EA DOGM128

An Overview of Research on the Electronic Assembly DOGM128 Dot-Matrix Display

Frank Pernice

ESE 476 Fall 2013

Statement of Work

Abstract

In many ESE 380 and 381 courses, the design projects often utilize a three-line LCD display with a built-in character set. While this is sufficient for most purposes, it does not allow for flexibility or creativity in most projects. As a result, the user-interfaces in these courses generally look the same across different lab teams. In industry, when a product reaches a consumer, most people will judge the user-interface over other features. RIM's Blackberry interface is a perfect example of this phenomenon. RIM has failed to make significant updates to their user interfaces and consequently have been overthrown by Android and iOS. Just because a product has the newest features and most advanced technology of the day does not mean it is immune from being judged on its user-interface. We must ask why it is that Google has overtaken Altavista and Facebook has overtaken MySpace in recent years. And then of course there's the recent healthcare.gov scandal to take into consideration. In all of these cases, an eloquent and functional user interface was a major driving factor (though not the only factor) in making or breaking these websites. User interfaces are so crucial that there are often computer science classes dedicated to the topic and entire businesses which revolve around this subject.

It is crucial that students interested in embedded systems design be introduced to user interface design and have the freedom to create a user interface on their own. Unlike the three-line LCD's, the Electronic Assembly DOGM128 allows the user to draw anything anywhere on the screen. There is no restriction on which characters may or may not be painted on the display. Of course, this degree of freedom introduces a number of complexities which did not need to be taken into consideration using the three-line display. Hence, included with this documentation is a fully-documented library which should make interfacing with the display rather trivial.

Table of Contents

Hardware

Initial Power Issue and Foray into New Microcontrollers

Out of the box, it is apparent that the DOGM128 is a 3.3V logic device while the ATMega128, the microcontroller of choice in ESE 381, is a 5V logic device (and it cannot be run at 3.3V). This posed an interesting challenge; though not a difficult one to solve (a voltage divider was used to step the logic level down). The challenge was in finding an alternative to the ATMega128 daughterboard which is currently being used. This was spurred on more by the desire to move away from the Futurlec daughterboard due to customer service issues as well as the desire to convert the lab to 3.3V devices (following current industry trends). As mentioned, the ATmega128 is strictly a 5V device and is incapable of running on a 3.3V source, despite previous beliefs. However, there are two solutions: the ATMega128L and the ATMEGA128A. The ATMega128L is a low-power device which is completely compatible with the ATMega128's instruction set. However, it is strictly a 3.3V device and cannot be operated at 5V. Also, it cannot be run at a frequency larger than 8MHz whereas the ATMega128 can run at up to 16MHz. This may be undesirable for many applications. The other alternative is the ATMega128A, which is both 5V and 3.3V compatible, is fully compatible with the ATMega128's instruction set, and can run up to 16MHz. However, obtaining a breadboard friendly daughterboard with either of these alternatives is easier said than done. The closest I was able to come was the CuteDigi ATMega128A System Board[1]; however, the pins are incompatible with the lab's breadboards and a breakout board must be fabricated.

EA DOGM128 Interface

One of the main attractions of the EA DOGM128 is that it communicates over serial peripheral interface (SPI), which is a staple topic in both ESE 380 and ESE 381. Like many devices used in these courses, it acts as an SPI slave. However, the DOGM128 has quite an interesting implementation of SPI. Information itself is transmitted over a traditional 3-wire interface. Those three wires consist of chip select ($\overline{SS}$), SPI clock (SCK), and master-out slave-in ($MOSI$); the screen does not send data back to the master, hence the absence of the master-in slave out ($MISO$) signal. The timing diagram provided in the datasheet (also shown below in **figure 1**) indicates that this device has a clock polarity ($CPOL$) of 0 and a clock phase ($CPHA$) of 0 and that information is written to the screen most-significant-bit first.

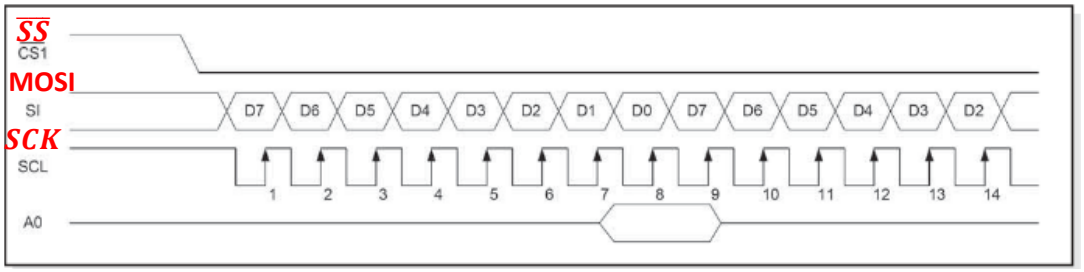

Figure 1: EA DOGM128 SPI Timing Diagram

The interesting part about this device is that it introduces two additional signals to its communication protocol, a reset signal and an information-type signal. As the name implies, the reset signal resets the device to factory defaults. The information-type signal (awkwardly referred to as $A0$ in the datasheet) determines whether the information being sent to the device is a command which tells the screen to do something or data to be displayed. It is important that this signal be appropriately set when sending commands or data to the display. It is equally important (though obvious) that the active-low reset signal be set to a logic 1 before trying to do anything with the display. This should not be an issue for students since this work is already done of them in the provided library. The screen can be operated with an SCK of up to 20MHz, given that it is wired to a 3.3V source (as the library assumes and as is shown in the schematics on the following pages). Unlike the 3-line displays, there should be no problems with the ATMega16 or the ATMega128 clocking in data too fast.

Backlight

Although the DOGM128 itself is not 5V tolerant, its compatible backlight is (it consists of nothing more than six LEDs). Therefore, they can ideally be driven by any voltage greater than their threshold voltage (provided proper current limiting resistors are in place). Due to initial issues with the display not powering on via the 3.3V regulator (LM3940IT), I chose to power the backlight separately so that I could debug the issue. The issue ended up being caused by a silly mistake (backwards ribbon cable) and the LM3940IT should be adequate enough to power both the backlight and the display since it provides up to 1A of current while the display plus the backlight should draw a typical maximum of 5mA when a white backlight is used. Since my priority was to develop a library and not mess around with wiring, I left the backlight running on 5V for the sake of time. Different colored backlights require different current limiting resistors as indicated by the schematics on the following pages. The provided values assume 5V backlight operation.

The part number for the backlight (which is sold separately from the DOGM128 and) is **EA LED55X46-Y**, where **Y** is a variable that represents the backlight must be soldered on color as indicated in **Table 1** below. The backlight should be soldered to the screen prior to lab use.

Y	Color
W	White
G	Yellow/Green
B	Blue
R	Red
A	Amber
E	Green
RGB	Full color RGB

Table 1: Y values for the EA LED55X46-Y

Circuitry Images and Schematics

The schematics provided on the following pages illustrate the interface between the EA DOGM128, the EA LED55X46-Y, and the ATMega128. Not shown on the schematics are the LM3940IT, the ATMega128, and the JTAG connections. The images below show the placement of the components on the main breadboard and the side board as well.

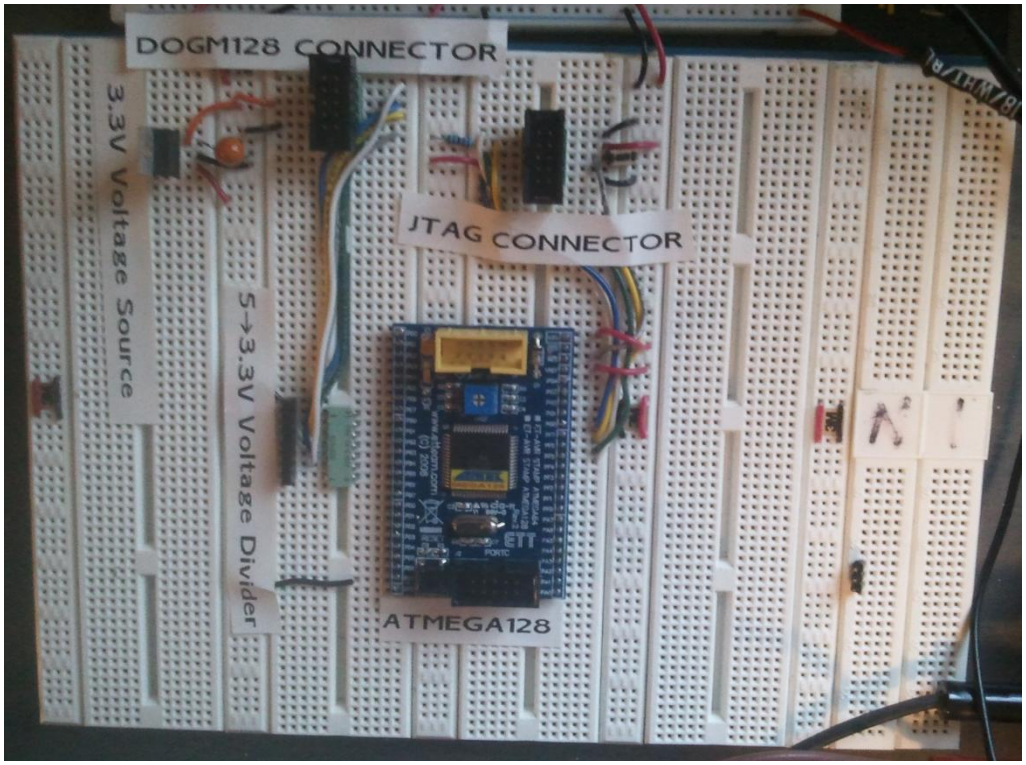

Figure 2: Main Breadboard

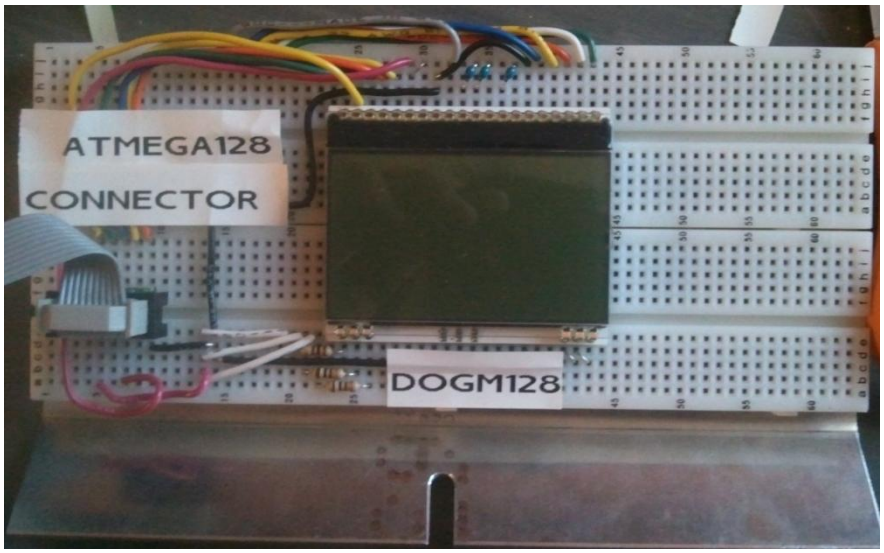

Figure 3: Side Board

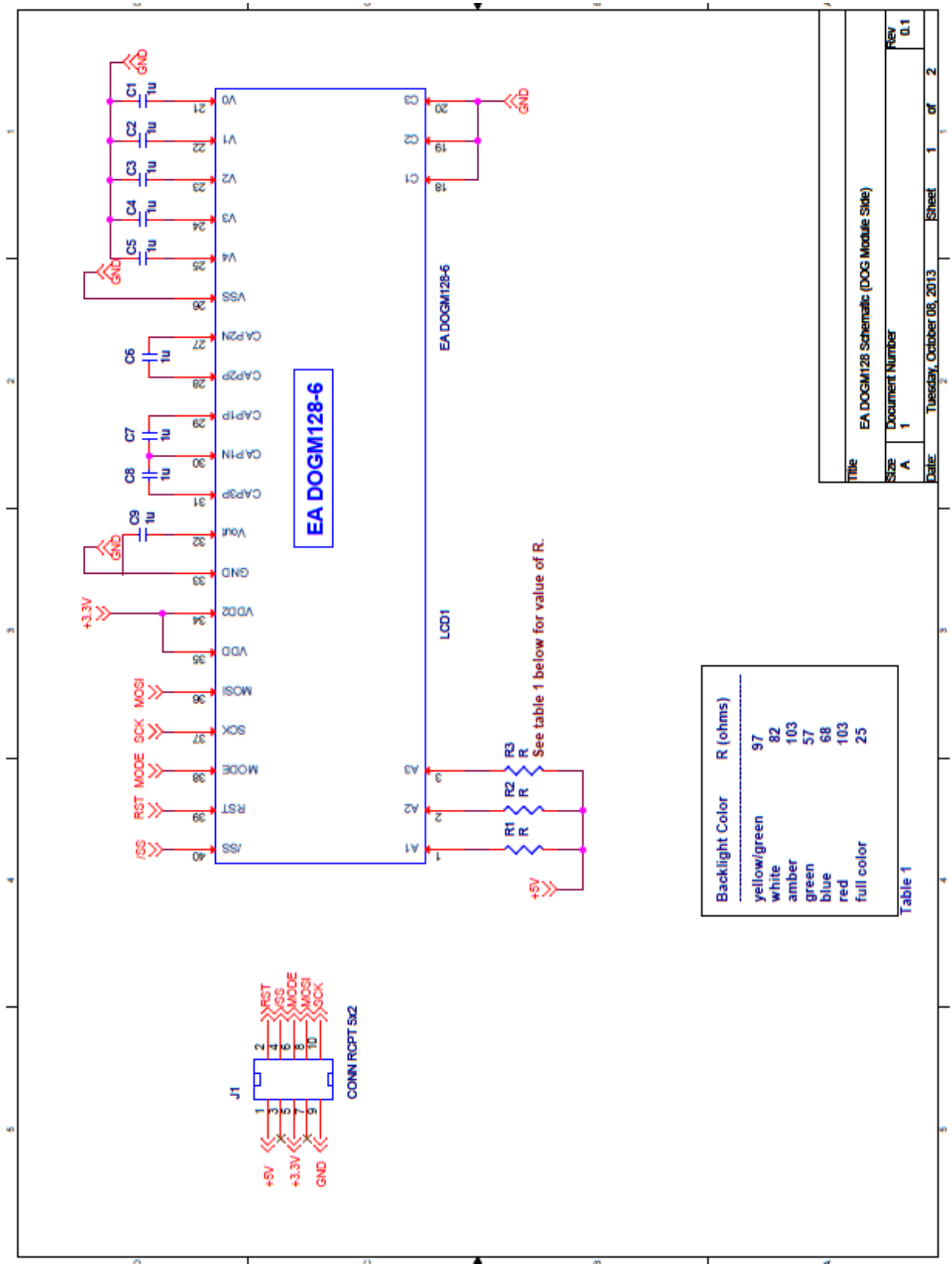

Figure 4: Side-board schematic illustrating the EA DOGM128 soldered on top of the EA LED55X46-Y

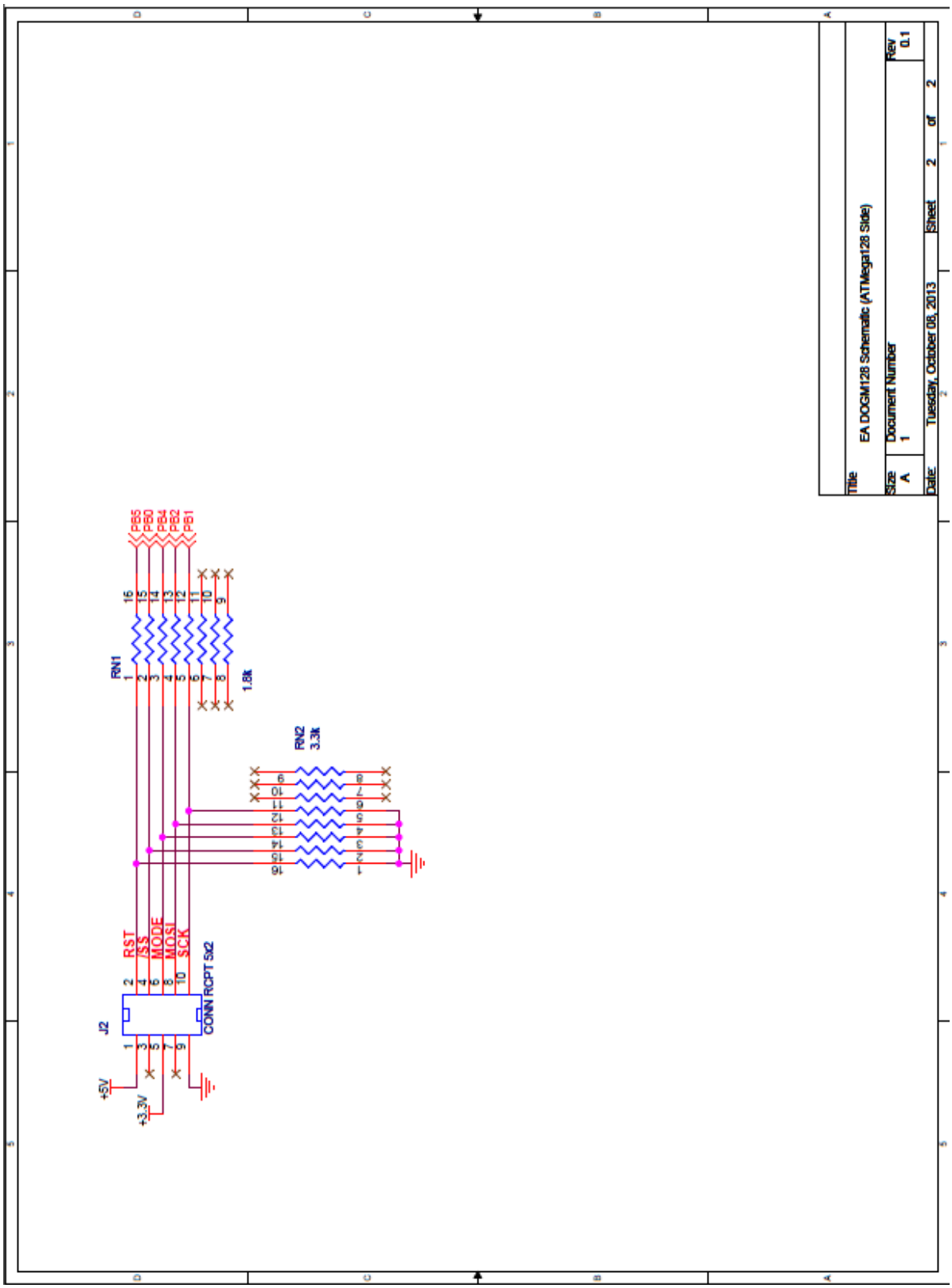

Figure 5: Main bread board schematic, showing voltage divider and connections between ATMega128 and the 10-pin header

Software

Overview of EA DOGM128

The display itself has a 128 x 64 pixel resolution. The 64-pixel height is divided into 8 sections, referred to as *pages*. Each *page* is 8 pixels high by 128 pixels wide. This is illustrated in the diagram below.

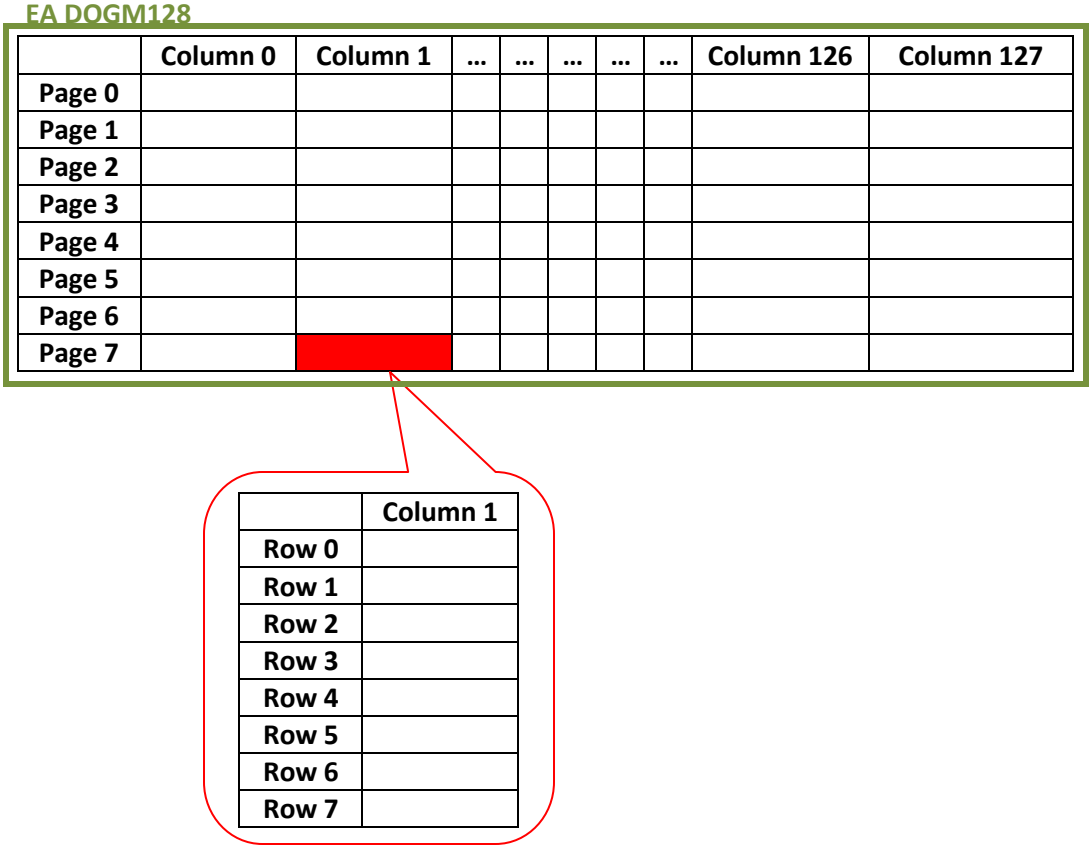

Figure 6: Illustration of screen layout

Each byte of **data** sent to the display will fill one page/column intersection. So, for example suppose we address page 7, column 1 as shown in **Figure 6**. If we write the value 0x5F (0b10011111) to the display, we will see that portion of the display populated as shown on the following page in **Figure 7**. The inquisitive reader will realize that by writing 0x5F to the display, we create an exclamation mark!

	Column 1
Row 0	1
Row 1	1
Row 2	1
Row 3	1
Row 4	1
Row 5	0
Row 6	0
Row 7	1

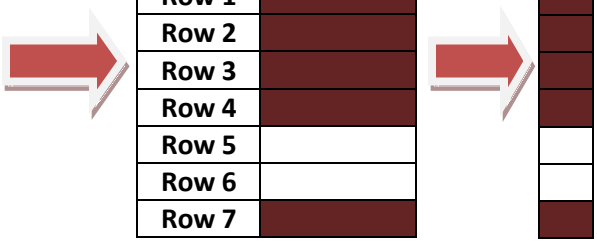

Figure 7: How a page/column intersection is populated

As each data byte is written to the display, the column address is automatically incremented. After writing 0x5F to page 7, column 1, the display automatically jumps to page 7, column 2. However, once we reach the end of the page (column 127), the column is **not** automatically reset to zero. We must do this manually. To better visualize this concept, imagine a typist using an old-fashioned typewriter[2]. The typist can type an entire line without interruption. However, once he or she reaches the end of the line, the typewriter sounds an all too familiar ring indicating that the end of line has been reached. Once the typist hears this ring, they must manually return the carriage to the left-side of the page and advance to the next line in order to continue typing. Writing to the EA DOGM128 utilizes the same exact principle.

Writing characters to the display

The DOGM128 is quite a "dumb" device in that it paints every pixel as instructed by the SPI master and little more. Unlike the three-line screens currently used in ESE 380 and ESE 381, this display has no built-in character set. In order to paint characters to the display, the user must do so pixel by pixel. In order to do this easily, it is customary to store an ASCII character set on the SPI master device in the form of a look-up table. The characters in my library are 5 pixels wide by 7 pixels high (5x7 pixels). This size was used because it is both very visible and it allows for a sufficient number of characters to be written to each line. It is also very easy and efficient to write characters at this size since we do not have to cross pages to fit an entire character (as would be the case with an 8x10 font, for example).

In order to take full advantage of C's *printf()* function, a large two-dimensional character buffer (array) was created to store every byte to be written to the display. The *putchar()* function is written so that current page and column are kept track of. It allows for the printing of the new-line character and also features text-wrapping should the user exceed the maximum of 21 characters per line. The user may either call *putchar()* directly or may call *printf()*, which will indirectly call *putchar()*. The user may also specify a page or column where they would like to begin drawing text via the dog*_set_column()* and dog*_set_page()* functions. Finally, a user can also write a character in-between pages by specifying a row and column though the dog*_putchar_select()* function. This function, of course, is less efficient than the

regular *putchar()* function since it must divide the character and determine which pixels belong to which page. An example of this is shown in the figure below.

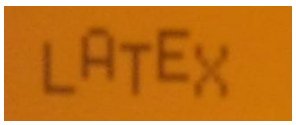

Figure 8: Use of the dog_*putchar_select()* function

Drawing Other Shapes and Objects.

In addition to drawing ASCII characters to the screen, users can draw the following shapes and objects to the screen using the functions provided in the library:

- individual pixels
- individual points (variable size dots)
- lines
- rectangles
- arcs
- circles

These shapes and objects also take advantage of the aforementioned display buffer. All of them are written to the buffer before they are painted to the screen. This allows the user to essentially paint an entire "scene" before actually writing it to the display. So suppose the user wants to underline some text. They can use *printf()* to write their desired phrase to the screen. Then they can draw a line below it by calling the *dog_set_line()* function or (preferably) the *dog_set_h_line()* function. Nothing will be sent to the screen until the user calls the *dog_print_buffer()* function. In this way, everything appears synchronously on the screen.

In order to draw the aforementioned shapes and objects to the screen, it is important to understand the coordinate system used by the library (shown below in **Figure 5**).

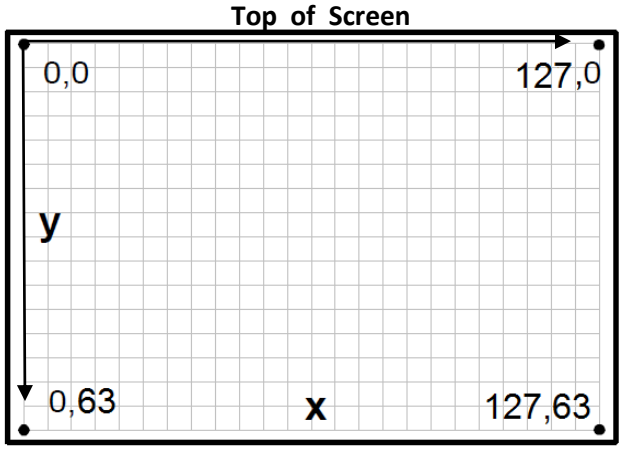

Figure 9: Coordinate System

As indicated by **Figure 5**, the Y coordinates increase as one moves south on the screen while X coordinates increase as one moves east as they do in a conventional Cartesian system. When drawing any shapes or objects to the screen, the user should keep this in mind.

All of the aforementioned shapes (except individual pixels) may be drawn with two thicknesses (1 pixel wide or 3 pixels wide). This is done though a function parameter for each respective shape or object as documented in the code. It is also possible to clear (erase), the aforementioned objects from the screen as well. This is also done through another function parameter. For instance, if the user wants to split a circle into quarters, they would draw a circle onto the screen and then draw "invisible" lines which run horizontally and vertically through the circle to erase portions of the circle.

As noted in the code documentation, arcs and circles are drawn with the same function, *dog_set_arc().* Circles are drawn merely by setting the start and end angle parameters equal to each other. However, it is important to understand how angles are drawn in this coordinate system. For speed, all angles are computed using a normalized sine table so that integer arithmetic can be used for all computations. Additionally, angles must be approximated so that they will fit into an 8-bit number. That means that the user must convert his or her desired angle from actual degrees to what is referred to as $degrees_{LCD}$ in this document. The formula for doing so is as follows:

$$degrees_{LCD} = degrees_{actual} \times \frac{32}{45}$$

So, for example

$0\ degrees_{actual} = 0\ degrees_{LCD}$
$90\ degrees_{actual} = 64\ degrees_{LCD}$
$180\ degrees_{actual} = 128\ degrees_{LCD}$
$270\ degrees_{actual} = 192\ degrees_{LCD}$
$360\ degrees_{actual} = 256\ degrees_{LCD} = 0\ degrees_{LCD}\ (Due\ to\ 8-bit\ overflow)$

In the case where this computation results in a fractional number, the number should be rounded or truncated to a whole number.

It must also be taken into consideration, when drawing arcs, that degrees increase in a clockwise fashion rather than the conventional counter-clockwise fashion due to the fact that Y increases as we move south along the screen. This is illustrated by **Figure 10** on the next page.

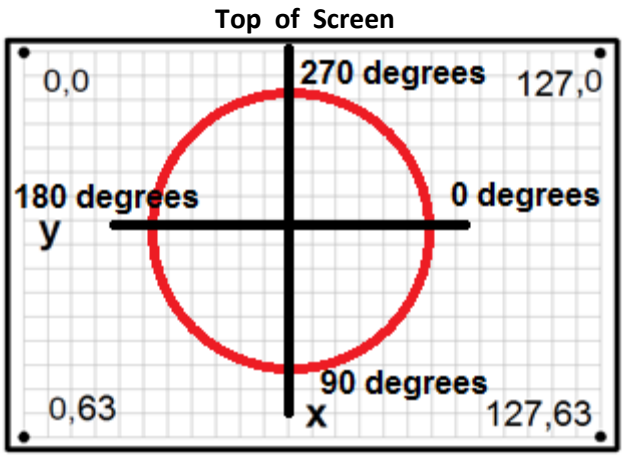

Figure 10: Angles with relation to X and Y.

Library Overview

Portability

The EA DOGM128 library has been developed to maximize portability among different microcontrollers and compilers. This has been accomplished through extensive macro use and the addition of a user configuration header file. The user configuration file allows the user to change a number of settings such as the SPI port and pins, the target device, and SPI register names and bits. This file has been pre-configured for the ATMega128 and the IAR compiler; however it may be modified as needed. This file also contains macros for the *reset* and *A0* ports and pins. This allows the user to move these pins as needed. The rest of the library should not need to be modified so long as this file is properly configured.

Function and Macro Lists

The following tables briefly list the functions and macros that may be called by the user. It omits private functions and macros which are used internally. For brevity, function parameters have been omitted from this list, they are further documented in the code.

MACROS: DOGM128_common	Description
DOG_INIT_SPI()	Initializes the SPI hardware. It configures the SPI Control Register and the SPI2X bit (for MCU's which support it) accordingly so that it may properly communicate the DOG module. It **DOES NOT** configure input and output pins; this must be done by the user. This macro is called by the dog_init() function; it should only be called when re-selecting the DOG module after using some other SPI device.

FUNCTIONS: DOGM128_common	Description
dog_init	Initializes the display, powers it on, sets the contrast, decides whether or not to invert the pixels, and clears the screen for initial use. This function should be called upon start-up before any other functions in this library and should never be called again afterwards.
dog_clear_display	Clears the display, however, it does not clear the display buffer.
dog_clear_buffer	Clears the display buffer
dog_print_buffer	Prints the contents of the display buffer to the screen.
dog_set_contrast	Sets screen contrast. A higher number implies higher contrast. Numbers are in the set [0-63].
dog_invert_pixels	Sets the pixel mode. If inverted, pixels will be normally on when the display is cleared and turn off as data is written to it. That is, the display will appear as it does in **figure 12** in subsequent pages.
dog_power	Turns the display on or off. **NOTE**: This function does not control the backlight, which is hard-wired. The display is initially powered on by the dog_init() function.

NOTE: In all of these functions, the chip select is asserted and unasserted automatically; the user does not have to worry about SPI whatsoever aside from setting the data direction register appropriately and initializing the display or re-initializing SPI with the DOG_INIT_SPI() macro.

FUNCTIONS: DOGM128_characters	Description
putchar	Used to place a single ASCII character on the display buffer. The printf() function utilizes this to print characters to the display buffer. This is restricted to placing characters entirely on a single page and cannot cross pages.
dog_set_page	Used to set the page for character writing
dog_set_column	Used to set the column for character writing
dog_putchar_select	Used to place a character on any portion of the display buffer. It may place characters in-between pages unlike the regular putchar function.

NOTE: When specifying a character position, the given row (or page) and column specify the top-left corner of the character.

FUNCTIONS: DOGM128_pixels	Description
dog_draw_pixel	Sets or clears a single pixel in the display buffer, provided an X and Y coordinate

FUNCTIONS: DOGM128_point	Description
dog_draw_point	Sets or clears a single pixel or multiple pixels in the display buffer, provided an X and Y coordinate for the point's center.

FUNCTIONS: DOGM128_lines	Description
dog_draw_line	Draws or erases a line of any slope to or from the display buffer respectively, provided start and end coordinates.
dog_draw_h_line	Draws or erases a horizontal line to or from the display buffer respectively, provided start and end coordinates. This function is more efficient than dog_draw_line() and should be used whenever possible.
dog_draw_v_line	Draws or erases a vertical line to or from the display buffer respectively, provided start and end coordinates. This function is more efficient than dog_draw_line() and should be used whenever possible.

FUNCTIONS: DOGM128_rectangle	Description
dog_draw_rectangle	Draws or erases an un-filled rectangle to or from the display buffer respectively, provided with a top-left and bottom-right coordinate.

FUNCTIONS: DOGM128_arc	Description
dog_draw_arc	Draws or clears an arc or circle to or from the display buffer respectively, provided start and end angles (computed using the formula on page 9), a center coordinate, and a radius.

File Hierarchy

Several file groups (header file + source file) within this library are dependent on the presence of other file groups in the project. If code size is an issue, certain portions of the library may be omitted (their include files commented out in the master driver header, **DOGM128_driver.h**, and the files themselves detached from the project) if they are not needed for a specific application. However, some files are critical to the library itself and must never be omitted in order to ensure proper library functionality. This is illustrated by **figure 11** on the following page.

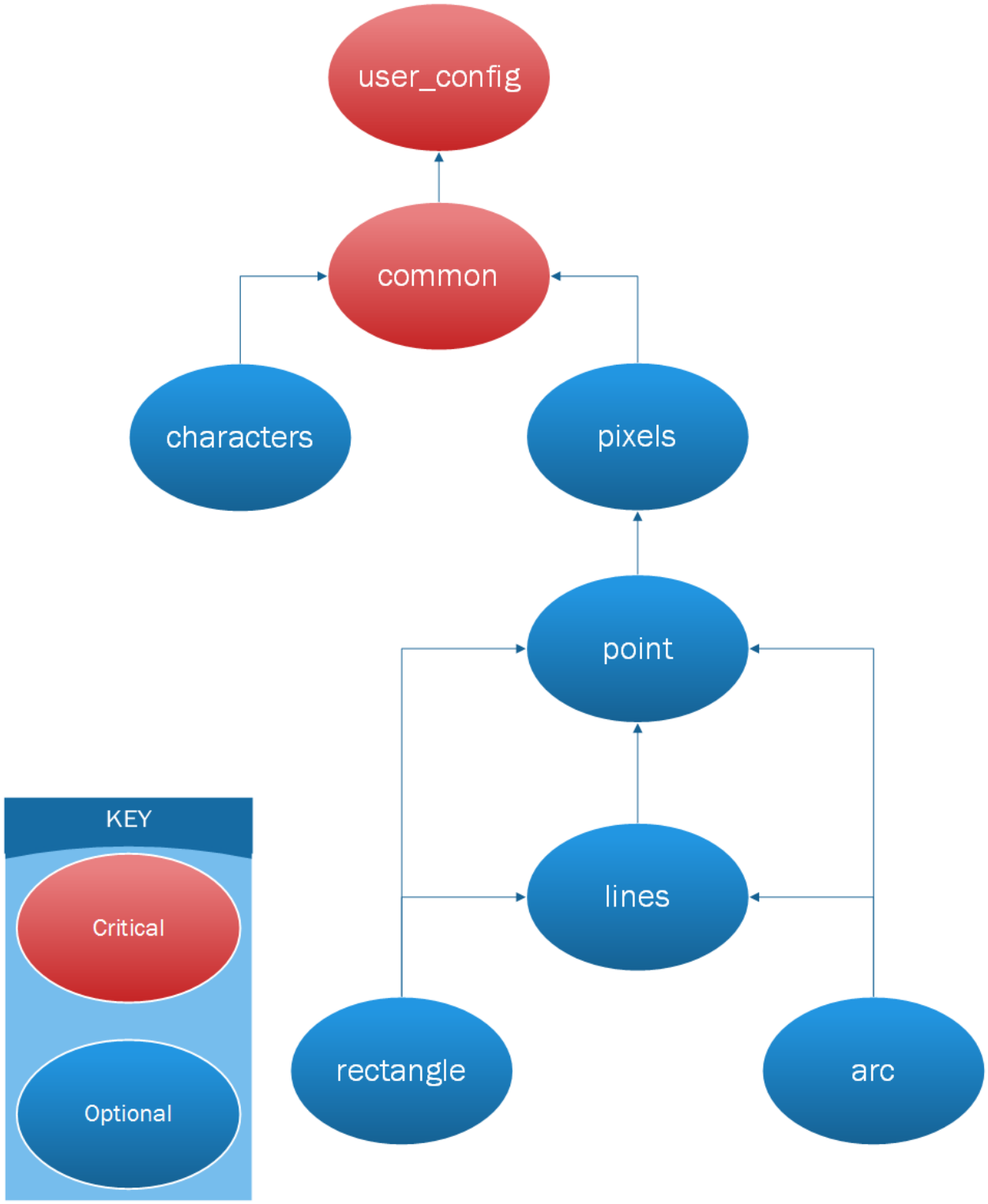

Figure 11: File Hierarchy

Using the Library Functions

The following code snippet demonstrates how a user should use this library. It is very important that the user sets the data direction register appropriately and remembers to print the buffer to the screen after writing to it.

```
#include <iom128.h>
#include <stdio.h>
#include "DOGM128_driver.h"

void main(void)
{
  DDRB = 0x37;     /* Set SS_bar, MOSI, SCK, reset, and A0 to outputs,
                    * leave rest alone for argument's sake.
                    */

  dog_init(DOG_INVERTED_DISPLAY,0x16);            /* Initialize Display */

  printf("Line %d: %c Hello World",1, 128);       /* Print Line */

  dog_draw_rectangle(0,8,127,63,0,'s');           /* Draw a rectangle */

  /* Print the "LATEX" Logo */
  dog_putchar_select(25, 20, 'L');
  dog_putchar_select(23, 26, 'A');
  dog_putchar_select(25, 32, 'T');
  dog_putchar_select(29, 38, 'E');
  dog_putchar_select(25, 44, 'X');

  dog_set_page(6);                                /* Set printing page */
  dog_set_column(20);                             /* Set printing column */
  printf("Hello Rectangle");                      /* Print "Hello Rectangle */

  dog_print_buffer();              /* IMPORTANT: Print buffer to screen */

  while(1);

}
```

Figure 12: Code Snippet and Resulting Display

Using the Library

To use this library, one only needs to add the source code to his or her project like any other file. Again, they may omit some of the files from the library (taking the aforementioned file hierarchy into consideration) by commenting out the respective header file form the master driver file, **DOGM128_driver.h**, and detaching the related files from the project.

Conclusion

The Electronic Arts DOGM128 should prove to be a very useful device and a suitable replacement to the 3-line displays currently being used. However, given the fact that the display has a much steeper learning curve, a library is essential to using it as an educational tool. Of course, this library does mostly everything a student in ESE 381 would possibly need, so letting them use this library directly may not be of much educational value. Some of the best coders in the world began their career not by writing software, but by hacking it instead. Perhaps it may be of educational value to "break" portions of this library and allow students to come up with their own solutions to the problems.

Links

1) http://www.cutedigi.com/development-tools/avr/cutedigi-atmega128a-system-board.html#googlebase
2) http://www.youtube.com/watch?v=PPo6sszdu78

References

1) Kraus, Oliver. (2011) dogm128: Library for the Dogm-Graphics-LCD [Computer program]. Available at http://code.google.com/p/dogm128 (Accessed October 2013)
2) Electronic Assembly, "DOGM Graphic Series" EA DOGM128-6 datasheet, [Revised Oct. 2009].