CEC Continuing Education Center

DesignNews

IoT Device Prototyping with STMicroelectronics Nucleo Development Boards

Day 4:

Prototyping NUCLEO-U575ZI-Q and NUCLEO-G071RB LoRa Nodes

Sponsored by

Digi-Key ELECTRONICS

informa markets

Webinar Logistics

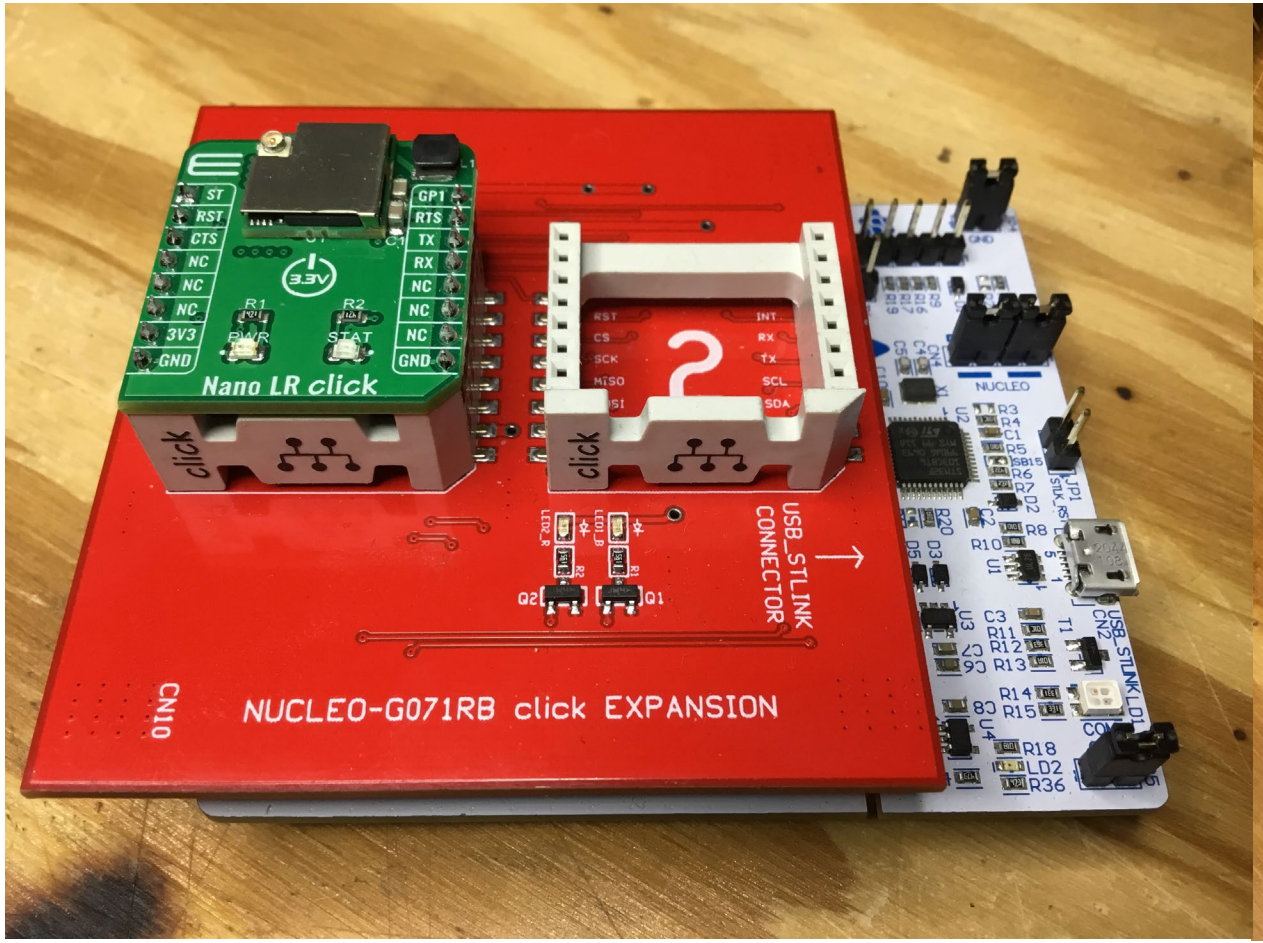
- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Attendee Chat’ by maximizing the chat widget in your dock.

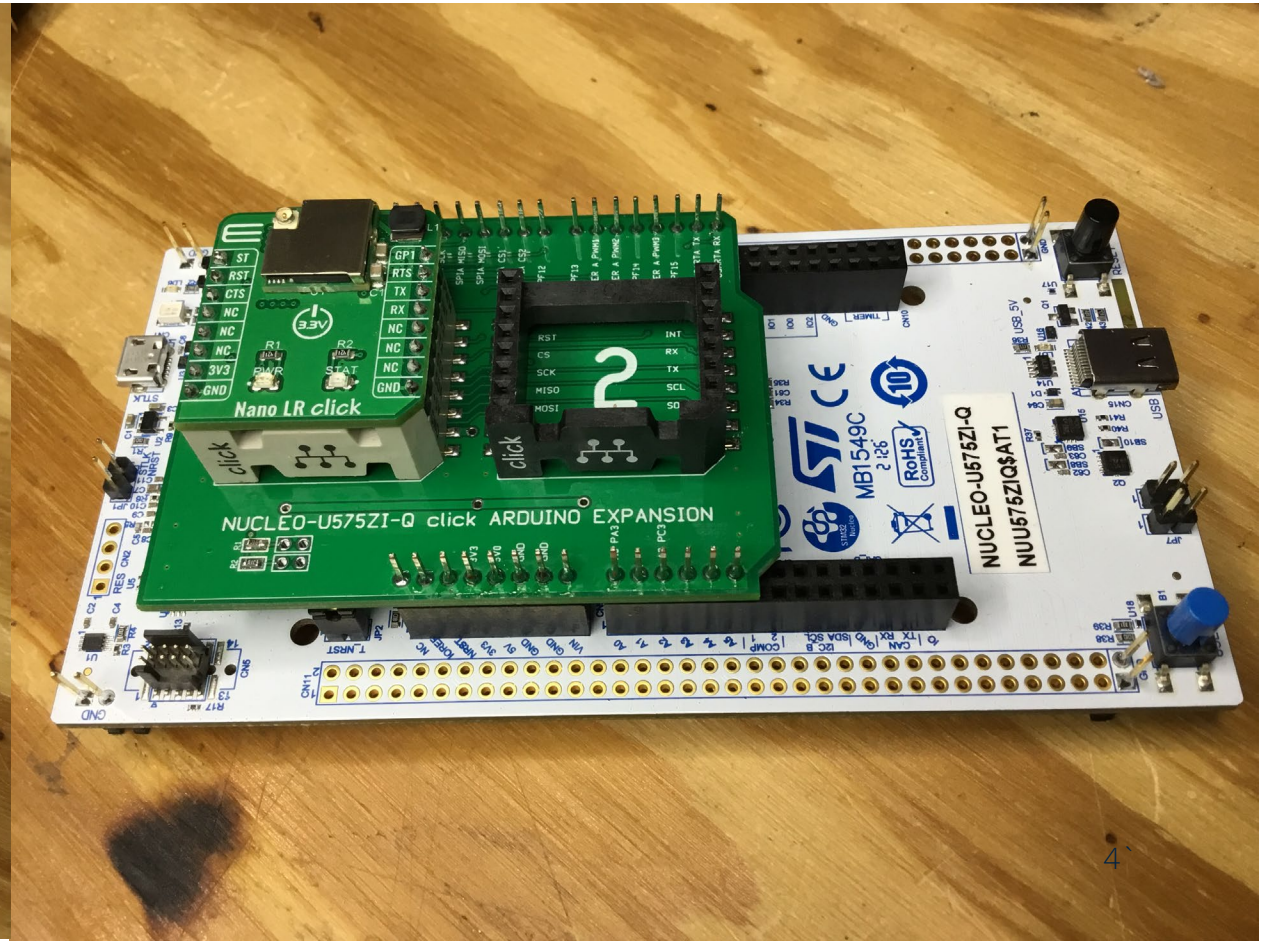
Fred Eady

Visit ‘Lecturer Profile’ in your console for more details.

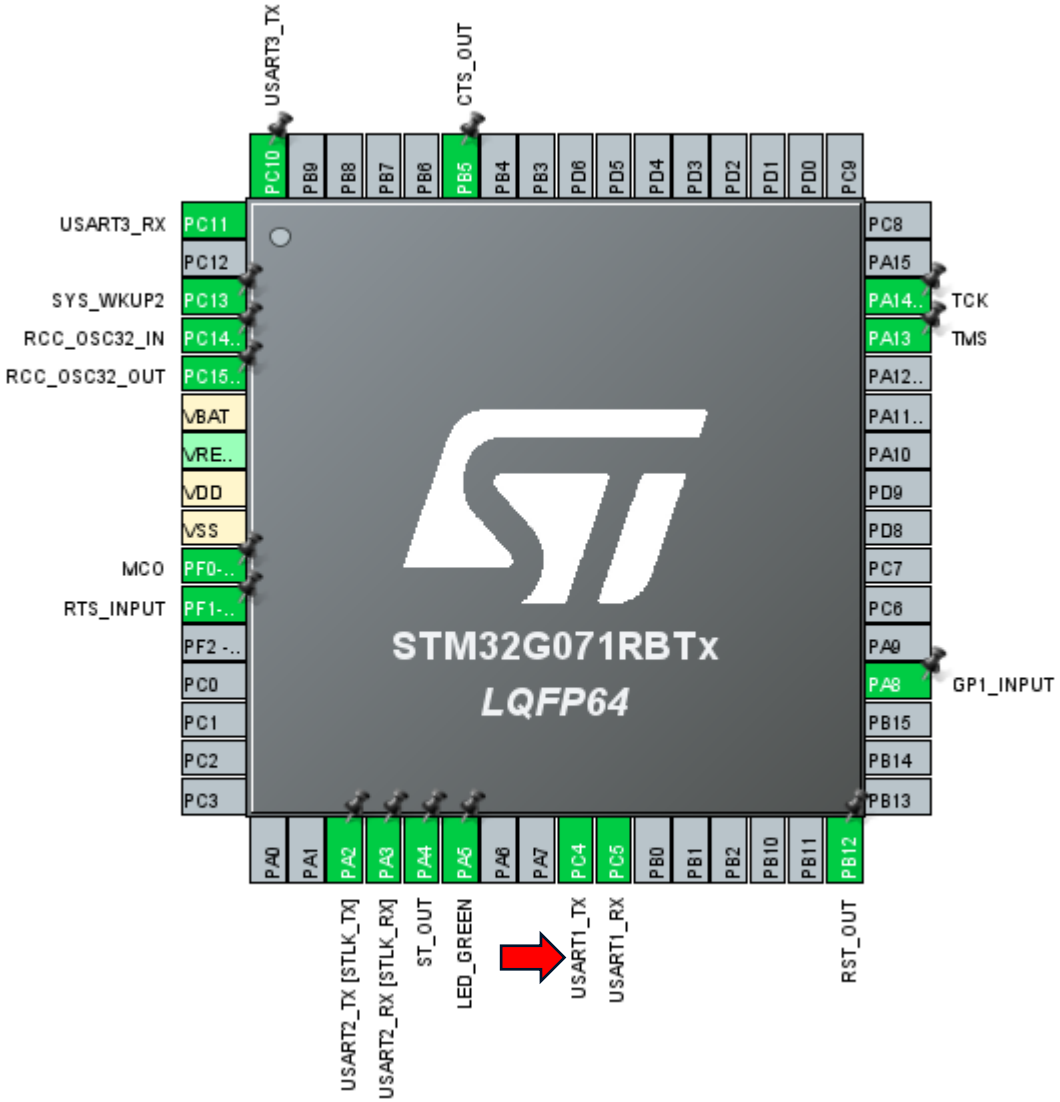
AGENDA

- **NUCLEO-G071RB LoRa Driver**
- **NUCLEO-U575ZI-Q LoRa Driver**

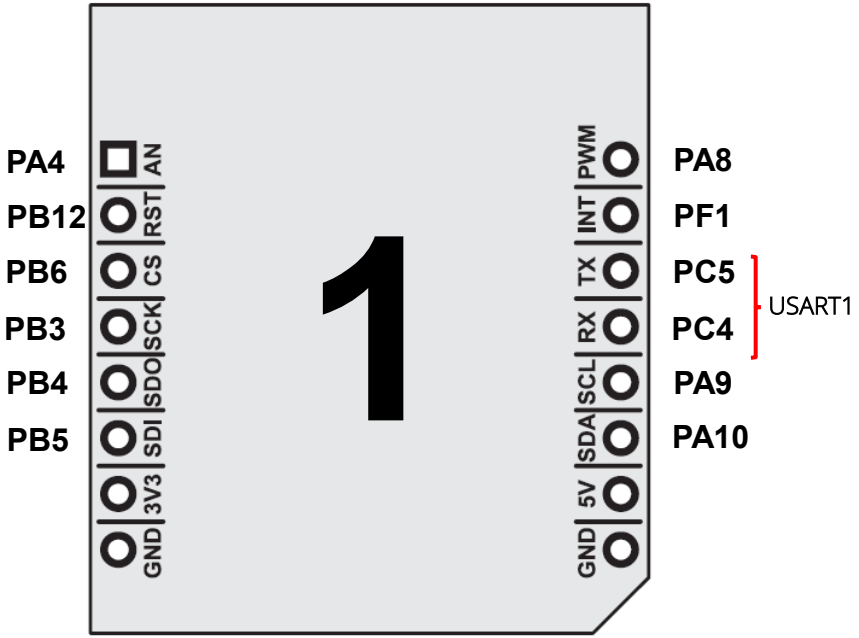


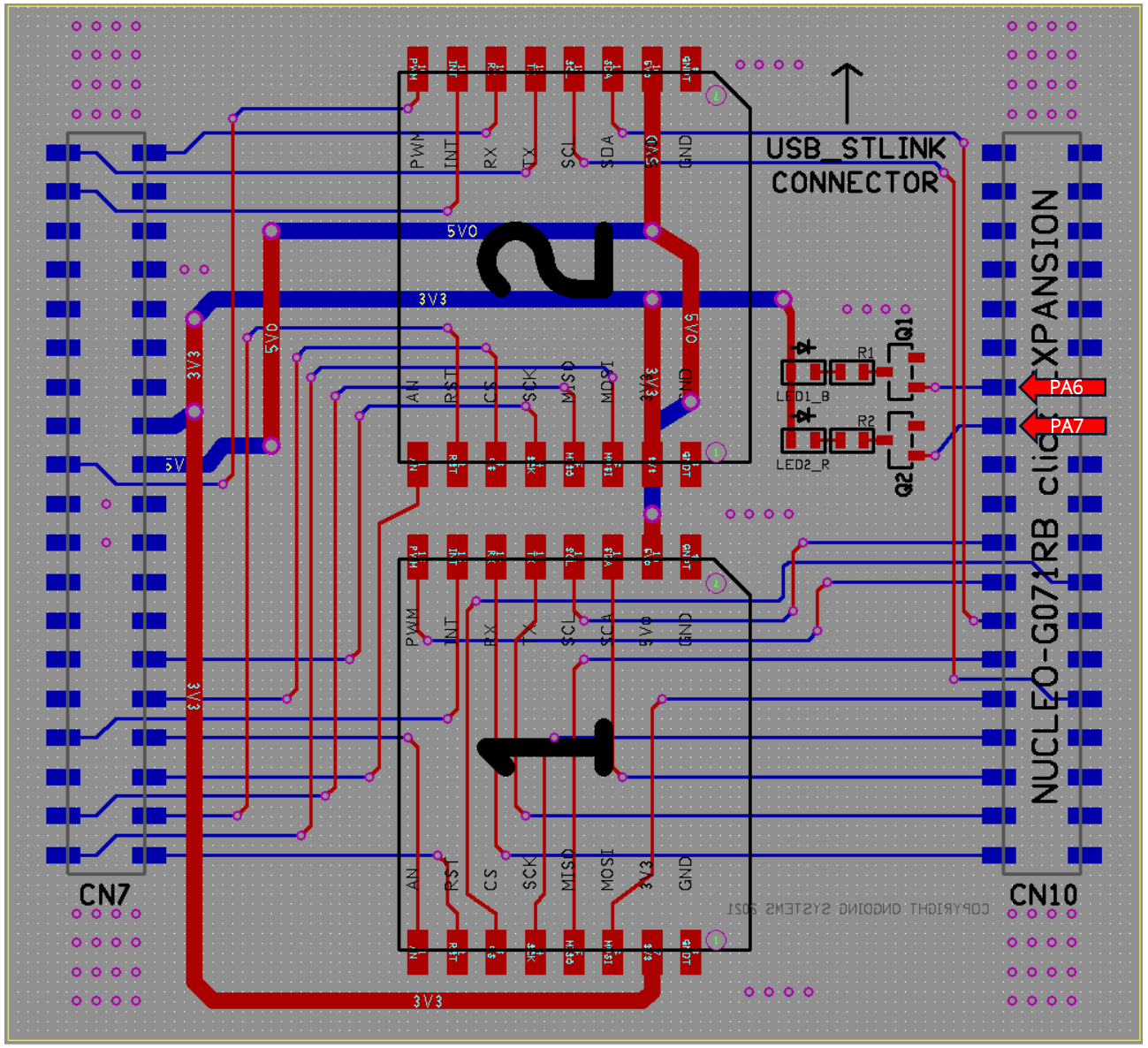


NUCLEO-G071RB LoRa Driver: Hardware Configuration

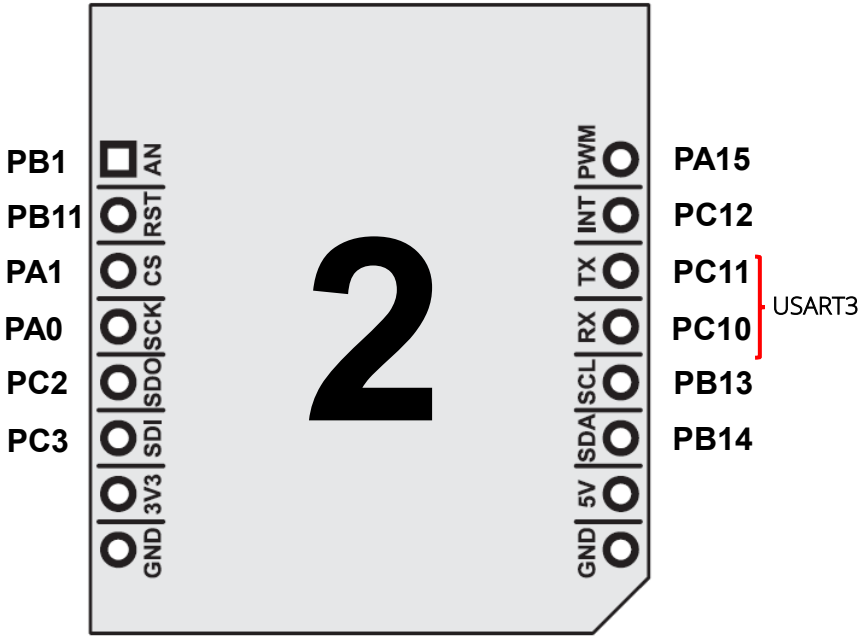


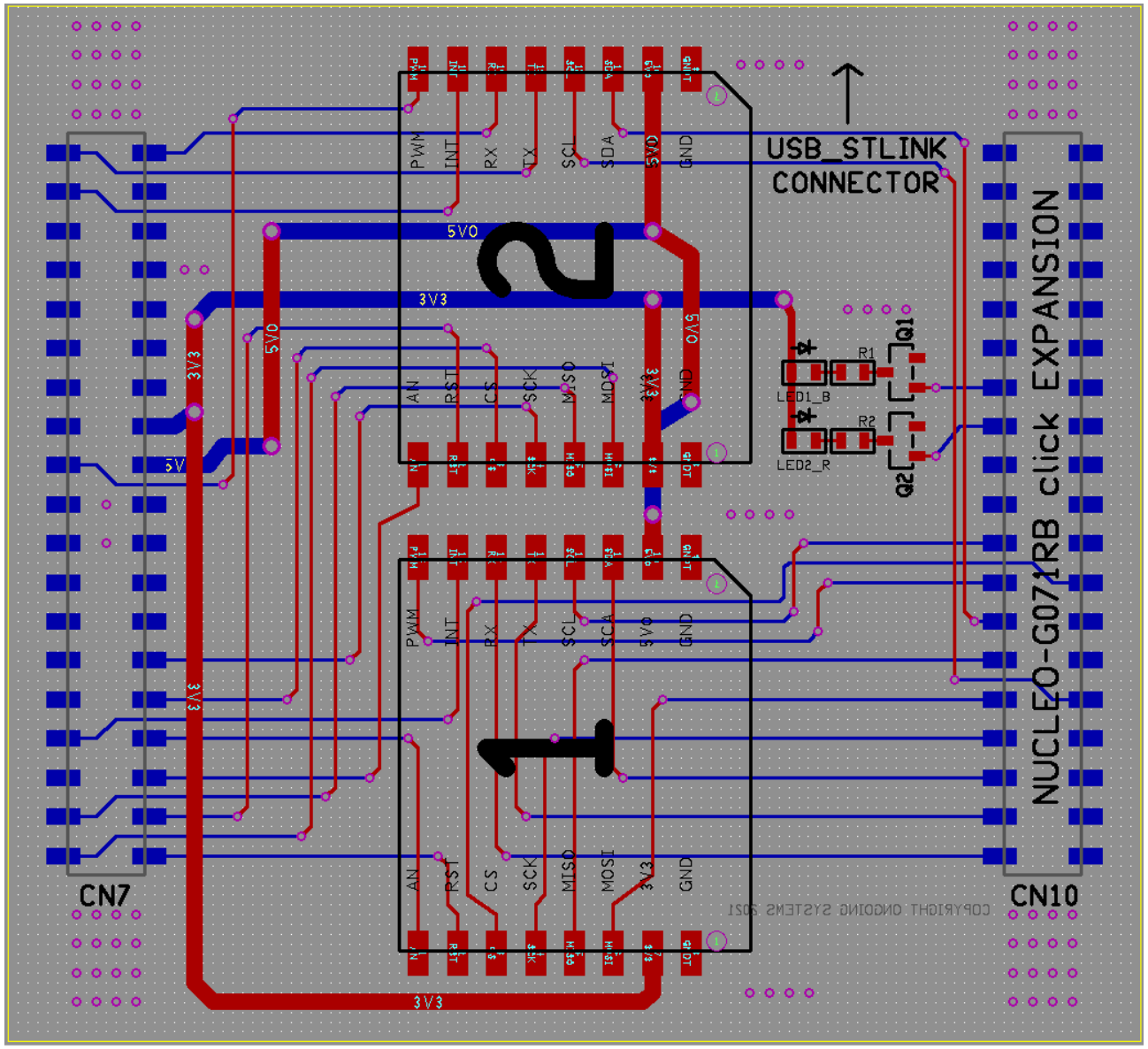
NUCLEO-G071RB LoRa Driver: Hardware Configuration



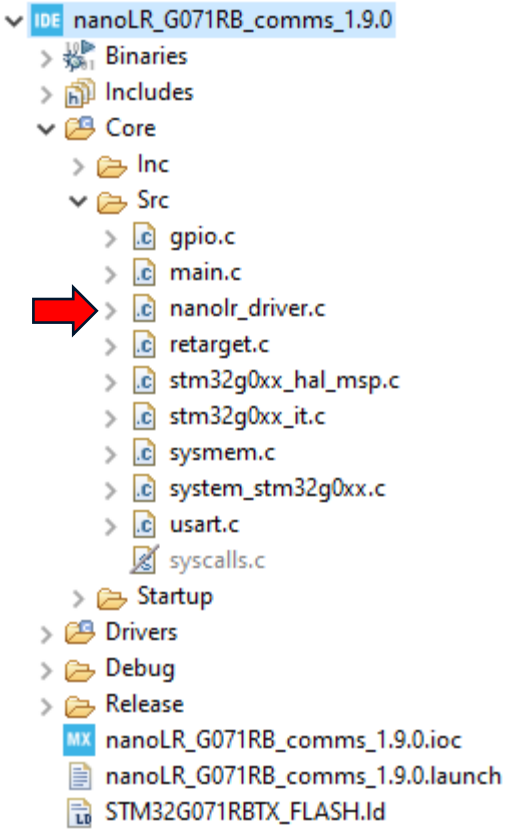


NUCLEO-G071RB LoRa Driver: Hardware Configuration

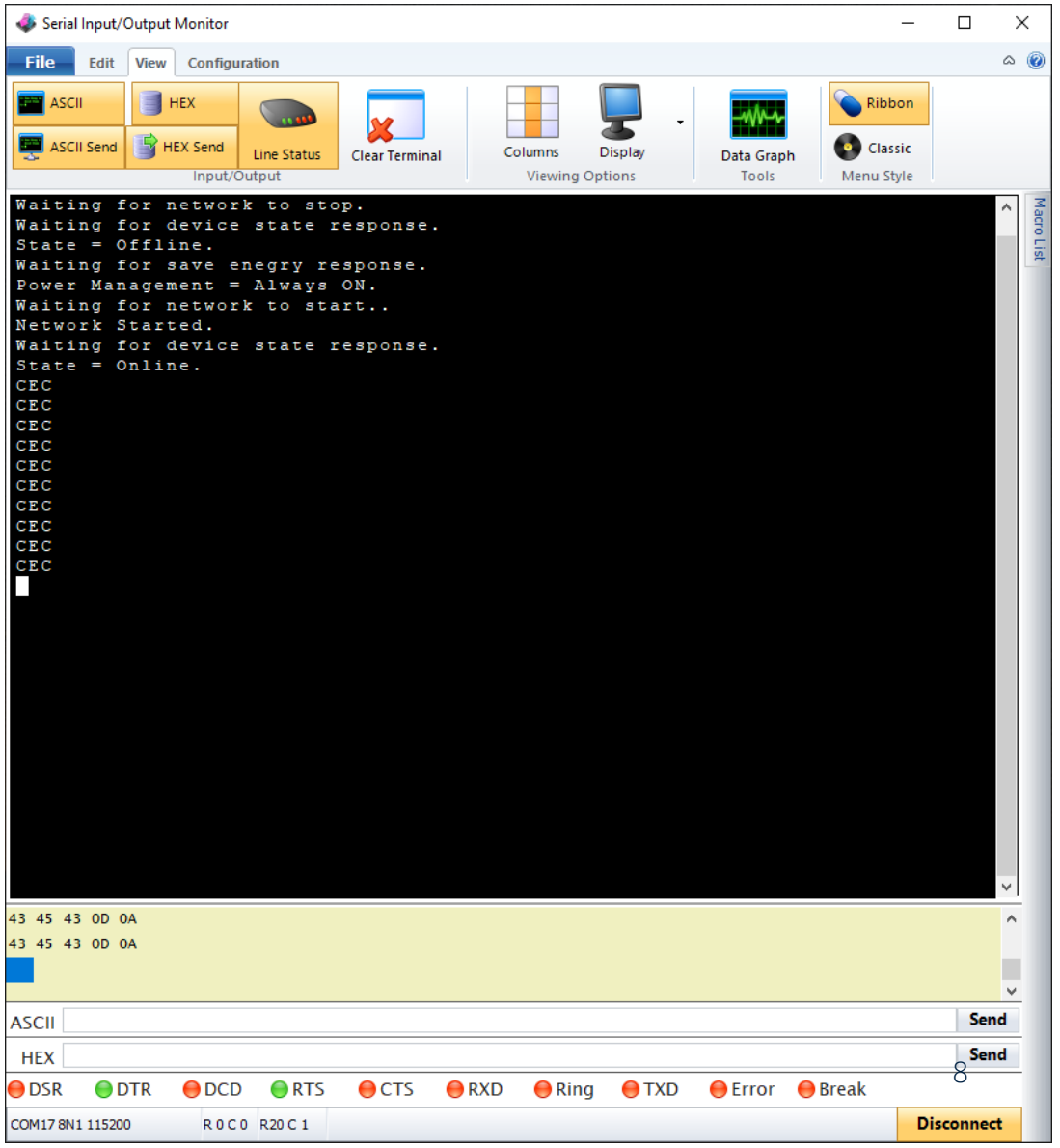




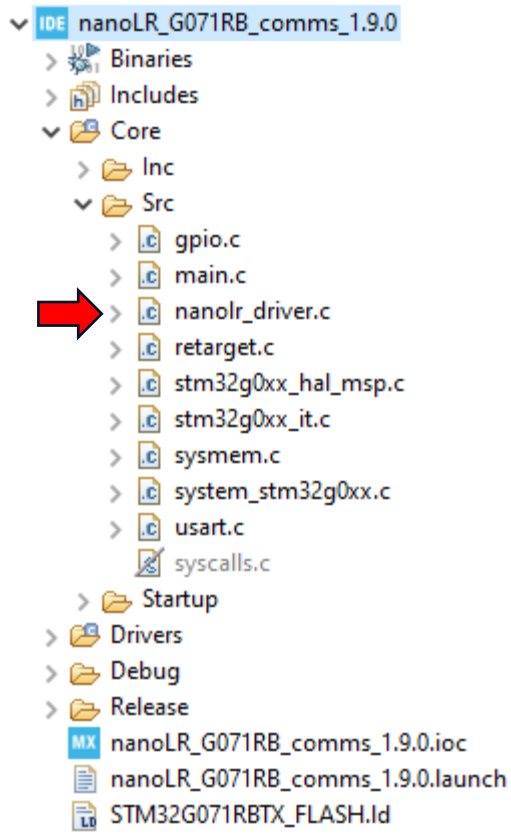
NUCLEO-G071RB LoRa Driver: nanolr_driver.c



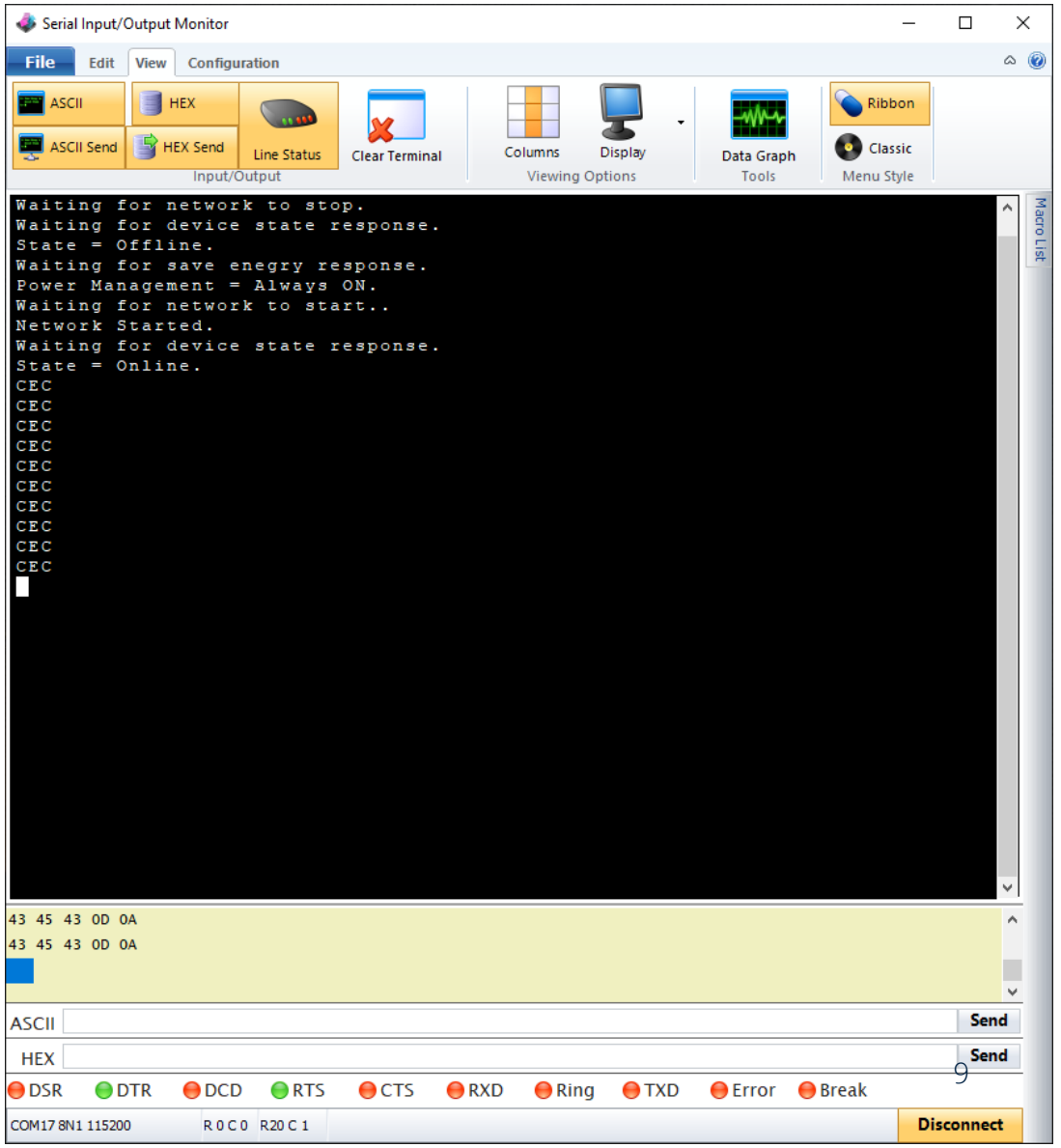

```c
void nanolr_init (void)
{
  uint8_t ticktocks;

  //set the ST pin and perform a hardware reset
  nanolr_set_st_pin(0);
  nanolr_reset();
  __HAL_UART_DISABLE_IT(&huart1,UART_IT_RXNE);
  USART1_RxHead = 0x00;
  USART1_RxTail = 0x00;
  __HAL_UART_ENABLE_IT(&huart1,UART_IT_RXNE);
  //send stop network command
  nanolr_network_stop();
  //wait for command response with 10 sec timeout
  HAL_Delay(100);
  printf("Waiting for network to stop.\r\n");
  ticktocks = 10;
  do{
     HAL_Delay(1000);
     ticktocks--;
  }while(!CharInRing() && ticktocks > 0);
  if (ticktocks == 0)
  {
    printf("Network stop timeout.\r\n");
    while(1);
  }
  //response received
  memset(rxBuf,0x00,sizeof(rxBuf));
  rxBuf[0] = readring();
  rxBuf[1] = readring();
  rxBuf[2] = readring();
  nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
  nanolr_pktlen &= 0xFF;
  for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
  {
    rxBuf[indx] = readring();
  }
  //check response checksum and get device state
  if(nanolr_response_parser() != 0)
  {
    printf("Network stop error.\r\n");
    while(1);
  }
```



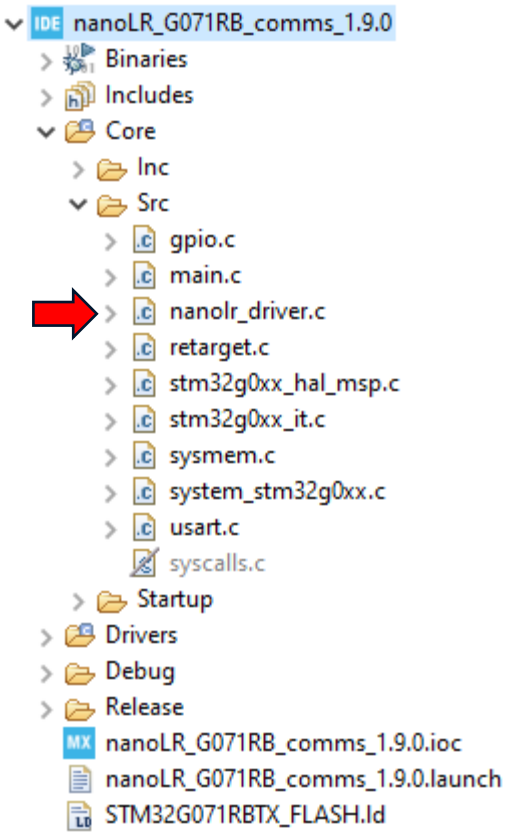
NUCLEO-G071RB LoRa Driver: nanolr_driver.c



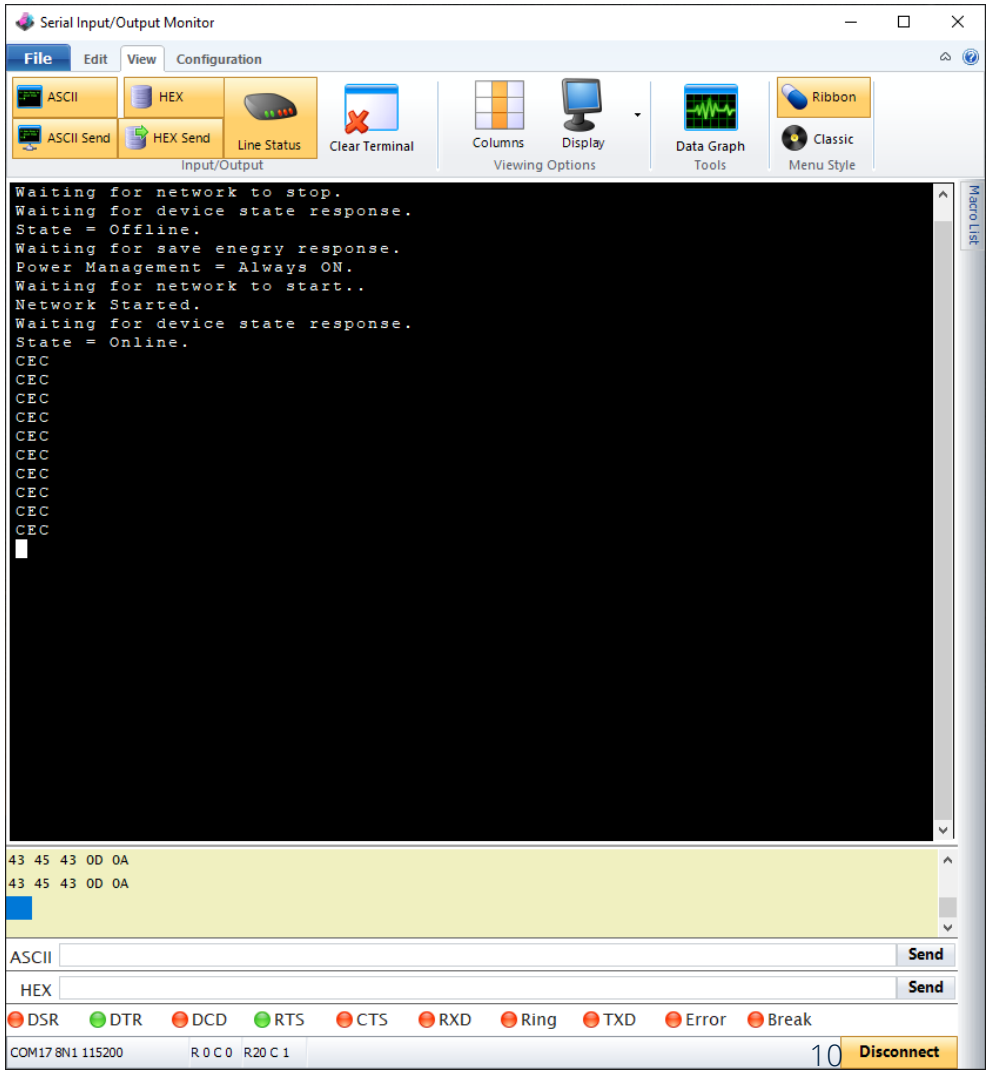

```c
//send get device state command
nanolr_get_device_state();
HAL_Delay(100);
//wait for command response with 10 sec timeout
printf("Waiting for device state response.\r\n");
ticktocks = 10;
do{
    HAL_Delay(1000);
    ticktocks--;
  }while(!CharInRing() && ticktocks > 0);
if (ticktocks == 0)
{
  printf("Get device state timeout.\r\n");
  while(1);
}
//device state data received
memset(rxBuf,0x00,sizeof(rxBuf));
rxBuf[0] = readring();
rxBuf[1] = readring();
rxBuf[2] = readring();
nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
nanolr_pktlen &= 0xFF;
for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
{
  rxBuf[indx] = readring();
}
if(nanolr_response_parser() != 0)
{
  printf("State request response error.\r\n");
  while(1);
}
  if(responsePkt.payload[0] == 0x20)
{
  printf("State = Offline.\r\n");
}
else
{
  printf("State = 0x%02X.\r\n",responsePkt.payload[0]);
}
```



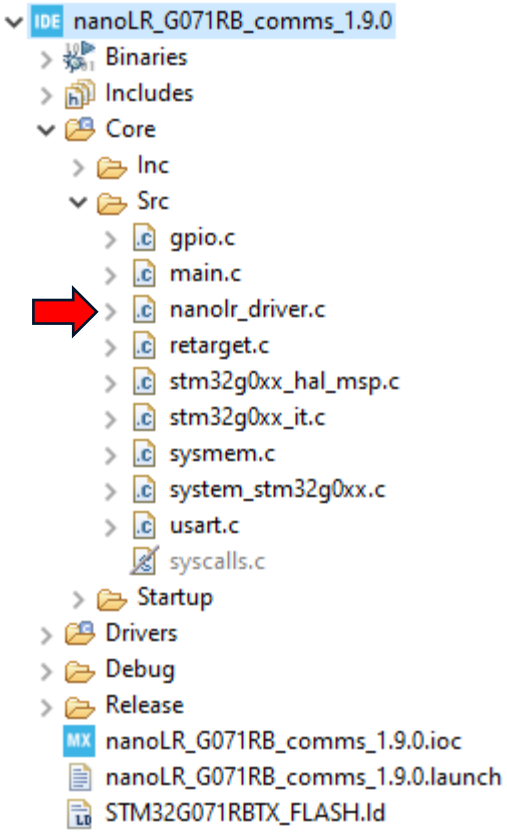
NUCLEO-G071RB LoRa Driver: nanolr_driver.c



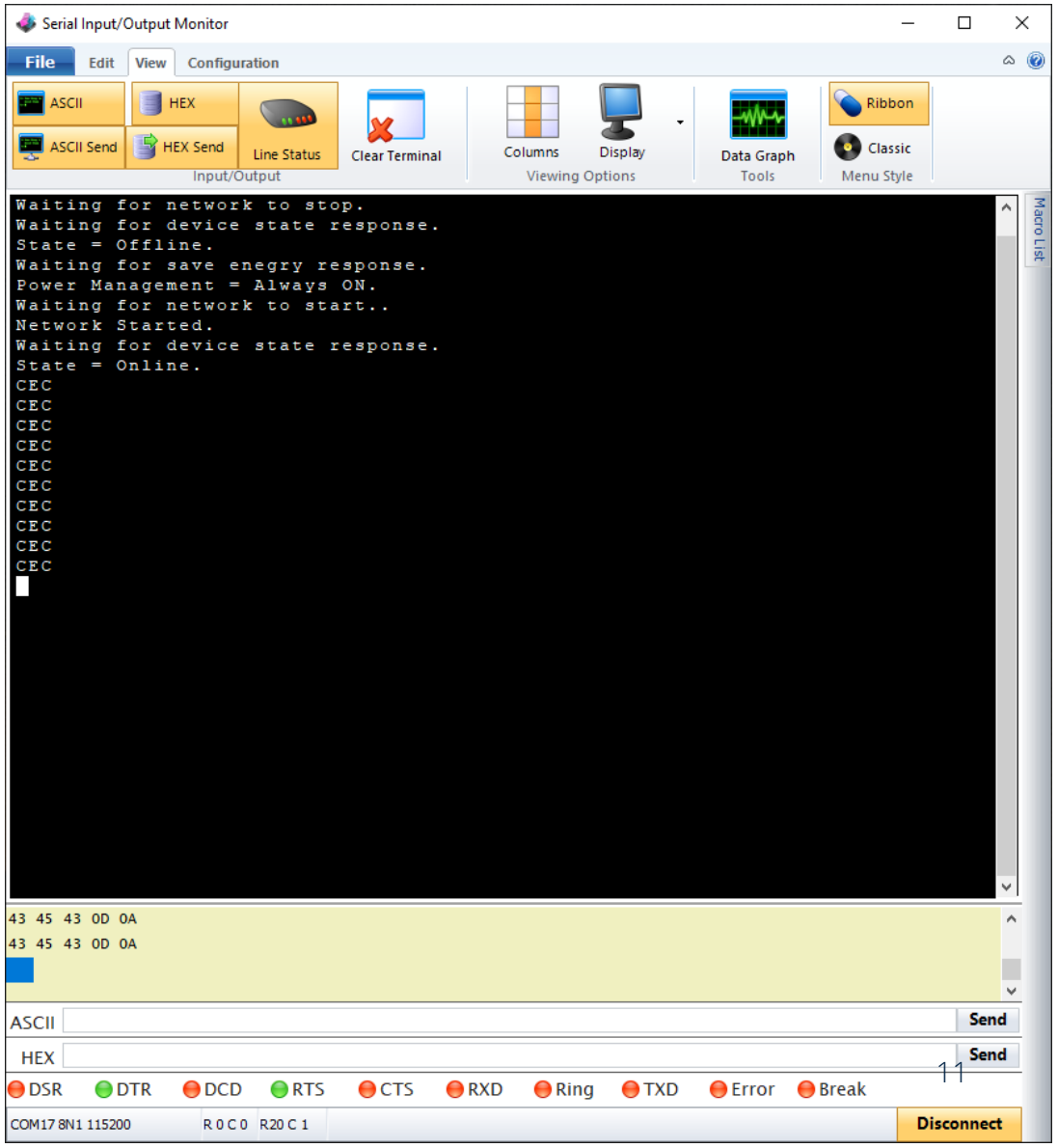

```c
//send always on command
nanolr_save_energy_always_on();
HAL_Delay(100);
//wait for command response with 10 sec timeout
printf("Waiting for save enegry response.\r\n");
ticktocks = 10;
do{
    HAL_Delay(1000);
    ticktocks--;
  }while(!CharInRing() && ticktocks > 0);
if (ticktocks == 0)
{
  printf("Save energy response timeout.\r\n");
  while(1);
}
//save energy response received
memset(rxBuf,0x00,sizeof(rxBuf));
rxBuf[0] = readring();
rxBuf[1] = readring();
rxBuf[2] = readring();
nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
nanolr_pktlen &= 0xFF;
for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
{
  rxBuf[indx] = readring();
}
if(nanolr_response_parser() != 0)
{
  printf("Save energy response error.\r\n)");
  while(1);
}
if(responsePkt.payload[0] == 0)
{
  printf("Power Management = Always ON.\r\n");
}
else
{
  printf("Power Management =     0x%02X.\r\n",responsePkt.payload[0]);
}
```



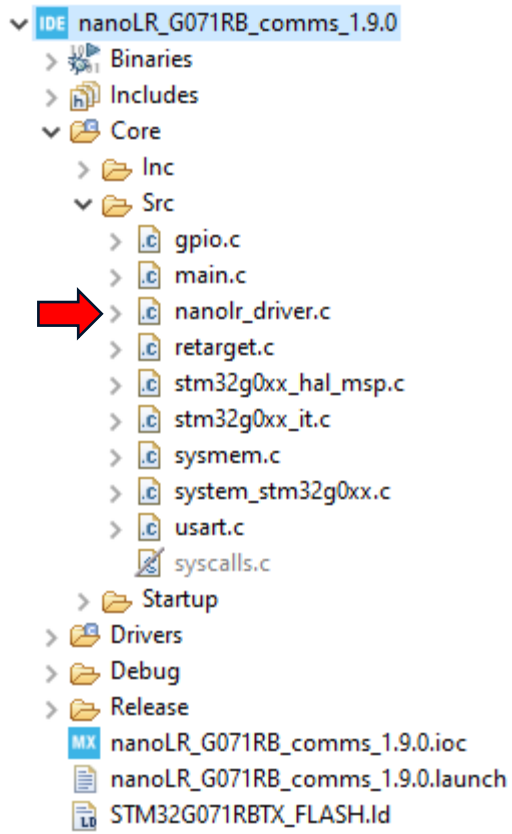
NUCLEO-G071RB LoRa Driver: nanolr_driver.c



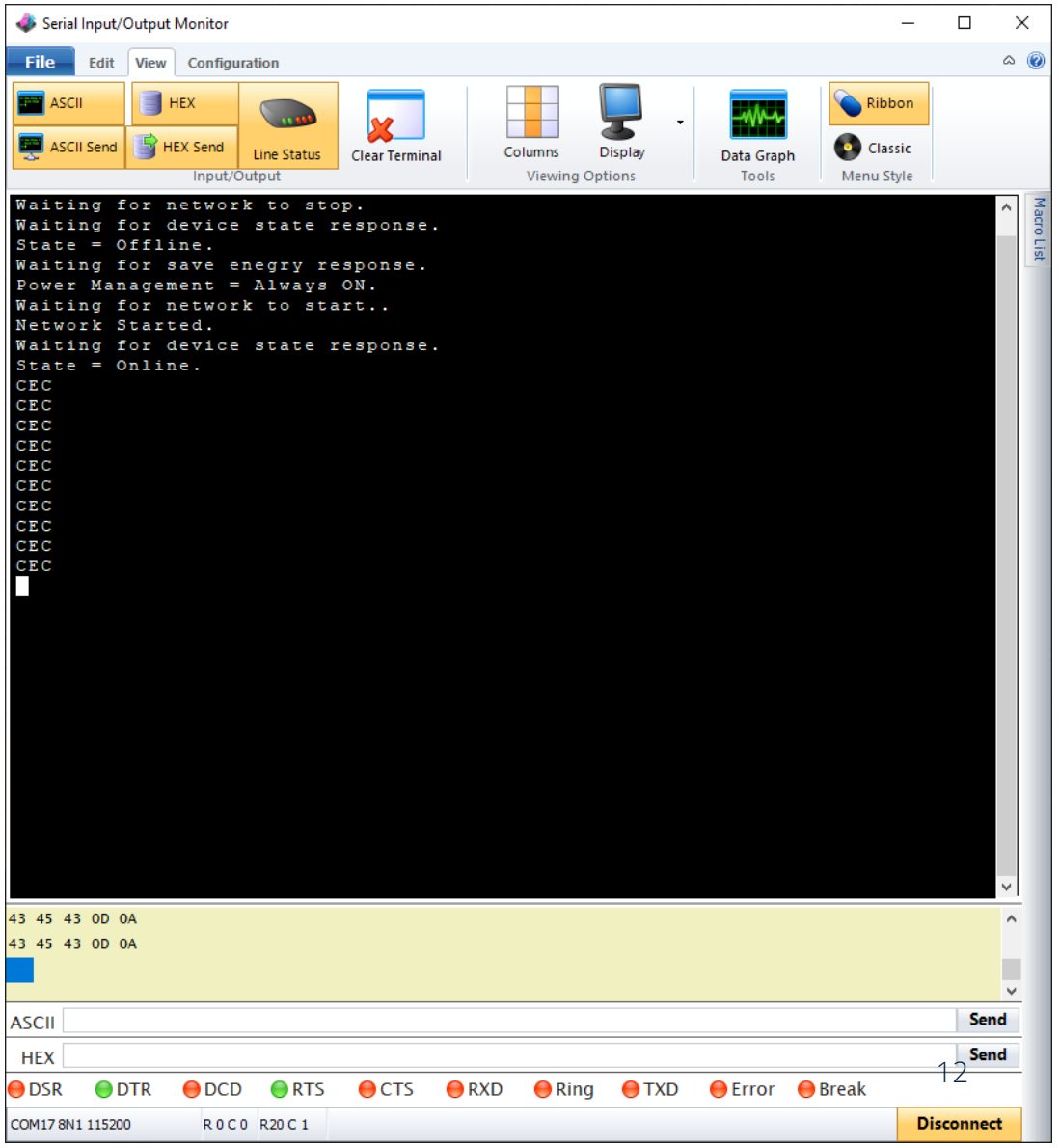

```
//send network start command
nanolr_network_start();
HAL_Delay(100);
//wait for command response with 10 sec timeout
printf("Waiting for network to start..\r\n");
ticktocks = 10;
do{
    HAL_Delay(1000);
    ticktocks--;
  }while(!CharInRing() && ticktocks > 0);
if (ticktocks == 0)
{
  printf("Network start timeout.\r\n");
  while(1);
}
//network start response received
memset(rxBuf,0x00,sizeof(rxBuf));
rxBuf[0] = readring();
rxBuf[1] = readring();
rxBuf[2] = readring();
nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
nanolr_pktlen &= 0xFF;
for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
{
  rxBuf[indx] = readring();
}
if(nanolr_response_parser() != 0)
{
  printf("Start response error.\r\n");
  while(1);
}
if(responsePkt.payload[0] == 0)
{
  printf("Network Started.\r\n");
}
else
{
  printf("Network start = 0x%02X.\r\n",responsePkt.payload[0]);
}
```



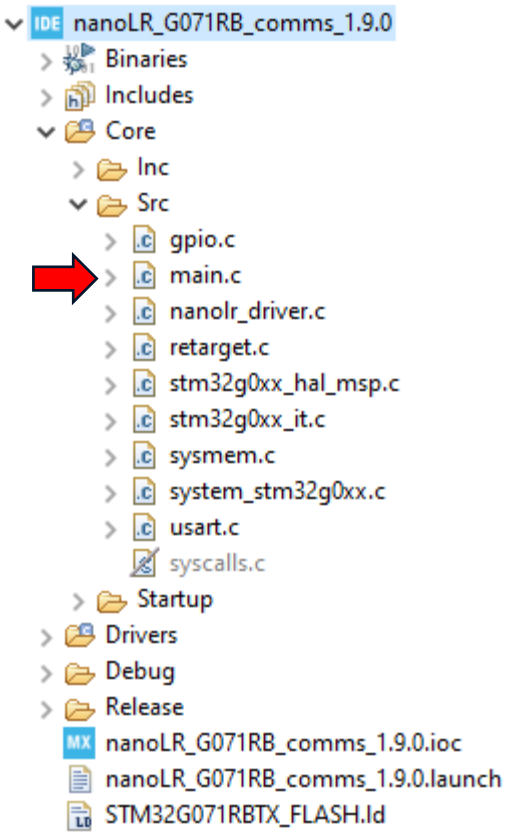
NUCLEO-G071RB LoRa Driver: nanolr_driver.c



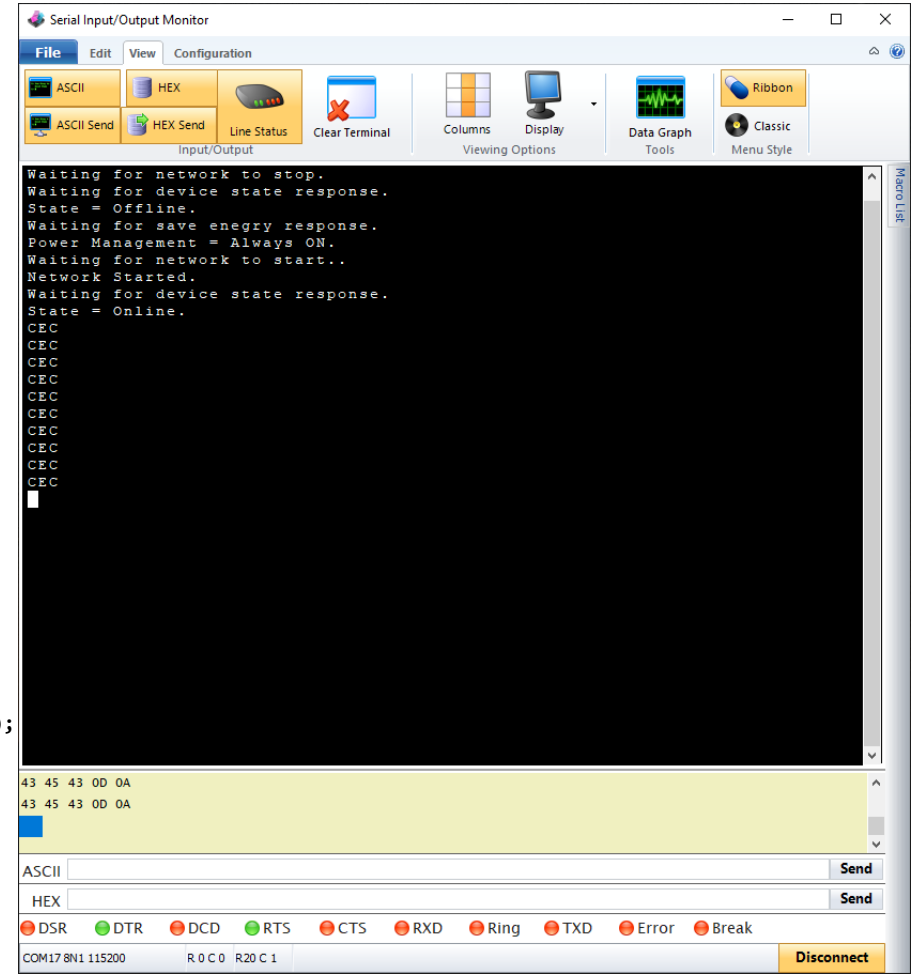

```
//send get device state command
 nanolr_get_device_state();
 HAL_Delay(100);
//wait for command response with 10 sec timeout
printf("Waiting for device state response.\r\n");
ticktocks = 10;
do{
    HAL_Delay(1000);
    ticktocks--;
  }while(!CharInRing() && ticktocks > 0);
if (ticktocks == 0)
{
  printf("Get device state timeout.\r\n");
  while(1);
}
//device state response received
memset(rxBuf,0x00,sizeof(rxBuf));
rxBuf[0] = readring();
rxBuf[1] = readring();
rxBuf[2] = readring();
nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
nanolr_pktlen &= 0xFF;
for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
{
  rxBuf[indx] = readring();
}
if(nanolr_response_parser() != 0)
{
  printf("Device state response error.\r\n");
  while(1);
}
if(responsePkt.payload[0] == 0x30)
{
  printf("State = Online.\r\n");
}
else
{
  printf("State = 0x%02X.\r\n",responsePkt.payload[0]);
}
```



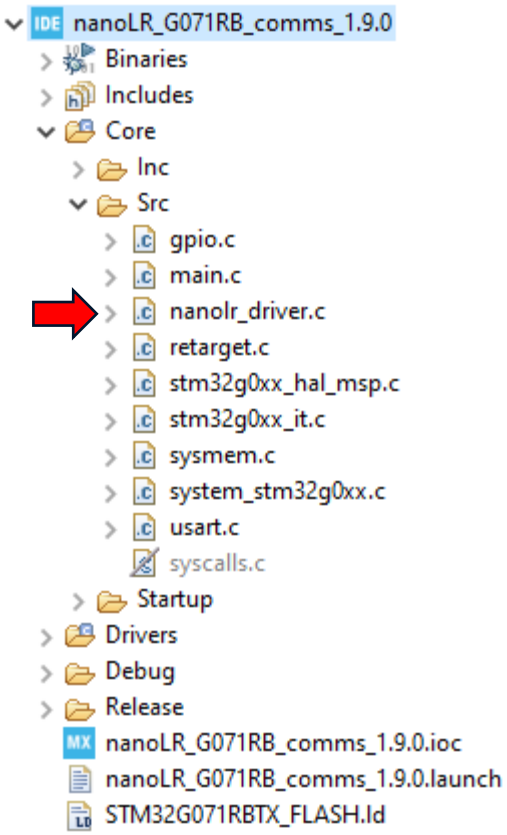
NUCLEO-G071RB LoRa Driver: main.c



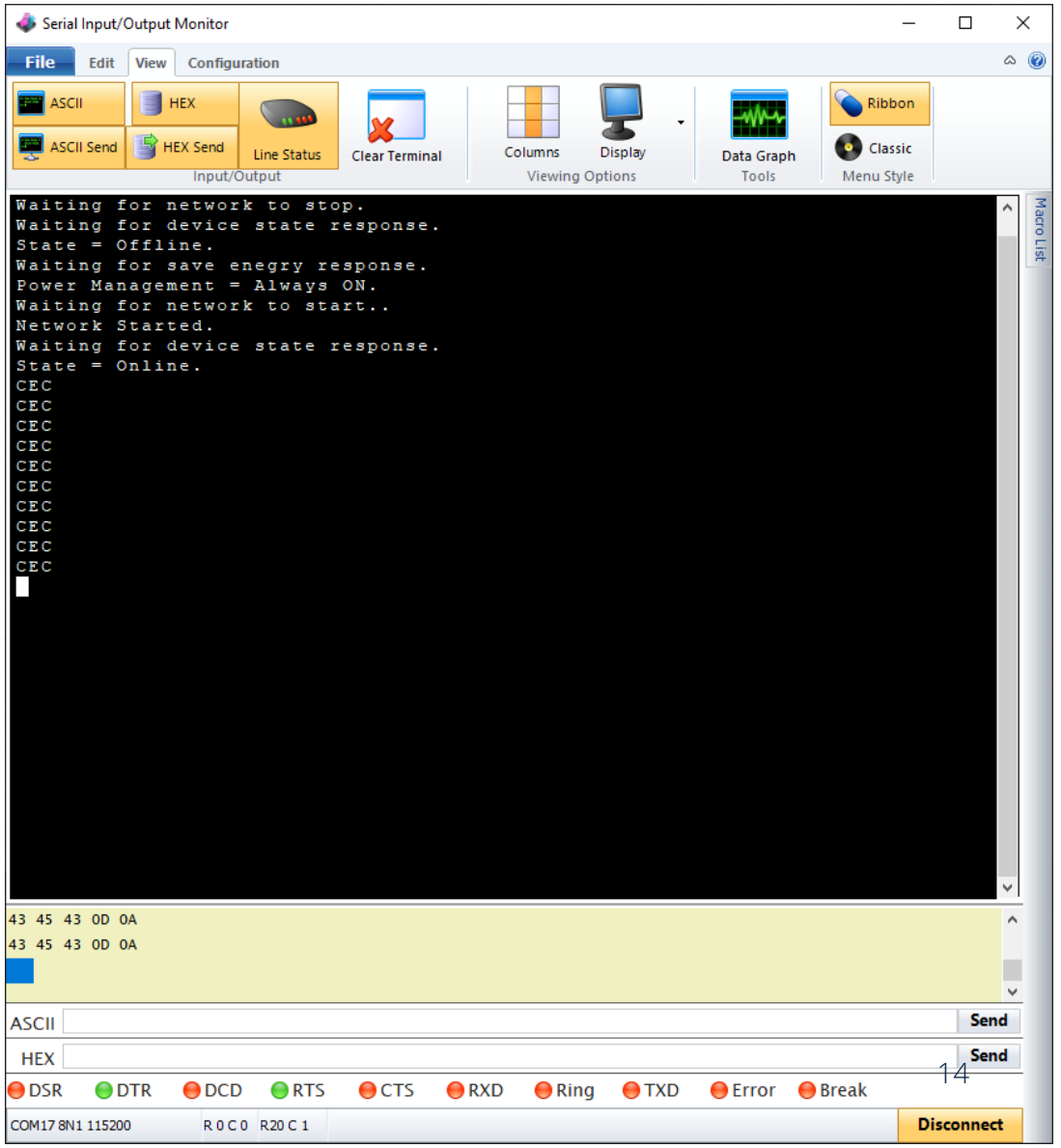

```
#ifdef receiver
if(CharInRing())
{
  HAL_Delay(100);
  memset(rxBuf,0x00,sizeof(rxBuf));
  rxBuf[0] = readring();
  rxBuf[1] = readring();
  rxBuf[2] = readring();
  if(rxBuf[2] == 0xE0)
  {
    nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
    nanolr_pktlen &= 0xFF;
    for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
    {
      rxBuf[indx] = readring();
    }
    //check response structure
    if(nanolr_response_parser() != 0)
    {
      //clear receive ring buffer
      //throw packet away
      __HAL_UART_DISABLE_IT(&huart1,UART_IT_RXNE);
      USART1_RxHead = 0x00;
      USART1_RxTail = 0x00;
      __HAL_UART_ENABLE_IT(&huart1,UART_IT_RXNE);
    }
    else
    {
      printf("%c%c%c\r\n",responsePkt.payload[8],responsePkt.payload[9],responsePkt.payload[10]);
    }

  }
  else
  {
    //clear receive ring buffer
    //throw packet away
    __HAL_UART_DISABLE_IT(&huart1,UART_IT_RXNE);
    USART1_RxHead = 0x00;
    USART1_RxTail = 0x00;
    __HAL_UART_ENABLE_IT(&huart1,UART_IT_RXNE);
  }
}
#endif
```



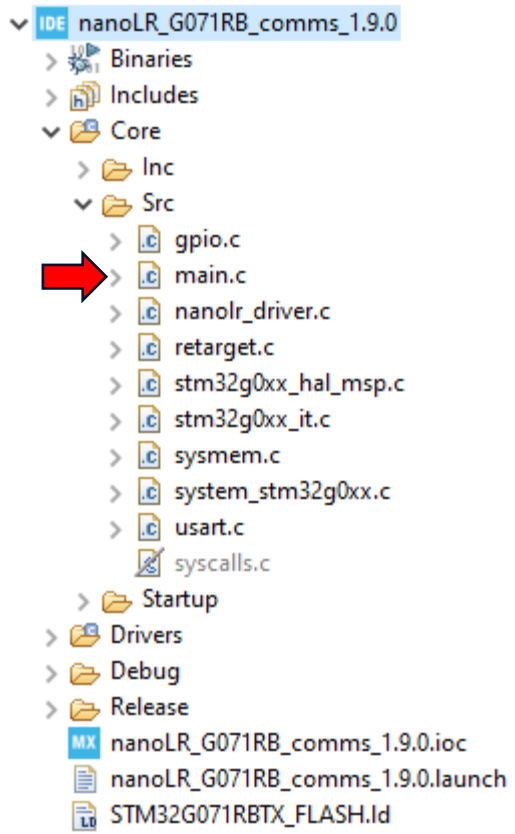
NUCLEO-G071RB LoRa Driver: nanolr_driver.c




```
// code in main.c
#ifdef transmitter
nanolr_send_data(0x03);
HAL_Delay(100);
#endif

// code in nanolr_driver.c
void nanolr_send_data ( uint8_t msglen )
{
uint16_t check_sum_tmp = 0;
memset(txBuf,0x00,sizeof(txBuf));

txBuf[0] = 0x00;
txBuf[1] = msglen + 8;
txBuf[2] = 0x50;
txBuf[3] = 0x00;
txBuf[4] = 0x00;
txBuf[5] = 0xFF;
txBuf[6] = 0xFF;
txBuf[7] = 'C';
txBuf[8] = 'E';
txBuf[9] = 'C';

    for ( uint8_t indx = 0; indx < 10; indx++ )
    {
        check_sum_tmp += txBuf[indx];
    }

    txBuf[10] = ( uint8_t ) ( check_sum_tmp & 0xFF );

    HAL_UART_Transmit(&huart1,txBuf,11,0xFFFF);
}
```



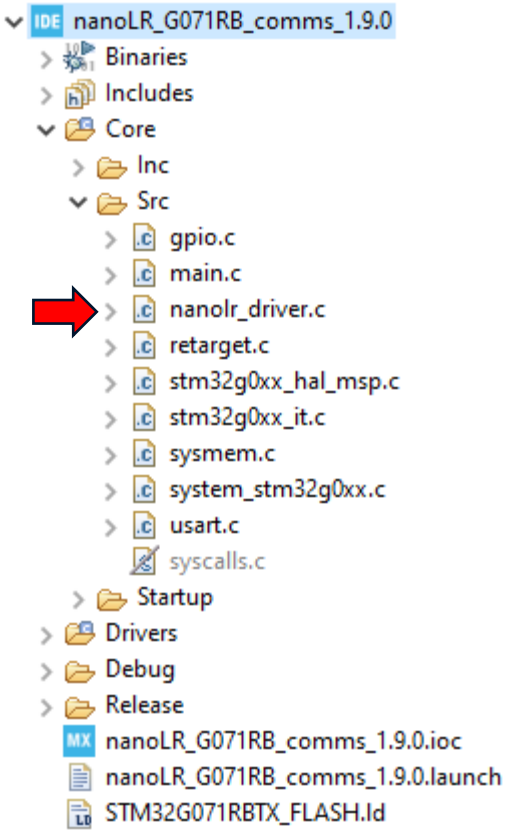
NUCLEO-G071RB LoRa Driver: main.c




```
if(CharInRing())
{
HAL_Delay(100);
memset(rxBuf,0x00,sizeof(rxBuf));
rxBuf[0] = readring();
rxBuf[1] = readring();
rxBuf[2] = readring();
if(rxBuf[2] == 0xE0)
{
nanolr_pktlen = make16(rxBuf[0],rxBuf[1]);
nanolr_pktlen &= 0xFF;
for(uint8_t indx = 3; indx < nanolr_pktlen ;indx++)
{
rxBuf[indx] = readring();
}
```

Expression	Type	Value
rxBuf	uint8_t [256]	0x2000008c
[0...99]	uint8_t [100]	0x2000008c
(x)= rxBuf[0]	uint8_t	0x0
(x)= rxBuf[1]	uint8_t	0xf
(x)= rxBuf[2]	uint8_t	0xe0
(x)= rxBuf[3]	uint8_t	0x0
(x)= rxBuf[4]	uint8_t	0x0
(x)= rxBuf[5]	uint8_t	0xff
(x)= rxBuf[6]	uint8_t	0xd1
(x)= rxBuf[7]	uint8_t	0x0
(x)= rxBuf[8]	uint8_t	0x1
(x)= rxBuf[9]	uint8_t	0xff
(x)= rxBuf[10]	uint8_t	0xff
(x)= rxBuf[11]	uint8_t	0x43
(x)= rxBuf[12]	uint8_t	0x45
(x)= rxBuf[13]	uint8_t	0x43
(x)= rxBuf[14]	uint8_t	0x89
(x)= rxBuf[15]	uint8_t	0x0
(x)= rxBuf[16]	uint8_t	0x0
(x)= rxBuf[17]	uint8_t	0x0

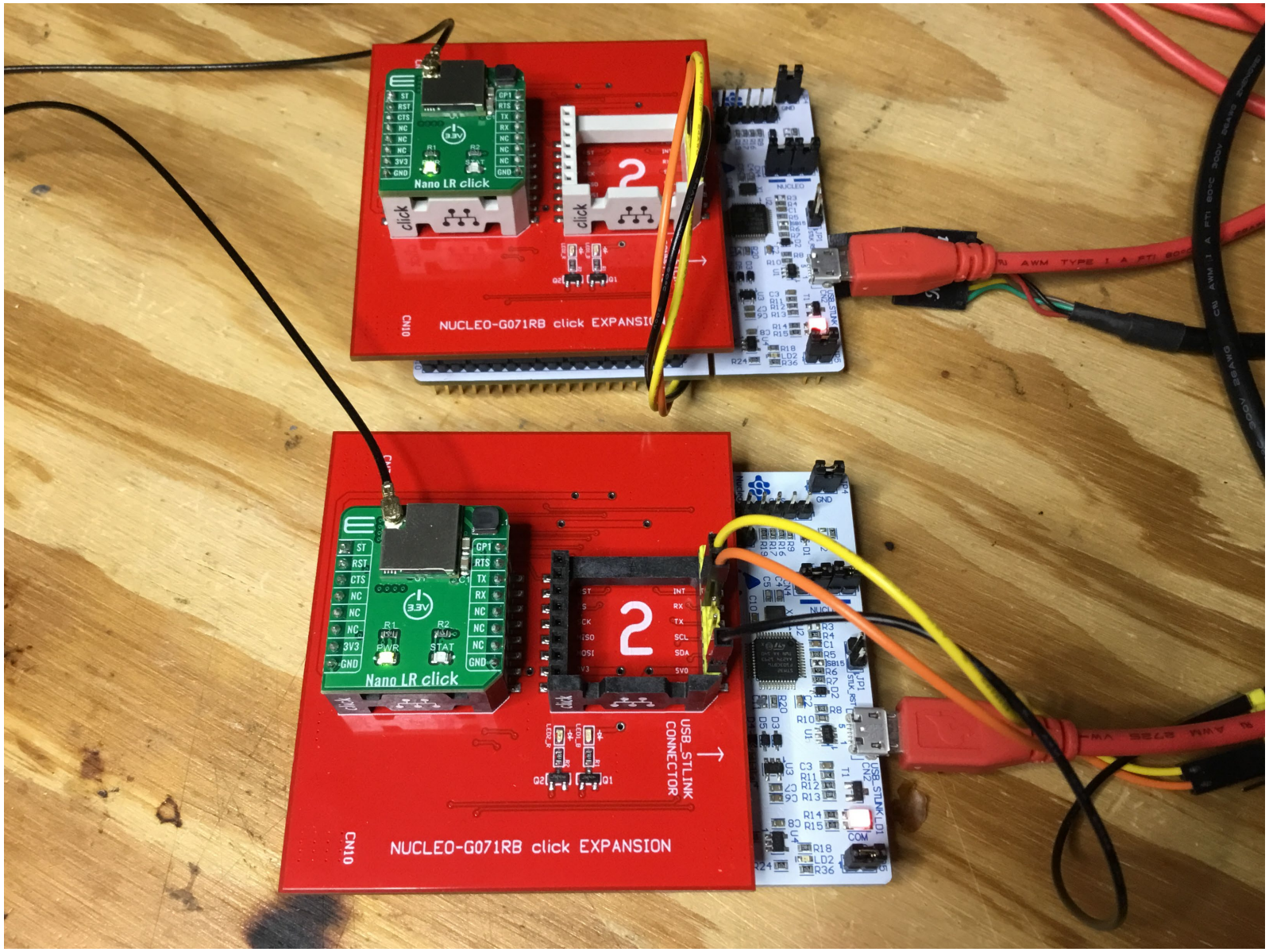
NUCLEO-G071RB LoRa Driver: nanolr_driver.c



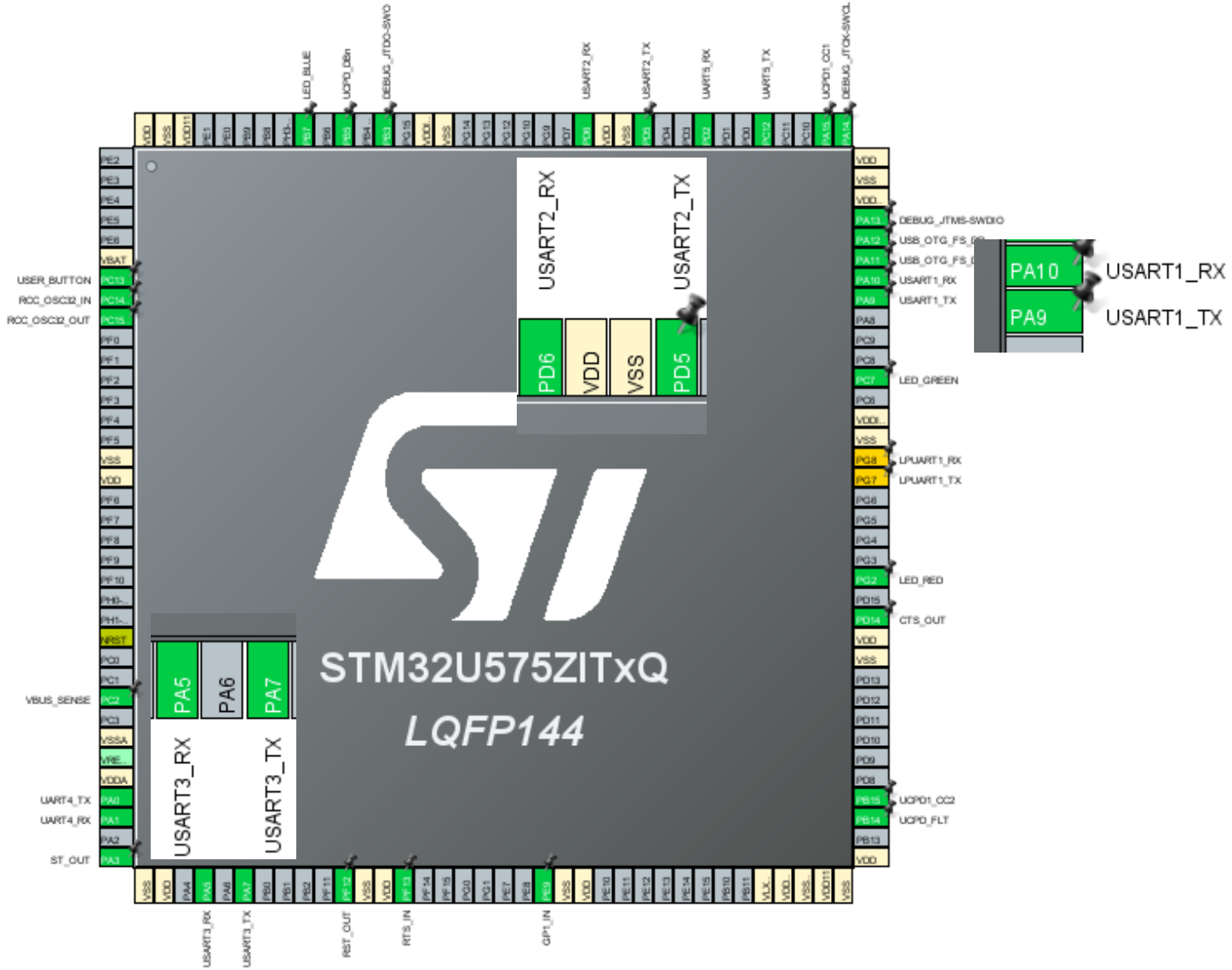

```c
//check response structure - code in main.c that kicks off nanolr_response_parser
if(nanolr_response_parser() != 0)
{



uint8_t nanolr_response_parser (void)
{
    uint8_t checksum = 0;
    //build responsePkt structure
    responsePkt.lenHI = rxBuf[0];
    responsePkt.lenLO = rxBuf[1];
    responsePkt.message_id = rxBuf[2];
    responsePkt.crc =    rxBuf[nanolr_pktlen - 1];
    //get payload bytes
    for ( uint16_t pktBite = 3; pktBite < nanolr_pktlen - 1; pktBite++ )
    {
        responsePkt.payload[pktBite - 3] = rxBuf[pktBite];
    }
    //compute received bytes checksum
    for ( uint16_t pktBite = 0; pktBite < nanolr_pktlen - 1; pktBite++ )
    {
        checksum += rxBuf[pktBite];
    }
    //compare computed checksum vs transmitted pkt checksum
    if ( checksum == responsePkt.crc )
    {
        return 0;
    }
```

Expression	Type	Value
rxBuf	uint8_t [256]	0x2000008c
responsePkt	nanolr_rxpkt_t	{...}
(x)= lenHI	uint8_t	0x0
(x)= lenLO	uint8_t	0xf
(x)= message_id	uint8_t	0xe0
payload	uint8_t [256]	0x20000213
[0...99]	uint8_t [100]	0x20000213
(x)= payload[0]	uint8_t	0x0
(x)= payload[1]	uint8_t	0x0
(x)= payload[2]	uint8_t	0xff
(x)= payload[3]	uint8_t	0xd1
(x)= payload[4]	uint8_t	0x0
(x)= payload[5]	uint8_t	0x1
(x)= payload[6]	uint8_t	0xff
(x)= payload[7]	uint8_t	0xff
(x)= payload[8]	uint8_t	0x43
(x)= payload[9]	uint8_t	0x45
(x)= payload[10]	uint8_t	0x43
(x)= payload[11]	uint8_t	0x0
(x)= payload[12]	uint8_t	0x0

NUCLEO-G071RB LoRa Driver: P2P LoRa Network

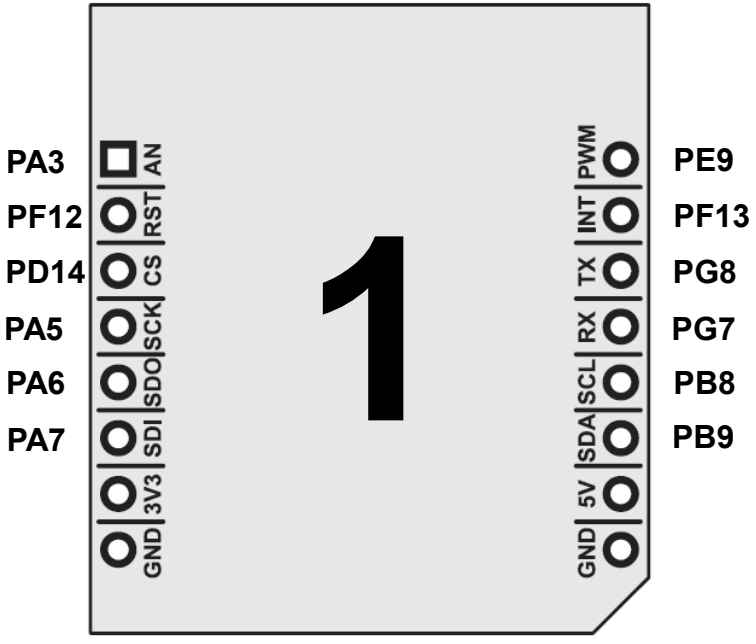


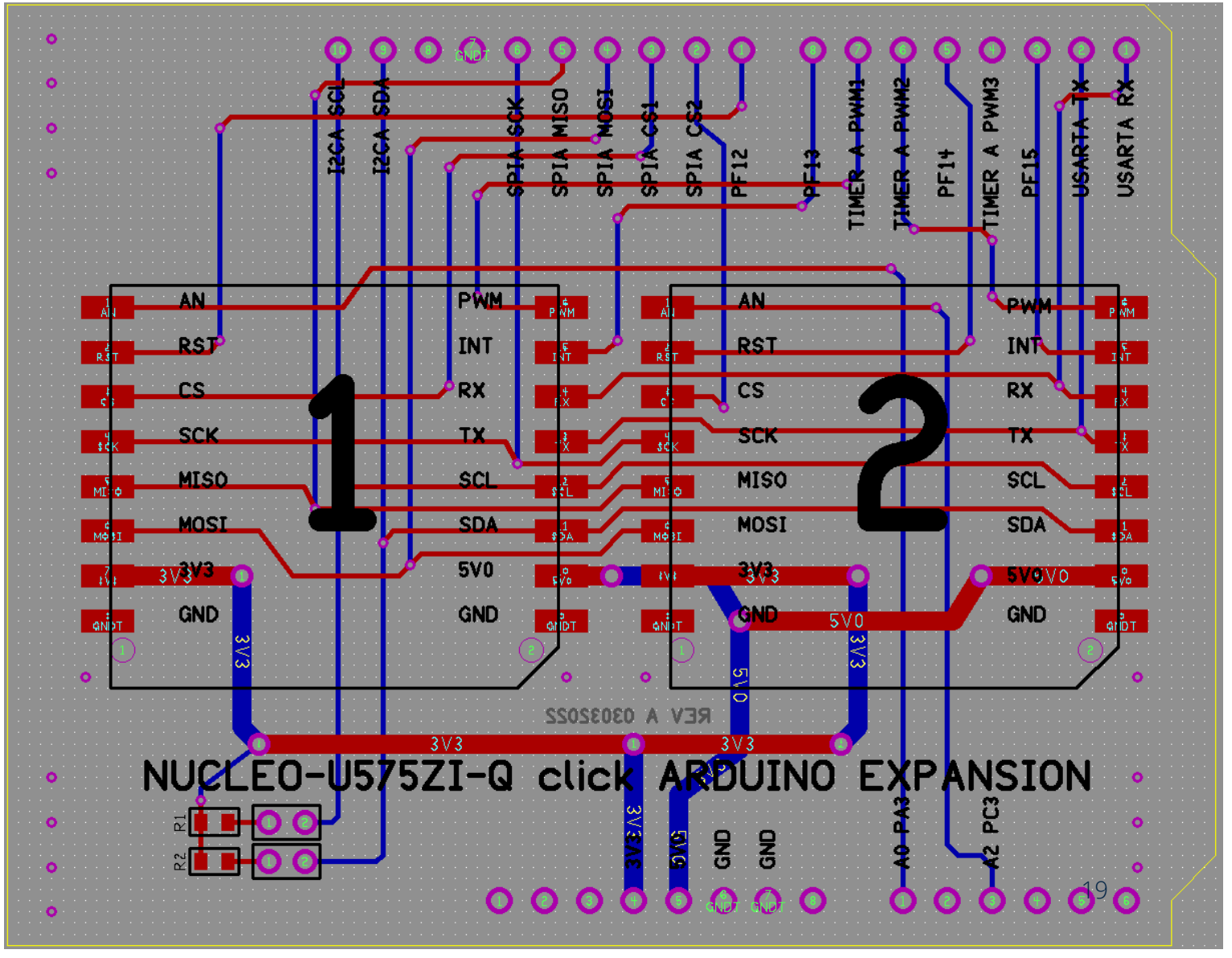
NUCLEO-U575ZI-Q LoRa Driver: Hardware Configuration



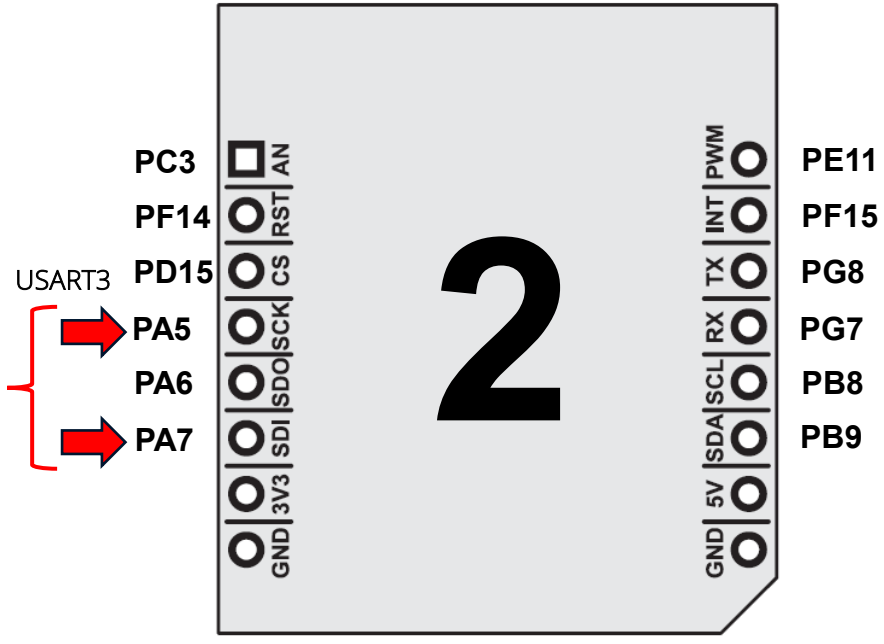
NUCLEO-U575ZI-Q LoRa Driver: Hardware Configuration

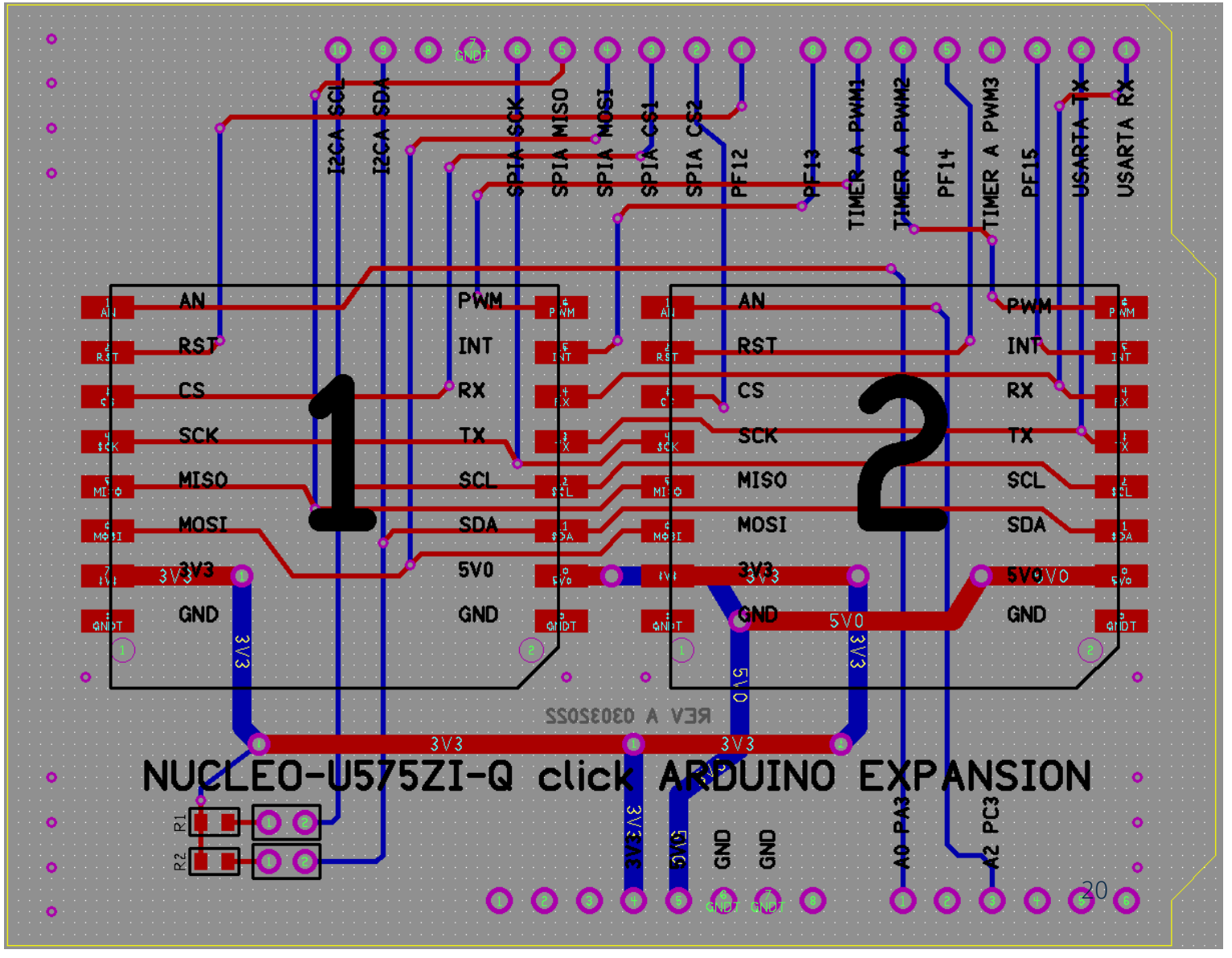
USART1
Rewired PG7 to PA9
Rewired PG8 to PA10



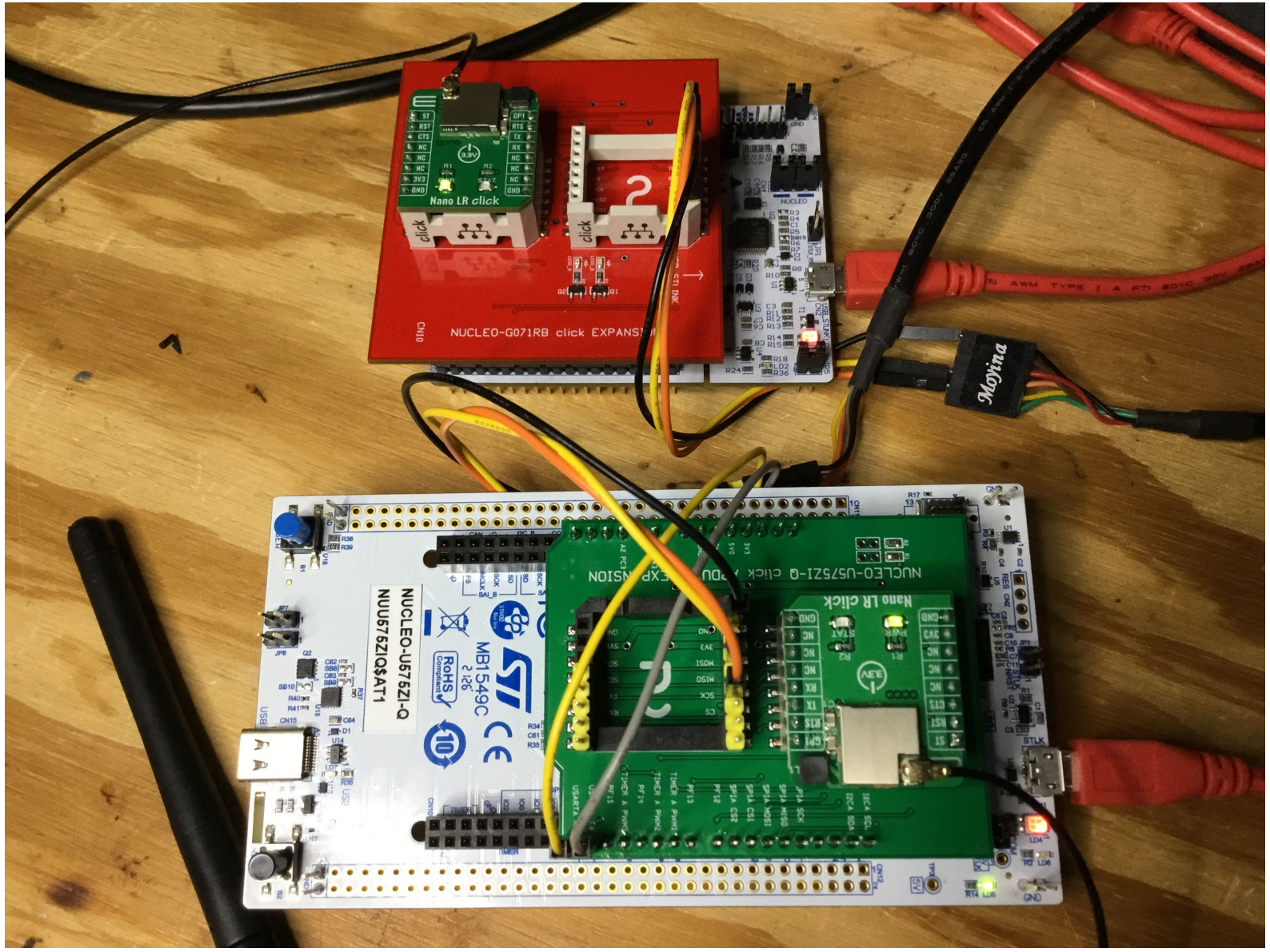


NUCLEO-U575ZI-Q LoRa Driver: Hardware Configuration



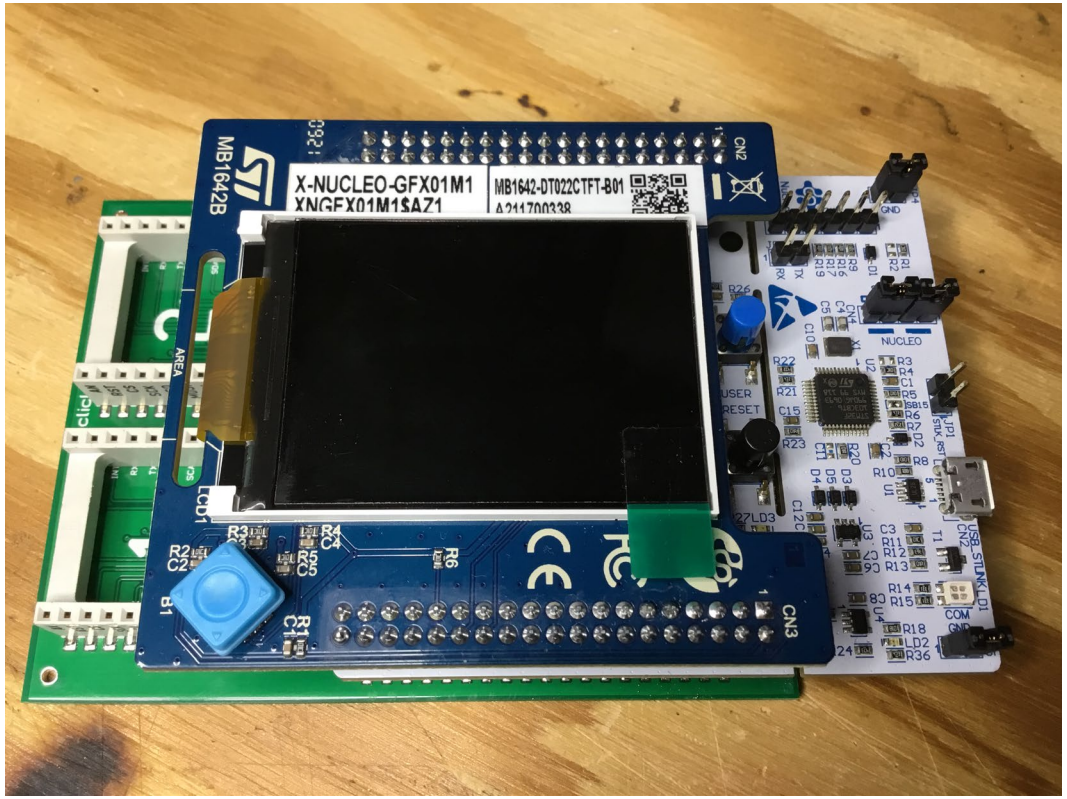


NUCLEO-U575ZI-Q LoRa Driver:
Runs NUCLEO_G071RB LoRa Driver



Thank you for attending!!!

MORE TO COME..



Please consider the resources below:

- Today's Project Download Package
- Nano LR click
- STM32U5 Series of Microcontrollers

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

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

informa markets