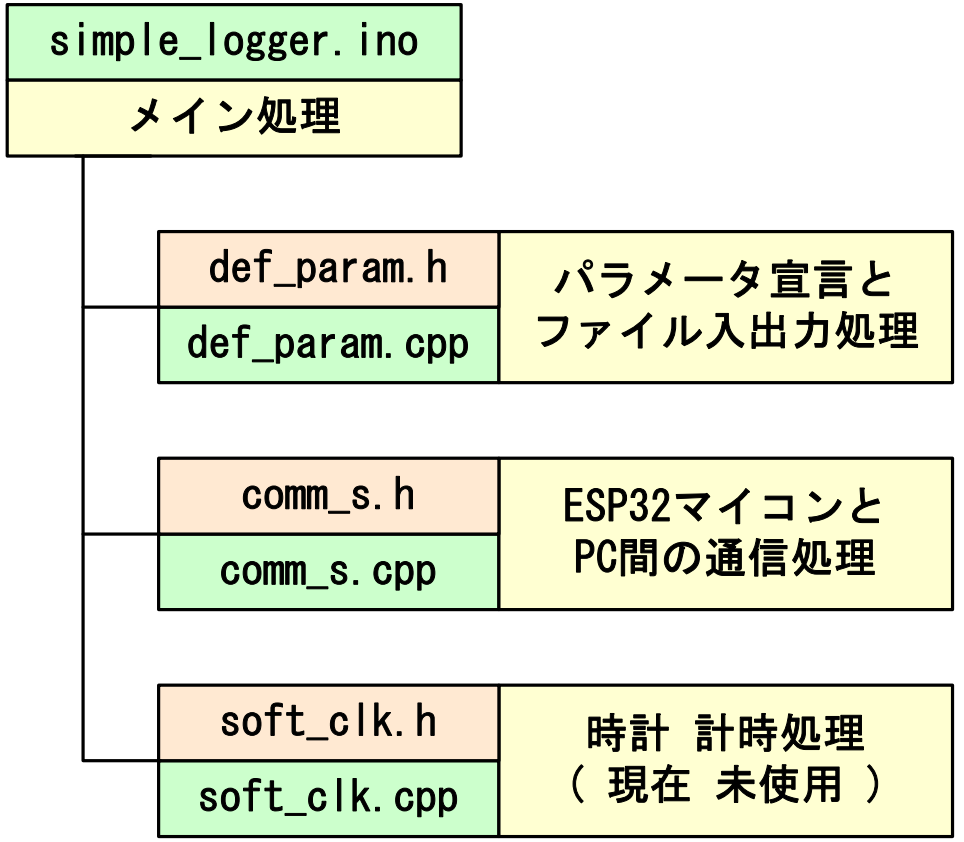
前回のプログラムの説明に関して

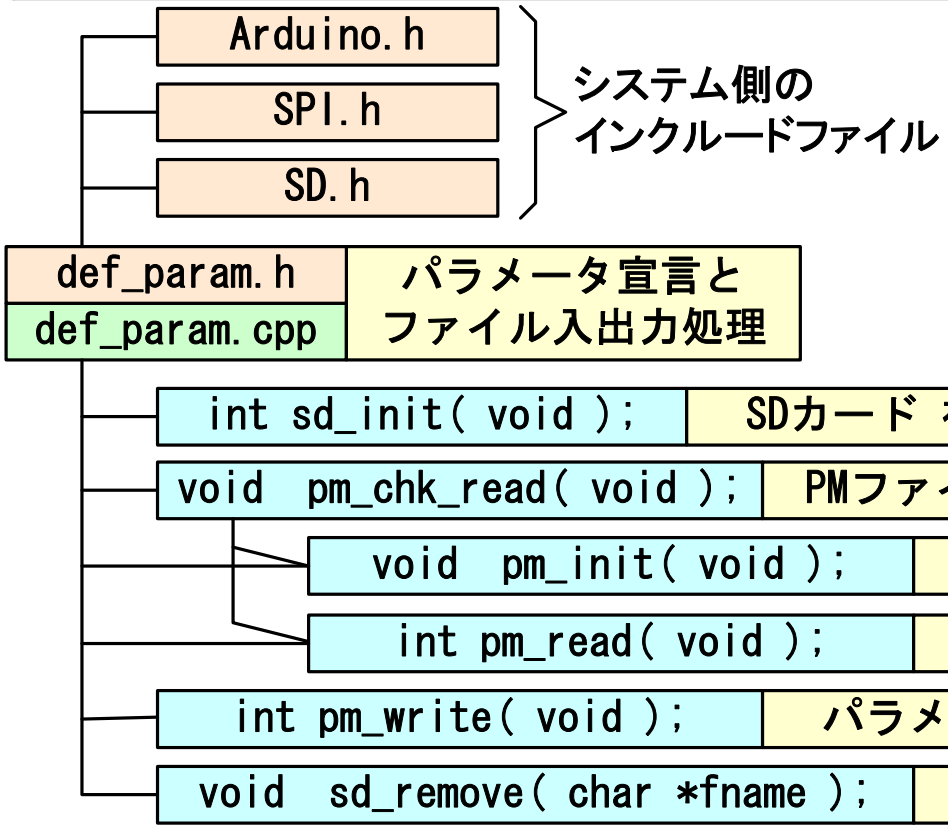
どこから説明を 始めればいいのか最初は悩みますね。 普通は、組み込みシステム的なものであれば、何を行うものであるかの 概要的なところから 説明が始まると 思います。

でも ここしばらくはバラバラに 加速度センサ、気温 湿度 気圧センサ、SDカードアクセスのパート毎の説明をしてきたので、その延長で説明します。 それと今回は 複合的に シリアル通信による、データ転送と、ファイルアクセスを組み合わせました。

PC側のプログラムの説明は 2つ前の動画にパソコン側のプログラム構成として 中央にパラメータ構造体を置き、右に表示 編集、下にパラメータ Writeと Read、左に パラメータの送信、受信機能という事で プログラムの構成要素を概要的に示しました。



パラメータ宣言と ファイル入出力処理

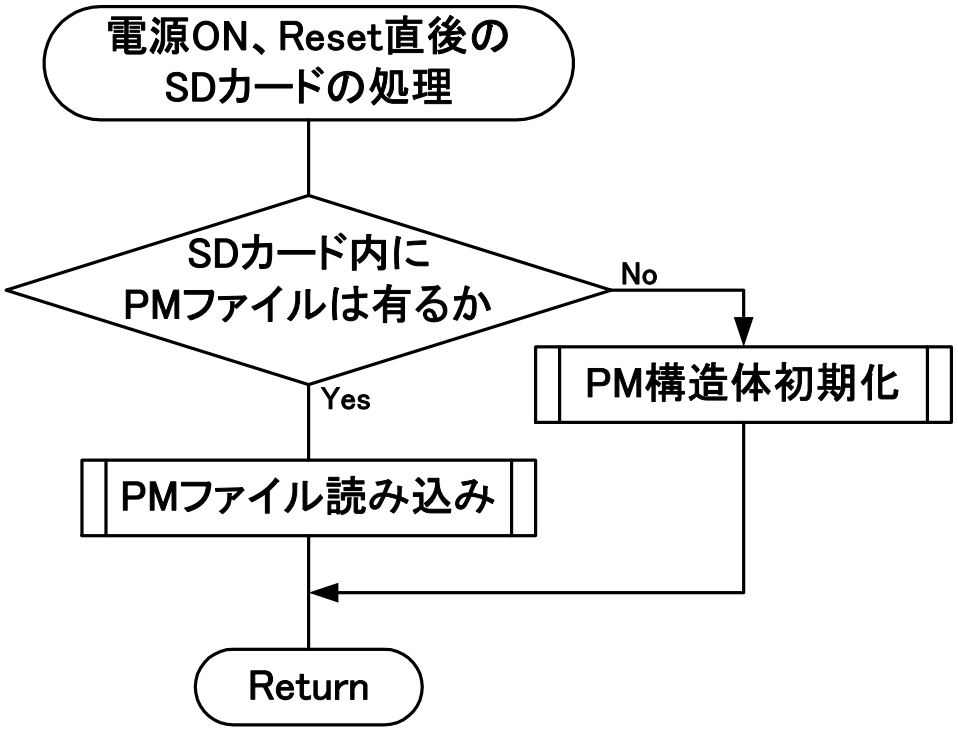


def_param.cpp内に実装している関数です。
パラメータ構造体の型宣言は def_param.h内にあります。 構造体変数の実態宣言は simple_logger.ino 内に あります。 パラメータファイルのファイル名の宣言は def_param.h内 先頭に
#define PM_FNAME "/LogParam.pm" の形で宣言しています。

ファイル名先頭の "/"は 取ると エラーになります。 このような、組み込みシステムでは、ファイルは ルートディレクトリに 置く事にします。

電源ON及び、Reset直後のパラメータに関わる処理シーケンスは　以下のようになります。

以下の処理は　def_param.cpp内の
void　def_pm::pm_chk_read(void); です。
右が　関数ソースです。




```
//**********************************
//**  PMファイル有無確認を行い　**
//**  あれば 読み出し　　　　　**
//**  無ければ PM構造体初期化　**
//**********************************
void  def_pm::pm_chk_read( void )
{
    if( SD.exists( PM_FNAME ))
      pm_read();  // 読み込み
    else
      pm_init();  // 仮初期化
}
```

上記ソースの　SD.exists(ファイル名) 関数は、ファイルが　存在するか確認する関数です。あれば、True、無ければ False です。

パラメータ構造体の SDカード読み書き

```
//**  パラメータファイル読み出し  **
int def_pm::pm_read( void )
{
  File  f;
  int   sts;

  f = SD.open( PM_FNAME, FILE_READ );
  if( f )
  {
    sts = f.read( (uint8_t*)&Hpm, sizeof( Hpm ));
    f.close();
    if( sts != sizeof( Hpm )) return  -2;
  }                    // サイズ不一致で 読み出し失敗
  else
    return  -1;    // オープン失敗

  return  sts;     // 正常終了 ( 読み出しバイト数 )
}
```

pm_read() 関数内にて ファイルをアクセスするための SDオブジェクトの関数を 説明します。 今回は、バイナリデータのブロックを 読み出す事を目的としたコーディングです。 ソース先頭で File f; が あります。 これは 指定されたファイル名、ファイルアクセスモードで SD.open関数でオープンされた ファイルをアクセスするための オブジェクトが f と なります。
sts = f.read((uint8_t*)&Hpm, sizeof(Hpm)); は、パラメータファイルの 読み出しです。第一引数は、パラメータ構造体のアドレスです。 キャストは この通りにやって下さい。 第二引数は 構造体データの サイズ(byte 値)です。
読み終わったら、速やかに f.close(); を行って下さい。

```
//**  パラメータファイル書き込み  **
int def_pm::pm_write( void )
{
  File  f;
  int   sts;

  f = SD.open( PM_FNAME, FILE_WRITE );
  if( f )
  {
    sts = f.write( (uint8_t*)&Hpm, sizeof( Hpm ));
    f.close();
    if( sts != sizeof( Hpm )) return  -2;
  }                    // サイズ不一致で 書き込み失敗
  else
    return  -1;    // オープン失敗

  return  sts;     // 正常終了( 書き込み byte数 )
}
```

pm_write() 関数内にて ファイルをアクセスするための SDオブジェクトの関数を 説明します。 今回は、バイナリデータのブロックを 書き込む事を目的としたコーディングです。 File f; の説明は前ページで行ったので省略します。 SD.open 関数の第二引数の FILE_WRITE が 書込みモード指定です。 読み出しの時は FILE_READです。 sts = f.write((uint8_t*)&Hpm, sizeof(Hpm)); は、パラメータファイルの 書き込みです。第一引数は、パラメータ構造体のアドレスです。 キャストは この通りにやって下さい。 第二引数は 構造体データの サイズ(byte 値)です。
書き終わったら、速やかに f.close(); を行って下さい。 ファイルアクセスは よく使うと思われるので、細かく説明しました。

ESP32マイコンと PC間の 通信処理

Arduino.h　システム側の インクルードファイル

comm_s.h / comm_s.cpp　ESP32マイコンと PC間の通信処理

- void start(int bps); — シリアル通信 使用開始
- void erase_recv_char(void); — 受信済み文字の消去
- check_recv_char(void); — 受信文字 確認
- recv_text_block(byte *buf, byte len); — パラメータファイル読み出し
- send1(byte b); — 1文字 送信
- send_txt(char *txt); — 文字列の送信
- void send_txtln(char *txt); — 文字列+CrLfの送信
- int hexstr_to_bin(char *hex, byte *bin); — Hex -> Bin 変換
- byte hexchar_to_byte(char c); — Hex 1文字を 4bitバイナリ値に変換
- byte byte_H4_hex1(byte b); — byte上位4bitを Hex1文字で出力

comm_s.cpp は def_param.cppに 比べ関数が2倍以上あります。 次のページに続きます。
で、関数が多数あると 何所から見たらいいのか 可読性の悪いところが あります。

しかし、基本となる処理は、パラメータデータの受信と 送信の2つです。

byte byte_L4_hex1(byte b);	byte下位4bitを Hex1文字で出力
int bin_to_hexstr(byte *bin, int blen, char *hex);	バイナリ配列を Hex文字列に変換
byte gen_hp_code(byte *bin, int len);	水平パリティコード生成
byte chk_hp_code(byte *bin, int len);	水平パリティコード照合
byte recv_h_cmd_1(void);	Hexコマンド １行受信
byte param_bultin(word adr, byte len, byte *buf);	パラメータ組立て
void recv_h_cmd_main(void);	Param Hex受信コマンド メイン
byte send_h_cmd_1(word adr, byte len);	Hexコマンド １行送信
send_h_cmd_main(void);	Param Hex送信コマンド メイン
param_extract_block(byte *buf, word adr, byte len);	Bpmから１ブロック取り出し

次に パラメータ受信と、送信の2つの処理のフローをお見せします。

パラメータデータの送受信処理

まず、事前に理解しておいてほしいのは、転送するパラメータ構造体を 先頭から 16byte毎にスライスして 送受信します。 その際にパラメータ構造体に バイナリデータが含まれているため、文字列として転送時 バイナリデータの値が Crコードと紛らわしい値が出てくる事もあります。 よって やや面倒ですが 16byteのバイナリデータを 16進数の Hex文字列データとして送ります。 その前に 16byteのバイナリデータの前に データ長 1byte と 構造体先頭を 0 とする。 byte単位のアドレス 2byteを 付けます。 これで 1+2+16 で 19となります。この19byteに対し 受信後 電文が壊れてないか検証するために 水平パリティコード 1byteを 19byteの電文の終りに 付けます。

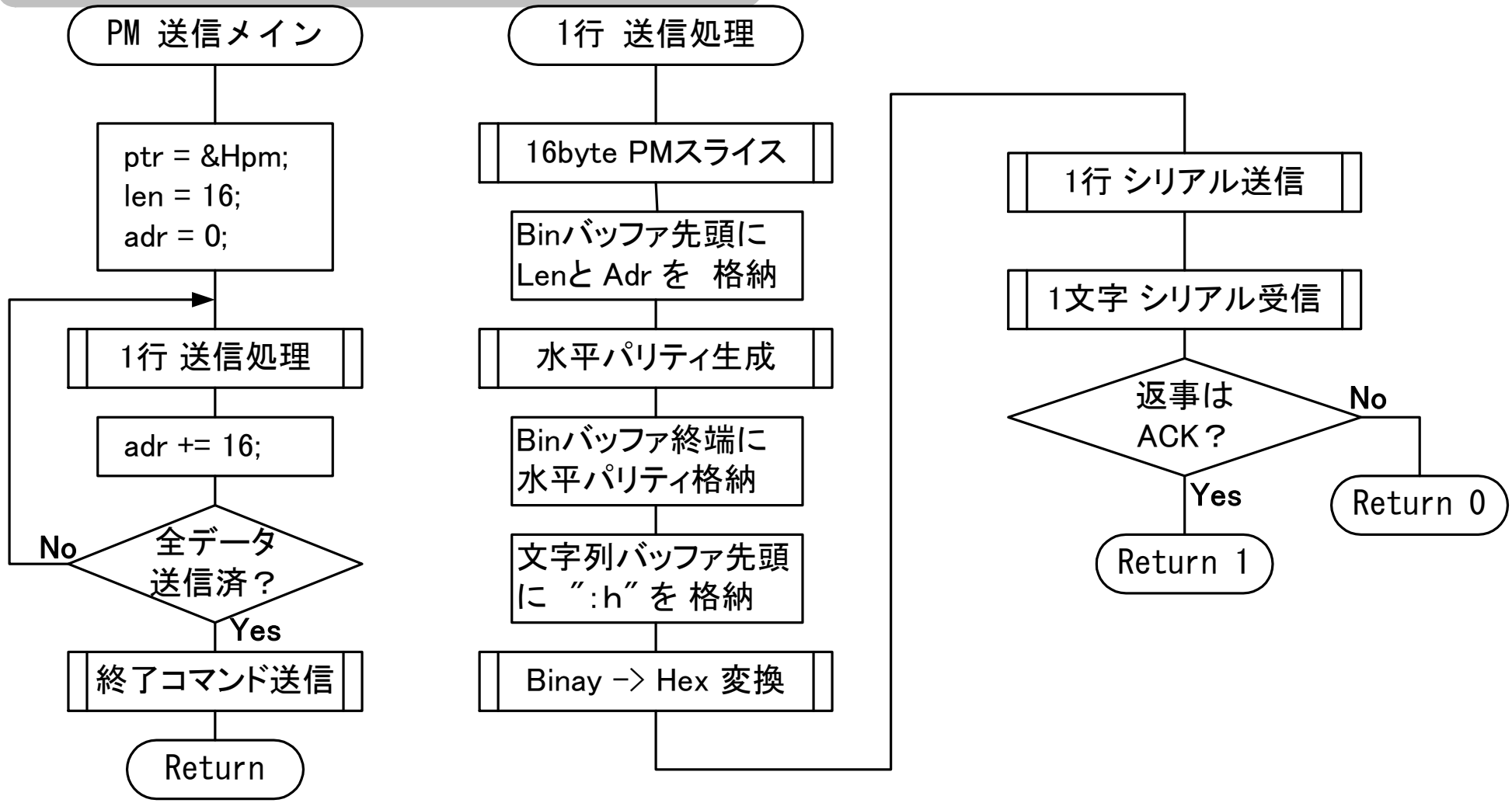
よってバイナリデータは 計 20byte と なります。 この 20byteを Hex文字列 40文字に変換します。 そして、電文の識別を 受信側でやりやすくするため、電文先頭に ″:H″ または ″:h″を付ける事にしました。 そして、文字列として送るので 電文の終りに CrLf コードを付けて送信します。 電文の例を示します。

:H10000067452301CDAB3412546974 6C6520737437

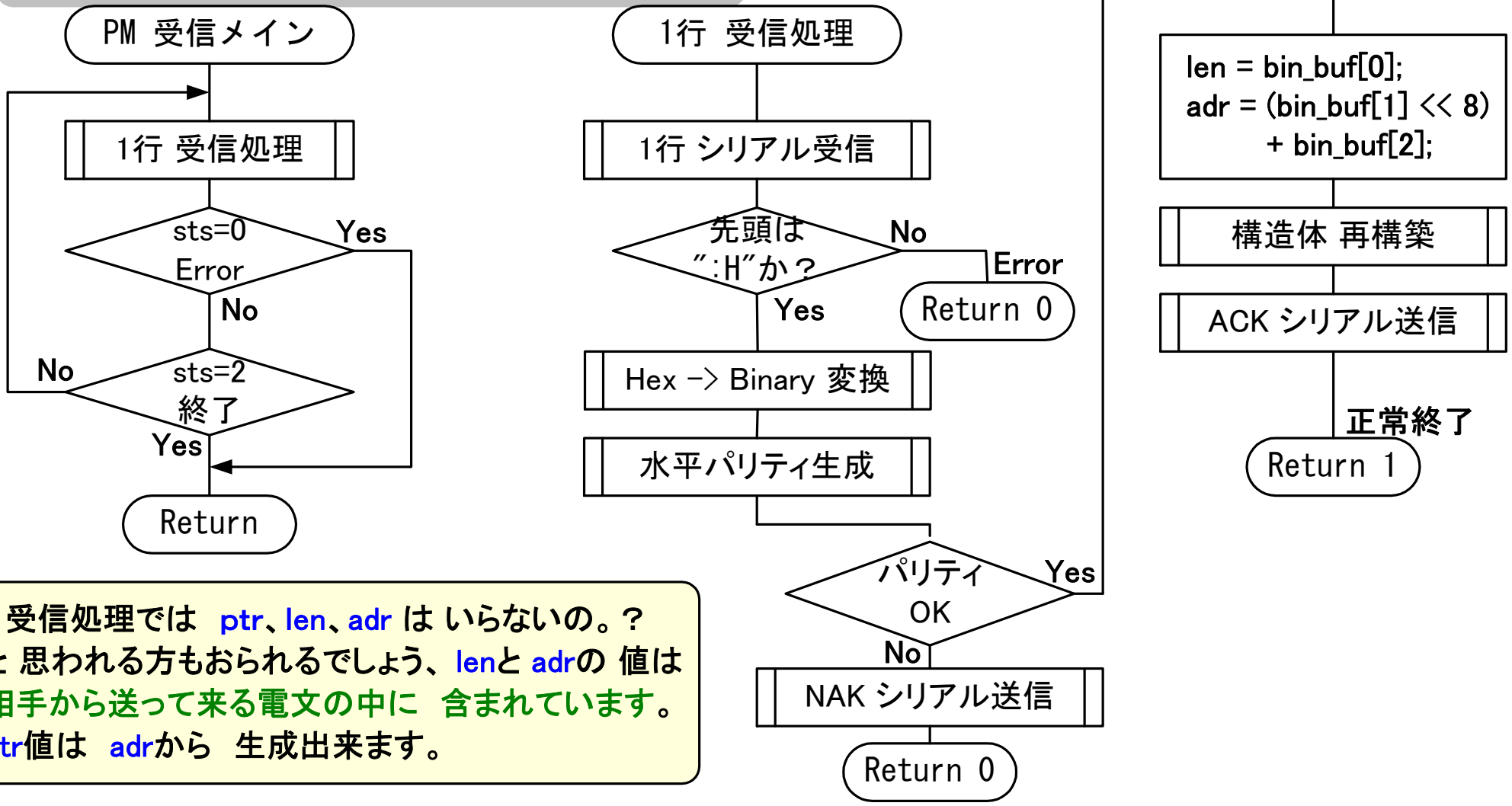
① 青の 10 が 16文字を意味します。
② 赤の 0000 が 構造体先頭からのアドレス値
③ 緑の 67 ～ 74 が 16byteにスライスしたバイナリデータを 32byteの Hex文字列に変換したものです。
④ 最後の 茶色 37 は 水平パリティコードです。これにより 電文が壊れてないか判断します。

今回は、エラー時 再送信制御は 行っていません。 Error表示をして その場で止まります。

パラメータデータの 送信時フロー



パラメータデータの 受信時フロー



受信処理では ptr、len、adr は いらないの。？
と 思われる方もおられるでしょう、lenと adrの 値は
相手から送って来る電文の中に 含まれています。
ptr値は adrから 生成出来ます。

comm_s プログラム　関数の階層表