FPGAcebook Messenger

Dylan Peters, Justin Havas, Noah Pritt

December 7, 2018

Contents

1 Introduction

Our project is a Facebook Messenger-style application that allows 2 FPGA boards to communicate with each other by sending text and audio data. Two users can each plug into their FPGA a keyboard, microphone, speakers, VGA display, and optionally an "emoji keyboard." The FPGA takes the input from each of these peripheral devices, processes it, then sends it over a GPIO cable to the other FPGA where it is displayed for your friend to LOL or ROFL at.

2 Project Design

The original project design can be seen in the diagram shown below in Figure 1. This diagram shows that each input - keyboard, microphone, and camera - and each output - speaker and monitor - communicates with a controller, before sending its data through the processor. In this way, the controllers, implemented in Verilog, act as hardware "drivers" for the processor.

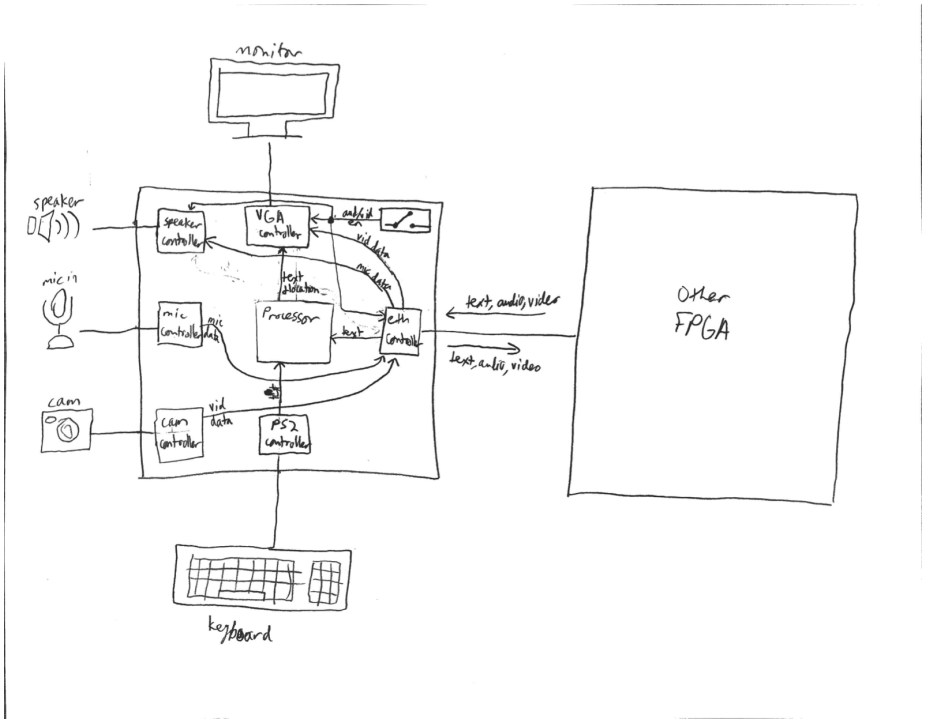


Figure 1: Original Project Design.

The final project was very similar in design to the original. A few notable changes are:

1. We did not implement video as we originally intended. Therefore, in the final project, there is no camera input and no video controller.

2. We added an additional custom input that we had not originally thought of. We created our own "emoji keyboard," which has 3D-printed emoji characters that can be pressed to type an emoji. This input device has a corresponding controller that eventually pipes its data into the processor.

The final version of the project is broken down into the following sections:

2.1 PS2 Keyboard

We created a controller to interface with a keyboard via the PS2 protocol. The controller crosses the clock domains between the PS2 keyboard and the processor. It reads the serial data from the keyboard, saves it into a register, maps the value to an ascii value, and saves the ascii value to a register that is exposed to the processor. The PS2 controller marks the ascii value as valid and then waits for the processor to assert that the ascii value has been read before listening for the next key input.

2.1.1 Inputs

1. **ps2_clock:** a clock produced by the keyboard, at a different frequency than the processor clock

2. **ps2_code:** a code produced by the keyboard when a letter is pressed or released

2.1.2 Outputs

1. **ascii:** 8-bit character buffer, containing the ascii code of the key that was pressed.

2. **text_ready:** flag telling the processor that the ascii code on **ascii** is valid.

2.2 Emoji Keyboard

We created an emoji keyboard that can be used to input special characters to the processor. The controller uses pull-down resistor networks to change the values of the GPIO pins when the pins are pressed.

2.2.1 Inputs

4 GPIO Pins: Based on the DE2-115 spec sheet, we concluded that the GPIO pins are pulled high internally when left floating. We also calculated the proper resistor values to pull the pins low while respecting the current limits of the pins.

2.2.2 Outputs

We added a Verilog controller that detects the change in the pin values, adds the corresponding emoji to the processor's input buffer, sends it to the other FPGA, and displays it on screen.

2.3 GPIO Communication

A central aspect of our project is communication between two separate FPGA boards. We chose to implement this communication by connecting the main GPIO banks of the two processors together.

The GPIO communication is handled by a GPIO Transmit module and a GPIO Receive module. The two boards must have opposite pin assignments for their transmit and receive pins in order to ensure that they are not both transmitting on the same pins at the same time.

The GPIO operates at a much slower clock frequency than the processor because the signal must traverse the length of the ribbon cables and stabilize at the other board's receive pins before being changed.

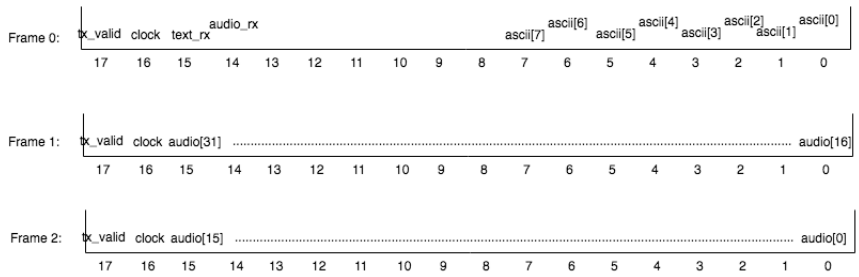


Figure 2: A single GPIO packet, showing the multiple frames required to send text and audio data.

2.3.1 Inputs

The inputs to the GPIO communication are:

1. 18 GPIO pins: 18 of the GPIO pins are used to receive data from the other board
2. text_tx: 8-bit bus containing the ascii code to transmit
3. audio_tx: 32-bit bus containing the audio value to transmit

2.3.2 Outputs

The outputs of the GPIO communication are:

1. 18 GPIO pins: 18 of the GPIO pins are used to transmit data to the other board
2. text_rx: 8-bit bus containing the ascii code of the text that was received
3. audio_rx: 32-bit bus containing the audio value that was received

2.4 Audio

The project takes in audio using a microphone on each board and transmits the audio to the other board, in order to allow users to "call" each other. It also plays sounds when a message is sent or received and when an emoji key is pressed.

2.4.1 Inputs

1. Microphone inputs in the mic port.
2. Key presses of the emoji keys
3. Send/receive signals

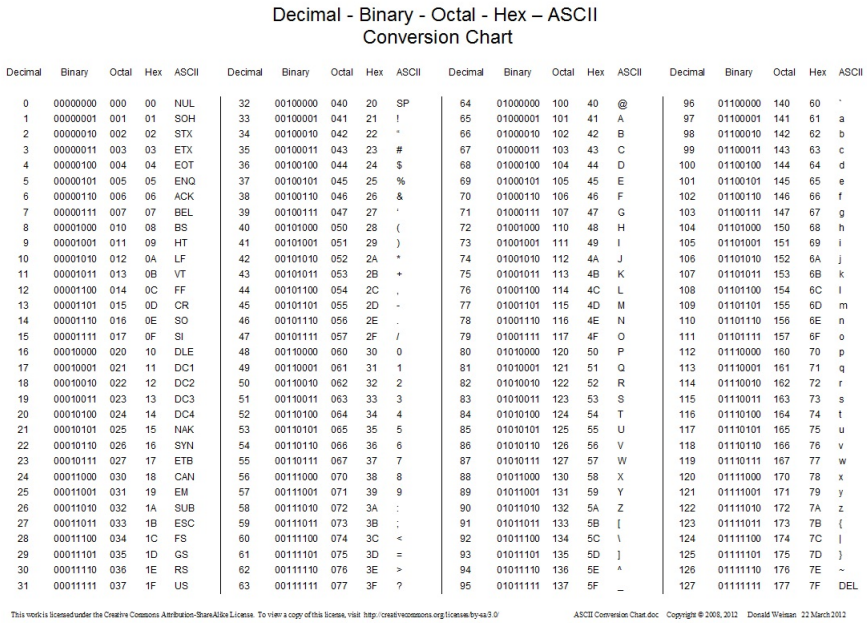
2.4.2 Outputs

1. 32-bit audio output to the "Line Out" port.

2.5 VGA/Graphics

2.5.1 Inputs

There are two inputs to the graphics part of the project: a 6-bit wire containing a horizontal counter and an 8-bit ascii code. This represents an input from the keyboard with a key input corresponding to the ascii code. The counter represents what part of the message this key was part of. No y-counter was needed, as we made the design decision to limit messages to one line.



Decimal - Binary - Octal - Hex – ASCII
Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Figure 3: Ascii Codes.

2.5.2 Creating .mif files from pictures

We decided to represent graphics via the vga with .mif files. Each picture had two corresponding .mif files. One listed each color that appears in the photo, while the other one mapped each pixel to one of those colors. In this way, we were able to store images in memory. Having the index file only list the colors that appear instead of the color for every single pixel saved memory. Similarly, we created a script that would "blend" colors in a photo together. Colors with similar rgb values were combined into one of those colors. This resulted in much less memory being used, but the quality did not decrease by very much. We also created scripts to

The project uses VGA to display graphics on a monitor. We chose our background as an image of facebook messenger. We then defined the area for text as a 500 X 400 space. We called this space "grid" throughout our project. We decided to present letters as 10x10 pixels. Thus, we can fit 50 words on a line and 40 lines in the grid. We created a ram memory element to match each space in the 50x40 grid with the ascii value to be displayed at that space. We then defined an iterator counter called ADDR to iterate through each pixel of the vga. We wrote an algorithm to determine for each pixel whether it is in the

grid or not. If it is in the grid, the corresponding ascii code is looked up from memory, and then the correct pixel number for that letter was determined. At that point, the corresponding mif code tells us which color the pixel should be.

3 Processor Changes

The processor for this final project had to undergo several changes in order to effectively be in control of all PS2 communication. The first was making an instruction that checked if a character was available from the PS2. This instruction would signal to the whether to compute the x,y coordinate address for the incoming character to be placed at. There were other custom instructions made that were similar this such as an instruction that checked if GPIO was transmitting, an instruction that checked if GPIO was receiving, and an instruction that checked if the incoming character from the PS2 was an enter.

Next, we needed to make sure that we stored the character into memory, so that it could be remembered when we began transmission. In order to store the character, I needed to create another custom instruction that loaded the character into a register. There was one other custom instruction used that was similar to this which loaded the incoming character being received into a register.

After the character was stored in memory, the processor needed to output the address that was just calculated for the incoming character for the VGA to use. This was done with an instruction that would output the character and x,y coordinates into a separate register in the processor. Once we wrote to this register, the processor would reset the PS2, so that the character that had just been read would not be read again. This resetting of the PS2 was done with another custom instruction. Finally, there was one more instruction that was used in order to output to the transmission register. This instruction was necessary so that the GPIO transmitter knew which character was to be sent at a given time.

4 Challenges Faced

4.1 Ethernet

One challenge that we faced was attempting to implement ethernet communication between boards. We spent approximately 2 weeks attempting to get FPGAs to communicate over the ethernet protocol but could not figure it out. This was a problem because it caused us to waste 2 weeks on something that did not give us any points.

We tried the following things to get ethernet to work:

1. Following online tutorials from Intel
2. Analyzing the spec sheet of the FPGA's ethernet controller.

3. Using different boards and ethernet cables.

4.2 GPIO

Another challenge we faced was attempting to get boards to communicate via the GPIO pins. The problem turned out to be a power issue. The solution was to power both boards from the power port and then connect every wire between the two boards except for the 5V and 3.3V power pins, in order to prevent voltage differences.

4.3 Text animation

Another challenge we faced was text movement. We wanted text displaying on the vga as it is displayed on facebook messenger. The text currently being entered should appear on a special line for text input, and as messages are sent and received the lines of text should move upwards. Implementing this required a second memory unit. It normally saves ascii values and addresses in the same way as the original memory unit. After a new message is received or sent, every line of code needs to shift upwards. In order to achieve this, all data stored in the primary memory unit has to be backed up 50 addresses, as 50 is the length of a line. The second memory unit is used to access these values as the addresses of the main unit change.

This task proved more challenging than we expected because of timing issues. We implemented it in a way that we should would work, and it was successful some of the time.

5 Testing

5.1 Waveforms

For processor-intensive tasks, we used waveforms frequently in order to detect when the processor began to exhibit behavior that was different from the expected behavior. Typically, we would output and examine the register-write-index, register-write-data, register-write-enable, and dmem-write-data buses. Examining these buses and comparing them to the expected register values helps to pinpoint exactly which line in the MIPS code is incorrect.

5.2 7-Segment Displays

In order to test or debug modules that interface with peripheral devices, we utilized the 7-segment displays in order to see the values of certain wires that we suspected carried the incorrect values.

6 Assembly Programs

6.1 Send and Receive Loop

The transmission and reception of characters inputted from the PS2 is all handled by the processor. Transmission began in the processor when the incoming character from the PS2 was an enter. Allowed the processor to start reading from the memory which held all the characters written in the past. How it worked was there were two registers in the register file that held the value of the address in memory we needed to access and the byte of memory within the word that held the character that we wanted to transmit. The processor would then load the specific word from memory, extract the character that it needed to access, and then place this character in a register outside of the register file that the GPIO transmitter could access. It then sent a signal to the GPIO transmitter to begin transmitting. The processor would then wait until the GPIO transmitter was done transmitting before it transmitted the next character. This process was repeated until the processor read an enter character from memory which signalled the end of a message. This enter character would be transmitted, but would then tell the processor the stop the transmission process and start checking for data from the PS2 or being received.

The reception process began in the same while loop as the that was checking if PS2 data was ready. How this worked was the processor was constantly be checking if it was getting input from the PS2 or from the GPIO receiver. If it was getting input from the receiver, the processor would calculate an address for the incoming character to be placed at on the VGA screen. It when then send the address and character to the same register that addresses computed for PS2 characters were written to, so that the VGA would know to display the received character. The processor would then wait for the receiving signal to go low, so that it wouldn't compute an address for the same character again. This process would then be repeated until an enter character was received by the GPIO receiver which would tell the processor to setup the address for the next character either to be received or typed.

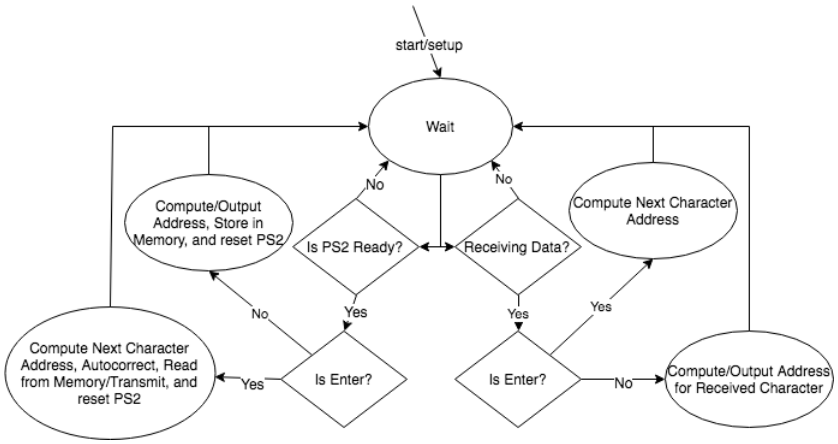


Figure 4: Send and Receive algorithm.

6.2 Autocorrect

In order to implement an autocorrect algorithm, I first downloaded a list of the 1,000 most common words in English. I wrote a Python script that reads these words, translates their characters into hexadecimal ascii representations, and adds them to a mif file. This "dictionary" takes up the second half of the 16kb dmem of the processor. When the user types a sentence, the processor keeps track of the beginning of that sentence. Upon hitting enter, before transmitting the message, the processor calls the autocorrect "method," which reads the message from memory, compares each word to the dictionary, and replaces any misspelled words with the correct spelling.

The autocorrect algorithm loops through each word in the sentence and performs the following operations:

1. Compare the unknown word to each word in the dictionary.
2. If any word in the dictionary is an exact match, return early.
3. If the unknown word has a near-match - defined as 1 letter difference - with one or more words in the dictionary, return the alphabetically-first near-match.
4. If the unknown word has no matches or near-matches in the dictionary, return the unknown word.

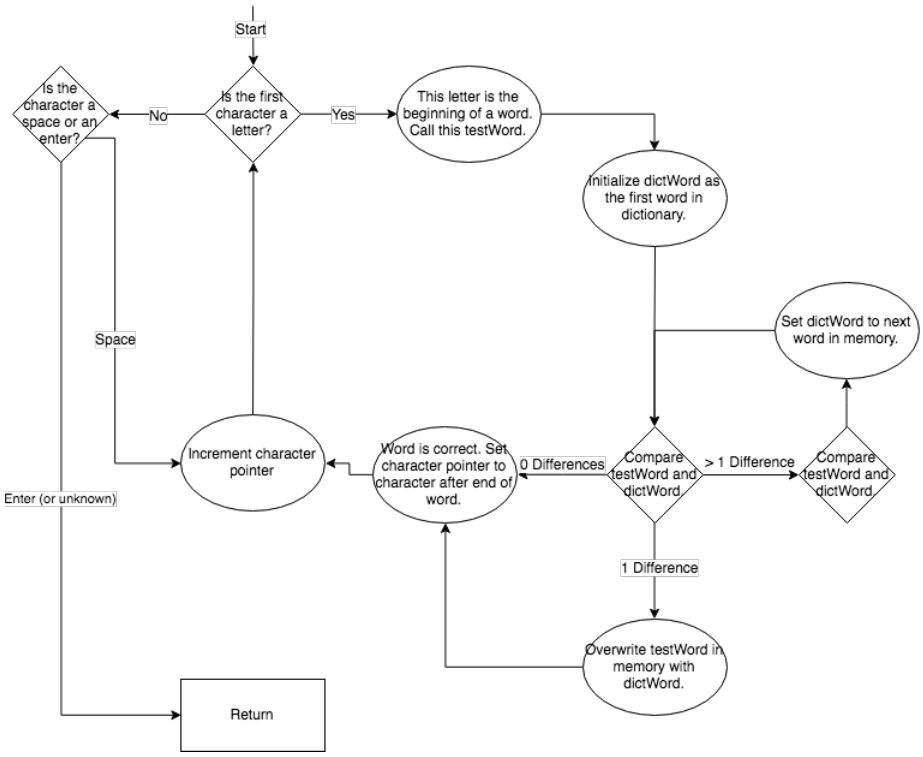


Figure 5: Flow chart of autocorrect algorithm.

7 Pictures

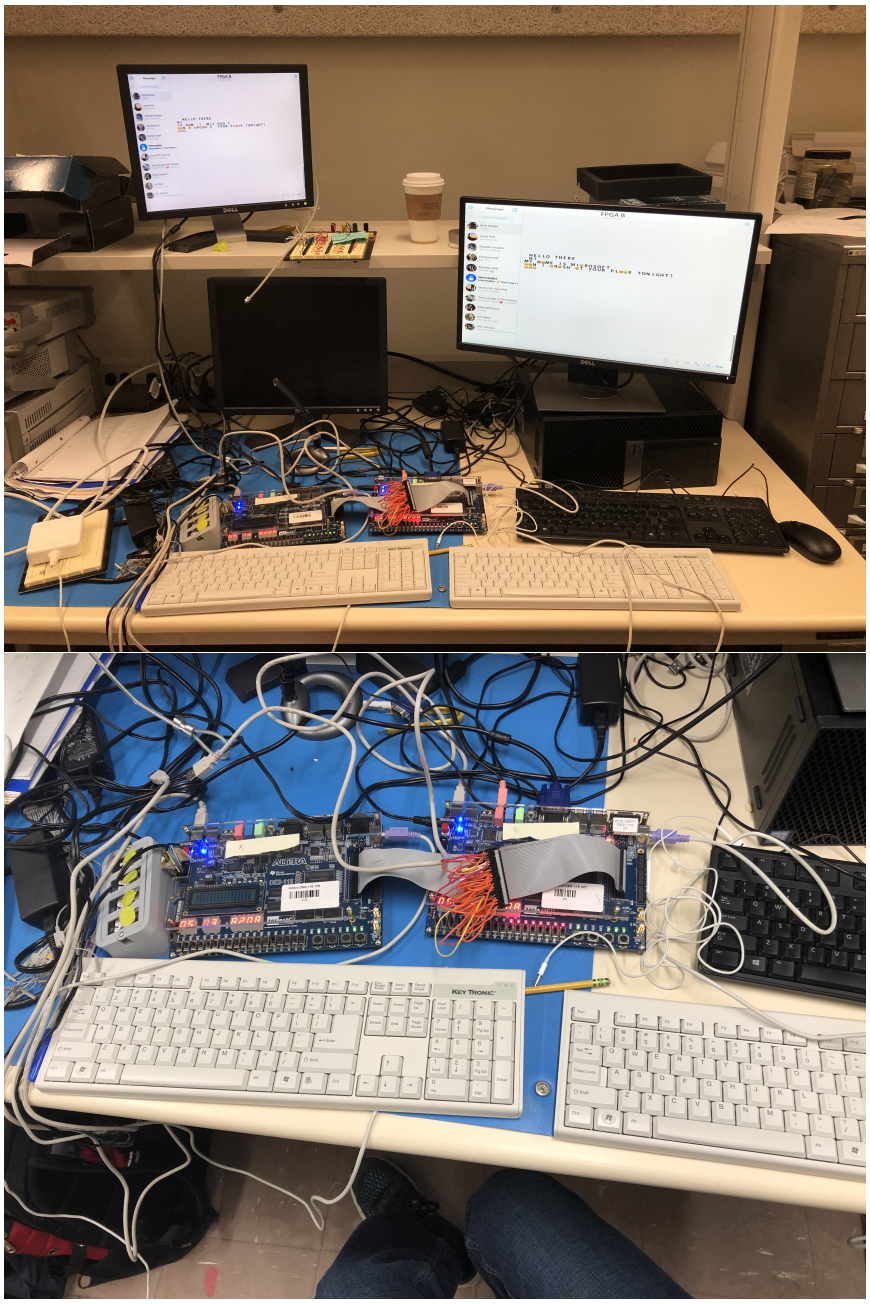


Figure 6: Final working project.

8 Next Steps

There are a number of features that we would like to add to our project but simply did not have the time or resources to do so. This feature backlog includes:

1. Interface with an SD card in order to store a larger dictionary for autocorrect. This would result in fewer incorrect autocorrect behaviors.

2. Draw colored bubbles behind the messages, so that it is easier to differentiate who sent each message.

3. Interface with ethernet in order to allow boards to communicate over longer cables. This would also allow multiple boards to be connected via a switch in order to create a group chat.

4. Decode video data from a webcam and send it over GPIO in order to create a video chat.

5. Add a talkback feature that says each letter when you press it in order to improve accessibility.