0. Preface

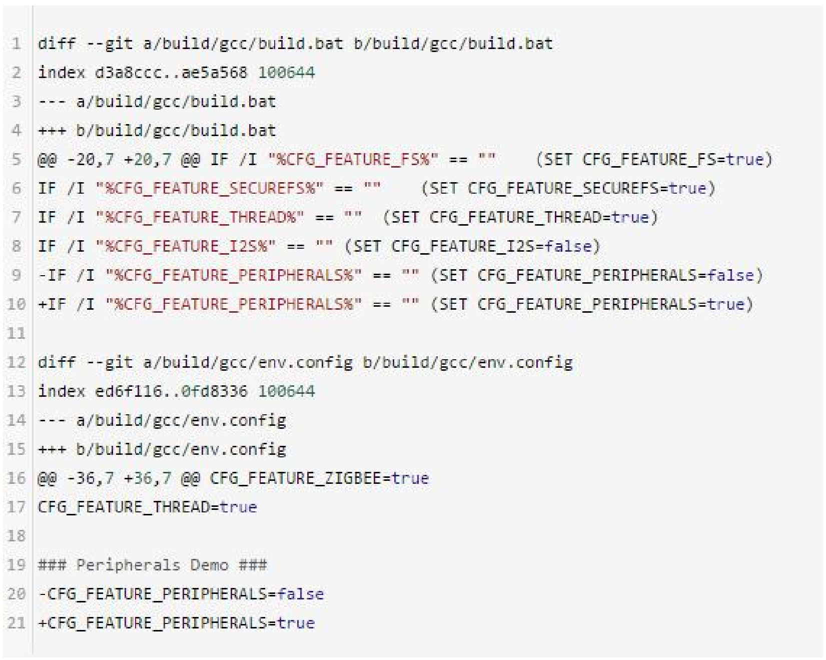
There are many demos while you are trying to use QCA4020 development kit, regarding to this demo, we decide to use QCLI_demo.The main objective of this demo is to create a smart lock using QCA4020, this is done by using an application on mobile phone to connect to the QCA4020 via BLE(Bluetooth Low Energy), and there will be PWM wave output through GPIO of QCA4020, at the same time, servo motor will have angle change at different PWM situations. Actually the servo motor of this demo is designed to simulate a smart lock.

1. Realization of PWM wave

1.1 Open Peripherals module

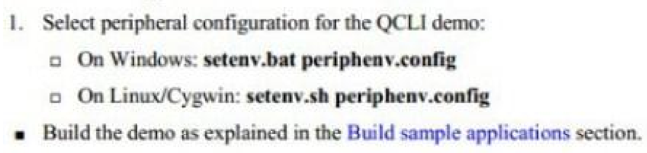
For PWM is involved in Peripherals module, before we are trying to use it,we need to modify the source code.The method is shown below:

1) Modify "build.bat"and "evn.config"



```
diff --git a/build/gcc/build.bat b/build/gcc/build.bat
index d3a8ccc..ae5a568 100644
--- a/build/gcc/build.bat
+++ b/build/gcc/build.bat
@@ -20,7 +20,7 @@ IF /I "%CFG_FEATURE_FS%" == ""    (SET CFG_FEATURE_FS=true)
IF /I "%CFG_FEATURE_SECUREFS%" == ""    (SET CFG_FEATURE_SECUREFS=true)
IF /I "%CFG_FEATURE_THREAD%" == ""  (SET CFG_FEATURE_THREAD=true)
IF /I "%CFG_FEATURE_I2S%" == "" (SET CFG_FEATURE_I2S=false)
-IF /I "%CFG_FEATURE_PERIPHERALS%" == "" (SET CFG_FEATURE_PERIPHERALS=false)
+IF /I "%CFG_FEATURE_PERIPHERALS%" == "" (SET CFG_FEATURE_PERIPHERALS=true)

diff --git a/build/gcc/env.config b/build/gcc/env.config
index ed6f116..0fd8336 100644
--- a/build/gcc/env.config
+++ b/build/gcc/env.config
@@ -36,7 +36,7 @@ CFG_FEATURE_ZIGBEE=true
CFG_FEATURE_THREAD=true

### Peripherals Demo ###
-CFG_FEATURE_PERIPHERALS=false
+CFG_FEATURE_PERIPHERALS=true
```

2) Set the compiling environent



1. Select peripheral configuration for the QCLI demo:
   - On Windows: **setenv.bat periphenv.config**
   - On Linux/Cygwin: **setenv.sh periphenv.config**

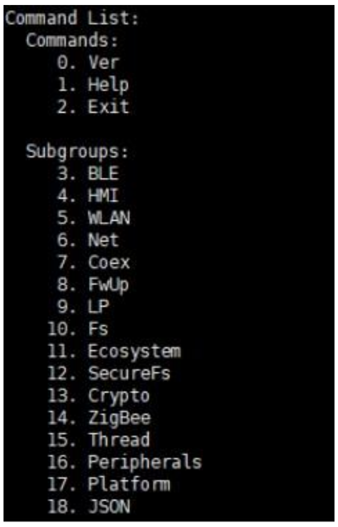
- Build the demo as explained in the Build sample applications section.

3) Compile the code and flash

Enter "./build/gcc/"directory,and executing command is as below:

```
.\build.bat prepare 4020 cdb
.\build.bat t 4020 cdb
```

Jump out pin1, 2 of the jumper J34, then download via USB, and downloading command is as below:

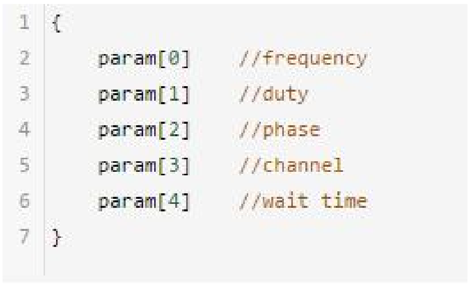


```
Command List:
  Commands:
     0. Ver
     1. Help
     2. Exit

  Subgroups:
     3. BLE
     4. HMI
     5. WLAN
     6. Net
     7. Coex
     8. FwUp
     9. LP
    10. Fs
    11. Ecosystem
    12. SecureFs
    13. Crypto
    14. ZigBee
    15. Thread
    16. Peripherals
    17. Platform
    18. JSON
```

1.2 Control servo motor via PWM

This part mainly exposed that servo motor will rotate different angles at different PWM situations. there will be divided to two function which are open_lock()" and "close_lock()".

The respective value of each element in PWM array is as below:



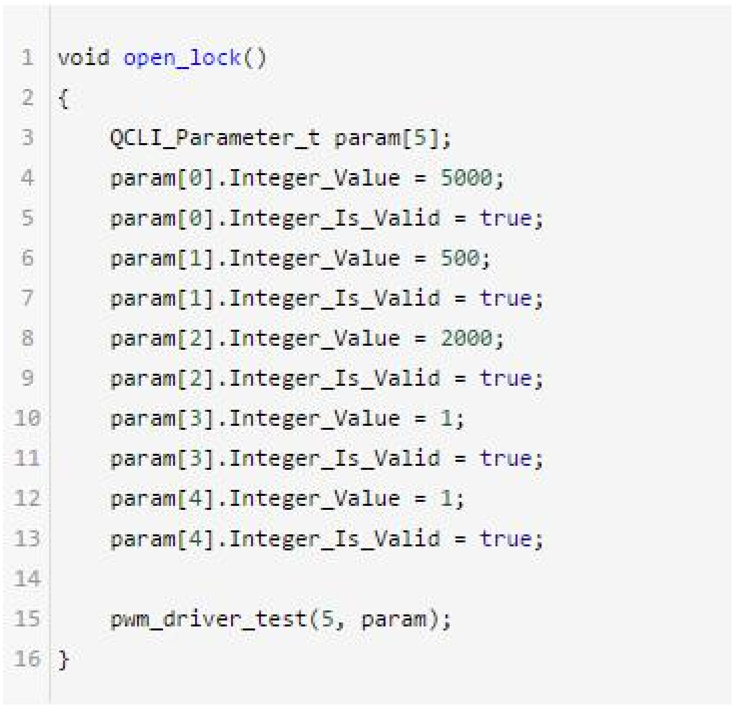
```
{
    param[0]    //frequency
    param[1]    //duty
    param[2]    //phase
    param[3]    //channel
    param[4]    //wait time
}
```

The valid time of pulse to the rotation angle of servo motor is as below sheet:

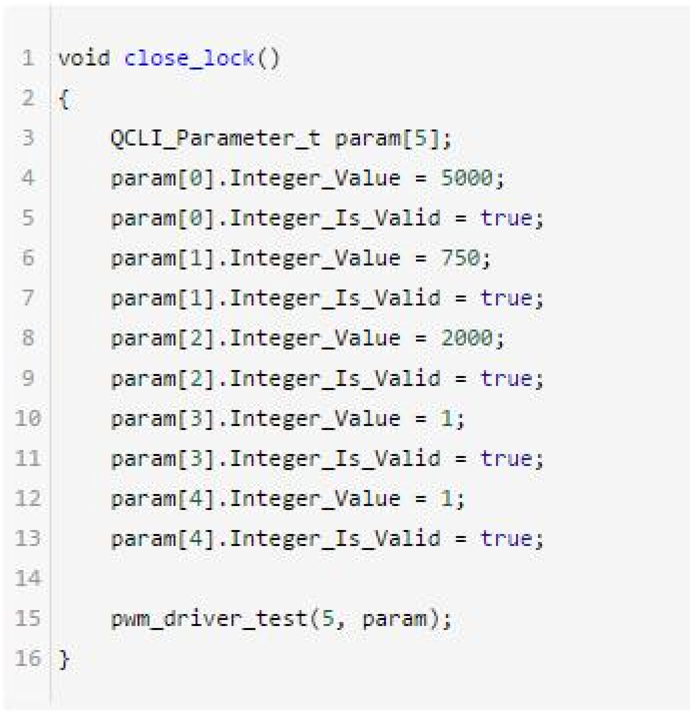
Time (ms)	Angle(°)
0.5	-90
1	-45
1.5	0
2	45
2.5	90

In this project, we just need to use the angle "-45"and "0".

add "open_lock()" and "close_lock" into .../QCLI_demo/src/spple/spple_demo.c".



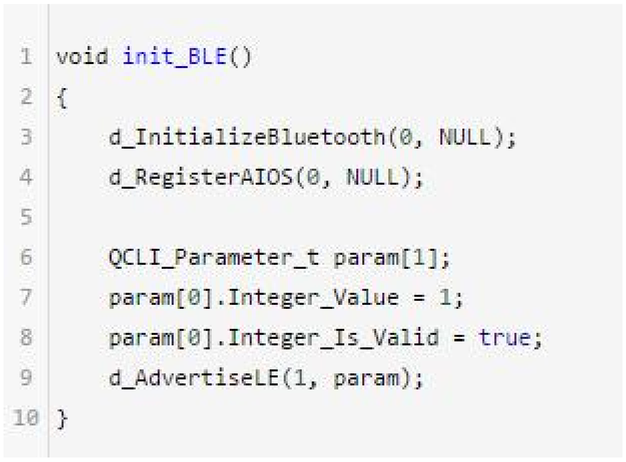
```
void open_lock()
{
    QCLI_Parameter_t param[5];
    param[0].Integer_Value = 5000;
    param[0].Integer_Is_Valid = true;
    param[1].Integer_Value = 500;
    param[1].Integer_Is_Valid = true;
    param[2].Integer_Value = 2000;
    param[2].Integer_Is_Valid = true;
    param[3].Integer_Value = 1;
    param[3].Integer_Is_Valid = true;
    param[4].Integer_Value = 1;
    param[4].Integer_Is_Valid = true;

    pwm_driver_test(5, param);
}
```



```
void close_lock()
{
    QCLI_Parameter_t param[5];
    param[0].Integer_Value = 5000;
    param[0].Integer_Is_Valid = true;
    param[1].Integer_Value = 750;
    param[1].Integer_Is_Valid = true;
    param[2].Integer_Value = 2000;
    param[2].Integer_Is_Valid = true;
    param[3].Integer_Value = 1;
    param[3].Integer_Is_Valid = true;
    param[4].Integer_Value = 1;
    param[4].Integer_Is_Valid = true;

    pwm_driver_test(5, param);
}
```

2. Realization of BLE

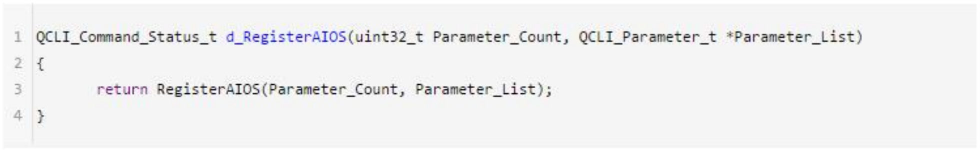
First of all,we need to add some function to init BLE and put them into ".../quartz/demo/QCLI_demo/src/qcli/pal.c",the code is as below:



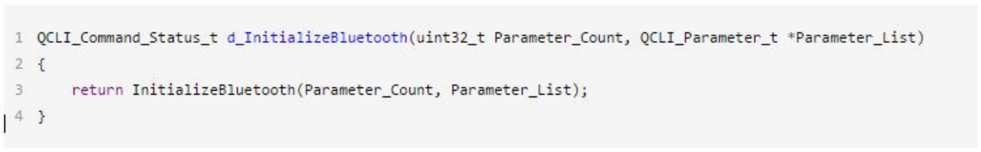
```c
void init_BLE()
{
    d_InitializeBluetooth(0, NULL);
    d_RegisterAIOS(0, NULL);

    QCLI_Parameter_t param[1];
    param[0].Integer_Value = 1;
    param[0].Integer_Is_Valid = true;
    d_AdvertiseLE(1, param);
}
```

Moreover,we still need to add the functions "d_RegisterAIOS()", "d_InitializeBluetooth()" and "d_AdvertiseLE()". The code location is in".../quartz/demo/QCLI_demo/src/spple/spple_demo.c"

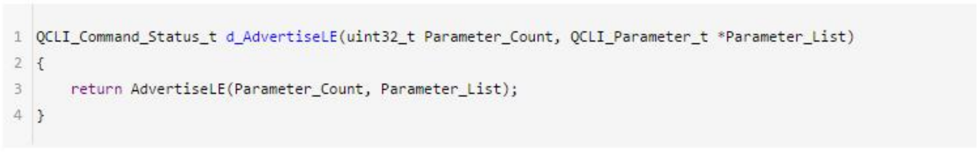
which is as below:



```c
QCLI_Command_Status_t d_RegisterAIOS(uint32_t Parameter_Count, QCLI_Parameter_t *Parameter_List)
{
        return RegisterAIOS(Parameter_Count, Parameter_List);
}
```



```c
QCLI_Command_Status_t d_InitializeBluetooth(uint32_t Parameter_Count, QCLI_Parameter_t *Parameter_List)
{
    return InitializeBluetooth(Parameter_Count, Parameter_List);
}
```



```c
QCLI_Command_Status_t d_AdvertiseLE(uint32_t Parameter_Count, QCLI_Parameter_t *Parameter_List)
{
    return AdvertiseLE(Parameter_Count, Parameter_List);
}
```

Last but not least,we need to add code into"callback function"of AIOS service, then we can control the sever motor, and the concreting operation is as below:

```c
  case QAPI_BLE_ACT_ANALOG_E:
      /* Store a pointer to the Characteristic instance  */
      /* data for readability.                           */
      /* * NOTE * We will use the Characteristic         */
      /*          information to access it.              */
      if((ServerInstancePtr = GetAIOSServerInstanceInfoPtr(CharacteristicInfo)) != NULL)
      {
          /* Store the Analog Characteristic.             */
          ServerInstancePtr->Data.Analog = Data.Analog;
+          if(ServerInstancePtr->Data.Analog > 0)
+         {
+             QCLI_Printf(aios_group, "Open lock--------------.\n");
+             open_lock();
+         }
+         else
+         {
+             QCLI_Printf(aios_group, "Close lock--------------.\n");
+             close_lock();
+         }

          /* Display the Analog Characteristic.           */
          DisplayAnalogCharacteristic(ServerInstancePtr->Data.Analog, CharacteristicInfo->ID);

          /* Send the write request response.             */
          if((Result = qapi_BLE_AIOS_Write_Characteristic_Request_Response(BluetoothStackID, InstanceID, TransactionID, QAPI_BLE_AIOS_ERROR_CODE_SUCCESS, CharacteristicInfo)) != 0)
          DisplayFunctionError("qapi_BLE_AIOS_Write_Characteristic_Request_Response", Result);
       }
```