

```

/*****/
/*
/*          maman
/*          le 20/08/2015
/*
/*
/*
/*      La carte chipkit pilote le module wifi MRF24WG pour envoyer des
/*      notifications sur smartphones par le biais de
/*      http://www.pushingbox.com
/*
/*
/*****/

/*****/
/*          Required libraries
/*
/*****/

#include "user.h"
#include "delay_edge.h"
#include <EEPROM.h>

#include <WiFiShieldOrPmodWiFi_G.h>    // This is for the MRF24WGxx on a pmodWiFi or WiFiShield
#include <DNETcK.h>
#include <DWIFicK.h>

/**
 *
 *      Parameters:
 *      None
 *
 *      Return Values:
 *      None
 *
 *      Description:
 *
 *      Arduino setup function.
 *
 *      Initialize the Serial Monitor, and initializes the
 *      connection
 *      Use DHCP to get the IP, mask, and gateway
 *      by default we connect to port 44300
 *
 * -----*/
void setup() {
    Serial.begin(9600);
    Serial.setTimeout(20);

    pinMode(BPp,INPUT);
    pinMode(BP1,INPUT);
    pinMode(ledp,OUTPUT);
    pinMode(led,OUTPUT);
    pinMode(ledServer,OUTPUT);
    pinMode(6,OUTPUT); // sortie libre
    //pinMode(0,OUTPUT); // sortie reset wifi

    digitalWrite(6,LOW);

```

```

    //digitalWrite(0,HIGH);
    digitalWrite(ledp,LOW);
    digitalWrite(led,LOW);
    digitalWrite(ledServer,LOW);

    WDTCONbits.ON = HIGH;    //WDT on
    if (EEPROM.read(EEP4) == HIGH) stateWork = LOADING;
}

/**
 *
 * Parameters:
 *     None
 *
 * Return Values:
 *     None
 *
 * Description:
 *
 * Arduino loop function.
 *
 * We are using the default timeout values for the DNETcK and TcpClient class
 * which usually is enough time for the Tcp functions to complete on their first call.
 *
 * This code will write some stings to the server and have the server echo it back
 *
 * -----*/
void loop() {
    WDTCONbits.WDTCLR = HIGH;    //Kick the dog!

    switch(stateWork) {
        case STOPPING:
            if (button(BPp,NO,&pinState_BPp,TBPp.top(50),&TBPp.Treset)) {
                Serial.println("WiFi TCP 1.0");
                Serial.println("Application pushingbox");
                Serial.println("Nunes, Copyright 2015");
                Serial.println();
                T2.reset();
            }
            else if ((pinState_BPp == endt0) && digitalRead(BPp) && T2.top(2000)) {
                stateWork = CONFIG;
                stateConfig = STARTCONFIG;
            }
            else if (!pinState_BPp && !digitalRead(BPp)) {
                EEPROM.write(EEP4,HIGH);
                stateWork = LOADING;
            }
            break;

        case CONFIG:
            break;

        case LOADING:

```

```

for (int i=0;EEPROM.read(i);i++,codePush[i]=0) {
  codePush[i] = EEPROM.read(i);
  if (i == 29) break;
}
for (int i=0;EEPROM.read(i+EEP1);i++,SSID[i]=0) {
  SSID[i] = EEPROM.read(i+EEP1);
  if (i == 29) break;
}
for (int i=0;EEPROM.read(i+EEP2);i++,codeWifi[i]=0) {
  codeWifi[i] = EEPROM.read(i+EEP2);
  if (i == 29) break;
}
valSecurity = EEPROM.read(EEP3);
stateWifi = START;
stateWork = RUNNING;
digitalWrite(ledp,HIGH);
Tled.reset();
break;

case RUNNING:
  if (!wifi_ok && Tled.top(200)) digitalWrite(led,ledState = !ledState);
  if (button(BPp,NO,&pinState_BPp,TBPp.top(100),&TBPp.Treset)) stateWifi = NOWIFI;
  else if (!pinState_BPp) {
    digitalWrite(ledp,LOW);
    digitalWrite(led,LOW);
    EEPROM.write(EEP4,LOW);
    stateWork = STOPPING;
  }
  if (button(BP1,NO,&pinState_BP1,TBP1.top(50),&TBP1.Treset)) {
    //strcpy(texte,text0);
    order = WRITE;
  }
  break;

default:
  break;
}

switch(stateConfig) {
  case STARTCONFIG:
    Serial.print("Configuration: Internet of Things");
    Serial.println();
    Serial.print("Code Pushingbox: ");
    while (digitalRead(BPp)) WDTCONbits.WDTCLR = HIGH; //Kick the dog!
    delay(100);
    iEEP = 0;
    stateConfig = CODEPBOX;
    break;

  case CODEPBOX:
    if (digitalRead(BPp)) {
      Serial.println("STOP");
      delay(100);
      while (digitalRead(BPp)) WDTCONbits.WDTCLR = HIGH; //Kick the dog!
    }

```

```

    stateWifi = SCAN;
    stateConfig = CODEINET;
}
else if (Serial.available()) {
    incomingByte = Serial.read();
    Serial.print(incomingByte);
    EEPROM.write(ieep++,incomingByte);
    if (!Serial.available() || (ieep > (sizeof(codePush)) - 2)) {
        EEPROM.write(ieep,0);
        if (Serial.available()) Serial.flush();
        Serial.println();
        Serial.println("Waiting answer");
        stateConfig = ANSWER;
        T2.reset();
    }
}
break;

case ANSWER:
    if (digitalRead(BPp)) {
        Serial.println("STOP");
        delay(100);
        while (digitalRead(BPp)) WDTCONbits.WDTCLR = HIGH; //Kick the dog!
        stateConfig = END;
    }
    else if (T2.top(1000)) {
        if (count0 < 5) {
            if (count0 != 4) {
                Serial.print("Scan to ");
                Serial.print(4-count0,DEC);
                Serial.println("s");
                count0++;
            }
            else {
                count0 = 0;
                stateWifi = SCAN;
                stateConfig = CODEINET;
            }
        }
    }
break;

case CODEINET:
    if ((stateWifi == WAIT) && !edge0) edge0 = HIGH, Serial.print("Choice index to scan: ");
    else if (Serial.available()) {
        incomingInt = Serial.parseInt();
        if (incomingInt > iNet) Serial.println("Error"), Serial.print("Choice index to scan: ");
        else {
            Serial.println(incomingInt,DEC);
            Serial.print("SSID: ");
            Serial.println(scanWifi[incomingInt].szSsid);
            strcpy(SSID,scanWifi[incomingInt].szSsid);
            EEPROM.write(EEP3,scanWifi[incomingInt].securityType);
            for (ieep=0;ieep < sizeof(SSID);ieep++) EEPROM.write(ieep+EEP1,SSID[ieep]);

```

```

        EEPROM.write(ieep+EEP1,0);
        ieep = 0;
        Serial.print("Code wifi: ");
        stateConfig = CODEWIFI;
    }
}
break;

case CODEWIFI:
    if (Serial.available()) {
        incomingByte = Serial.read();
        Serial.print(incomingByte);
        EEPROM.write(ieep+++EEP2,incomingByte);
        if (!Serial.available() || (ieep > (sizeof(codeWifi)) - 2)) {
            EEPROM.write(ieep+EEP2,0);
            if (Serial.available()) Serial.flush();
            Serial.println();
            stateConfig = END;
        }
    }
    break;

case END:
    edge0 = LOW;
    Serial.println("End configuration");
    Serial.println("Close monitor");
    EEPROM.write(EEP4,HIGH);
    while (1);
    break;

case STOPCONFIG:
    break;

default:
    break;
}

switch(stateWifi) {
    case NONE:
        if(DWIFICK::isConnected(conID,&status)) {
            if (!wifi_ok) {
                wifi_ok = HIGH;
                digitalWrite(led,HIGH);
                Serial.println("Is Connected");
                Serial.print("connection status: ");
                Serial.println(status,DEC);
            }
        }
        else if(DNETCK::isStatusAnError(status)) {
            Serial.println("WiFi not connected");
            order = RESTART;
        }

        if ((order == WRITE) || (order == RESTART)) {

```

```

    //strcpy(text_pushingbox,texte);
    stateWifi = order;
    order = NONE;
}
break;

case START:
    if (!valSecurity) conID = DWIFICK::connect(SSID,&status); // no security - OPEN
    else if (valSecurity == 3) {
        conID =
        DWIFICK::connect(DWIFICK::WF_SECURITY_WPA_WITH_PASS_PHRASE,SSID,codeWifi,&status);
    }
    else if (valSecurity == 5) {
        conID =
        DWIFICK::connect(DWIFICK::WF_SECURITY_WPA2_WITH_PASS_PHRASE,SSID,codeWifi,&status);
    }
    else if (valSecurity == 7) {
        conID =
        DWIFICK::connect(DWIFICK::WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE,SSID,codeWifi,&status);
    }
    else conID = DWIFICK::connect(DWIFICK::WF_SECURITY_WEP_40,SSID,codeWifi,&status);

    if(conID != DWIFICK::INVALID_CONNECTION_ID) {
        Serial.print("Connection Created, ConID = ");
        Serial.println(conID,DEC);
    }
    else {
        Serial.print("Unable to connection, status: ");
        Serial.println(status,DEC);
    }
    // use DHCP to get our IP and network addresses
    DNETcK::begin();
    stateWifi = TEMPOWIFI;
    T4.reset();
    break;

case TEMPOWIFI:
    if(T4.top(30000)) stateWifi = DONE;
    break;

case RESTART:
    Serial.println("Disconnecting WiFi");
    startWait = START;
    stateWifi = DISCONNECT;
    break;

case NOWIFI:
    startWait = WAIT;
    stateWifi = DISCONNECT;
    break;

case DISCONNECT:
    wifi_ok = LOW;
    digitalWrite(led,LOW);

```

```

IEC0bits.INT2IE = LOW;
// terminate our internet engine
DNETcK::end();
//disconnect the WiFi, this will free the connection ID as well.
DWIFICK::disconnect(conID);
IFS0bits.INT2IF = LOW;
IEC0bits.INT2IE = HIGH;
stateWifi = startWait;
break;

// write out the strings
case WRITE:
    // make a connection to server
    tcpClient.connect(serverName,portServer);

    if(tcpClient.isConnected()) {
        Serial.println("connected to the TCP server");
        //Sending request to PushingBox when the pin is HIGHT
        sendToPushingBox(codePush,text_pushingbox);
        digitalWrite(ledServer,HIGH);
        stateWifi = READ;
        Tl.reset();
    }
    else {
        Serial.println("connection TCP server failed");
        Serial.println("to temporize for an other request");
        digitalWrite(led,LOW);
        stateWifi = TEMPOTCP;
        Ttcp.reset();
    }
    break;

case TEMPOTCP:
    if (Ttcp.top(40000)) {
        IEC0bits.INT2IE = LOW;
        wf_reset = LOW; //Attention! faire ceci reset chipkit !!!
        delay(20);
        wf_reset = HIGH;
        stateWifi = RESTART;
    }
    break;

// look in the strings
case READ:
    // see if we got anything to read
    if((cbRead = tcpClient.available()) > 0) {
        cbRead = cbRead < sizeof(rgbRead) ? cbRead : sizeof(rgbRead);
        cbRead = tcpClient.readStream(rgbRead,cbRead);
        for(int i=0;i < cbRead; i++) Serial.print((char) rgbRead[i]);
    }
    // give us some time to get everything echo'ed back
    else if (Tl.top(5000)) {
        Serial.println();
        stateWifi = CLOSE;
    }

```

```

    }
    break;

// done, so close up the tcpClient
case CLOSE:
    tcpClient.close();
    Serial.println("Closing TcpClient!");
    digitalWrite(ledServer,LOW);
    stateWifi = DONE;
    break;

case SCAN:
    // Set my default wait time to nothing
    DNETcK::setDefaultBlockTime(DNETcK::msImmediate);
    // start a scan
    DWIFicK::beginScan();
    Serial.println("Scanning...");
    stateWifi = WAITFORSCAN;
    break;

case WAITFORSCAN:
    if(DWIFicK::isScanDone(&cNetworks,&status)) {
        Serial.println("Scan done");
        stateWifi = PRINTAPINFO;
    }
    else if(DNETcK::isStatusAnError(status)) {
        Serial.print("Scan Failed");
        stateWifi = ERROR;
    }
    break;

case PRINTAPINFO:
    if(iNetwork < cNetworks) {
        DWIFicK::SCANINFO scanInfo;
        int j;
        if(DWIFicK::getScanInfo(iNetwork,&scanInfo)) {
            Serial.println();
            Serial.print("Scan info for index: ");
            Serial.println(iNetwork,DEC);
            Serial.print("SSID: ");
            Serial.println(scanInfo.szSsid);
            Serial.print("Secuity type: ");
            //Serial.println(scanInfo.securityType,DEC);
            if (!scanInfo.securityType) Serial.println("OPEN");
            else if (scanInfo.securityType == 3) Serial.println("WPA with key");
            else if (scanInfo.securityType == 5) Serial.println("WPA2 with key");
            else if (scanInfo.securityType == 7) Serial.println("WPA auto with key");
            else Serial.println("WEP_40");
            Serial.print("Channel: ");
            Serial.println(scanInfo.channel,DEC);
            Serial.print("Signal Strength: ");
            Serial.println(scanInfo.signalStrength,DEC);
            Serial.print("Count of supported bit rates: ");
            Serial.println(scanInfo.cBasicRates,DEC);

```



```

Serial.print("    Supported Rate: ");
for( j=0; j < scanInfo.cBasicRates; j++) {
    Serial.print(scanInfo.basicRates[j],DEC);
    Serial.print(" bps    ");
}
Serial.println();
Serial.print("SSID MAC: ");
for(j=0; j < sizeof(scanInfo.ssidMAC); j++) {
    if(scanInfo.ssidMAC[j] < 16) Serial.print(0,HEX);
    Serial.print(scanInfo.ssidMAC[j],HEX);
}
Serial.println();
Serial.print("Beacon Period: ");
Serial.println(scanInfo.beconPeriod,DEC);
Serial.print("dtimPeriod: ");
Serial.println(scanInfo.dtimPeriod,DEC);
Serial.print("atimWindow: ");
Serial.println(scanInfo.atimWindow,DEC);
}
else {
    Serial.print("Unable to get scan info for iNetwork: ");
    Serial.println(iNetwork,DEC);
}
iNet = iNetwork;
scanWifi[iWifi++] = scanInfo;
iNetwork++;
}
else {
    iNetwork = 0;
    iWifi = 0;
    stateWifi = STOP;
}
break;

case ERROR:
    Serial.print("Status value: ");
    Serial.println(status,DEC);
    stateWifi = STOP;
    break;

case STOP:
    Serial.println("End of Scan");
    Serial.println();
    DNETcK::end();
    stateWifi = WAIT;
    break;

case WAIT:
    break;

case DONE:
    stateWifi = NONE;

default:

```

```
        break;
    }
    // keep the stack alive each pass through the loop()
    DNETcK::periodicTasks();
}
```