```
/*

*/

#include <SoftwareSerial.h>

#define ESP_SERIAL_RX 4
#define ESP_SERIAL_TX 3

SoftwareSerial espSerial(ESP_SERIAL_RX, ESP_SERIAL_TX, false);

#define CHAR_NULL '\0'

#define SPI_BUF_SEND_LEN 200
#define SPI_BUF_RECV_LEN 200

//charspiBufSend[SPI_BUF_SEND_LEN];
//charspiBufRecv[SPI_BUF_RECV_LEN];

char spiBufRecv[ SPI_BUF_RECV_LEN ];
bool spiRecvReady = false; // a command is received from spi and ready to be
processed to the network output

char spiBufSend[ SPI_BUF_SEND_LEN ];
int spiBufSendPos = 0;
//bool spiSendReady = false; // a command is received from esp and ready to be
processed to the spi output

int port;

void setup() {
  Serial.begin( 9600);

  Serial.print( "\n****************");
  Serial.print( "\n****************");
  Serial.print( "\n****************");
  Serial.print( "\n");
  espSerial.begin( 9600 );
}



void loop() {

  static int stage = 0;
```

```
switch ( stage)
{
  case 0: // reset & init
    espSendCommand( "RST", "ready" );
    Serial.print("\nyeah ");
    Serial.print(stage);
    stage++;
    break;

  case 1:
   espSendCommand("RFPOWER=82", "OK");
    Serial.print("\nyeah ");
    Serial.print(stage);
    stage++;
    break;

  case 2:
   espSendCommand( "CWMODE_CUR=1", "OK");
    Serial.print("\nyeah ");
    Serial.print(stage);
    stage = 4;
    break;

  //    case 3:
 //     espSendCommand( "CWSAP_CUR=\"ESP_1\",\"mahadaga1\",1,4,1,0", "OK");
  //      Serial.print("\nyeah ");
  //      Serial.print(stage);
  //      stage=4;
  //      break;

  case 4:
    espSendCommand( "CWLAP", "OK");
    Serial.print("\nyeah ");
    Serial.print(stage);
    stage++;
    break;

  case 5:
   espSendCommand( "CWJAP_CUR=\"AW2\",\"dudelange\"", "OK");
    Serial.print("\nyeah ");
    Serial.print(stage);
    stage++;
    break;

  case 6:
    espSendCommand( "CIFSR", "OK");
```

```
     Serial.print("\nyeah ");
     Serial.print(stage);
     stage++;
     break;

    case 7:
     espSendCommand( "CIPMUX=1", "OK");
     Serial.print("\nyeah ");
     Serial.print(stage);
     stage++;
     break;

    case 8:
     espSendCommand( "CIPSERVER=0", "OK");
     Serial.print("\nyeah ");
     Serial.print(stage);
     stage++;
     break;

    case 9:
     espSendCommand( "CIPSERVER=1", "OK");
     Serial.print("\nyeah ");
     Serial.print(stage);
     stage++;
     break;

    case 10:

     //    while ( espSerial.available())
     //     {
     //      char receivedChar;
     //      receivedChar = espSerial.read();
     //
     //      Serial.print( receivedChar );
     //     }


     espServer();
     ePIC();
     break;

  }

}

#define RECEIVE_BUFFER_LEN 20
```

```
bool espSendCommand( char* command, char* expectedResponse )
{
  char buffer[RECEIVE_BUFFER_LEN];
  bool match = false;
  int responsePosition = 0;
  long time;
  bool ret = true;


  for ( int inx = 0; inx < RECEIVE_BUFFER_LEN; inx++)
  {
    buffer[inx] = CHAR_NULL;
  }

  Serial.print( "\n\n****************\n");

  espSerial.print( "AT+" );
  espSerial.print( command );
  espSerial.print( "\r\n" );

  while ( match == false )
  {
    if ( espSerial.available())
    {
      char receivedChar;
      receivedChar = espSerial.read();

      Serial.print( receivedChar );

      if ( receivedChar == expectedResponse[responsePosition] )
      {
        responsePosition++;
        if (expectedResponse[responsePosition] == CHAR_NULL)
        {
          match = true;
        }
      }
      else
      {
        responsePosition = 0;
      }
    }
  }
  return ret;
}
```

```
void ePIC( void )
{
  // here we mimic the PIC getting data fromt he ESP and sending it to the power box
over SPI
  // we also emulate receiving data from the power box SPI and buffer it to the ESP
  // use the serial terminal to emulate this

  //this should mimic how the PIC sends and receives data on the SPI line

  ePICSend();
  ePICRecv();

}

void ePICRecv( void )
{
  // receive from SPI
  // for this demo we recieve from Serial

  static int spiBufRecvPos = 0;
  static bool inCommand;

  while ( Serial.available() )
  {
    char rChar;
    rChar = Serial.read();

    if ( inCommand == false)
    {
      if ( rChar == '!' )
      {
        inCommand = true;
      }
    }

    if ( inCommand == true )
    {
      spiBufRecv[ spiBufRecvPos ] = rChar;
      spiBufRecvPos++;

      if ( spiBufRecvPos >= SPI_BUF_RECV_LEN )
      {
        spiBufRecvPos = ( SPI_BUF_RECV_LEN - 1 );
      }
      spiBufRecv[ spiBufRecvPos ] = CHAR_NULL;

      if ( rChar == '*' )
      {
        inCommand = false;

        espSend( spiBufRecv );
        spiBufRecvPos = 0;
      }
    }
  }
}

void ePICSend( void )
{
  // send to SPI
  // for this demo we send to Serial

  if ( spiBufSend[0] != CHAR_NULL )
  {
    Serial.print( "spi s:" );
    Serial.print( spiBufSend );
    Serial.print( "\r\n");
    spiBufSend[0] = CHAR_NULL;
  }
  spiBufSendPos = 0;
}

void ePICAddCommand( char* command )
{
  int commandPos = 0;
  bool endCommand = false;

  while ( endCommand == false )
  {
    spiBufSend[ spiBufSendPos ] = command[ commandPos ];
    if ( command[ commandPos ] == '*' )
    {
      endCommand = true;
    }
    spiBufSendPos++;
    if ( spiBufSendPos >= SPI_BUF_SEND_LEN )
    {
      spiBufSendPos = (SPI_BUF_SEND_LEN - 1);
    }
    commandPos++;
    //    if( commandPos >= ESP_RECV_COMMAND_BUF_LEN )
    //     {
    //       // we should neevr get here
    //       // if we do there are bigger problems
    //     }
  }
  spiBufSend[ spiBufSendPos ] = CHAR_NULL;

}

void espSend( char* espCommand )
{
  // espCommand should be a null terminated string

  int len = 0;
  bool commandEnd = false;


  while ( commandEnd == false )
  {

    if (  espCommand[len] == '*' )
    {
      commandEnd = true;
    }
    len++;
  }

  // set up ESP command
  delay(25);
  espSerial.print( "AT+CIPSENDBUF=");
  espSerial.print( port );
  espSerial.print( "," );
  espSerial.print( len  );
  espSerial.print( "\r\n");

//  Serial.print( "AT+CIPSEND=");
//  Serial.print( port );
//  Serial.print( "," );
//  Serial.print( len  );
//  Serial.print( "\r\n");
  delay(25);
  Serial.print ("spi r:");
  for ( int inx = 0; inx < len; inx++)
  {
    espSerial.print( espCommand[inx] );
    Serial.print( espCommand[inx]);
  }
  Serial.print( "\r\n");
}



void espServer( void )
{
  // +IPD, port, count: DATA
  // +IPD, 4, 11, !get;power*

  // we ar at start waiting to process commenad from ESP8266
  // our buffer always starts with what is received from ESP8266 and then we process
when we receive /r/n
#define ESP_BUF_LEN 200


  static char espBuf[ESP_BUF_LEN];
  static int espBufPos = 0;

  while (espSerial.available())
  {
    bool process;
    process = false;
    char rChar;
    rChar = espSerial.read();

    //    Serial.print( rChar );

    switch ( rChar )
    {
      case '\r':
        break;
      case '\n':
        process = true;
        break;
      default:
        espBuf[ espBufPos ] = rChar;
        espBufPos++;
        if ( espBufPos >= ESP_BUF_LEN )
        {
          espBufPos = (ESP_BUF_LEN - 1);
        }
        espBuf[ espBufPos ] = CHAR_NULL;
    }

    if ( process == true)
    {

      if ( strncmp( "+IPD,", espBuf, 5) == 0 )
      {
        // get port - assume always a single digit in position 5
#define ESP_BUF_DATA_START_PORT 5
#define ESP_BUF_DATA_START_LEN 7
        int dataLen;
        char tempBuf[3];
       tempBuf[0] = espBuf[ESP_BUF_DATA_START_PORT];
        tempBuf[1] = CHAR_NULL;

        port = atoi( tempBuf );

       tempBuf[0] = espBuf[ESP_BUF_DATA_START_LEN];
        switch (  espBuf[ESP_BUF_DATA_START_LEN + 1] )
        {
          case ':':
            tempBuf[1] = CHAR_NULL;
            espBufPos = 9;
            break;
          default:
           tempBuf[1] = espBuf[ESP_BUF_DATA_START_LEN + 1];
            tempBuf[2] = CHAR_NULL;
            espBufPos = 10;
        }

        dataLen = atoi( tempBuf );

        // parse through the data and build EMMS commadn string from it
        // starts with '!' - ends with '*'
        // dont care what is in between, but ignore things outside
        // so
        //  step through data and look for start char
        //  then add to data until end char
        //  it doesn't matter if we exit this loop or not
        //  keep EMMS command going until we read '*'

        static bool inCommand = false;
#define ESP_RECV_COMMAND_BUF_LEN 200
       static char espRecvBufCommand[ESP_RECV_COMMAND_BUF_LEN];
        static int espRecvBufCommandPos = 0;
        espBufPos = 0;
        while ( espBuf[ espBufPos ] != CHAR_NULL )
        {
          if ( inCommand == false )
           {
            if ( espBuf[ espBufPos ] == '!' )
             {
              inCommand = true;

             }
           }
          if ( inCommand == true )
           {

           espRecvBufCommand[ espRecvBufCommandPos ] = espBuf[ espBufPos ];
            espRecvBufCommandPos++;
           if ( espRecvBufCommandPos >= ESP_RECV_COMMAND_BUF_LEN )
             {
              espRecvBufCommandPos = ( ESP_RECV_COMMAND_BUF_LEN - 1 );
             }

            if ( espBuf[ espBufPos ] == '*' )
             {
              espRecvBufCommand[ espRecvBufCommandPos ] = CHAR_NULL;
              inCommand = false;

              ePICAddCommand( espRecvBufCommand );
              espRecvBufCommandPos = 0;
             }



           }
          espBufPos++;
          if ( espBufPos >= ESP_BUF_LEN )
           {
            // this should never happen - can't handle it right now
           }
        }
      }
      espBufPos = 0;
    }
  }

}
```