

MIC-1

ECE20 Project

WS22

Inhaltsverzeichnis

1	Hardware to exchange programmes	1
1.1	Memory-Manager	1
1.1.1	Task	1
1.1.2	Schematic	2
1.1.3	Code	3
1.2	UART module and multiplexing the UART interfaces	4
1.2.1	Task	4
1.2.2	Schematic / Block-design	5
1.2.3	Code	6
1.3	Sending Data via UART (Data Syntax)	7
2	VGA	9
2.1	Task	9
2.2	Original Design	9
2.2.1	State Machine	10
2.3	New Block Design	10
2.3.1	State Machine	11
2.3.2	Static Screen	12
3	GUI	13
3.0.1	GUI	13
3.0.2	Protocol	14
3.0.3	Code	18
3.0.4	Send data	23
3.0.5	Read data	24
A	VGA code	25
A.1	Original VGA Controller code	25
A.2	New VGA Controller code	29
A.3	Python script for making the Static Screen file	36
A.4	ScreenBufferMem module code	38

1 Hardware to exchange programmes

By generating the bitstream of the top-level design a program will be stored in the Main-Memory of the MIC-1. This program can be defined inside *SystemVerilog/defines_add.sv*.

In order to exchange to program which is stored inside the Main-Memory a Memory-Manager, UART-Register and a additional Multiplexer was designed.

1.1 Memory-Manager

The purpose of the Memory-Manager is to switch the connection of the main-memory from the MIC-1 to the UART-Register.

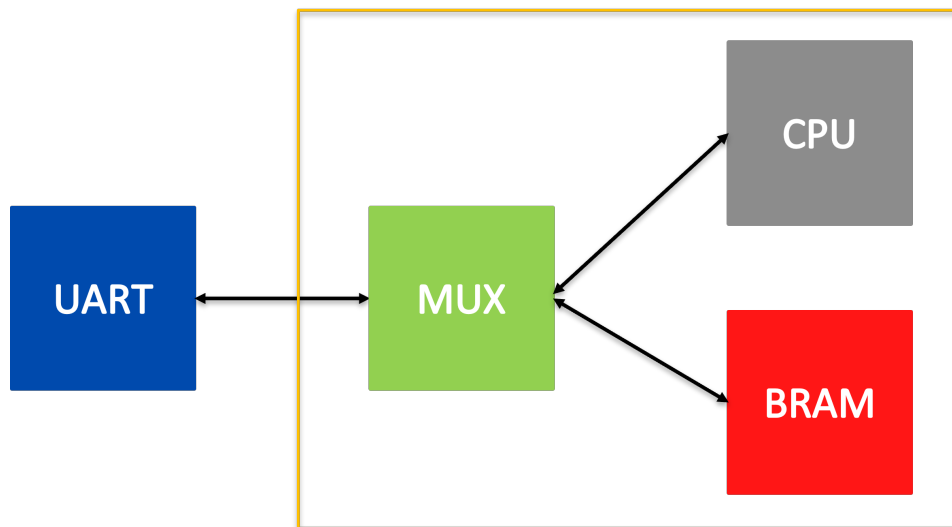


Abbildung 1.1: Blockdiagram of the new connection

1.1.1 Task

First it was needed to create a multiplexer, the corresponding code can be seen in the code section. This module was later than integrated in the *mic1_soc*". For the integration of the module we needed to break up the old connections from the main-memory to the *mic1*". The new inputs from the Uart and the outputs of the *mic1*" were assigned as inputs of the memory-manager and the outputs of the memory-manager assigned input of the main-memory. By this we can switch the inputs of the main-memory" between the *mic1* and *uart* and overwrite the existing program.

1.1.2 Schematic

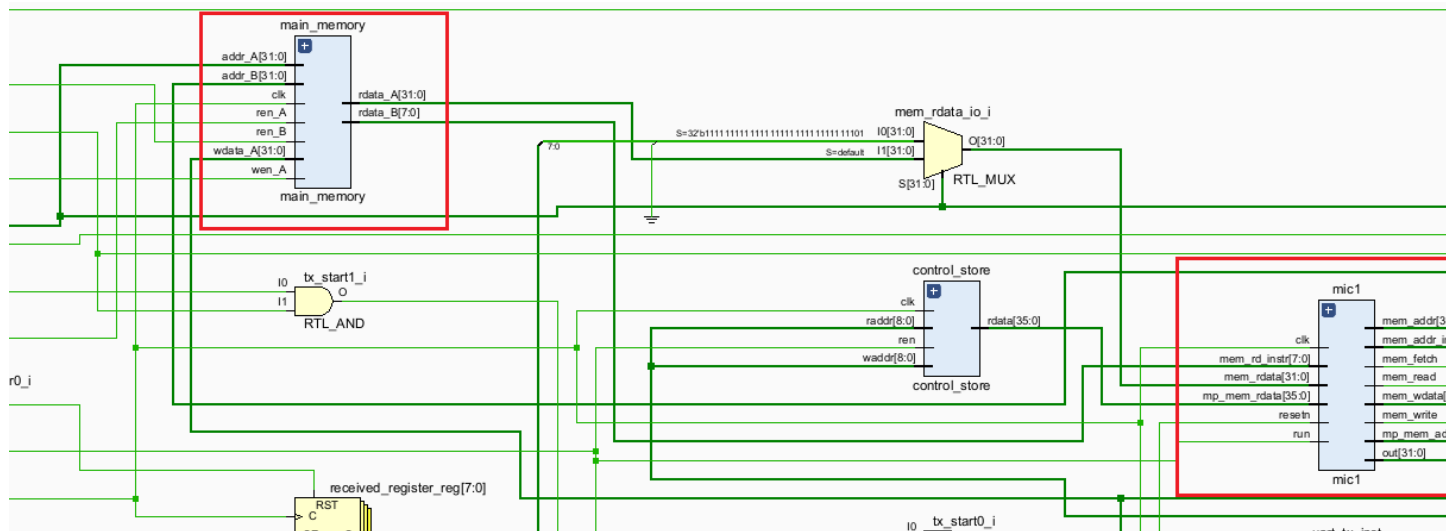


Abbildung 1.2: Schematic of the old connection

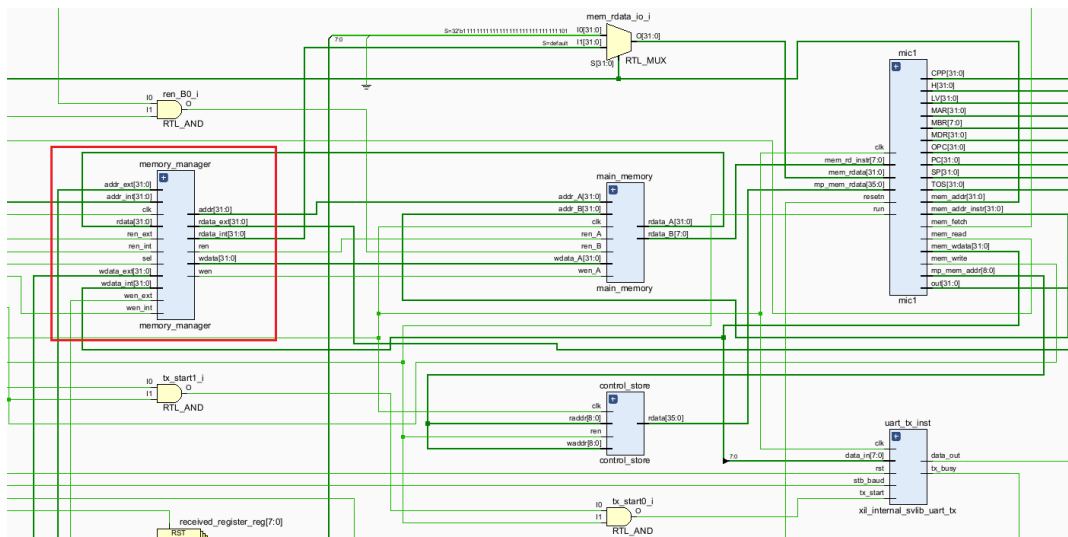


Abbildung 1.3: Schematic of the new connection

1.1.3 Code

Listing 1.1: Memory-Manager.

```
1 module memory_manager(  
2     input logic clk, sel,  
3     input logic wen_int, ren_int,  
4     input logic wen_ext, ren_ext,  
5     input logic [31:0] addr_int, wdata_int,  
6     input logic [31:0] addr_ext, wdata_ext,  
7     input logic [31:0] rdata,  
8     output logic wen, ren,  
9     output logic [31:0] addr, wdata, rdata_int, rdata_ext  
10 );  
11  
12 always_comb begin  
13     case (sel)  
14         1'b0:  
15             begin  
16                 wen = wen_int;  
17                 ren = ren_int;  
18                 addr = addr_int;  
19                 wdata = wdata_int;  
20                 rdata_int = rdata;  
21                 rdata_ext = 'h00000000;  
22             end  
23         1'b1:  
24             begin  
25                 wen = wen_ext;  
26                 ren = ren_ext;  
27                 addr = addr_ext;  
28                 wdata = wdata_ext;  
29                 rdata_int = 'h00000000;  
30                 rdata_ext = rdata;  
31             end  
32     endcase  
33 end  
34 endmodule
```

Listing 1.2: Integration of the Memory-Manager in the MIC-1 SOC.

```
1     memory_manager#(  
2     )memory_manager (  
3         .clk      (clk),  
4         .sel      (sel_uart),  
5         .wen_int  (mem_write && mic1_run),  
6         .ren_int  (mem_read  && mic1_run),  
7         .addr_int (mem_addr),  
8         .wdata_int (mem_wdata),  
9         .wen_ext  (w_en),  
10        .ren_ext  (r_en),  
11        .addr_ext (w_address),  
12        .wdata_ext (w_data),  
13        .rdata_int (mem_rdata),  
14        .rdata_ext (r_data),  
15        .wen      (mm_wen),  
16        .ren      (mm_ren),  
17        .addr     (mm_addr),  
18        .wdata    (mm_wdata),  
19        .rdata    (mm_rdata)  
20    );  
21 endmodule
```

1.2 UART module and multiplexing the UART interfaces

1.2.1 Task

The already existing implementation of the MIC-1 processor, shall be extended with a UART-Interface, which enables to write data directly into the the registers of its BRAM. In the beginning the UART IP-Core was tested with a Virtual I/O block, to get familiar with its functionality. From this point the Interface was successively extended with a BRAM Block, to simulate the memory of the microprocessor. At this stage the implementation was ready to be tested with the GUI's interface. In the last process the virtual components were removed and connected with the existing top level design. A system analyzer could show the output at this moment, which was eventually replaced with the VGA Interface.

Previously the program to be run on the microprocessor was generated within the Bitstream. With the addition of the UART Module a new microprogramm can be sent to the Block Ram. Since the input data to the MBR (Memory Buffer register) and the UART Module use the same UART interface a multiplexer has to be added, to avoid collision of data. This multiplexer derives whether data shall be sent to the memory or straight to the MIC's ALU via the MBR and allows the communication to strictly either and keeps the other on IDLE (HIGH = "d1"). The switch is mapped to a hardware switch on the Basys3 board (T1) and well as the TX and RX are mapped to hardware ports (TX = B18, RX = A18). The clock was connected to the synchronized system clock.

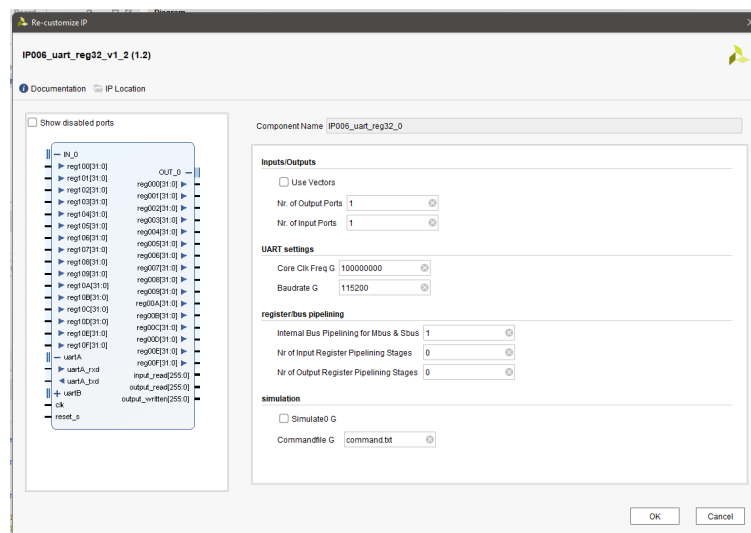


Abbildung 1.4: Addressing of the UART module

1.2.2 Schematic / Block-design

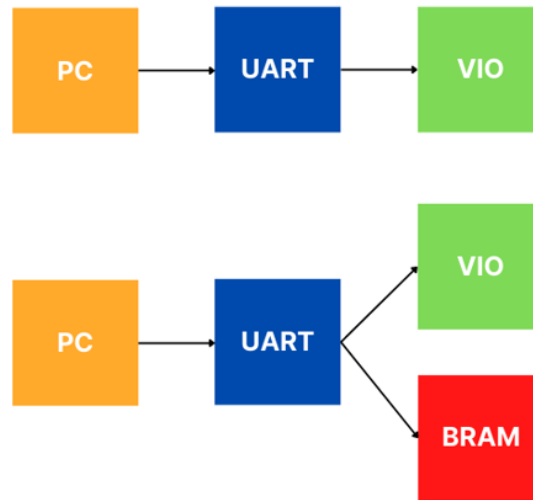


Abbildung 1.5: Blockdiagram of the two first testing stages

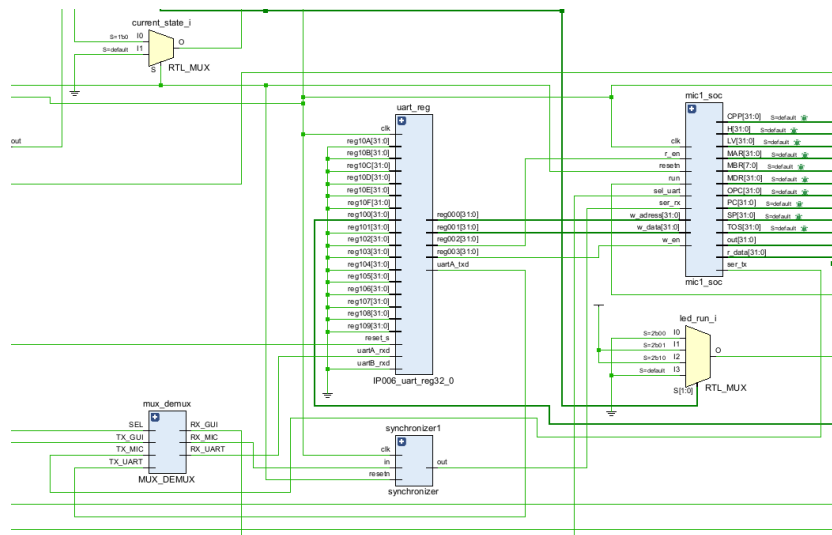


Abbildung 1.6: Schematic of the UART module including the MUX connected to the MIC SOC

All the tests were performed in the Block Design domain. The final implementation was done manually in VHDL.

1.2.3 Code

Listing 1.3: Source Code of the *mux_demux*.

```
1  'timescale 1ns / 1ps
2  module MUX_DEMUX(
3      input SEL,
4      input TX_GUI,
5      input TX_UART,
6      input TX_MIC,
7      output RX_UART,
8      output RX_MIC,
9      output RX_GUI
10 );
11 reg RX_UART, RX_MIC, RX_GUI;
12 always@*
13     begin
14         case (SEL)
15             0 : begin
16                 RX_UART = TX_GUI;
17                 RX_GUI = TX_UART;
18                 RX_MIC = 1;
19             end
20             1 : begin
21                 RX_MIC = TX_GUI;
22                 RX_GUI = TX_MIC;
23                 RX_UART = 1;
24             end
25         endcase
26     end
27 end
28 endmodule
```

1.3 Sending Data via UART (Data Syntax)

The UART module has four 32-bit output registers (reg000-003) which communicate with the BRAM. The first register contains the address of the memory cell, the second the data, reg002 enables writing while reg003 toggles the GLOBAL ENABLE. To address the UART output register the command to be sent looks like following:

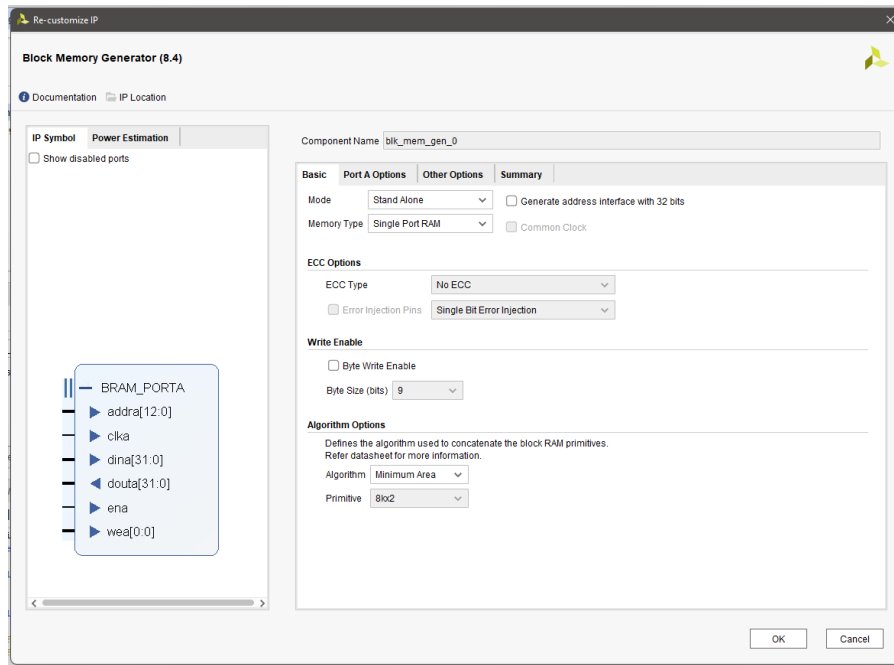


Abbildung 1.7: BRAM addressing



Abbildung 1.8: Syntax to send data to the UART output registers

with the first character reading (r) or writing (w) will be selected. The next 4 digit hex value addresses the output register, while the final 8 digits contain the actual data in hexadecimal.

and example to send 0xFFFF to address 0 of the BRAM:

```
w0000 0000 0000 -> select address 0
w0001 0000 FFFF -> write data 0xFFFF
w0002 0000 0001 -> enable the BRA
w0003 0000 0001 -> write enable
```







Name	Value	Activity	Direction	VIO
>  UART_GUI_i/blk_mem_gen_0_douta[31:0]	[H] 0000_FFFF		Input	hw_vio_1
>  UART_GUI_i/IP006_uart_reg32_0_reg000[31:0]	[H] 0000_0001		Input	hw_vio_1
>  UART_GUI_i/IP006_uart_reg32_0_reg001[31:0]	[H] 0000_FFFF		Input	hw_vio_1
>  UART_GUI_i/IP006_uart_reg32_0_reg002[31:0]	[H] 0000_0001		Input	hw_vio_1
>  UART_GUI_i/IP006_uart_reg32_0_reg003[31:0]	[H] 0000_0001		Input	hw_vio_1
>  UART_GUI_i/probe_in5_0_1[15:0]	[H] FFFF		Input	hw_vio_1

Abbildung 1.9: output at

2 VGA

2.1 Task

The purpose of the VGA controller will be to display the important registers of the MIC-1 microcontroller on the screen. These registers are the Stack Pointer, the Accumulator, the Program Counter and a few more. Therefore a VGA Controller has to be implemented in Vivado for the Basys3 FPGA board, which will be connected with the display over a VGA cable. The Controller will be implemented by using Verilog modules in Vivado and connecting them in a Block Design. In this implementation, a resolution of 640 pixels by 480 pixels is used for the screen, a refresh rate of 60Hz and 12 bits per pixel for the colors (red, green and blue). This part of the project will be an additional feature on an existing VGA Controller that one of the contributors of this project already worked on.

2.2 Original Design

In Figure 2.1 the original Block Design is shown. This design will be changed to fulfill the desired requirements, stated here above.

This implementation of the VGA Controller is a simple text editor. The inputs of the Block Design iIncr, iExtra, iStop are buttons and iSw0, iSw1 are sliders. Each slider could be on or off and for each position of the sliders the buttons have a different purpose. The implementation of the buttons could show a lower case character on the screen or make a space, enter, change the characters to upper case... The iRst is a button to reset the whole screen and the iClk is the input clock which is used for updating the screen. It is initialized at 100MHz but the Clocking Wizard module in Figure 2.1 decrements it to 25MHz. This clock signal ensures that we have a refresh rate of 60Hz.

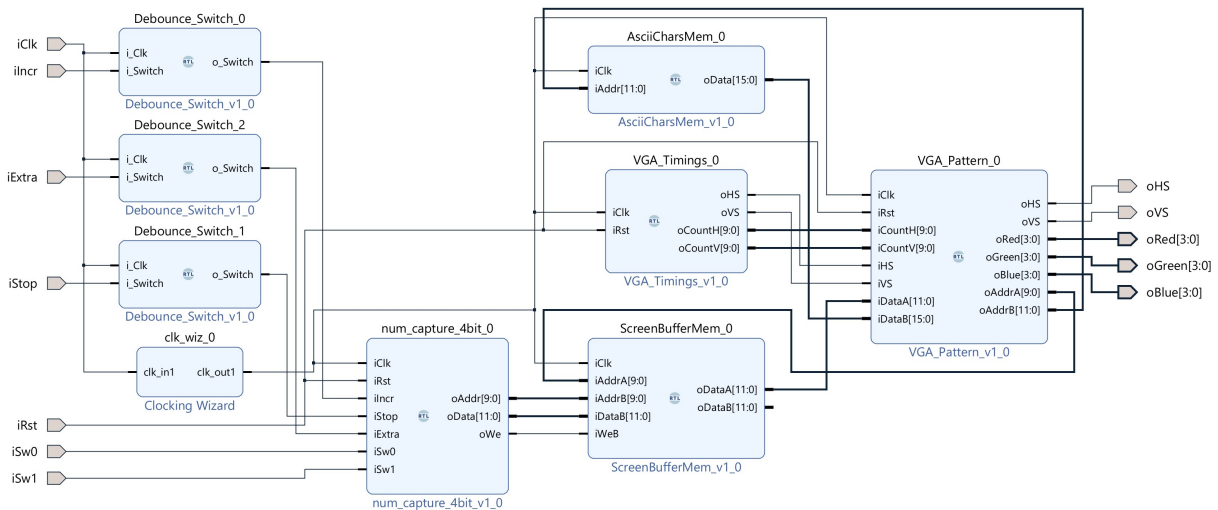


Abbildung 2.1: Original Block Design

The outputs of the block design are directly connected with the VGA module on the Basys3 board. The output ports oHS and oVS are respectively the horizontal sync and the vertical sync. This means that those represent the position which the VGA is updating the color of a pixel on the screen. The oRed, oGreen and oBlue are each 4 bit signal. Those are used to set the color of the pixel when the VGA is updating it.

The modules that are relevant for this project are the Debounce_Switch, the num_capture_4bit and the ScreenBufferMem. The num_capture_4bit is the module where the State Machine is defined and is discussed in

section 2.2.1. The ScreenBufferMem module is important to show the a Static part on the screen, this is also discussed further in the document in section 2.3.2. The Debounce_Switch module is used because when the switch is switched there is a small time when the slider is in an undefined state. This is unwanted, because if a button is pressed and the switch is undefined, there is no case that is valid, so it will give an error. In this case, this is because 0 and 1. It is shown in Figure 2.2. Here in the figure, 5 Volts corresponds that the Switch is ON, at 0 Volts the switch is OFF. For the buttons a Debounce module is not needed.

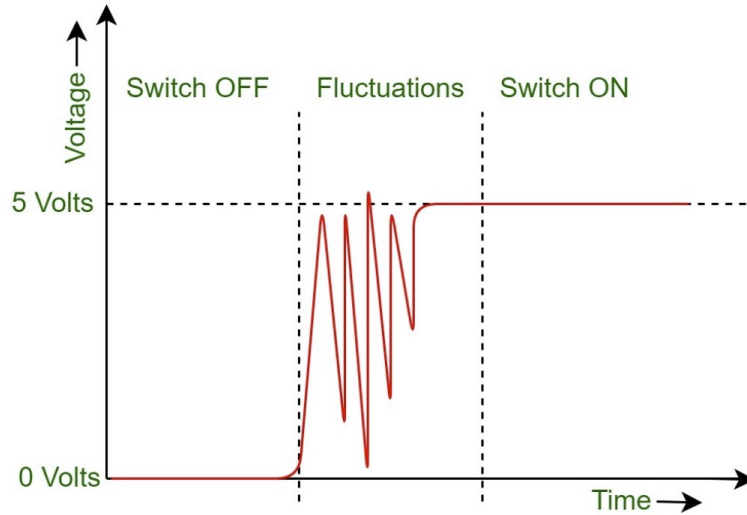


Abbildung 2.2: Debounce Switch

2.2.1 State Machine

This initial State Machine will start updating the screen in the top left pixel of the screen. Then it will go from left to right on the screen and when it reaches the right side, the row variable will be incremented so that the next pixel, is the pixel on the second row, first column. This will go on till it reaches the end of the screen and then the position will be reset to the initial position, the left top pixel.

The code of the State Machine is divided in four parts. The first one is the input, outputs and all internal variables and signals A.1. The next is the Next State Logic A.2. Then the Output Logic A.3 and the last one is the State Register A.4.

2.3 New Block Design

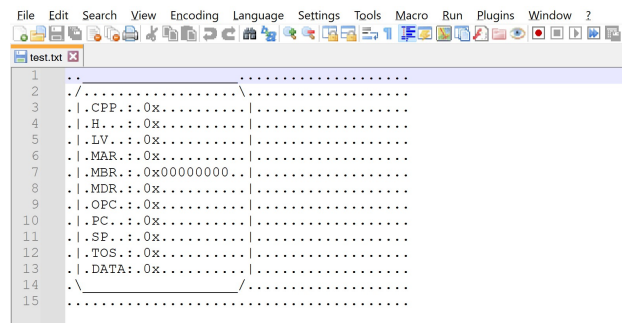
The new Block Design is shown in Figure 2.3. The difference from the original Block Design 2.1 is that there are a number of new inputs. Only the implementation of the reset button (iRst) and the clock (iClk) are kept the same. There is also an extra input switch (iSwUpdate), when the switch is ON the screen is updated otherwise the screen will keep on displaying the same values. Another additional output is added, this is the clock signal that is 25MHz (oClk25). This is added because this whole Block Design is added to the MIC1-project in Vivado and to have no complications it needs to work on the same clock signal. This means that the oClk25 is the input clock signal of the MIC1-project. Next the internal code of the num_capture_4bit module is changed, because there is the State Machine 2.3.1 defined.

2.3.2 Static Screen

The purpose of the VGA Module is that it can show the internal registers of the MIC1 microcontroller. Thus on the screen will be the names of the registers and those values. The values need to be updated all the time but the names are static. In the ScreenBufferMem module from Figure 2.3 a memory file can be uploaded that represents the static screen. Beware that the screen is divided in blocks to display the character. The screen has a size of 15x40 blocks.

If you want to change the static screen, follow these steps:

1. Make a .txt file with 15 rows and 40 columns that represent your new static screen, example 2.4. (Notepad++ recommended)
2. Save the .txt file on your computer.
3. Run this Python script A.9, it will open a file explorer where you need to select the .txt file.
4. The .mem file needed for the static screen gets saved in your Downloads folder.
5. Add the .mem file in the Vivado project and change the file in the ScreenBufferMem module to the new .mem file on line 21 in the code A.10.



```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
test.txt
1 .
2 .
3 . .CPP.:.0x.....\
4 . .H...:.0x.....|
5 . .LV...:.0x.....|
6 . .MAR...:.0x.....|
7 . .MBR...:.0x00000000..|
8 . .MDR...:.0x.....|
9 . .OPC...:.0x.....|
10 . .PC...:.0x.....|
11 . .SP...:.0x.....|
12 . .TOS...:.0x.....|
13 . .DATA:.0x.....|
14 . \
15 .
```

Abbildung 2.4: Example of a static screen in Notepad++

3 GUI

3.0.1 GUI

A Python file which includes a graphical user interface (GUI) was created to exchange the program on the MIC. With this GUI, it is easy and intuitive to interact with the MIC by connecting the Basys 3 board with a serial port. The TKinter framework was chosen as it is already included in the Python standard library and therefore no additional dependencies are required. TKinter is also very user- friendly.

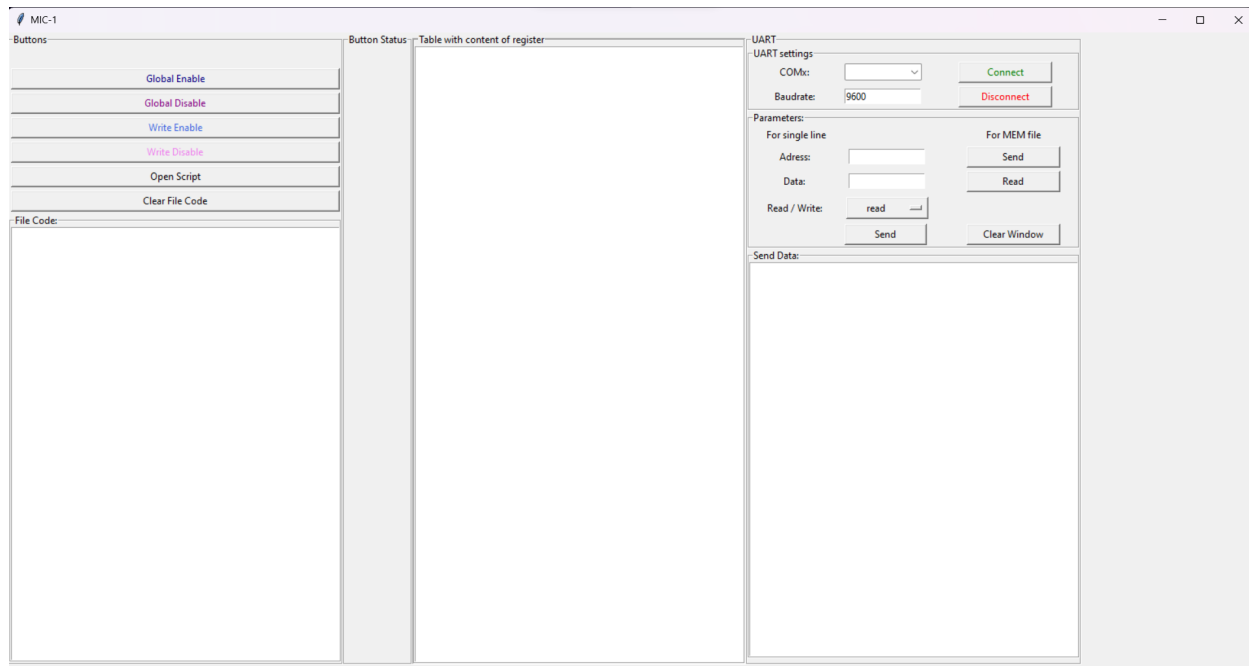


Abbildung 3.1: GUI

The GUI includes various Buttons. The upper four buttons are used when sending a single line to the MIC-1.

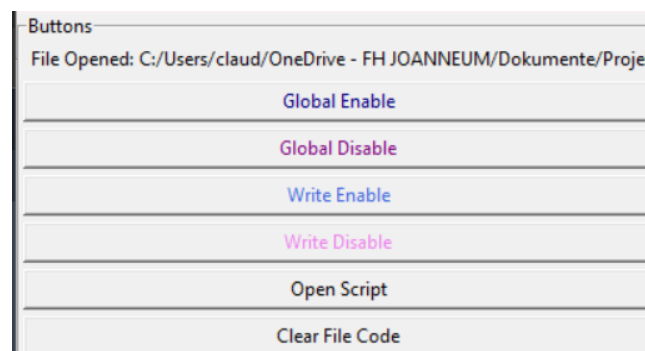


Abbildung 3.2: Buttons of the GUI

3.0.2 Protocol

Before sending a file, the MIC-1 should be connected via UART. To do that choose the COM-Port in the Window *COMx*:

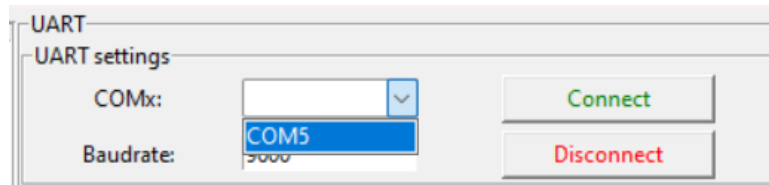


Abbildung 3.3: UART Connection

When the MIC-1 is successfully connected, the word Connect is displayed in the button table.

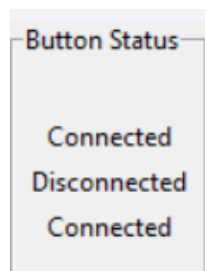


Abbildung 3.4: Completed UART Connection

The File path is displayed in the upper left corner of the GUI. The number of lines in the file is shown in the Button Status Table. In addition, the content of the file is displayed in the *File Code* Window.

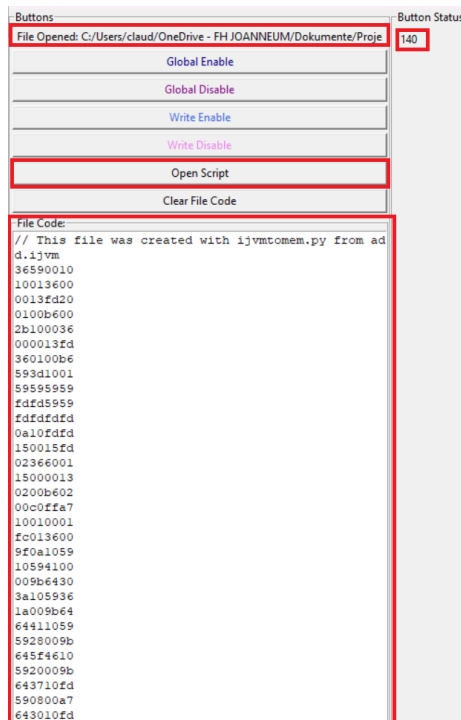


Abbildung 3.5: Open a mem file

The mem file can be selected after pressing the button *Open Script*.

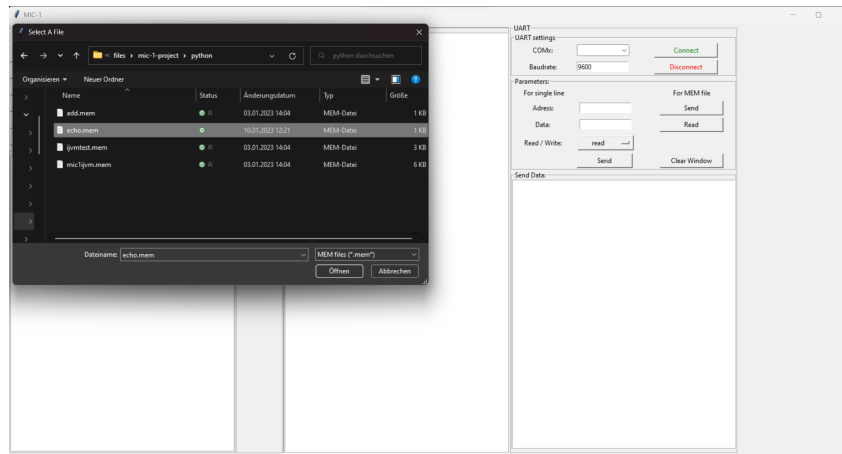


Abbildung 3.6: Search for Mem file in explorer

When the user presses the *Send* button for the MEM file the file data get send address by address (line by line) to the registers of the microcontroller.

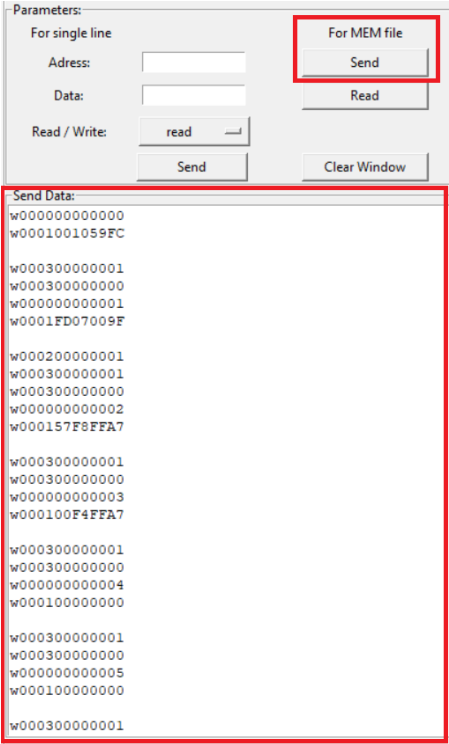


Abbildung 3.7: Send mem file to MIC-1

To read back the memory from the MIC-1 the user has to press the *Read* Button. The GUI reads back the data from every address and displays it in the *Table with content of register*.

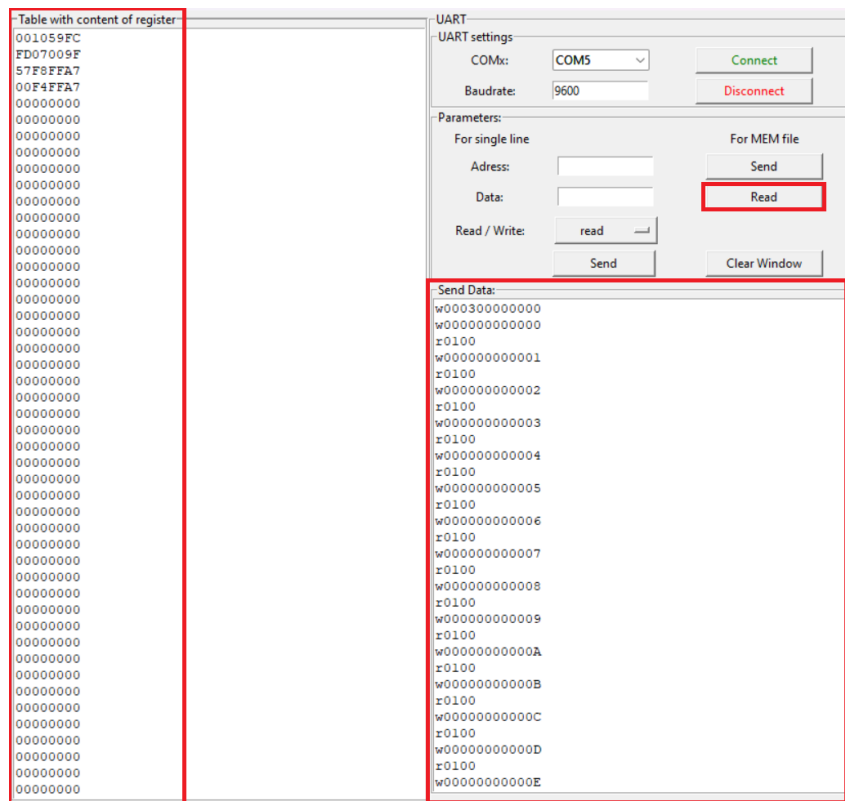


Abbildung 3.8: Read registers of MIC-1

3.0.3 Code

First the MIC1 class is defined with an `__init__` method. In the `__init__` method, several global variables (such as `serial_data`, `filter_data`, `serial_object`...) are defined. These variables are later used in different methods. Also a Tkinter window and its title is created in the `__init__`. The size of the window is set to fill the entire screen. In the last line the `create_widgets` method is called, which creates the buttons and fields that are displayed in the GUI. The `create_widgets` method is going to be described later.

The run method start the Tkinter main loop, which updates the GUI and waits for user input.

Listing 3.1: Initialisation and main loop

```
1
2 class MIC1:
3     def __init__(self):
4         #declare global variable
5         self.serial_data = ''
6         self.filter_data = ''
7         self.update_period = 5
8         self.serial_object = None
9         self.file = None
10        self.filename = None
11        self.BRAMenable = "w000200000001"
12        self.Wenable = "w000300000001"
13        self.BRAMdisable = "w000200000000"
14        self.Wdisable = "w000300000000"
15
16        #create a window
17        self.window = tk.Tk()
18        self.window.title('MIC-1')
19        w, h = self.window.winfo_screenwidth(), self.window.winfo_screenheight()
20        self.window.geometry("%dx%d+0+0" % (w, h))
21        self.uartState = False
22        self.create_widgets()
23
24        #mainloop
25        def run(self):
26            self.window.mainloop()
```

The connect method is used to establish a connection with the connected Baysy3 board over a serial port. It uses the Serial class from the 'serial' module to open a serial port specified by the user in the GUI and sets the baud rate. It also starts a new thread with the `get_data` method which runs in the background and continuously reads data from the device and updates the status window. After the device is connected, Connected is printed in the button status window, so the user knows the connection was established successfully.

To then close the serial port connection again a disconnect method is made. If the device is disconnected, "Disconnected" is printed in the button status window.

Listing 3.2: UART Connection

```
1
2 #connect uart
3 def connect(self):
4     global serial_object
5
6     port = self.entryCOM.get()
7     baud = self.entryBaudrate.get()
8     self.serial_object = serial.Serial('COM' + str(port))
9     self.serial_object.baudrate = baud
10
11     t1 = threading.Thread(target=self.get_data)
```

```

12         t1.daemon = True
13         t1.start()
14
15         connected = Label(self.buttontable, text="Connected")
16         connected.grid()
17
18         #disconnect uart
19         def disconnect(self):
20             global serial_object
21
22             self.serial_object.close()
23
24             disconnected = Label(self.buttontable, text="Disconnected")
25             disconnected.grid()

```

The `get_data` method reads data from the device using the 'serial_object' variable defined earlier and inserts the data it in the status window in the GUI.

Listing 3.3: `get_data` method

```

1
2         #get data from mic and display in status window
3         def get_data(self):
4             global serial_object
5             global filter_data
6
7             while (1):
8                 try:
9                     self.status.insert(END, self.serial_object.read(8))
10                    self.status.insert(END, "\n")
11
12
13                except TypeError:
14                    pass
15                self.window.update()
16                self.status.insert(END, "\n")
17                self.status.update()

```

The `processButtonSend` method is called when the user clicks the `SSend` button. Before pressing the `SSend` button the user also must select between `read`, `write` or `"1 line"`.

If the `read` option is selected, it sends the address and the read command to the device. The address is taken from the entry address widget and the read command is `r0100`. The commands are encoded as bytes and sent through the serial connection represented by the `serial_object` variable. The received data is logged in the status window. The sent commands are displayed in the text window.

If the `"write"` option is selected, it sends both the address and the data to the device. The address is taken from the entry address widget and the data is taken from the entry data widget. First the address is sent and then the data. The sent commands are displayed in the text window.

If the `"1 line"` option is selected, it sends the data that is typed in from the user as a single command to the device. The data is taken from the entry data widget. The sent command and any received data are logged in the text and status window.

In all three cases, the function uses the `write` method of the `serial_object` variable to send the commands and the `read` method to receive any data. The `time.sleep` function is used to wait for a short period of time to allow the device to process the command and respond.

Listing 3.4: `processButtonSend`

```

1  #read from mic or write to mic, data and address
2  def processButtonSend(self):
3      global serial_object
4      global text
5      selection = self.action.get()
6
7      #if read is selected, send address and read command
8      if selection == "read":
9          send_to_address = self.entryaddress.get()
10         send_address = send_to_address
11         self.serial_object.write(str.encode("w0000000000" + send_address))
12         self.text.insert(END, str.encode("w0000000000" + send_address))
13         self.text.insert(END, "\n")
14         self.serial_object.write(str.encode("r0100"))
15         self.text.insert(END, str.encode("r0100"))
16         time.sleep(0.1)
17         self.text.insert(END, "\n")
18         self.status.insert(END, "\n")
19         self.status.insert(END, self.serial_object.read())
20         self.status.insert(END, "\n")
21
22     #if write is selected, send address and data
23     if selection == "write":
24         send_data = self.entrydata.get()
25         send_to_address = self.entryaddress.get()
26         send_address = send_to_address
27         self.serial_object.write(str.encode("w0000000000" + send_to_address))
28         self.text.insert(END, str.encode("w0000000000" + send_address))
29         self.text.insert(END, "\n")
30         self.serial_object.write(str.encode("w0001" + send_data))
31         self.text.insert(END, str.encode("w0001" + send_data))
32         self.text.insert(END, "\n")
33
34     #if 1 line is selected, just sends the one line that was entered in the
35     #data entry box
36     if selection == "1 line":
37         send_data = self.entrydata.get()
38         self.serial_object.write(str.encode(send_data))
39         self.text.insert(END, str.encode(send_data))
40         self.text.insert(END, "\n")
41         self.status.insert(END, "\n")
42         self.status.insert(END, self.serial_object.read())
43         self.status.insert(END, "\n")

```

The method called `create_widgets` is used to create and layout various elements (widgets). Buttons, labels, text boxes, entries and lableframes were used. Elements are created by calling a specific class constructor, such as `tk.Label`, `tk.Entry`, or `tk.Button`, and passing the root window as the first argument. Various properties of the widget can be set, such as its size, position, background color, font, and event handlers, by passing additional arguments to the constructor or by calling specific methods on the widget object. The `grid` method is used to position the element within the window, using row and column coordinates.

Listing 3.5: `create_widgets`

```

1  #creates all widgets for the gui
2  def create_widgets(self):
3      #create frame for buttons
4      self.buttonframe = LabelFrame(self.window, labelanchor='nw', text="
5          Buttons")
6      self.buttonframe.grid(row=2, column=1, sticky=N + S)
7
8      #global/write enable buttons

```

```

8     self.globalenablebutton = Button(self.buttonframe, text='Global Enable',
9                                     fg='red', command=self.clickGlobalEnable)
10    self.globalenablebutton.grid(row=3, column=1, sticky=W+E, padx=2, pady=2)
11
12    self.globaldisablebutton = Button(self.buttonframe, text='Global Disable
13    ',
14                                     fg='blue', command=self.clickGlobalDisable)
15    self.globaldisablebutton.grid(row=4, column=1, sticky=W+E, padx=2, pady
16    =2)
17
18    self.writeenablebutton = Button(self.buttonframe, text='Write Enable',
19    fg='violet', command=self.clickWriteEnable)
20    self.writeenablebutton.grid(row=5, column=1, sticky=W+E, padx=2, pady=2)
21
22    self.writedisablebutton = Button(self.buttonframe, text='Write Disable',
23    fg='purple', command=self.clickWriteDisable)
24    self.writedisablebutton.grid(row=6, column=1, sticky=W+E, padx=2, pady=2)
25
26    #call a python script
27    self.callscript = Button(self.buttonframe, text="Open Script", command=
28    self.fileDialog)
29    self.callscript.grid(row=7, column=1, sticky=W+E, padx=2, pady=2)
30
31    #Frame to display file code
32    self.textframe = LabelFrame(self.buttonframe, labelanchor='nw', text='
33    File Code:')
34    self.textframe.grid(row=8, column=1, sticky=S + SW)
35    self.frameText = Text(self.textframe, width=50, height=30)
36    self.frameText.grid(row=1, column=1, sticky=S + W)
37
38    #lable to display path of the file that is open
39    self.label = ttk.Label(self.buttonframe, text="")
40    self.label.grid(row=0, column=1, sticky="ew", padx=5, pady=3)
41
42    #create frame for the status of the button and to display number of
43    lines from the imported file
44    self.buttonstatusframe = LabelFrame(
45    self.window, labelanchor='nw', text="Button Status", width=25)
46    self.buttonstatusframe.grid(row=2, column=2, sticky=N + S + W + E)
47
48    #canvas to write the status (disconnected/connected uart)
49    self.buttontable = tk.Canvas(self.buttonstatusframe,
50    width=100, height=400)
51    self.buttontable.grid(column=2, row=1, padx=5, pady=5)
52
53    #space to display number of lines from the imported file
54    self.numberlines = Label(self.buttontable, text='')
55    self.numberlines.grid()
56
57    #create frame to display the content of the register
58    self.statusframe = LabelFrame(self.window, labelanchor='nw',
59    text="Table with content of register", width=50,
60    bd=3)
61    self.statusframe.grid(row=2, column=3, sticky=N + S)
62
63    #textbox to display status of register
64    self.status = Text(self.statusframe, width=50, height=40)
65    self.status.grid(row=1, column=1, sticky=N + S)
66
67    # create frame for uart
68    self.dataframe = LabelFrame(self.window, text='UART', width=50)
69    self.dataframe.grid(column=5, row=2, sticky=N+W+S)
70
71    #frame for uart settings (com/baudrate)
72    self.uartframe = LabelFrame(self.dataframe, text='UART settings', width
73    =50)

```

```

68     self._uartframe.grid(column=1, row=1, sticky=N+W+S+E)
69
70     #lable and entry for the COM
71     self.labelCOM = Label(self._uartframe, text="COMx: ", width=15)
72     COM = StringVar()
73     self.entryCOM = Entry(self._uartframe, textvariable=COM, width=15)
74     self.labelCOM.grid(row=1, column=1, padx=2, pady=2)
75     self.entryCOM.grid(row=1, column=2, padx=2, pady=2)
76
77     #lable and entry for the baudrate
78     self.labelBaudrate = Label(self._uartframe, text="Baudrate: ", width=15)
79     self.Baudrate = StringVar(value="9600")
80     self.entryBaudrate = Entry(self._uartframe, textvariable=self.Baudrate,
81                               width=15)
82     self.labelBaudrate.grid(row=2, column=1, padx=2, pady=2)
83     self.entryBaudrate.grid(row=2, column=2, padx=2, pady=2)
84
85     #blank lable between buttons and entry boxes
86     self.blank_label = Label(self._uartframe, text="", width=5)
87     self.blank_label.grid(row=1, column=3, columnspan=1)
88
89     #connect button for uart
90     self.buttonC = tk.Button(self._uartframe, text="Connect", command=self.
91                              connect, width=15)
92     self.buttonC.grid(row=1, column=4, padx=2, pady=2, sticky=E)
93
94     #connect button for uart
95     self.buttonD = tk.Button(self._uartframe, text="Disconnect", command=self
96                              .disconnect, width=15)
97     self.buttonD.grid(row=2, column=4, padx=2, pady=2, sticky=E)
98
99     #frame to communicate (write/read) with the MIC
100    self.adressframe = LabelFrame(self.dataframe, text='Parameters:', width
101                                  =50)
102    self.adressframe.grid(column=1, row=3, sticky=N+W+S+E)
103
104    #lable read and write
105    self.labelaction = Label(self.adressframe, text="Read / Write: ", width
106                              =15)
107    self.labelaction.grid(row=4, column=1, padx=2, pady=2)
108
109    #to choose if read from or write to the MIC or jus send one line
110    self.choices = ['read', 'write', '1 line']
111    self.action = StringVar()
112    self.action.set('read')
113    self.actiondata = OptionMenu(self.adressframe, self.action, *self.
114                                 choices)
115    self.actiondata.grid(row=4, column=2, padx=2, pady=2)
116
117    #button to send file
118    self.buttonSendmem = Button(self.adressframe, text="Send mem", command=
119                                self.processButtonSendmem, width=15)
120    self.buttonSendmem.grid(row=5, column=4, padx=2, pady=2, sticky=tk.W)
121
122    #read button
123    self.buttonRead = Button(self.adressframe, text="Read", command=self.
124                              processButtonRead, width=15)
125    self.buttonRead.grid(row=6, column=4, padx=2, pady=2, sticky=tk.W)
126
127    #button to send adress/data
128    self.buttonSend2 = Button(self.adressframe, text="Send", command=self.
129                               processButtonSend, width=15)
130    self.buttonSend2.grid(row=4, column=4, padx=2, pady=2, sticky=tk.W)
131
132    #blank lable to keeo the gui clean
133    self.blank_label2 = Label(self.adressframe, text="", width=5)
134    self.blank_label2.grid(row=1, column=3, columnspan=1)

```



```

126
127     #lable and entry box for adress
128     self.labeladress = Label(self.adressframe, text="Adress: ", width=15)
129     self.adress = StringVar()
130     self.entryadress = Entry(self.adressframe, textvariable=self.adress,
131                             width=15)
132     self.labeladress.grid(row=5, column=1, padx=2, pady=2)
133     self.entryadress.grid(row=5, column=2, padx=2, pady=2)
134
135     #lable and entry box for data
136     self.labelData = Label(self.adressframe, text="Data: ", width=15)
137     self.Data = StringVar()
138     self.entrydata = Entry(self.adressframe, textvariable=self.Data, width
139                           =15)
140     self.labelData.grid(row=6, column=1, padx=2, pady=2)
141     self.entrydata.grid(row=6, column=2, padx=2, pady=2)
142
143     #button to clear the status and the send data window
144     self.buttonRead = Button(self.adressframe, text="Clear Window", command=
145                             self.processButtonDelete, width=15)
146     self.buttonRead.grid(row=7, column=4, padx=2, pady=2, sticky=tk.W)
147
148     #frame to display send data
149     self.labelOutText = LabelFrame(self.dataframe, text="Send Data:", width
150                                  =50)
151     self.labelOutText.grid(row=6, column=1, sticky=S+W)
152     self.text = Text(self.labelOutText, width=50, height=26)
153     self.text.grid(row=7, column=1, sticky=S+W+E)

```

3.0.4 Send data

The function called processButtonSendmem sends the contents of a file to the serial object.

The function operates as a loop that runs num_lines times. num_lines Is the variable that holds the number of lines that the file has that is going to be send. For each iteration, the following steps are performed:

The address of the memory location to be written to is set by concatenating "0"with the current iteration number, padded with zeros to a total of 2 characters. First the write command (the string "w0000000000followed by the current address) is sent. The file is opened and the line of the file corresponding to the current iteration is read and stored in the memdata variable. The contents of memdata are converted to uppercase and sent to the serial object as a write command (the string "w0001followed by the uppercase contents of memdata). Because the MIC needs uppercase letters. BRAMenable is send to enable the BRAM. Then the Wenable and Wdisable commands are sent. The address is incremented and converted to uppercase hexadecimal format. The resulting string is padded with zeros to a total of 2 characters. Then the file is closed. After the loop has completed, the address is set to 0 and the file is closed.

Listing 3.6: processButtonSendmem

```

1
2     #send the whole (mem) file
3     def processButtonSendmem(self):
4         global file
5         global filename
6         global serial_object
7         global BRAMenable
8         global BRAMdisable
9         global Wenable
10        global Wdisable
11        global num_lines
12

```

```

13     address = "0".zfill(2)
14     for intaddress in range(1,num_lines):
15         self.serial_object.write(str.encode("w0000000000" + address))
16         self.text.insert(END, str.encode("w0000000000" + address))
17         self.text.insert(END, "\n")
18         file = open(filename, 'r')
19         memdata = file.readlines()[intaddress]
20         memdataupper = str.upper(memdata)
21         self.serial_object.write(str.encode("w0001" + (memdataupper)))
22         self.text.insert(END, str.encode("w0001" + (memdataupper)))
23         self.text.insert(END, "\n")
24         if address == "01":
25             self.serial_object.write(str.encode(self.BRAMenable))
26             self.text.insert(END, str.encode(self.BRAMenable))
27             self.text.insert(END, "\n")
28             self.serial_object.write(str.encode(self.Wenable))
29             self.text.insert(END, str.encode(self.Wenable))
30             self.text.insert(END, "\n")
31             self.serial_object.write(str.encode(self.Wdisable))
32             self.text.insert(END, str.encode(self.Wdisable))
33             self.text.insert(END, "\n")
34             intconaddress = int(address, base=16)
35             intconaddress +=1
36             if intconaddress % 16 == 0:
37                 hexa = hex(intconaddress)
38                 hexaddress = hexa.strip('0x')
39                 hadress = str(hexaddress + "0")
40             else:
41                 hexa = hex(intconaddress)
42                 hexaddress = hexa.strip('0x')
43                 hadress = str(hexaddress)
44             uppercaseaddress = str.upper(hadress)
45             address = uppercaseaddress.zfill(2)
46         file.close()
47     address = "0"
48     file.close()

```

3.0.5 Read data

The function `processButtonRead` reads the registers of the MIC-1. For that the function sends the `Wdisable` command in the beginning.

The function operates as a loop that runs `num_lines` times. `num_lines` is the variable that holds the number of registers the function has to read out. It is given by the function `processButtonSendmem`. For each iteration, the following steps are performed:

The address of the memory location to be written to is set by concatenating "0" with the current iteration number, padded with zeros to a total of 2 characters. First the write command (the string "w0000000000" followed by the current address) is sent. To read out the registers the read command (the string "r0100") is sent. The address is incremented and converted to uppercase hexadecimal format. The resulting string is padded with zeros to a total of 2 characters. After the loop has completed, the address is set to 0.

Listing 3.7: `processButtonRead`

```

1
2     global BRAMenable
3     global Wenable
4     global serial_object
5     global num_lines
6
7     self.serial_object.write(str.encode(self.Wdisable))

```

```

8         self.text.insert(END, str.encode((self.Wdisable)))
9         self.text.insert(END, "\n")
10
11         address = "0".zfill(2)
12         for i in range(1,num_lines):
13             self.serial_object.write(str.encode("w0000000000" + address))
14             self.text.insert(END, str.encode("w0000000000" + address))
15             self.text.insert(END, "\n")
16             self.serial_object.write(str.encode("r0100"))
17             self.text.insert(END, str.encode("r0100"))
18             self.text.insert(END, "\n")
19             #self.status.insert(END, "\n")
20             #self.status.insert(END, self.serial_object.readline())
21             time.sleep(0.1)
22
23             intconaddress = int(address, base=16)
24             intconaddress +=1
25             if intconaddress % 16 == 0:
26                 hexa = hex(intconaddress)
27                 hexaddress = hexa.strip('0x')
28                 hadress = str(hexaddress + "0")
29             else:
30                 hexa = hex(intconaddress)
31                 hexaddress = hexa.strip('0x')
32                 hadress = str(hexaddress)
33             uppercaseaddress = str.upper(hadress)
34             address = uppercaseaddress.zfill(2)
35         address = "0"

```

A VGA code

A.1 Original VGA Controller code

Listing A.1: Input, outputs and all internal variables and signals

```

1 module num_capture_4bit
2     //Input, Output definition and the internal signals
3     (
4         input wire iClk, iRst, iInc, iStop, iExtra, iSw0, iSw1,
5
6         output wire [9:0] oAddr,
7         output wire [11:0] oData,
8         output wire oWe);
9
10    reg [11:0] rNum_Curr, rNum_Next;
11    reg rWeN, rWe, rToggle_Curr, rToggle_Next;
12    reg [9:0] rAddr_Next, rAddr_Curr;
13    reg [4:0] rFSM_Curr, wFSM_Next;
14
15    //State Definition
16    localparam sInit = 4'b0000;
17    localparam sIdle = 4'b0001;
18    localparam sPush = 4'b0010;
19    localparam sIncA = 4'b0011;
20    localparam sIncB = 4'b0100;
21    localparam sStop = 4'b0101;
22    localparam sStopInc = 4'b0110;
23    localparam sTog = 4'b0111;
24    localparam sTogInc = 4'b1000;
25    localparam sRst = 4'b1001;
26    localparam sSpace = 4'b1010;
27    localparam sSpaceInc = 4'b1011;

```

```

28     localparam sBreak    = 4'b1100;
29     localparam sBreakInc = 4'b1101;
30     localparam sDelete   = 4'b1110;
31     localparam sDelInc   = 4'b1111;

```

Listing A.2: Next State Logic

```

1     //State Register without reset
2     always @(posedge iClk)
3     begin
4         rFSM_Curr <= wFSM_Next;
5     end
6
7     //Next State Logic
8     always @(posedge iClk)
9     begin
10        case (rFSM_Curr)
11
12            sInit   :   if (rAddr_Curr > 0)
13                        wFSM_Next <= sInit;
14                        else
15                            wFSM_Next <= sIdle;
16
17            sIdle   :   if (iStop == 0 && iIncr == 0 && iRst == 0 && iExtra ==
18                        0)
19                        wFSM_Next <= sIdle;
20                        else if (iRst == 1)
21                            wFSM_Next <= sRst;
22                        else if (iIncr == 1)
23                            wFSM_Next <= sPush;
24                        else if (iStop == 1)
25                            wFSM_Next <= sStop;
26                        else if (iExtra == 1 && iSw0 == 0 && iSw1 == 0)
27                            wFSM_Next <= sTog;
28                        else if (iExtra == 1 && iSw0 == 1 && iSw1 == 1)
29                            wFSM_Next <= sSpace;
30                        else if (iExtra == 1 && iSw0 == 0 && iSw1 == 1)
31                            wFSM_Next <= sBreak;
32                        else if (iExtra == 1 && iSw0 == 1 && iSw1 == 0)
33                            wFSM_Next <= sDelete;
34
35            sPush    :   if (iIncr == 1)
36                        wFSM_Next <= sPush;
37                        else if (iIncr == 0)
38                            if (rToggle_Curr == 0)
39                                wFSM_Next <= sIncA;
40                            else
41                                wFSM_Next <= sIncB;
42
43            sIncA     :   wFSM_Next <= sIdle;
44
45            sIncB     :   wFSM_Next <= sIdle;
46
47            sStop     :   if (iStop == 1)
48                        wFSM_Next <= sStop;
49                        else if (iStop == 0)
50                            wFSM_Next <= sStopInc;
51
52            sStopInc  :   wFSM_Next <= sIdle;
53
54            sTog      :   if (iExtra == 1 && iSw0 == 0 && iSw1 == 0)
55                        wFSM_Next <= sTog;
56                        else if (iExtra == 0 && iSw0 == 0 && iSw1 == 0)
57                            wFSM_Next <= sTogInc;

```

```

58         sToglnc : wFSM_Next <= sIdle;
59
60         sRst    : wFSM_Next <= sInit;
61
62         sSpace  : if ( iExtra == 1 && iSw0 == 1 && iSw1 == 1)
63                   wFSM_Next <= sSpace;
64                   else
65                     wFSM_Next <= sSpaceInc;
66
67         sSpaceInc : wFSM_Next <= sIdle;
68
69         sBreak   : if ( iExtra == 1 && iSw0 == 0 && iSw1 == 1)
70                   wFSM_Next <= sBreak;
71                   else
72                     wFSM_Next <= sBreakInc;
73
74         sBreakInc : wFSM_Next <= sIdle;
75
76         sDelete  : if ( iExtra == 1 && iSw0 == 1 && iSw1 == 0)
77                   wFSM_Next <= sDelete;
78                   else
79                     wFSM_Next <= sDelInc;
80
81         sDelInc  : wFSM_Next <= sIdle;
82
83         default  : wFSM_Next <= sIdle;
84
85     endcase
86 end

```

Listing A.3: Output Logic

```

1  //Output Logic
2  //Only in function of states
3  always @(posedge iClk)
4  begin
5      case (rFSM_Curr)
6          sIncA    : rAddr_Next <= rAddr_Curr;
7
8          sIncB    : rAddr_Next <= rAddr_Curr;
9
10         sInit    : if ( rAddr_Curr > 0 )
11                   rAddr_Next <= rAddr_Curr - 1;
12                   else
13                     rAddr_Next <= 0;
14
15         sIdle     : rAddr_Next <= rAddr_Curr;
16
17         sPush     : rAddr_Next <= rAddr_Curr;
18
19         sStop     : rAddr_Next <= rAddr_Curr;
20
21         sStopInc  : if ( rAddr_Curr < 599)
22                   rAddr_Next <= rAddr_Curr + 1;
23                   else
24                     rAddr_Next <= 0;
25
26         sRst      : rAddr_Next <= 600;
27
28         sSpace    : rAddr_Next <= rAddr_Curr;
29
30         sSpaceInc : if ( rAddr_Curr < 598)
31                   rAddr_Next <= rAddr_Curr + 2;
32                   else
33                     rAddr_Next <= 0;

```

```

34
35     sBreak   :   rAddr_Next <= rAddr_Curr;
36
37     sBreakInc :   if (rAddr_Curr >= 560)
38                   rAddr_Next <= 0;
39                   else
40                       rAddr_Next <= rAddr_Curr + 40 - (rAddr_Curr % 40);
41
42     sDelete  :   rAddr_Next <= rAddr_Curr;
43
44     sDelInc  :   if (rAddr_Curr == 0)
45                   rAddr_Next <= rAddr_Curr;
46                   else
47                       rAddr_Next <= rAddr_Curr - 1;
48
49     default  :   rAddr_Next <= rAddr_Curr;
50 endcase
51 end

```

Listing A.4: State Register

```

1  always @(posedge iClk)
2      begin
3          case (rFSM_Curr)
4              sIncA   :   if ( rNum_Curr == 0 )
5                           rNum_Next <= 512;
6                           else if ( rNum_Curr >= 512 && rNum_Curr <= 768 )
7                               rNum_Next <= rNum_Curr + 32;
8                           else if ( rNum_Curr == 800 )
9                               rNum_Next <= 1056;
10                          else if ( rNum_Curr < 1216 )
11                              rNum_Next <= rNum_Curr + 32;
12
13                          // in case of switch while a lowercase letter is on screen
14                          else if ( rNum_Curr >= 2080 && rNum_Curr < 2240 )
15                              rNum_Next <= rNum_Curr - 2080 + 1056;
16                          else
17                              rNum_Next <= 512;
18
19                          // Alternate way of displaying numbers > 9, enabled by iSwitchA)
20                          sIncB   :   if ( rNum_Curr == 0 )
21                                         rNum_Next <= 512;
22                                         else if ( rNum_Curr >= 512 && rNum_Curr <= 768 )
23                                             rNum_Next <= rNum_Curr + 32;
24                                         else if ( rNum_Curr == 800 )
25                                             rNum_Next <= 2080;
26
27                          // in case of switch while a UPPERCASE letter is on screen
28                          else if ( rNum_Curr >= 1056 && rNum_Curr < 1216 )
29                              rNum_Next <= rNum_Curr + 2080 - 1056;
30                          else if ( rNum_Curr <= 2208 )
31                              rNum_Next <= rNum_Curr + 32;
32                          else
33                              rNum_Next <= 512;
34
35              sInit   :   rNum_Next <= 0;
36
37              sIdle   :   rNum_Next <= rNum_Curr;
38
39              sPush    :   rNum_Next <= rNum_Curr;
40
41              sStop    :   rNum_Next <= rNum_Curr;
42
43              sStopInc :   rNum_Next <= 512;
44

```

```

45     sRst      :   rNum_Next <= 0;
46
47     sSpace    :   rNum_Next <= rNum_Curr;
48
49     sSpaceInc :   rNum_Next <= 0;
50
51     sBreak    :   rNum_Next <= rNum_Curr;
52
53     sBreakInc :   rNum_Next <= 0;
54
55     sDelete   :   rNum_Next <= 0;
56
57     sDelInc   :   rNum_Next <= 0;
58
59     default   :   rNum_Next <= rNum_Curr;
60
61 endcase
62 end
63
64 always @(posedge iClk)
65 begin
66     case(rFSM_Curr)
67
68         sToggleInc :   if (rToggle_Curr == 0)
69                         rToggle_Next <= 1;
70                         else
71                         rToggle_Next <= 0;
72
73         default    :   rToggle_Next <= rToggle_Next;
74
75     endcase
76 end
77
78 always @(posedge iClk)
79 begin
80     rAddr_Curr    <= rAddr_Next;
81     rNum_Curr     <= rNum_Next;
82     rToggle_Curr  <= rToggle_Next;
83 end
84
85 assign oAddr = rAddr_Curr;
86 assign oWe   = (rFSM_Curr == sIdle || rFSM_Curr == sInit || rFSM_Curr ==
87               sDelInc) ? 1 : 0;
88 assign oData = rNum_Curr;
89 endmodule

```

A.2 New VGA Controller code

Listing A.5: New input, outputs and all internal variables and signals

```

1  module num_capture_4bit
2
3      (
4          input wire iClk, iRst, iSwUpdate, //change to switch
5          input wire[47:0] iCPP, iH,
6                      iLV, iMAR,
7                      iMDR, iOPC,
8                      iPC, iSP,
9                      iTOS, iData,
10         input wire[11:0] iMBR,
11
12         output wire[9:0] oAddr,

```

```

13     output wire [11:0] oData ,
14     output wire oWe
15 );
16 reg [11:0] rNum_Curr, rNum_Next;
17 reg       rWeN, rWe;
18 reg [9:0] rAddr_Next, rAddr_Curr;
19 reg [4:0] rFSM_Curr, wFSM_Next;
20
21 //State Definition
22 localparam sInit      = 4'b0000;
23 localparam sIdle      = 4'b0001;
24 localparam sUpdate0   = 4'b0010;
25 localparam sUpdate1   = 4'b0011;
26 localparam sUpdate2   = 4'b0100;
27 localparam sUpdate3   = 4'b0101;
28 localparam sUpdate4   = 4'b0110;
29 localparam sUpdate5   = 4'b0111;
30 localparam sUpdate6   = 4'b1000;
31 localparam sUpdate7   = 4'b1001;
32 localparam sUpdate8   = 4'b1010;
33 localparam sUpdate9   = 4'b1011;
34 localparam sUpdate10  = 4'b1100;
35 localparam sRst       = 4'b1101;

```

Listing A.6: New Next State Logic

```

1  //State Register without reset
2  always @(posedge iClk)
3  begin
4      rFSM_Curr <= wFSM_Next;
5  end
6
7  //Next State Logic
8  always @(posedge iClk)
9  begin
10     case (rFSM_Curr)
11
12         sRst      : wFSM_Next <= sInit;
13
14         sInit     : if (rAddr_Curr < 497)
15                     wFSM_Next <= sInit;
16                     else
17                     wFSM_Next <= sIdle;
18
19         sIdle     : if (iSwUpdate == 1)
20                     wFSM_Next <= sUpdate0;
21                     else if (iRst == 1)
22                     wFSM_Next <= sRst;
23                     else
24                     wFSM_Next <= sIdle;
25
26         sUpdate0  : if (rAddr_Curr == 97) //End of HEX line
27                     wFSM_Next <= sUpdate1;
28                     else
29                     wFSM_Next <= sUpdate0;
30
31         sUpdate1  : if (rAddr_Curr == 137) //End of HEX line
32                     wFSM_Next <= sUpdate2;
33                     else
34                     wFSM_Next <= sUpdate1;
35
36         sUpdate2  : if (rAddr_Curr == 177) //End of HEX line
37                     wFSM_Next <= sUpdate3;
38                     else
39                     wFSM_Next <= sUpdate2;

```



```

40
41     sUpdate3 : if (rAddr_Curr == 217) //End of HEX line
42                 wFSM_Next <= sUpdate4;
43     else
44                 wFSM_Next <= sUpdate3;
45
46     sUpdate4 : if (rAddr_Curr == 257) //End of HEX line
47                 wFSM_Next <= sUpdate5;
48     else
49                 wFSM_Next <= sUpdate4;
50
51     sUpdate5 : if (rAddr_Curr == 297) //End of HEX line
52                 wFSM_Next <= sUpdate6;
53     else
54                 wFSM_Next <= sUpdate5;
55
56     sUpdate6 : if (rAddr_Curr == 337) //End of HEX line
57                 wFSM_Next <= sUpdate7;
58     else
59                 wFSM_Next <= sUpdate6;
60
61     sUpdate7 : if (rAddr_Curr == 377) //End of HEX line
62                 wFSM_Next <= sUpdate8;
63     else
64                 wFSM_Next <= sUpdate7;
65
66     sUpdate8 : if (rAddr_Curr == 417) //End of HEX line
67                 wFSM_Next <= sUpdate9;
68     else
69                 wFSM_Next <= sUpdate8;
70
71     sUpdate9 : if (rAddr_Curr == 457) //End of HEX line
72                 wFSM_Next <= sUpdate10;
73     else
74                 wFSM_Next <= sUpdate9;
75
76     sUpdate10 : if (rAddr_Curr == 497) //End of HEX line
77                  wFSM_Next <= sIdle;
78     else
79                  wFSM_Next <= sUpdate10;
80
81     default : wFSM_Next <= sIdle;
82
83     endcase
84 end

```

Listing A.7: New Output Logic

```

1  //Output Logic
2  //only in function of states
3  always @(posedge iClk)
4  begin
5      case (rFSM_Curr)
6
7          sRst      : rAddr_Next <= 0;
8
9          //Only delete the data not the static values
10         sInit     : if (rAddr_Curr < 91)
11                       rAddr_Next <= 91;
12                     else if (rAddr_Curr == 98)
13                       rAddr_Next <= 131;
14                     else if (rAddr_Curr == 138)
15                       rAddr_Next <= 171;
16                     else if (rAddr_Curr == 178)
17                       rAddr_Next <= 211;

```

```

18         else if (rAddr_Curr == 218)
19             rAddr_Next <= 251;
20         else if (rAddr_Curr == 258)
21             rAddr_Next <= 291;
22         else if (rAddr_Curr == 298)
23             rAddr_Next <= 331;
24         else if (rAddr_Curr == 338)
25             rAddr_Next <= 371;
26         else if (rAddr_Curr == 378)
27             rAddr_Next <= 411;
28         else if (rAddr_Curr == 418)
29             rAddr_Next <= 451;
30         else if (rAddr_Curr == 458)
31             rAddr_Next <= 491;
32         else
33             rAddr_Next = rAddr_Curr + 1;
34
35     sIdle      : rAddr_Next <= 0;
36
37     sUpdate0   : if (rAddr_Curr < 91)
38                   rAddr_Next <= 91;
39                   else
40                       rAddr_Next <= rAddr_Curr + 1;
41
42     sUpdate1   : if (rAddr_Curr < 131)
43                   rAddr_Next <= 131;
44                   else
45                       rAddr_Next <= rAddr_Curr + 1;
46
47     sUpdate2   : if (rAddr_Curr < 171)
48                   rAddr_Next <= 171;
49                   else
50                       rAddr_Next <= rAddr_Curr + 1;
51
52     sUpdate3   : if (rAddr_Curr < 211)
53                   rAddr_Next <= 211;
54                   else
55                       rAddr_Next <= rAddr_Curr + 1;
56
57     sUpdate4   : if (rAddr_Curr < 251)
58                   rAddr_Next <= 251;
59                   else
60                       rAddr_Next <= rAddr_Curr + 1;
61
62     sUpdate5   : if (rAddr_Curr < 291)
63                   rAddr_Next <= 291;
64                   else
65                       rAddr_Next <= rAddr_Curr + 1;
66
67     sUpdate6   : if (rAddr_Curr < 331)
68                   rAddr_Next <= 331;
69                   else
70                       rAddr_Next <= rAddr_Curr + 1;
71
72     sUpdate7   : if (rAddr_Curr < 371)
73                   rAddr_Next <= 371;
74                   else
75                       rAddr_Next <= rAddr_Curr + 1;
76
77     sUpdate8   : if (rAddr_Curr < 411)
78                   rAddr_Next <= 411;
79                   else
80                       rAddr_Next <= rAddr_Curr + 1;
81
82     sUpdate9   : if (rAddr_Curr < 451)
83                   rAddr_Next <= 451;
84                   else

```

```

85         rAddr_Next <= rAddr_Curr + 1;
86
87     sUpdate10 :    if (rAddr_Curr < 491)
88                   rAddr_Next <= 491;
89     else
90         rAddr_Next <= rAddr_Curr + 1;
91
92     default    :    rAddr_Next <= rAddr_Curr;
93
94 endcase
95 end

```

Listing A.8: New State Register

```

1  always @(posedge iClk)
2      begin
3          case (rFSM_Curr)
4
5              sRst      :    rNum_Next <= 448;
6
7              sInit     :    rNum_Next <= 448;
8
9              sIdle     :    rNum_Next <= 0;
10
11             sUpdate0 :    if (rAddr_Curr < 91)
12                           rNum_Next <= {1'b0, iCPP[47:42], 5'b000000};
13             else if (rAddr_Curr == 91)
14                 rNum_Next <= {1'b0, iCPP[41:36], 5'b000000};
15             else if (rAddr_Curr == 92)
16                 rNum_Next <= {1'b0, iCPP[35:30], 5'b000000};
17             else if (rAddr_Curr == 93)
18                 rNum_Next <= {1'b0, iCPP[29:24], 5'b000000};
19             else if (rAddr_Curr == 94)
20                 rNum_Next <= {1'b0, iCPP[23:18], 5'b000000};
21             else if (rAddr_Curr == 95)
22                 rNum_Next <= {1'b0, iCPP[17:12], 5'b000000};
23             else if (rAddr_Curr == 96)
24                 rNum_Next <= {1'b0, iCPP[11:6], 5'b000000};
25             else if (rAddr_Curr == 97)
26                 rNum_Next <= {1'b0, iCPP[5:0], 5'b000000};
27
28             sUpdate1 :    if (rAddr_Curr < 131)
29                           rNum_Next <= {1'b0, iH[47:42], 5'b000000};
30             else if (rAddr_Curr == 131)
31                 rNum_Next <= {1'b0, iH[41:36], 5'b000000};
32             else if (rAddr_Curr == 132)
33                 rNum_Next <= {1'b0, iH[35:30], 5'b000000};
34             else if (rAddr_Curr == 133)
35                 rNum_Next <= {1'b0, iH[29:24], 5'b000000};
36             else if (rAddr_Curr == 134)
37                 rNum_Next <= {1'b0, iH[23:18], 5'b000000};
38             else if (rAddr_Curr == 135)
39                 rNum_Next <= {1'b0, iH[17:12], 5'b000000};
40             else if (rAddr_Curr == 136)
41                 rNum_Next <= {1'b0, iH[11:6], 5'b000000};
42             else if (rAddr_Curr == 137)
43                 rNum_Next <= {1'b0, iH[5:0], 5'b000000};
44
45             sUpdate2 :    if (rAddr_Curr < 171)
46                           rNum_Next <= {1'b0, iLV[47:42], 5'b000000};
47             else if (rAddr_Curr == 171)
48                 rNum_Next <= {1'b0, iLV[41:36], 5'b000000};
49             else if (rAddr_Curr == 172)
50                 rNum_Next <= {1'b0, iLV[35:30], 5'b000000};
51             else if (rAddr_Curr == 173)

```

```

52         rNum_Next <= {1'b0, iLV[29:24], 5'b000000};
53     else if (rAddr_Curr == 174)
54         rNum_Next <= {1'b0, iLV[23:18], 5'b000000};
55     else if (rAddr_Curr == 175)
56         rNum_Next <= {1'b0, iLV[17:12], 5'b000000};
57     else if (rAddr_Curr == 176)
58         rNum_Next <= {1'b0, iLV[11:6], 5'b000000};
59     else if (rAddr_Curr == 177)
60         rNum_Next <= {1'b0, iLV[5:0], 5'b000000};
61
62     sUpdate3 : if (rAddr_Curr < 211)
63         rNum_Next <= {1'b0, iMAR[47:42], 5'b000000};
64     else if (rAddr_Curr == 211)
65         rNum_Next <= {1'b0, iMAR[41:36], 5'b000000};
66     else if (rAddr_Curr == 212)
67         rNum_Next <= {1'b0, iMAR[35:30], 5'b000000};
68     else if (rAddr_Curr == 213)
69         rNum_Next <= {1'b0, iMAR[29:24], 5'b000000};
70     else if (rAddr_Curr == 214)
71         rNum_Next <= {1'b0, iMAR[23:18], 5'b000000};
72     else if (rAddr_Curr == 215)
73         rNum_Next <= {1'b0, iMAR[17:12], 5'b000000};
74     else if (rAddr_Curr == 216)
75         rNum_Next <= {1'b0, iMAR[11:6], 5'b000000};
76     else if (rAddr_Curr == 217)
77         rNum_Next <= {1'b0, iMAR[5:0], 5'b000000};
78
79     sUpdate4 : if (rAddr_Curr == 256)
80         rNum_Next <= {1'b0, iMBR[11:6], 5'b000000};
81     else if (rAddr_Curr == 257)
82         rNum_Next <= {1'b0, iMBR[5:0], 5'b000000};
83     else
84         rNum_Next <= 0;
85
86     sUpdate5 : if (rAddr_Curr < 291)
87         rNum_Next <= {1'b0, iMDR[47:42], 5'b000000};
88     else if (rAddr_Curr == 291)
89         rNum_Next <= {1'b0, iMDR[41:36], 5'b000000};
90     else if (rAddr_Curr == 292)
91         rNum_Next <= {1'b0, iMDR[35:30], 5'b000000};
92     else if (rAddr_Curr == 293)
93         rNum_Next <= {1'b0, iMDR[29:24], 5'b000000};
94     else if (rAddr_Curr == 294)
95         rNum_Next <= {1'b0, iMDR[23:18], 5'b000000};
96     else if (rAddr_Curr == 295)
97         rNum_Next <= {1'b0, iMDR[17:12], 5'b000000};
98     else if (rAddr_Curr == 296)
99         rNum_Next <= {1'b0, iMDR[11:6], 5'b000000};
100    else if (rAddr_Curr == 297)
101        rNum_Next <= {1'b0, iMDR[5:0], 5'b000000};
102
103    sUpdate6 : if (rAddr_Curr < 331)
104        rNum_Next <= {1'b0, iOPC[47:42], 5'b000000};
105    else if (rAddr_Curr == 331)
106        rNum_Next <= {1'b0, iOPC[41:36], 5'b000000};
107    else if (rAddr_Curr == 332)
108        rNum_Next <= {1'b0, iOPC[35:30], 5'b000000};
109    else if (rAddr_Curr == 333)
110        rNum_Next <= {1'b0, iOPC[29:24], 5'b000000};
111    else if (rAddr_Curr == 334)
112        rNum_Next <= {1'b0, iOPC[23:18], 5'b000000};
113    else if (rAddr_Curr == 335)
114        rNum_Next <= {1'b0, iOPC[17:12], 5'b000000};
115    else if (rAddr_Curr == 336)
116        rNum_Next <= {1'b0, iOPC[11:6], 5'b000000};
117    else if (rAddr_Curr == 337)
118        rNum_Next <= {1'b0, iOPC[5:0], 5'b000000};

```

```

119
120     sUpdate7 :    if (rAddr_Curr < 371)
121                   rNum_Next <= {1'b0, iPC[47:42], 5'b000000};
122     else if (rAddr_Curr == 371)
123                   rNum_Next <= {1'b0, iPC[41:36], 5'b000000};
124     else if (rAddr_Curr == 372)
125                   rNum_Next <= {1'b0, iPC[35:30], 5'b000000};
126     else if (rAddr_Curr == 373)
127                   rNum_Next <= {1'b0, iPC[29:24], 5'b000000};
128     else if (rAddr_Curr == 374)
129                   rNum_Next <= {1'b0, iPC[23:18], 5'b000000};
130     else if (rAddr_Curr == 375)
131                   rNum_Next <= {1'b0, iPC[17:12], 5'b000000};
132     else if (rAddr_Curr == 376)
133                   rNum_Next <= {1'b0, iPC[11:6], 5'b000000};
134     else if (rAddr_Curr == 377)
135                   rNum_Next <= {1'b0, iPC[5:0], 5'b000000};
136
137     sUpdate8 :    if (rAddr_Curr < 411)
138                   rNum_Next <= {1'b0, iSP[47:42], 5'b000000};
139     else if (rAddr_Curr == 411)
140                   rNum_Next <= {1'b0, iSP[41:36], 5'b000000};
141     else if (rAddr_Curr == 412)
142                   rNum_Next <= {1'b0, iSP[35:30], 5'b000000};
143     else if (rAddr_Curr == 413)
144                   rNum_Next <= {1'b0, iSP[29:24], 5'b000000};
145     else if (rAddr_Curr == 414)
146                   rNum_Next <= {1'b0, iSP[23:18], 5'b000000};
147     else if (rAddr_Curr == 415)
148                   rNum_Next <= {1'b0, iSP[17:12], 5'b000000};
149     else if (rAddr_Curr == 416)
150                   rNum_Next <= {1'b0, iSP[11:6], 5'b000000};
151     else if (rAddr_Curr == 417)
152                   rNum_Next <= {1'b0, iSP[5:0], 5'b000000};
153
154     sUpdate9 :    if (rAddr_Curr < 451)
155                   rNum_Next <= {1'b0, iTOS[47:42], 5'b000000};
156     else if (rAddr_Curr == 451)
157                   rNum_Next <= {1'b0, iTOS[41:36], 5'b000000};
158     else if (rAddr_Curr == 452)
159                   rNum_Next <= {1'b0, iTOS[35:30], 5'b000000};
160     else if (rAddr_Curr == 453)
161                   rNum_Next <= {1'b0, iTOS[29:24], 5'b000000};
162     else if (rAddr_Curr == 454)
163                   rNum_Next <= {1'b0, iTOS[23:18], 5'b000000};
164     else if (rAddr_Curr == 455)
165                   rNum_Next <= {1'b0, iTOS[17:12], 5'b000000};
166     else if (rAddr_Curr == 456)
167                   rNum_Next <= {1'b0, iTOS[11:6], 5'b000000};
168     else if (rAddr_Curr == 457)
169                   rNum_Next <= {1'b0, iTOS[5:0], 5'b000000};
170
171     sUpdate10 :   if (rAddr_Curr < 491)
172                   rNum_Next <= {1'b0, iData[47:42], 5'b000000};
173     else if (rAddr_Curr == 491)
174                   rNum_Next <= {1'b0, iData[41:36], 5'b000000}; m
175     else if (rAddr_Curr == 492)
176                   rNum_Next <= {1'b0, iData[35:30], 5'b000000};
177     else if (rAddr_Curr == 493)
178                   rNum_Next <= {1'b0, iData[29:24], 5'b000000};
179     else if (rAddr_Curr == 494)
180                   rNum_Next <= {1'b0, iData[23:18], 5'b000000};
181     else if (rAddr_Curr == 495)
182                   rNum_Next <= {1'b0, iData[17:12], 5'b000000};
183     else if (rAddr_Curr == 496)
184                   rNum_Next <= {1'b0, iData[11:6], 5'b000000};
185     else if (rAddr_Curr == 497)

```

```

186             rNum_Next <= {1'b0, iData[5:0], 5'b000000};
187
188         default : rNum_Next <= rNum_Curr;
189
190     endcase
191 end
192
193
194     always @(posedge iClk)
195     begin
196         rAddr_Curr <= rAddr_Next;
197         rNum_Curr <= rNum_Next;
198     end
199
200     assign oAddr = rAddr_Curr;
201     assign oWe = (rFSM_Curr == sIdle
202         || rFSM_Curr == sInit
203         || rFSM_Curr == sUpdate0
204         || rFSM_Curr == sUpdate1
205         || rFSM_Curr == sUpdate2
206         || rFSM_Curr == sUpdate3
207         || rFSM_Curr == sUpdate4
208         || rFSM_Curr == sUpdate5
209         || rFSM_Curr == sUpdate6
210         || rFSM_Curr == sUpdate7
211         || rFSM_Curr == sUpdate8
212         || rFSM_Curr == sUpdate9
213         || rFSM_Curr == sUpdate10) ? 1 : 0;
214     assign oData = rNum_Curr;
215
216 endmodule

```

A.3 Python script for making the Static Screen file

Listing A.9: Static Screen code

```

1  from tkinter.filedialog import askopenfilename
2  from pathlib import Path
3
4  downloads_path = str(Path.home() / "Downloads")
5
6  lookuptable = {
7      " ": 0,
8      "!" : 32,
9      "\"": 64,
10     "#": 96,
11     "$": 128,
12     "%": 160,
13     "&": 192,
14     "\": 224,
15     "(": 256,
16     ")": 288,
17     "*": 320,
18     "+": 352,
19     ",": 384,
20     "-": 416,
21     # Dot is 448, but to make it easier to make your initial file we put a dot
       # for a space
22     ".": 0,
23     "/" : 480,
24     "0": 512,
25     "1": 544,
26     "2": 576,

```

```

27      "3": 608,
28      "4": 640,
29      "5": 672,
30      "6": 704,
31      "7": 736,
32      "8": 768,
33      "9": 800,
34      ":": 832,
35      ";": 864,
36      "<": 896,
37      "=": 928,
38      ">": 960,
39      "?": 992,
40      "@": 1024,
41      "A": 1056,
42      "B": 1088,
43      "C": 1120,
44      "D": 1152,
45      "E": 1184,
46      "F": 1216,
47      "G": 1248,
48      "H": 1280,
49      "I": 1312,
50      "J": 1344,
51      "K": 1376,
52      "L": 1408,
53      "M": 1440,
54      "N": 1472,
55      "O": 1504,
56      "P": 1536,
57      "Q": 1568,
58      "R": 1600,
59      "S": 1632,
60      "T": 1664,
61      "U": 1696,
62      "V": 1728,
63      "W": 1760,
64      "X": 1792,
65      "Y": 1824,
66      "Z": 1856,
67      "[": 1888,
68      "\\": 1920,
69      "]": 1952,
70      "^": 1984,
71      "`": 2016,
72      "\'": 2048,
73      "a": 2080,
74      "b": 2112,
75      "c": 2144,
76      "d": 2176,
77      "e": 2208,
78      "f": 2240,
79      "g": 2272,
80      "h": 2304,
81      "i": 2336,
82      "j": 2368,
83      "k": 2400,
84      "l": 2432,
85      "m": 2464,
86      "n": 2496,
87      "o": 2528,
88      "p": 2560,
89      "q": 2592,
90      "r": 2624,
91      "s": 2656,
92      "t": 2688,
93      "u": 2720,

```

```

94     "v": 2752,
95     "w": 2784,
96     "x": 2816,
97     "y": 2848,
98     "z": 2880,
99     "{": 2912,
100    "|": 2944,
101    "}": 2976,
102    "~": 3008
103 }
104
105
106 def makefile(hex_char):
107     file1 = open(downloads_path + "/mystatic.txt", "w")
108     file1.writelines(hex_char)
109     file1.close()
110
111
112 def convert(file):
113     # Convert file
114     hex_char = []
115     for i in range(len(file)):
116         dec = lookuptable.get(file[i])
117         # Remove 0x and add an enter after line
118         hex_char.append(str(hex(dec)[2:] + "\n"))
119
120     makefile(hex_char)
121
122
123 def choosefile():
124     # Choose the file you want to convert
125     filename = askopenfilename()
126     file = open(filename, "r").read().replace("\n", "")
127     print(file)
128
129     convert(file)
130
131
132 if __name__ == '__main__':
133     choosefile()

```

A.4 ScreenBufferMem module code

Listing A.10: ScreenBufferMem module code

```

1  'timescale 1ns / 1ps
2
3  module ScreenBufferMem #(
4      parameter WIDTH = 12,
5      parameter DEPTH = 600
6  )
7  (
8      input wire iClk,
9      input wire [$clog2(DEPTH)-1:0] iAddrA, iAddrB,
10     input wire [WIDTH-1:0] iDataB,
11     input wire iWeB,
12     output wire [WIDTH-1:0] oDataA, oDataB
13 );
14
15 // define the memory
16 reg [WIDTH-1:0] rMem [DEPTH-1:0];
17
18 // Initial contents of the memory

```



```

19  initial
20  begin
21      $readmemh("static_mems.mem", rMem);
22  end
23
24  // Logic for Port A
25  // Supports only synchronous reading
26  reg [WIDTH-1:0] rDataA;
27
28  always @(posedge iClk)
29  begin
30      rDataA <= rMem[iAddrA];
31  end
32
33  assign oDataA = rDataA;
34
35  // Logic for Port B
36  // Supports synchronous reading and writing
37  reg [WIDTH-1:0] rDataB;
38
39  always @(posedge iClk)
40  begin
41      if (iWeB)
42          rMem[iAddrB] <= iDataB;
43          rDataB <= rMem[iAddrB];
44  end
45
46  assign oDataB = rDataB;
47
48  endmodule

```
