6	Disadvantages of this solution . . . . .	18
<b>5</b>	<b>Implementation</b>	<b>19</b>
5.1	Comparing Solution 1 and Solution 2 . . . . .	19
<b>6</b>	<b>Conclusion</b>	<b>21</b>
<b>7</b>	<b>Future Work</b>	<b>22</b>

<b>8</b>	<b>Testing Microcontroller Circuits in Windows OS</b>	<b>23</b>
8.1	Introduction . . . . .	23
8.2	Microcontroller circuits in Linux OS . . . . .	23
8.3	Microcontroller circuits in Windows OS . . . . .	23
	8.3.1 Debugging . . . . .	24
8.4	Result . . . . .	25
8.5	Future Solution . . . . .	25
	<b>Bibliography</b>	<b>26</b>

# Chapter 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 might be 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 corresponding hex files. Now, these files are flashed/ dumped into the microcontrollers by 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, a HEX file content must be uploaded onto the microcontroller.

Since the microcontroller understands only the machine language which consists of zeroes and ones, it is difficult practically not possible for humans to write codes in such a manner. Hence, by using high-level languages we write down the code, and then using a compiler, the high-level language code is converted into machine language which is stored in the hex file format. A HEX file is a text file with an extension .hex. A HEX file contents may look like as shown in the fig. 1.1 as below.

```
:100000000EC015C014C013C012C011C010C00FC064
:100010000EC00DC00CC00BC00AC009C008C011241E
:100020001FBECFE5D2E0DEBFCDBF02D012C0E8CF09
:1000300081E087BB88BBE7EEF3E03197F1F700C0C2
:10004000000018BAEBEDF5E03197F1F700C00000C1
:06005000F1CFF894FFCF90
:000000001FF
```

Figure 1.1: HEX file contents

### 1.3 eSim Microcontrollers

At present eSim provides the Attiny series microcontrollers[2] . By 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. Hence, there is a need of uploading the HEX files onto the microcontroller. Once the HEX file is uploaded, the circuit can be successfully simulated.

# Chapter 2

## Current Scenario

### 2.1 Uploading HEX files onto microcontroller in eSim

As mentioned that eSim provides the Attiny series of microcontrollers, for simulating the circuits containing them, one has to upload the hex file onto the microcontroller, so that the circuit behaves and works properly.

For simulating the microcontrollers in eSim, one has to set up the NGHDL server[3] for the respective Attiny microcontroller instance which may include 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 in which the HEX content of the respective C code/ assembly code must be loaded.

On simulating the circuit on eSim, the contents of the hex.txt are fetched and according to it, 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 being followed for the same i.e., uploading the HEX file or the contents of the HEX file onto the microcontroller is manually copying the contents of the HEX file and pasting it into the hex.txt present in the DUTghdl folder.

The current workflow procedure that is carried out manually is shown in the below figure 2.1.

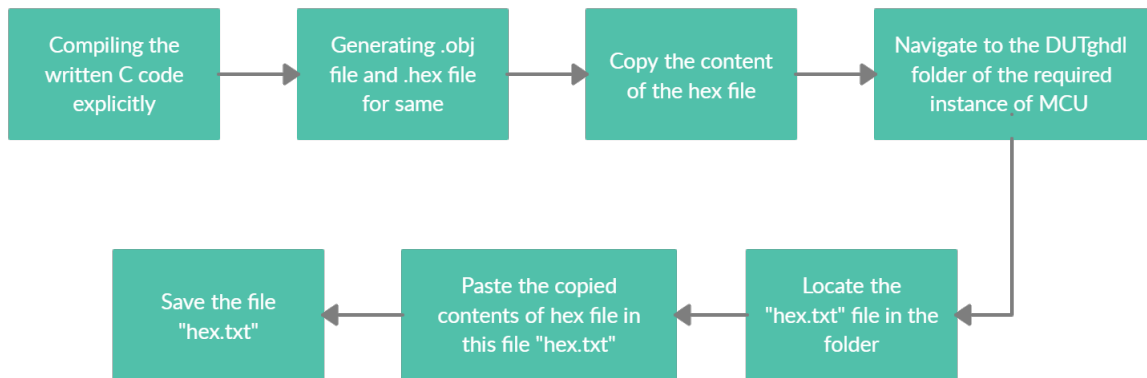


Figure 2.1: Current workflow

### 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 Attiny85 instance of microcontroller in Linux OS is:

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



# Chapter 3

## Problem Statement

### 3.1 Problem

As stated in the previous section 2.2, the current workflow for uploading the HEX file onto a microcontroller, which is tedious and has to be manually carried out.

There is a need of adding a feature into the eSim, where the user can be asked to upload the HEX file from within the software. The manual operation that is carried out, can lead to errors. Remembering the path for locating the "hex.txt" is also not feasible.

# Chapter 4

## Solutions

The solution for the problem stated in section 3.1., is an implementation of a GUI interface[5] for uploading the HEX files onto the microcontrollers. This feature can be added into the eSim, and when the circuit comprises any of the microcontrollers, the user will be asked to upload the HEX file associated with the respective circuit. Hence the manual process which is associated will be eliminated and the whole uploading process (the content of the HEX file) will become an automated task. Also, remembering the path of the respective DUTghdl folder will no longer be required.

Two solutions are proposed as mentioned below in section 4.1. and section 4.2. respectively.

### 4.1 Solution - 1: User uploads directly the HEX file

#### 4.1.1 Introduction

If any circuit comprises of any of the instances of the microcontroller, then for these circuits there will be an option to **add** and **upload** the HEX file. This feature is provided in the **KicadtoNgspice module**, under the **Ngspice Model** tab. Once the user finishes up with the circuit designing, on converting the current circuit to Ngspice Model, for setting up the various transient parameters, or the AC/DC parameters, the sources details, the user has to execute the KicadtoNgspice 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. needs to be entered by the user if required or else the default values are selected.

In this section, the uploading of HEX file is added. 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.

## 4.1.2 Workflow

The user will have to follow simple 3 steps:

1. Press Add Hex file button
2. Browse the HEX file
3. Press the Upload Hex file button

The workflow block diagram is shown in the below figure 4.1.

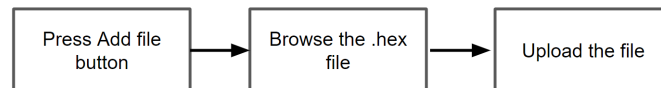


Figure 4.1: Adding HEX file directly - workflow

### Step 1: Press Add Hex file button

User on clicking the Add Hex file button as shown in figure 4.2, a window will be prompted allowing user to navigate through the computer to upload the HEX file for the microcontroller circuit. This will only show .hex files and no other files present in any of the directories. Once the user chooses the right .hex file for the circuit,

### Step 2: Browse the HEX file

On clicking the Add hex file button, the user will be prompted to browse the .hex file as shown in the figure 4.3 for the microcontroller as mentioned above. Once the user has selected the .hex file, the user will clicks on the open button present.

### Step 3: Upload the file

This process has two different methods listed below:

#### Method 1:

After selecting the appropriate .hex file for the microcontroller, the user will have to click the 'upload hex file' button present just beside the 'add hex file' button. After this, the contents of the selected .hex file will be pasted in the hex.txt file present in the DUTghdl folder.

#### Method 2:

User on clicking on the 'Upload hex file' button, a window will be prompted as shown in the figure 4.3, which will directly open in the ghdl folder, from where the differnt instances of microcontrollers like Attiny25/45/85 folders will be present.

This method will make the completely dynamic and allow the user to choose the respective Attiny microcontroller DUTghdl folder, rather than explicitly hard coding the paths for same as done in method 1.

On choosing the respective DUTghdl folder and opening the same, the code will copy the contents of the .hex file selected to the respective selected DUTghdl folder's 'hex.txt' file.

### 4.1.3 GUI Interface

The following figure shows the GUI interface developed for the solution 1 is shown in the figure 4.2.

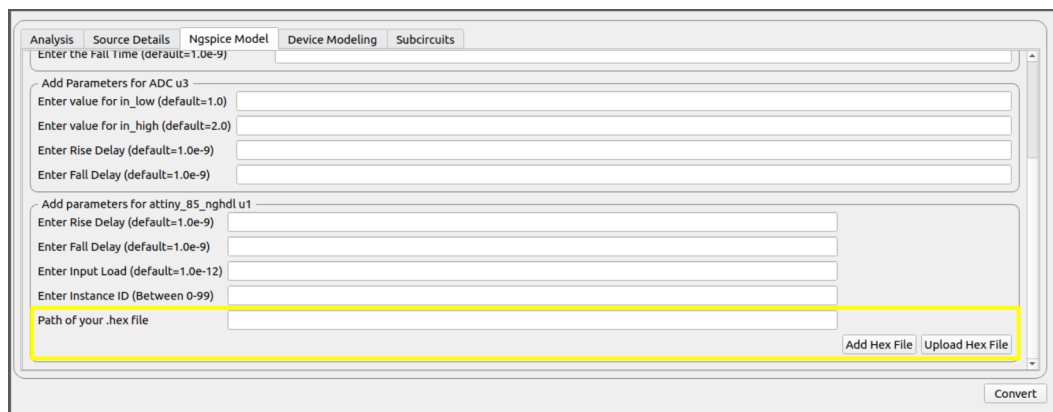


Figure 4.2: GUI interface for solution 1 under Ngspice model tab

### 4.1.4 Code Snippets

The definition for the 'Add hex file' button defined under the 'Model.py' file is:

---

```
def addHex(self):
    """
    This function is use to keep track of all Device Model widget
    """
    #print("Calling Track Device Model Library funtion")
    init_path = '../..../'
```

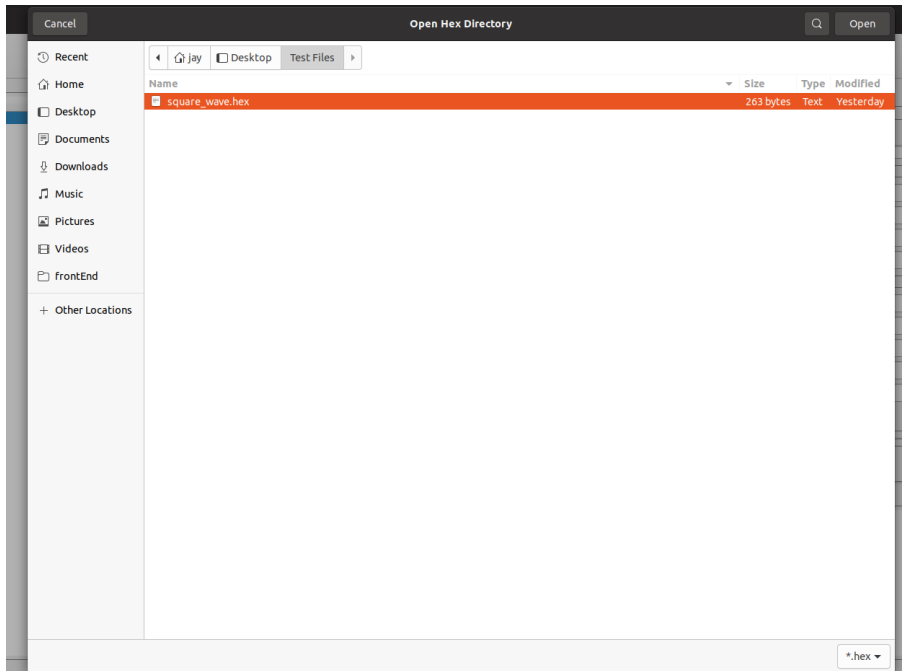


Figure 4.3: Window prompted on clicking 'Add hex file' button

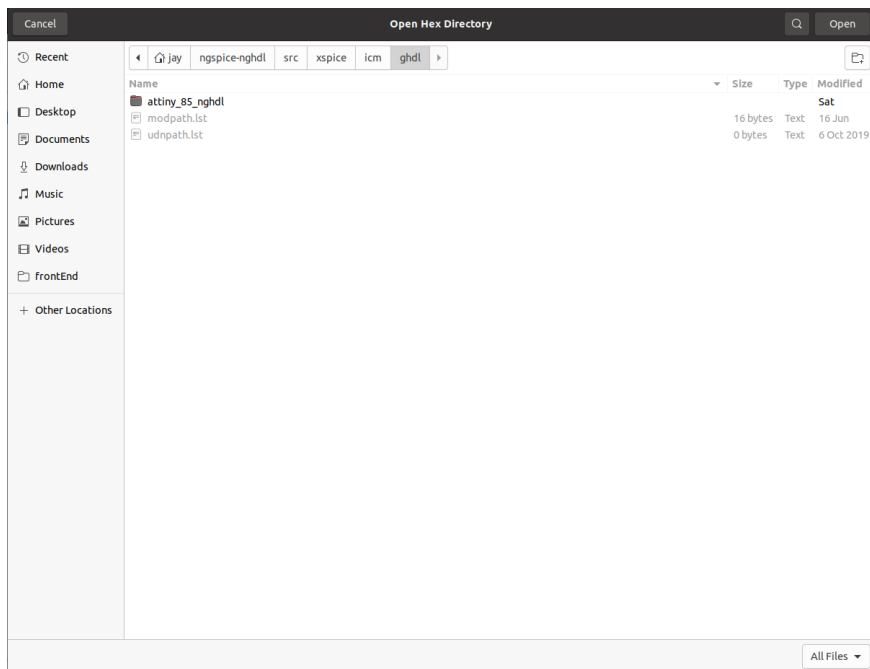


Figure 4.4: On clicking 'Upload hex file' button for method2 stated in 4.1.2 step 3

```

if os.name == 'nt':
    init_path = ''
self.hexfile = QtCore.QDir.toNativeSeparators(
    QtWidgets.QFileDialog.getOpenFileName(
        self, "Open Hex Directory",
        init_path + "home", "*.hex"
    )[0]
)
self.text = open(self.hexfile).read()
chosen_file_path = os.path.abspath(self.hexfile)

```

---

The definition for the 'Upload hex file' button following the method 1 as stated above under section 4.1.2 step 3:

```

def uploadHex(self):
    """
    This function is use to keep track of all Device Model widget
    """
    #print("Calling Track Device Model Library funtion")
    #path for different instances can be added.
    path1 = os.path.expanduser('~')
    path2 =
    ↪ "/ngspice-nghdl/src/xspice/icm/ghdl/attiny_85_nghdl/DUTghdl/hex.txt"
    path = path1 + path2 #final path for Attiny5 DUTghdl folder
    self.file=open(path,'w')
    self.file.write(self.text)
    self.file.close()

```

---

The definition for the 'Upload hex file' button following the method 2 as stated above under section 4.1.2 step 3:

```

def uploadHex(self):
    """
    This function is use to keep track of all Device Model widget
    """
    #print("Calling Track Device Model Library funtion")

    init_path = '/home/jay/ngspice-nghdl/src/xspice/icm/ghdl'
    if os.name == 'nt':
        init_path = ''

    self.hexloc = QtWidgets.QFileDialog.getExistingDirectory(
        self, "Open Hex Directory",
        init_path)
    print(self.hexloc)

```

```
self.file = open(self.hexloc+"/hex.txt", "w")
self.file.write(self.text)
self.file.close()
```

---

### 4.1.5 Advantages of this solution

The following are the advantages of this solution:

1. The manual operation of uploading HEX file content eliminated.
2. Dynamic i.e., when the circuit contains microcontroller instance, the HEX file upload parameter is enabled.
3. User-friendly GUI for uploading HEX file.

### 4.1.6 Disadvantages of this solution

The following are the disadvantages of this solution:

1. Compiling the C code explicitly.
2. Generating the HEX file explicitly.

## 4.2 Solution - 2: User uploads/ creates .C file, compiles and uploads the HEX file

### 4.2.1 Introduction

This solution creates a model which can be referred to as "hexgen" model. Similar to the KicadtoNgpice module, simulation, NGHDL, etc. one extra module for generation and uploading of HEX file is added.

Hence when the circuit contains any instance of microcontroller, the user will be accessing this module for uploading the HEX file onto the microcontroller. This model will not only be limited to uploading direct HEX files, but this will allow users to upload the corresponding C program files, or create a new file having extension .C. Write or edit the code from the editor space provided on the GUI. Once the user finishes up with the uploading / creating .C file and editing same, the user needs to press the upload button.

On pressing the upload button present on the GUI the process for the user will be finished, provided that the user has upload error-free code.

In the backend, the uploaded .C file will be compiled with avr-gcc compiler (as of now limited to Attiny series of microcontrollers), generate the .obj file and .hex file for same. Copy the content of the generated .hex file into the required DUTghdl folder's "hex.txt" file. Once the contents are pasted, the generated .obj file, and the .hex file will be removed.

## 4.2.2 Workflow

The user will have to follow the below mentioned steps:

1. Open the .c to hex converter module or the hexgen model
2. Browse the required .C file from the computer and open
3. Edit the uploaded .C file if needed
4. Press the upload button to upload the HEX file

The workflow block diagram is shown in the below figure

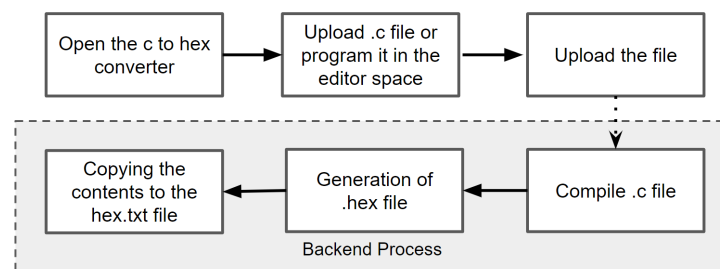


Figure 4.5: hexgen model - workflow

### Step 1: Open the .c to hex converter or hexgen model

The user can open the hexgen model from the main menu/ welcome screen of the eSim.

### Step 2: Browsing the .C file

The user will browse the .C file from the prompted window which is shown on pressing the browse button present on the GUI. Once the file is selected, the contents of the file will be shown in the editor block of the GUI as shown in the figure 4.7.

### Step 3: Edit the code if require

As mentioned above on choosing the file, the contents of the file is shown in the editor block. The user can edit the code if require which is shown in the figure 4.7.

### Step 4: Upload the file

Once the file is browsed, and the contents of the file are checked, the user will press the upload button.



## Backend Process

After the user clicks the upload button present on the GUI as shown in the figure. The following backend processes takes place:

- Compilation of the C code using avr-gcc compiler.
- Calling the shell script, which has automated the task of generating the .obj and .hex files.
- After generation of .hex file, copy the content of the .hex file from the local directory.
- Change the directory to the respective DUTghdl folder, locate the hex.txt file.
- Pastes the content inside the hex.txt file.
- Deletes the generated .obj file and .hex file present which were generated in step 3, in the specified directory.

### 4.2.3 GUI Interface

The following figures show the GUI Interface developed for the solution 2 is shown in the figure 4.6.

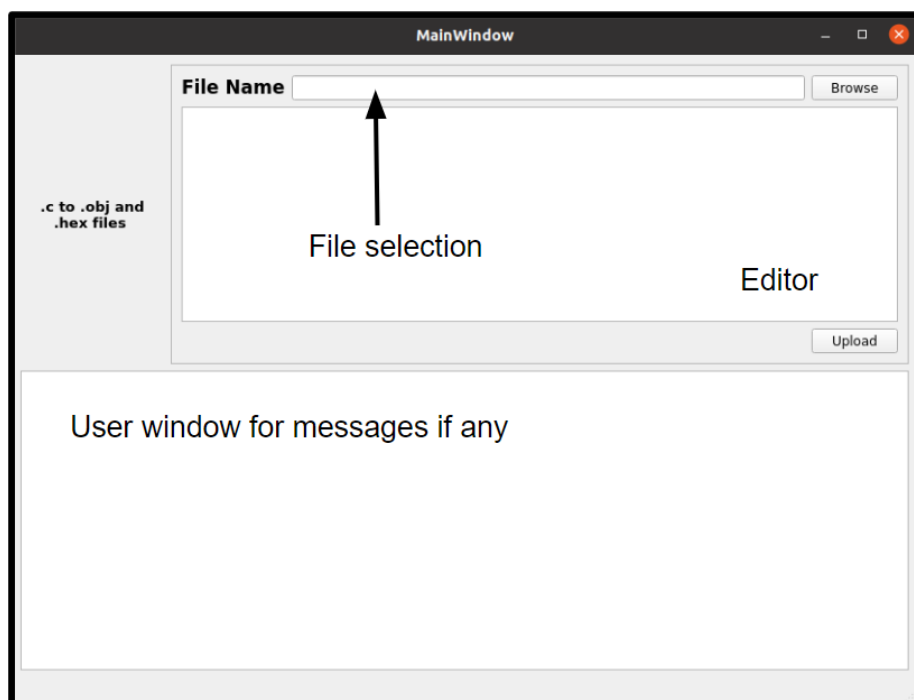


Figure 4.6: The GUI Interface for hexgen model

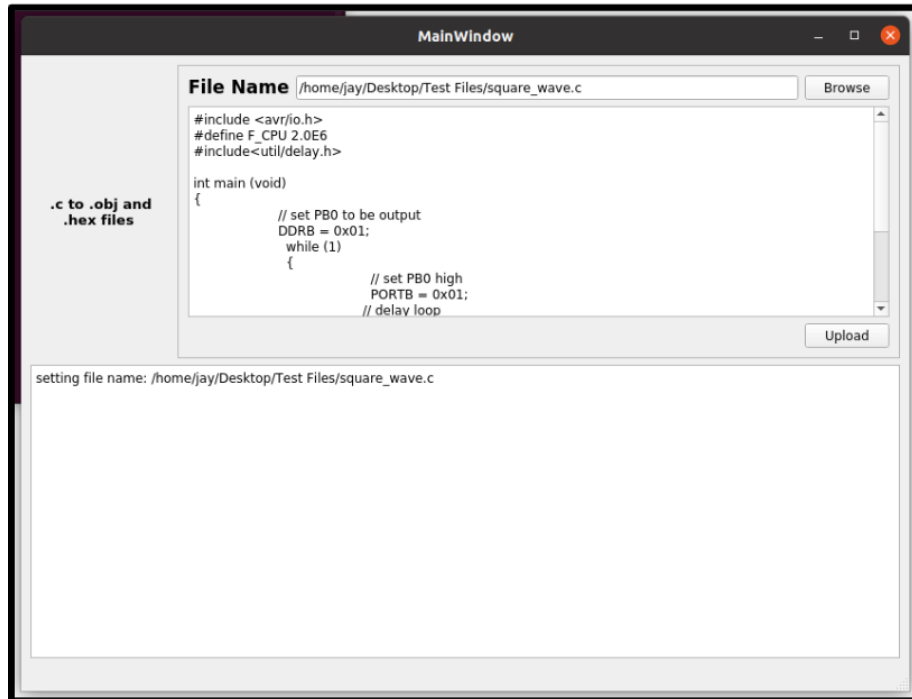


Figure 4.7: C code file uploaded, editable in editor space

#### 4.2.4 Code Snippets

The following is the code for the shell scripting part present in "script.sh" file.

The above script is called when the "upload" button is pressed. The python code for definition of upload button press is shown below which is present in the model.py:

---

```
def writeDoc( self, text ):
    ''' definition for upload button
    '''
    if self.isValid( self.fileName ):
        base = os.path.basename(self.fileName)
        fileName = os.path.splitext(base)[0]
        path = os.path.split(self.fileName)
        attiny_85_path =
        ↪  "/home/jay/ngspice-nghdl/src/xspice/icm/ghdl/attiny_85_nghdl/DUTghdl"
        '''
        this path can be passed to the shell script
        also for future when multiple different MCUs are implemented the path
        ↪  can be defined
        '''
        subprocess.call(['bash', 'script.sh', path[0], fileName])
```

---

### **4.2.5 Advantages of this solution**

The following are the advantages of this solution:

1. The manual operation of uploading HEX file content eliminated.
2. User-friendly to code and upload the hex file.
3. No need of explicit compilation and generation of hex files.
4. User can edit the file that has been selected.

### **4.2.6 Disadvantages of this solution**

The following are the disadvantages of this solution:

1. Need of avr-gcc compiler being installed on the system, or else should be included in eSim installation package.
2. Compiler dependency, creates problem for showing up the errors if any in code on the GUI.
3. Difficult to implement same logic in Windows OS.
4. The size of the overall software eSim increases.

# Chapter 5

## Implementation

As mentioned in the previous section about the solutions for the problem of HEX file content to be uploaded, solution 1 has been implemented as of now.

### 5.1 Comparing Solution 1 and Solution 2

As the advantages and disadvantages associated with each of the solutions i.e. solution 1 and solution 2 mentioned above are stated in sections 4.1.5, 4.1.6, and 4.2.5, 4.2.6 respectively.

On comparing both the solutions, solution 1 has been selected and implemented as of now with eSim for HEX file content upload. The reasons behind selecting solution 1 over solution 2 are:

1. No dependency of any compiler in solution 1, whereas dependency of the compiler in solution 2.
2. Solution 1 can also work in Windows OS, whereas the stated solution 2 can only work in Linux OS because of the avr-gcc compiler being used, again alternate compilers for Windows can be explored and Implemented. But this will again make compiler dependent.
3. Considering the compilation feature to be a part of the software increases the demand of making our GUI more dynamic and user-friendly which includes the interface or the model to be powerful for showing the errors if any in the code.
4. As of now, the eSim software provides only the Attiny microcontrollers, but when in the future this is expanded to other types of microcontrollers, the need for different compilers and compilation processes will arise. This can be managed, however, the size of the software gets affected.
5. Since, the eSim software is meant for the students and the size of the software being one of the most important factors, implementing solution 2 will completely increase the size of the software as compared to solution 1.

Hence, due to the above-mentioned reasons solution 1 as of now is implemented, to make the manual process of uploading the HEX file content to the required folder. In the future, this can be improved, and even the compilation part can be added.

# Chapter 6

## Conclusion

Solution 1 - the direct uploading of HEX files onto the microcontrollers by implementing the GUI interface for same dynamically is carried out.

The user will have to only generate the HEX file by using any external compiler, online HEX file generation tool, or by Arduino IDE, etc. Once the HEX file is ready, the user only has to upload the same through the eSim software without worrying about the path of the DUTghdl folder or the target folder. Hence the problem stated above which was carried out manually has been automated.

However, kindly note that the situation of having a multiple instance of the same microcontroller in one circuit has not be addressed as of now. The current solution provided for uploading of the hex files will be applicable to the multiple instances of same microcontroller if used in a circuit. But the problem being faced is about the NGHDL server being set up.

# Chapter 7

## Future Work

Solution 2 stated in section 4.2 can be improvised and implemented in the future. By adding the feature of the compilation within the software, makes it more user-friendly and reliable for users to use it.

Similarly, for Windows OS other alternatives can be explored few of which is by using the Arduino IDE. More exploration and research for the same can be carried out and implemented keeping in mind more microcontrollers that will be added to the system.

As mentioned in the conclusion section about the multiple instance of the same microcontroller in the same circuit, the respective microcontroller folder in present in the DUTghdl folder must contain multiple hex.txt files. This would be equal to the number of microcontrollers of that particular instance being used. Hence while uploading hex file to the microcontroller, the user than can select the respective hex.txt for it. This solution can be implemented instead of having two different folders for the same microcontroller in the DUTghdl folder.

# Chapter 8

## Testing Microcontroller Circuits in Windows OS

### 8.1 Introduction

The Attiny microcontrollers implemented in the eSim software work properly in Linux OS[4]. These microcontrollers need to be tested in Windows OS. The circuits which are already designed for testing the microcontrollers implementation which includes square wave generation, triangular wave generation, etc. need to be tested in the Windows OS.

### 8.2 Microcontroller circuits in Linux OS

The circuits used for testing are the square wave generation and the triangular wave generation circuits [7] available on the github repo of eSim[6]. To simulate the microcontroller circuits in eSim, the NGHDL server needs to be set up for the respective instance of the microcontroller (Attiny25/45/85). Once it is done, the hex.txt content must be updated with the testing code as per the requirement. The results for the simulation were obtained in Linux for microcontroller circuits.

### 8.3 Microcontroller circuits in Windows OS

To simulate the microcontroller circuits in eSim, the NGHDL server has to be set up as mentioned for the respective microcontroller instance. This can be set up by uploading the VHDL file for the required microcontroller by using the NGHDL feature in the eSim.

The "attiny85nghdl.vhdl" file was uploaded in the NGHDL server successfully. The "startserver.sh", "tiny85c.c", and "ghdlaccess.vhdl" files were copied to the DUT-ghdl folder created in the ngspice-ghdl folder. The hex.txt was copied and the contents of same were updated as per the circuit.



### 8.3.1 Debugging

On simulating the circuit, an error occurred which stated:

This error snippet is shown in the below figure 8.1. The above mentioned error

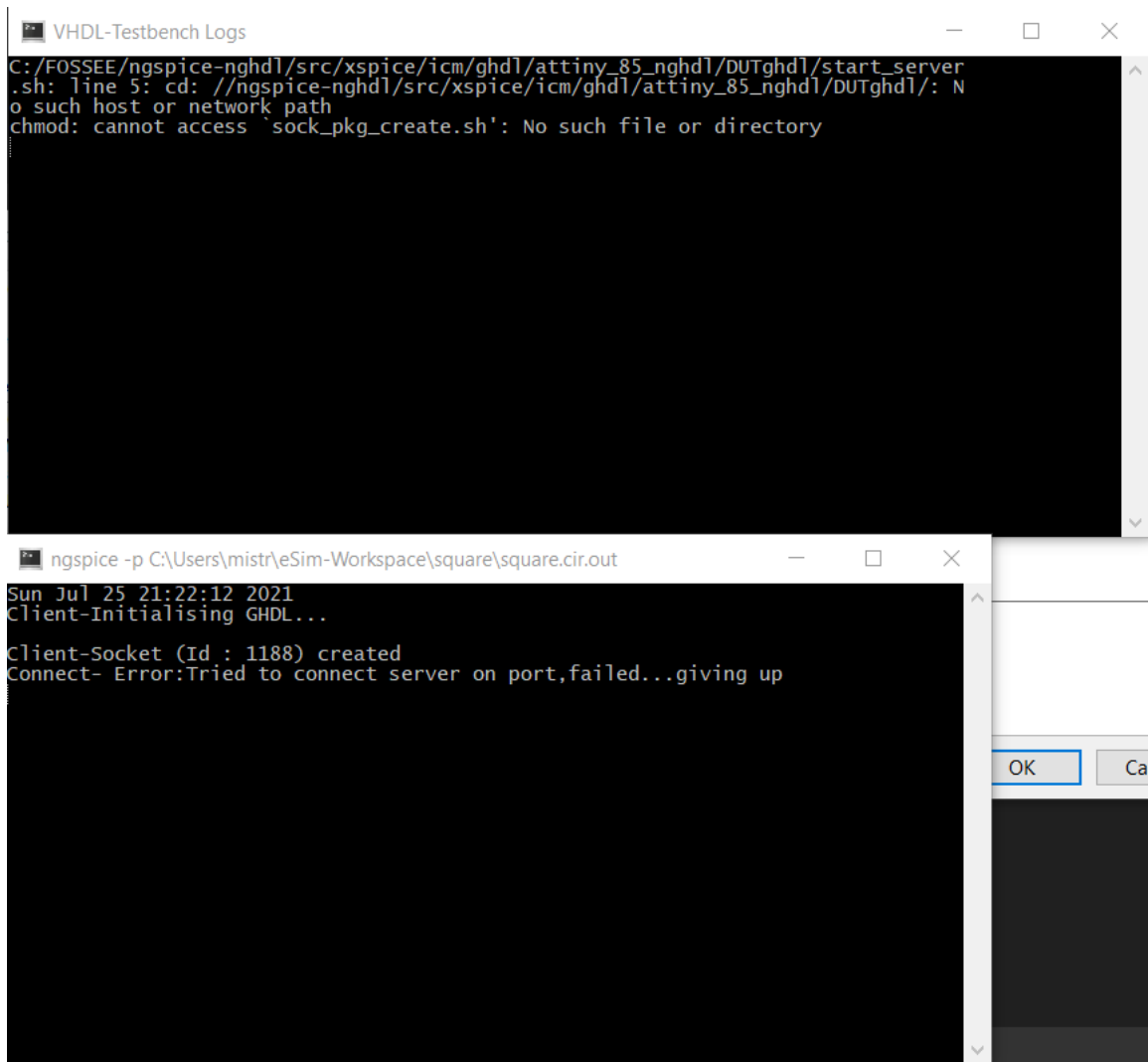


Figure 8.1: Error while setting up NGHDL server for Attiny 85 in Windows OS

is due to the wrong path selection. The path for my installed eSim is:

The startserver.sh file gives command of changing the directory to the path which is correct for the Linux OS, but not always valid in Windows OS.

The startserver.sh file contains:

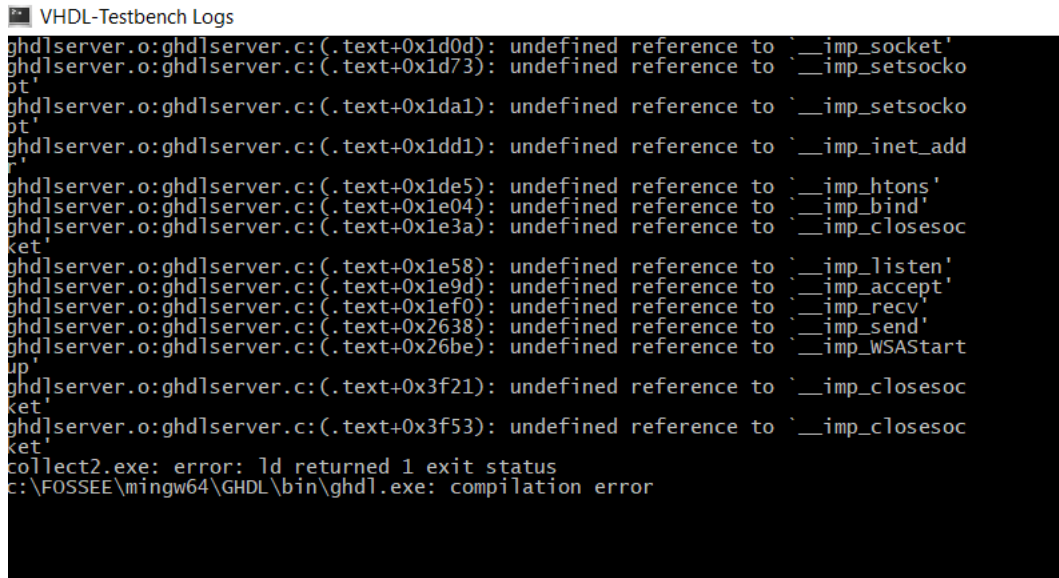
As from above it can be seen that the third line command is causing the error as mentioned above.

Hence, by changing this command and providing the right path, the error can be resolved.

This command for debugging purpose, I changed to as:

The above mentioned error of not able to locate any such file or directory got eliminated, but another error came while setting up the server. This error was related ghdlserver.o:ghdlserver.c in which undefined reference being made was alerted.

The snippet for the error is:



```
VHDL-Testbench Logs
ghdlserver.o:ghdlserver.c:(.text+0x1d0d): undefined reference to `__imp_socket'
ghdlserver.o:ghdlserver.c:(.text+0x1d73): undefined reference to `__imp_setsockopt'
ghdlserver.o:ghdlserver.c:(.text+0x1da1): undefined reference to `__imp_setsockopt'
ghdlserver.o:ghdlserver.c:(.text+0x1dd1): undefined reference to `__imp_inet_addr'
ghdlserver.o:ghdlserver.c:(.text+0x1de5): undefined reference to `__imp_htons'
ghdlserver.o:ghdlserver.c:(.text+0x1e04): undefined reference to `__imp_bind'
ghdlserver.o:ghdlserver.c:(.text+0x1e3a): undefined reference to `__imp_closesocket'
ghdlserver.o:ghdlserver.c:(.text+0x1e58): undefined reference to `__imp_listen'
ghdlserver.o:ghdlserver.c:(.text+0x1e9d): undefined reference to `__imp_accept'
ghdlserver.o:ghdlserver.c:(.text+0x1ef0): undefined reference to `__imp_recv'
ghdlserver.o:ghdlserver.c:(.text+0x2638): undefined reference to `__imp_send'
ghdlserver.o:ghdlserver.c:(.text+0x26be): undefined reference to `__imp_WSASocket'
ghdlserver.o:ghdlserver.c:(.text+0x3f21): undefined reference to `__imp_closesocket'
ghdlserver.o:ghdlserver.c:(.text+0x3f53): undefined reference to `__imp_closesocket'
collect2.exe: error: ld returned 1 exit status
c:\FOSSEE\mingw64\GHDL\bin\ghdl.exe: compilation error
```

Figure 8.2: ghdlserver error while setting up the server

## 8.4 Result

The NGHDL server for the microcontroller was not set up due to the above-mentioned errors. Hence, the microcontroller circuits as mentioned which were readily simulated in Linux OS, were not able to be simulated in Windows OS.

## 8.5 Future Solution

The path related problems in Windows will be encountered often. Hence, the path of the installed eSim must be obtained and for Windows the microcontroller implementation must be changed so that the correct path is selected.

# 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] PyQt5 Module.  
URL: <https://pypi.org/project/PyQt5/>
  
- [6] Microcontroller Testing Circuits.  
URL: <https://github.com/FOSSEE/nghdl/tree/attiny-alpha/Attiny>
  
- [7] Square wave and triangular wave examples.  
URL : <https://github.com/FOSSEE/nghdl/tree/attiny-alpha/Attiny/ATtiny85/Examples>