Embedded Security
Using the Gowin GW1NSE2 for Authentication

Contents **3**

1. Device Preparation

Since GW1NSE and GW1NSER are pre-provisioned, we only need to do provision on GW1NS2 to initialize the SecureFPGA board for development purposes. The provision steps are as follows:

1. Connect FPGA board via both USB mini cable and RS232 UART cable with PC.

2. Identify the USB-to-Serial port number and other USB serial ports in 'Device Manager.'

3. Find gw_secure_chip_demo.bat under gw_secure_chip_bat_demo file directory. Edit it and change the COM port number to the one shown in the device manager.

4. Open the command prompt and run 'gw_secure_chip_demo.bat'. Follow the instructions. Recycle power to the FPGA when it's necessary. After that, press 'Any key', then an Activation Code will be generated. Loaded and read back to the flash of the device. Finally, the Broadkey user program will be loaded.

1.1 Explain of the provisioning procedure

0. Erase MCU flash

programmer_cli.exe --run 21 --device GW1NS-2C

1. Load FPGA stream file

programmer_cli.exe --device GW1NS-2C --run 6 -f %path%\FPGA stream file

2. Load BroadKey Provision binary file

programmer_cli.exe --device GW1NS-2C --run 30 -m %path%\BkProvision.bin

3. Create device activation code binary file

iid-provision.exe enroll -p %com% -b 9600 -o %path%\out\device_ac.bin

4. Create csr_max.out

iid-provision.exe csr -p %com% -b 9600 -a %path%\out\device_ac.bin -k 12345678901234567890123456789012 --subject_c US --subject_cn 123456789012345678901234567890123456789012345678901234567890123456789012345678901234 --subject_o 12345678901234567890123456789012 -o %path%\csr_max.out

5. Load device activation code binary file

programmer_cli.exe --device GW1NS-2C --run 25 -a %path%\device_ac.bin

6. Upload device activation code binary file

programmer_cli.exe --device GW1NS-2C --run 26 -o %path%\ac_readback.bin

7. Load User application

programmer_cli.exe --device GW1NS-2C --run 27 -m %path%\broadkey_user.bin

There are two different tools used here one is Gowin programmer for downloading the data stream file to FPGA device and another is Intrinsic ID tool for activating Intrinsic software library.

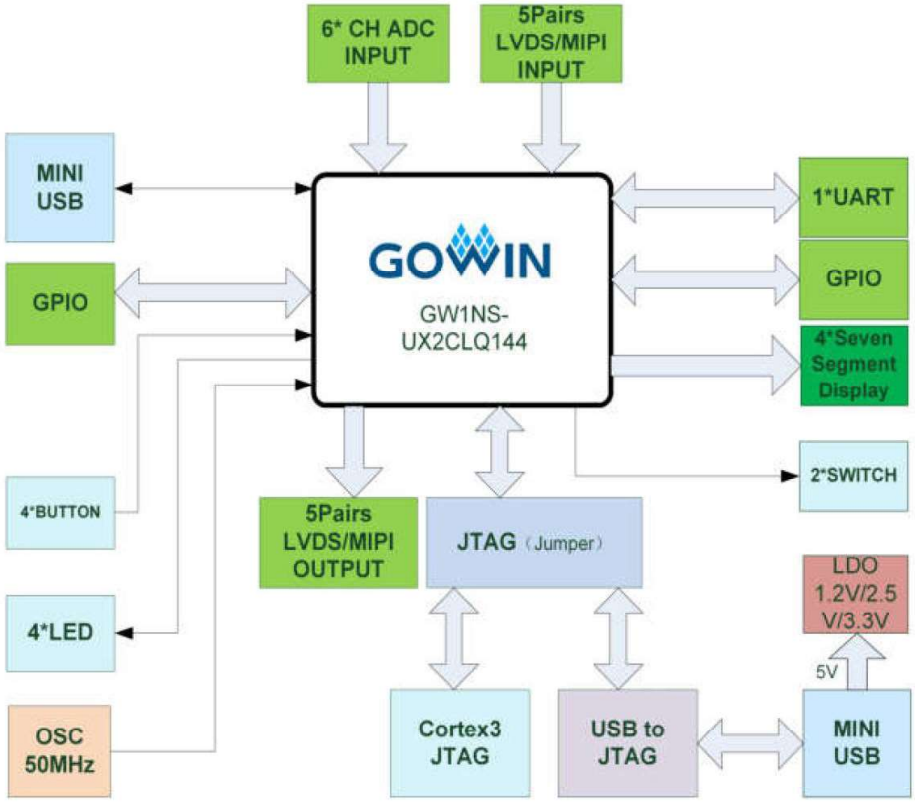
After successful provision and registering the chip, The device active code will be downloaded to the MCU flash's last 1k byte address. The MCU flash partition is showing as follows:

```
MCU Flash partition
/**************************
MCU Flash, 128kBytes  0kByte ~127kByte
0x00000
0x1FFFF
/*********************************************************/
0. User partition,  126kBytes, 0kByte ~ 125kByte
0x00000
0x1F7FF
/*********************************************************/
1. cert.pem partition,  1kBytes, 126kByte
0x1F800
0x1FBFF
/*******************************
2. device_ac.bin partition,  1kBytes, 127kByte
0x1FC00
0x1FFFF
/*********************************************************/
```

2. Development Environment Configuration

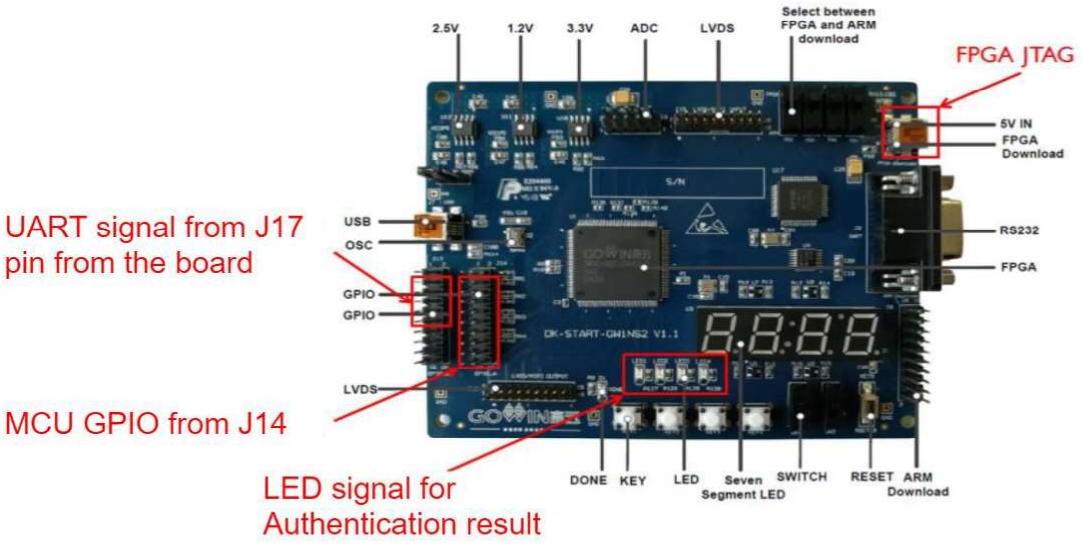
For developing embedded application, it's crucial to know what hardware resources you can manipulate and then configure the development environment.

2.1 Hardware resources



DK-START-GW1NS2 placement

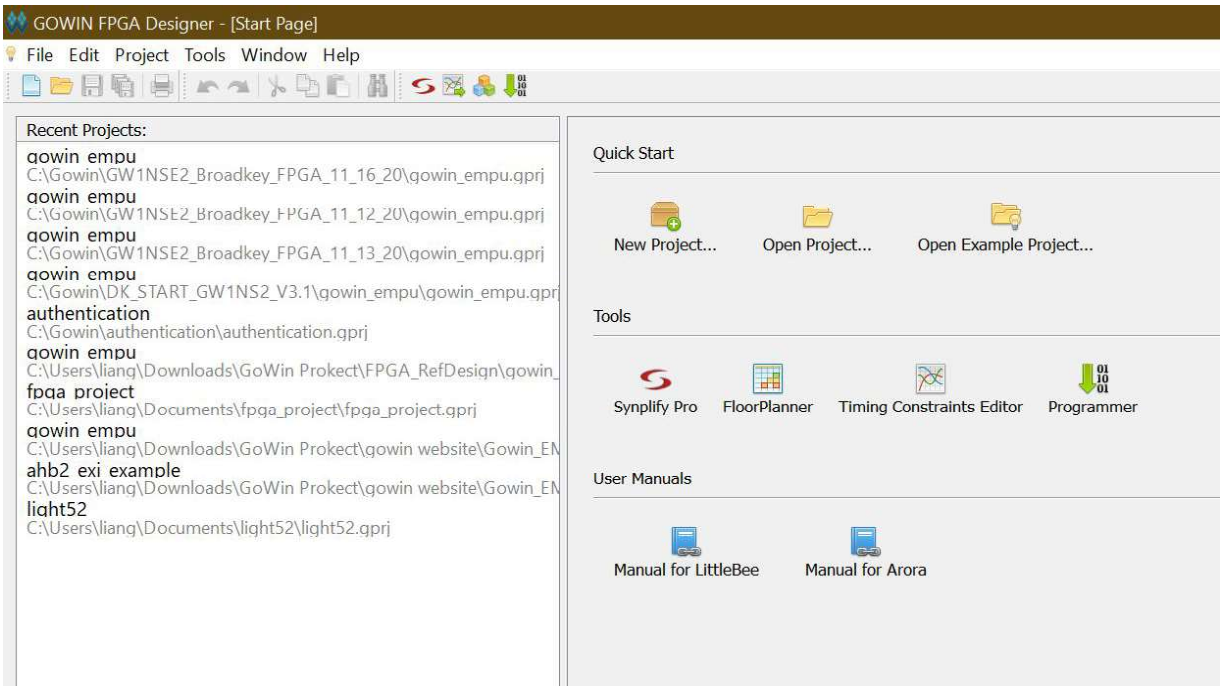
The resource that we have to use including: 1*UART, GPIO, 4*LED and 4*Seven Segment Display.



2.2 Gowin FPGA chip design software

Gowin software is an integrated circuit design and implementation tool for Gowin FPGA chip. It provides a complete FPGA design and verification environment based on GUI, integrating the design tools from the HDL to the FPGA bit stream (bitstream) download. It is easy to use with rich functions and the excellent performance.

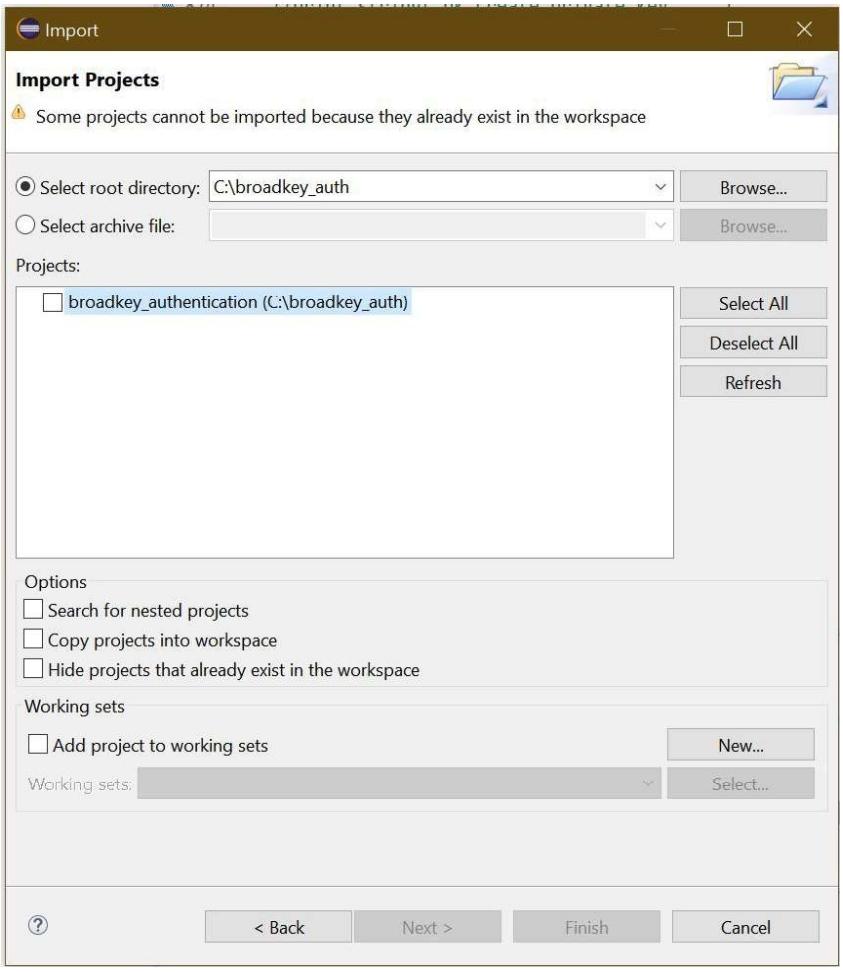
After downloading, double-click the Gowin software installation package and install according to the prompt. The installation directory can be changed as required. After installation, the shortcut will be created on the PC desktop.



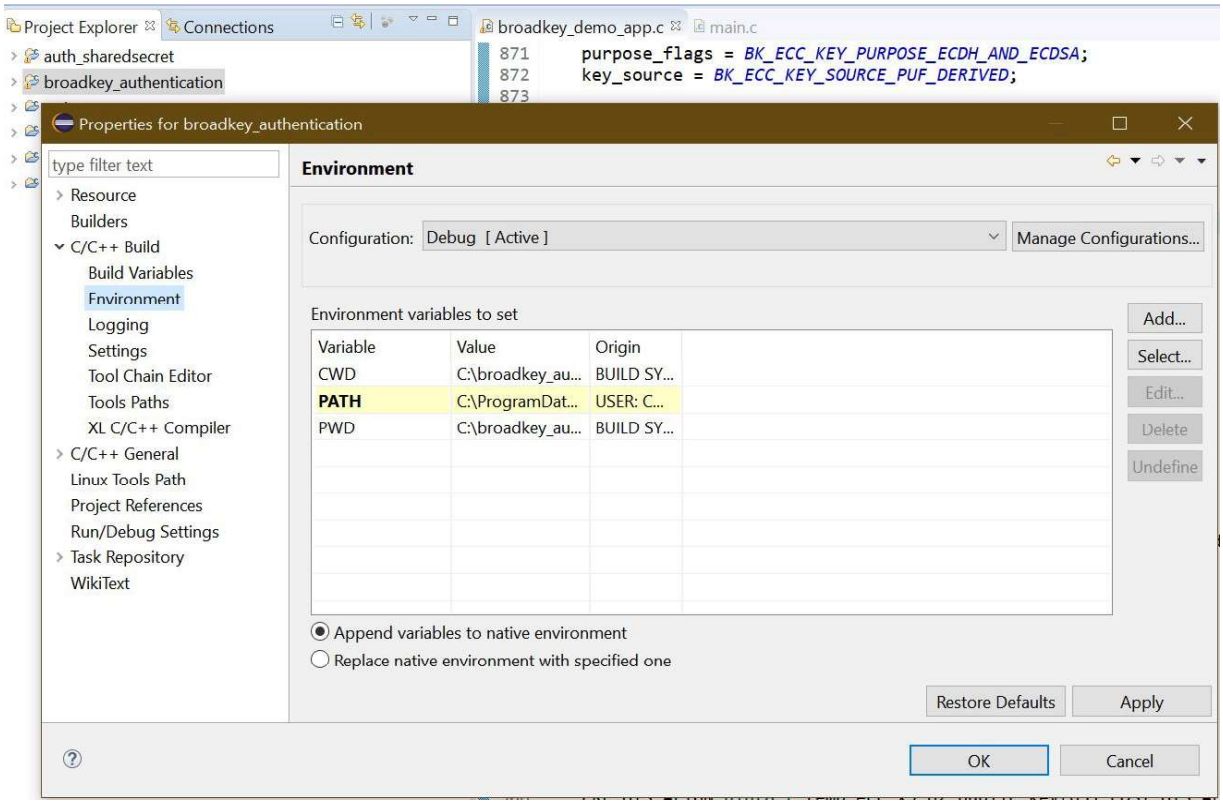
Gowin FPGA Designer

2.3 MCU development software

Following the ARM Eclipse development environment building tutorial to install the software to the computer, and import the broadkey_authentication project.



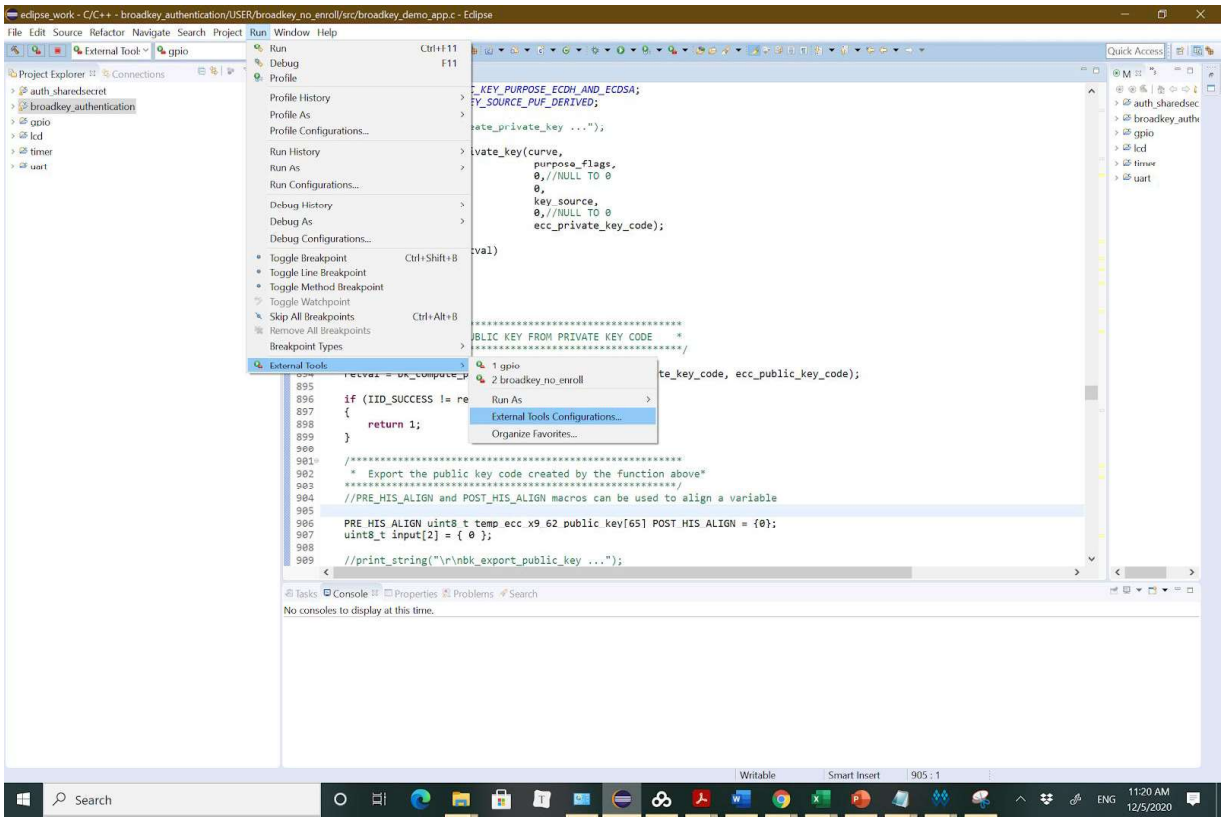
The key part is to configure the project properties after import your project. Setting the PATH variable in the Environment of C/C++Build section.



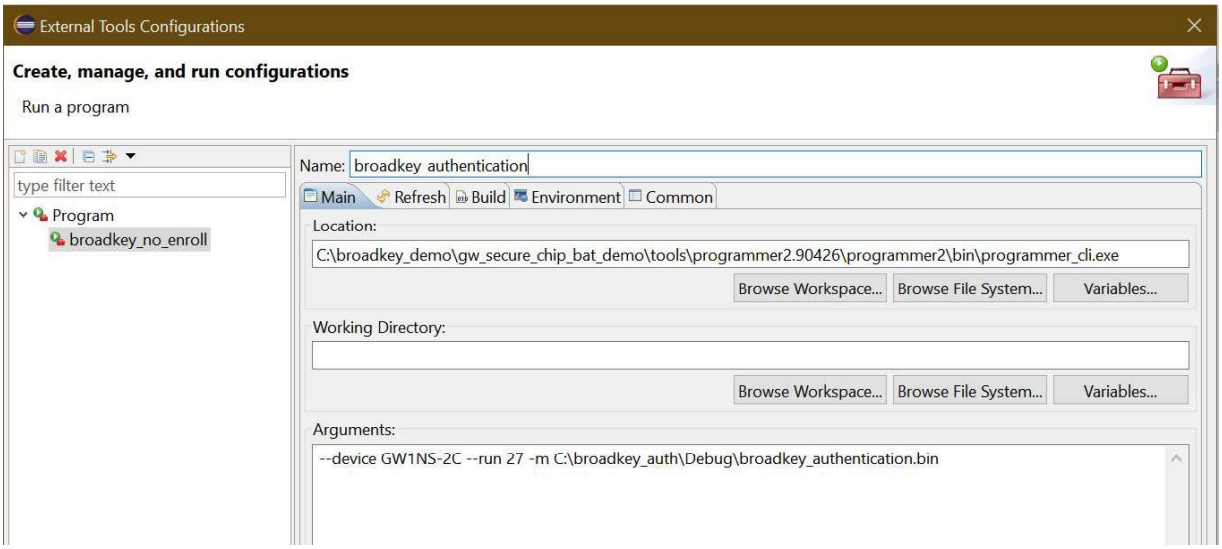
Putting the correct path of Java environment , ARM Embedded tools and MCU Eclipse build tools.

C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\eclipse_tools\tools**GNU Tools ARM Embedded\5.4 2016q2\bin**;C:\eclipse_tools_181108a\eclipse_tools\tools**GNU MCU Eclipse\Build Tools\2.11-20180428-1604\bin**

If all the variables are setting correct, you can build the project successfully. The last step is to download the binary file of the project to the DK-START-GW1NS2 board by using Gowin programmer. This is can be configured in the Run-->External Tools-->External Tools Configurations setion.



Creating a new configuration named boradkey_authentication and input the location of Gowin programmer and assign the Arguments which is very important to ensure downloading the binary file to the correct location of MCU flash.



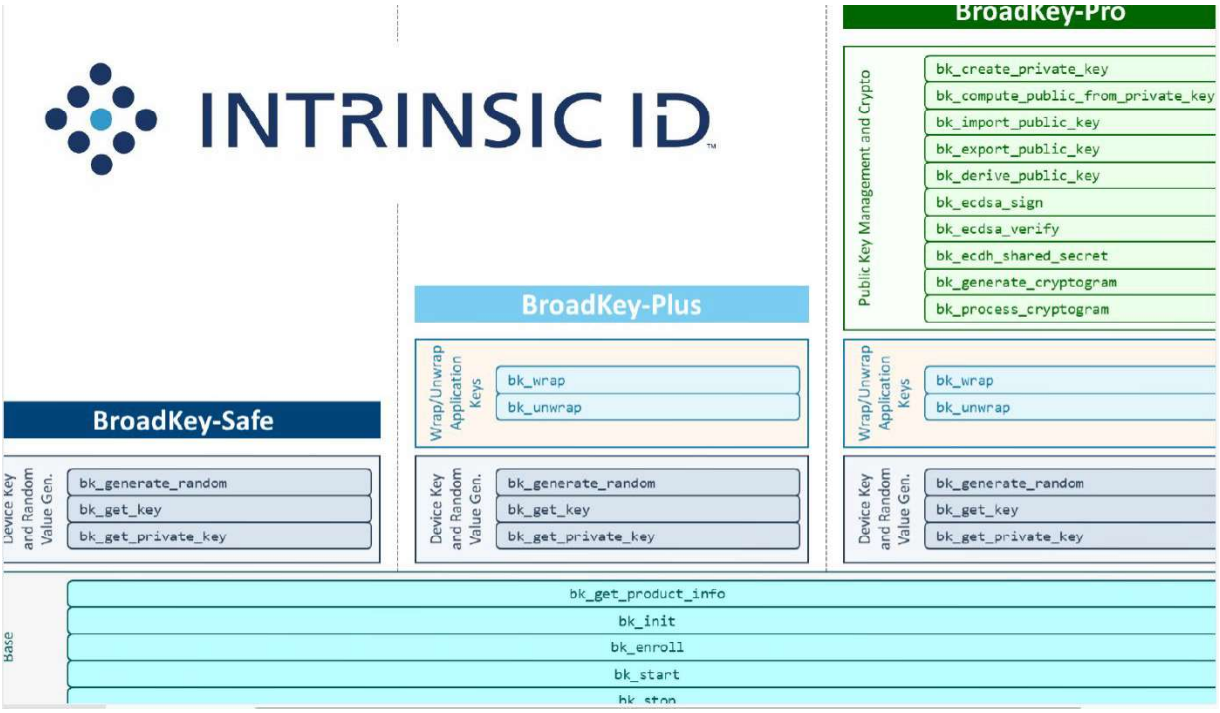
--device GW1NS-2C --run 27 -m C:\broadkey_auth\Debug\broadkey_authentication.bin

The downloading process can also be achieved by creating a windows console batch file which is more convenient.

```
::*********************************************************************/
@echo off

set com=COM22

set path=%cd%
set programmer_path=%path%\tools\programmer2.90426\programmer2\bin

echo.
echo com=%com%
echo programmer_path=%programmer_path%
echo path=%path%
echo.

echo.
echo Load broadkey_authentication.bin
%programmer_path%\programmer_cli.exe --device GW1NS-2C --run 27 -m
%path%\broadkey_authentication.bin
::echo programmer_cli.exe 27 errorlevel=%errorlevel%

echo.
echo Turn off the power,and turn on the power,the uart console will print the device_ac data.
echo Uart com=%com%, baudrate=115200

echo.
echo Finished.
echo.

pause

::/*********************************************************************
::END FILE
::*********************************************************************/
```

3. Authentication Framework Design

3.1 BroadKey Function Sets



3.1.1 bk_init

This function is used to initialize the BroadKey software module before use, after each device power-up or reset. When bk_init returns IID_SUCCESS, BroadKey moves from the Uninitialized to the Initialized state.

3.1.2 bk_generate_random

This function will generate a sequence of random bytes.

3.1.3 bk_import_public_key

Since the host needs to verify the identity of the device, the device public key is needed. This function imports an elliptic curve public key from a provided X9.62 binary format (uncompressed) to a corresponding public key code format.

3.1.4 bk_ecdsa_verify

This function verifies the ECDSA signature of a message or a hash of message with an elliptic curve public key in the internal public key code format.

bk_ecdsa_verify has no explicit output parameters, only inputs. Its result is contained in its return code. If IID_SUCCESS is returned, the signature on the message is successfully verified. If IID_INVALID_SIGNATURE is returned, the signature on the message is invalid.

3.1.5 bk_create_private_key

This function transforms an elliptic curve private key into a protected private key code which is only usable within the same cryptographic context, and on the same unique device, it was created on.

In our implementation, the parameter of private_key is not used because of the following note:

For key sources other than BK_ECC_KEY_SOURCE_USER_PROVIDED, this input is not used.

3.1.6 bk_compute_public_from_private_key

This function computes the elliptic curve public key corresponding to a private key code created with bk_create_private_key, and outputs the public key in a corresponding public key code format. The curve and purpose flags of the public key (code) will be the same as the one of the provided private key (code).

3.1.7 bk_export_public_key

This function exports a public key from BroadKey's public key code format to an X9.62 (uncompressed) binary elliptic curve public key format.11 The curve on which the public key is defined, as well as the purpose flags stored alongside the key in the public key code, are returned as well.

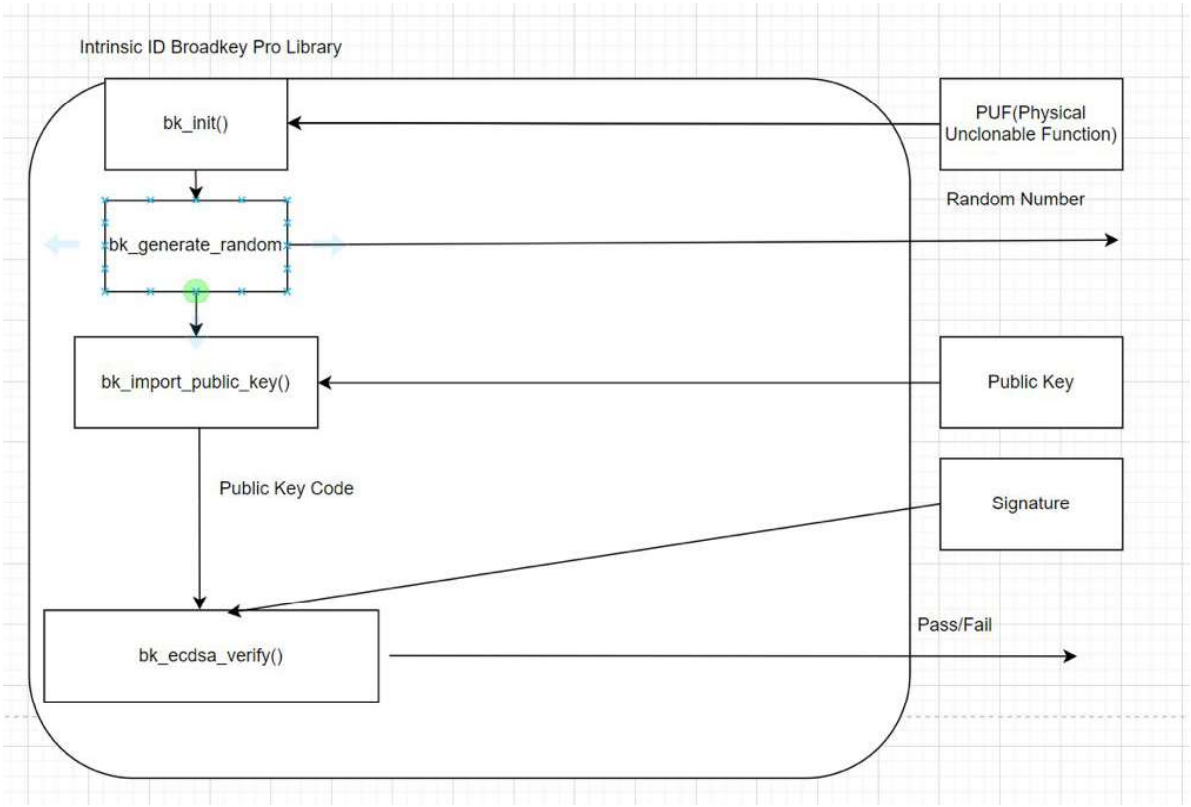
3.1.8 bk_ecdsa_sign

This function signs a message or a hash of a message using ECDSA with an elliptic curve private key in the internal private key code format. Signing can be done with either a random seed or a deterministically derived seed as indicated by the calling application.

3.2 Authentication Procedure

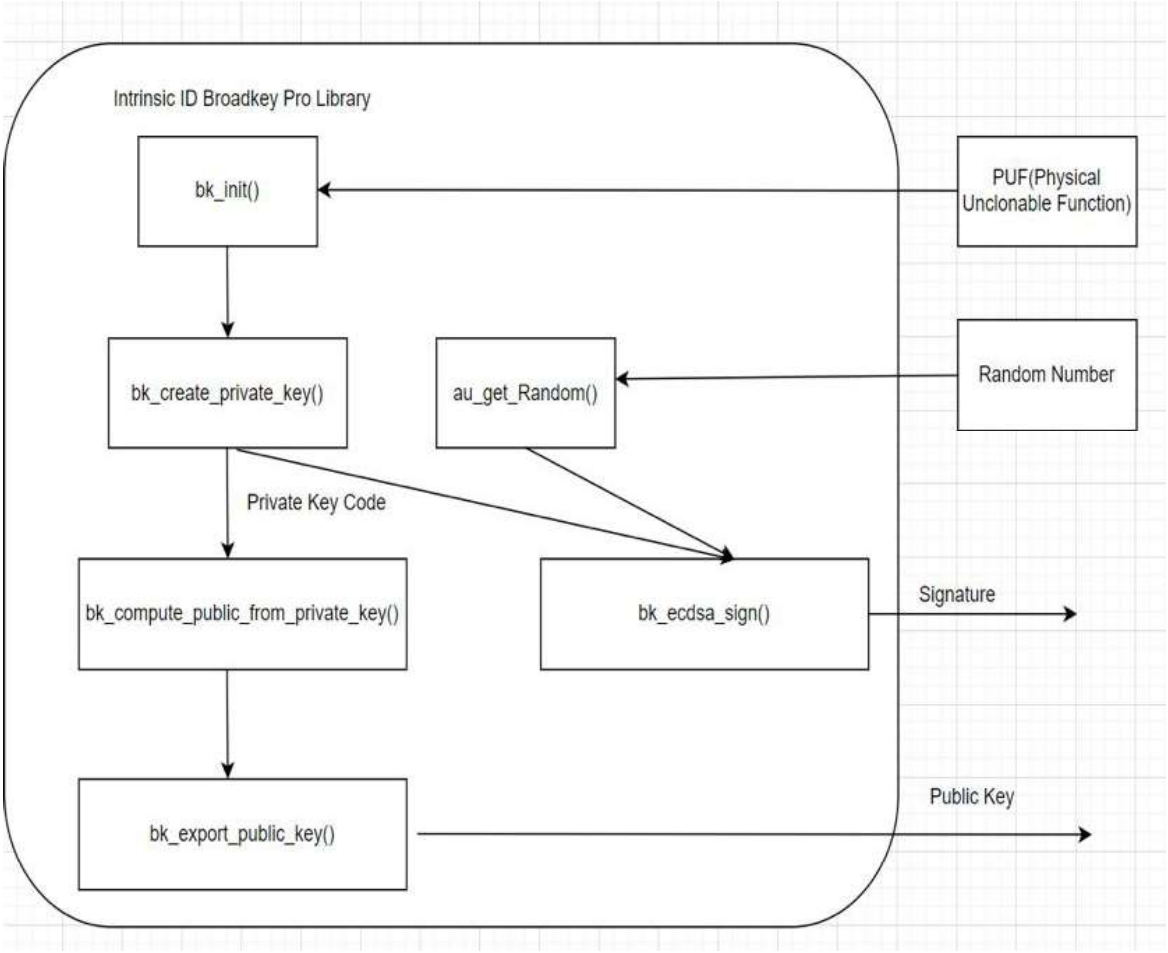
The authentication procedure for the host board:

1. **Initialize the host board using bk_init().**
2. **The host board generates a random number using bk_generate_random and sends it to the client board.**
3. **Waiting for the client board to reply.**
4. **After receiving the public key and signature from the client board, convert the public key to public key code and verify the signature using bk_ecdsa_verify().**
5. **It will either pass or fail the verification to finish the authentication process.**



The authentication procedure for the client board:

1. **Initialize the client board using bk_init().**
2. **Create a private key code using bk_create_private_key().**
3. **Generate a public key based on the private key code using bk_compute_public_from_private_key().**
4. **Receiving the random number from the host board.**
5. **Use the random number and private key to generate a signature.**
6. **Send the signature and public key to the host board.**



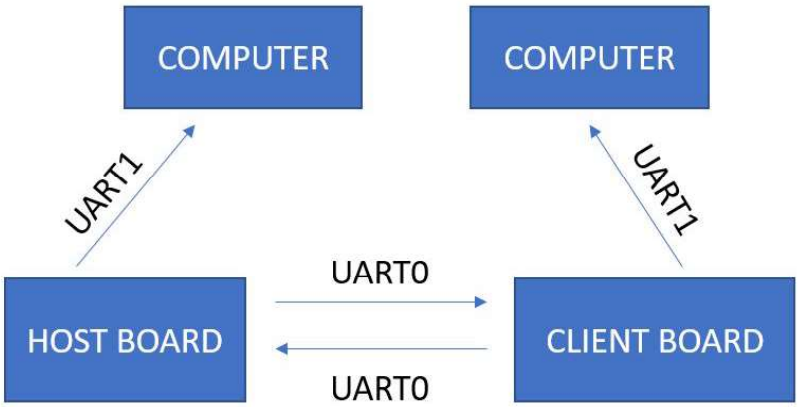
3.3 Borads Communication Scheme Design

The first scheme for two boards communication is to use two uart ports on each board. The purpose of having two uart ports:

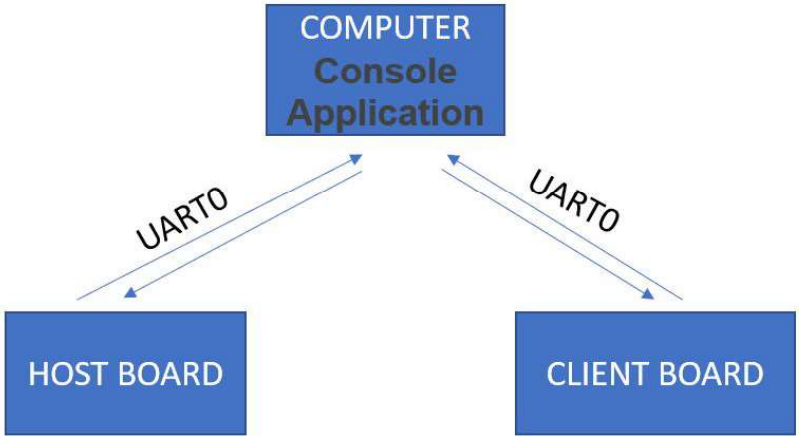
Uart0： random number;
public key
signed signature

Uart1: debug

visualization



Another is just using one uart ports each board and using a console to show the hole authentication procedure and forward information between boards.



The function of Console Application:

visualization of hole procedure;

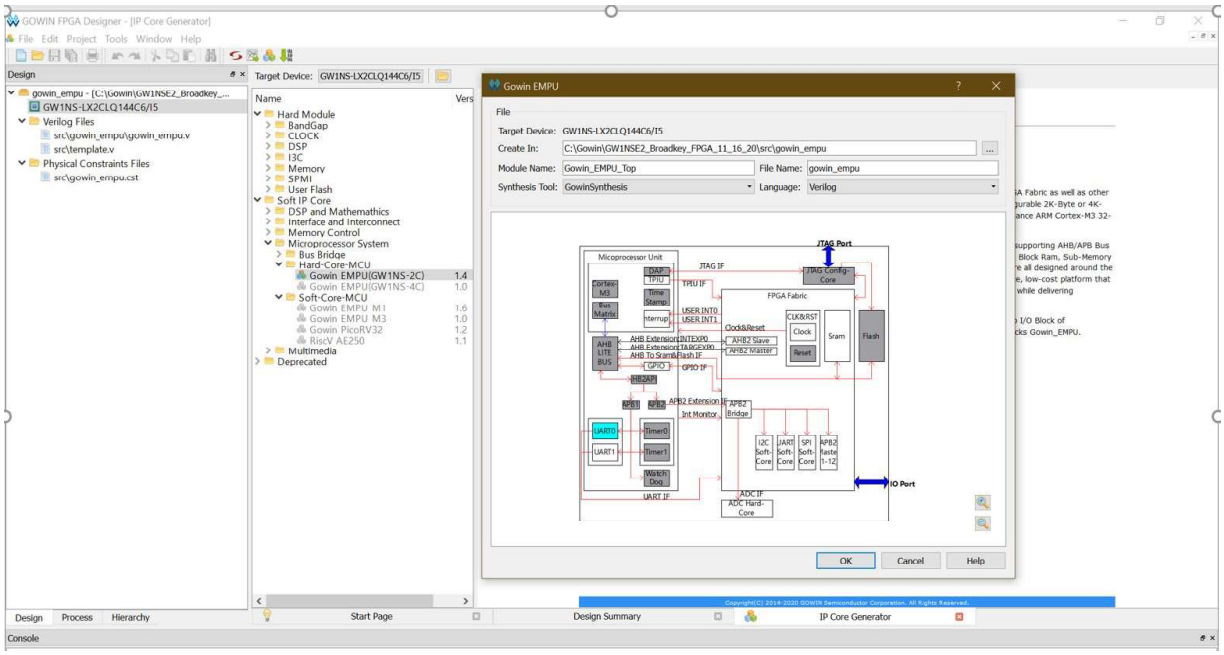
Forward information

We use the second solution for the the communication scheme.

4. Implementation

4.1 FPGA Application

We customized hardware resources for the project by using Gowin FPGA progrmmer.



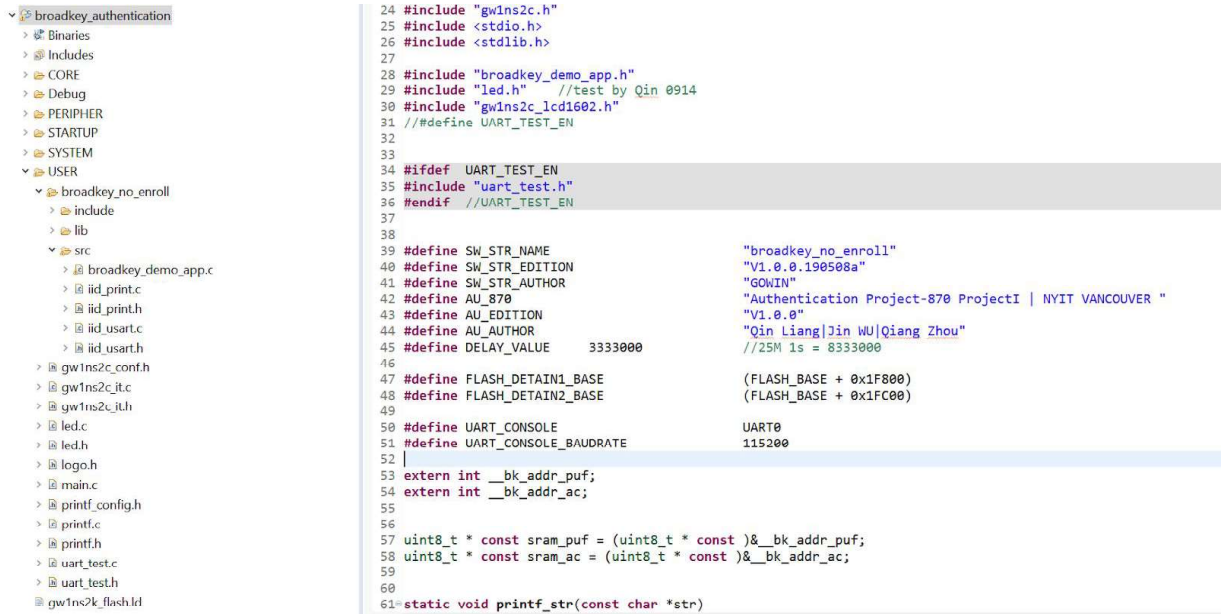
The details step for adding an additional uart prot include:

1. **Tools->IP Core Generator->click ‘Open IP config file’ icon -> open ‘gowin_empu.ipc’**
2. **Double click on ‘UART1’ in the diagram -> check ‘Enable UART1’ -> click ‘ok’**
3. **Click ‘ok’ to Gowin EMPU window (IP will generate) -> click ‘ok’ to ‘Do you want to added generated file(s)…’**
4. **Looks like template.v is the top level Verilog module. So, we have to add uart1 ports to it and connect those ports to the processor module declaration (see updated template.v).**
5. **Update CST for uart1_rxd and uart1_txd with the header pins you’d like to use.**
6. **Resynthesize**

4.2 MCU Application

The main functions of this project are achieved in the MCU parts that the implementation of the authentication project is based on the broadkey_no_enroll demo applicaton and related firmware provided by Gowin.We have to build our project by adding or modify some files in the demo app.

The main parts of the project are showed below:



The files include:

1. **“main.c”: The entry point of the project;**

Firs Define the project information, UART port, puf and active code address.

```
#define SW_STR_NAME              "broadkey_no_enroll"
#define SW_STR_EDITION           "V1.0.0.190508a"
#define SW_STR_AUTHOR            "GOWIN"
#defineAU_870                    "Authentication Project-870 ProjectI | NYIT VANCOUVER "
#define AU_EDITION               "V1.0.0"
#define AU_AUTHOR                "Qin Liang|Jin WU|Qiang Zhou"

#define FLASH_DETAIN1_BASE                    (FLASH_BASE + 0x1F800)
#define FLASH_DETAIN2_BASE                    (FLASH_BASE + 0x1FC00)

#define UART_CONSOLE                                  UART0
#define UART_CONSOLE_BAUDRATE                         115200

extern int __bk_addr_puf;
extern int __bk_addr_ac;


uint8_t * const sram_puf = (uint8_t * const )&__bk_addr_puf;
uint8_t * const sram_ac = (uint8_t * const )&__bk_addr_ac;
```

Next is setup the microcontroller system\updat the SystemCoreClock variable and initialize the uart port;

```
SystemInit();
//Init Uart
UartInit();

void UartInit(void)
{
  UART_InitTypeDef UART_InitStruct;

  UART_InitStruct.UART_Mode.UARTMode_Tx = ENABLE;
  UART_InitStruct.UART_Mode.UARTMode_Rx = ENABLE;
  UART_InitStruct.UART_Int.UARTInt_Tx = DISABLE;
  UART_InitStruct.UART_Int.UARTInt_Rx = DISABLE;
  UART_InitStruct.UART_Ovr.UARTOvr_Tx = DISABLE;
  UART_InitStruct.UART_Ovr.UARTOvr_Rx = DISABLE;
  UART_InitStruct.UART_Hstm = DISABLE;
  UART_InitStruct.UART_BaudRate = 115200;//Baud Rate
  //UART_CONSOLE,UART0
  UART_Init(UART_CONSOLE,&UART_InitStruct);
}
```

Then the main program will jump into “boardkey_demo_app.c” for Authentication procedure. If return successfully, the led will be flashing and lcd will show show message.

```
broadkey_demo_app(sram_puf, sram_ac);

if(succeed) {
        Lcd_Pin_Config();
        Lcd_Init();
        Lcd_Write_String("1",1);

        GPIO0->OUTENSET = 0xffffffff;

        while (1)
        {
                GPIO_SetBit(GPIO0, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);
                GPIO_ResetBit(GPIO0, GPIO_Pin_0);
                Delay(DELAY_VALUE);

                GPIO_SetBit(GPIO0, GPIO_Pin_0 | GPIO_Pin_2 | GPIO_Pin_3);
                GPIO_ResetBit(GPIO0, GPIO_Pin_1);
                Delay(DELAY_VALUE);
```

```
                GPIO_SetBit(GPIO0, GPIO_Pin_1 | GPIO_Pin_0 | GPIO_Pin_3);
                GPIO_ResetBit(GPIO0, GPIO_Pin_2);
                Delay(DELAY_VALUE);

                GPIO_SetBit(GPIO0,GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_0);
                GPIO_ResetBit(GPIO0, GPIO_Pin_3);
                Delay(DELAY_VALUE);
        }
}
```

2. **"boardkey_demo_app.c":The authentication procedure was done in this part.**

The main functions include:

```
*****initialize and enroll********
static uint8_t demo_version(void);
static uint8_t demo_init(uint8_t  * const sram_puf);
static uint8_t demo_enroll_or_start(uint8_t  * const sram_ac);

*****Create random number and get random number*******
static uint8_t demo_get_random_bytes(uint8_t  * const random_arr);
static uint8_t au_get_randomnum(uint8_t * const message,
                                        const uint32_t message_length);

*****Create public key code and export to public key******
static uint8_t demo_create_private_and_public_key_codes();
*****import public key and convert to public key code*****
static uint8_t demo_import_and_export_public_key(void);


*****Signature create and verify*****
static uint8_t demo_ecdsa_sign_and_verify(const bk_ecc_private_key_code_t * const
ecc_private_key_code,
                        const bk_ecc_public_key_code_t  * const ecc_public_key_code,
                        const uint8_t                * const message,
                        const uint32_t                     message_length);
```

These functions are all called by boardkey_demo_app function:

```
void broadkey_demo_app(uint8_t * const sram_puf, uint8_t * const sram_ac)
{
   PRE_HIS_ALIGN bk_ecc_private_key_code_t ecc_private_key_code POST_HIS_ALIGN;
  PRE_HIS_ALIGN bk_ecc_public_key_code_t ecc_public_key_code POST_HIS_ALIGN;
   //sending random number to host_board and also using for verifying signature by Qin 11/19/2020;
```

```
   PRE_HIS_ALIGN uint8_t h_message[16] POST_HIS_ALIGN = {0};
   //using for create signature by client_board by Qin 11/19/2020;
   PRE_HIS_ALIGN uint8_t c_message[16] POST_HIS_ALIGN = {0};

   iid_return_t retval;

   /* Setup USART with Baudrate 115200 */
   iid_usart_setup(115200);
   /************************
   * Initialize BroadKey *
   ************************/
   if (demo_init(sram_puf))
   {
      goto EXIT;
   }
   /*****************************
    *  BroadKey Enroll or Start  *
    *****************************/
   if (demo_enroll_or_start(sram_ac))
   {
      goto EXIT;
   }

     /*******************************
    *  BroadKey  Get Random Bytes  *
    *******************************/
#if defined(HOST_BOARD)
   if (demo_get_random_bytes(h_message))
   {
      goto EXIT;
   }
#endif //defined(HOST_BOARD)


#if defined(CLIENT_BOARD)
   if(au_get_randomnum(c_message, sizeof(c_message))){
          goto EXIT;
      }
#endif //defined(CLIENT_BOARD)
   /************************************************
    *  BroadKey CREATE PRIVATE AND PUBLIC KEY CODE  *
    ************************************************/
   if (demo_create_private_and_public_key_codes(&ecc_private_key_code, &ecc_public_key_code))
   {
      goto EXIT;
   }

    /***************************************
    *  BroadKey ECDSA SIGN AND VERIFY KEY  *
    ***************************************/
#if defined(CLIENT_BOARD)
   if (demo_ecdsa_sign_and_verify(&ecc_private_key_code, &ecc_public_key_code, c_message,
sizeof(c_message)))
   {
      goto EXIT;
```

```
    }
#endif //defined(CLIENT_BOARD)

//add by Qin 11/19/2020
#if defined(HOST_BOARD)
        if (demo_ecdsa_sign_and_verify(&ecc_private_key_code, &ecc_public_key_code, h_message,
sizeof(h_message)))
        {
            goto EXIT;
        }
#endif //defined(HOST_BOARD)

#endif /* defined(BK_CONFIGURATION_PRO_ENABLED) */

EXIT:
#if defined(CLIENT_BOARD)
    print_string("\r\n--- End of the Authentication of client_board Demo Application! ---\r\n");
#endif
#if defined(HOST_BOARD)
    print_string("\r\n--- End of the Authentication of host_board Demo Application! ---\r\n");
#endif
    return;
}
```

4.3 Console Application

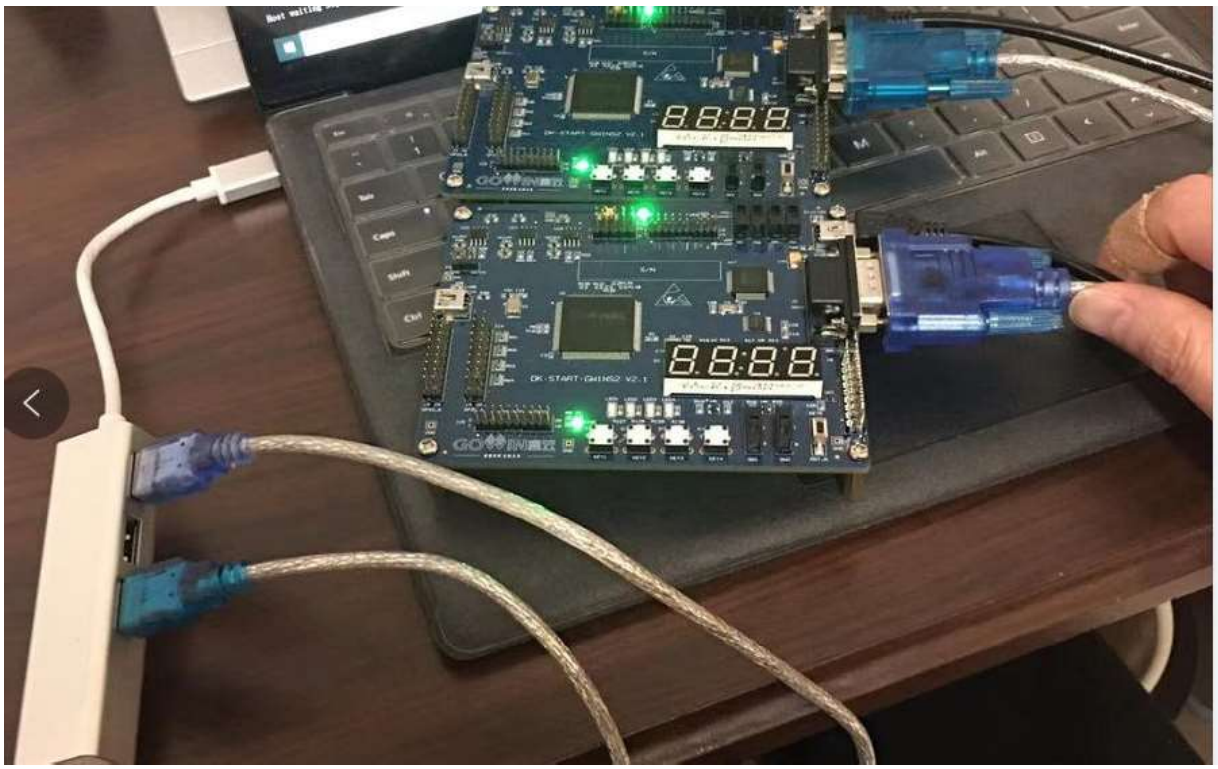
Using node.js platform for the console application. The reason why we choose node.js is that it's single threaded and non blocking. This make request response flow smoother and faster and it can be used to process a large number of requests from IOT device.Another reason is that it's event driven allowing us to run various task without waiting for other tasks to complete.

```
const SerialPort = require('serialport');

var Readline = SerialPort.parsers.Readline;
var parser = new Readline();
var HostParser = new Readline();

//client, on the firt usb port of usb-hub
const port = new SerialPort('COM3', {baudRate : 115200},function (err) {
  if (err) {
    return console.log('Error: ', err.message);
  }
});

//host, on the third usb port of usb-hub
const hostport = new SerialPort('COM11', {baudRate : 115200},function (err) {
  if (err) {
    return console.log('Error: ', err.message);
  }
});
```

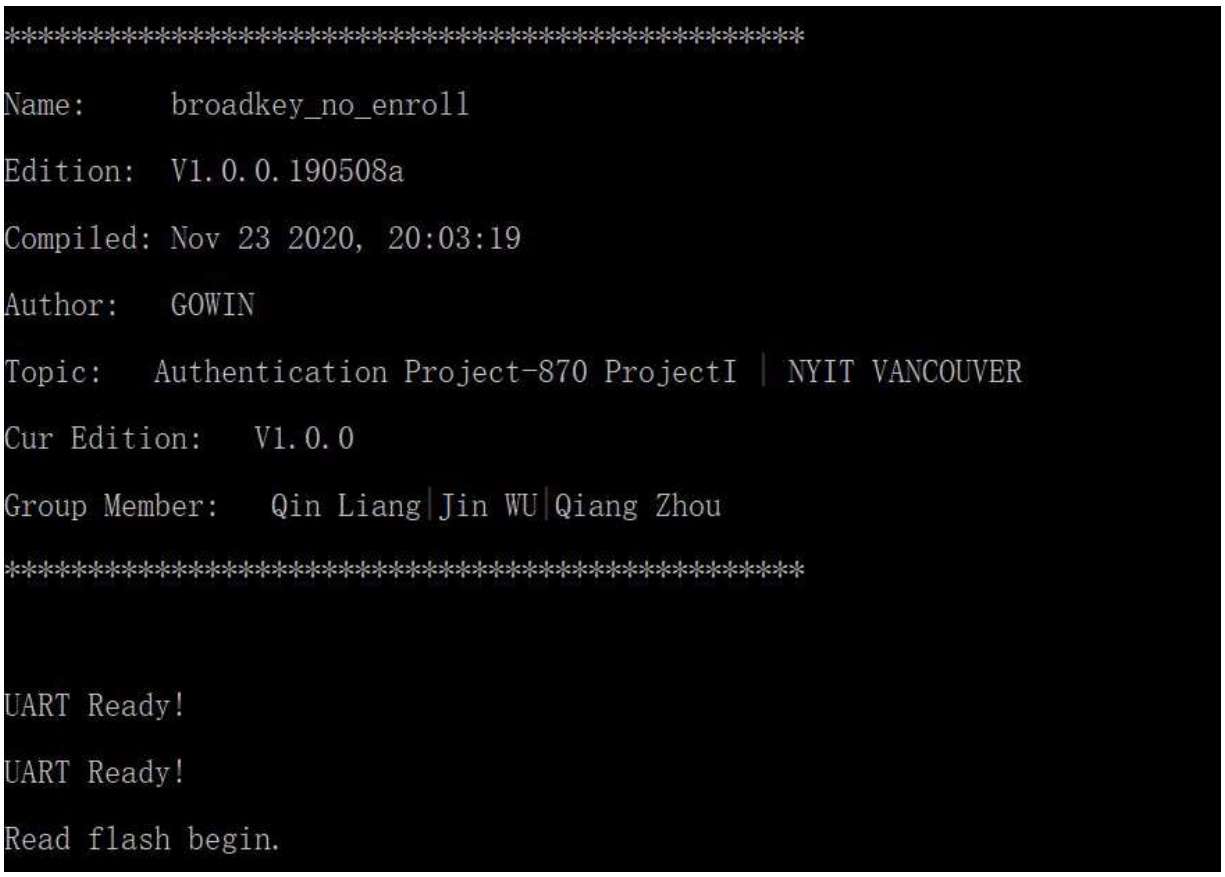
```
hostport.pipe(HostParser);

parser.on('data', clientSerialData);
HostParser.on('data', hostSerialData);

setTimeout(askForRandom, 1000);
```

5. Validation and testing

- Connect the two boards to a computer



- Initialization of the host and client boards



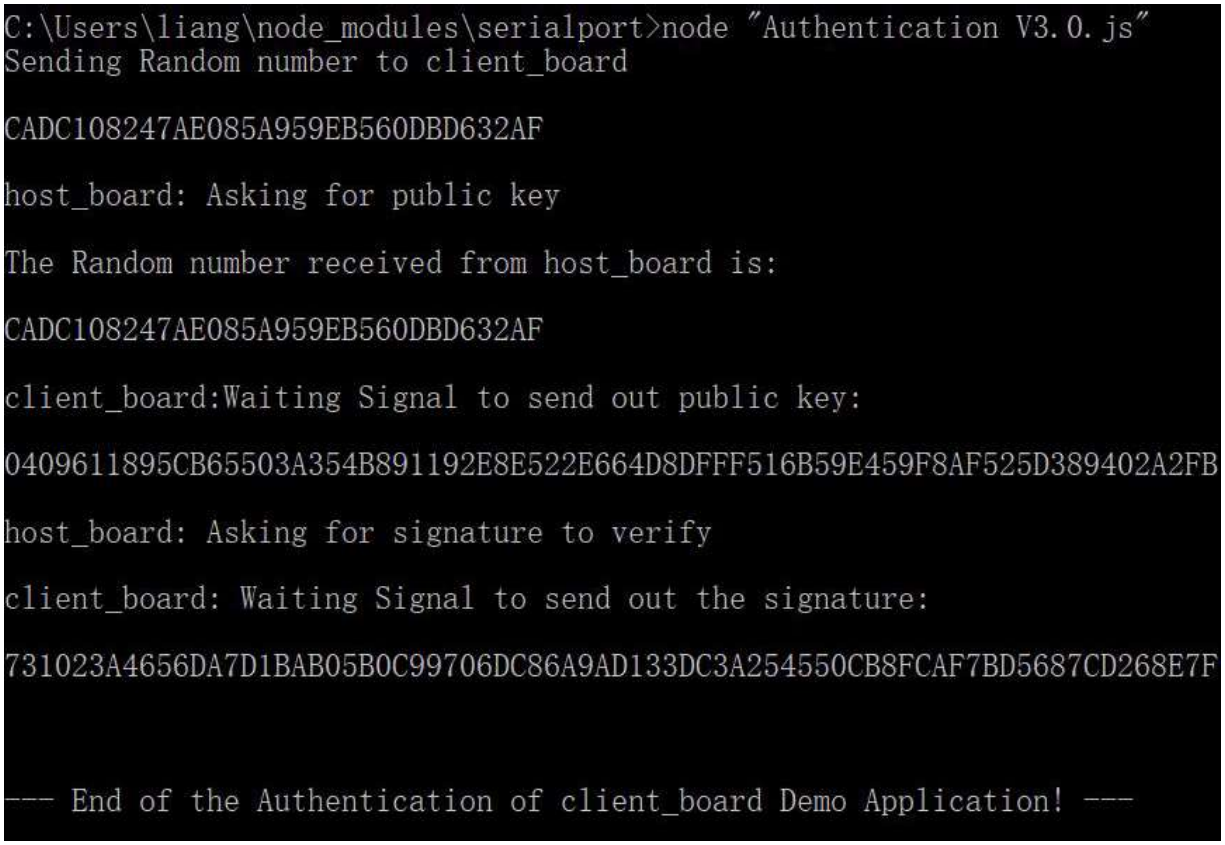
- Client board is ready, waiting for a random number sent by host board.

```
15 BB B8 F8 1D 77 AC E7 26 EA 3E A9 5F 69 EF B2
C0 89 5D F4 7A 0C BE 74 AA EC AC 1A CB 85 54 CE
56 0B EC F1 69 11 9C A6 10 B7 F6 47 D0 11 F6 00
2A 6B 72 6C CC D7 BB E3 75 60 04 8E E6 DA 1F 8C
8A 10 93 35 1B EF 8F 49 E7 79 FC D5 2B DA F3 BD
A0 79 4C 9E 27 0B 5D 42
End.


bk_start ...Board initialization done!
client_boad, waiting for Random number
```

- Host board is ready, waiting for a signal to send out random number. The signal also start of the procedure of Authentication.

```
A9 AD FF B7 C6 CE 69 7F 8A FC 3E 55 BC C9 D4 10
7E 44 F7 5C B0 EA 41 4E 98 F5 0A 05 CA C2 C3 7A
BF 9A 67 25 16 C4 35 45
End.


bk_start ...Board initialization done!
Host waiting Signal to send out the Random number:
```

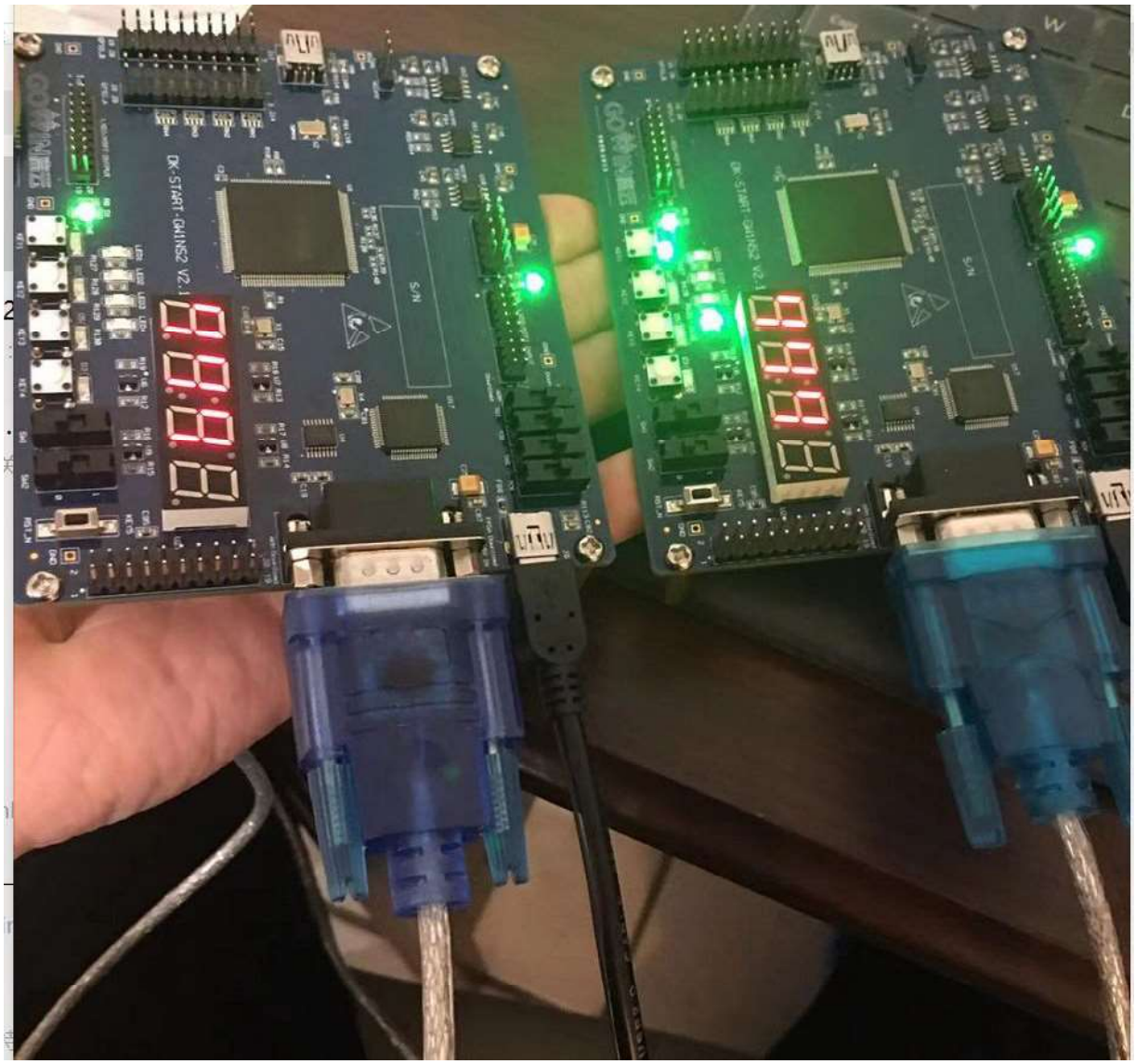
- Run authentication console application, which will send a start signal to host board then the authentication will be performed automatically.



- The host board verifies that signature signed by client board

```
The User ECC Public Key, to be imported, is:

0409611895CB65503A354B891192E8E522E664D8DFFF516B59E459F8AF525D389402A2FB5CB

 Changing public key to Ecc Public Key Code ...done!

The ECC Public Key Code created by imported public key is:

6C7856D68022F3289C2EEC5A48D98C3B000000400000000845300100000000003F68A06D09FF
060E2422DA43A3870918244033AA4BCC5EDC027

The client board signed signature need to be verified is:

731023A4656DA7D1BAB05B0C99706DC86A9AD133DC3A254550CB8FCAF7BD5687CD268E7F947

 The host board is using imported public key to verify signature:...

Signature Verified!

 done!

--- End of the Authentication of host_board Demo Application! ---
```

- The lcd and led will show signals when authentication finished successfully.



6. Conclusion

The goal of this authentication project that creates an example for Gowin's customers showing them how to perform device authentication was achieved. Two Gowin secure fpga boards were used for this authentication project. The version of the boards is DK-START-GW1NS2 V2.1. Host device sends a random number to a client device which is used to create a signature by client device. After that Host verified this signed signature using the client's public key and the random number it sent before. This is the typical authentication procedure that uses the Intrinsic ID ECDSA signature generation and verification algorithm.

Reference:

[1]. DK-START-GW1NSE2 Development Board Quick Start User Guide.
https://gowinsemi.com/upload/database_doc/902/document/5e5de5e903996.pdf
[2]. Developing Secure Hardware Systems White Paper.
https://gowinsemi.com/upload/database_doc/681/document/5d1b0f0e20cf0.pdf
[3].GOWIN DK-START-GW1NS2 V2.1 Development Kit
https://www.mouser.ca/new/gowin/gowin-dk-start-gw1ns-v21-dev-kit/
[4].DK-START-GW1NS2 V2.1 User Manual
https://www.mouser.ca/datasheet/2/1033/DBUG358-1.4E_DK-START-GW1NS2_V2.1_Development_Boar-1830730.pdf
[5]. Intrinsic ID demo key data sheet
https://synergygallery.renesas.com/media/products/130/313/en-US/IID-DK2-4-DS.pdf
[6]. Lab: Serial Communication with Node.js
https://itp.nyu.edu/physcomp/labs/labs-serial-communication/lab-serial-communication-with-node-js/