Research 1 Technical Report

Jayden Lombardi
Champlain College
jayden.lombardi@mymail.champlain.edu

December 9, 2024

1 Overview

The goal of this project is to make a simple "Hello World" base project program for the Nintendo 3DS using devkitPro and devkitARM tools, writing in C. This program will display text on screen based on button input, and will also have a toggleable strobe mode. This project will provide insight into base-level DS programming, and allow for users to expand upon and create any program or game they would like to load onto their DS.

1.1 Objectives

1. Display "Hello [Button]" based on the button pressed; A, B, X, Y
2. Move a pixel on the top screen that corresponds with the D-Pad
3. Reset everything on the screen when left bumper is pressed

With Adequate Time:

1. Display "Hello [Button]" At the current location of the pixel
2. Move a pixel on the bottom screen that corresponds with the Joystick
3. Left bumper resets everything to a black screen, pixel is now white
4. Right bumper resets everything to a white screen, pixel is now black

1.2 Application Functionality:

1. Touch and dragging on touchscreen will draw a white line
2. Pressing SELECT button will toggle Strobe mode on and off - other commands can still be used during strobe mode
3. Pressing A button will load an image of link on the bottom screen
4. Pressing B button will clear everything on screen
5. Pressing START button will quit the program

1.3 Why?

Why are we researching this? Why are we coding to a 3DS?
Primarily, it is an exploration of the Nintendo 3DS, it's architecture and devkitPro tools. It is a look into how the Nintendo 3DS handles instructions, memory, display and user input/output. This project will provide valuable insight into how to link code, and how the C language translates user input into interactive outputs on screen. Learning how to store button states and access said data for conditional branches. It will serve as the base research for a programmer looking to get their hands dirty in Assembly, Linking, devkitPro and learn a bit more about gaming consoles and how they run.

2 Implementation

2.1 Environment

- ARM assembly for the 3DS - ARM instructions + memory management
- Visual Studios 2022 for writing Assembly and C code
- MSYS2 to compile and debug code
- Citra to debug and test on laptop before switching to 3ds
- Luma3DS and homebrew to homebrew my 3DS
- My personal jail-broken 3DS for testing and debugging
- 3DS Internal registers and memory
- SD Card and SD card reader to transfer files to 3DS from Laptop

2.2 Tools

1. Initialization
    - Set up 3DS internal registers to prepare for button inputs
    - Zero out and clear out any previously held data
2. User Input Handling
    - Involves checking a specific memory address
    - Each button as well as the touch screen corresponds to a memory address, the value changes when the button is pressed
3. Outputting to Display
    - Output is done after input is handled
    - Have an output Hello function and an output Character function
    - Make sure output to the correct buffer
4. Reset Functionality
    - Clear display memory
    - Setting background color to white

3 Challenges and Solutions

1 = Problem
2 = Solution

3.1 Macro

- Understanding Nintendo 3DS Architecture
    1. One major problem will definitely be understanding the Nintendo 3DS architecture. I have the smallest bit of experience with architectures in general, and this is all brand new for me. The 3DS uses ARM which I am not too familiar with, but will gain. The 3DS has specific memory-mapped input and output set ups, which I will need to become accustomed to.
    2. I will need to research and study both ARM documentation and the unique set up of registers and memory in the 3DS. Carefully dealing and altering the 3DS memory, testing and debugging each function separately to verify each part of the code works.

- Limited Debugging

    1. The 3DS does not have a built in debugger to help me debug the problems with my code. To make matters worse, most common debuggers are not compatible with a 3DS.
    2. Using an emulation system on my laptop (Citra and MSYS2 in tandem) to observe how the memory and registers are changing when the code is stepped through.

- Setting up and Linking devkitPro and devkitARM

    1. The problem I struggled with the most in this entire project was correctly setting up and linking the devkitPro tools, and the 3ds library to my project. This step took about 4 days in total of consistent work on the project. I was attempting to manually link and create a project through devkitPro. This meant manually setting paths, manually creating a Visual Studio project and directory, and attempting to link the library to it from devkitARM. This process threw numerous errors at me across multiple days, all of which I researched and preserved through.
    2. The solution in the end was a single link online to someone else with the same exact problem (see Micro 5). There was a single response to the forum post, and it explained the devkitARM comes with pre-built pre-linked example projects. I felt very stupid not looking for these examples from the start but chose to persevere anyway. I made a copy of the most basic example and saw where the fault lied in my previous linking. After finally having a working environment to code and compile in after 4 days I was ecstatic.

3.2 Micro

- Managing Registers Efficiently and Correctly

    1. There are limited General purpose registers, and it is a good idea to use the intended registers for their intended purposes. Excessive loading to one register will slow down the program.
    2. Once again, I am not too familiar with ARM or the ARM registers, and do not know the most efficient way to go about using them. Planning on and researching the best way to use registers will be of great use to my project. As well as searching my code for excessive loading and storing areas.

- Memory Management for Display

    1. More specifically the 3DS memory management for display. The last graphics programming class I took, I struggled a bit in grasping a few of the concepts. The 3DS has it's own system for managing display, that I will have to address correctly.
    2. Refreshing my brain on graphics programming will help get a good general knowledge of display management. Researching specifically 3DS management on the Nintendo website will help me get accustomed to specifically displaying things to the 3DS screen.

- Displaying an Image on Screen

    1. Another big but specific problem I faced was loading an image to the screen. I spent about 2-3 days on this before I had to call it quits and continue with the production, it was something I really wanted but was almost unable to complete. I was attempting to load a png on the bottom screen of the 3ds, but the 3ds does not accept png's so I had to switch the format of the image (see Micro 4). I tested multiple different for loops, as well as multiple different images to load to the bottom screen. Multiple times the images would load with incorrect colors, incorrect format, and compressed heavily either vertically or horizontally. In the end, the final image loaded is the furthest I was able to achieve.
    2. In the end I never was able to fix it 100 percent. The image of Link loads but it is compressed and rotated 90 degrees counterclockwise. I ruled out a problem with the colors through the RGBtoRGB565 function, and removed the offset so it would just print in the top left corner. After this the solution was simply to try numerous different for-loop iterations, and find the

one that worked best. In the end I was left with the double for-loop (see Code Explanations). I believe the function to also be a bit specific to this image, as others do not load correctly. In the future, I would really like to sort this out, I believe with more time it is within my ability to do so.

- Putting Image Into correct format

  1. As stated in the previous problem, finding and setting the correct format for the image being loaded was just as (if not more) important as the code used to load to the buffer. I needed to research the 3DS architecture to find and understand what type of file, size, color format, endian system, and depth the console uses when loading images. If any of these values were slightly off I would not know the source of the errors, from the image format or the code? I went through a wide range of file types (.bin, .sgi, .txt) before settling on .RAW. After settling on this it still took about a day to find an application that would format the image correctly.
  2. The solution in the end was installing and making use of MagickImage (see ImgMagick). Through this program I was able to open the command prompt terminal and convert any png I wish, while also setting every bit of information myself within the terminal. This allowed me to personally create and format the RAW file so it would accurately fit and match the 3DS architecture. A very useful tool that I will most definitely use in the future. I ran the following line within command prompt to turn the image from a png to a correctly formatted RAW.
     `magick "C:\devkitPro\assets\link.png"`
     `-resize 320x240 -depth 8 -endian LSB RGB:link.raw`

- Cannot Find 3dsxcrt0.o

  1. After spending about 4 days of attempting to correctly link devkit tools and the libctru 3DS library, I was blessed to fall upon This link. Similar to the post I was getting the continuous error that the application was unable to find the file `"3dsx_crt0.o"` even though that file did indeed exist. Both me and the forum poster thought it was due to the fact that devkitPro was looking in the wrong directory for the file, so we both copied and pasted the crt0.o file into the correct directory. After doing so, I received an error that every function within the 3DS libctru library used VFP register arguments while my code did not. I spent about half a day trying to fix this, attempting to change paths and formatting of all of my current files. I was completely unsuccessful and went to the SmartSPACE programming tutor for help.
  2. After about an hour in the SmartSPACE scratching my head I decided to look up the exact issue of the crt0.o file. After finally finding the forum post, I saw that the problem was the fact that the tutorial I was following was almost out of date, and 3DS homebrew code has greatly improved since it first started. The solution was found from a response underneath the forum, explaining that I should use the official devkitPro examples to start base code. This indeed worked! And although I spent 4 days trying to link the application I believe I learned a lot more about how devkitPro works in general, and it aided me in solving problems down the road.

4 Code Explanations

```c
//function sets the color of the top and bottom screens to a given color
//<u32 color>       color to switch to (32 bit unsigned int)
void setScreenColor(u32 color)
{
    //get the frame buffer of the top screen
    u32* framebuffer = (u32*)gfxGetFramebuffer(GFX_TOP, GFX_LEFT, NULL, NULL);  //top screen
    //loop thru every pixel in the buffer and set color
    //top resolution: 800x240
    for (int i = 0; i < 800 * 240; i++)
    {
        framebuffer[i] = color;
    }

    //get the frame buffer of the bottom screen
    framebuffer = (u32*)gfxGetFramebuffer(GFX_BOTTOM, GFX_LEFT, NULL, NULL);  //bottom screen
    //loop thru every pixel in the buffer and set color
    //bottom resolution: 320x240
    for (int i = 0; i < 320 * 240; i++)
    {
        framebuffer[i] = color;
    }
}
```

- Function used to set the color of both screens to the input 'color'
- First the top screen frame buffer is pulled
- Then each pixel on the screen is set to the color (resolution 800x240)
- Second the bottom screen frame buffer is pulled
- Then each pixel on the screen is set to the color (resolution 320x240)

```c
//function prints given text to the center of the given screen
//<screen>      print console of where the words should be output
//<text>        pointer to string of words to be output
void printCenteredText(PrintConsole* screen, const char* text)
{
    //calculate length of the string
    int textLength = strlen(text);

    //calculate center of screen based off console width and height
    int startX = (screen->consoleWidth - textLength) / 2;
    int startY = (screen->consoleHeight - 1) / 2;

    //set the cursor position to the calculated points
    screen->cursorX = startX;
    screen->cursorY = startY;

    //print text here
    printf("%s", text);
}
```

- Functioned used to print text in the center of a given screen
- First string length is calculated
- Then center of the screen is calculated based on console width and height, and how long the text is
- Cursor is set to the position and text is printed

```
//function to continuously draw a square where the users is touching
//<x>               x and y have been flopped (for whatever reason this is what worked correctly)
//<y>               y = touch.PX and x = touch.PY
//<size>            length of the square to be drawn
//<u32 color>       color of the square (32 unsigned int)
void drawSquare(int y, int x, int size, u32 color)
{
    //get the frame buffer of the bottom screen
    //only the bottom screen has touchscreen capabilities
    u32* frameBuffer = (u32*)gfxGetFramebuffer(GFX_BOTTOM, GFX_LEFT, NULL, NULL);

    //loop through each pixel in the square
    //dx horizontal offset, dy vertical offset
    for (int dx = 0; dx < size; dx++)
    {
        for (int dy = 0; dy < size; dy++)
        {
            //calculate index of pixel to be colored
            //(x + dx) = horizontal pos
            //(y + dy) = vertical pos
            //multiply by 240 to account for framerbuffers layout
            //divide by 2 to account for 16-bit color format
            frameBuffer[(((x + dx) + (y + dy) * 240) / 2)] = color;
        }
    }
}
```

- Function used to continuously draw a square on the touchscreen
- Square is drawn at the given x, y value with the given size and color
- First bottom screen is pulled (only screen with touchscreen capabilities)
- Each pixel on the screen is looped through
- Calculate where the square should be currently drawn and add it to the framebuffer

```
//function to convert RGB values to RGB565
//<u8 r>    red component of color, (0-255), 8-bit unsigned int
//<u8 g>    green component of color, (0-255), 8-bit unsigned int
//<u8 b>0   blue component of color, (0-255), 8-bit unsigned int
//<Returns> returns 16-bit converted rgb565 value
u32 rgbToRGB565(u8 r, u8 g, u8 b) {
    //convert each color channel to appropriate number of bits
    u32 red = (r >> 3) & 0x1F;      // 5 bits for red   / right shift 3 and mask (11111)
    u32 green = (g >> 2) & 0x3F;    // 6 bits for green / right shift 2 and mask (111111)
    u32 blue = (b >> 3) & 0x1F;     // 5 bits for blue  / right shift 3 and mask (11111)

    // combine RGB into single 16-bit rgb565 value
    u32 color565 = (red << 11) | (green << 5) | blue;
    return color565;// | 0x00000000;
}
```

- Function to switch from RGB to RGB565 given r,g,b values
- 5 bits for red = right shift 3
- 6 bits for green = right shift 2
- 5 bits for blue = right shift 3
- Then combine into single 16 bit value and return

```
//function to load images onto 3ds screen
//<filepath>        path to image to load (string/char)
//<framebuffer>     pointer to frame to write to
//<width>           image to load width in pixels, int
//<height>          image to load height in pixels, int
void loadImage(const char* filePath, u32* framebuffer, int width, int height)
```

- Signature of the function used to load a given image
- Filepath is path to the image on the sd card
- Framebuffer is pointer to which screen to write to
- Width and Height of the given image in pixels

```
//read the image data from the file, into memory
fread(imageData, 1, imageSize, imageFile);
//close file after reading
fclose(imageFile);

//loop through every single pixel in the image
for (int y = 0; y < height; y++)
{
    for (int x = 0; x < width; x++)
    {
        //for each pixel extract individual RGB components
        //image data is stored in row-major order with 3 bytes per pixel
        u8 r = imageData[(y * width + x) * 3 + 0]; // red component    byte1
        u8 g = imageData[(y * width + x) * 3 + 1]; // green component  byte2
        u8 b = imageData[(y * width + x) * 3 + 2]; // blue component   byte3

        //old debugging statements
        //printf("r: %u, g: %u, b: %u\n", r, g, b);
        //printf("Color: %x\n", rgbToRGB565(r, g, b));

        //convert from RGB to RGB565 (16-bit color format)
        //store converted color in framebuffer at position
        //& 0xFFFF to ensure we are only using the lower 16 bits (RGB565 format)
        framebuffer[y * 320 + x] = rgbToRGB565(r, g, b) & 0xFFFF;

    }
}

//free the allocated memory after processing is complete
free(imageData);
```

- Logic for loading the image to the screen
- First open the image data in fread, read from file into memory
- Close the image file after reading to not corrupt
- Then loop through each pixel in the image
- For each pixel extract the u8 r g and b values (3 bytes per pixel)
- Use the RGBtoRGB565 to convert to 16 bit color format
- Store converted color in framebuffer at pixel, only using the lower bits

```
//TOUCH SCREEN FUNCTIONALITY
//first read touch input
hidTouchRead(&touch);

//if touched position is within bounds...
if ((touch.px >= 0) && (touch.px <= 320) && (touch.py >= 0) && (touch.py <= 240))
{
    //draw a square at the position (square draws a line if user holds down)
    //invert the Y-axis for proper drawing
    int correctedY = 240 - touch.py;
    drawSquare(touch.px, correctedY, 10, RGB565(255, 255, 255));
}
```

- Logic within the main loop for touchscreen capabilities
- First read the touch input through hidTouchRead();
- If the touched position is within bounds of the lower screen draw the square at the given position
- correctedY is used to invert the y-axis so it draw properly.

5 Future Plans

Ideas and projects I am going to pursue over winter break as I really enjoyed the creation of this project:

1. Display text at the current location of the pixel
2. Linking DPAD controls to movement
3. Linking Joystick controls to movement
4. Left bumper resets everything to a black screen, Right bumper resets to a white screen
5. Get any audio file to play
6. Get the 3 Dimensional aspect of the top screen working
7. Change the colors of the LEDs on the bottom of the 3DS
8. Correctly load an Image
9. Basic animations/textures

6 Reflection

All i all yada yadfa yada Things to improve upon:

1. Rah
2. Rah2

7 Sources

- DevKitPro Resources:
    1. DevKit Installation
    2. DevKit Pacman Inst.
    3. Setting up DevkitPro environment - Windows
    4. DevKit 3ds Library Github
    5. No Such File or Directory FIX - DevkitPro
    6. HomeBrew YouTube Tutorial
- MSYS2 Installation:
    1. Official MSYS2 Website Install
    2. Install MSYS2 Environment
- Citra Resources:
    1. Citra Official Install
    2. Citra Gen Info
    3. I have previously used Citra to emulate Pokemon games online, and knew somewhat how the emulator worked!
- ImageMagick Resources:
    1. ImageMagick Official Install
    2. ImageMagick Commands
    3. Stack Overflow ImageMagick