F256Dev: Compiler, Tools, and C Library for the Foenix F256 Family of Computers

Contents

1. Introduction . . . 3
2. Audience . . . 3
3. Requirements . . . 4
4. Installation . . . 4
5. Configuration . . . 4
6. Upgrading . . . 4
7. Assumptions . . . 5
8. Amalgamated or Not? . . . 5
9. Usage . . . 5
    9.1. Setting Up Your Project . . . 5
    9.2. Creating Your Main Source File . . . 6
    9.3. Building Your Project . . . 6
    9.4. Running Your Project . . . 6
    9.5. Debugging . . . 7
10. Definitions . . . 7
11. API Overview . . . 7
12. API Details . . . 8
    12.1. Constants . . . 8
    12.2. Types . . . 8
    12.3. Helpers . . . 8
        12.3.1. PEEK . . . 8
        12.3.2. POKE . . . 8
        12.3.3. PEEKW . . . 9
        12.3.4. POKEW . . . 9
        12.3.5. POKEA . . . 9
        12.3.6. PEEKD . . . 9
        12.3.7. POKED . . . 9
        12.3.8. FAR_PEEK . . . 9
        12.3.9. FAR_POKE . . . 9
        12.3.10. FAR_PEEKW . . . 10
        12.3.11. FAR_POKEW . . . 10
        12.3.12. LOW_BYTE . . . 10

12.3.13. HIGH_BYTE . . . 10
12.3.14. SWAP_NIBBLES . . . 10
12.3.15. SWAP_UINT16 . . . 10
12.3.16. CHECK_BIT . . . 10
12.3.17. TOGGLE_BIT . . . 11
12.3.18. CLEAR_BIT . . . 11
12.3.19. SET_BIT . . . 11
12.4. Kernel . . . 11
12.4.1. kernelCall . . . 12
12.4.2. kernelEvent . . . 12
12.4.3. kernelGetPending . . . 12
12.4.4. kernelNextEvent . . . 12
12.5. DMA . . . 12
12.5.1. dma2dFill . . . 13
12.5.2. dmaFill . . . 13
12.6. Math Co-Processor . . . 13
12.6.1. mathSignedDivision . . . 13
12.6.2. mathSignedDivisionRemainder . . . 13
12.6.3. mathSignedMultiply . . . 13
12.6.4. mathUnsignedAddition . . . 13
12.6.5. mathUnsignedDivision . . . 14
12.6.6. mathUnsignedDivisionRemainder . . . 14
12.6.7. mathUnsignedMultiply . . . 14
12.7. Random Numbers . . . 14
12.7.1. randomRead . . . 14
12.7.2. randomSeed . . . 14
12.8. Text . . . 15
12.9. Graphics . . . 16
12.9.1. graphicsDefineColor . . . 17
12.9.2. graphicsSetLayerBitmap . . . 17
12.9.3. graphicsSetLayerTile . . . 17
12.9.4. graphicsWaitVerticalBlank . . . 17
12.10. Bitmaps . . . 17
12.10.1. bitmapClear . . . 18
12.10.2. bitmapGetResolution . . . 18
12.10.3. bitmapLine . . . 18
12.10.4. bitmapPutPixel . . . 18
12.10.5. bitmapSetActive . . . 18
12.10.6. bitmapSetAddress . . . 19
12.10.7. bitmapSetCLUT . . . 19
12.10.8. bitmapSetColor . . . 19

12.10.9. bitmapSetVisible . . . 19
12.11. Sprites . . . 19
12.11.1. spriteDefine . . . 20
12.11.2. spriteSetPosition . . . 20
12.11.3. spriteSetVisible . . . 20
12.12. Tiles . . . 20
12.12.1. tileDefineTileMap . . . 20
12.12.2. tileDefineTileSet . . . 21
12.12.3. tileSetScroll . . . 21
12.12.4. tileSetVisible . . . 21
12.13. File I/O . . . 21
12.14. Platform . . . 22
12.14.1. __putchar . . . 22
12.14.2. getchar . . . 22
13. Using Code Overlays . . . 23
14. Embedding Binary Data . . . 24
14.1. EMBED . . . 24
15. Optional Options . . . 25
16. Support . . . 26

1. Introduction

F256Dev is a C development system for the Foenix F256 family of "modern retro" computers. It consists of the ***LLVM-MOS*** compiler toolchain, the ***F256lib*** programming library, several helpful tools, build and run scripts, and more.

F256lib is a C library targeting the F256 family. It provides easy access to the platform's various features (graphics, text, I/O, etc.) from within a C program.

In addition, ***F256lib*** also unleashes the power of the F256 by automatically handling most of the sometimes tricky memory management and paging required to truly harness the power of the system.

TIP Although fairly complete, not every feature of the Foenix F256 computers are supported yet. Not finding a feature you need? Let us know!

2. Audience

This document is not an introduction to programming. It assumes you are comfortable with your host OS and working from the command line.

Example pathnames in this document are usually in Windows format. Linux and MacOS folks, you can cope.

3. Requirements

- Foenix F256jr, F256K, (or emulator) with a 65c02 or 65816 CPU.
- Host PC running Linux (x64), Windows (x64), or MacOS (Apple Silicon).

On Windows, you will also need:

- WinRAR or 7-Zip installed.
- Python 3.x installed and in your PATH.

For MacOS and Linux:

- Ensure you have Python 3.x installed.

NOTE	Although the 65816 is supported, it is currently treated as a 65c02.

4. Installation

Download the appropriate installation script:

Platform	Script
64 bit Intel Windows	f256-install.bat
64 bit Intel Linux	f256-install.sh
64 bit ARM MacOS	f256-install.sh

Create a new, empty directory, with no spaces in the name (or in the names of the parent directories) and place this file into it. Recommended locations:

- On Windows: `C:\F256`
- On Linux and MacOS: `~/f256`

From that folder, execute the script. It will download additional dependencies and install them. Aside from some Python modules needed by FoenixManager, everything will stay inside this new folder. Uninstalling is just a matter of deleting it.

5. Configuration

You will need to modify `foenixmgr.ini` in the main installation folder. The only thing that should need updated is the COM port used to communicate with your F256. See: https://github.com/pweingar/FoenixMgr

6. Upgrading

Just re-run your `f256-install` script!

WARNING	This will **delete** the `f256dev` folder! Do **NOT** put your own data in this folder!

7. Assumptions

This package was designed to be easy to use for new programmers. As such, some liberties were taken that aren't entirely common practice. There are no project files, makefiles, or other build configuration systems. The included `f256build` and `f256run` scripts assume you have arranged your code in directories like the example programs. If you are more advanced than this, feel free to go crazy.

While we've tried to simplify things, do not be afraid to dig into the source code for ***F256lib*** and the included tools!

> Use the Source, Luke!
>
> — Some Code Monkey Jedi

This manual also assumes you have familiariazed yourself with the hardware documentation for the Foenix F256. Without knowing the capabilities of the machine, it's going to be hard to program in any lanugage with any toolkit!

8. Amalgamated or Not?

F256lib is distributed in two formats. *Amalgamated* (everything in a single header file) or as individual source files for each section of the API.

We **highly** recommend using the amalgamated build unless you are interested in working on the library itself. The included build tools do not support using the non-amalgamated version out-of-the-box.

9. Usage

Using ***F256lib*** is pretty painless for a low-level C library. Simply include the header in each of your source files.

9.1. Setting Up Your Project

The main requirement is that each program have its own parent directory named for that program and inside it, a "src" directory where the actual code lives. Example:

```
C:\F256\CODE\MYPROGRAM\SRC
```

This would be a project named `MYPROGRAM` located in a `CODE` folder under the folder you installed the toolkit in. (You do not have to locate your projects under the toolkit folder.)

9.2. Creating Your Main Source File

In the file that contains your `main()` function, start with the following:

```
#define F256LIB_IMPLEMENTATION
#include "f256lib.h"

int main(int argc, char *argv[]) {
   printf("Hello from F256 Land!");
   while(1)
      ;
   return 0;  // Reaching this will reset the F256.
}
```

9.3. Building Your Project

To build this, you would run `f256build` from the `C:\F256` folder as follows:

```
f256build.bat code\myprogram
```

UNIX folks, flip your slashes!

```
./f256build.sh code/myprogram
```

NOTE When using `f256build` to build your projects, the include path will be set automatically.

After a successful build, an overview of memory usage will be displayed. Double-check this to ensure it appears "sane" for your program! If you run out of space in either near memory or an overlay bank, you'll see errors here.

9.4. Running Your Project

If you properly configured your `foenixmgr.ini` and have connected your F256 to your host PC, simply use `f256run` to start your program:

```
f256run.bat code\myprogram
```

UNIX folks, still flip your slashes!

```
./f256run.sh code/myprogram
```

9.5. Debugging

If your program fails to compile or isn't doing what you expect, a lot of useful information can be found in the `.builddir` directory that is created under your program's project directory.

10. Definitions

- `block`: an 8k segment of memory, aligned on 8k boundaries.
- `far memory`: RAM not directly accessable by the CPU (from 0x10000 on).
- `low memory`: the first 64k of memory (0x0000 to 0xffff).
- `overlay`: code that lives in far memory that is automatically paged in on demand.
- `slot`: an 8k segment of near memory, aligned on 8k boundaries, where blocks can be paged in.
- `word`: on the F256, a word is 16 bits.

TIP	Be sure to check the F256 hardware documentation to ensure the memory range you wish to use is valid for what you are trying to accomplish. For example, not all `far memory` addresses can be accessed by the video subsystem. Know your hardware!

11. API Overview

The ***F256lib*** API is broken down into several general sections:

Constants	Provides friendly names for often used hardware addresses, kernel functions, and more.
Types	Helpful data types to keep you code ambiguity-free.
Helpers	Functions for safely dealing with memory and bits.
Kernel	Access to the TinyCore MicroKernel.
DMA	Direct memory access for fast fills and copies.
Math Co-Processor	Highly performant common math functions.
Random Numbers	Easy hardware-assisted random numbers.
Text	Text display layer handling.
Graphics	Common graphic functions shared by the bitmap, sprite, and tile layers.
Bitmaps	Bitmap graphic features.
Sprites	Functions for defining and manipulating sprites.
Tiles	Tile-based graphics features.
File I/O	MicroKernel file I/O wrapped as standard C file functions.
Platform	Bare minimum features needed to support the compiler standard library.

NOTE	Several sections of the documentation mention "screen refresh rate" dependent settings. At this time, ***F256lib*** only supports 60 Hz.

12. API Details

The following section is a detailed description of the API provided by ***F256lib***. Only the portions of the API that are considered "stable" are documented. You may discover other (potentially dangerous) goodies in the source code.

12.1. Constants

A *lot* of constants are provided by ***F256lib***. Hardware registers, kernel functions, return values, and more. If they were all listed here, this document would be longer than anyone cares to read. For the ugly details, please see the following:

- `f256dev\f256lib\f_api.h` contains kernel information.
- `f256dev\include` is full of header files containing handy constants.

12.2. Types

F256lib provides clear, helpful, data types. It's highly recommended that you use `stdint.h`-style types for integer data to avoid size confusion. In addition, we recommend using `char` only when you really mean character data, `byte` for unsigned bytes, and `bool` for boolean values. `true` and `false` are also provided.

12.3. Helpers

These helper functions are convenience features for dealing directly with memory. Using them ensures the compiler will not optimize out memory accesses that appear to "do nothing" but may be significant to the hardware.

12.3.1. PEEK

```
byte value = PEEK(uint16_t address);
```

Returns the 8 bit value of a given low memory address.

12.3.2. POKE

```
POKE(uint16_t address, byte value);
```

Writes an 8 bit value to a given low memory address.

12.3.3. PEEKW

```
uint16_t value = PEEKW(uint16_t address);
```

Returns the 16 bit value starting at a given low memory address.

12.3.4. POKEW

```
POKEW(uint16_t address, uint16_t value);
```

Writes a 16 bit value starting at a given low memory address.

12.3.5. POKEA

```
POKEA(uint16_t address, uint32_t value);
```

Writes a 24 bit value starting at a given low memory address. The remaining 8 bits of the `value` argument are ignored.

12.3.6. PEEKD

```
uint32_t value = PEEKD(uint16_t address);
```

Returns the 32 bit value starting at a given low memory address.

12.3.7. POKED

```
POKED(uint16_t address, uint32_t value);
```

Writes a 32 bit value starting at a given low memory address.

12.3.8. FAR_PEEK

```
byte value = FAR_PEEK(uint32_t address);
```

Returns the 8 bit value of a given far memory address.

12.3.9. FAR_POKE

```
FAR_POKE(uint32_t address, byte value);
```

Writes an 8 bit value to a given far memory address.

12.3.10. FAR_PEEKW

```
uint16_t value = FAR_PEEKW(uint32_t address);
```

Returns the 16 bit value starting at a given far memory address.

12.3.11. FAR_POKEW

```
FAR_POKEW(uint32_t address, uint16_t value);
```

Writes a 16 bit value starting at a given far memory address.

12.3.12. LOW_BYTE

```
byte result = LOW_BYTE(int16_t value);
```

Returns the lower byte of a 16 bit word.

12.3.13. HIGH_BYTE

```
byte result = HIGH_BYTE(int16_t value);
```

Returns the upper byte of a 16 bit word.

12.3.14. SWAP_NIBBLES

```
byte result = SWAP_NIBBLES(byte value);
```

Returns the byte with the nibbles swapped.

12.3.15. SWAP_UINT16

```
uint16_t result = SWAP_UINT16(uint16_t value);
```

Returns the word with the upper and lower bytes swapped.

12.3.16. CHECK_BIT

```
byte result = CHECK_BIT(byte value, byte position);
```

Returns non-zero if the bit at the indicated position is set.

12.3.17. TOGGLE_BIT

```
byte result = TOGGLE_BIT(byte value, byte position);
```

Returns the byte with the bit at the indicated position flipped.

12.3.18. CLEAR_BIT

```
byte result = CLEAR_BIT(byte value, byte position);
```

Returns the byte with the bit at the indicated position cleared.

12.3.19. SET_BIT

```
byte result = SET_BIT(byte value, byte position);
```

Returns the byte with the bit at the indicated position set.

12.4. Kernel

Kernel functions provide simplified access to the features provided by Gadget's TinyCore MicroKernel. MicroKernel documentation is beyond the scope of this document.

Kernel interaction relies on two global variables:

```
char kernelError;
kernelArgsT *kernelArgs;
```

- `kernelError` will be set to the error code, if any, returned by the kernel.
- `kernelArgs` provides structures to pass data to and receive data from the kernel.

Keyboard Reading Example:

```
do {
   kernelNextEvent();
   if (kernelEventData.type == kernelEvent(key.PRESSED)) {
      printf("Key %c pressed.\n", kernelEventData.key.ascii);
   }
} while(true);
```

TIP	Take a look at the `f256dev\examples\sprites` program and `f256dev\f256lib\f_file.c` from the ***F256lib*** source code for kernel usage examples.

IMPORTANT	Using these kernel functions along with file I/O statements or `getchar()` can result in missing an event. File I/O and keyboard reading have their own kernel polling loops and will discard any events they aren't waiting on for themselves.

12.4.1. kernelCall

```
char kernelCall(function);
```

Calls one of the functions provided by the kernel. You will need to fill in the proper fields in the `kernelArgs` structure prior to calling. `kernelArgs` will be populated, as appropriate, if the call was successful.

12.4.2. kernelEvent

```
size_t eventType = kernelEvent(size_t event);
```

Converts a named kernel `event` to the value expected by the `kernelEventData.type` structure. See the above example for usage.

12.4.3. kernelGetPending

```
byte kernelGetPending(void);
```

Returns the number of pending kernel events that need to be handled by your program.

12.4.4. kernelNextEvent

```
void kernelNextEvent(void);
```

When using the kernel, you must call this function to "pump" the kernel's event queue. Failing to call this often enough can starve the kernel of event objects.

12.5. DMA

IMPORTANT	At the moment, DMA access is... hit and miss. Feel free to try it but don't expect it to be stable. More DMA features will be available in the future.

12.5.1. dma2dFill

```
void dma2dFill(uint32_t start, uint16_t width, uint16_t height, uint16_t stride, byte
value);
```

Fills a rectangular section of memory (near or far) with a value.

12.5.2. dmaFill

```
void dmaFill(uint32_t start, uint32_t length, byte value);
```

Fills a linear section of memory (near or far) with a value.

12.6. Math Co-Processor

These functions provide vastly faster math operations than using the CPU alone. Use them whenever possible!

12.6.1. mathSignedDivision

```
int16_t  mathSignedDivision(int16_t a, int16_t b);
```

Performs signed division on two 16 bit signed integer values. Any remainder is discarded.

12.6.2. mathSignedDivisionRemainder

```
int16_t  mathSignedDivisionRemainder(int16_t a, int16_t b, int16_t *remainder);
```

Performs signed division on two 16 bit signed integer values. The remainder is returned in the pointer passed as the `remainder` argument.

12.6.3. mathSignedMultiply

```
int32_t  mathSignedMultiply(int16_t a, int16_t b);
```

Performs signed multiplication of two signed 16 bit integer values.

NOTE	The return value is 32 bits.

12.6.4. mathUnsignedAddition

```
uint32_t mathUnsignedAddition(uint32_t a, uint32_t b);
```

Performs unsigned multiplication of two unsigned 16 bit integer values.

NOTE	The return value is 32 bits.

12.6.5. mathUnsignedDivision

```
uint16_t mathUnsignedDivision(uint16_t a, uint16_t b);
```

Performs unsigned division on two 16 bit unsigned integer values. Any remainder is discarded.

12.6.6. mathUnsignedDivisionRemainder

```
uint16_t mathUnsignedDivisionRemainder(uint16_t a, uint16_t b, uint16_t *remainder);
```

Performs unsigned division on two 16 bit unsigned integer values. The remainder is returned in the pointer passed as the `remainder` argument.

12.6.7. mathUnsignedMultiply

```
uint32_t mathUnsignedMultiply(uint16_t a, uint16_t b);
```

Performs unsigned multiplication of two unsigned 16 bit integer values.

NOTE	The return value is 32 bits.

12.7. Random Numbers

Generating good psuedo-random numbers is a significant challenge. Fortunately the Foenix F256 has hardware-assisted random number generation.

12.7.1. randomRead

```
uint16_t randomRead(void);
```

Returns the next psuedo-random unsigned 16 bit integer.

12.7.2. randomSeed

```
void     randomSeed(uint16_t seed);
```

Specifies the starting value for the psuedo-random number generator. Use this to generate a reproducable sequence of "random" values.

On startup, the psuedo-random number generator is seeded from the real-time clock, if available.

12.8. Text

On startup, the text layer is cleared, background colors are disabled, the foreground color is set to white, the background to black, the cursor disabled, and the font set to double-height producing an 80 column by 25 line display.

```
void textClear(void);
```

Clears the text layer to the current text colors. Returns the cursor to the upper-left corner.

```
void textDefineBackgroundColor(byte slot, byte r, byte g, byte b);
```

Defines one of the 16 available colors in the text background color table. `slot` specifies which color to define, 0 through 15. `r`, `g`, and `b` specify how much of each color component to use, 0 through 255.

```
void textDefineForegroundColor(byte slot, byte r, byte g, byte b);
```

Defines one of the 16 available colors in the text foreground color table. `slot` specifies which color to define, 0 through 15. `r`, `g`, and `b` specify how much of each color component to use, 0 through 255.

```
void textEnableBackgroundColors(bool b);
```

Enabling background colors obscures any graphics layers behind the text layer.

```
void textGetXY(byte *x, byte *y);
```

Returns the position of the cursor in the pointers `x` and `y`.

```
void textGotoXY(byte x, byte y);
```

Moves the cursor to a new position at `x`, `y`. The valid values for the new location depend on the current screen refresh rate and text doubling settings.

```
void textPrint(char *message);
```

Displays the string `message` at the current cursor position. This function uses much less code than using `printf()`.

```
void textPrintInt(int32_t value);
```

Displays the signed integer `value` at the current cursor position. This function uses much less code than using `printf()`.

```
void textPrintUInt(uint32_t value);
```

Displays the unsigned integer `value` at the current cursor position. This function uses much less code than using `printf()`.

```
void textSetColor(byte f, byte b);
```

Specifies which color slots to use for the foreground and background colors of future text output.

```
void textSetCursor(byte c);
```

Specifies which ASCII character code to use as a cursor. Setting the cursor to `0` will disable it.

```
void textSetDouble(bool x, bool y);
```

Specifies wether or not to double the size of displayed characters on the `x`, `y`, or both axis. Depending on the video refresh rate, this allows you to produce text displays of the following sizes:

- At 60 Hz:
  - 80x60
  - 40x60
  - 80x30
  - 40x30
- At 70 Hz:
  - 80x50
  - 40x50
  - 80x25
  - 40x25

12.9. Graphics

Functions in this category affect the graphics system in general. They are used to configure layers, colors, etc. To actually produce something on the display, you'll need to use these functions in conjunction with functions from another graphics subsystem.

The graphics system consists of three layers. By default, each layer is set to its associated `bitmap`. (Layer 0 is Bitmap 0 and so on.)

12.9.1. graphicsDefineColor

```
void graphicsDefineColor(byte clut, byte slot, byte r, byte g, byte b);
```

Defines one of the 256 available colors in one of the four graphics color tables. `clut` is which color lookup table to modify, 0 through 4. `slot` specifies which color index to define, 0 through 255. `r`, `g`, and `b` specify how much of each color component to use, 0 through 255.

12.9.2. graphicsSetLayerBitmap

```
void graphicsSetLayerBitmap(byte layer, byte which);
```

Specifies that `layer` (0 to 3) should display bitmap number `which` (0 to 3).

12.9.3. graphicsSetLayerTile

```
void graphicsSetLayerTile(byte layer, byte which);
```

Specifies that `layer` (0 to 3) should display tilemap number `which` (0 to 3).

12.9.4. graphicsWaitVerticalBlank

```
void graphicsWaitVerticalBlank(void);
```

Pauses program execution until the start of a vertical blank.

12.10. Bitmaps

The Foenix F256 can display up to three full-screen bitmaps at a time. Depending on the screen refresh, the bitmaps are:

- At 60 Hz:
  - 320x240 pixels
- At 70 Hz:
  - 320x200 pixels

On startup, all three bitmaps are assigned to each associated graphics layer, but are not visible. The three bitmaps are located at the following memory addresses at the top of the far memory available to the Vicky graphics chip:

- Page 0 - 0x6c000 → 0x7ebff
- Page 1 - 0x58000 → 0x6abff
- Page 2 - 0x44000 → 0x56bff

IMPORTANT	Be sure your program avoids using these memory ranges if you use these pages!

NOTE	You may have noticed each bitmap page is aligned on an 8k boundary leaving 5k unused between each page. If you need this memory, go for it.

12.10.1. bitmapClear

```
void bitmapClear(void);
```

Clears the currently active bitmap to the current color.

12.10.2. bitmapGetResolution

```
void bitmapGetResolution(uint16_t *x, uint16_t *y);
```

Returns the size of the current bitmap in the pointers `x` and `y`.

12.10.3. bitmapLine

```
void bitmapLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
```

Draws a line on the current bitmap in the current color from `(x1, y1)` to `(x2, y2)`.

12.10.4. bitmapPutPixel

```
void bitmapPutPixel(uint16_t x, uint16_t y);
```

Plots a single pixel on the current bitmap in the current color at `(x, y)`.

12.10.5. bitmapSetActive

```
void bitmapSetActive(byte p);
```

Specifies the currently active bitmap from 0 to 3.

12.10.6. bitmapSetAddress

```
void bitmapSetAddress(byte p, uint32_t a);
```

Allows you to change where in far memory a bitmap is located. Bitmap page `p` will be assigned to start at memory address `a`. Addresses must be aligned on an 8k boundary (evenly divisible by 8192).

12.10.7. bitmapSetCLUT

```
void bitmapSetCLUT(byte clut);
```

Specifies which color look up table to use for the current bitmap. Each bitmap can have its own palette of 256 colors.

12.10.8. bitmapSetColor

```
void bitmapSetColor(byte c);
```

Specifies the current color index in the current color table to use for future drawing operations on the current bitmap.

12.10.9. bitmapSetVisible

```
void bitmapSetVisible(byte p, bool v);
```

Specifies if bitmap page `p` is visible or not.

12.11. Sprites

The Foenix F256 provides 64 independent sprites with three sprite layers. Sprites are small bitmaps of the following sizes:

- 8x8
- 16x16
- 24x24
- 32x32

To allow sprites to smootly enter and exit the screen, sprite coordinates are larger than bitmap and tile coordinates. Sprite coordinates are:

- For 60 Hz:
  - 352x272

- For 70 Hz:
  - 352x232

The first 32 pixels are off the screen to the left and top. Sprites beyond the horizontal and vertical resolutions shown are off the right and bottom of the screen. See the hardware documentation for more details.

12.11.1. spriteDefine

```
void spriteDefine(byte s, uint32_t address, byte size, byte CLUT, byte layer);
```

Assigns a block of memory to a given sprite. Sprite number `s` (0 to 63) will pull its pixel data starting at `address` which can be in either near or far memory. `size` is either 8, 16, 24, or 32. Each sprite can pull its color information from one of the four (0 to 3) color lookup tables specified in `CLUT`. `layer` specifies which of the three sprite layers you wish to use (0 to 2).

12.11.2. spriteSetPosition

```
void spriteSetPosition(byte s, uint16_t x, uint16_t y);
```

Positions sprite number `s` at location `(x, y)`.

TIP Remember! Sprite coordinates are not the same as other graphics coordinates!

12.11.3. spriteSetVisible

```
void spriteSetVisible(byte s, bool v);
```

Determines if sprite number `s` is visible on the display. All sprites begin life hidden.

12.12. Tiles

Tile maps are composed of individual "tiles" of pixels that are either 8x8 or 16x16 in size. Tile maps can be smoothly scrolled horizontally and vertically. Like bitmaps and sprites, each

12.12.1. tileDefineTileMap

```
void tileDefineTileMap(byte t, uint32_t address, byte tileSize, uint16_t mapSizeX,
uint16_t mapSizeY);
```

Defines a tile map from a block of RAM. `t` is which map to define (0 to 2). `address` is where in near or far memory the map data is located. `tileSize` specifies the size of your tiles (either 8 or 16). Finally, `mapSizeX` and `mapSizeY` determine the size of the map.

12.12.2. tileDefineTileSet

```
void tileDefineTileSet(byte t, uint32_t address, bool square);
```

Specifies where in memory to find a given tile set. `t` determines which of the eight available (0 to 7) tile sets we're defining. `address` is where in near or far memory the tile data is located. `square` indicates if the tile data in memory is laid out in a square or a single tile width vertical strip.

12.12.3. tileSetScroll

```
void tileSetScroll(byte t, byte xPixels, uint16_t xTiles, byte yPixels, uint16_t
yTiles);
```

Changes the position of a given tile map `t`. `xPixels` and `yPixels` allow for fine scrolling of the tile map while `xTiles` and `yTiles` determine the upper left tile to be displayed.

12.12.4. tileSetVisible

```
void tileSetVisible(byte t, bool v);
```

Determines the visibility `v` of a particular tile map `t` (0 to 2).

12.13. File I/O

File I/O from the MicroKernel is mapped to standard C library file routines with a few minor changes for how the F256 handles drives.

Supported functions are listed below. For usage details, see the documentation for the standard C library.

- `int closedir(DIR *dirp);`
- `int fclose(FILE *stream);`
- `FILE *fopen(const char *pathname, const char *mode);`
- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `int fseek(FILE *stream, long offset, int whence);`
- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `int mkdir(const char *pathname, mode_t mode);`
- `DIR *opendir(const char *name);`
- `struct dirent *readdir(DIR *dirp);`
- `int rename(const char *oldpath, const char *newpath);`
- `void rewind(FILE *stream);`

- `int rmdir(const char *pathname);`
- `int unlink(const char *pathname);`

The following macros are available for identifying which type of directory entry is being returned in `dirent->d_type`:

- `_DE_ISREG(d_type)`
- `_DE_ISDIR(d_type)`
- `_DE_ISLBL(d_type)`
- `_DE_ISLNK(d_type)`

NOTE	Seeking is only supported forward from the start of the file. `SEEK_SET` is defined for use as `whence`. Currently not supported by IEC drives.
NOTE	Not all IEC devices support subdirectories.
NOTE	Pathnames on the F256 use the forward slash (`/`) to delimit subdirectories. Similar to Windows, the F256 uses the concept of a "drive". Unlike Windows, drives are numbered, not lettered, and start at `0` which is the built-in SD card reader. An example path with a drive and subdirectory looks like this: `0:/DOCS/README.TXT`
IMPORTANT	Do not mix kernel access with file I/O. Using file I/O inside a kernel event loop will cause the loop to miss events!

12.14. Platform

The absolute lowest level features needed to make the compiler and its library work.

12.14.1. __putchar

```
void __putchar(char c);
```

Displays the character `c` on the screen.

12.14.2. getchar

```
int getchar(void);
```

Reads a single key from the keyboard.

IMPORTANT	If you are using kernel functions, do not call `getchar` inside your kernel event loop. Use the kernel event `key.PRESSED` instead to avoid losing events. Likewise, avoid standard C library routines that expect input from the keyboard.

13. Using Code Overlays

One of the most powerful features of this toolchain is the ability to write programs that use more than the 64k of RAM addressable by the CPU. Using the MMU (Memory Management Unit) of the F256, ***F256lib*** is able to automatically swap code in from far memory on demand. The only additional work needed by the programmer is to specify which bits of code are to be relocated to far memory. This is achieved with a `#define SEGMENT_name` statement.

Example:

```
#define F256LIB_IMPLEMENTATION
#include "f256lib.h"


void firstSegment(int arg1, int arg2);
void secondSegment(int arg1, int arg2);
void moreFirstSegment(int arg1, int arg2);


// This is the first segment we've defined.
// The linker will place this code in the first available
// far memory slot (default 0x10000).
#define SEGMENT_FIRST

void firstSegment(int arg1, int arg2) {
   printf("firstSegment = %d\n", arg1 + arg2);
   secondSegment(arg1, arg2);
}

// This is the second segment we've defined.
// The linker will place this code in the next available
// far memory slot (default 0x12000).
#define SEGMENT_SECOND

void secondSegment(int arg1, int arg2) {
   printf("secondSegment = %d\n", arg1 + arg2);
   moreFirstSegment(arg1, arg2);
}

// Back to the first segment.
// The linker will place this code immediately
// after the previous first segment code.
#define SEGMENT_FIRST
```

```
void moreFirstSegment(int arg1, int arg2) {
   printf("moreFirstSegment = %d\n", arg1 + arg2);
}

// Back to near memory, the 64k visible to the processor.
// By default, this segment begins at 0x300.
#define SEGMENT_MAIN

int main(int argc, char *argv[]) {
   (void)argc;
   (void)argv;

   firstSegment(1, 2);

   // Spin.
   for (;;);

   return 0;
}
```

TIP	When building your program, the amount of space being used in each segment will be displayed (or an error that you have overflowed a segment). Use this information to help decide how to segment your program.

IMPORTANT	You cannot change where the overlay tool will place your far memory code. It always starts at `0x10000` and increments the address `0x2000` (8k) for each additional segment. When using overlays, be careful to not overwrite this memory with other data!

NOTE	Segment names can be anything you like. `SEGMENT_MAIN` is predefined and refers to the near 64k accessable by the CPU.

NOTE	The `SEGMENT_` defines are per-C file. Placing one into a header file that is included into a C file will have no effect. Each C file starts in `SEGMENT_MAIN`.

NOTE	At the moment, you can only move code into far memory, not variables.

14. Embedding Binary Data

Embedding binary data (sprites, tiles, bitmaps, whatever) into your program at a specific memory address can be easily achieved with the `EMBED` macro.

14.1. EMBED

```
EMBED(name, "filename", address)
```

Embeds the contents of `"filename"` at near or far memory `address`. The embedded data will be available by name through the following symbols:

- `const char name_start[];`
- `const char name_end[];`

Where `name` is the prefix of each symbol.

NOTE	`EMBED` arguments **must** be literals. You cannot use #defines or variables.
NOTE	`EMBED` names (first argument) **must** begin with unique prefixes. For example, "TILES" and "TILES_PALETTE" will cause an error due to "TILES" being a complete substring of "TILES_PALETTE".

15. Optional Options

Before including ***F256lib***, you may wish to define one or more of the following to customize the library for your application:

`#DEFINE`	**Description**
`SWAP_RESTORE`	When using functions such as `FAR_PEEK` or bitmap features, return the swap slot back to it's original memory location before returning. By default, the swap slot is left in whatever state the last function that used it switched it to.
`WITHOUT_GRAPHICS`	Shortcut for `WITHOUT_BITMAP`, `WITHOUT_TILE`, and `WITHOUT_SPRITE`.
`WITHOUT_BITMAP`	Disables bitmap functions.
`WITHOUT_TILE`	Disables tilemap functions.
`WITHOUT_SPRITE`	Disables sprite functions.
`WITHOUT_FILE`	Disables file handling support.
`WITHOUT_KERNEL`	Disables microkernel support. Also disables `FILE`, `PLATFORM`, and `MAIN`.
`WITHOUT_TEXT`	Disables text functions. Also disables `PLATFORM`.
`WITHOUT_MATH`	Disables math co-processor support. Also disables `TEXT`, `PLATFORM` and `BITMAP`.

#DEFINE	Description
WITHOUT_MAIN	Removes support for the standardized `int main(int argc, char *argv[])` function and replaces it with `int main(void)`. You must also manually call `f256Init()` at the start of your program.
WITHOUT_PLATFORM	Removes basic C library I/O support for `getchar()` and `__putchar()`. This removes `printf()`.
WITHOUT_DMA	Removes DMA support. In the future, will also disable `BITMAP`.

Example:

```
#define WITHOUT_FILE
#define WITHOUT_SPRITE
#define WITHOUT_TILE
#define F256LIB_IMPLEMENTATION
#include "f256lib.h"

int main(int argc, char *argv[]) {
   return 0;
}
```

The compiler is really good at removing dead code so this isn't usually needed. It's primarily helpful to purposefully diable features so you receive errors if you attempt to use them.

16. Support

Drop by the Foenix Retro Systems Discord! I am almost always online as "sduensin".

Additional information on the F256 family of computers can be found on the F256 wiki.

As a last resort, you can email me, Scott Duensing, at scott@kangaroopunch.com and I'll probably ask you to join the Discord. :-)