



DesignNews

IoT Device Prototyping with STMicroelectronics Nucleo Development Boards

Day 3: Prototyping with STM32 IoT Middleware

Sponsored by



Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Attendee Chat’ by maximizing the chat widget in your dock.

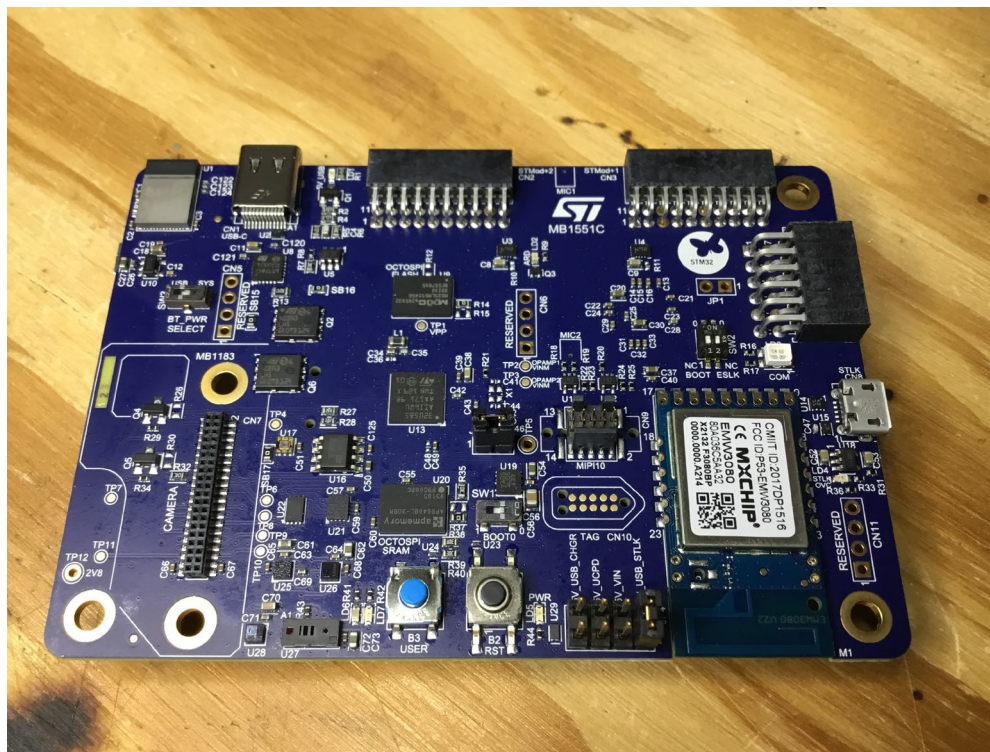


Fred Eady

Visit 'Lecturer Profile' in your console for more details.

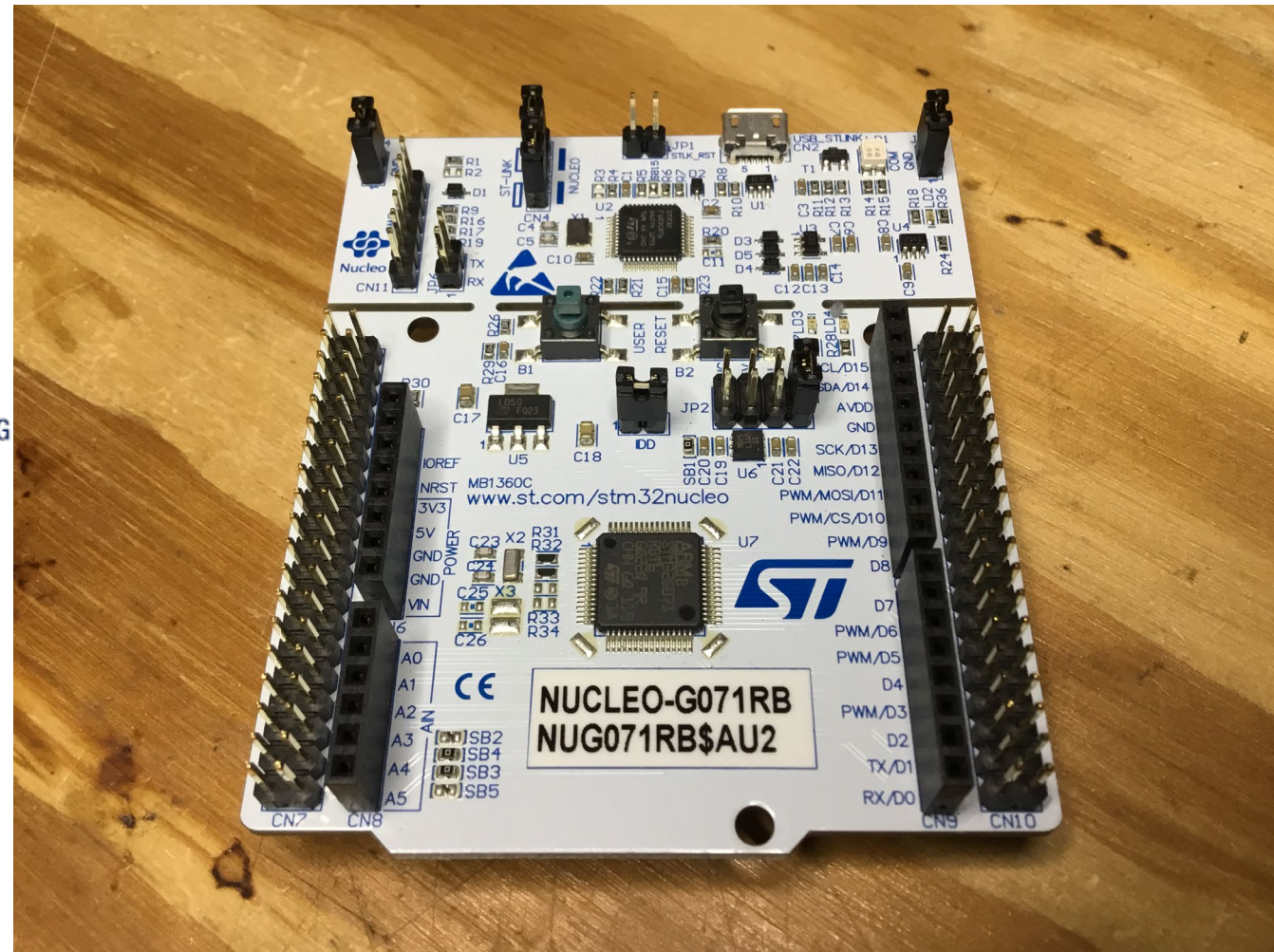
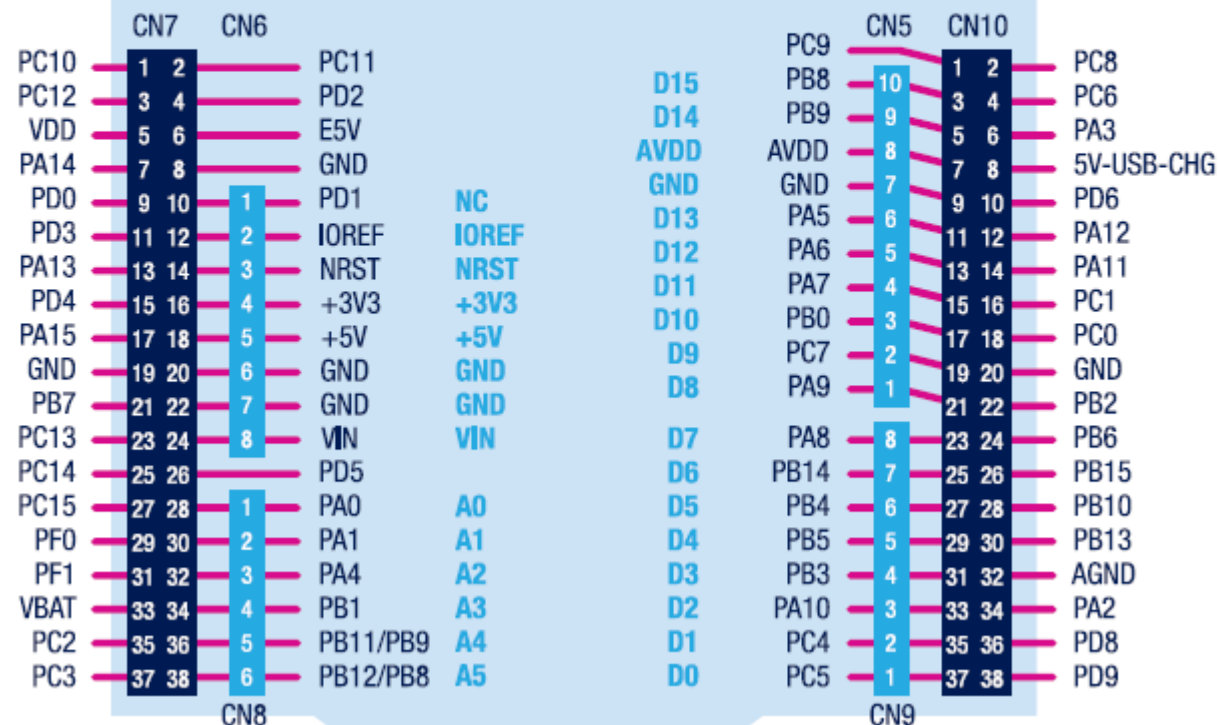
AGENDA

- **NUCLEO-G071RB click EXPANSION (a.k.a. the "SLED")**
- **NUCLEO-U575ZI-Q click ARDUINO EXPANSION**
- **FatFs Driver**
- **STM32U585 Nx_TCP_Echo_Client**



NUCLEO-G071RB click EXPANSION

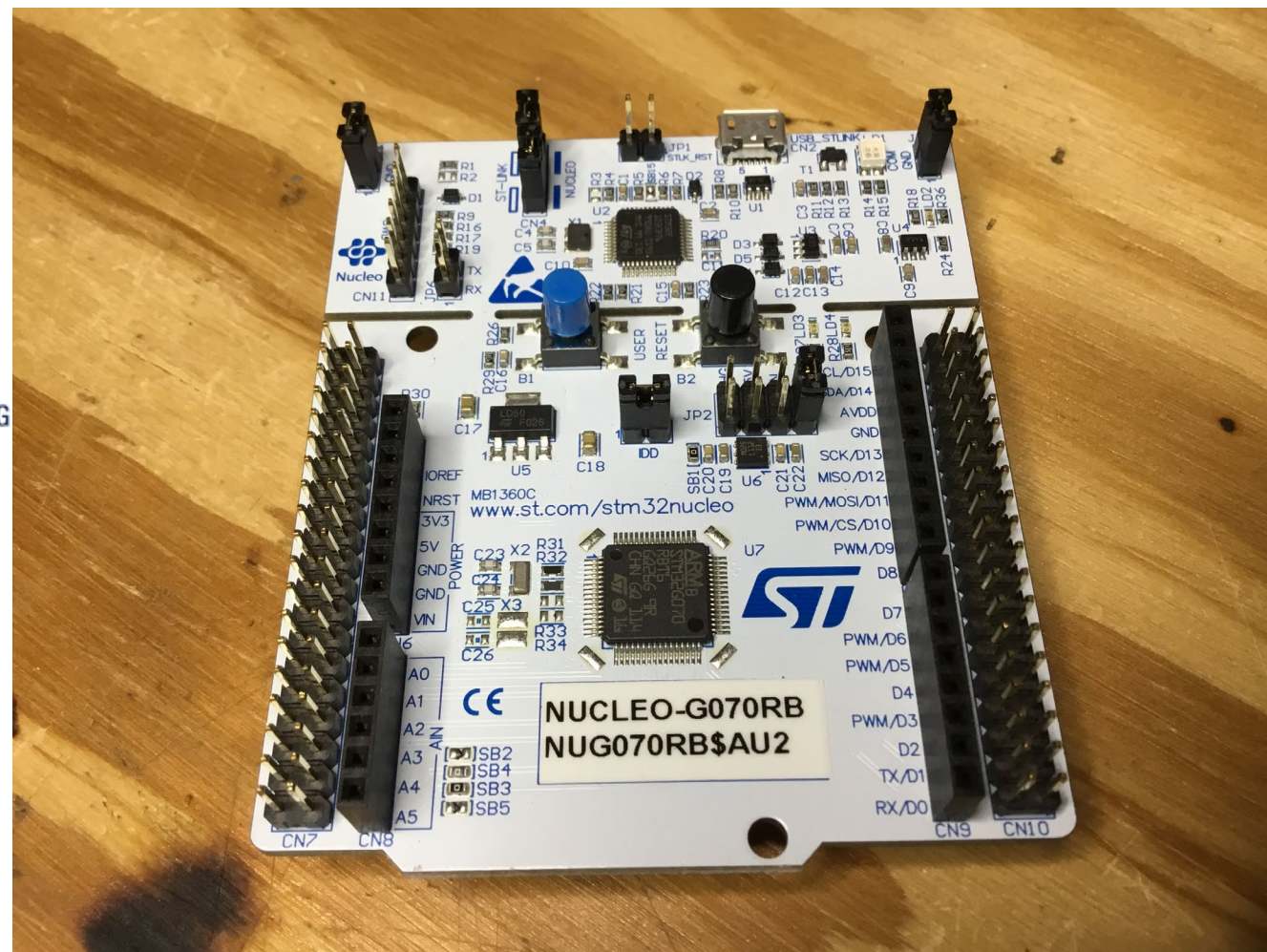
NUCLEO-G071RB



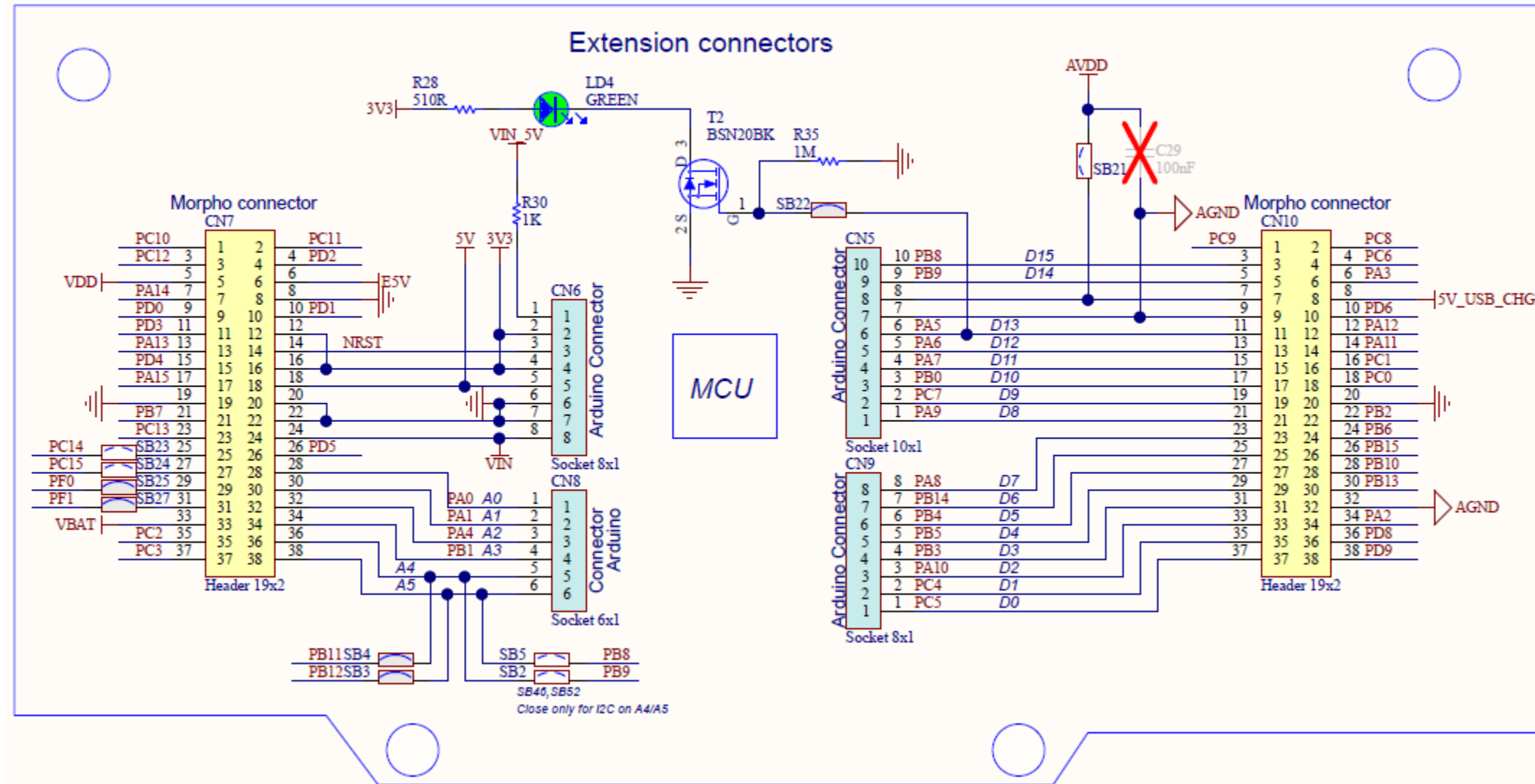
NUCLEO-G071RB click EXPANSION

NUCLEO-G070RB

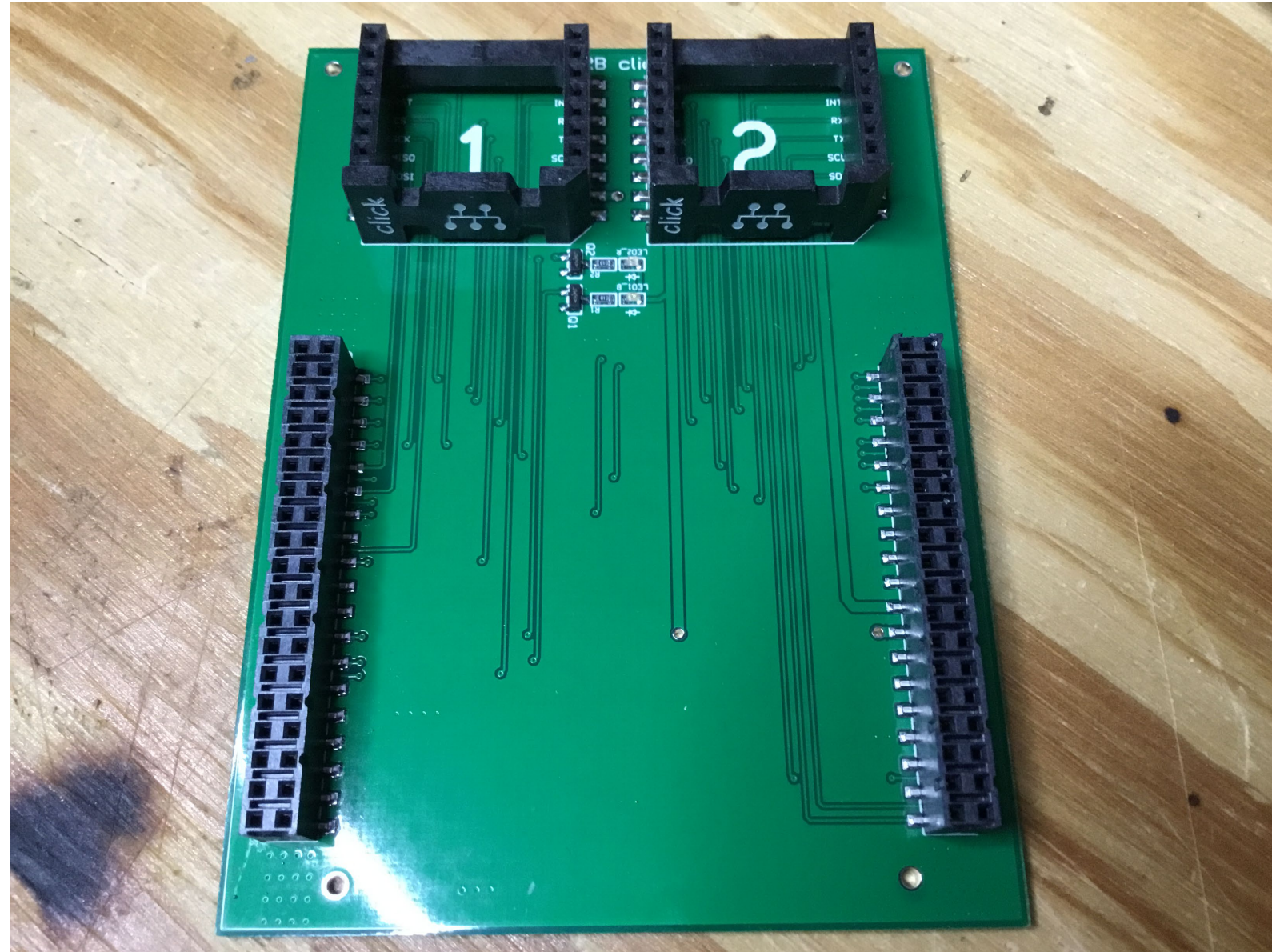
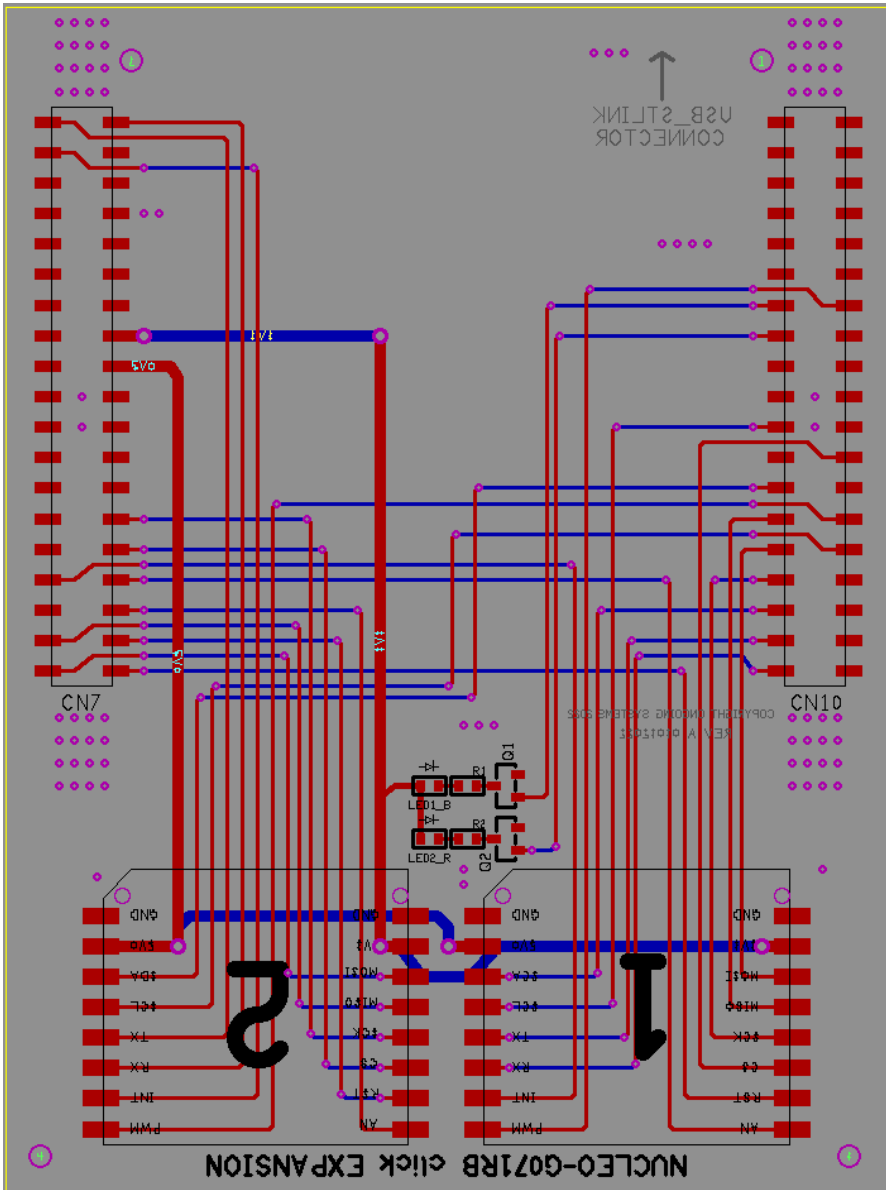
CN7		CN6				CN5		CN10			
PC10	1	2		PC11		PC9	10	1	2	PC8	
PC12	3	4		PD2		PB8	9	3	4	PC6	
VDD	5	6		E5V		PB9	8	5	6	PA3	
PA14	7	8		GND		GND	7	7	8	PD6	
PD0	9	10	1	PD1	NC	PA5	6	9	10	PA12	
PD3	11	12	2	IOREF	IOREF	PA6	5	11	12	PA11	
PA13	13	14	3	NRST	NRST	PA7	4	13	14	PC1	
PD4	15	16	4	+3V3	+3V3	PB0	3	15	16	PC0	
PA15	17	18	5	+5V	+5V	PC7	2	17	18	GND	
GND	19	20	6	GND	GND	PA9	1	19	20	PB2	
PB7	21	22	7	GND	GND	PA8	8	21	22	PB6	
PC13	23	24	8	VIN	VIN	PB14	7	23	24	PB15	
PC14	25	26		PD5		PB4	6	25	26	PB10	
PC15	27	28	1	PA0	A0	PB5	5	27	28	PB13	
PF0	29	30	2	PA1	A1	PB3	4	29	30	AGND	
PF1	31	32	3	PA4	A2	PA10	3	31	32	PA2	
VBAT	33	34	4	PB1	A3	PC4	2	33	34	PD8	
PC2	35	36	5	PB11/PB9	A4	PC5	1	35	36	PD9	
PC3	37	38	6	PB12/PB8	A5			37	38		

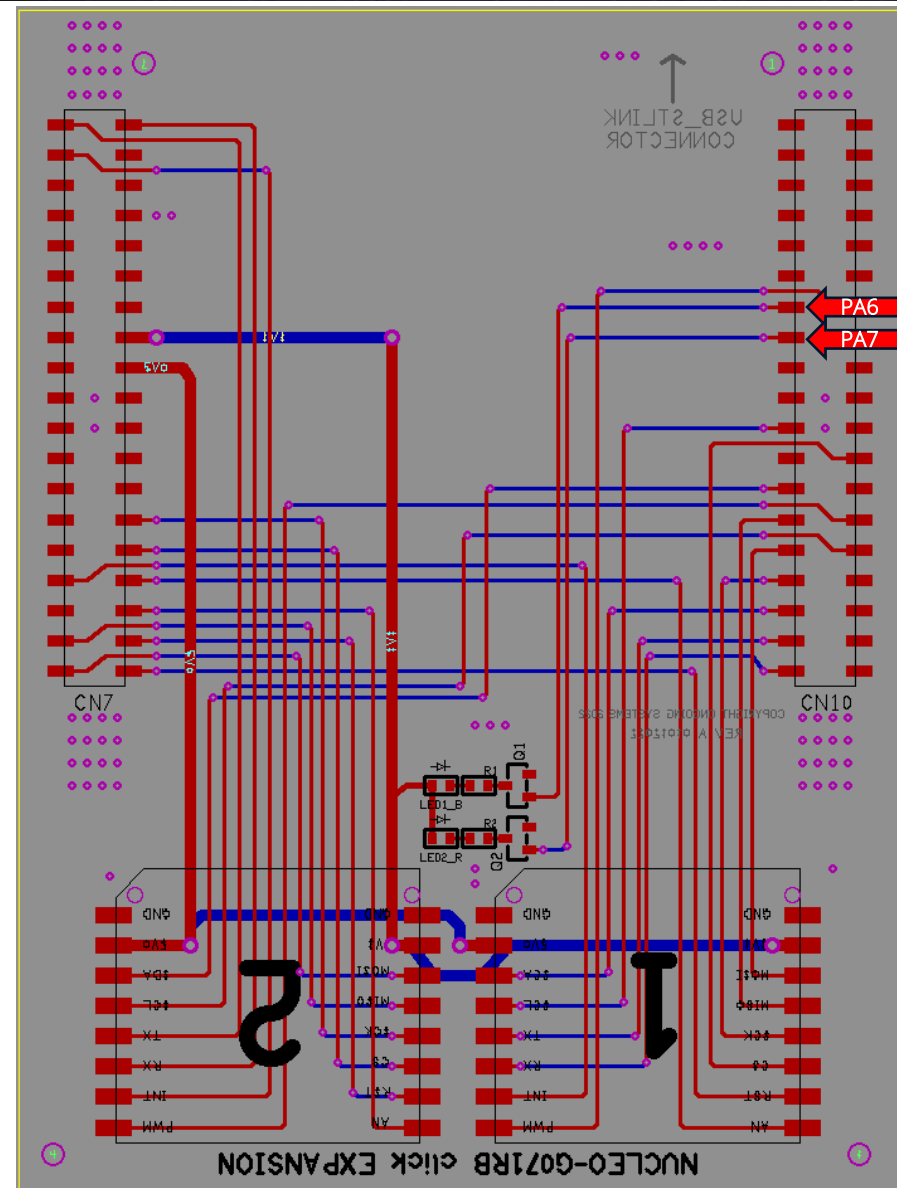
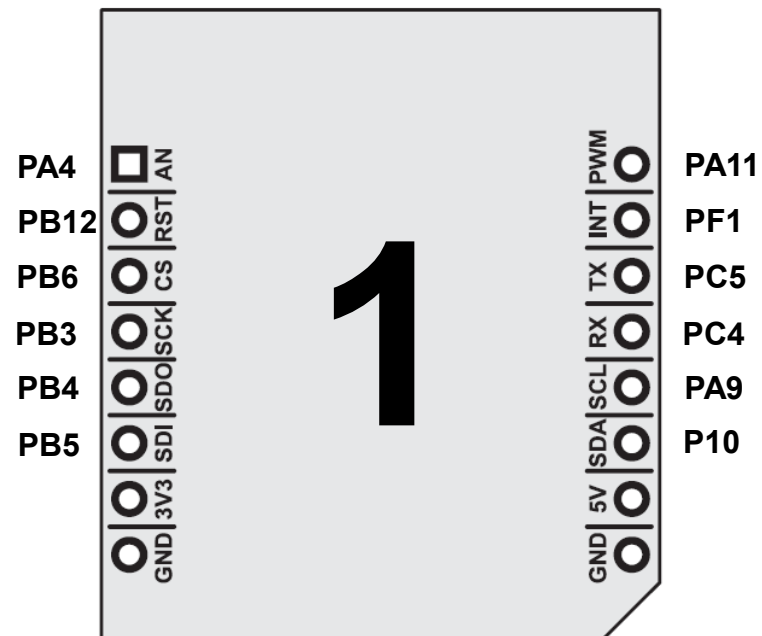


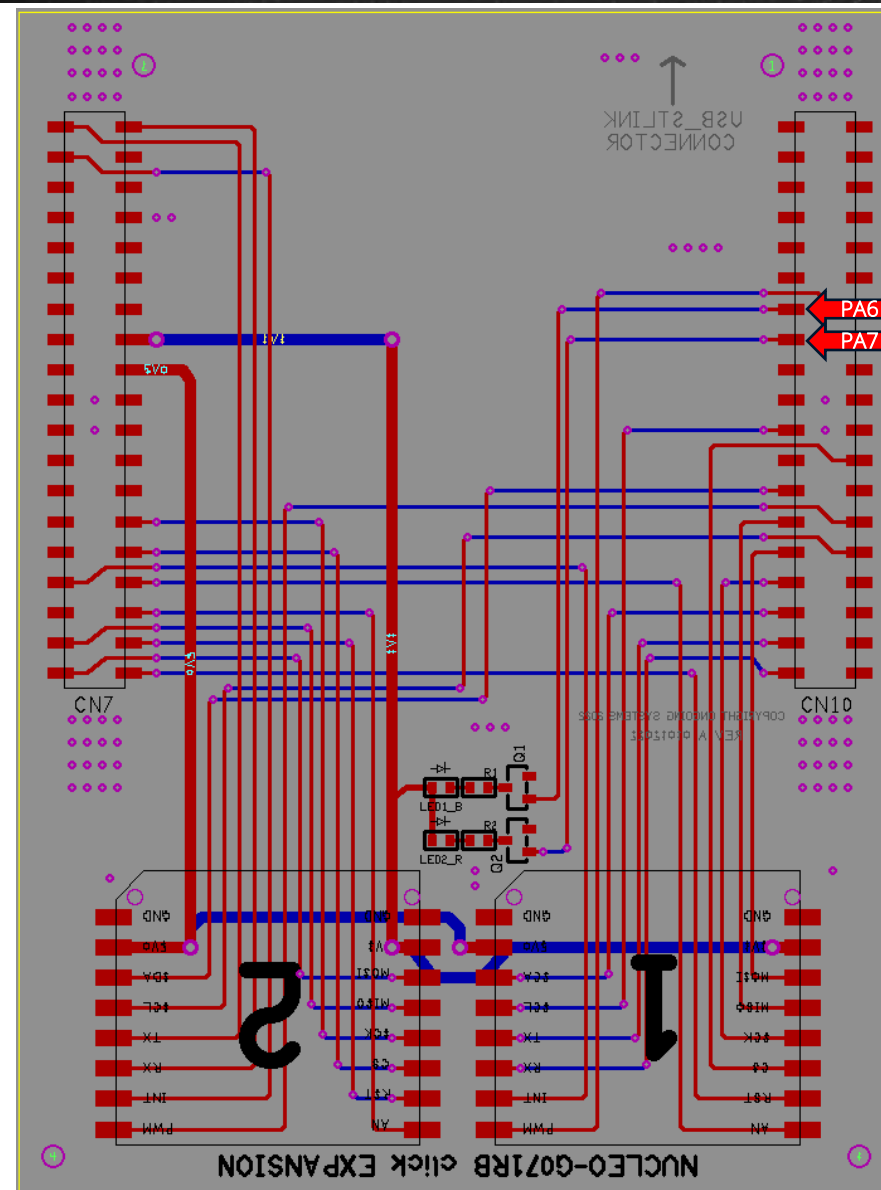
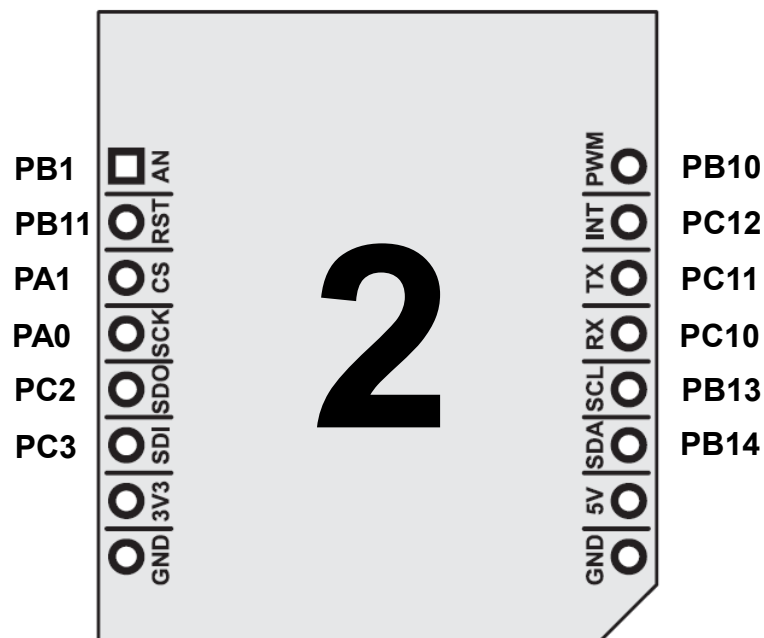
NUCLEO-G071RB click EXPANSION: G070RB Schematic



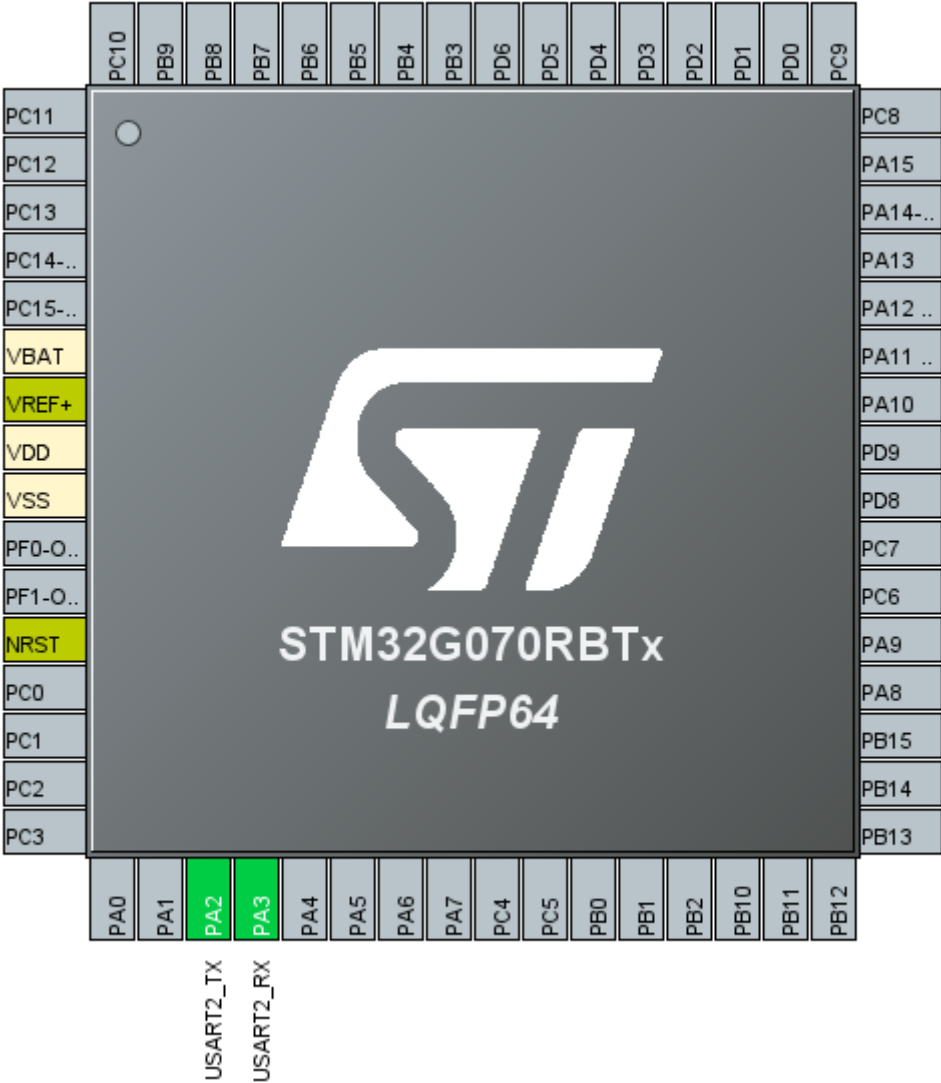
NUCLEO-G071RB click EXPANSION (SLED)



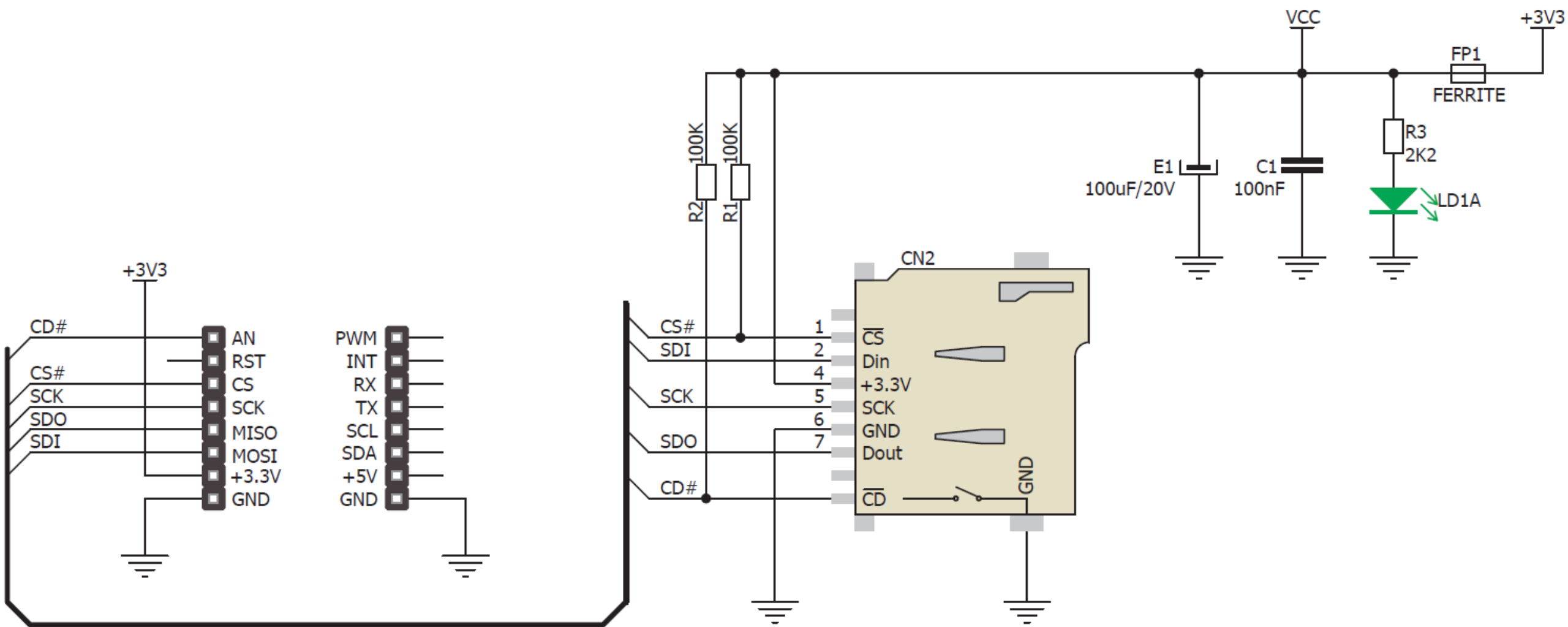
NUCLEO-G071RB click EXPANSION (SLED)

NUCLEO-G071RB click EXPANSION (SLED)

FatFs Driver: Hardware Configuration



FatFs Driver: Hardware Configuration



FatFs Driver: stm32g0xx_nucleo.h

```

/*##### SPI1 #####*/
#define NUCLEO_SPIx          SPI1
#define NUCLEO_SPIx_CLK_ENABLE()  __HAL_RCC_SPI1_CLK_ENABLE()

#define NUCLEO_SPIx_SCK_AF    GPIO_AF0_SPI1
#define NUCLEO_SPIx_SCK_GPIO_PORT  GPIOB
#define NUCLEO_SPIx_SCK_PIN   GPIO_PIN_3
#define NUCLEO_SPIx_SCK_GPIO_CLK_ENABLE()  __HAL_RCC_GPIOB_CLK_ENABLE()
#define NUCLEO_SPIx_SCK_GPIO_CLK_DISABLE() __HAL_RCC_GPIOB_CLK_DISABLE()

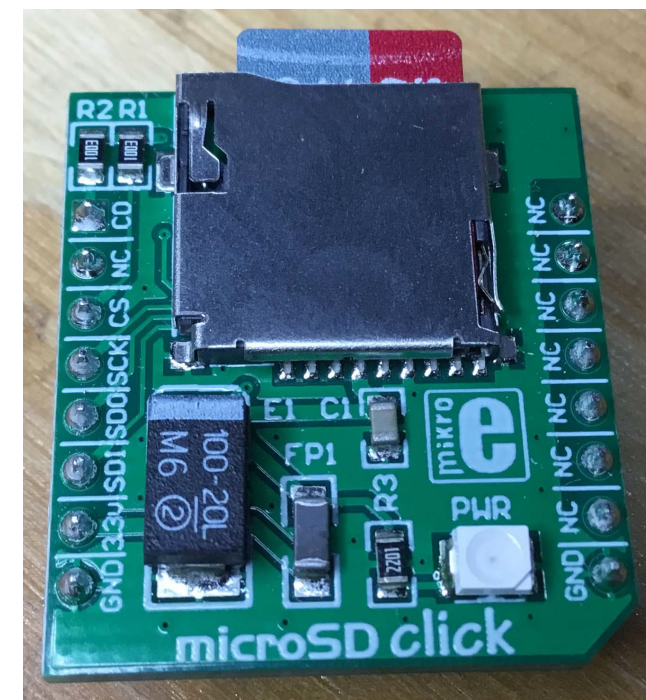
#define NUCLEO_SPIx_MISO_MOSI_AF    GPIO_AF0_SPI1
#define NUCLEO_SPIx_MISO_MOSI_GPIO_PORT  GPIOB
#define NUCLEO_SPIx_MISO_MOSI_GPIO_CLK_ENABLE()  __HAL_RCC_GPIOB_CLK_ENABLE()
#define NUCLEO_SPIx_MISO_MOSI_GPIO_CLK_DISABLE() __HAL_RCC_GPIOB_CLK_DISABLE()
#define NUCLEO_SPIx_MISO_PIN        GPIO_PIN_4
#define NUCLEO_SPIx_MOSI_PIN        GPIO_PIN_5

/* Maximum Timeout values for flags waiting loops. These timeouts are not based
on accurate values, they just guarantee that the application will not remain
stuck if the SPI communication is corrupted.
You may modify these timeout values depending on CPU frequency and application
conditions (interrupts routines ...). */
#define NUCLEO_SPIx_TIMEOUT_MAX      1000

/**
 * @brief SD Control Lines management
 */
#define SD_CS_LOW()      HAL_GPIO_WritePin(SD_CS_GPIO_PORT, SD_CS_PIN, GPIO_PIN_RESET)
#define SD_CS_HIGH()     HAL_GPIO_WritePin(SD_CS_GPIO_PORT, SD_CS_PIN, GPIO_PIN_SET)

/**
 * @brief SD Control Interface pins (shield D4)
 */
#define SD_CS_PIN        GPIO_PIN_6
#define SD_CS_GPIO_PORT  GPIOB
#define SD_CS_GPIO_CLK_ENABLE()  __HAL_RCC_GPIOB_CLK_ENABLE()
#define SD_CS_GPIO_CLK_DISABLE() __HAL_RCC_GPIOB_CLK_DISABLE()

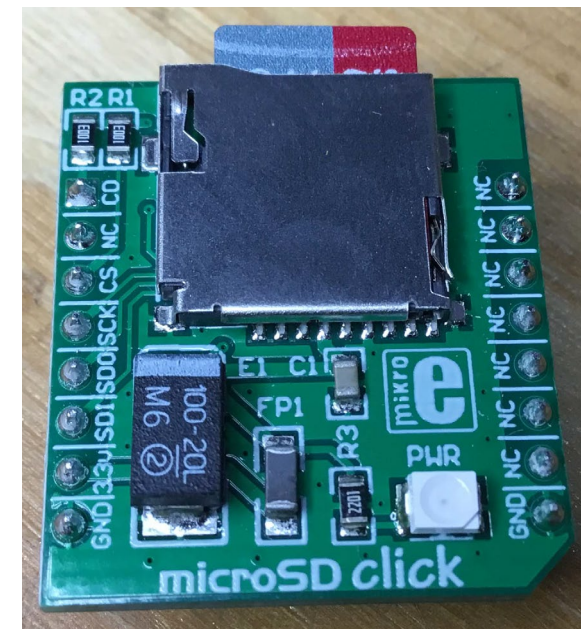
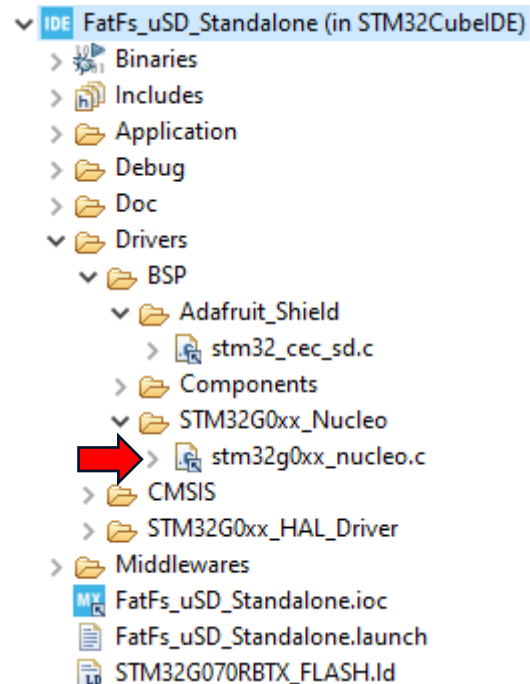
```



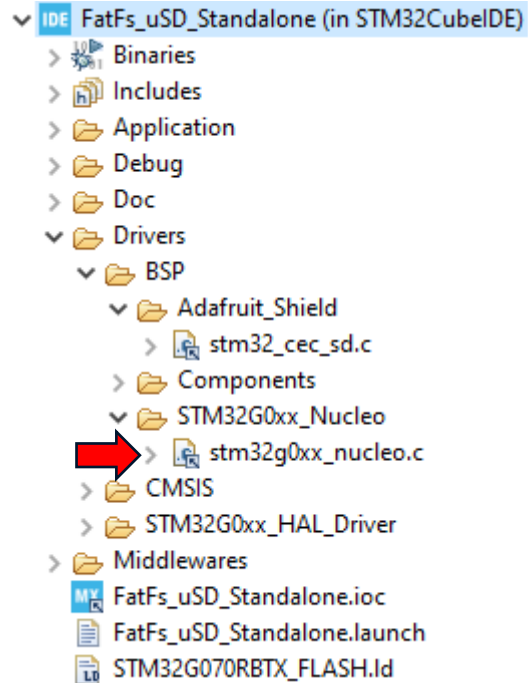
FatFs Driver: **stm32g0xx_nucleo.c**

```
static void SPIx_Init(void)
{
    if(HAL_SPI_GetState(&hnucleo_Spi) == HAL_SPI_STATE_RESET)
    {
        /* SPI Config */
        hnucleo_Spi.Instance = NUCLEO_SPIx;
        /* SPI baudrate is set to 12 MHz maximum (PCLK1/SPI_BaudRatePrescaler = 48/4 = 12 MHz)
        to verify these constraints:
        - ST7735 LCD SPI interface max baudrate is 15MHz for write and 6.66MHz for read
        Since the provided driver doesn't use read capability from LCD, only constraint
        on write baudrate is considered.
        - SD card SPI interface max baudrate is 25MHz for write/read
        - PCLK1 max frequency is 48 MHz
        */
        hnucleo_Spi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
        hnucleo_Spi.Init.Direction = SPI_DIRECTION_2LINES;
        hnucleo_Spi.Init.CLKPhase = SPI_PHASE_2EDGE;
        hnucleo_Spi.Init.CLKPolarity = SPI_POLARITY_HIGH;
        hnucleo_Spi.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
        hnucleo_Spi.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
        hnucleo_Spi.Init.CRCPolynomial = 7;
        hnucleo_Spi.Init.DataSize = SPI_DATASIZE_8BIT;
        hnucleo_Spi.Init.FirstBit = SPI_FIRSTBIT_MSB;
        hnucleo_Spi.Init.NSS = SPI_NSS_SOFT;
        hnucleo_Spi.Init.TIMode = SPI_TIMODE_DISABLE;
        hnucleo_Spi.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
        hnucleo_Spi.Init.Mode = SPI_MODE_MASTER;

        SPIx_MspInit();
        HAL_SPI_Init(&hnucleo_Spi);
    }
}
```



FatFs Driver: `stm32g0xx_nucleo.c`



```
static void SPIx_MspInit(void)
{
    GPIO_InitTypeDef  gpioinitstruct = {0};

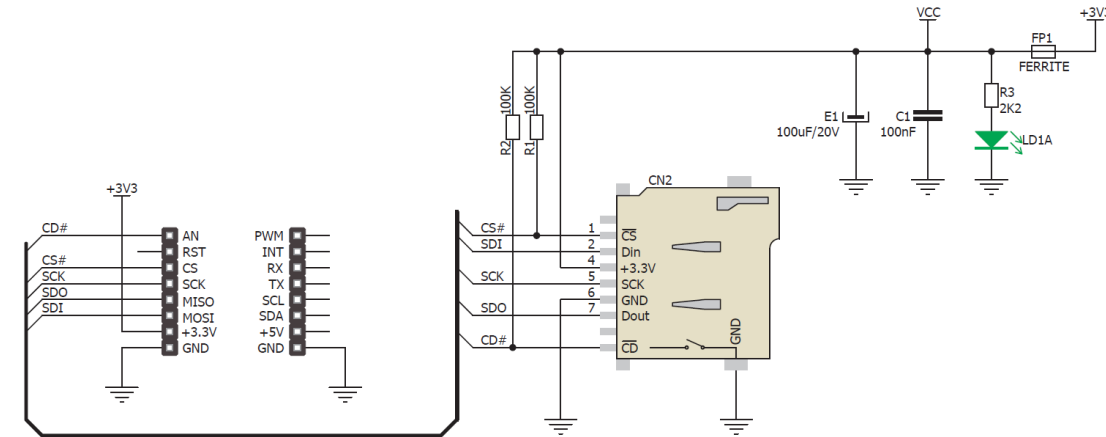
    /*** Configure the GPIOs ***/
    /* Enable GPIO clock */
    NUCLEO_SPIx_SCK_GPIO_CLK_ENABLE();
    NUCLEO_SPIx_MISO_MOSI_GPIO_CLK_ENABLE();

    /* Configure SPI SCK */
    gpioinitstruct.Pin = NUCLEO_SPIx_SCK_PIN;
    gpioinitstruct.Mode = GPIO_MODE_AF_PP;
    gpioinitstruct.Pull = GPIO_PULLUP;
    gpioinitstruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    gpioinitstruct.Alternate = NUCLEO_SPIx_SCK_AF;
    HAL_GPIO_Init(NUCLEO_SPIx_SCK_GPIO_PORT, &gpioinitstruct);

    /* Configure SPI MISO and MOSI */
    gpioinitstruct.Pin = NUCLEO_SPIx_MOSI_PIN;
    gpioinitstruct.Alternate = NUCLEO_SPIx_MISO_MOSI_AF;
    gpioinitstruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(NUCLEO_SPIx_MISO_MOSI_GPIO_PORT, &gpioinitstruct);

    gpioinitstruct.Pin = NUCLEO_SPIx_MISO_PIN;
    HAL_GPIO_Init(NUCLEO_SPIx_MISO_MOSI_GPIO_PORT, &gpioinitstruct);

    /*** Configure the SPI peripheral ***/
    /* Enable SPI clock */
    NUCLEO_SPIx_CLK_ENABLE();
}
```



FatFs Driver: app_fatfs.c

```
static int32_t FS_FileOperations(void)
{
    FRESULT res; /* FatFs function common result code */
    uint32_t byteswritten, bytesread; /* File write/read counts */
    uint8_t wtext[] = "This is CEC's NUCLEO-G07RB working with FatFs and uSD diskio driver!";
    /* File write buffer */
    uint8_t rtext[100]; /* File read buffer */

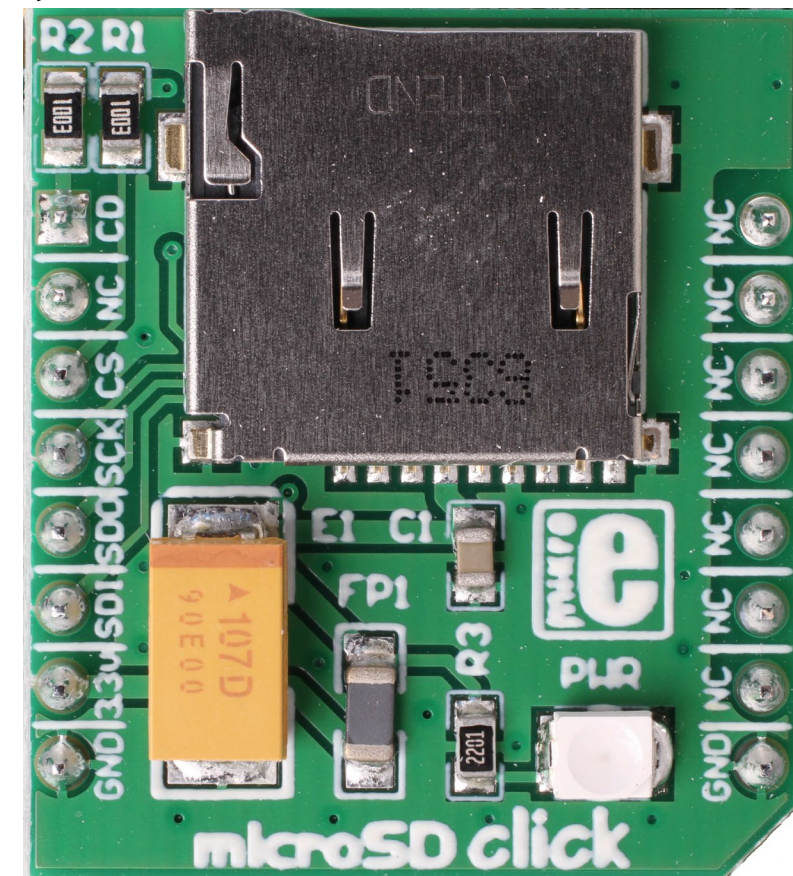
    /* Register the file system object to the FatFs module */
    if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) == FR_OK)
    {
        /* Create and Open a new text file object with write access */
        if(f_open(&SDFFile, "STM32CEC.TXT", FA_CREATE_ALWAYS | FA_WRITE) == FR_OK)
        {
            /* Write data to the text file */
            res = f_write(&SDFFile, wtext, sizeof(wtext), (void *)&byteswritten);

            if((byteswritten > 0) && (res == FR_OK))
            {
                /* Close the open text file */
                f_close(&SDFFile);

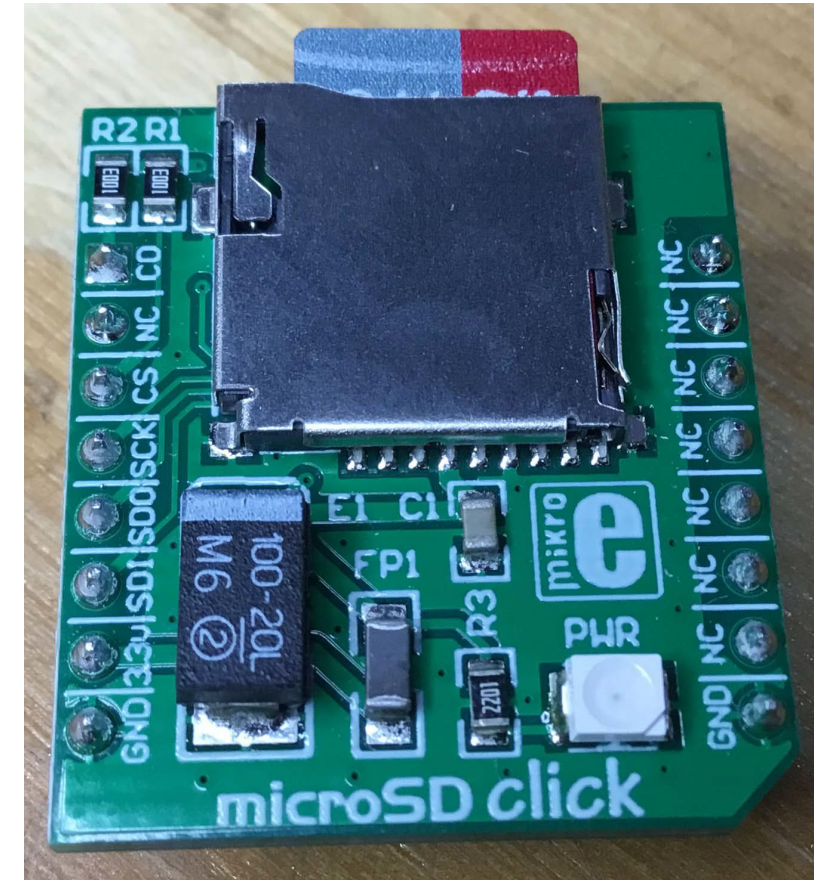
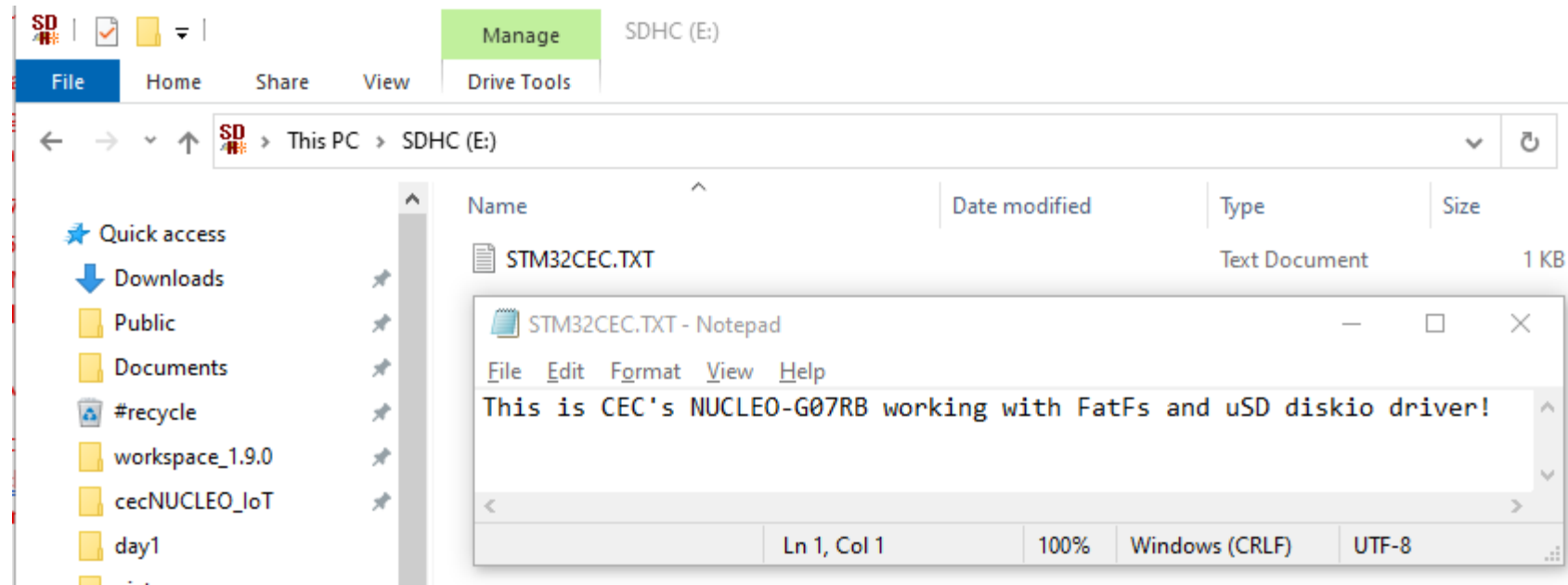
                /* Open the text file object with read access */
                if(f_open(&SDFFile, "STM32CEC.TXT", FA_READ) == FR_OK)
                {
                    /* Read data from the text file */
                    res = f_read(&SDFFile, rtext, sizeof(rtext), (void *)&bytesread);

                    if((bytesread > 0) && (res == FR_OK))
                    {
                        /* Close the open text file */
                        f_close(&SDFFile);

                        /* Compare read data with the expected data */
                        if((bytesread == byteswritten))
                        {
                            if(Buffercmp((uint32_t *)rtext, (uint32_t *)wtext, sizeof(rtext)))
                            {
                                /* Success no error occurrence */
                                return 0;
                            }
                        }
                    }
                }
            }
        }
    }
}
```



FatFs Driver: Fire It UP!!



STM32U585 Nx_TCP_Echo_Client: **app_netxduo.h**

IDE Nx_TCP_Echo_Client (in STM32CubeIDE)

- > Binaries
- > Includes
- > Application
 - > User
 - > AZURE_RTOS
 - > Core
 - > NetXDuo
 - > App
 - > **app_netxduo.c**
 - > Startup
 - > Debug
 - > Drivers
 - > Middlewares
 - > Release
 - Nx_TCP_Echo_Client.ioc
 - Nx_TCP_Echo_Client.launch
 - STM32U585AIIX_FLASH.ld

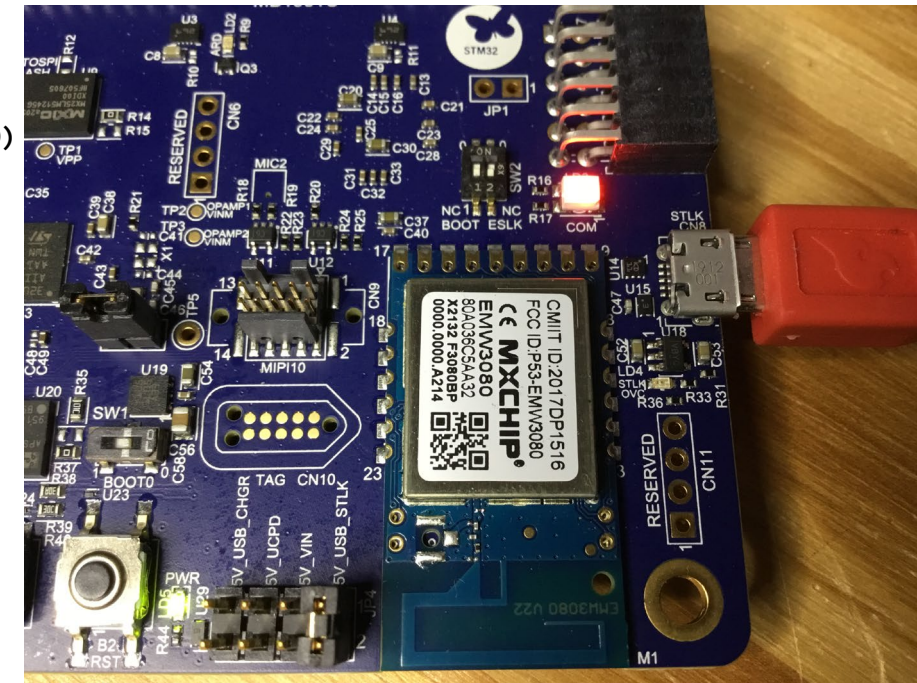
```
#define PAYLOAD_SIZE          1544
#define NX_PACKET_POOL_SIZE   (( PAYLOAD_SIZE + sizeof(NX_PACKET)) * 10)
#define WINDOW_SIZE          512

#define DEFAULT_MEMORY_SIZE   1024
#define DEFAULT_PRIORITY      10

#define NULL_ADDRESS          0

#define DEFAULT_PORT          6000
#define TCP_SERVER_PORT       6001
#define TCP_SERVER_ADDRESS    IP_ADDRESS(192, 168, 1, 73)

#define MAX_PACKET_COUNT      5
#define DEFAULT_MESSAGE       "TCP Client on STM32U5-IOT\r\n"
#define DEFAULT_TIMEOUT       10 * NX_IP_PERIODIC_RATE
```



STM32U585 Nx_TCP_Echo_Client: **app_netxduo.c**

IDE Nx_TCP_Echo_Client (in STM32CubeIDE)

- > Binaries
- > Includes
- ▼ Application
 - ▼ User
 - > AZURE_RTOS
 - > Core
 - ▼ NetXDuo
 - ▼ App
 - ➔ **app_netxduo.c**
 - > Startup
- > Debug
- > Drivers
- > Middlewares
- > Release
- Nx_TCP_Echo_Client.ioc
- Nx_TCP_Echo_Client.launch
- STM32U585AIIX_FLASH.ld

```
/* create the TCP socket */
```

```
ret = nx_tcp_socket_create(&IpInstance, &TCPSocket, "TCP Server Socket", NX_IP_NORMAL, NX_FRAGMENT_OKAY,  
                          NX_IP_TIME_TO_LIVE, WINDOW_SIZE, NX_NULL, NX_NULL);
```

```
if (ret != NX_SUCCESS)  
{  
    Error_Handler();  
}
```

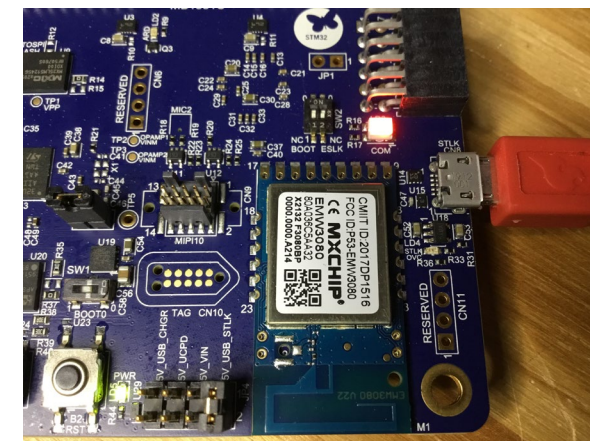
```
/* bind the client socket for the DEFAULT_PORT */
```

```
ret = nx_tcp_client_socket_bind(&TCPSocket, DEFAULT_PORT, NX_WAIT_FOREVER);
```

```
if (ret != NX_SUCCESS)  
{  
    Error_Handler();  
}
```

```
/* connect to the remote server on the specified port */
```

```
ret = nx_tcp_client_socket_connect(&TCPSocket, TCP_SERVER_ADDRESS, TCP_SERVER_PORT, NX_WAIT_FOREVER);
```



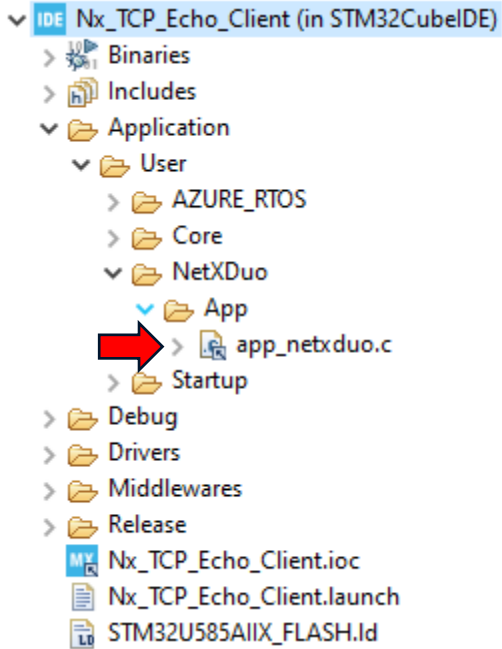
STM32U585 Nx_TCP_Echo_Client: app_netxduo.c

```
while(count++ < MAX_PACKET_COUNT)
{
    TX_MEMSET(data_buffer, '\0', sizeof(data_buffer));

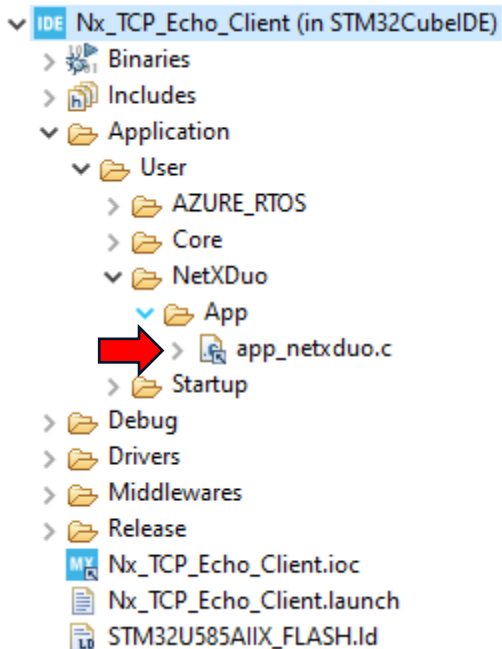
    /* allocate the packet to send over the TCP socket */
    ret = nx_packet_allocate(&AppPool, &data_packet, NX_TCP_PACKET, TX_WAIT_FOREVER);

    if (ret != NX_SUCCESS)
    {
        break;
    }

    /* append the message to send into the packet */
    ret = nx_packet_data_append(data_packet, (VOID *)DEFAULT_MESSAGE, sizeof(DEFAULT_MESSAGE), &AppPool, TX_WAIT_FOREVER);
```



STM32U585 Nx_TCP_Echo_Client: app_netxduo.c



```

/* send the packet over the TCP socket */
ret = nx_tcp_socket_send(&TCPSocket, data_packet, DEFAULT_TIMEOUT);

if (ret != NX_SUCCESS)
{
    break;
}

/* wait for the server response */
ret = nx_tcp_socket_receive(&TCPSocket, &server_packet, DEFAULT_TIMEOUT);

if (ret == NX_SUCCESS)
{
    /* get the server IP address and port */
    nx_udp_source_extract(server_packet, &source_ip_address, &source_port);

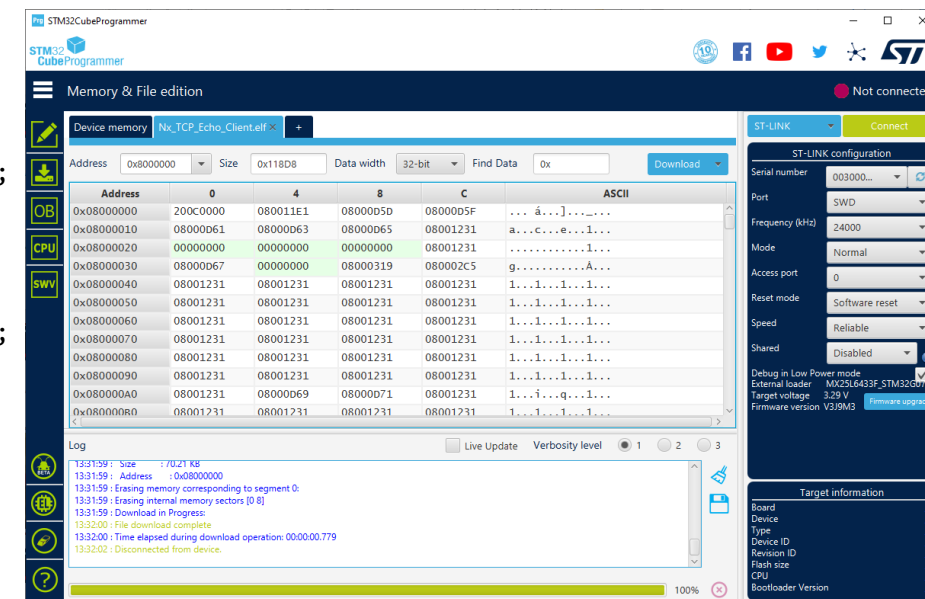
    /* retrieve the data sent by the server */
    nx_packet_data_retrieve(server_packet, data_buffer, &bytes_read);

    /* print the received data */
    PRINT_DATA(source_ip_address, source_port, data_buffer);

    /* release the server packet */
    nx_packet_release(server_packet);

    /* toggle the green led on success */
    BSP_LED_Toggle(LED_GREEN);
}

```



STM32U585 Nx_TCP_Echo_Client: Fire It UP!!

RealTerm: Serial Capture Program 2.0.0.70

```
Nx_TCP_Echo_Client application started..
STM32 IPAddress: 192.168.1.128
[192.168.1.73:20504] -> 'TCP Client on STM32U5-IOT
[192.168.1.73:20504] -> 'TCP Client on STM32U5-IOT
[192.168.1.73:20504] -> 'TCP Client on STM32U5-IOT
[192.168.1.73:20504] -> 'TCP Client on STM32U5-IOT
[192.168.1.73:20504] -> 'TCP Client on STM32U5-IOT
-----
SUCCESS : 5 / 5 packets sent
```

Display Port Capture Pins Send Echo Port I2C I2C-2 I2CMisc Misc

Baud 115200 Port 16 Open Spy ☒ Change ☒

Parity: ☒ None ☐ Odd ☐ Even ☐ Mark ☐ Space

Data Bits: ☒ 8 bits ☐ 7 bits ☐ 6 bits ☐ 5 bits

Stop Bits: ☒ 1 bit ☐ 2 bits

Hardware Flow Control: ☒ None ☐ RTS/CTS ☐ DTR/DSR ☐ RS485-rt

Software Flow Control: ☐ Receive Xon Char: 17 ☐ Transmit Xoff Char: 19

Winsock is: ☐ Raw ☒ Telnet

Status: ☐ Connected ☐ RXD (2) ☐ TXD (3) ☐ CTS (8) ☐ DCD (1) ☐ DSR (6) ☐ Ring (9) ☐ BREAK ☐ Error

Char Count:910 CPS:0 Port: 16 115200 8N1 No

Hercules SETUP utility by HW-group.com

UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About

Received data

```
TCP Client on STM32U5-IOT
TCP Client on STM32U5-IOT
TCP Client on STM32U5-IOT
TCP Client on STM32U5-IOT
TCP Client on STM32U5-IOT
```

Sent data

Server status

Port: 6001

TEA authorization

TEA key

1: 01020304 3: 090A0B0C

2: 05060708 4: 0D0E0F10

☐ Client authorization

Client connection status

```
1:17:16 PM: All connections clo
1:17:30 PM: All connections clo
1:18:35 PM: 192.168.1.128 Clie
1:19:01 PM: All connections clo
1:19:13 PM: 192.168.1.128 Clie
1:19:21 PM: 192.168.1.128 Clie
1:19:22 PM: 192.168.1.128 Clie
1:21:32 PM: 192.168.1.128 Clie
```

Clients count: -1

Send

+welcome ☐ HEX

Cursor decode

HEX: 0A Decimal: 10 Decoder Input:

Server settings

☒ Server echo ☐ Redirect to UDP

HWgroup
www.HW-group.com
Hercules SETUP utility
Version 3.2.8

Thank you for attending!!!

Please consider the resources below:

- **Today's Project Download Package**
- **AZURE RTOS**
- **FatFs**

To get today's Project Download Package please send an email request to:
therealfreaddy@gmail.com

MORE TO COME..





DesignNews

Thank You

Sponsored by

