PrismFX: A Color Display for KBIDE

By

1.	Jihun	Lee	M.6/19	No. 3
2.	Rachen	Paophan	M.6/18	No. 13
3.	Phukawee	Kesmut	M.5/18	No. 10

Mattayom levels 4-6

School

Rayongwittayakom School

The Secondary Educational Service Area Office Rayong

Project Advisors:

1. Teacher Robert Norby
2. Teacher Sakpon Pongaksron

This report is a component of the Computer Project Competition, submitted for the English-Medium-Instruction Programmes (Central Region, Zone C) in the Academic Year 2025.

Table of Contents

ABSTRACT

We noticed there's no color display for KBIDE. So, the project is completely original and educational. The reason why we chose KBIDE is because it has a friendly interface with its drag-and-drop features and the perfect program for creating something and letting your creativity run free. It's also helpful for Thai students with the KBIDE's bilingual feature so Thai students can still be able to code. We used a 240x240 RGB color display using the ST7789 controller chip for the display. For the program, we used a C++ program to program each features and blocks for the plugin in KBIDE. Then we'll stress-test to see if there's a bug or a glitch. The features will be basic features like drawing shapes, displaying images etc. Whatever you want to create and display on the board, It's all up to your imagination and creativity. We used KidBright V1.2 for coding and KidBright V1.7 for testing. The reason why is because the V1.2 can tell us what went wrong in the coding so we could find the cause and fix it easily. After testing out all the features and stress-testing, The PrismFX plugin was working as it should. The plugin worked as it should have and the project is ready to send as an official professional product. Even if there are hiccups here and there, we managed to find the cause and fix it. Overall, the plugin is completed. It met our design goals and it is ready to be packaged as an official professional product.

ACKNOWLEDGEMENT

We, as a group, would like to express our gratitude to the **Chiang Mai Makers Club** for providing us the detailed and extremely useful information on how to make a KBIDE plugin. They were very essential in making this project. Without their help, this project would've been more difficult.

We would also like to thank our supervisors: **Teacher Robert Norby** for instructing us on how to make this project successful and guiding us through, **Teacher Alfredo Medina** for helping us with the 3D-Printing of the board stands and handling Github and **Teacher Sakpon Pongaksron** for all his help and many years he has been our teacher. He helped us with the photography and the display. Without their guidance, this project of ours wouldn't be possible.

Lastly, We would like to thank this competition for giving us the opportunity to show our talents and capabilities in programming and my colleagues who played an important role of helping each other as a team to make this project possible.

CHAPTER 1: INTRODUCTION

Background

We've been using the KidBright microcontroller board for some years now alongside the Mbits, microbits and Arduino boards. The boards and IDEs are fairly similar to each other, each with their own pros and cons. With KidBright's bilingual feature, It makes the platform more attractive to Thai students. Alongside its simple-to-use drag and drop features to get the basic knowledge of programming. It's easily recommended for beginners who want to get into the field of programming and start to be comfortable with programming.

But we did notice that there are no color displays for the KidBright IDE (KBIDE). So, that is the reason why we created this plugin to try out something new and see what the KBIDE is capable of, challenging ourselves along the way. PrismFX controls a 240x240 RGB color display using the ST7789 controller chip. This is a fairly common and inexpensive display so it's easily accessible to anyone to buy and use.

Purpose

We want to design and build a color display module for use with the KBIDE so that Thai students can code in Thai or English using the bilingual feature of KBIDE. Also, there are no other color displays for KBIDE. So the project is completely original and educational for Thai students to learn about programming and letting creativity run free. The display should have the following features: It can print text in multiple sizes and colors, Draw geometric shapes, pixels, and lines in selected colors, Plot graphs with legends in multiple colors, Can change orientation to any of the four sides and be able to display images on the panel. (240x240)

Why a color display and plugin for KBIDE?

- This is an original project. We haven't been able to find an existing color display with a plugin for the KBIDE.

- This is an educational project. The KidBright was designed by Thai engineers for Thai students. Extending the KidBright's capabilities which also extends the original purpose as well.

- This project can indirectly benefit Thailand by inspiring more students into appreciating STEM studies and careers.

CHAPTER 2: REVIEW OF RELATED LITERATURE

Design

Hardware:

- The display selected is an inexpensive (about ฿70 @ aliexpress) 240x240 TFT RGB display using 16 bit color.

- The display connects to the KidBright board using the SPI bus available on the KidBright board.

Source: AliExpress.com

- To be compatible with other SPI boards, the hardware interface must conform to the SPI bus interface standards defined in the NECTEC document. “See appendix A”

Source: AliExpress.com

- Circuit boards. We use EasyEDA to create a circuit board from this schematics. The software doesn't allow you to make electrical mistakes. You just have to place the parts and add the silk-screen texts and logos. We ordered the board from PCBWay in Shenzhen for about 500 baht for 10 boards including shipping.

Software:

- The plugin code is written in C++. The framework for the PrismFX plugin is the LCD_SPI plugin. The geometric and printing functions evolved from raspberry pi code from buydisplay.com. The plot blocks are original.
- The Blockly block descriptions and function generators are written using javascript. Blockly is a Google product with well-developed tools and documentation. https://developers.google.com/blockly/guides/create-custom-blocks/blockly-developer-tools
- The msg folder will have the en.js and th.js files defining all of the block definitions, labels, and tool tips in both Thai and English.
- Help files exist in an easily accessible website, preferably Github.

CHAPTER 3: RESEARCH METHODOLOGY

Implementation

Hardware:

No one in our group has any experience doing digital electronic design. But this isn't a big issue though. We did what any small company without in-house expertise does. We contracted the job out. Our advisors are engineers and provided a schematic and a functional breadboard. They did insist that we lay out the printed circuit board (PCB) ourselves. Using the EasyEDA web app, it is not difficult. The hardest part was finding parts in EasyEDA's libraries that physically matched the components on the breadboarded prototype. We could not use surface mount components without better soldering equipment and skills. The details of the circuit design are included as Appendix A. We had the 10 PCBs manufactured by PCBway in Shenzhen, China for about ฿500. A second version of the board was required because our team members changed and we wanted better artwork. PCBs are cheap and easy to lay out if all dimensions are <=100mm and they are 2 layers only.

Software:

Designing and developing a new plugin is a challenging project. Fortunately, the Chiang Mai Makers Club has published some very useful guides and examples.

The portable version 1.2 of the KBIDE makes development much easier.
https://github.com/MakerAsia/KBProIDE/releases/tag/v.1.2.0

https://www.slideshare.net/slideshow/kidbright-plugin-development-123284325/123284325

The PrismFX plugin files

C++ code (PrismFX.cpp & PrismFX.h)

The LCD-SPI plugin example was the starting point for our project. Like our project, the LCD_SPI board is a display device connected to the KidBright's SPI bus. It has 4 lines of 20 ASCII only characters with no color or graphics capability but still it is a working plugin.

https://gitlab.com/kidbright/kbide/-/tree/master/plugins/display/lcd_spi

LCD-SPI plugin source code

We also borrowed code extensively from buydisplay.com's sample code for the 240x240-spi display which we are using.

https://www.buydisplay.com/ips-1-54-inch-tft-lcd-display-module-240x240-spi-for-arduino-raspberry-pi

Blockly block definitions and code generators (blocks.js & generators.js)

blocks.js

Each block requires a javascript function to define its appearance and behavior. Most of the statements are obvious and common to all blocks. There is a common function at the top of the file, prismfx_init, that is called by almost all of the block functions. It is used to select the PrismFX board ID required on all blocks except the num2str and picker blocks.

The plugin blocks are written in javascript using Blockly. Blockly is a Google product specifically designed for creating code blocks for scratch, microbit, kidbright, and other block programming IDEs. Google has some good tutorials on how to get started with Blockly.

https://developers.google.com/blockly/guides/create-custom-blocks/blockly-developer-tools

After reading and trying these tutorials, it became clear how to create blocks for Kidbright easily. With the exception of the fully complete color picker, there are only three types of statements required to define the name and input parameters for the PrismFX blocks.

For the block title

```
this.appendDummyInput().appendField(Blockly.Msg.PRISMFX_Clear);
```

For a drop down list selection

```
this.appendDummyInput().appendField(Blockly.Msg.PRISMFX_Rotation)
                                        .appendField(new Blockly.FieldDropdown([
                                        [Blockly.Msg.PRISMFX_Clear_Top    , "0"],
                                        [Blockly.Msg.PRISMFX_Clear_Left   , "1"],
                                        [Blockly.Msg.PRISMFX_Clear_Bottom, "2"],
                                        [Blockly.Msg.PRISMFX_Clear_Right , "3"]
                                        ]), "Rotation");
```

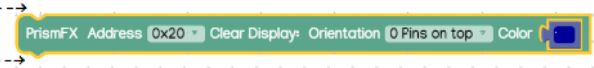
For a variable - setCheck can be {any, Number, String, Boolean, Array, other}

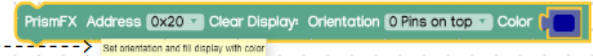
```
this.appendValueInput("COLOR").setCheck("Number").appendField(Blockly.Msg.PRISMFX_Color);
```

Relationships between a block, its definition in blocks.js and generators.js, and the C++ code produced by the generators.js.

The appearance and function of each block is defined by a javascript function in the file 'blocks.js'

```
Blockly.Blocks["prismfx.clear"] = {                   // This is the function for the 'Clear Display' block in the file 'blocks.js'
        init: function() {
                // init
                prismfx_init(this);                   // This is another function used by most of the blocks to add PrismFX and the Address drop down list field.
                this.appendDummyInput().appendField(Blockly.Msg.PRISMFX_Clear);   // The Dummy just adds a label, here the name of the block 'Clear Display:'
                this.appendDummyInput().appendField(Blockly.Msg.PRISMFX_Rotation) // This symbol is 'Orientation'
                                       .appendField(new Blockly.FieldDropdown([
                                       [Blockly.Msg.PRISMFX_Clear_Top   , "0"],
                                       [Blockly.Msg.PRISMFX_Clear_Left  , "1"],   // defines the drop down list options  ---------------->
                                       [Blockly.Msg.PRISMFX_Clear_Bottom, "2"],
                                       [Blockly.Msg.PRISMFX_Clear_Right , "3"]
                                       ]), "Rotation");
                this.appendValueInput("COLOR").setCheck("Number").appendField(Blockly.Msg.PRISMFX_Color);// setCheck ={any, Number, String, Boolean, Array, other}
                this.setInputsInline(true);                                  //                          ,---------->
                this.setPreviousStatement(true);                             // Puts the notch on the top.'
                this.setNextStatement(true);                                 // Puts the tab on the bottom.---------->
                this.setColour(160);                                         // Defines the hue of the block. range 0-360
                this.setTooltip(Blockly.Msg.PRISMFX_Clear_ToolTip);  // This tool tip is displayed when mouse over
                this.setHelpUrl("");                                         // not yet implemented              \---------->
};
```






```
Blockly.JavaScript['prismfx.clear'] = function(block) {
        var col = Blockly.JavaScript.valueToCode(block, 'COLOR', Blockly.JavaScript.ORDER_ASSIGNMENT) || '0';    // Extract the value of 'COLOR', if empty, use 0
        var rot = block.getFieldValue('Rotation');                                                               // Get the selected value for the drop down list
        return 'DEV_SPI.PrismFX('+block.getFieldValue('ADDRESS')+').clear('+rot+','+col+');\n';                  // ADDRESS is defined in the prismfx_init function
};
```

```
DEV_SPI.PrismFX(32).clear(0,153);          // This is the C++ code returned by the generators.js function that will be compiled and uploaded into to the Kidbright.
```

prismfx.clear	The function names must be the same in the blocks.js and the generators.js
Blockly.Msg.PRISMFX_Clear	These symbols are defined in English in msg\en.js and in Thai in msg\th.js. The definition is what is displayed in the block.
Rotation, COLOR, ADDRESS	These variables are defined and initialized in blocks.js. Their values are extracted in generators.js. They must be the same.

Typically, the six files, msg\th.js, msg\en.js, plugin.cpp, plugin.h, blocks.js, and generators.js, are all that are necessary for a Kidbright plugin. Since we had a color display with 240x240 pixels, we wanted to add the ability to display bitmap images. Since bitmaps require significant memory resources the limit is two bitmap images. The two bitmaps are named image1.h and image2.h. Instructions for converting a bitmap into an image#.h header files are in the doc folder..The default images are, image1.h has the RYW logo and image2.h has a smart watch screen. We later added more files making the new limit 4 image files.

The complete plugin code and definition files are in our public github account https://github.com/RywProject/PrismFX.

Software development, C++:

1. ST7789 graphic controller initialization – This was modified from the buydisplay code but exists in many other sources.

2. Clear Display block – (not difficult)
There is only one command used to send data to the ST7789, setAddressWindow. In setAddressWindow, the following is done:
send 0x2a set column address command, follow by the x start and end addresses, both 16 bit addresses
send 0x2b set row address command, followed by the y start and end addresses, both 16 bit addresses
send 0x2c memory write command, followed by the data (color required to fill the defined rectangle.

To plot a pixel at (x,y) - fill a 1x1 rectangle

send 0x2a, x high byte, x low byte, x high byte, x low byte

send 0x2b, y high byte, y low byte, y high byte, y low byte

send 0x2c, color high byte, color low byte

To clear the display, fill a 240x240 rectangle

send 0x2a, 0, 0, 0, 239 // x from 0 to 239 inclusive

send 0x2b, 0, 0, 0, 239 // y from 0 to 239 inclusive

send 0x2c, (color high byte, color low byte) 57,600 times

3. Next draw pixel, draw line (horizontal and vertical only) and draw rectangle were implemented. These were easy as they are all simple rectangles.

4. Then the Set Text Color and Print blocks, small (6x8) and medium (10x15) size fonts. A large font (12x20) was considered but not implemented. It would not be difficult but not a priority. These were somewhat challenging. There was some code from buydisplay.com that was useful to get started but we went way beyond with embedded newlines, wrapping, maintaining separate cursors for the different size fonts, and transparent printing. This was challenging, taking several weeks of effort. It works quite well.

5. Then the Num⟶Str block. We have seen many messy blocks for formatting numbers but implemented as different blocks, generally messy, and not aligning decimal points or units position. They also only formatted floating point numbers. We wanted a versatile block that could format numbers in multiple ways, fixed width, integer, hex, hex with leading zeroes, floating point, and exponential. We hoped that it could be done all in the javascript block, but because we allow variables and expressions as well as constants, this had to be implemented in the C++ code. This was all original code though not very challenging.

6. The plot functions were next. Init plot defined the variable, units, color, and min and max (or auto range if left empty) for one of up to three variables.
The Plot New Data adds a new point to the graph for up to three variables.
The ST7789 chip does not have a horizontal scroll function so we chose to display a cursor moving across the display, left to right. The point before the cursor is the latest point. The point after the cursor is the oldest point and about to be overwritten. This is completely original code and was somewhat challenging, especially the autoranging. It required several weeks of effort.

7. The screen orientation was implemented at this point. By default the pins are on the top of the display. Often it is more convenient to be able to have a display that is easy to read when in other orientations. This is done in the Clear Display block once in the initialization code. This was not obvious and the buydisplay code was referenced to see how it was implemented.

8. The RGB color definition block was added here. The color picker block was free and very convenient. It is part of Blockly's standard blocks. But it could not display all colors and cannot dynamically change colors. The RGB block allows for constants, variables, and expressions to be used for each of the three color intensity values.

9. We wanted to be able to display bitmaps. Much of this code was adapted from Bodmer's TFT_eSPI Arduino library. We originally allowed only 2 image files since they are huge, up to 115200 bytes for a full screen display. We later increased the limit to 4 image files.

10. Lastly, we finally finished oblique lines, triangles and circles.

CHAPTER 4: RESULTS AND DISCUSSION

Experiment Notes

A. General Testing: Making sure that the plugin works generally.
   - PrismFX is developed on the KidBright version 1.2.0 which is a development system and was imported to Kidbright version 1.7.0. The same version our school has. We also verified that the PrismFX plugin works on Kidbright versions 1.5.0 and 1.6.0.

B. Stress-Testing: Feeding it garbage either to find bugs, glitches or certain outcomes.
   - Changing the address of the board, It usually won't change anything on the board. As the board is normally 0x20.
   - Putting any block that's not numbers nor color in the space behind the color won't connect the pieces together.
   - Putting numbers into the color space will also change the color of the screen.

C. Bugs and Glitches Encountered
   - Using a variable named y1 in KBIDE V1.2 results in a compilation error. This bug resulted in several hours of lost effort. The only work around is to not use y1 as a variable. This error is not in any of the later versions.
   - The plugin, while working well in KBIDE V1.2, resulted in compilation errors in V1.7. After much frustration, it was discovered that KBIDE V1.7 files and folder names were case sensitive.
   - In the "Draw Image" block, changing the address to anything other than 0x20 will cause the color of the image to glitch. Not fixed. Difficult to replicate.
   - When printing, The new line character should advance the line by 1 and set the column to 1. This worked with the small font but failed with the medium font. Fixed
   - In the BMP280 plugin source code at line 20, The log level code hasn't been commented out yet. The log overwhelms the processor as it gathers a lot of information, making it slow to process data.

Block testing

For each block

For each parameter of the block:

- Leaving it empty = default
- If it's a string, we test by using short strings or long strings of text.
- If it's numbers, we test by inputting constants, variables or expressions.
- If it's a drop-down list, We tested each option of the list.

If you want to know more information about the data types of each block, range or all the defaults, you can refer to the User's Manual.

SPI chain testing

We're testing with an SPI_LCD, our original prototype breadboard and 2 PrismFX boards each with a different SPI address.

With multiple boards

- It initially worked but later failed. An analysis determined that the "Draw Image" block that was added after the initial test caused the failure. Included file guards were missing from all four image header files. After adding the guards, everything worked well.

with other SPI chain boards (we don't have any yet)

Results

- **Print Block**: The cursor for the medium font had the line and column cursor position switched.
- **Draw Image Block**: If the image doesn't fit on the screen, garbage was displayed. The fix was to verify that everything fits on the screen before printing. If not, nothing is displayed.

Conclusion

The project met our design goals and is ready to be packaged as a professional product. Some limitations of the FreeRTOS used by the KidBright limited the performance of the display. All blocks work as expected and the SPI chain (Multiple boards controlled by one KidBright) works as per NECTEC's design. The bilingual support feature of KidBright is well-implemented. This board has been tested on KBIDE versions 1.2, 1.5, 1.6, and 1.7. We only used the version 1.4 of the KidBright board because it's what we have in school. But the project could still be improved further. For example: KidBright for microBlock IDE, Making large texts, Add more features and blocks to play around with and improve on the old blocks and make it better.

Appendices

Appendix A: Hardware

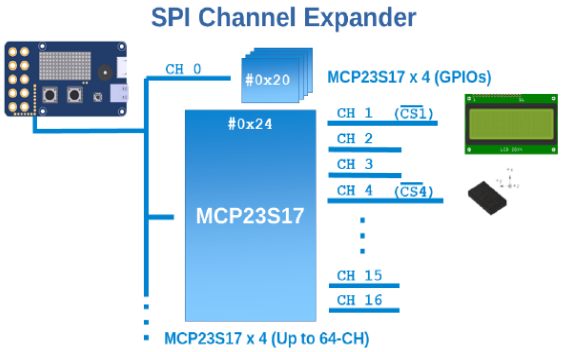
Hardware design

The 240x240 RGB display used in this project has an ST7789 controller chip. The ST7789 communicates with the Kidbright via the SPI (Small Peripheral Interface) bus. Typically, this bus requires 3 common wires (clock SCK, data from MPU MOSI, and data to MPU MISO) plus 1 wire per device for the chip select. The ST7789 cannot send any data so the MISO pin is not used.

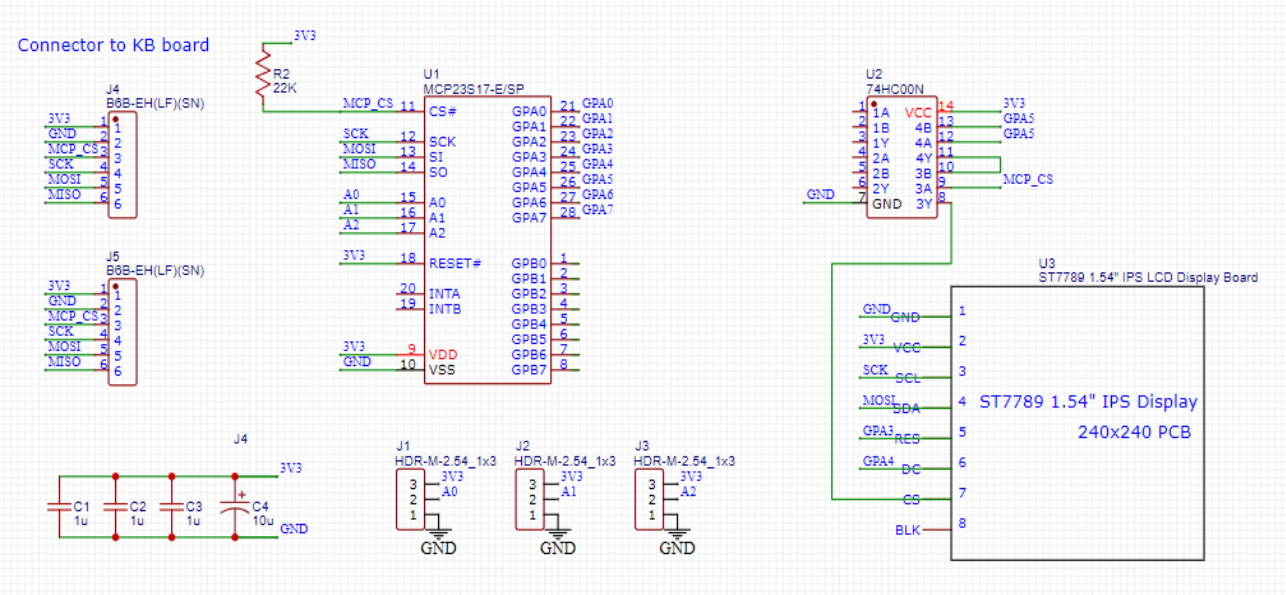
The expansion capabilities of the Kidbright board are quite ambitious. Using MCP23S17 GPIO expander chips, the Kidbright can theoretically control more than 64 SPI devices using only 1 output pin, typically GPIO0, to select a MCP23S17 which then controls the SPI devices' chip select pins. The ST7789 graphics controller requires 3 GPIO expander pins (chip select, reset, & command/data) which requires using an entire MCP23S17 chip to control the display. A maximum of 8 PrismFX boards can be connected to a Kidbright, if no other SPI devices are connected.

The document https://www.mdpi.com/2071-1050/14/21/14528 gives a detailed description of the design philosophy and implementation of the Kidbright. The following graphic is from Figure 8, Section 4.2 of this Kidbright document.

The implementation of the Kidbright SPI expansion board requirements are fairly easy due the features of the MCP23S17 chip. This is the design of the
PrismFX color graphic board for Kidbright.



SPI Channel Expander



PrismFX Schematics

Function of Components

J1, J2, & J3 allow the selection of the board address, from 0x20 to 0x27. Bit 5 is always high, bits 0-2 of the address are determined by jumpers J1, J2, & J3. The default address in the PrismFX blocks is 0x20 or 32 decimal. This corresponds to all jumpers connecting the MCP23S17 address pins to GND.

J4 & J5 are connectors that connect the PrismFX to the Kidbright or chain to another PrismFX board.

74HC00 is used to ensure that the ST7789 graphics controller is never selected when the MCP23S17 chip is being selected which could result in unintentionally sending data to the both chips in error.

ST7789 graphics controller's BLK (backlight) pin is not connected. This is an error. It should be connected to 3V3. The first order of displays had the BLK pin internally connected to power so it seemed unnecessary to connect it again. The second order of displays left the BLK pin open. It had to be externally jumpered to 3V3. A subsequent PCB redesign does connect the display board's BLK pin to 3V3.

The MCP23S17 pins GPA3, GPA4, & GPA5 control the ST7789 Reset, C/D, & CS pins. The MISO (master in, slave out) pin is connected to the MCP23S17 chip. Data can be read from the chip but the initial test software stub supplied with the board never reads the chip.

Appendix B: Useful Links

Useful Links for developing a plugin:

This is the source code for our project:

https://github.com/RywProject/PrismFX

Chiang Mai Makers Club provided many examples that were useful in our project:

https://github.com/orgs/cmmakerclub/repositories?page=6

This was an excellent source of information for making a plugin for KidBright. It was very essential for our project:

https://medium.com/chiang-mai-maker-club/วิธีสร้างปลั๊กอินสำหรับใช้งานในโปรแกรม-kb-ide-ตอนที่-2-f530ef5a0295

This is the only other SPI bus plugin that we used as an example:

https://gitlab.com/kidbright/kbide/-/tree/master/plugins/display/lcd_spi

This is the source for the display module and also for some of the C codes for the geometric shape drawing:

https://www.buydisplay.com/ips-1-54-inch-tft-lcd-display-module-240x240-spi-for-arduino-raspberry-pi

This is a useful tool for developing blocks for the plugin.

https://developers.google.com/blockly/guides/create-custom-blocks/blockly-developer-tools

KidBright 1.7 is nearly impossible to develop code in. V1.2 has some useful feedback for software development. We used V1.2 for all of our software development. We use V1.7 for testing:

https://github.com/MakerAsia/KBProIDE/releases/tag/v.1.2.0

https://google.github.io/blockly-samples/ (Source of Color Picker Block)

This is the history and design philosophy of the KidBright micro-controller:

https://www.proquest.com/docview/2769921125?sourcetype=Scholarly%20Journals.

This adds more information on making plugins for KidBright from the Chiang Mai Makers Club:

https://www.slideshare.net/slideshow/kidbright-plugin-development-123284325/123284325

This link is for our User's Manual. This is important if you want to see the details and basic information of each blocks:

https://docs.google.com/document/d/1pwndkqk9D7paSAlg7AxSDeCKJPtCby9sdMDgZs2rVRc/edit?tab=t.0

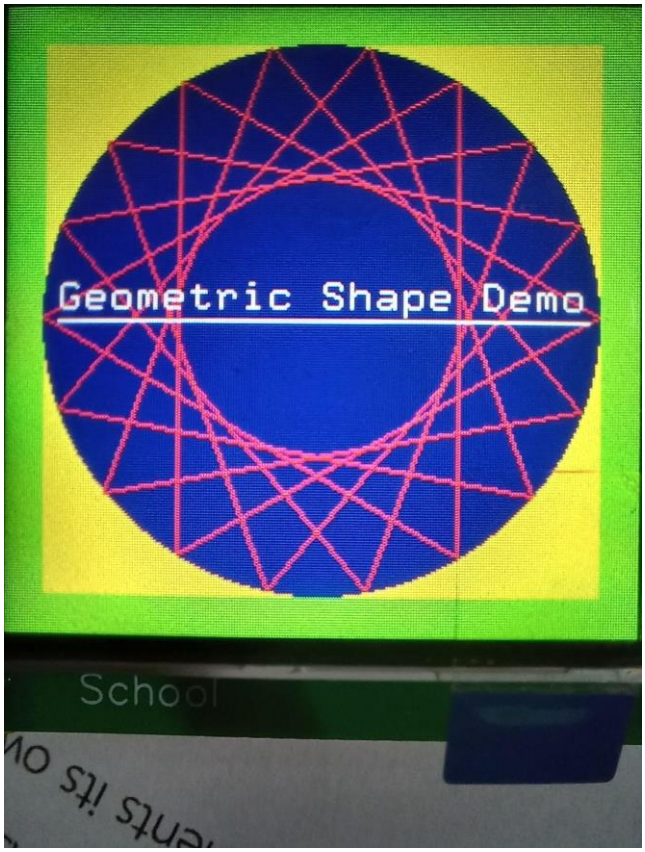
Arduino Library: TFT_eSPI created by Bodmer 2/12/16:

Some of the code for the image block was derived from this library.

Appendix C: Demos

Demo's from PrismFX plugin

Each of the blocks have been used in a different way, which creates a variety of uses. We decided to create a demo to demonstrate the uses of each block from our plugin. These are some of the demos.



Geometric Shape Demo:

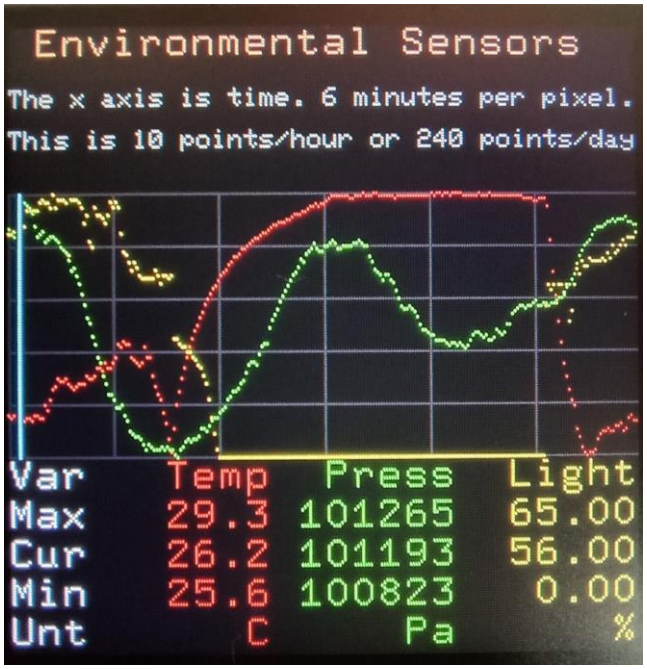
This demo started with a green square covering the whole board as the most back layer. Then, a smaller yellow square was placed in front of the green square, making the green square resemble a thick green outline. After that, the blue circle was inscribed in the yellow square as the frame. Then, multiple red-outlined triangles were placed inside the triangle and were slightly rotated each from each other to create a pattern where a circle was formed amongst the rotating triangles. Lastly, This text is an example of transparent printing where the text color is the same as the text background color. This prints the foreground only leaving the background unchanged.. Then, the line was added beneath the text as an underline.

Environment Graph:

This demo plots the temperature by using a BMP280 sensor and the light sensor is built in KidBright

The x-axis is time, 360 seconds (6 minutes) per pixel. The display is 240x240 pixels so exactly 1 day of history is displayed.

The graph start time is about 10:15. At 16:30 (above the T in Temp) the lights and



A/C were turned off. After this the light slowly declines while the temperature increases. At about 7:00 (above the i in Light) the next day the light and A/C are turned on. Air pressure varies as it does. It is clear that the A/C cannot keep up with students coming and going, often leaving the door open.

Trig Function Plot:

This is the header of the plot. It's 64 pixels high. You can fit 8 lines of small texts or 4 lines of Medium text.

The graph is 101 pixels high. The light cyan bar is called the "cursor". It moves from the left to the right, looping.

It's constantly being fed with new data every few seconds. The data on the left of the cursor is the new one and on the right is the old one that's about to be replaced. The top of the graph is the maximum value and the bottom part is the minimum value. Grid lines are 40 pixels horizontally and 20 pixels vertically.

Flag Theme Image:

The Thai flag is drawn by using five fill-in colored rectangles. Red and white rectangles are around the same size meanwhile the blue rectangle is twice the size of the red or white rectangles.

The two pictures below are from the “Draw Image” block. The two images are different because of the drop down list of images. They are image3 and image4. They are placed next to each other by changing the other image’s line of position.

Color Saturation Levels:

Color saturations levels uses this “ST7789 Graphics chip that uses 16 bit color” (5 bits Red, 6 bits Green, 5 Bits Blue)

The Color Saturation levels that display in this screen is calculated by checking if the variable “x” is less than 240 which is the screen size, while x < 240 the 3 colors which is Red, Green, and Blue will be increased by 16, and also to make the color saturation change by the position we make each color “X1" and “X2” increase along the “x” value

Appendix D: Development Pictures