Interfacing with a 6551 UART

Drake Hayes

May 2, 2023

Abstract

This project involves connecting an Arduino to a 6502 computer and sending messages from the Arduino to the 6502 computer to be displayed on its LCD screen using asynchronous serial communication. We achieve this asynchronous serial communication by using the 6551 UART chip.

1 Introduction

The 6502 microprocessor, which first hit the market in the 80s, continues to sell hundreds of thousands a year, which makes it a worthwhile study. In this project, we expanded upon Ben Eater's 6502 computer to be able to interface with a modern Arduino. Connecting our Arduino to our iteration of Eater's 6502 computer's 6551 UART chip, we send messages to it and it prints them on its LCD display. We address the marketability of our project as well, pointing out that since hundreds of thousands of 6502 microprocessor chips sell a year, it's necessary being able to adapt them to more modern technology.

2 Literature Review

Ben Eater's tutorials on how to build and program a 6502 computer were foundational to this project. In his videos he explains how to wire up the 6502 microprocessor chip with the RAM, ROM, virtual interface adapter, and the LCD display and how to write 6502 assembly code that tells the 6502 computer to write "Hello World!" to its screen. In his most recent videos, he also explains how to connect a 6551 UART chip to this circuit. It's my hope in this paper to expand upon Eater's tutorials and explain how one might connect an Arduino to this 6502 computer.

Mike Billington's software blog was also useful in this project. In his post "Adding a Serial Port to My 6502 Computer," he explains how he connected a 6551 UART chip to his implementation of Eater's 6502 computer and how he was able to connect his PC to it. His post was particularly useful in figuring out how to write the software for the 6551 UART chip. After connecting his PC to his 6502 computer, he wrote a program which accepts user input from the PC and displays the typed messages on the LCD screen.

Lee Davidson's work (which can be accessed through on the ACIA 6551 – Retro Computing website) with 6551 serial communication was also helpful in figuring out the software, providing code for transmitting and receiving data.

The 65c02 Assembly Wikibooks and 6502.org Tutorial and Aids websites were similarly useful resources. They provide lists of the different 6502 op-codes and detailed descriptions of each, which were helpful in both writing my own software and in understanding that provided by Eater and Billington.

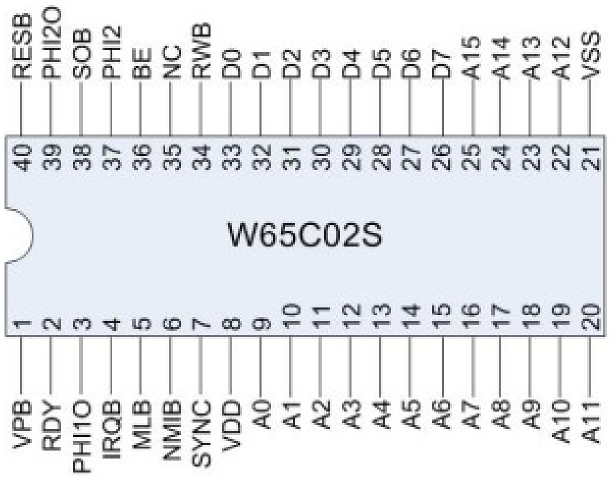
Honeywell's documentation on RS232 standard versus RS232 TTL was useful in determining that we did not need the MAX232 chip for the Arduino and that we could simply connect the Arduino directly to the 6551 UART chip.

Campbell provides a good explanation of how UART communication works in general, demonstrating how the transmitter of one must be connected to the receiver of the other. She also points out conditions to be mindful of when transmitting data using UARTs; both should expect the same type of data with the same number of data bits, stop bits, and parity bits, and they both should be communicating at the same "speed" or baud rate.

Mallari's article "How to Set Up UART Communication on the Arduino" was helpful in figuring out both how to connect the Arduino to the 6502 computer and how to program it to transmit data across its serial port.

3 Theory

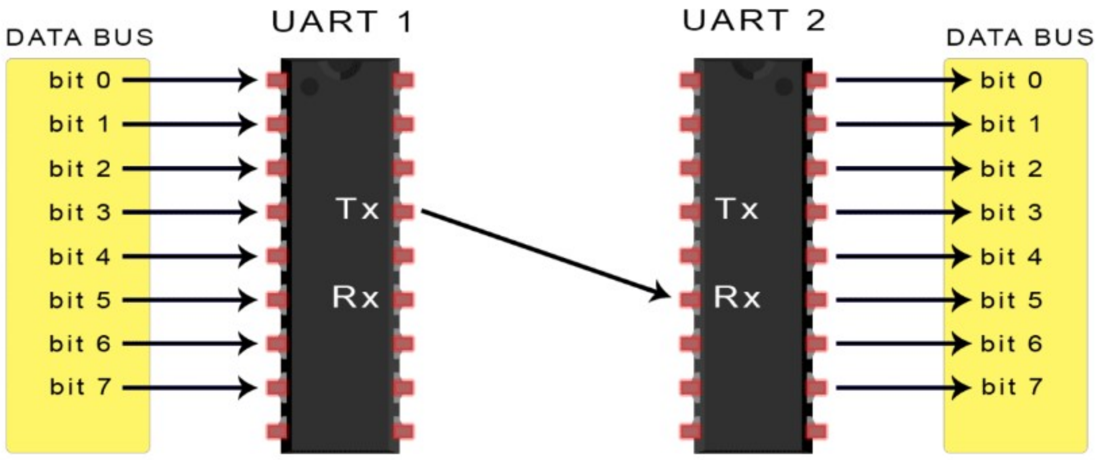
4 6502 microprocessor



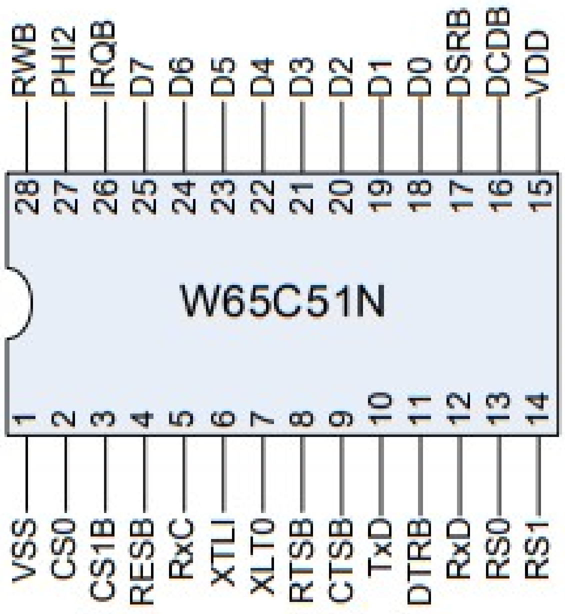
Our circuit includes a 6502 microprocessor. This is the same microprocessor used in the Commodore 64, Commodore PET, Apple 1, and the original Famicom/NES. Like any other microprocessor, the 6502 chip has its own op codes, which allows us to write programs to it. Pictured here is its pin out.

4.1 UART

UART stands for universal asynchronous receiver-transmitter. It is synonymous with ACIA, which stands for asynchronous communications interface adapter. UARTs are used in asynchronous serial communication between devices.



UARTs are capable of transmitting and receiving data. Connecting two UARTs together, it's important to connect the transmitter of the one to the receiver of the other and the transmitter of the other to the receiver of the one as shown. The transmitting UART receives data in parallel form and converts it to serial data, adding start, stop, and parity bits, which indicate the type and size of data being transmitted. When the receiving UART receives this data packet, it converts it removes the start, stop, and parity bits and converts it back into parallel form. Then, it sends this data to its CPU across its data bus.



Our 6502 computer is using a 6551 UART chip to communicate with our Arduino. Pictured here is the pin out.

4.2 RS232 Standard versus RS232 TTL

RS232 describes recommended standard for serial communication. There are two primary types of RS232: RS232 standard and RS232 TTL. RS232 standard operates on a -15V to +15V scale, where logical low falls within the voltage ranges of -15V to -3V and where logical high falls within the voltage ranges of 3V to 15V. RS232 TTL on the other hand operates on a 0V to 5V scale, where logical low falls within the ranges of 0V to 0.4V and logical high falls within the ranges of 3V to 5V. Both our Arduino and 6502 computer use RS232 TTL. This has allowed us to connect our Arduino directly to our 6502 computer's 6551 UART chip

4.3 Chip Select Logic

Our 6502 computer uses a series of duel-input NAND gates to handle the chip selection process. Most computers implement a separate chip to handle this process, but since our computer is relatively simple, we can get away with using Boolean logic to parse out which chips to write to and read from. Implementing these NAND gates in this way garners a simple chip selection process; however, it does this at the expense of memory. So, we can memory map our 6551 UART chip to either addresses \$4000 to \$4FFF or addresses \$5000 to \$5FFF. In our code, we've written it to addresses \$5000 to \$5FFF. But, one could just as easily write it to addresses \$4000 to \$4FFF.

4.4 ACIA Registers

Our 6551 UART chip has four registers: status, data, command, and control. The status register relates information about the state of the 6551 chip. Bits 0, 1, and 2 record whether an error has occurred. Bit 3 indicates whether or not the receiver data register is full. Bit 4 the transmitter data register. Bits 5 and 6 indicate the states of the data carrier and data set ready respectively, and bit 7 indicates if an interrupt has occurred.

The command register controls a number of modes and functions of the 6551 chip. Bit 0 sets data terminal ready or not; bit 1 enables the receiver interrupt request; bits 2 and 3 handles the transmitter interrupt status; bit 4 enables receiver echo mode; bit 5 enables parity mode; and bits 6 and 7 handles the parity mode control.

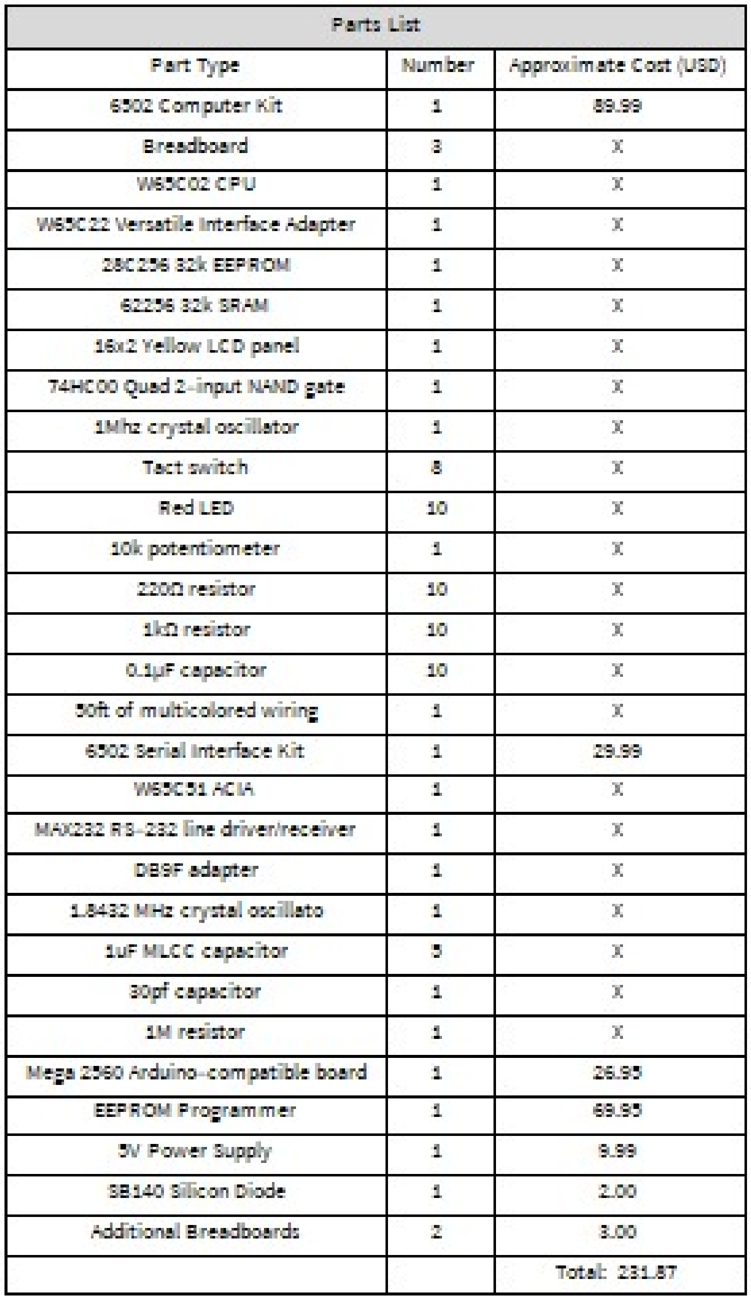
The control register determines the type and speed of the data being transmitted and received between the UARTs. Bits 0 through 4 determine the data's baud rate, which is the speed in bits per second at which it's being sent back an forth. Bits 5 and 6 determine the length of the data being sent, and bit 7 defines the number of stop bits.

4.5 6551 Hardware Bug

The 6551 UART chip has some hardware bugs which prevent the forth bit of the status register from operating as we'd expect. The baud clock appears not to oscillate, which impairs its ability to transmit data. There exist some software workarounds, but transmitting data from the 6551 UART chip is not as straightforward as sending data to it.

5 Setup

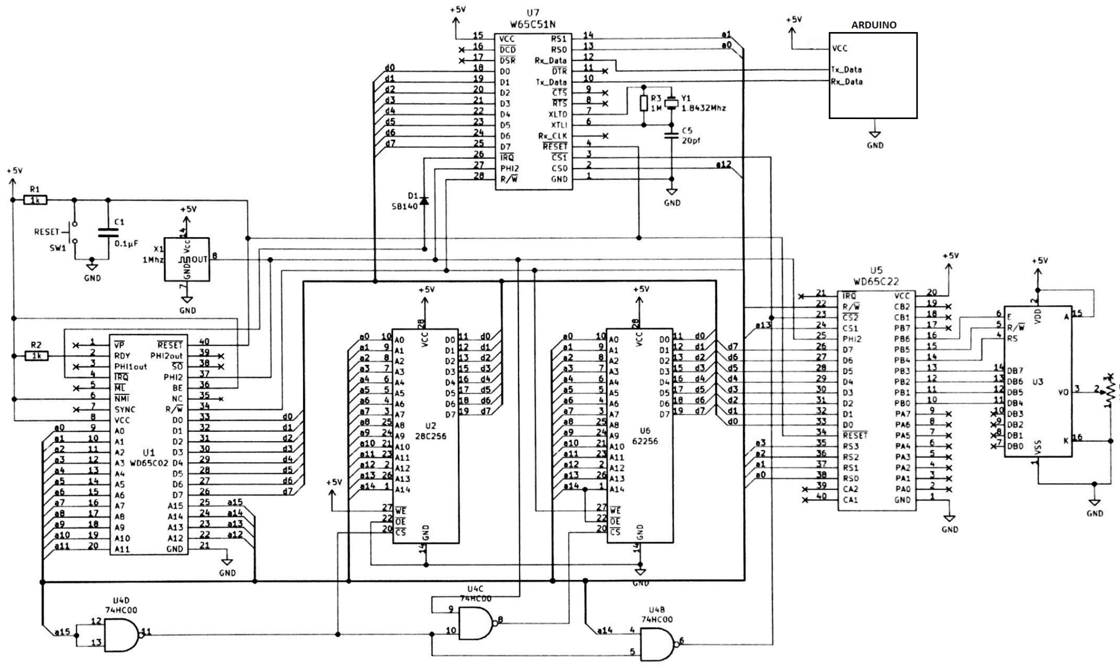
5.1 Materials



Parts List		
Part Type	Number	Approximate Cost (USD)
6502 Computer Kit	1	89.99
Breadboard	3	X
W65C02 CPU	1	X
W65C22 Versatile Interface Adapter	1	X
28C256 32k EEPROM	1	X
62256 32k SRAM	1	X
16x2 Yellow LCD panel	1	X
74HC00 Quad 2-input NAND gate	1	X
1Mhz crystal oscillator	1	X
Tact switch	8	X
Red LED	10	X
10k potentiometer	1	X
220Ω resistor	10	X
1kΩ resistor	10	X
0.1μF capacitor	10	X
50ft of multicolored wiring	1	X
6502 Serial Interface Kit	1	29.99
W65C51 ACIA	1	X
MAX232 RS-232 line driver/receiver	1	X
DB9F adapter	1	X
1.8432 MHz crystal oscillato	1	X
1uF MLCC capacitor	5	X
30pf capacitor	1	X
1M resistor	1	X
Mega 2560 Arduino-compatible board	1	26.95
EEPROM Programmer	1	69.95
5V Power Supply	1	9.99
SB140 Silicon Diode	1	2.00
Additional Breadboards	2	3.00
		Total: 231.87

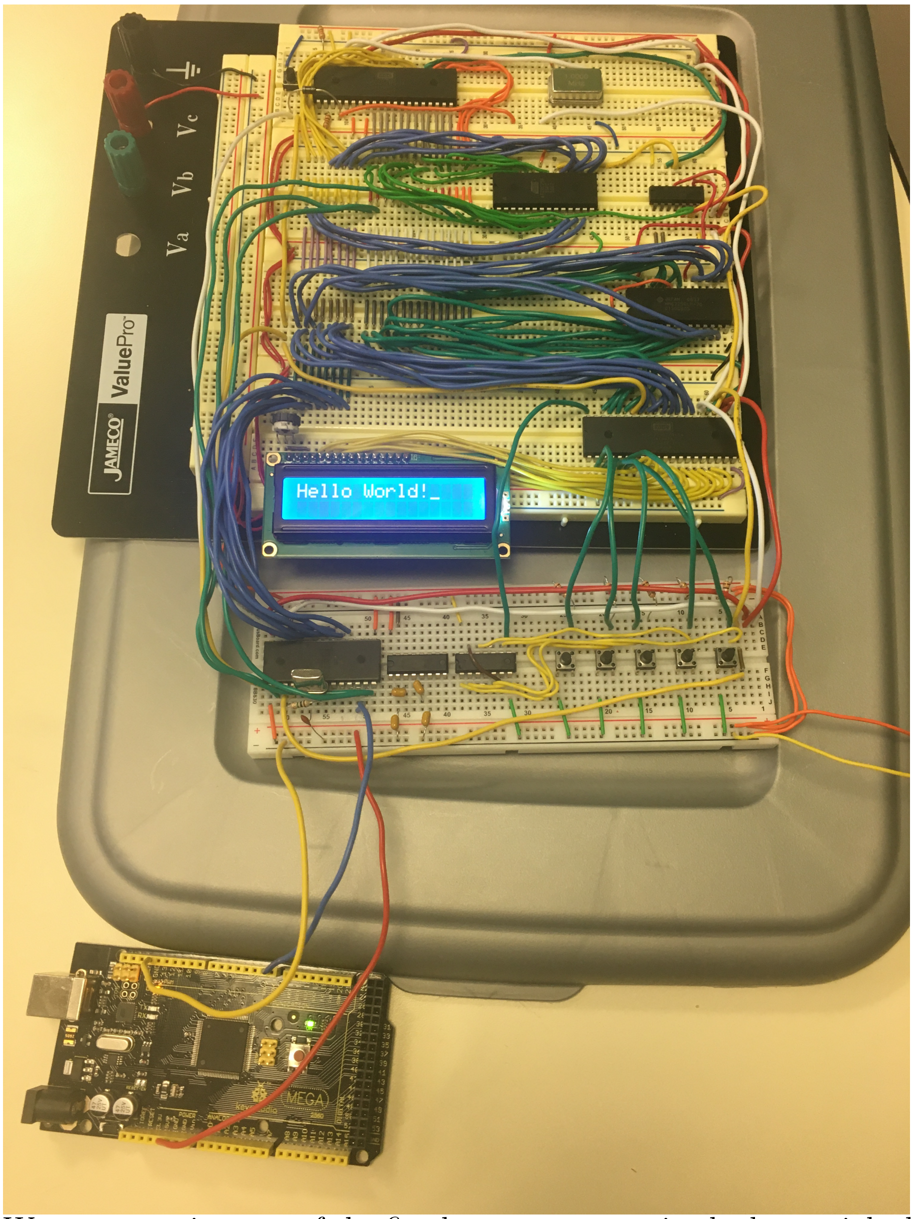
I highly recommend purchasing Ben Eater's 6502 Computer Kit and 6502 Serial Interface Kit. These two kits contain all of the materials needed to construct the 6502 computer as well as LEDs for testing and additional capacitors, resistors, and switches. I ended up not using the MAX232 chip or DB9F adapter, but these components would be useful for interfacing a PC instead of an Arduino with the 6502 computer.

5.2 Circuit Schematic



This schematic is an adaptation of Ben Eater's schematic of the 6502 CPU with 6551 UART and MAX232 chips.

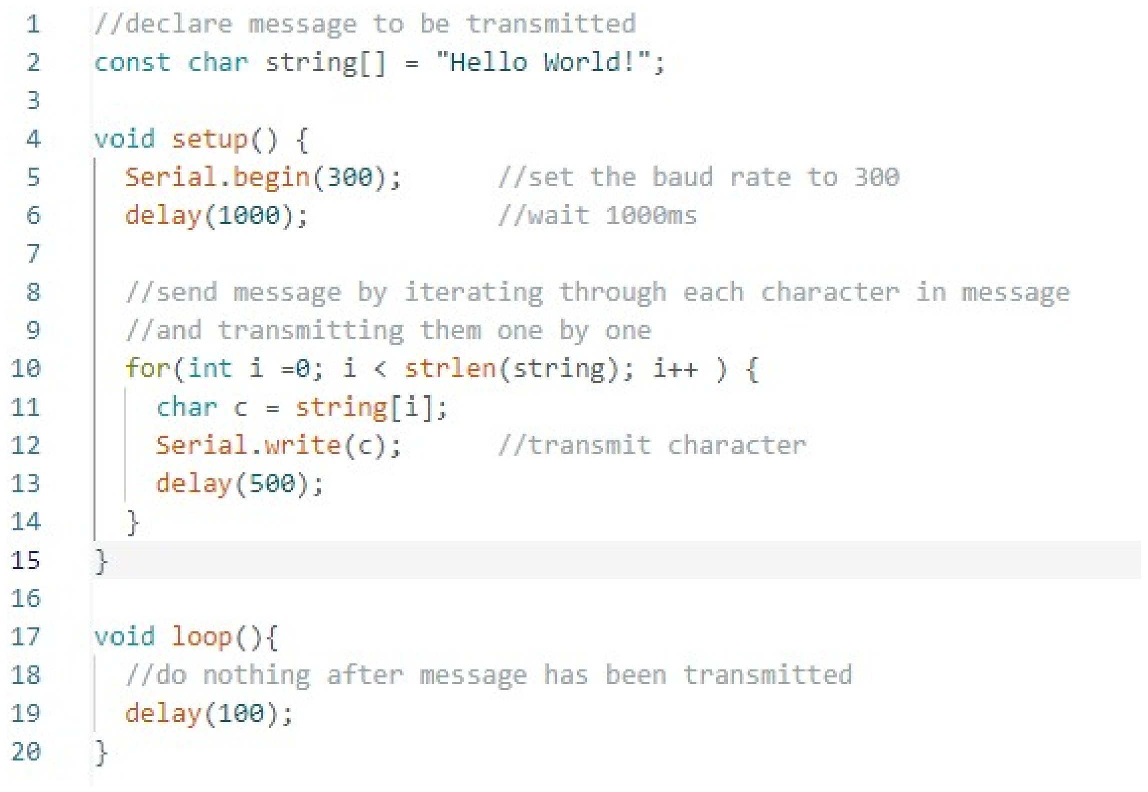
5.3 Breadboard



We are not using any of the five buttons present in the lower right-hand corner of the board, nor are we using the MAX232 or 74HC08 chips located to the right of the 6551 UART chip. Notice, however, in our schematic we've connected pin 10 of our 6551 UART chip to the RX pin of our Arduino; we don't have that here because no data is being transmitting back to our Arduino.

6 Software

6.1 Arduino Code



```
//declare message to be transmitted
const char string[] = "Hello World!";

void setup() {
  Serial.begin(300);        //set the baud rate to 300
  delay(1000);              //wait 1000ms

  //send message by iterating through each character in message
  //and transmitting them one by one
  for(int i =0; i < strlen(string); i++ ) {
    char c = string[i];
    Serial.write(c);        //transmit character
    delay(500);
  }
}

void loop(){
  //do nothing after message has been transmitted
  delay(100);
}
```

We start our Arduino code by declaring what message we would like to send to the 6502 computer. We can make this message anything we would like. I've chosen the message: "Hello World!"

Most of what happens on the Arduino end happens in the voidsetup() loop. We've set the baud rate to be 300. We do this first because, in order to transmit and receive data, the Arduino and 6502 computer must be talking at the same speed. After a small delay, the Arduino starts sending data to the 6502 computer. Using a for loop, we iterate through each character in our string and send them one at a time.

Once the Arduino has sent the last character in our string, it enters the voidloop() loop. At this point, the program has finished; the Arduino is going to sit in this loop until it has been reset.

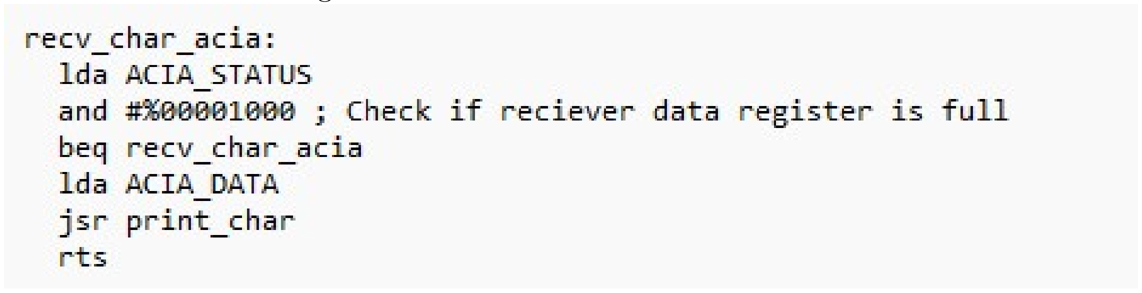
6.2 6502 Code

```
PORTB = $6000
PORTA = $6001
DDRB = $6002
DDRA = $6003

ACIA_DATA = $5000
ACIA_STATUS = $5001
ACIA_CMD = $5002
ACIA_CTRL = $5003

E  = %10000000
RW = %01000000
RS = %00100000
```

We start by defining the memory maps of ports A and B on the versatile interface adapter and the memory maps of the data, status, command, and control registers of the 6551 UART chip. Then we define the enable, read/write, and register select values of the VIA.

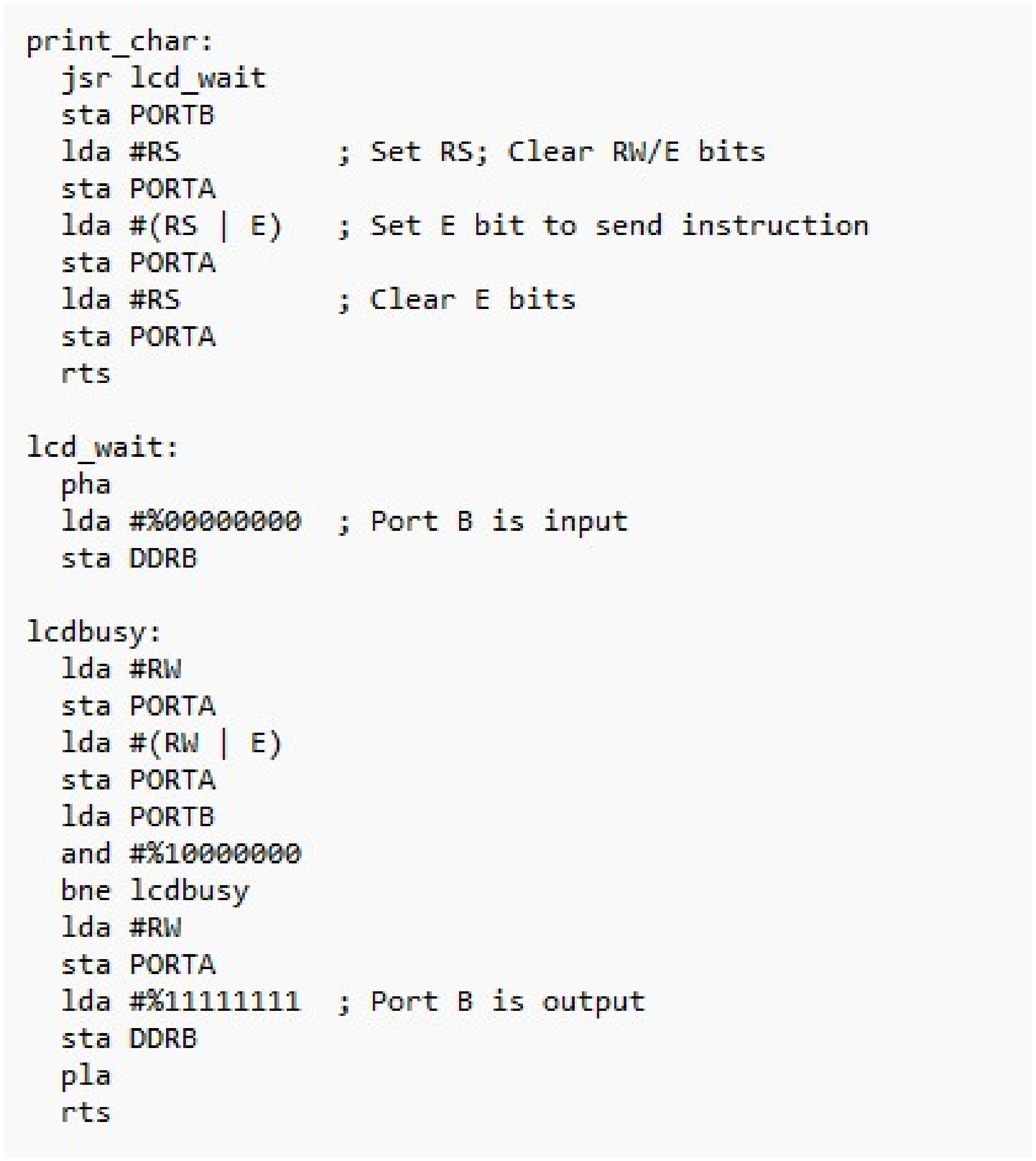
```
  .org $8000

reset:
  ldx #$ff
  txs

  lda #%00000000 ; Reset the ACIA status register
  sta ACIA_STATUS
  lda #%00001011 ; Disable transmitter and reciever interupts; data terminal ready
  sta ACIA_CMD
  lda #%00010110 ; Set baud rate to 300
  sta ACIA_CTRL

  lda #%11111111 ; Set all pins on port B to output
  sta DDRB
  lda #%11100000 ; Set top 3 pins on port A to output
  sta DDRA

  lda #%00111000 ; Set 8-bit mode; 2-line display; 5x8 font
  jsr lcd_instruction
  lda #%00001110 ; Display on; cursor on; blink off
  jsr lcd_instruction
  lda #%00000110 ; Increment and shift cursor; don't shift display
  jsr lcd_instruction
  lda #$00000001 ; Clear display
  jsr lcd_instruction
  ldx #0
```

In this block of code, we reset the status register by loading it with 0s. Then we define our initial conditions for the command and control registers of our 6551 UART chip. Loading the command register with the byte 00001011 disables transmitter and receiver interrupts and sets the data terminal to ready. It also turns off parity mode and sets receiver mode to normal. Loading the control register with be byte 00010110 sets the baud rate to 300. This must match what we set the baud rate of the Arduino to. We go on to initialize our LCD display. Here we’ve set it to 8-bit mode with a 2-line display and 5x8 pixels font. We’ve also turned off auto-scrolling.



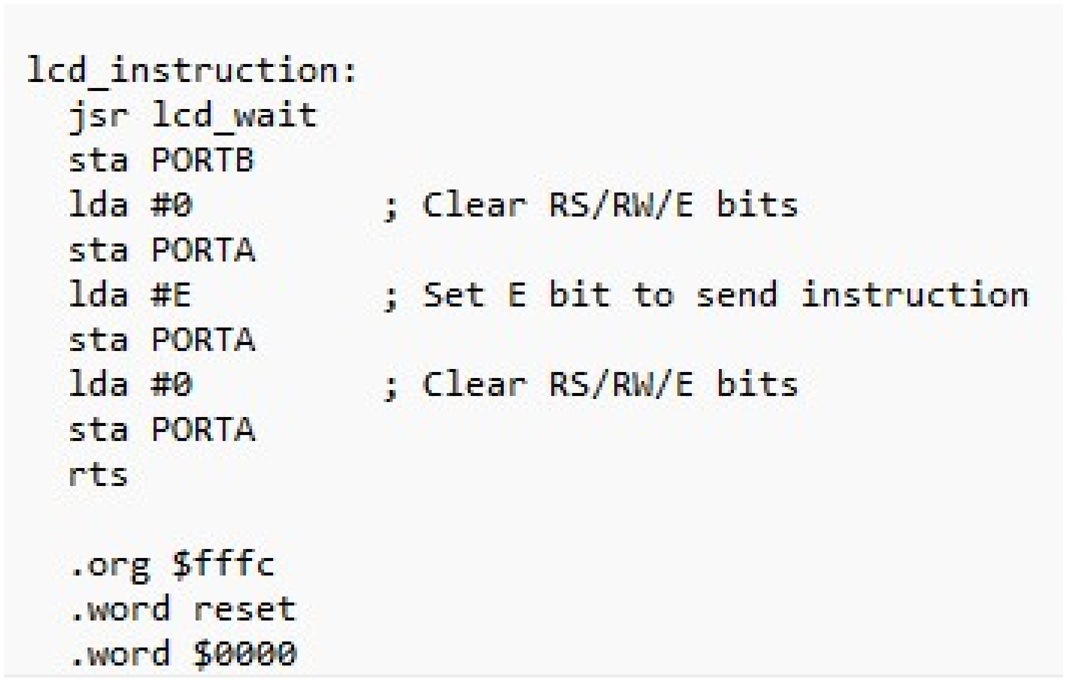
```
recv_char_acia:
  lda ACIA_STATUS
  and #%00001000 ; Check if reciever data register is full
  beq recv_char_acia
  lda ACIA_DATA
  jsr print_char
  rts
```

This block of code checks to see if the receiver data register is full. It does so by and-ing the byte in the status register with 00001000. If the two bytes are different (i.e. the receiver data register is empty), the program will loop back up to recv_char_acia. Otherwise, it’ll load the byte sitting in the data register into the accumulator and jump to the print_char subroutine.



```
print_char:
  jsr lcd_wait
  sta PORTB
  lda #RS         ; Set RS; Clear RW/E bits
  sta PORTA
  lda #(RS | E)   ; Set E bit to send instruction
  sta PORTA
  lda #RS         ; Clear E bits
  sta PORTA
  rts

lcd_wait:
  pha
  lda #%00000000  ; Port B is input
  sta DDRB

lcdbusy:
  lda #RW
  sta PORTA
  lda #(RW | E)
  sta PORTA
  lda PORTB
  and #%10000000
  bne lcdbusy
  lda #RW
  sta PORTA
  lda #%11111111  ; Port B is output
  sta DDRB
  pla
  rts
```

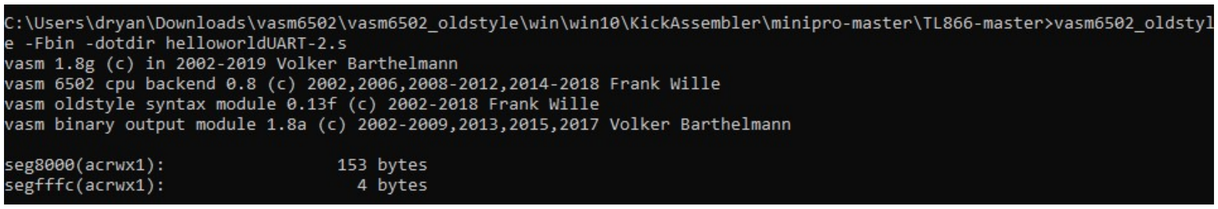
The print_char subroutine handles printing characters to the LCD screen. If the LCD is in the process of printing a character to its screen, we don't want to overwrite this. So, the first thing that happens in the print_char subroutine is it jumps to the lcd_wait subroutine, which together with the lcd_busy subroutine makes sure that the LCD is ready to print a new character to its screen.



```
lcd_instruction:
  jsr lcd_wait
  sta PORTB
  lda #0          ; Clear RS/RW/E bits
  sta PORTA
  lda #E          ; Set E bit to send instruction
  sta PORTA
  lda #0          ; Clear RS/RW/E bits
  sta PORTA
  rts

  .org $fffc
  .word reset
  .word $0000
```

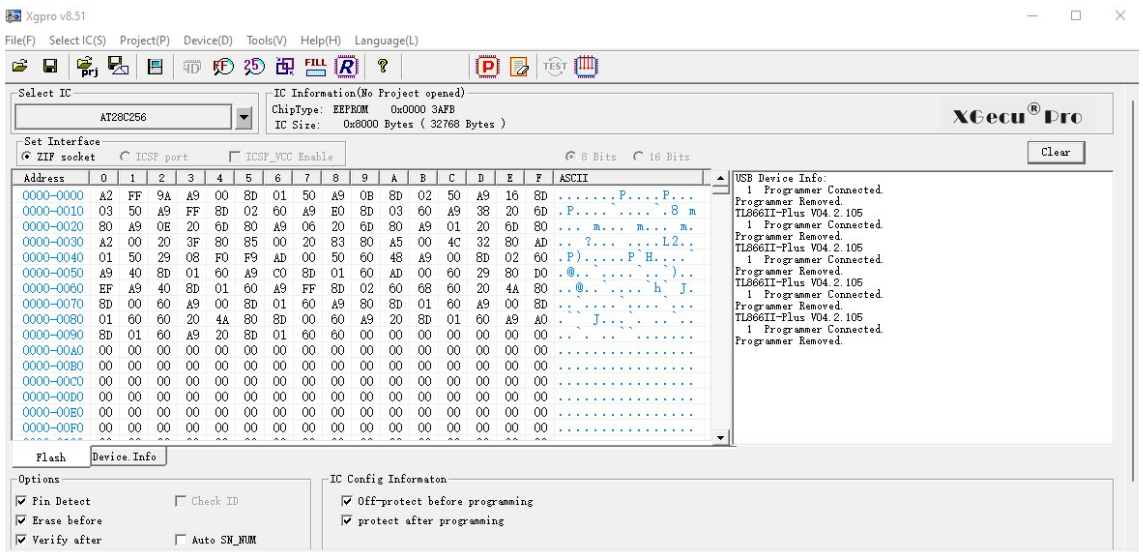
The lcd_instruction subroutine allows us to write instructions to our LCD display by loading a byte into the accumulator. We used this subroutine during the initialization stages. Much of this code has been adapted from Eater's and Billington's work with the 6502 microprocessor.

6.3 Xgpro v8.51



```
C:\Users\dryan\Downloads\vasm6502\vasm6502_oldstyle\win\win10\KickAssembler\minipro-master\TL866-master>vasm6502_oldstyl
e -Fbin -dotdir helloworldUART-2.s
vasm 1.8g (c) in 2002-2019 Volker Barthelmann
vasm 6502 cpu backend 0.8 (c) 2002,2006,2008-2012,2014-2018 Frank Wille
vasm oldstyle syntax module 0.13f (c) 2002-2018 Frank Wille
vasm binary output module 1.8a (c) 2002-2009,2013,2015,2017 Volker Barthelmann

seg8000(acrwx1):                 153 bytes
segfffc(acrwx1):                   4 bytes
```

I used vasm6502 to assemble my 6502 assembly code. Here are the instructions that I used to do so.


Xgpro v8.51
File(F) Select IC(S) Project(P) Device(D) Tools(V) Help(H) Language(L)
XGecu® Pro
Clear
Flash
Device.Info

```
Select IC: AT28C256
IC Information(No Project opened)
ChipType:  EEPROM    0x0000 3AFB
IC Size:    0x8000 Bytes ( 32768 Bytes )
Set Interface
(•) ZIF socket   ( ) ICSP port   [ ] ICSP_VCC Enable        (•) 8 Bits  ( ) 16 Bits

Address    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  ASCII
0000-0000  A2 FF 9A A9 00 8D 01 50 A9 0B 8D 02 50 A9 16 8D  .......P....P...
0000-0010  03 50 A9 FF 8D 02 60 A9 E0 8D 03 60 A9 38 20 6D  .P.... .... .8 m
0000-0020  80 A9 0E 20 6D 80 A9 06 20 6D 80 A9 01 20 6D 80  ... m... m... m.
0000-0030  A2 00 20 3F 80 85 00 20 83 80 A5 00 4C 32 80 AD  .. ?... ....L2..
0000-0040  01 50 29 08 F0 F9 AD 00 50 60 48 A9 00 8D 02 60  .P).....P H....
0000-0050  A9 40 8D 01 60 A9 C0 8D 01 60 AD 00 60 29 80 D0  .@.. .... .. )..
0000-0060  EF A9 40 8D 01 60 A9 FF 8D 02 60 68 60 20 4A 80  ..@.. .... h  J.
0000-0070  8D 00 60 A9 00 8D 01 60 A9 80 8D 01 60 A9 00 8D  .. .... .... ...
0000-0080  01 60 60 20 4A 80 8D 00 60 A9 20 8D 01 60 A9 A0  .   J... . .. ..
0000-0090  8D 01 60 A9 20 8D 01 60 60 00 00 00 00 00 00 00  .. . ..  .......
0000-00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000-00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000-00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000-00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000-00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000-00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

USB Device Info:
  1  Programmer Connected.
Programmer Removed.
TL866II-Plus V04.2.105
  1  Programmer Connected.
Programmer Removed.
TL866II-Plus V04.2.105
  1  Programmer Connected.
Programmer Removed.
TL866II-Plus V04.2.105
  1  Programmer Connected.
Programmer Removed.
TL866II-Plus V04.2.105
  1  Programmer Connected.
Programmer Removed.

Options
[✓] Pin Detect        [ ] Check ID
[✓] Erase before
[✓] Verify after      [ ] Auto SN_NUM

IC Config Informaton
[✓] Off-protect before programming
[✓] protect after programming
```

To burn programs to the EEPROM, I used TL866 II Plus EEPROM Programmer. The software for this EEPROM programmer can be downloaded at the XGecu website. This software is called Xgpro; picture here is its interface. After inserting the EEPROM into the programmer and connecting the programmer to our PC, we select the IC chip (in this case AT28C256), erase the pre-existing program off of it, open up the a.out file generated using vasm6502, and burn it to the EEPROM.

7 Operation

This circuit implements an Arduino to write messages to a 6502 computer. We can program our Arduino to write whatever message that we want. In this case, we've programmed it with the message: "Hello World!". Our Arduino writes these messages to our 6502 computer by breaking it down into bytes and transmitting them one by one to the receiver pin on the 6551 UART chip. Once the 6551 UART chip receives a byte, the third bit of the status register (which indicates whether or not the receiver data register is full) is flipped from a 0 to a 1. This byte is then loaded into the data register and sent to the versatile interface adapter, which tells the LCD to print this byte to its screen. When this byte is sent to the VIA, the receiver data register is emptied and the third bit of the status register is flipped back to a 0, thus preparing it to receive the next byte from the Arduino. This process will repeat until the Arduino has finished sending its message, at which point it will enter a do-nothing loop.

8 Applications

Despite first hitting the market in the 80s, the 6502 microprocessor chip is still in use today. Although it lacks many of the capabilities of more modern microprocessors, it still sells hundreds of thousands a year. This is, in large part, due to the fact that corporations that still use equipment manufactured in the 80s regularly buy 6502 microprocessor chips. It's not always in a corporation's best interest to upgrade a working machine, especially if that machinery costs upwards of thousands of dollars. Seeing how the average 6502 chip goes for approximately $25.00, if the only thing breaking down is the microprocessor, then it's much more cost effective to just replace the chip. So, in this way, the 6502 chip lives on.

Observing that these chips are still widely used, the utility of being able to interface with these chips is immediately apparent. Writing to our EEPROM involves prying it from the circuit board, attaching it to an EEPROM programmer, burning a program to it (which we can only do a finite number of times), and reinserting it into the breadboard. Implementing the Arduino allows us to bypass all of this and write messages to be printed directly to LCD display. If a corporation using 80s-era equipment wants to alter the functionality of said equipment, it can avoid prying apart the circuitry by interfacing with it's serial communication capabilities. This might involve connecting a more modern microprocessor like an Arduino to its UART and transmitting data using RS232.

Being able to interface with this older hardware might also be useful in adhering to OSHA regulations. If this older equipment falls out of favor with OSHA, then instead of replacing it, connecting it with modern sensors or micro-controllers might be a viable solution. Such could be achieved by interfacing the two using UART serial communication.

9 Future Works

Potential future works include adding another transmission line going from our 6502 computer to the Arduino. Right now our 6502 computer can only receive data. Adding this new transmission line would give us the capability of transmitting data the other way back to the Arduino. One issue with our current setup is the Arduino starts transmitting data right as its connected to power. So if it's not connected to our 6502 computer when we power it up, our 6502 computer is not going to receive the message when we do. If we were able to transmit data both ways, we could tell the Arduino to wait until we've reset our 6502 computer and then start transmitting data.

Another future project could be adding in hardware interrupts. We currently are not using the five buttons in the lower right-hand corner of our board. We could add connect these buttons so as to start or pause data transmissions, change the messages being sent, or modify how the messages are displayed on the LCD screen.

10 Conclusion

We've successfully interfaced with our 6502 computer to send messages to it using an Arduino. We can now print messages to our 6502 computer's LCD screen without having to remove its EEPROM chip and burn 6502 assembly code to it. The asynchronous serial communication protocols that we used in this project such as RS232 have been around for several decades and continue to be used today. Thus, being able to interface old technology and new technology is vital in the new digital age where technology is developing at an unprecedented pace.

11 References

Eater, B. Build a 6502 computer. Accessed April 30, 2023, from eater.net/6502.

Billinton, M. Adding a serial port to my 6502 computer. Accessed April 30, 2023, from mike42.me/blog/2021-07-adding-a-serial-port-to-my-6502-computer.

ACIA 6551 – Retro Computing. Accessed April 30, 2023, from retro.hansotten.nl/6502-sbc/lee-davison-web-site/acia-6551/.

65c02 Assembly – Wikibooks. Accessed 30, 2023, from en.wikibooks.org/wiki/65c02_Assembly.

6502.org: Tutorials and Aids. Accessed April 30, 2023, from 6502.org/tutorials/ 6502opcodes.html.

Mallari, J. (2021, November 16). How to Set Up UART Communication on the Arduino. Circuit Basics. Accessed April 30, 2023, from www.circuitbasics.com/how-to-set-up-uart-communication-for-arduino/.

Campbell, S. (2021, November 14). Basics of UART Communication. Circuit Basics. Accessed April 30, 2023, from www.circuitbasics.com/basics-uart-communication/.

RS232 TTL and the RS232 standard. Honeywell. Accessed 30, 2023, from honeywellaidc.force.com/supportppr/servlet/fileField?entityId=ka06O000000C7qEQAS&field=File_1__Body__s.

12 Appendix

12.1 User's Manual

1. Connect power and ground to the circuit. Be sure to use a 5V DC supply. The LCD should light up, and nothing besides solid black or white squares should appear on the screen.

2. Press the 6502 computer's reset button, which is located nearest the 6502 microprocessor chip. The white or black squares should disappear, leaving nothing but the cursor.

3. If after several seconds nothing appears on the LCD screen, then this means that the Arduino has already transmitted all of its data. Simply press the reset button on the Arduino to restart its transmission. After a couple of seconds, the message should appear on the LCD screen.

4. Pressing the 6502 computer's reset button again clears the LCD screen, and pressing the Arduino's reset button reprints the message on the LCD screen.

5. To shut off the circuit, simply remove power and ground.

12.2 Data sheets

W65C22 (W65C22N and W65C22S) Versatile Interface Adapter (VIA) Datasheet. The Western Design Center. Accessible from eater.net/datasheets/w65c22.pdf.

HD44780U (LCD-II). HITACHI. Accessible from eater.net/datasheets/HD44780.pdf.

256K (32K x 8) Paged Parallel EEPROM AT28C256. ATMEL. Accessible from eater.net/datasheets/28c256.pdf.

HM62256B Series 256k SRAM (32-kword × 8-bit). HITACHI. Accessible from eater.net/datasheets/hm62256b.pdf.

W65C02S 8–bit Microprocessor. Western Design Center. Accessible from eater.net/datasheets/w65c02s.pdf.

MM74HC00 Quad 2-Input NAND Gate. Fairchild Semiconductor. Accessible from eater.net/datasheets/74hc00.pdf.

W65C51N Asynchronous Communications Interface Adapter (ACIA). Western Design Center. Accessible from www.westerndesigncenter.com/wdc/documentation/w65c51n.pdf.

12.3 Additional Resources

vasm6502. Provided by moonbeam87. Downloadable at github.com/TJMicroelectronicsLab/6502-VASM/blob/master/vasm6502_oldstyle

Xgpro v8.51. Provided by XGecu. Downloadable at www.xgecu.com/EN/download.html.