



# IoT Device Prototyping with Microchip Curiosity Development Boards

## Day 3:

Constructing a 32-bit Wi-Fi Node Using the PIC32MM USB Curiosity Development Board

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.

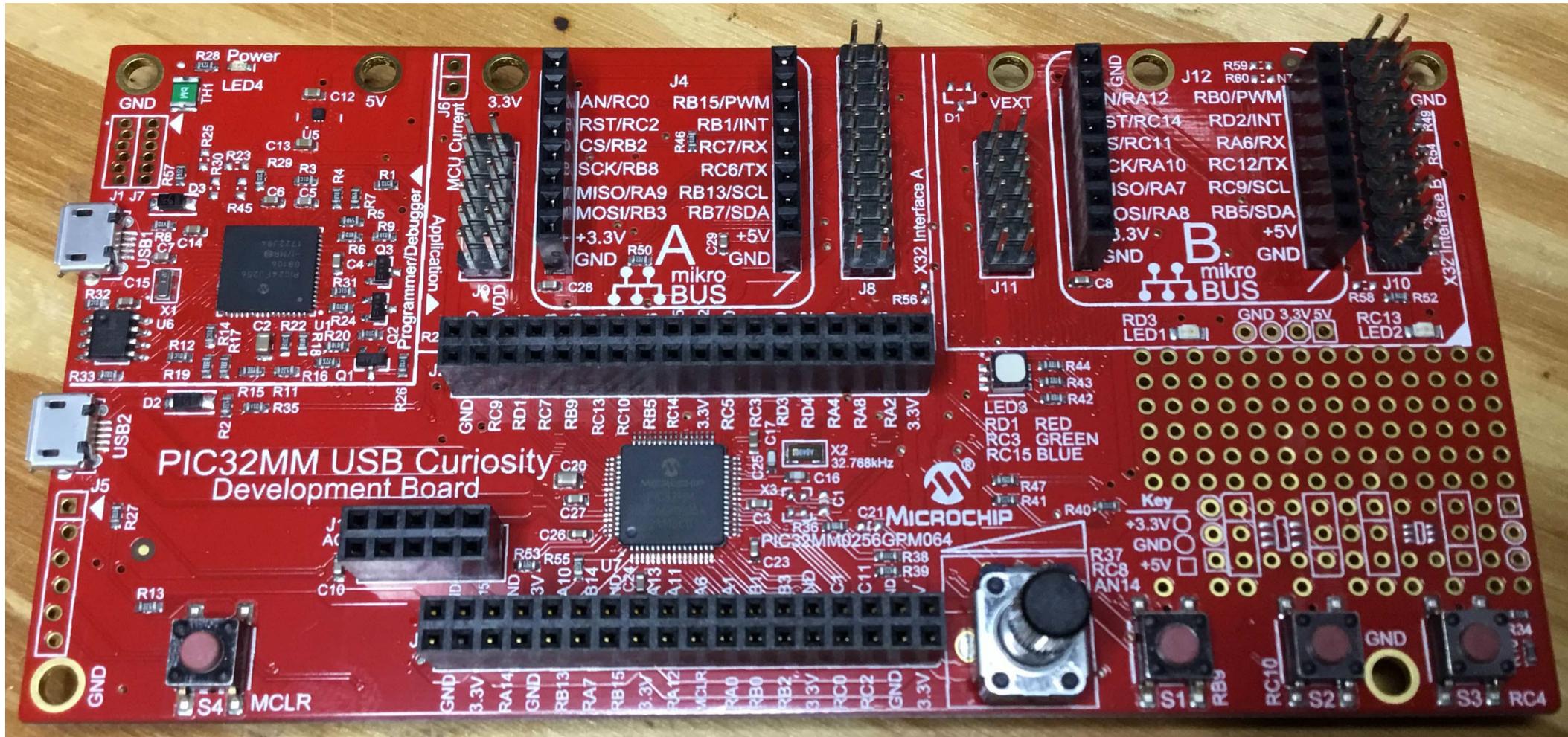


## Fred Eady

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

# AGENDA

## Coding a PIC32MM0256GPM064 WizFi360 Driver



# Create the PIC32MM0256GPM064 MPLAB X Project

The screenshot shows the 'New Project' wizard in MPLAB X IDE, divided into two panels: 'Select Compiler' and 'Select Device'.

**Select Compiler Panel:**

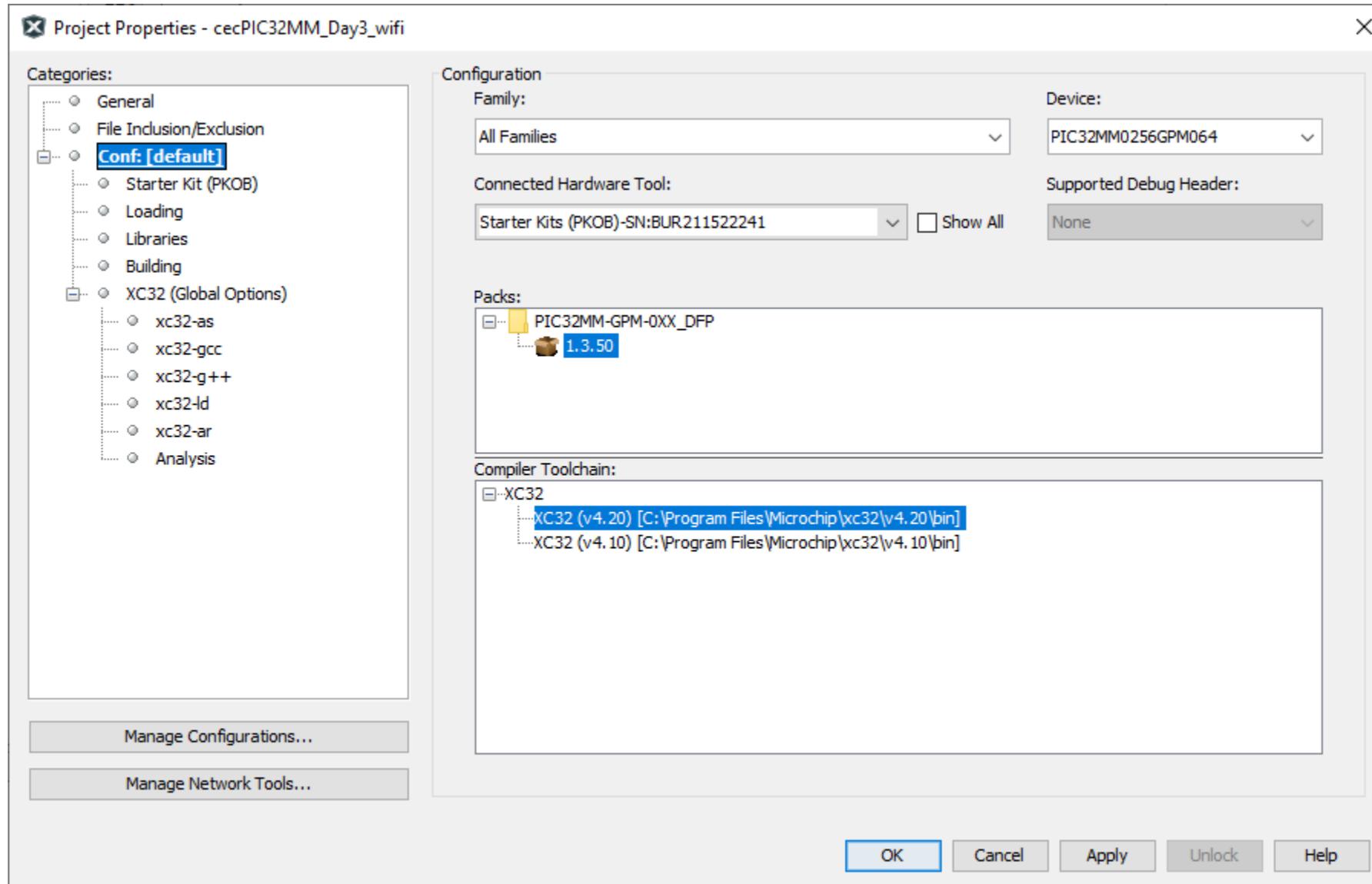
- Steps:**
  1. Choose Project
  2. Select Device
  3. Select Header
  4. Select Plugin Board
  5. **Select Compiler**
  6. Select Project Name and Folder
- Compiler Toolchains:**
  - XC32
  - XC32 (v4.20) [C:\Program Files\Microchip\xc32\v4.20\bin] (highlighted)
  - XC32 (v4.10) [C:\Program Files\Microchip\xc32\v4.10\bin]

**Select Device Panel:**

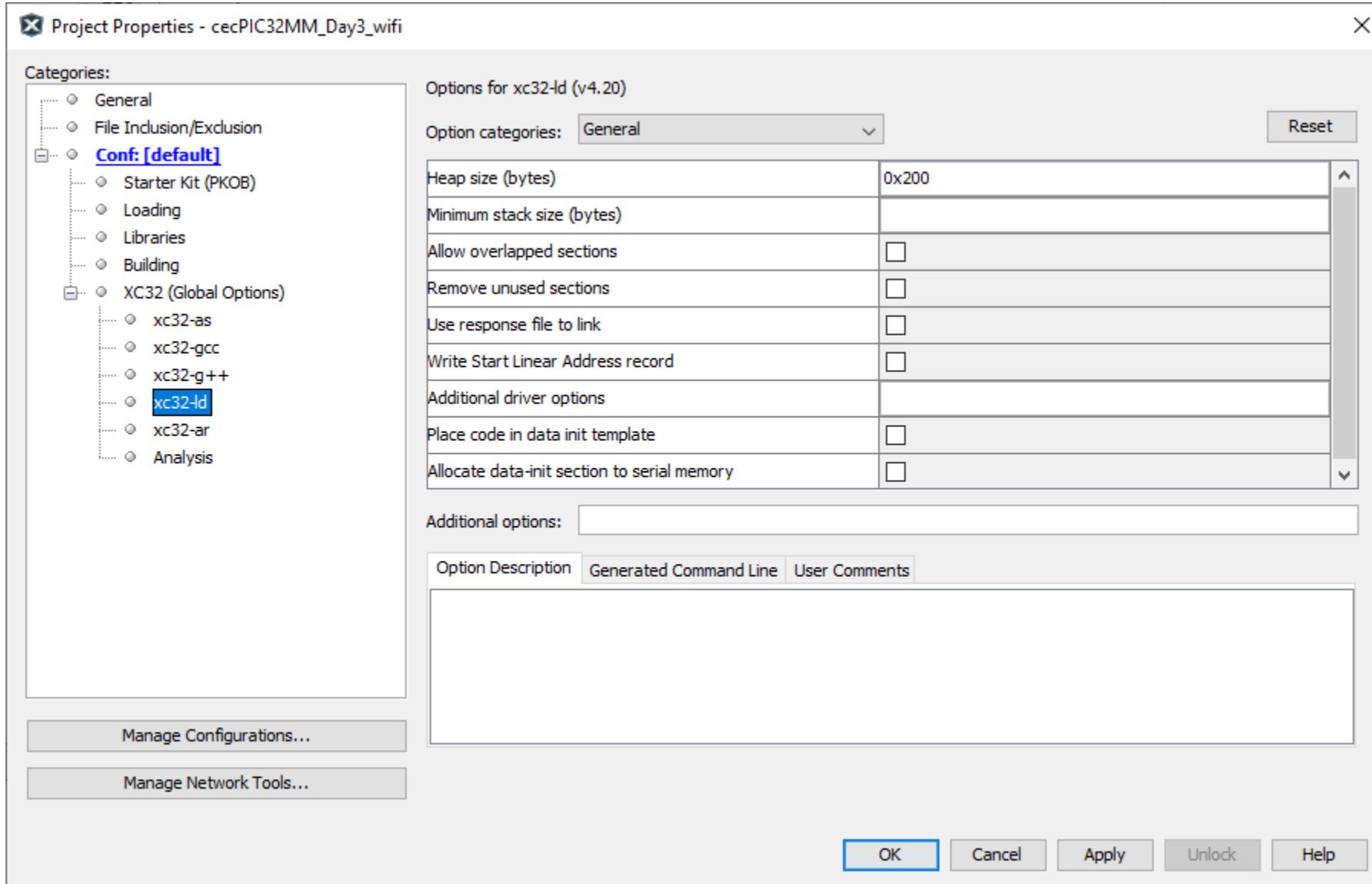
- Family:** 32-bit MCUs (PIC32)
- Device:** PIC32MM0256GPM064 (highlighted)
- Tool:** No Tool
- Show All

Navigation buttons at the bottom of each panel include '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Next >' button is highlighted in both panels.

# Create the PIC32MM0256GPM064 MPLAB X Project



## Define the Heap Size



Project Properties - cecPIC32MM\_Day3\_wifi

Categories:

- General
- File Inclusion/Exclusion
- Conf: [default]**
  - Starter Kit (PKOB)
  - Loading
  - Libraries
  - Building
  - XC32 (Global Options)
    - xc32-as
    - xc32-gcc
    - xc32-g++
    - xc32-ld**
    - xc32-ar
    - Analysis

Options for xc32-ld (v4.20)

Option categories: General Reset

Heap size (bytes)	0x200
Minimum stack size (bytes)	
Allow overlapped sections	<input type="checkbox"/>
Remove unused sections	<input type="checkbox"/>
Use response file to link	<input type="checkbox"/>
Write Start Linear Address record	<input type="checkbox"/>
Additional driver options	
Place code in data init template	<input type="checkbox"/>
Allocate data-init section to serial memory	<input type="checkbox"/>

Additional options:

Option Description	Generated Command Line	User Comments

Manage Configurations...  
Manage Network Tools...

OK Cancel Apply Unlock Help

# MCC Content Manager Selections

The screenshot displays the 'MCC Content Manager Wizard' interface. At the top, it shows the title 'MCC Content Manager Wizard' and two steps: '1. Content Type' and '2. Required Device Content'. The current step is '1. Content Type', with the instruction 'Select a Content Type'. Three options are presented in a row:

- MCC Melody**: Supports the MCC Builder, Supports content versioning at driver level, An iteration of MCC Generated Code, Works both on- and off-line. Includes a 'Select MCC Melody' button and a link to 'Release notes and supported devices'.
- MCC Classic**: Development process you are accustomed to, All components and libraries that you have used before. Includes a 'Select MCC Classic' button and a link to 'Release notes and supported devices'.
- MPLAB® Harmony**: Embedded Software Development Framework for 32-bit Microcontrollers and Microprocessors. Includes a 'Select MPLAB Harmony' button and a link to 'Release notes and supported devices'.

Below this row, two larger panels are shown, each with a red arrow pointing to it from the row above. The left panel is for 'MCC Classic' and the right panel is for 'MPLAB® Harmony'. Both panels contain the same descriptive text and 'Select' buttons as the row above, along with their respective 'Release notes and supported devices' links.

# MCC Configuration – System

**System Module Configuration:**

- Clock:** 8000000 Hz, FRC Oscillator (8.0 MHz) Clock Source
- PLL:** PLL Enable (checked), Multiplier: 3:1, Divider: 1:1
- System Clocks:** SYSCLK (24 MHz), PBCLK (24 MHz), USB Clock (12 MHz)
- Clock Output Pin Configuration:** System clock is connected to CLKO/OSC2 pin
- ICD:** Emulator Pin Placement: Communicate on PGEC2/PGED2

**Pin Manager Package View (PIC32MM0256GPM064):**

- Highlighted Pins (Green):** RA0|PGC2, RA1|PGD2, RC13|IO\_RC13\_LED|GPIO, RD3|IO\_RD3\_LED|GPIO
- Other Pins (Blue):** RA7, RB14, RB15, AVSS1, AVDD1, RA13, RA12, RA11, NMCLR, RA6, RB0, RB1, RB2, RB3, RA10, RB13, VUS8V3, RB11, RB10, RA14, RA15, VDD1, VCAP, RC9, RAS, RD1, RC8, RC7|U2RX, RC6|U2TX, RB9, RB8, RB7, RC10, RB6, RB5, RC15, RC14, RC12, VDD6, VSS6, RC5, RC4, RC3, RD0, RD2, RD4, RA9, RA4, RA8, RB4, RA2, CLKO, RA2, VSS4, VDD4, RC1, RC2|IO\_RC2\_RST|GPIO, RC11, RC10|RC0\_WP|GPIO, VDD3, VSS3

# MCC Configuration – EUSART

main.c x uart2.c x uart2.h x Pin Module x Interrupt Module x System Module x UART2 x

Easy Setup Registers

Hardware Settings

Enable UART

Clock Source: PBCLK

Baud Rate: 115200 Error Rate = 0.160

Parity: None

Data Bits: 8

Stop Bits: 1

Flow Control: None

Enable UART Interrupts

Software Settings

Redirect Printf to UART

Software Transmit Buffer Size: 0x8 Bytes

Software Receive Buffer Size: 0x8 Bytes

Pin Manager: Package View x

MICROCHIP  
PIC32MM0256GPM064

Pin	Function
64	RA10
63	RB13
62	VUSB3V3
61	RB11
60	RB10
59	RA14
58	RA15
57	VDD1
56	VCAP
55	RC9
54	RA5
53	RD1
52	RC8
51	RC7 U2RX
50	RC6 U2TX
49	RB9
48	RB8
47	RC13 IO_RC13_LED GPIO
46	RB7
45	RC10
44	RB6
43	RB5
42	RC15
41	RC14
40	RC12
39	VDD6
38	VSS6
37	RC5
36	RC4
35	RC3
34	RD0
33	RD3 IO_RD3_LED GPIO
32	RD2
31	RD4
30	RA9
29	RA4
28	RB4
27	RA8
26	CLKO
25	RA2
24	VSS4
23	VDD4
22	RC11
21	RC2 IO_RC2_RST GPIO
20	RC1
19	RC0 IO_RC0_WP GPIO
18	VSS3
17	VDD3
16	RB3
15	RB2
14	RB1
13	RB0
12	RA1 PGD2
11	RA0 PGC2
10	RA6
9	NMCLR
8	RA11
7	RA12
6	RA13
5	AVDD1
4	AVSS1
3	RB15
2	RB14
1	RA7



## EUSART User Function

```
/* From usart2.h
```

```
void UART2_ResetRxQueue(void);
```

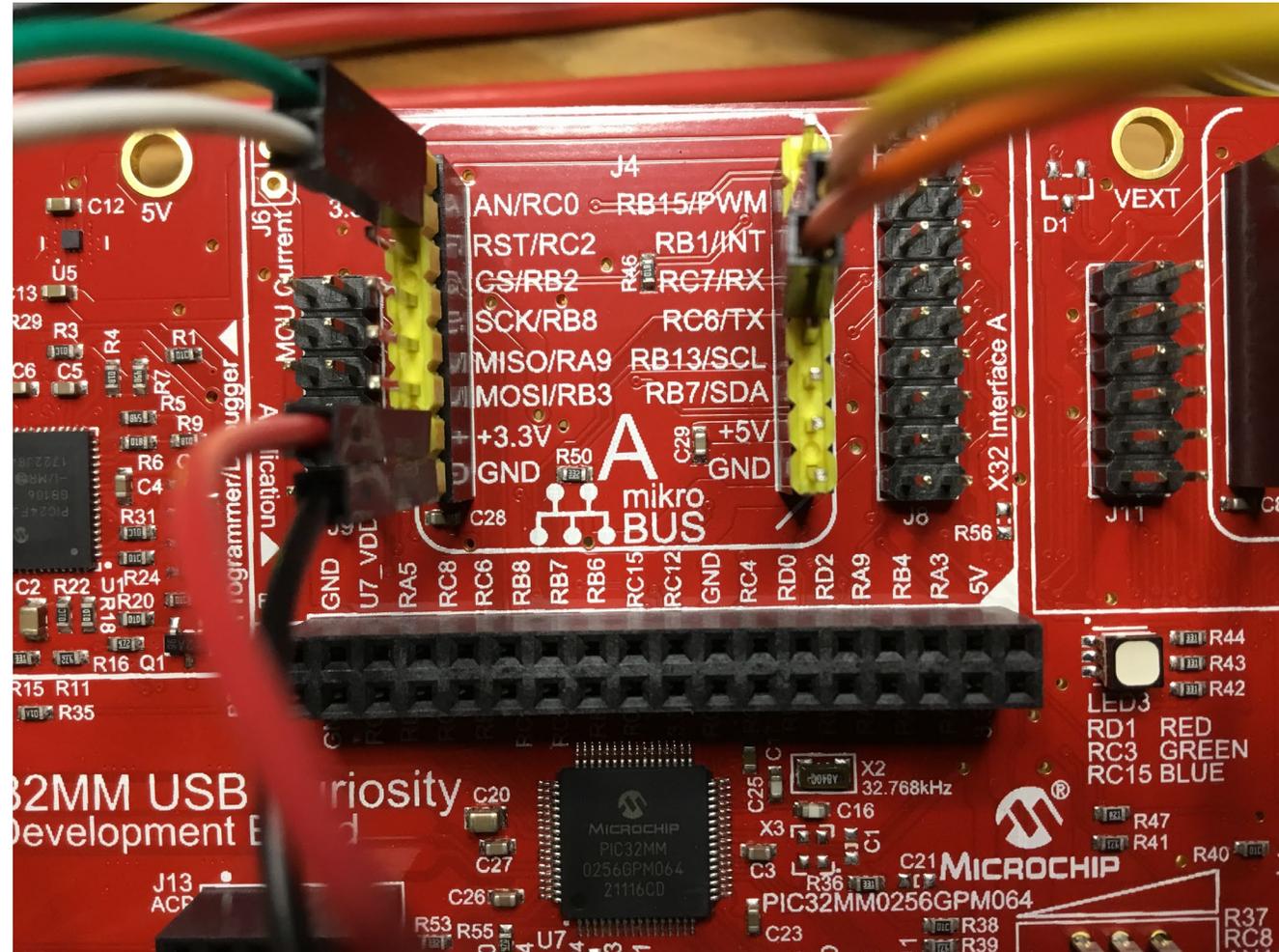
```
/* From usart2.c
```

```
void UART2_ResetRxQueue(void)
```

```
{
```

```
    IEC1bits.U2RXIE = 0; //disable Rx interrupt
    rxHead = rxQueue; //rxHead to top of rxQueue
    rxTail = rxQueue; //rxTail to top of rxQueue
    rxOverflowed = false; //reset overflow flag
    IEC1bits.U2RXIE = 1; //enable Rx interrupt
```

```
}
```

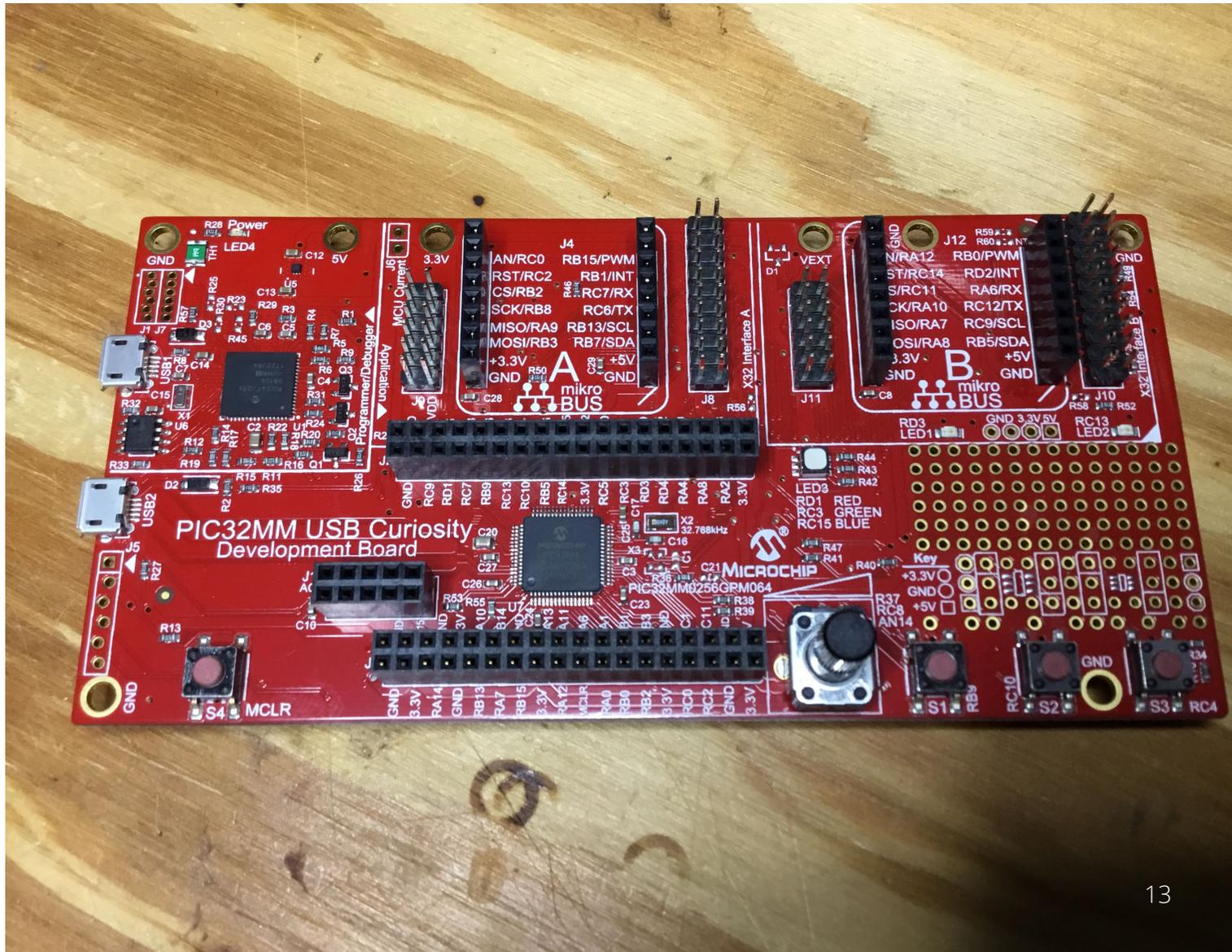


## PIC32MM Delay Functions

```
#define SYS_FREQ 24000000
//*****
//* DELAY FUNCTIONS
//*****

void delay_us(unsigned int us)
{
    // Convert microseconds into clock ticks
    // Core Timer updates every 2 ticks
    us *= SYS_FREQ / 1000000 / 2;
    // Set Core Timer count to 0
    _CP0_SET_COUNT(0);
    // Wait until Core Timer count reaches the
    //us number we calculated earlier
    while (us > _CP0_GET_COUNT());
}

void delay_ms(int ms)
{
    delay_us(ms * 1000);
}
```



## End of Command Function

```
/** *****  
/** CHECK FOR OK  
/** *****  
uint8_t chk_atok(void)  
{  
    rc = 0;  
    IO_RD3_LED_SetLow();  
    while(!UART2_IsRxReady());  
    delay_ms(100);  
  
    do{  
        rxBuf[indx++] = UART2_Read();  
    }while(UART2_IsRxReady());  
  
    indx -= 3;  
    if(rxBuf[indx] == 'K')  
    {  
        rc = 1;  
    }  
    return rc;  
}
```



## Station Mode Function

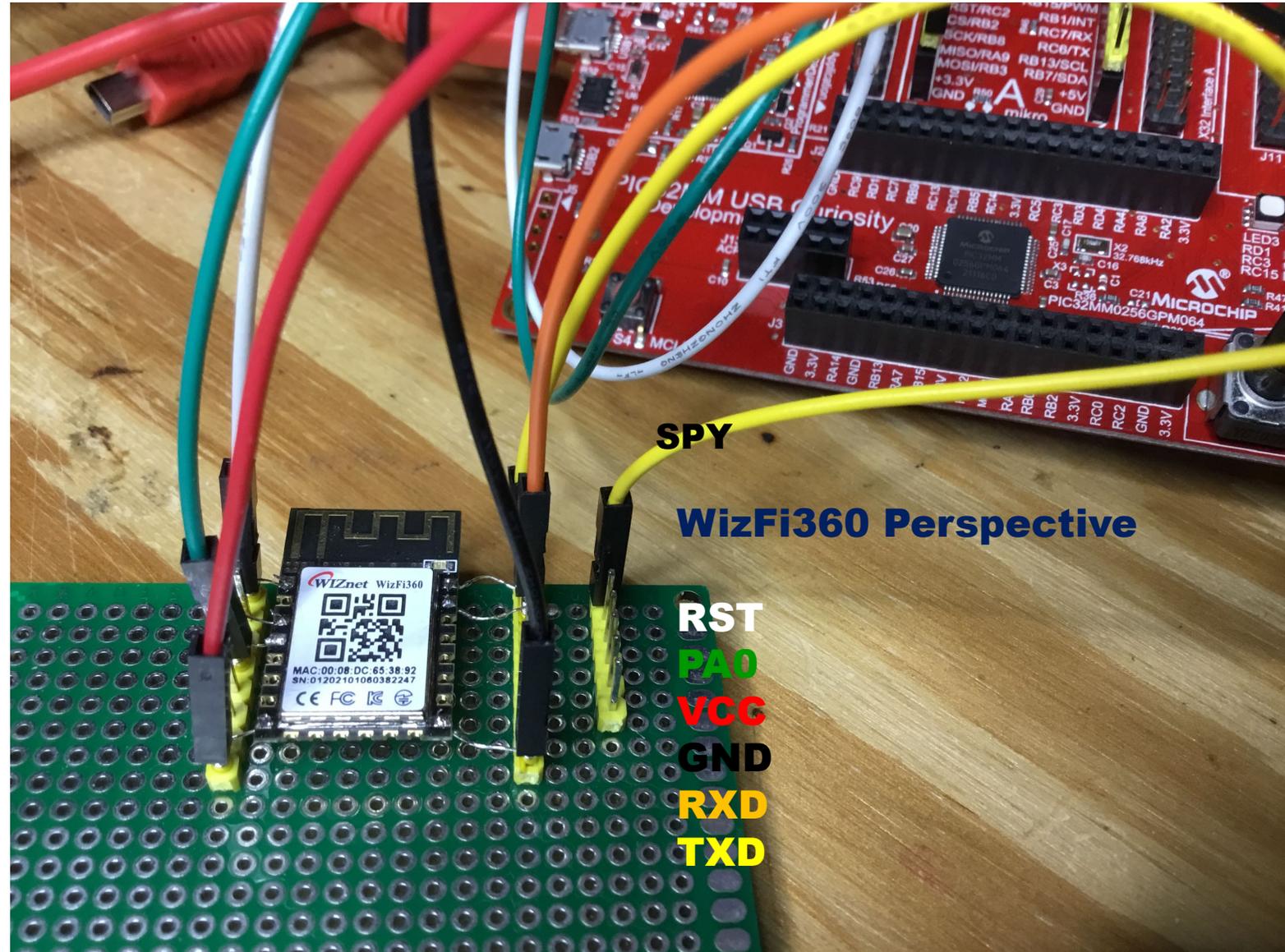
```

//*****
//* SET STATION MODE
//*****
uint8_t setStationMode(void)
{
    rc = 0;
    UART2_ResetRxQueue();
    indx = 0;

    printf("AT+CWMODE_CUR=1\r\n");

    if(chk_atok())
    {
        rc = 1;
    }
    return rc;
}

```



## Single Connection Mode Function

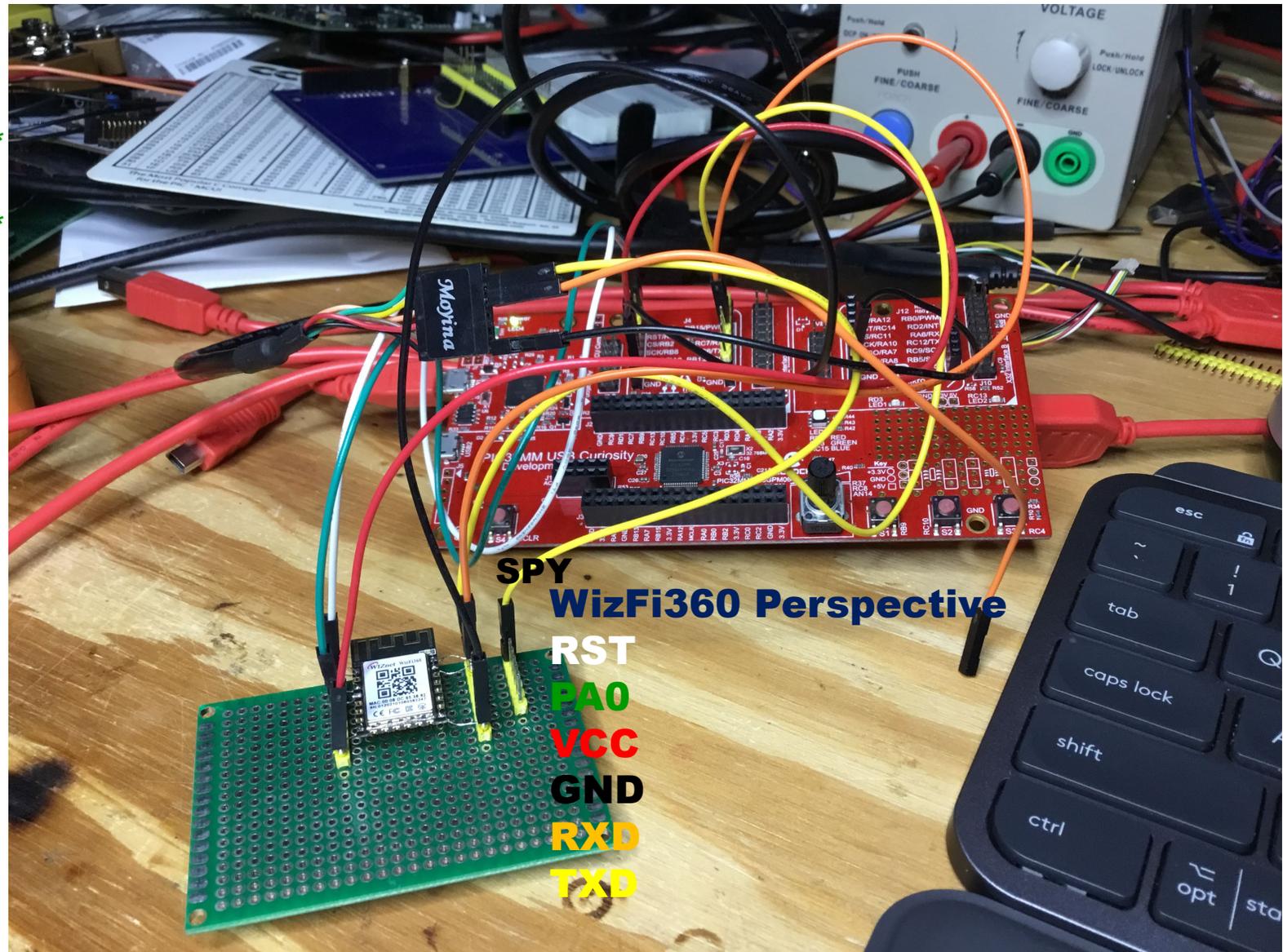
```

//*****
/* SET SINGLE CONNECTION MODE
//*****
uint8_t setSingleConnectionMode(void)
{
    rc = 0;
    UART2_ResetRxQueue();
    indx = 0;

    printf("AT+CIPMUX=0\r\n");

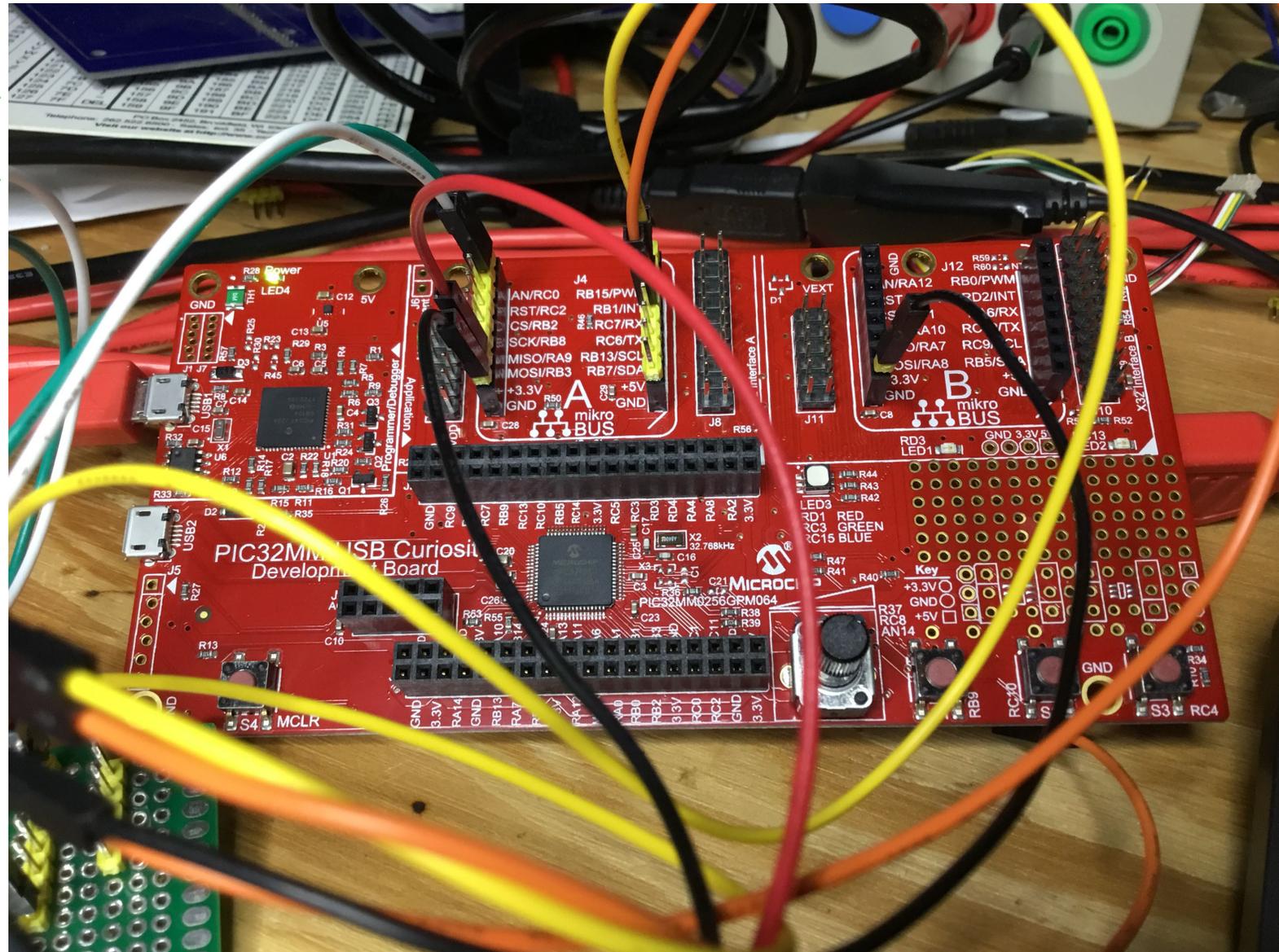
    if(chk_atok())
    {
        rc = 1;
    }
    return rc;
}

```



## DHCP Enable Function

```
/**  
** SET DHCP ENABLE  
**  
uint8_t setDhcpEnable(void)  
{  
    rc = 0;  
    UART2_ResetRxQueue();  
    indx = 0;  
  
    printf("AT+CWDHCP_CUR=1,1\r\n");  
    if(chk_atok())  
    {  
        rc = 1;  
    }  
    return rc;  
}
```



## Connect to the Access Point Function

```
//*****  
//* CONNECT TO THE AP  
//*****  
uint8_t connect2AP(void)  
{  
    uint8_t tmr;  
    rc = 0;  
    UART2_ResetRxQueue();  
    indx = 0;  
  
    printf("AT+CWJAP_CUR=\"edtpnet2\", \"password\"\r\n");  
    for(tmr=0;tmr<11;tmr++)  
    {  
        delay_ms(1000);  
    }  
  
    if(chk_atok())  
    {  
        rc = 1;  
    }  
    return rc;  
}
```



## Connect to the Server Function

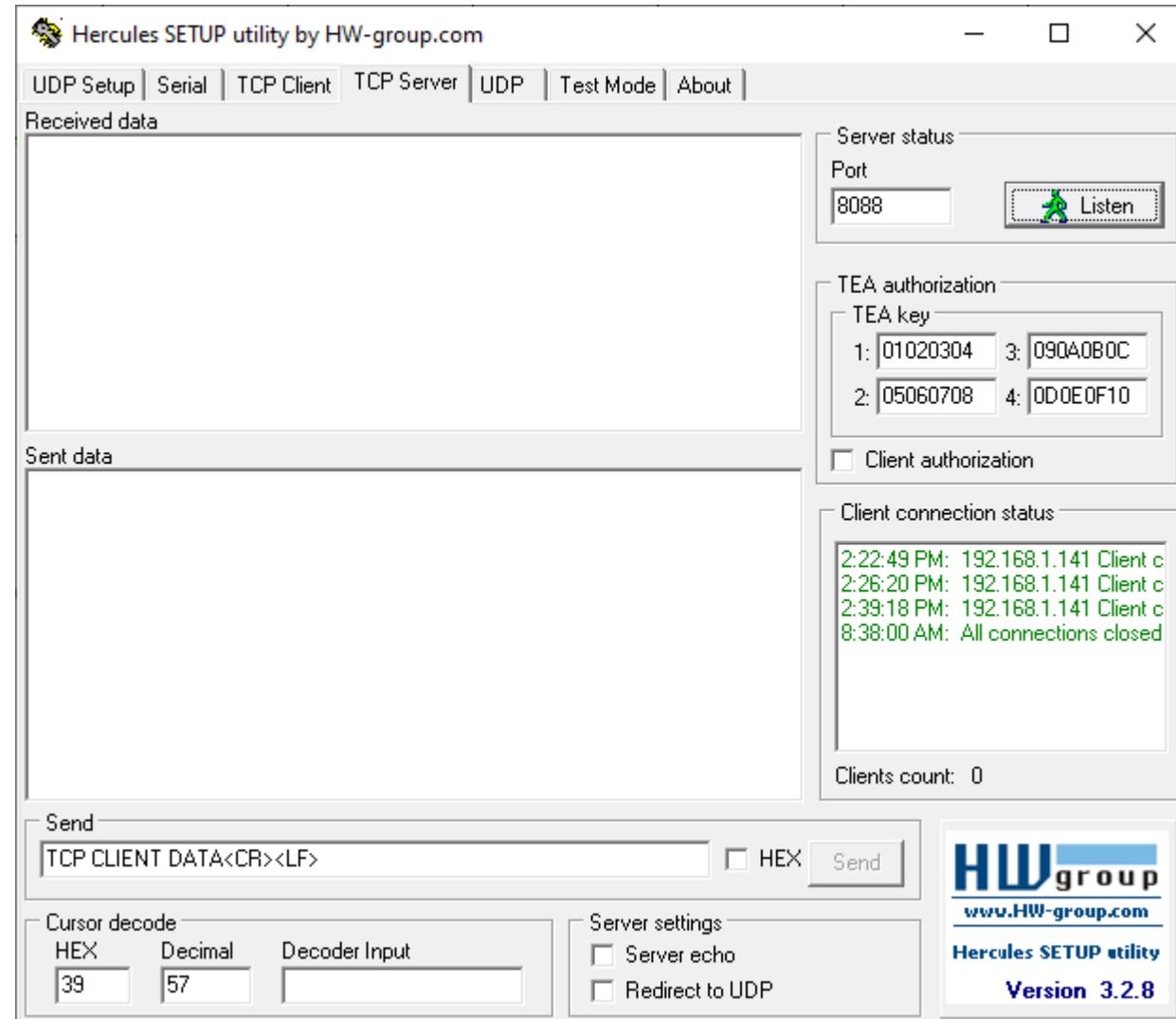
```

//*****
/* CONNECT TO THE SERVER
//*****
uint8_t connect2Server(void)
{
    rc = 0;
    UART2_ResetRxQueue();
    indx = 0;

    printf("AT+CIPSTART=\"TCP\", \"192.168.1.235\", 8088\r\n");
    delay_ms(2000);

    if(chk_atok())
    {
        rc = 1;
    }
    return rc;
}

```



Hercules SETUP utility by HW-group.com

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

Received data

Sent data

Server status

Port: 8088 Listen

TEA authorization

TEA key

1: 01020304 3: 090A0B0C

2: 05060708 4: 0D0E0F10

Client authorization

Client connection status

2:22:49 PM: 192.168.1.141 Client c  
2:26:20 PM: 192.168.1.141 Client c  
2:39:18 PM: 192.168.1.141 Client c  
8:38:00 AM: All connections closed

Clients count: 0

Send

TCP CLIENT DATA<CR><LF>  HEX Send

Cursor decode

HEX	Decimal	Decoder Input
39	57	

Server settings

Server echo  
 Redirect to UDP

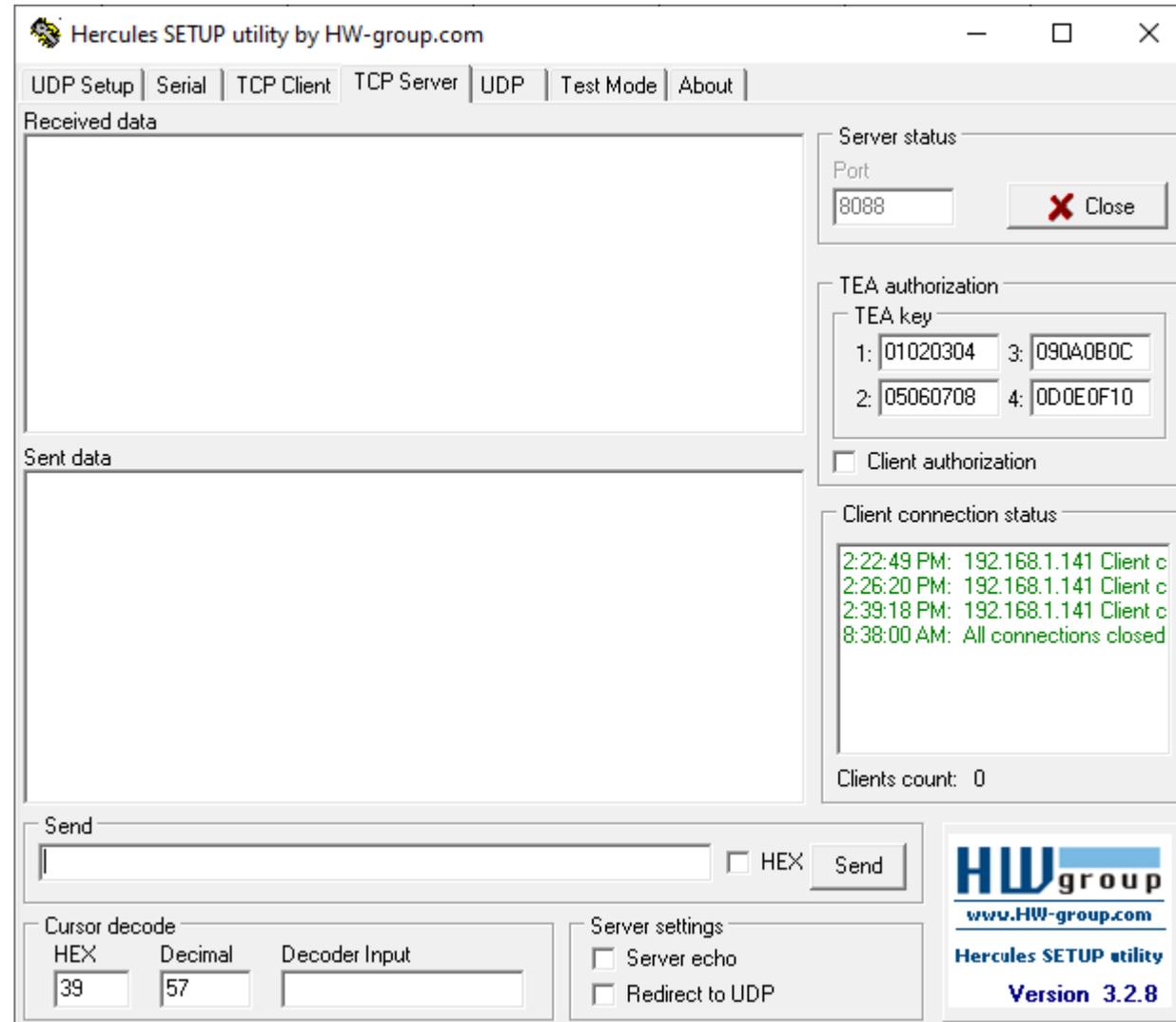
HWgroup  
www.HW-group.com  
Hercules SETUP utility  
Version 3.2.8

## Send to the Server Function

```

//*****
//* SEND DATA TO THE SERVER
//*****
uint8_t send2Server(void)
{
    uint8_t bite = 0x30;
    rc = 0;
    UART2_ResetRxQueue();
    indx = 0;
    printf("AT+CIPSEND=10\r\n");
    delay_ms(1000);
    do{
        if(UART2_IsRxReady())
        {
            rxBuf[indx++] = UART2_Read();
        }
    }while( UART2_IsRxReady());
    if(rxBuf[indx-2] == '>')
    {
        indx = 0;
        for(scratch8=0;scratch8<10;scratch8++)
        {
            printf("%c",bite++);
        }
        delay_ms(1000);
        do{
            if( UART2_IsRxReady())
            {
                rxBuf[indx++] = UART2_Read();
            }
        }while( UART2_IsRxReady());
        indx -= 3;
        if(rxBuf[indx] == 'K')
        {
            rc = 1;
        }
    }
    return rc;
}

```



## Main Function

```

//*****
//* MAIN FUNCTION
//*****
int main()
{
    // initialize the device
    SYSTEM_Initialize();

    okat = 0x00;
    indx = 0x00;
    IO_RC2_RST_SetLow();
    IO_RC0_WP_SetLow();
    delay_ms(100);
    IO_RC2_RST_SetHigh();
    delay_ms(100);
    while(!UART2_IsRxReady());
    delay_ms(100);
    do{
        rxBuf[indx++] = UART2_Read();
    }while(UART2_IsRxReady());
}

```

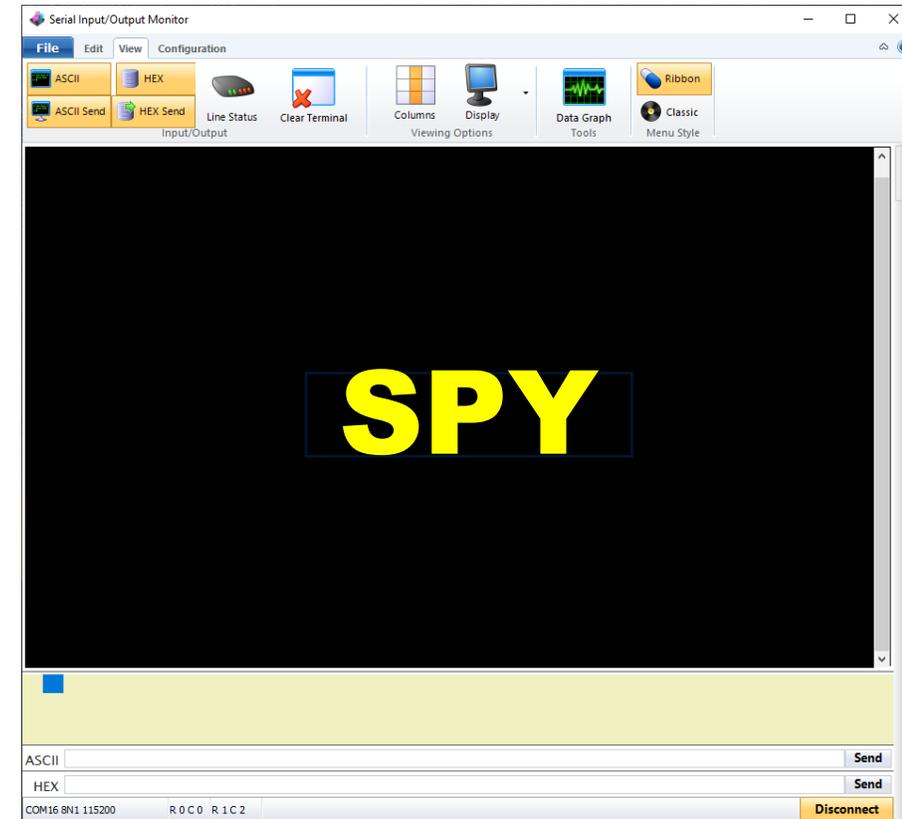
```

//*****
//* MAIN FUNCTION
//*****
do{
    indx = 0;
    memset(rxBuf,0x00,sizeof(rxBuf));

    printf("AT\r\n");
    while(!UART2_IsRxReady());
    delay_ms(100);
    do{
        rxBuf[indx++] = UART2_Read();
    }while(UART2_IsRxReady());

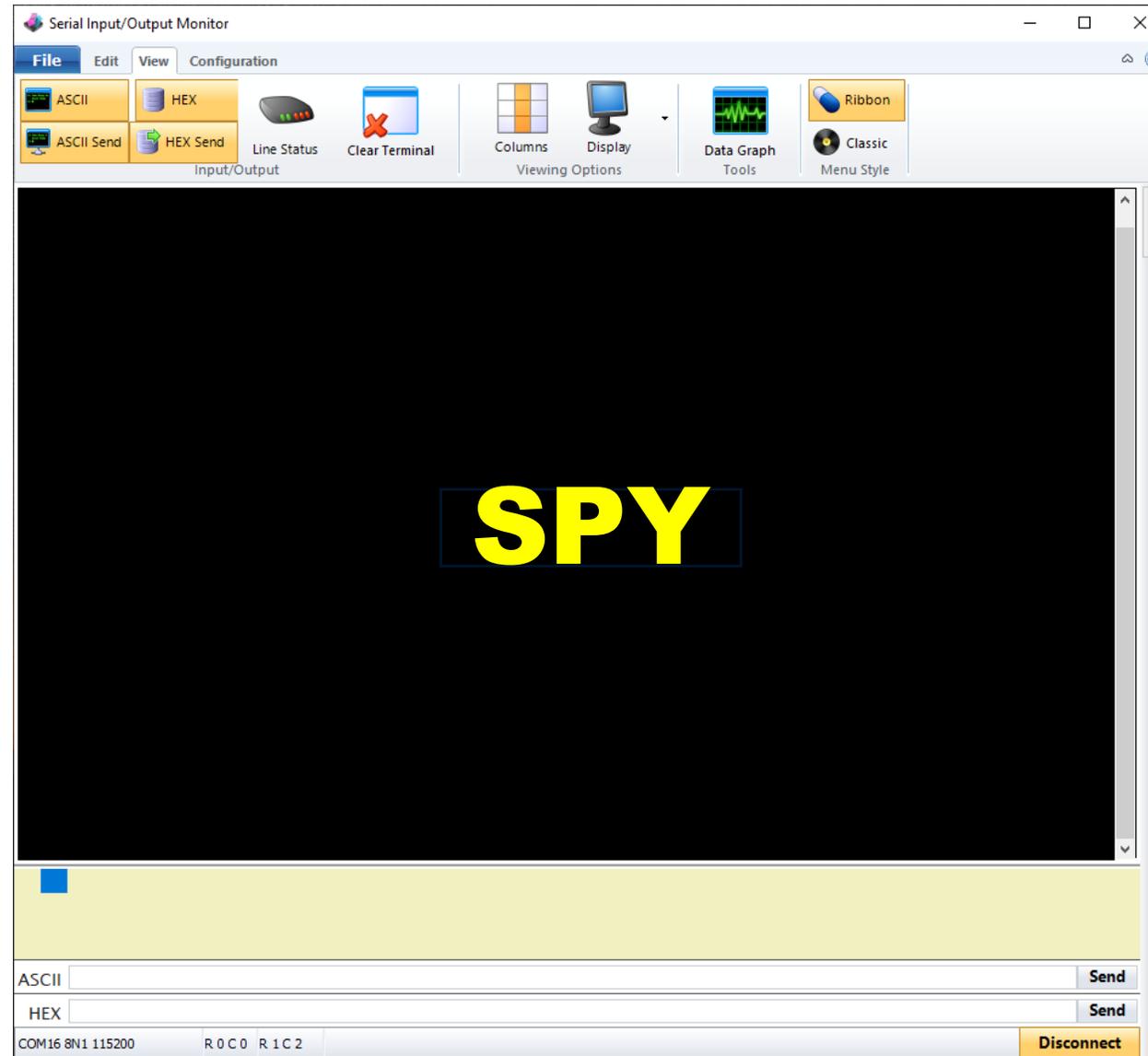
    indx -= 3;
    if(rxBuf[indx] == 'K')
    {
        okat = 1;
    }
}while(okat == 0);
}

```

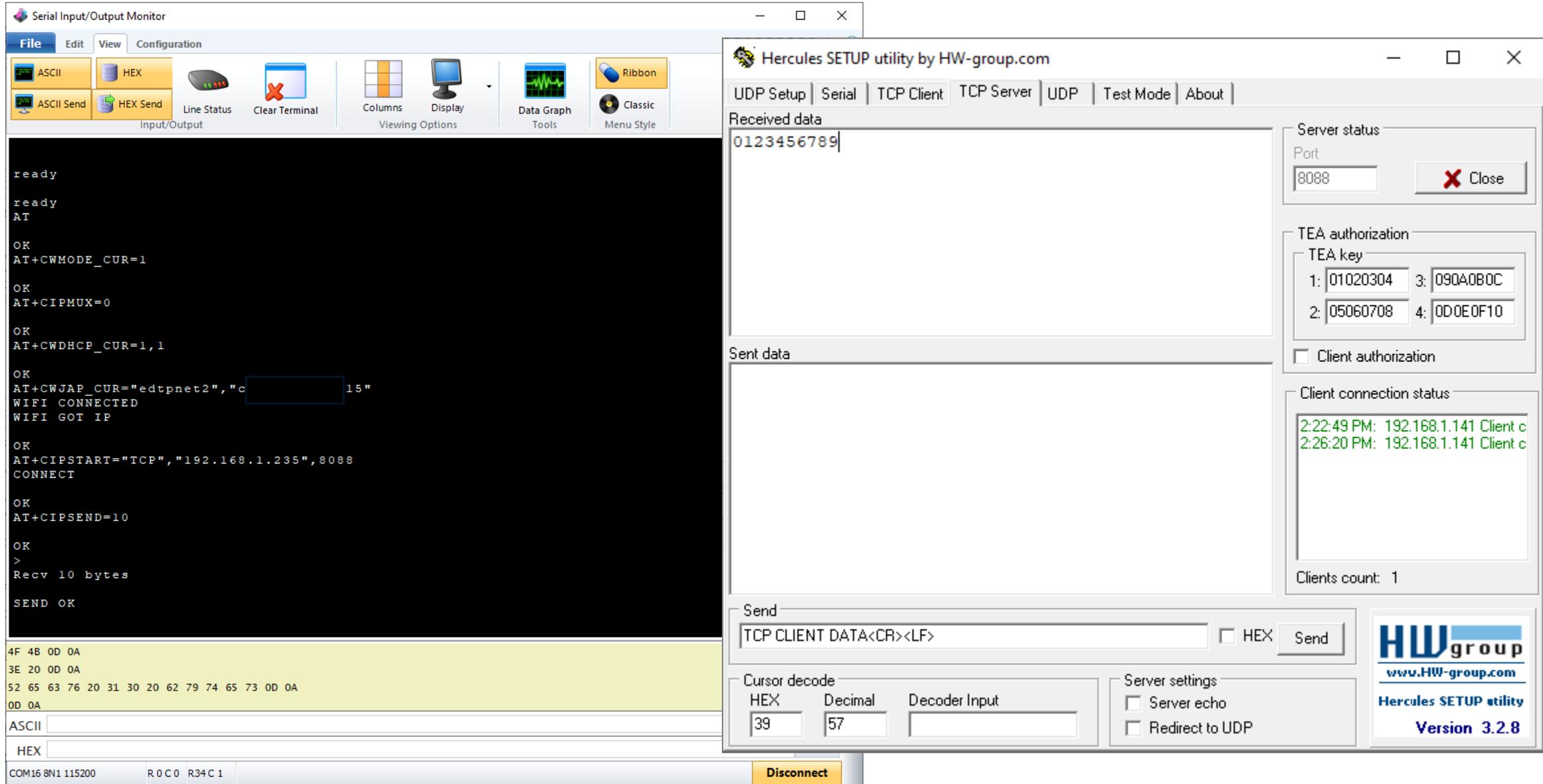


## Main Function

```
if(!setStationMode())  
{  
  do{  
    delay_ms(200);  
    IO_RD3_LED_Toggle();  
  }while(1);  
}  
if(!setSingleConnectionMode())  
{  
}  
if(!setDhcpEnable())  
{  
}  
if(!connect2AP())  
{  
}  
if(!connect2Server())  
{  
}  
if(!send2Server())  
{  
}
```



## Execute the Main Function



The image shows two overlapping windows. The background window is the 'Serial Input/Output Monitor' displaying AT commands and their responses. The foreground window is the 'Hercules SETUP utility by HW-group.com' showing network configuration options.

**Serial Input/Output Monitor Output:**

```

ready
ready
AT
OK
AT+CWMODE_CUR=1
OK
AT+CIPMUX=0
OK
AT+CWDHCP_CUR=1,1
OK
AT+CWJAP_CUR="edtpnet2","c[redacted]15"
WIFI CONNECTED
WIFI GOT IP
OK
AT+CIPSTART="TCP","192.168.1.235",8088
CONNECT
OK
AT+CIPSEND=10
OK
>
Recv 10 bytes

SEND OK
4F 4B 0D 0A
3E 20 0D 0A
52 65 63 76 20 31 30 20 62 79 74 65 73 0D 0A
OD 0A
ASCII [ ]
HEX [ ]
COM16 8N1 115200 R 0 C 0 R34 C 1
  
```

**Hercules SETUP utility Configuration:**

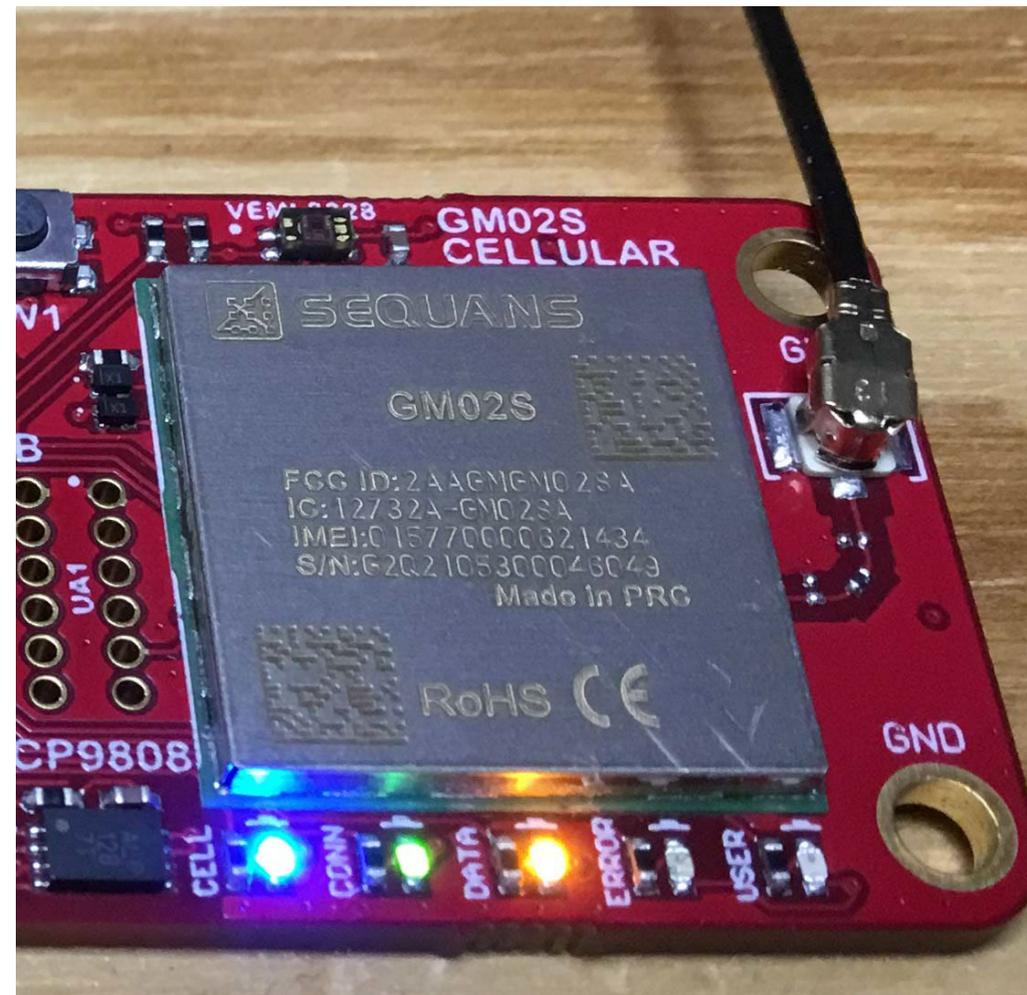
- Port: 8088
- TEA key: 1: 01020304, 2: 05060708, 3: 090A0B0C, 4: 0D0E0F10
- Client authorization:
- Client connection status: 2:22:49 PM: 192.168.1.141 Client c, 2:26:20 PM: 192.168.1.141 Client c
- Clients count: 1
- Send: TCP CLIENT DATA<CR><LF>
- Cursor decode: HEX 39, Decimal 57
- Server settings:  Server echo,  Redirect to UDP

**MORE TO COME..**

# Thank you for attending!!!

Please consider the resources below:

- [microchip.com](http://microchip.com)
- [wiznet.io](http://wiznet.io)
- [WizFi360 User Manual](#)





**DesignNews**

Thank You

Sponsored by

