ECEN 3320-002
Assembly Language Programming
Lab Assignment # 3.4

Cameron Biniamow
University of Nebraska-Lincoln
Department of Electrical and Computer Engineering
Peter Kiewit Institute
Due: 12/11/2020

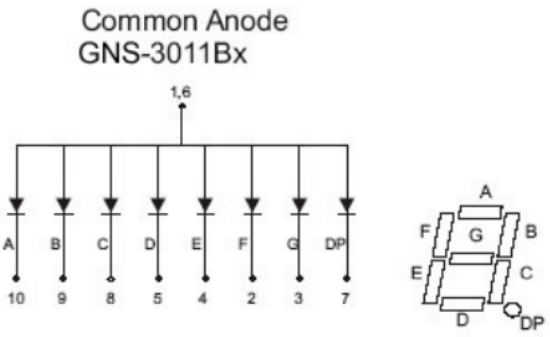
Summary

Lab 3.4, Configuring GPIO with STM32CubeIDE, introduces the configuration of the GPIO pins on the STM32F103C8 development board using the STM32CubeIDE software. Through implementing a 7-segment display driver, the STM32F103C8 is configured to continuously display each hexadecimal value 0 – F for 1 second. Once Lab 3.4 is completed, a greater understanding of pin configuration within STM32CubeIDE, writing in C language, and controlling a 7-segment display are gained.

Background

The objective of Lab 3.4 is to create a 7-segment display driver using STM32CubeIDE to upload to the STM32F103C8. The 7-segment display driver will be implemented in such a way that stores the values needed to display each hexadecimal value 0 – F and call upon them at any point in time. For this lab, each hexadecimal value 0 – F will be displayed on the 7-segment for 1 second, starting with 0. The program will be set to increment to the next hexadecimal value after the 1 second has past until ‘F’ is displayed. Once ‘F’ is displayed for 1 second, the program will loop back around to displaying ‘0’. This loop will be continuous and upon restart, ‘0’ will be displayed.

The GNS-3011 BH-11 7-segment display is a common anode display that is capable of displaying numbers and letters using the 7-segment format as well as displaying a decimal point. Due to the 7-segment display being common anode, in order to turn ON a segment of the display, a LOW signal must be sent to the respective pin. The 7-segment display is powered through pins 1 and 6 from the STM32F103C8 3.3V pin, which allows for direct connections from the pins of the STM32F103C8 to the 7-segment display without causing an overcurrent issue.



In order to write a value or character to the 7-segment display, the segments needing illuminated have a LOW signal written to them and the segments that are to be OFF need a HIGH signal written to them. As can be seen below, the table shows the state of each pin for displaying each value and the respective hexadecimal value that will be used to write to the 7-segment display.

Displayed Value	PA7 dp	PA6 g	PA5 f	PA4 e	PA3 d	PA2 c	PA1 b	PA0 a	Hex Value
0	1	1	0	0	0	0	0	0	0xC0
1	1	1	1	1	1	0	0	1	0xF9
2	1	0	1	0	0	1	0	0	0xA4
3	1	0	1	1	0	0	0	0	0xB0
4	1	0	0	1	1	0	0	1	0x99
5	1	0	0	1	0	0	1	0	0x92
6	1	0	0	0	0	0	1	0	0x82
7	1	1	1	1	1	0	0	0	0xF8
8	1	0	0	0	0	0	0	0	0x80
9	1	0	0	1	1	0	0	0	0x98
A	1	0	0	0	1	0	0	0	0x88
B	1	0	0	0	0	0	1	1	0x83
C	1	1	0	0	0	1	1	0	0xC6
D	1	0	1	0	0	0	0	1	0xA1
E	1	0	0	0	0	1	1	0	0x86
F	1	0	0	0	1	1	1	0	0x8E

Procedure

Creating a 7-segment driver for the STM32F103C8 using STM32CubeIDE begins by creating a new STM32 project. A new project can also be created through the following steps:

- File >> New >> STM32 Project

Following the creation of a new project, the Target Selection window appears. Within the Target Selection window, “STM32F103C8” is searched for in the Part Number box. On the right half of the Target Selection window, the STM32F103C8 controller is selected and the Next button is

clicked. The new project is titled, and the Finish button is selected.

Once the new project is created, controller configuration may begin through the Device Configuration window. The device is initially configured to allow for debugging by performing the following steps:

1. Pinout & Configuration >> System Core >> SYS
2. Debug >> Serial Wire
3. Timebase Source >> SysTick

In order to configure the GPIO pins as outputs, the desired pins are clicked on while viewing the .ioc file and selecting "GPIO_Output". Since the 7-segment display requires 8 pins for functioning, the pins PA0 – PA7 are selected. While any pin that is not already configured can be selected, pins PA0 – PA7 are selected due to their placement on the STM32F103C8 board. Once all 8 pins have been configured as outputs, setup is confirmed through the following steps:

1. System Core >> GPIO
2. Ensure that the selected pins are listed
3. Confirm that the selected pins are OUTPUTS and they are outputting a LOW signal

Following the configuration of the desired pins, the source code is generated through the following:

- Project >> Generate Code

Now that the configuration code is generated by the STM32CubeIDE, the 7-segment display code is entered in the main function. The setup code will be placed between the following lines of code:

```
MX_GPIO_Init();
	/* USER CODE BEGIN 2 */

	/***ENTER CODE HERE***/

	/* USER CODE END 2 */
```

The looped code will be placed between the following lines of code:

```
while (1)
{
	/* USER CODE END WHILE */

	/* USER CODE BEGIN 3 */

	/***ENTER CODE HERE***/

}
	/* USER CODE END 3 */
```

Within the setup code, the hex values are defined for each of the values that will be displayed on the 7-segment display. Since the 7-segment display used in this lab is a common anode, a LOW signal is needed to turn ON the respective section of the display. The values in the table shown in the Background of this report are used and are assigned to their respective variable, which is named with the following format:

```
Val0, Val1, Val2, ..., ValF
```

After the hexadecimal values have been assigned, the setup code for the 7-segment display is completed.

Within the infinite loop portion of code, the hexadecimal values are sent to the 7-segment display through PORTA with the following line of code:

```
GPIOA->ODR = Val_;
```

Where _ is the desired value to be written to the display. Since the objective is to continuously display each value 0 – F in numerical order with a 1 second delay in between values, the following line of code is used after the value is sent to PORTA.

```
HAL_Delay(1000);
```

As a whole, the infinite loop operates by writing the value for 0 to the 7-segment display, holds the value for 1 second then writes the value for 1 and holds the value for 1 second. The remaining code follows the same format where each value is written to the 7-segment display and then followed by a 1 second delay. Once the value for F is written to the 7-segment display and holds for 1 second, a loop occurs back to writing 0 to the display and holding for 1 second. The 7-segment display will now continuously display each value 0 – F for 1 second each and will loop back to 0 after F is displayed.

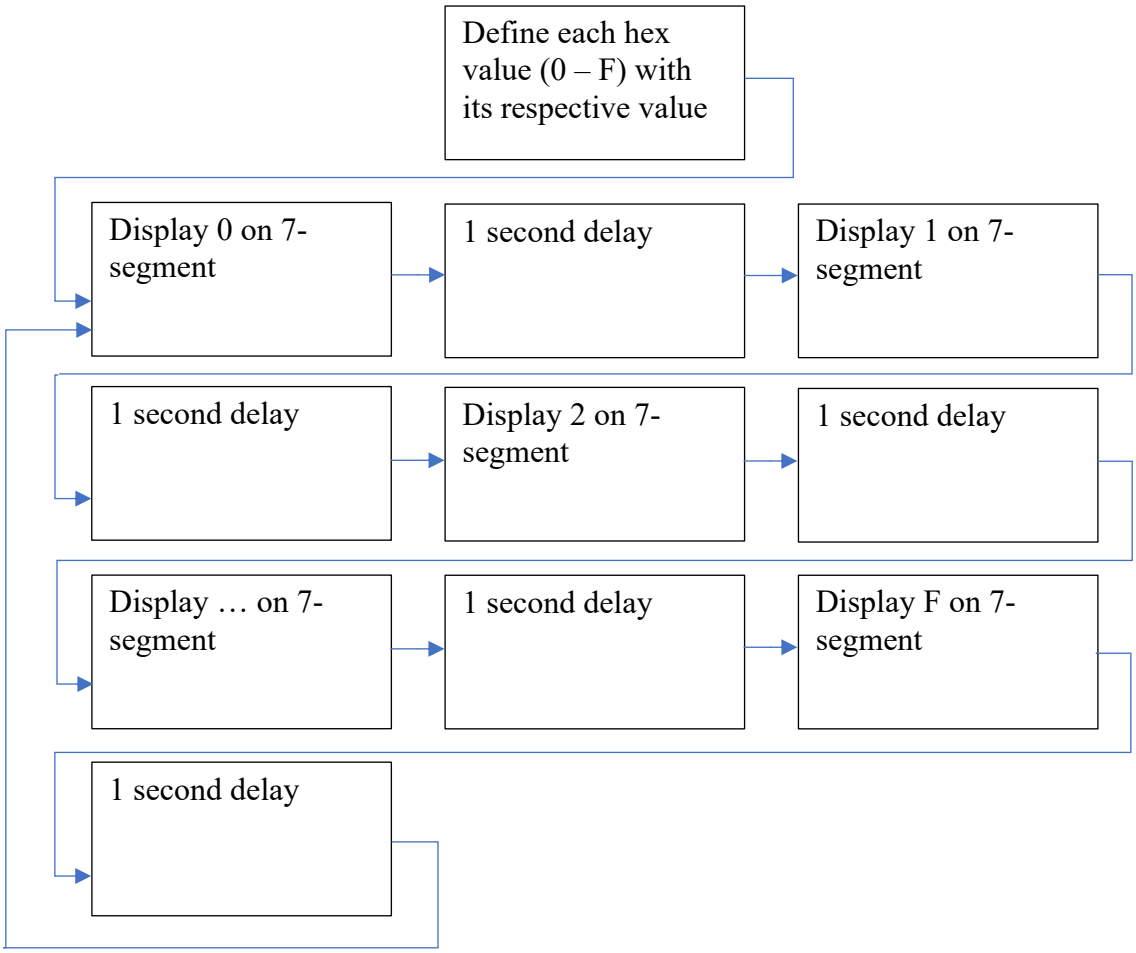
Results

After following the procedure as it is explained, the 7-segment display driver was implemented in STM32CubeIDE and uploaded to the STM32F103C8 for testing. Upon initial testing, the 7-segment display was displaying the inverse of the desired values for 1 second before changing to the inverse of the next value. It was quickly determined that the defined values for each hexadecimal value needing displayed, was inputted as if the 7-segment display needed a HIGH signal to illuminate a segment. After finding the complement of the defined values, the correct values replaced the original defined values, and the program was once again uploaded to the STM32F103C8 for testing. It was quickly determined that the 7-segment display driver operated as expected and without error as each hexadecimal value, starting with 0, was

displayed for 1 second before incrementing to the next hexadecimal value. Upon the 7-segment display displaying ‘F’ for 1 second, the program looped back around to displaying ‘0’ and remained in a continuous loop.

Answers to Posted Questions

1. Explain what the lab code does.
    - The lab code operates by initializing each hexadecimal value and its respective value needed for it to display on the 7-segment display. Once each value is defined, each value is sent to PORTAs ODR register in the infinite loop. In between each value that is being sent to PORTA, a 1 second delay occurs to allow for the value to be displayed and held on the 7-segment for 1 second. Since the values are sent to PORTA inside of the infinite loop, the values 0 – F only need to be written once as the program will loop back around continuously.
2. Create a block diagram for the Lab 3.4 code.



Conclusion

Lab 3.4, the final in a four-part series of labs that revolve around the configuration of the STM32F103C8 and its abilities, introduced the GPIO configuration using STM32CubeIDE. The GPIO configuration was implemented through the creation of a 7-segment display driver where each hexadecimal value 0 – F can be displayed. Specifically, for Lab 3.4 the driver was created and used to display each value, starting with '0', for 1 second and then incrementing to the next hexadecimal value until 'F' is displayed. Once 'F' is displayed, the program loops back around to displaying '0' and does so continuously. Following the design and implementation of the 7-segment display driver, the program was uploaded to the STM32F103C8 and tested for accuracy. It was determined that through testing, the 7-segment display driver worked as expected and operated without error. The key takeaways from this lab were the ability to choose and configure pins on the STM32F103C8, navigating through STM32CubeIDE and its GPIO features, and configuration with a 7-segment display.

Appendix

Figure I: STM32F103C8 7-Segment Display C Source Code

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
```

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
  int Val0 = 0xC0;
  int Val1 = 0xF9;
  int Val2 = 0xA4;
  int Val3 = 0xB0;
  int Val4 = 0x99;
  int Val5 = 0x92;
  int Val6 = 0x82;
  int Val7 = 0xF8;
  int Val8 = 0x80;
  int Val9 = 0x98;
  int ValA = 0x88;
  int ValB = 0x83;
  int ValC = 0xC6;
  int ValD = 0xA1;
  int ValE = 0x86;
  int ValF = 0x8E;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        GPIOA->ODR = Val0;   // 0
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val1;   // 1
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val2;   // 2
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val3;   // 3
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val4;   // 4
        HAL_Delay(1000);     // 1 SECOND DELAY
```

```
        GPIOA->ODR = Val5;   // 5
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val6;   // 6
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val7;   // 7
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val8;   // 8
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = Val9;   // 9
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = ValA;   // A
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = ValB;   // B
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = ValC;   // C
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = ValD;   // D
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = ValE;   // E
        HAL_Delay(1000);     // 1 SECOND DELAY

        GPIOA->ODR = ValF;   // F
        HAL_Delay(1000);     // 1 SECOND DELAY
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                          |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7,
GPIO_PIN_RESET);

  /*Configure GPIO pins : PA0 PA1 PA2 PA3
                           PA4 PA5 PA6 PA7 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                          |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Figure II: 7-Segment Displaying 0

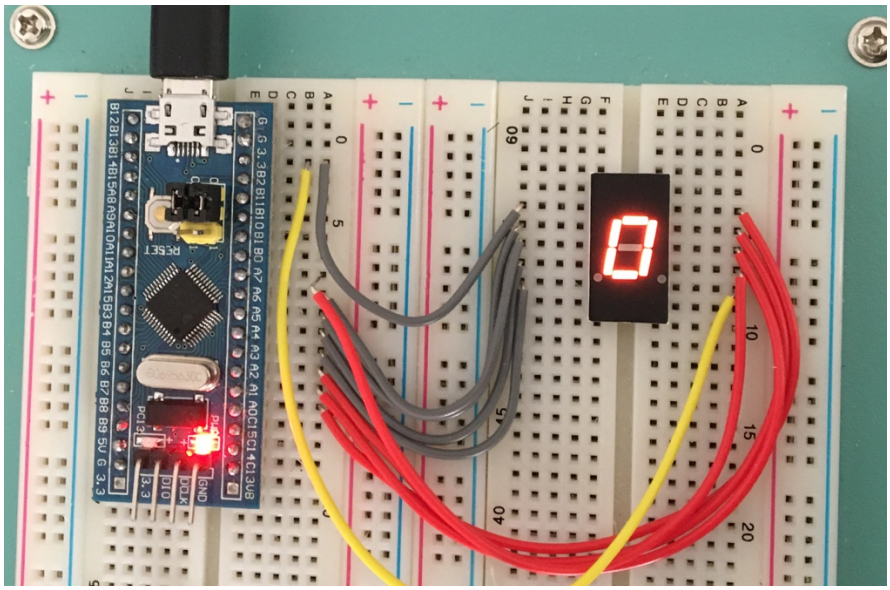


Figure III: 7-Segment Displaying 1

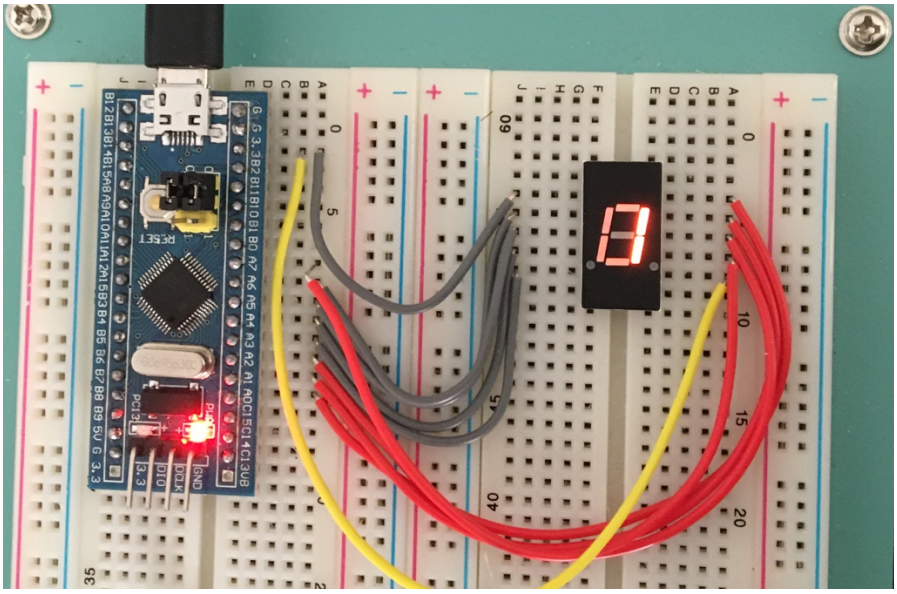


Figure IV: 7-Segment Displaying 2

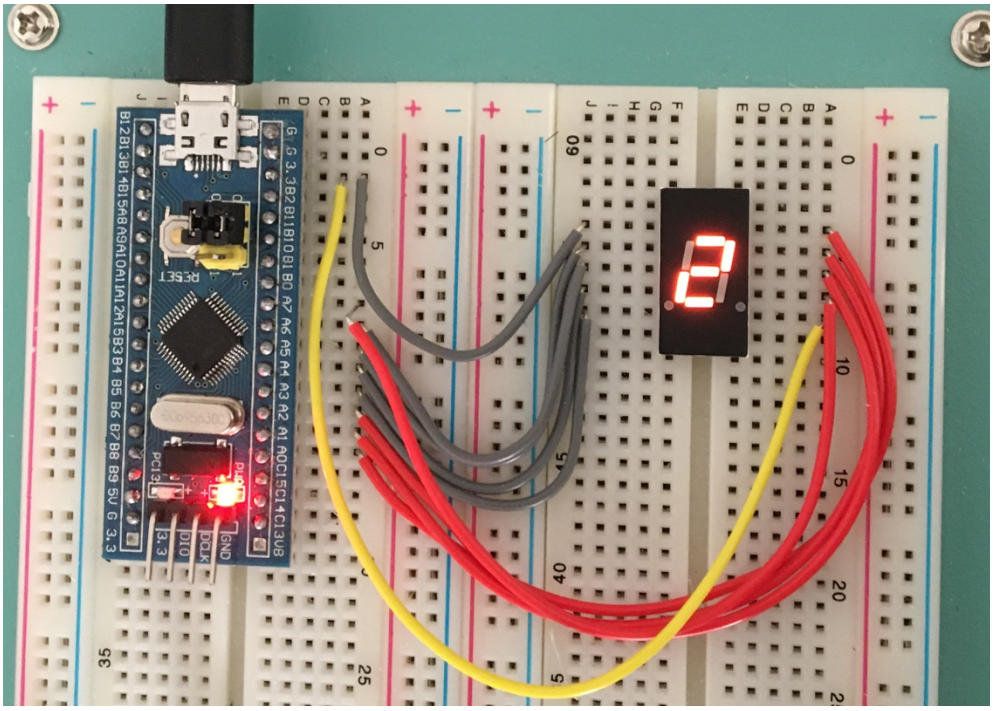


Figure V: 7-Segment Displaying 3

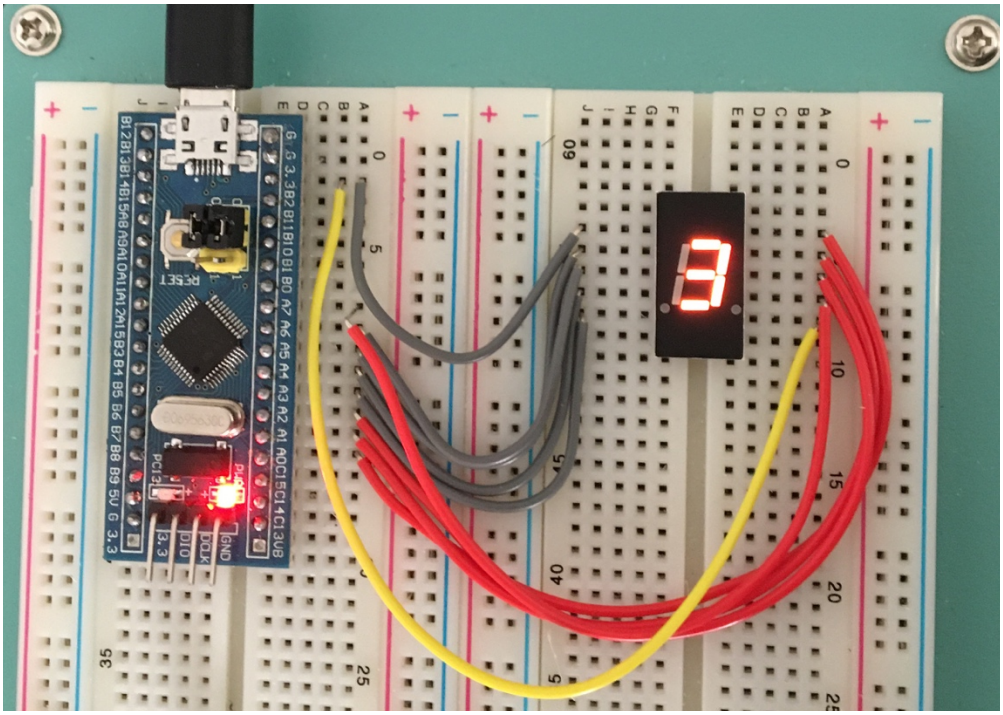


Figure VI: 7-Segment Displaying 4

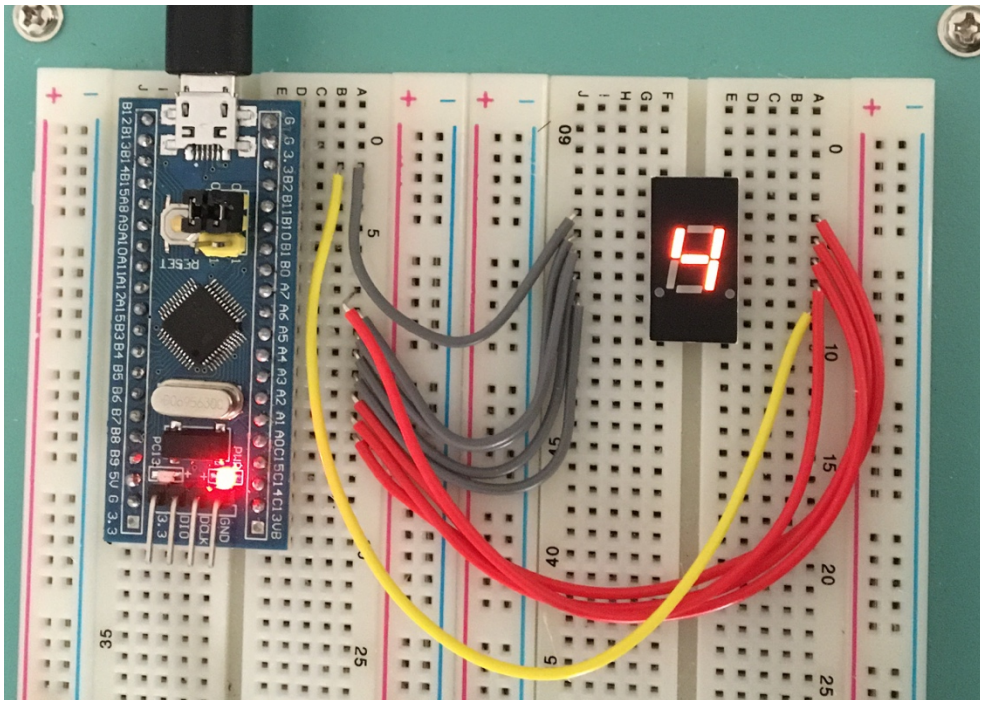


Figure VII: 7-Segment Displaying 5

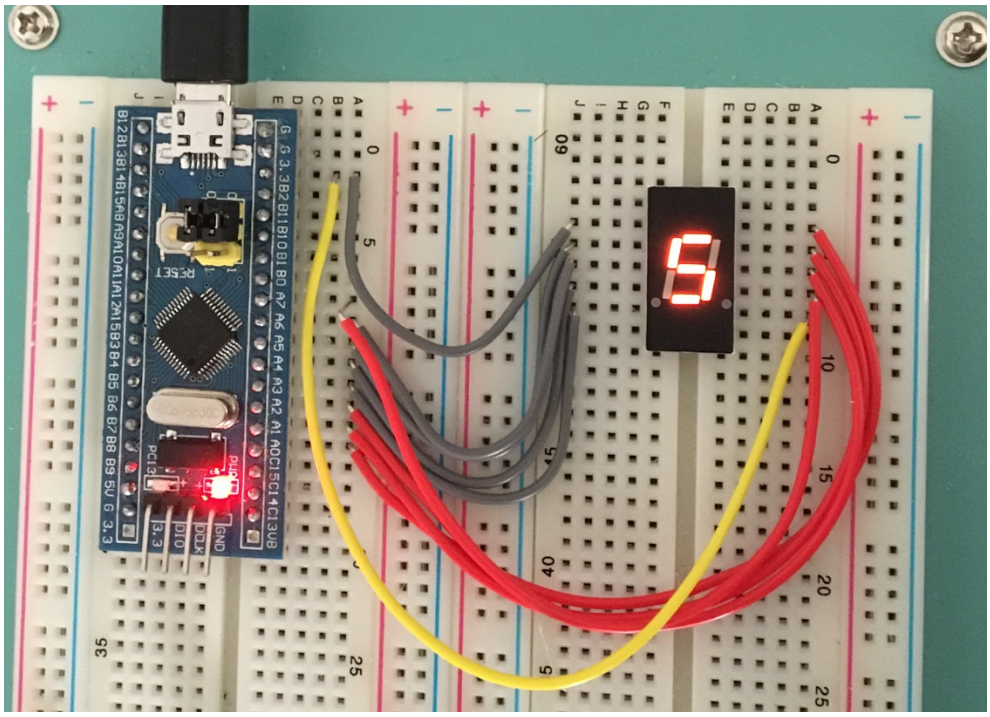


Figure VIII: 7-Segment Displaying 6

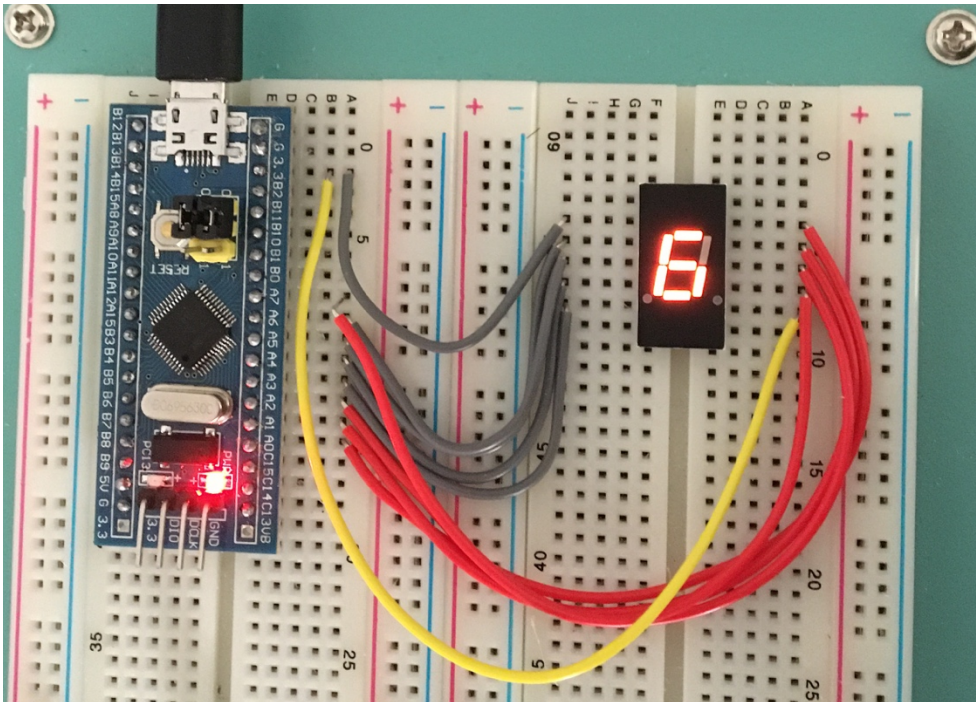


Figure IX: 7-Segment Displaying 7

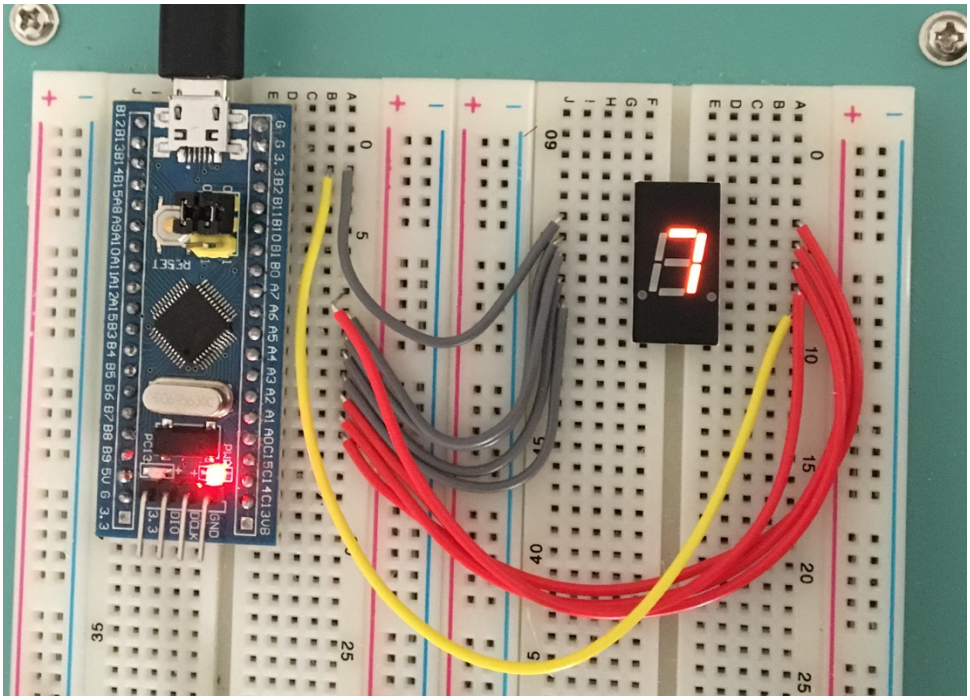


Figure X: 7-Segment Displaying 8

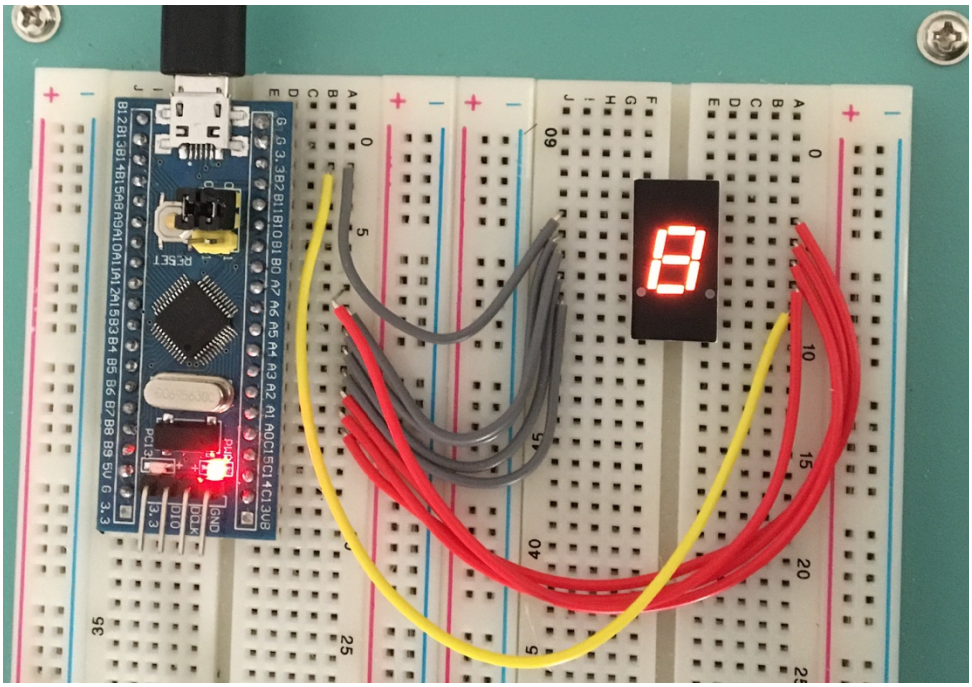


Figure XI: 7-Segment Displaying 9

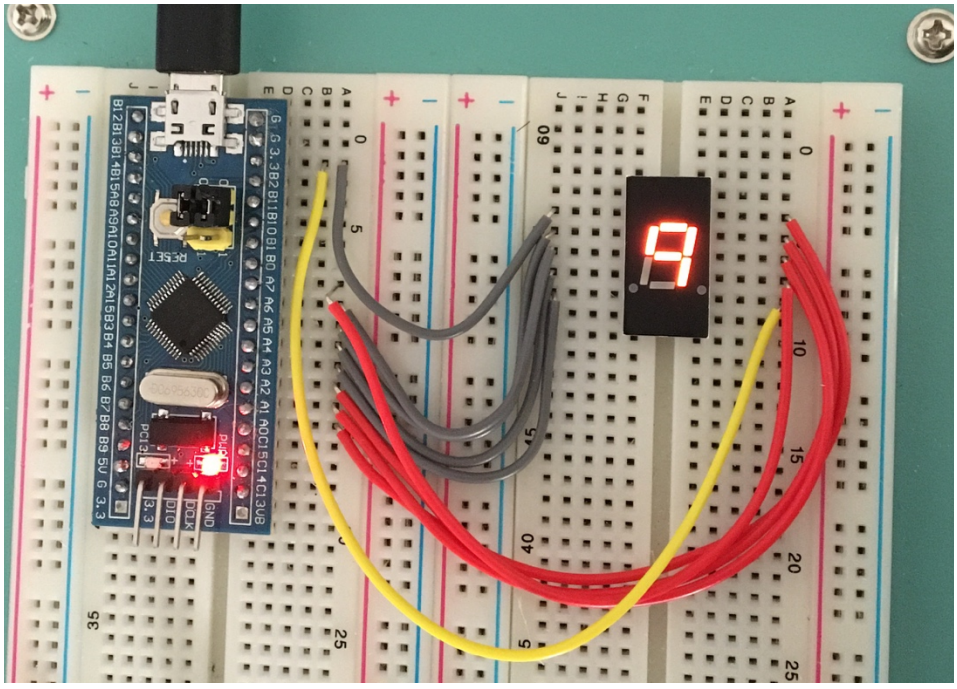


Figure XII: 7-Segment Displaying A

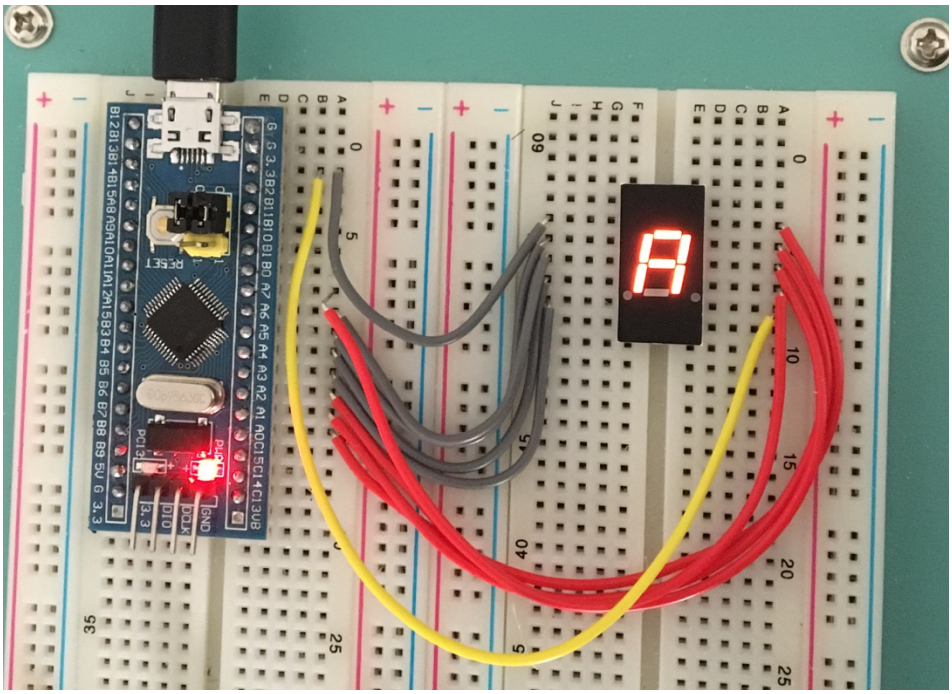


Figure XIII: 7-Segment Displaying B

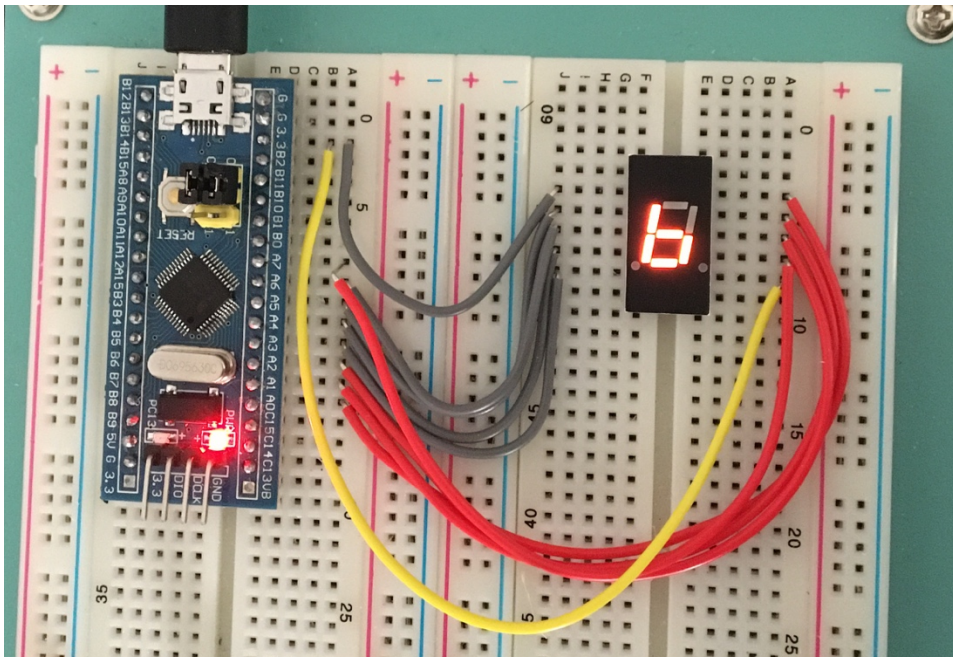


Figure XIV: 7-Segment Displaying C

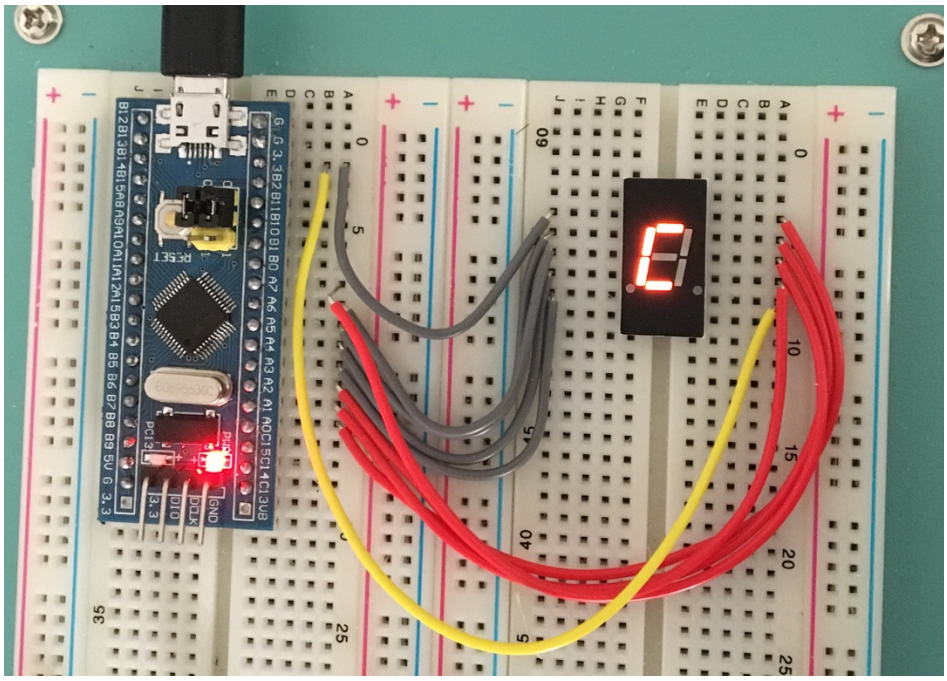


Figure XV: 7-Segment Displaying D

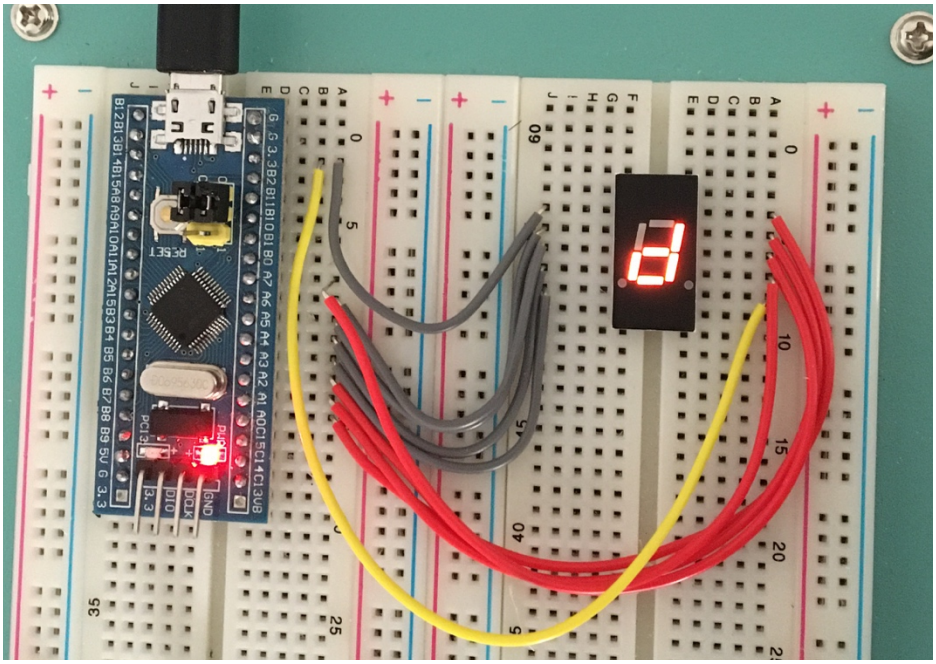


Figure XVI: 7-Segment Displaying E

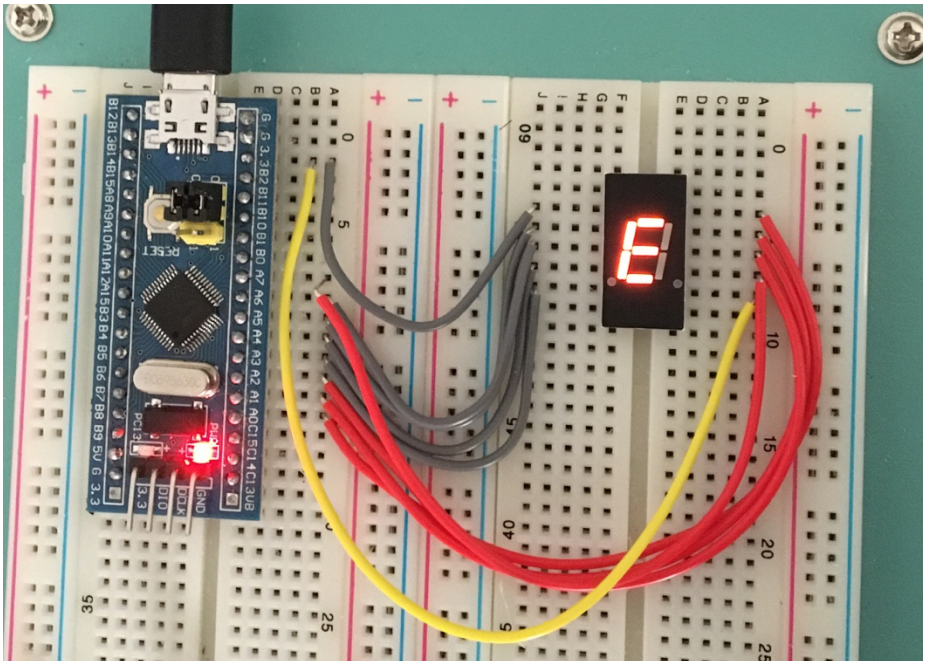


Figure XVII: 7-Segment Displaying F