KH-LF57K

Developer’s User Manual

“The best 1980s computer made from 1970s and 1990s technology!”
-Koppany Horvath

Rev 1.0 - 2017 / 05 / 09

Contents

Additional Resources: https://github.com/koppanyh/KH-LF57K

General Description

The KH-LF57K computer is a microcomputer designed by the KH-Labs development team with the goal of being built from the bare minimum parts required to run a Forth operating system on a Zilog Z80 processor with memory, a ROM, and serial communications capabilities. The name KH-LF57K stands for **KH-L**abs **F**orth OS **57** **K**B of RAM Microcomputer.

This computer was designed and built for the initial purpose of entering the University of La Verne's second annual Mini Maker Fair. It took a week to write and debug the operating system. The Z80 assembly language was never used by the development team before.

Future plans for the computer involve research into 1980s software development along with work on optimizing the Forth operating system. Work on upgrading the hardware and speeding up the computer is also planned. If possible, working on giving the computer the look and feel of an authentic 1980s computer, particularly with the style of CBM computers, will also be attempted.

Hardware Specs

- 8 bit Zilog Z80 processor
- 10.34 KHz clock speed
- 57 KB of RAM
- 8 KB of ROM memory
- 115200 baud serial module
- CMOS memory switching logic

Addressing/Serial

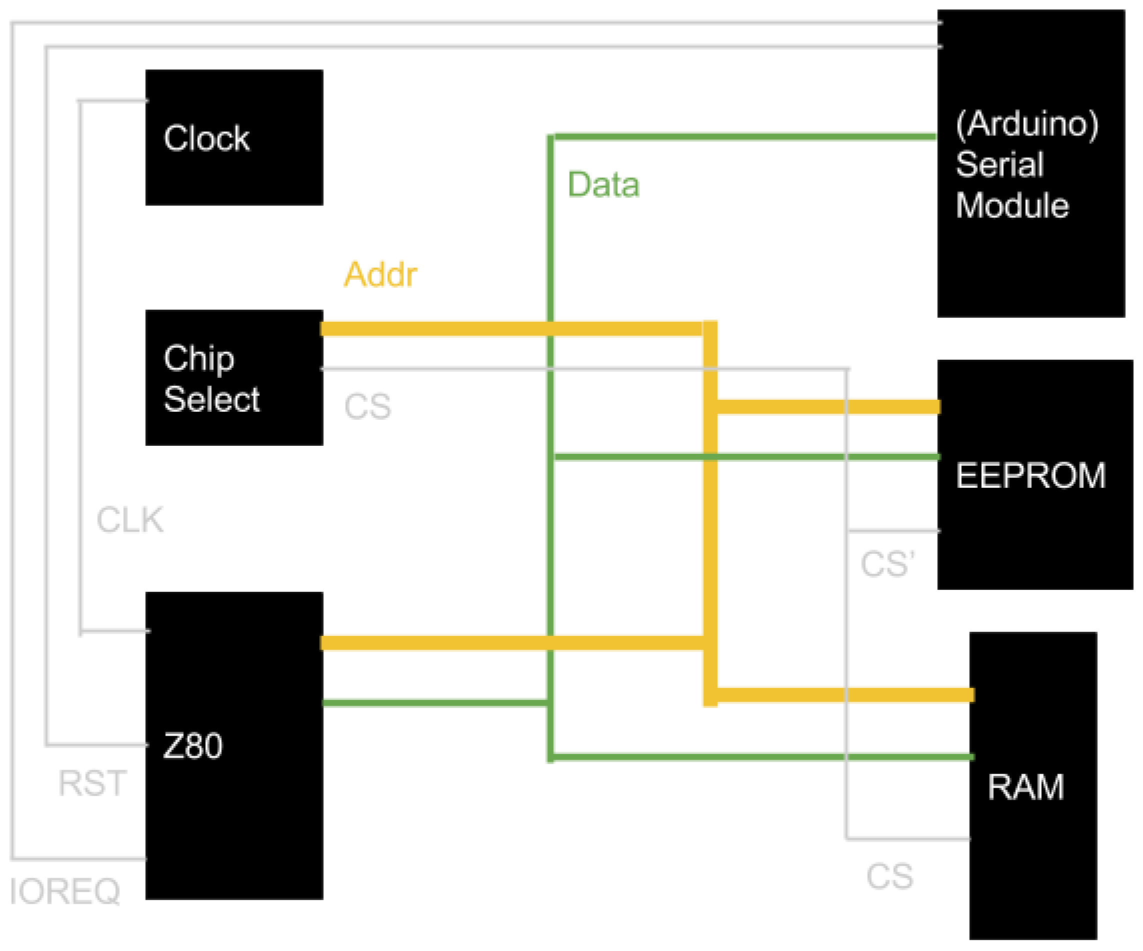
The addressing capabilities of the KH-L57K computer are split between the ROM, RAM, and serial module.

The ROM can be addressed between addresses 0x0000 and 0x1FFF.

The RAM can be addressed between 0x2000 and 0xFFFF.

The serial module can be addressed using the IO request line. The first IO request will have to be sending data to the serial module as the serial module will be in a listening mode, then the second IO request will receive data from the serial module as the serial module will be waiting to transmit. The serial module then switches to receiving mode and waits for another byte from the processor. The serial module is capable of asynchronously transmitting and receiving between the processor and a serial terminal at 115200 baud. The serial module can buffer at most 64 bytes of data from a serial terminal waiting to be read by the processor. The serial module does not buffer data from the processor but instantly transmits each byte to the serial terminal.

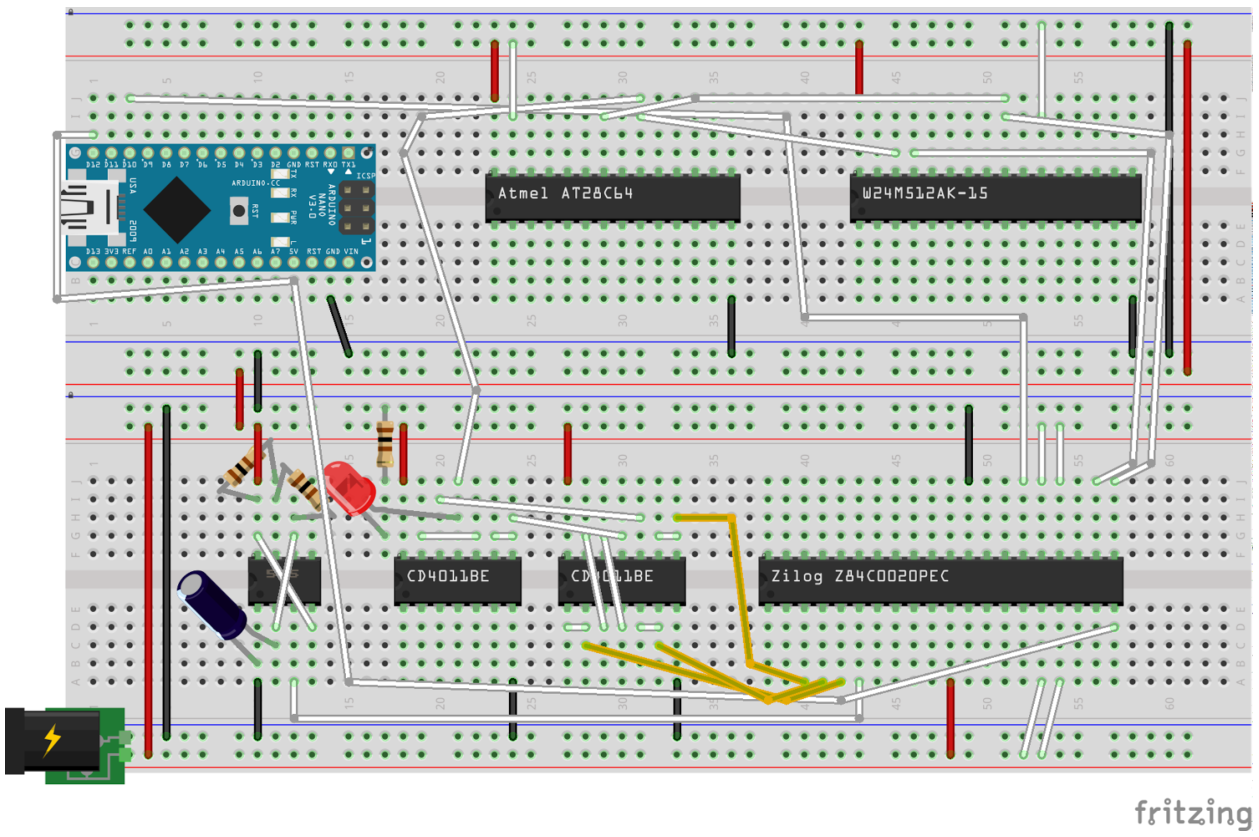
Block Diagram



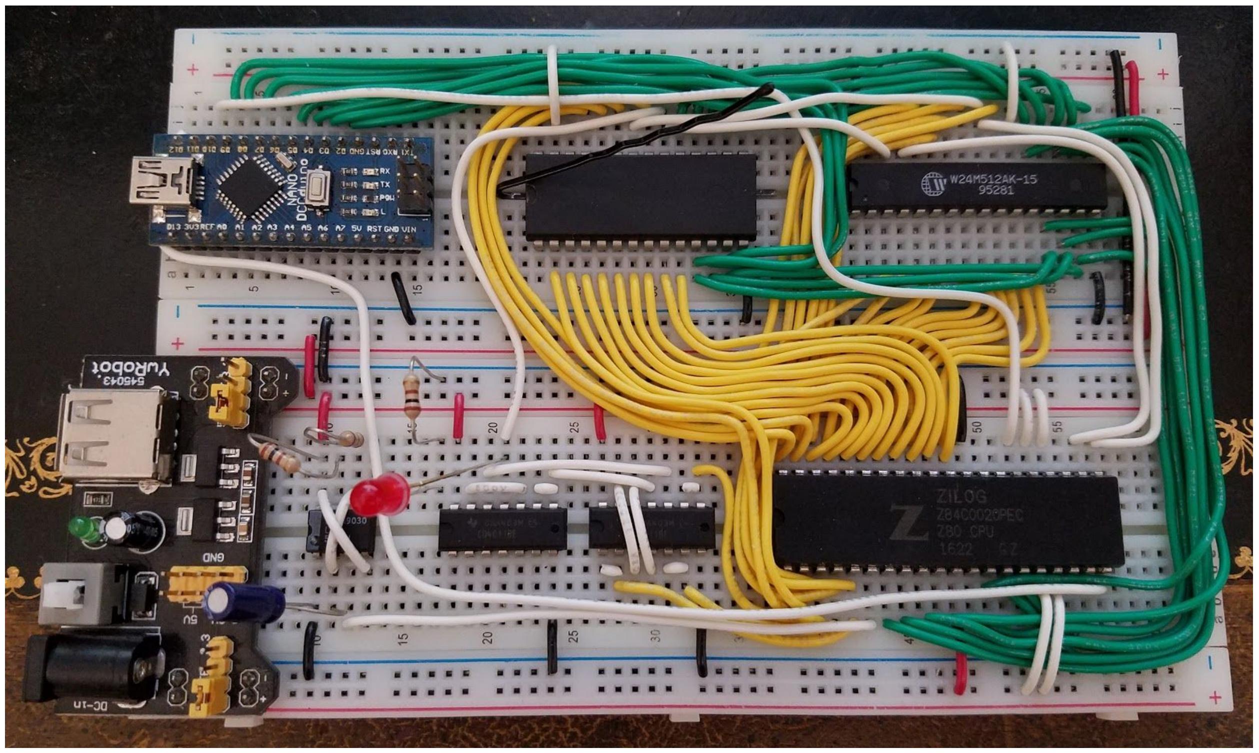
Parts List

- 1x MB102 Breadboard Power Supply
- 1x Zilog Z84C0020PEC CMOS Z80 CPU
- 1x AT28C64-15PC 8KX8 EEPROM
- 1x W24M512AK-15 64KX8 SRAM
- 1x Arduino Nano Microcontroller
- 1x 0.47 µf Electrolytic Capacitor
- 2x CD4011 Quad NAND Gates
- 1x Red Difuse LED
- 1x LM555 Timer IC
- 3x 100 Ω Resistors

Schematics



Atmel AT28C64
W24M512AK-15
CD4011BE
CD4011BE
Zilog Z84C0020PEC
fritzing



The first figure is a rough representation of the layout of the computer without address or data lines. To complete the schematic, the data lines simply need to be connected to between the corresponding data pins on the processor, ROM, RAM, and serial module. The serial module uses pins D2-D9 as data pins 0-7 correspondingly. The address lines need to be connected between the address pins on the processor, ROM, RAM, just like the data lines. The serial module does not require address lines to operate.

The second figure shows a built KH-LF57K computer. The figure demonstrates the connections of the address (yellow) and data (green) lines between the processor (bottom right), RAM (top right), ROM (top middle), and serial module (top left).

In the first figure, there are 3 address lines going from the processor to the CMOS memory switching logic chips (bottom middle). These are 2 2-input quad NAND chips wired to work as a 3-input OR gate. Addressing the RAM will set the top 3 address bits high and trigger a high response on the OR gate which will select the RAM chip for reading and writing. Addressing the ROM under that will deactivate the OR gate and select the ROM chip.

A 555 timer chip is used to provide the clock signal at about 10.34 KHz. The 555 timer is configured as an astable oscillator. Unfortunately this particular clock circuit is not the most accurate and is subject to problems such as thermal drift and slightly different frequencies per unit due to the precisions and tolerances of the components being slightly different. The processor can run at 20 MHz but no more than 1 MHz is recommended as it may be too fast to interface with the serial module.

Forth OS

The Forth language was chosen for the operating system of the KH-LF57K computer because of its ease of implementation and the small resources that are required to run it. The Forth language can generally fit in less space than BASIC, is natively extensible through the use of the built-in compilation capabilities, and has been around since 1970.

Forth is a stack-based language that uses reverse Polish notation (or post-order notation) and is also its own development environment. Words, functions in Forth, are created using the concept of concatenative programming. A more detailed explanation of the Forth language can be found on Wikipedia.com at the following address:
https://en.wikipedia.org/wiki/Forth_(programming_language)

The Forth version in use at the time of this revision is 0.9.4; keep in mind that this is a custom implementation of Forth and the version number is not related to any other Forth implementations or standards.

An example of adding 5 and 4 and printing it out in Forth can be seen below:

```
5 4 + .
```

This code pushes 5 and 4 onto the top of the stack. Then it adds the top 2 items of the stack together with the plus operator. Then it prints out the result at the top of the stack with the period operator.

An example of defining a Forth word that prints the double of a number at the top of the stack can be seen below:

```
: double 2 * . ;
```

This code starts the compilation sequence with the colon operator and creates the header for the word `double`. It then defines the word based on the operators following the name. When called, this code will push 2 to the top of the stack. It then will multiply the 2 with the second number on the stack using the multiplication operator and will leave the result at the top of the stack. Then it will use the period operator to print out the result at the top of the stack. The semicolon operator ends the compilation sequence and the word `double` can be seen as the first word when the `words` word is called.

To use the new `double` word, simply enter in a number and call `double` as seem below:

```
7 double
```

This code will push 7 to the top of the stack and then call the word `double` to multiply the number by 2 and print the result.

Here is a table of all the Forth words that have been implemented in version 0.9.4 of the Forth OS. For the flags, M means it is machine code, I means it is an immediate word, C means it it compiled. Lack of M means it is written in Forth, lack of I means it compiles normally, lack of C means that it can be used out of compilation.

Name	Flags	Stack Action	Description
LIT	MC	(-- n1)	get next number from program and push to stack
,	M	(n1 --)	writes n1 into the memory at WEND and updates WEND position
+	M	(n1 n2 -- n3)	add n2 to n1
-	M	(n1 n2 -- n3)	subtract n2 from n1
*	M	(n1 n2 -- n3)	multiply n2 to n1
/mod	M	(n1 n2 -- n3 n4)	divide and mod n2 from n1
emit	M	(n1 --)	output the character of the ascii value of n1
,	M	(n1 --)	display n1 followed by a space
CREATE	M	(n1 --)	create the header for a word/constant/variable from string address n1
c!	M	(n1 n2 --)	save the low 8 bits of n1 to the address n2
c@	M	(n1 n2 --)	get the byte at the address n1
!	M	(n1 n2 --)	save n1 to the address n2

@	M	(n1 -- n2)	get the value at the address n1
STATE		(-- n1)	leave the address of the state variable in the stack, 0 is interpret, -1 is compile
LATEST		(-- n1)	leave the address of the latest word variable in the stack
WEND		(-- n1)	leave the address of the end of words variable in the stack
BKEY	MC	(-- n1)	gets next char from input buffer and leaves in stack
BWORD	MC	(-- n1)	gets next word in input buffer and leaves address in stack
:	I	(--)	start compiling sequence, create and state to -1
;	IC	(--)	add 0xffff to the end of program and make state 0
key	M	(-- n1)	get the ascii value of the first key pressed
word	M	(-- n1)	gets a string from input and leaves the address of the string
number	M	(-- n1)	gets a number from input
words	M	(--)	display all the words in the forth dictionary
swap	M	(n1 n2 -- n2 n1)	switch the top 2 elements on the stack
dup	M	(n1 -- n1 n1)	duplicates top of stack
drop	M	(n1 --)	pop n1 from stack
=	M	(n1 n2 -- n3)	true if n1 equals n2, else false
<>	M	(n1 n2 -- n3)	true if n1 not equals n2, else false
<	M	(n1 n2 -- n3)	true if n1 less than n2, else false
>	M	(n1 n2 -- n3)	true if n1 greater than n2, else false
<=	M	(n1 n2 -- n3)	true if n1 less than equal n2, else false
>=	M	(n1 n2 -- n3)	true if n1 greater than equal n2, else false
and	M	(n1 n2 -- n3)	logic and n1 & n2
or	M	(n1 n2 -- n3)	logic or n1 & n2

xor	M	(n1 n2 -- n3)	logic xor n1 & n2orr
.s	M	(--)	display all the numbers in the stack without popping the stack
reboot	M	(--)	restarts the computer
>r	M	(n1 --)	push n1 to the return stack
r>	M	(-- n1)	pop n1 from the return stack
r@	M	(-- n1)	copy the top value from the return stack
>l	M	(n1 --)	push n1 to the loop stack
l>	M	(-- n1)	pop n1 from the loop stack
type	M	(n1 --)	string is displayed starting from the address n1
FIND	M	(-- n1)	find the address n1 of a word with name in wordbuff
WBUFF		(-- n1)	leaves the address of the start of the word buffer in the stack
EXEC	M	(n1 --)	executes the word with address n1
BRANCH	MC	(--)	replace the top of return stack with next number in program
0BRANCH	MC	(--)	replace the top of return stack with next number in program if top of stack is 0
VERSION		(-- n1)	leave the forth version * 100 on the stack
constant	I	(n1 --)	create a word that leaves the value n1 in the stack
variable	I	(--)	create a word that leaves the address for a variable in the stack

Useful Forth Words

As can be seen from the table above, there are many Forth words that have been defined and added into version 0.9.4 of the operating system, but not all of the most common words have been added. This was due to time constraints and accepting the trade-off of not having the words but having the capability to produce them on the spot.

The following code is the human definition (prefixed by a comma) of the words followed by their Forth definition.

```
; /        ( n1 n2 -- n3 ) divide n1 by n2, leave result n3
: / /mod drop ;

; mod      ( n1 n2 -- n3 ) get n1 modulus n2, leave result n3
: mod /mod swap drop ;

; negate   ( n1 -- n2 ) leaves 2's complement of n1 in stack
: negate 0 swap - ;

; over     ( n1 n2 -- n1 n2 n1 ) duplicates second number in stack
: over >l dup l> swap ;

; rot      ( n1 n2 n3 -- n2 n3 n1 ) rotates third number to top
: rot >l swap l> swap ;

; -rot     ( n1 n2 n3 -- n3 n1 n2 ) rotates top to third place
: -rot swap >l swap l> ;

; cells    ( n1 -- n2 ) multiplies n1 with size of number cell (2)
: cells 2 * ;

; cr       ( -- ) prints a carriage return and newline
: cr 13 emit 10 emit ;

; invert   ( n1 -- n2 ) logic not n1
: invert -1 xor ;

; allot    ( n1 -- ) increment program end counter by n1
: allot WEND @ + WEND ! ;

; IMMEDIATE     ( -- ) activate immediate flag on latest word
: IMMEDIATE 64 LATEST @ c@ or LATEST @ c! ; IMMEDIATE

; COMPILE  ( -- ) activate compile flag on latest word
: COMPILE IMMEDIATE 32 LATEST @ c@ or LATEST @ c! ;

; INSADDR  ( -- n1 ) insert address of following word in compiling
word
: INSADDR IMMEDIATE BWORD drop FIND , ;
```

```
; begin    ( -- ) specified return address for repeat/until
: begin IMMEDIATE WEND @ >l ;

; until    ( n1 -- ) jump to begin statement in begin-until loop if
n1 false
: until IMMEDIATE INSADDR LIT INSADDR 0BRANCH , l> , ;
; (        ( -- ) start of comment, ends at first ) character
: ( IMMEDIATE begin BKEY 41 = until BKEY drop ;
```

Flashing the ROM

The ROM chip can be flashed from an EEPROM programmer circuit built from an Arduino Mega microcontroller. The firmware for the EEPROM programmer can be found in the **Code Listings** section.

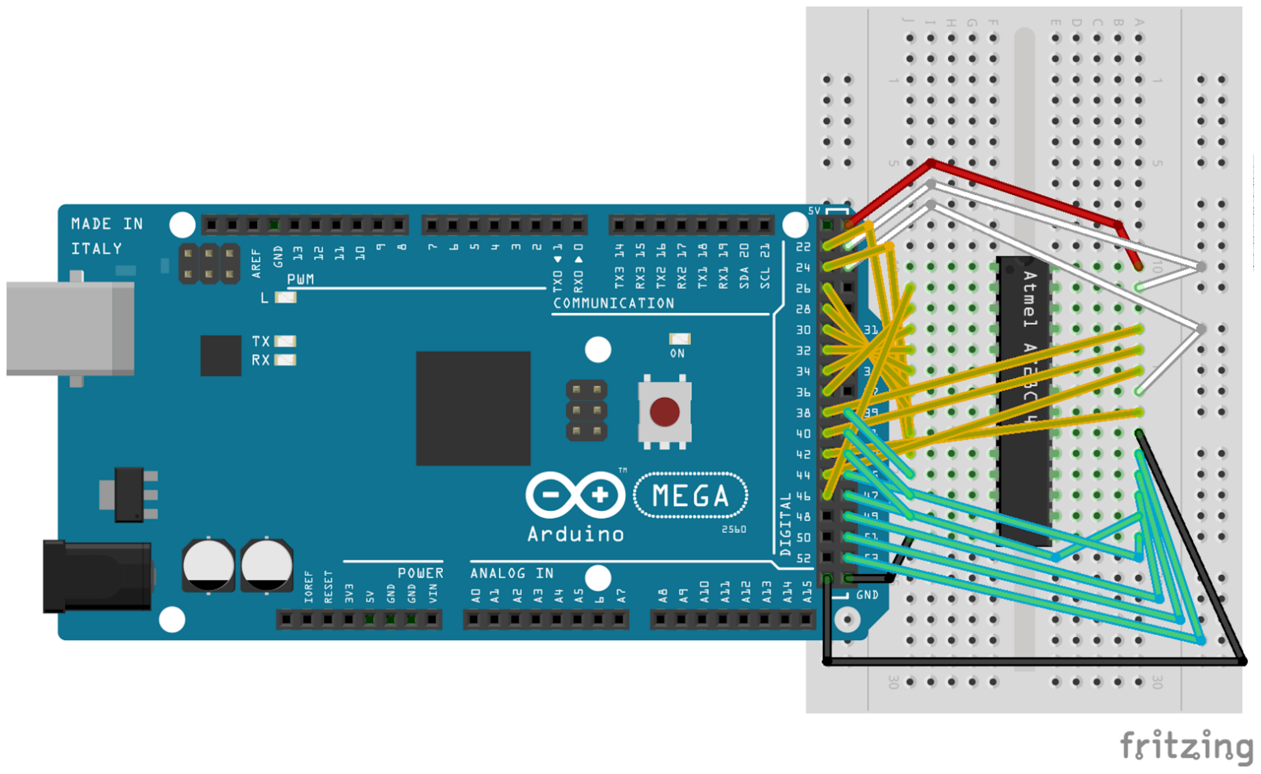
The Arduino will have to be wired up in a similar fashion to the figure below. The even numbered pins from 22 to 46 are connected to the address lines 0 to 12 respectively. The odd numbered pins from 39 to 53 are connected to the data lines 0 to 7 respectively. The Chip Enable pin of the EEPROM is connected to ground. The Write Enable pin is connected to pin 23. The Output Enable pin is connected to pin 25 of the Arduino. Voltage lines can be hooked up to the 5V and GND pins of the Arduino.

The EEPROM programmer firmware for the Arduino will allow the EEPROM chip to be programmed over a serial connection, thus avoiding having to reprogram the Arduino with a new ROM image every time the ROM needs to be updated.

The recommended software to handle the serial upload is CuteCom, website: http://cutecom.sourceforge.net/. The serial speed setting should be set to 115200 baud. The device or port is dependent on the system the Arduino is connected to. The data bits and stop bits should be set to 8 and 1 respectively. The char delay should be set to 10 ms to ensure that the Arduino has time to write a byte before receiving another one. The ROM image can be send over the Send File option. The output screen will show messaged from the Arduino stating that it is connected, showing the size of the program, and displaying the contents of the program along with a “Done” message when it has written the specified amount of bytes.

The ROM file itself has 2 bytes in the beginning as a header to hold the size of the program. The ROM will be written to the EEPROM starting from address 0 of the chip until the address that is minus one of the header size.

The ROM source code is found in the **Code Listings** section under the “Z80 assembly for the Forth OS” section. The source is written to be compiled by version 1.8 of the z80asm assembler found at this website: http://www.nongnu.org/z80asm/. When compiled, it will automatically add the 2 bytes size header with the size specified by the end of the “endOfProg” header near the bottom.



Code Listings

Due to the length of the programs, the code will not be listed in this manual directly. The listings for the programs can be found at the following links.

The Z80 assembly for the Forth OS:
https://github.com/koppanyh/KH-LF57K/blob/master/forth/z80os.asm

The C++ prototype program that the Forth OS was ported from:
https://github.com/koppanyh/KH-LF57K/blob/master/forth/forth.cpp

The firmware for the Arduino Nano based serial/control module:
https://github.com/koppanyh/KH-LF57K/blob/master/arduino/z80controller/z80controller.ino

The firmware for the Arduino Mega 2560 based EEPROM programmer:
https://github.com/koppanyh/KH-LF57K/blob/master/arduino/eepromprog/eepromprog.ino