libcpm

David Latham

September, 2025

Contents

1 Introduction **2**

1.1 About the name 2

1.2 What Is libcpm ? 2

1.3 Usage 2

2 Headers **4**

2.1 C Runtime 4

2.2 tms99xx 4

2.3 stdlib 4

2.4 stdio 4

2.5 joystick 5

2.6 cp/m 5

3 Graphics Programming **6**

4 Examples **6**

Appendices **7**

Appendix A Build **7**

Appendix B Skeleton Project **7**

1 Introduction

This guide is intended provide some commentary that can be read along with the source code to showcase the various components of the library and how they can be used in your own applications.

You should read the comments in the header files. They are updated during development and will be the most accurate.

1.1 About the name

Naming this is hard. I know this is a stale meme, but it's true. I could not come up with a meaningful name for this library that would indicate that it's for the Z80-Retro! computer *AND* it's for CP/M.

1.2 What Is libcpm ?

libcpm is a C library that you can link to in your applications for use with the FUZIX-Compiler-Kit.[1] The library is specifically targeted at the Z80-Retro![2] Single Board Computer by John Winans and the resident CP/M 2.2 operating sytem.

Some standard libraries are implemented where they can be easily backed by the CP/M 2.2 BDOS function calls. Additionally the library includes functions for working with the TMS9918 and Atari style joystick ports on the VDP daughter board designed for the Z80-Retro!.

For instructions on how to install the compiler and this library, see the `./BUILD.md` documentation.

1.3 Usage

You can use the by including the headers you need, calling the functions in your code and finally compiling and linking. See: "Listing: 1.3 - Example Makefile" on page 3

The process is something like this:

- COMPILE fcc -O2 -mz80 -Iinclude -I /opt/fcc/lib/z80/include -c -o main.o main.c
- LINK ldz80 -b C0x100 -o main.bin crt0.o main.o libcpm.a
- TRUNCATE dd if=main.bin of=main.com skip=1 bs=256

The linker documentation is very minimal because it's a very minimal linker. The `‘-b‘` switch tells the linker to output a binary file without relocatable code. The -C0x100 tells the linker to begin the `‘code‘` segment at 0x100 which is the beginning of the TPA for CP/M.

Because the linker always starts filling code from 0x0000 we need to remove the first 256 bytes using the `‘dd‘` command.

[1] https://codeberg.org/etchedpixels/fuzix-compiler-kit.git
[2] https://github.com/z80-retro

```
TOP=.
CC=/opt/fcc/bin/fcc
AS=/opt/fcc/bin/asz80
LD=/opt/fcc/bin/ldz80

CFLAGS=-O2 -mz80 -I $(TOP)/../include -I /opt/fcc/lib/z80/include
LDFLAGS=-b -C0x100
LIBS=\
     $(TOP)/../libcpm.a \
     /opt/fcc/lib/z80/libz80.a \
     /opt/fcc/lib/z80/libc.a

CRT=$(TOP)/../crt0.o

all: clean malloc.com fileio.com testtms.com copy

malloc.bin: malloc.o
  $(LD) $(LDFLAGS) -o $@ $(CRT) $^ $(LIBS)

malloc.com: malloc.bin
  dd if=$^ of=$@ skip=1 bs=256

fileio.bin: fileio.o
  $(LD) $(LDFLAGS) -o $@ $(CRT) $^ $(LIBS)

fileio.com: fileio.bin
  dd if=$^ of=$@ skip=1 bs=256

testtms.bin: testtms.o
  $(LD) $(LDFLAGS) -o $@ $(CRT) $^ $(LIBS)

testtms.com: testtms.bin
  dd if=$^ of=$@ skip=1 bs=256


clean:
  rm -fv malloc.bin malloc.com fileio.bin fileio.com
  find . -name "*.o" -exec rm -fv {} \;
```

Listing 1: Example Makefile

This example does not show the actual FCC commands explicitly. Make is automating that step for us.

2 Headers

The headers are all located in the projects ”include” directory. You just need to `#include` the ones you need and make sure to link to the `libcpm.a` library.

As the code is split out into multiple translation units, your resulting binary should include almost no wasted code.

2.1 C Runtime

There is a provided `crt0.o` object file which should be included in the linking stage. The C runtime performs the following actions:

prep stack Preserve the CP/M stack pointer and set up a new stack pointer at the top of the TPA.

init_sys Calls the `_init_sys` routine to initialise the ”sys_open_files” array.

execute Call the `main()` function

return Return to CP/M (restoring the CP/M stack and freeing the tms_buffer along the way)

2.2 tms99xx

See Graphics Programming on page 6 for details.

2.3 stdlib

There are two parts to the malloc / free implementation in this library. The first is the implementation of the ”sbrk()” system call and the second is the malloc and free functions themselves.

The sbrk() function was copied and adapted from the HiTech C compiler project.

The malloc and free functions are transcribed directly from ”The C Programming Language - Second Edition” By Biran W. Kernignhan and Dennis M. Ritchie.

There are is no defragmentation or garbage collection. Malloc doesn’t usually make all that much sense in embedded environments.

- ABS
- EXIT
- FREE From The C Programming Language Book
- ITOA
- MALLOC From The C Programming Language Book
- PUTS Print a zero terminated string constant to the screen. No formatting.

2.4 stdio

- PRINTF Simple implimentation of `printf` with support for the %c, %s, %d and %x format specifiers only.

2.5 joystick

- JOY(UINT8_T IDX) read the joystick given by idx. 0 = J3, 1 = J4

Also provided in the `joy.h` header are the button and axis mappings. The bits are aligned with how they are laid out on the 2063 VDP PCB.

```
    #define JOY_MAP_LEFT      0x04 //0b00000100
    #define JOY_MAP_RIGHT     0x20 //0b00100000
    #define JOY_MAP_UP        0x80 //0b10000000
    #define JOY_MAP_DOWN      0x40 //0b01000000
    #define JOY_MAP_BUTTON    0x01 //0b00000001

```

Listing 2: Joystick Button Mapping

2.6 cp/m

The CP/M header provides C wrappers for almost all the CP/M BDOS function calls.

For example, if you want to check for keyboard input without blocking, you can call the `cpm_rawio()` function which returns the ascii char or 0. Unlike `cpm_conin()`, this function does not emit the typed character to the terminal.

The standard argc, **argv approach for handling command line arguments is currently managed by a seprate function called `cpm_parse_args(char **argv, uint8_t count)` This will populate an array of pointers with pointers to space sperated tokens from the CPM commandline buffer (0x80). In fact the function makes a copy of this buffer first in case CPM needs it for something later on.

You can find an example of how to use the fileio functions in the ”test” folder. The ”fcntl.h” header file contains lots of additional information.

3 Graphics Programming

Figure 1 shows the game loop state machine. The game state is initialised before entering the loop. The loop itself, consists of reading user input, updating the game state and frame-buffer, waiting for vsync, rendering the frame-buffer and looping back to user input.

The VSYNC signal from the VDP provides a stable 60 *hertz* timer which is used to normalize game speed on different CPU clock frequencies. The standard gameloop waits for the VSYNC signal before rendering to the display to leverage the fast write cycle times during the vertical blanking interval.

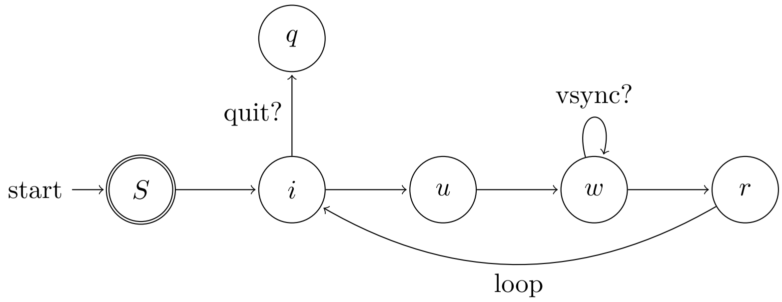

Figure 1: Game Loop

There are a range of TMS99xx functions in the libcpm to initialise the VDP in the various display modes and to manage the frame-buffer.

They can be grouped into the following categories:

- Initialization Init the VDP in a display mode, enable interrupts and sprite sizes
- Loading data Load patterns, names and colors
- Plotting to the frame buffer Plot tiles or pixels in the XY coordinate space into the frame buffer memory.
- Sprites Update sprite attributes to move them around or hide them.
- Flushing the frame buffer to the video memory Stream the framebuffer into VDP memory and flush the sprites array into the Sprite Attribute Table in VDP memory.

4 Examples

There are a few exsamples in the `Examples` folder in the Git repository. Each one is designed to demonstrate specific features of this library.

Appendices

Appendix A Build

These are the instructions for getting setup with the compiler-kit so that you can build this library and use it in your code.

Appendix B Skeleton Project

Some text