平成 20 年度

信州大学大学院工学系研究科修士学位論文

# シリアルカメラを用いた
# 監視システムの検討と製作



05TA549B

土田　文武

# 目　次


第１章　序論 1
　１－１　背景 1
　１－２　関連する製品 1
　１－３　本研究のねらい 1
第２章　使用部材 2
第３章　開発環境 3
　３－１　クロス開発環境の構築 3
　３－２　MS104-SH4 での Linux の起動確認 4
第４章　構成検討 5
　４－１　装備する機能とプログラム 5
　４－２　システム構成の検討 5
　４－３　構成の概略図 11
　４－４　回路図 11
第５章　製作 13
　５－１　基板製作 13
　５－２　コード実装（PIC16F877 モジュール） 14
第６章　MS104-SH4 の設定 22
　６－１　シリアルポートの設定 22
　６－２　シリアルポート受信プログラム（MS104-SH4） 24
　６－３　画像データのビットマップ変換と確認 27
　６－４　ビットずれ対策の検討 31
　６－５　画像取得の自動化 40
第７章　Web サーバでの表示するための設定 41
　７－１　設定 41
　７－２　Web ブラウザ，携帯電話での表示の注意点 41
第８章　結論 42
第９章　応用アプリケーション例 43
第１０章　参考にさせていただいたサイト、お世話になったサイト 43

# 図　目　次

図 1：MS104-SH4 ボード..... 2
図 2：シリアルカメラ ..... 2
図 3：クロス開発環境と実行環境 ..... 3
図 4：Linux ボードの２つの動作環境..... 4
図 5：SH4 と COM ポートの接続回路..... 6
図 6：AKI-PIC877 モジュール（秋月電子）..... 7
図 7：調歩同期方式のデータシーケンス ..... 8
図 8：クロック同期方式のデータシーケンス ..... 8
図 9：データバイト間で RS-232C 通信を行う場合 ..... 9
図 10：データ受信の空き時間に送信 ..... 10
図 11：システム構成図 ..... 11
図 12：クロック同期方式→調歩同期方式変換基板回路図 ..... 12
図 13：基板実装 ..... 13
図 14：監視システム全体 ..... 13
図 15：クロック同期方式→調歩同期方式変換基板回路図（確定版） ..... 20
図 16：発振器の実装図と周辺の回路図 ..... 21
図 17：RS-232C ドライバ IC の換装..... 25
図 18：YUV422 のデータ形式..... 27
図 19：シリアルカメラで撮影された画像第 1 号 ..... 30
図 20：携帯電話での確認画面 ..... 42

# 表　目　次

表 1：調歩同期方式とクロック同期方式の違い ..... 7
表 2：ビットマップファイルの構造 ..... 27

# プログラム　目次

リスト 1：クロック同期 → 調歩同期変換プログラム ..... 14
リスト 2：RS-232C 受信テストプログラム..... 24
リスト 3：YUV422 形式→ビットマップ形式変換プログラム yuv2bmp.c..... 28
リスト 4：クロック同期 → 調歩同期変換プログラム（ビットずれ対策版） .. 31
リスト 5：RS-232C 受信プログラム（ビットずれ対策版）getbmp.c..... 36

# 第１章　序論

## １－１　背景

近年、セキュリティに関する意識が高まりつつある。画像による監視は、セキュリティ上非常に有効な手段である。また、インターネットの普及に伴い、携帯電話などの情報端末により簡単にホームページの閲覧が可能となっている。

今回、シリアルカメラを監視カメラとして用い、ホームページで撮影画像を公開することにより、遠隔地でも携帯端末により監視画像を確認できるシステムを構築することとした。監視対象は、自分の部屋や近所のごみ集積所に対する放火監視等を想定している。

## １－２　関連する製品

インターネットを介した画像監視システムとしては､WEB カメラとして非常に多くの製品が発売されている。しかし、これら多くの製品は画像を表示するだけの機能しか持たない。今回製作する監視システムはこれらの製品と同じ機能を持つが、OS を搭載し、個人で応用アプリケーションを簡単に追加できるようなものを目標としている。

## １－３　本研究のねらい

シリアルカメラからの画像データの受信処理、画像データ変換・公開を他にホストコンピュータを必要としないワンボードのコンピュータで構築することを通して、情報処理に対する理解と実務に応用できる技術力を深めることを目標とする。

ホームページで同じカメラを用いて画像を表示する実験を公開しているサイトもあるが、FPGA で読み出して一度 RAM に転送しそれを改めて読む方式であり、仕組みが大掛かりである。今回は、簡易的な PIC を用いた回路で高速シリアル通信によってデータ受信が出来るかの挑戦でもある。

## 第２章　使用部材

・ワンボードコンピュータ

アルファプロジェクト製　MS104-SH4

CPU：SH7750R（SH-4）　クロック：235.9296MHz

FlashROM：16MB　SDRAM：32MB

シリアルポート×２　I/O ポート×８ピン

PC/104 規格バス有　OS：Linux

CF カードインターフェース（TypeI）

10/100BASE-TX×１　5V 単一電源動作

図 1：MS104-SH4 ボード

選定理由

・Linux が動作すること

・組み込みシステムで使われるような CPU であること

最近では、x86 ベース CPU のボードも増えてきているが、自分として SH4 に挑戦してみたかったため

・今後の拡張性を考え、性能的に余裕のあること

・シリアルカメラ

東芝製　デジタルカメラユニット　DMR-C1

デジタルメモリレコーダ用（現在入手困難）

10 万画素(352×288)CMOS イメージセンサー

撮影距離範囲：約 30cm ～ ∞

インターフェース：クロック同期シリアル

図 2：シリアルカメラ

選定理由

・カメラからのデータが RAW データであること

・データの取り込みが比較的簡単（シリアル）であること

# 第３章　開発環境

## ３－１　クロス開発環境の構築

組み込み系のプログラム開発は、組み込み機器のメモリ容量が限られていることなどから、通常デスクトップ PC 上でその組み込み機器で動作するバイナリプログラムを作成し、それを組み込み機器に実装されているフラッシュメモリまたはコンパクトフラッシュメモリに転送して実行させる。

そのため、通常の PC にターゲットとなる CPU のバイナリを作成できるクロスコンパイラをインストールする必要がある。環境構築にあたって必要となるものは

・gcc（GNU Compiler Collection：コンパイラ，プリプロセッサ）
・binutils（GNU バイナリユーティリティ：アセンブラ，リンカ）
・libc（C 標準ライブラリ）

これらは、Windows の Cygwin 上でも可能だが、Linux が動作する PC 上で構築するほうが問題が少ない。

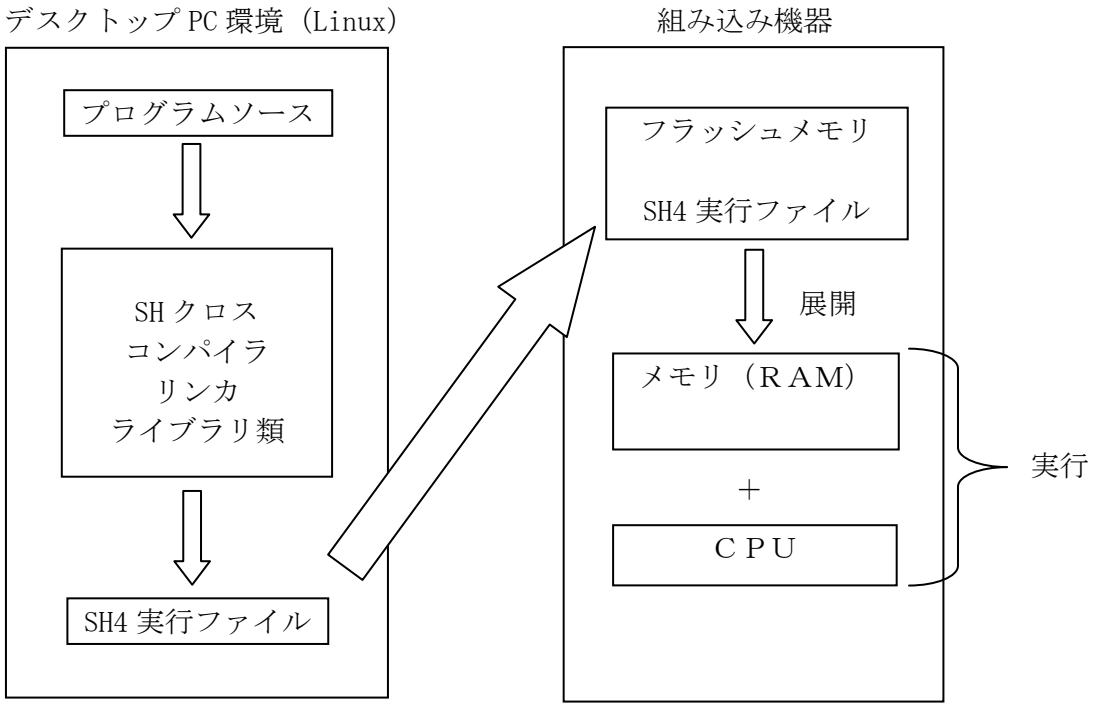


図 3：クロス開発環境と実行環境

Linux をインストールしてあるデスクトップ PC を所有しているため、そこにクロス開発環境を構築することにした。MS104-SH4 には別途開発ツールが存在し、クロス開発環境がインストールスクリプトで簡単に構築できるためそれを利用する。

他の方法として、仮想コンピュータを Windows 環境で動作させ、そこに Linux をインストールして開発環境を構築する方法もあるが、ここでは紹介に留めておく。

## ３－２　MS104-SH4 での Linux の起動確認

MS104-SH4 は購入時にオンボードフラッシュロムに Linux が書き込まれている。この Linux を起動させるためには、シリアルポートを PC に接続し、電源 ON にすると、起動コンソールのメッセージがハイパーターミナルや TeraTerm といったソフトによって確認できる。

購入時から入っている Linux は、かなり機能が限定された非常に小さな構成となっており、もちろんコンパイラなどは入っていない。よって、プログラムを作成するためには、前節で構築したクロス開発環境が必要となる。

CF カードスロットも搭載しているため、この CF カード上に Linux をインストールして、CF カードからのブートにも対応している。

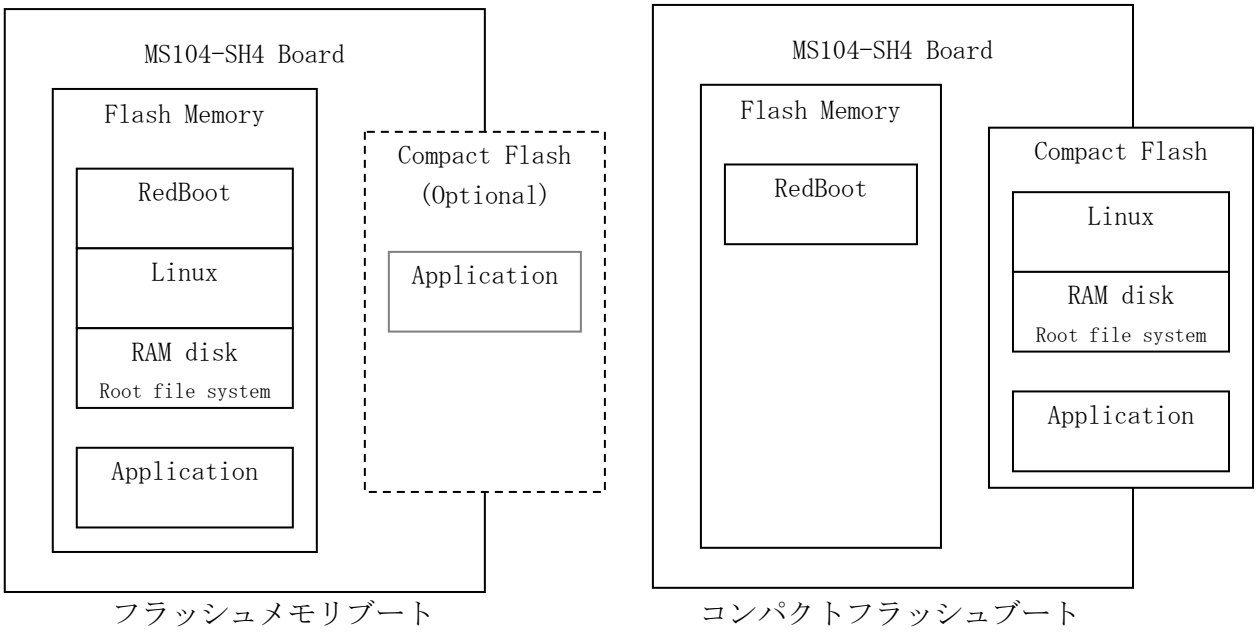


図 4：Linux ボードの２つの動作環境

OS を起動するためには、まずブートローダーという OS をメモリ上に展開するプログラムが必要となる。このブートローダーが、OS のカーネルとファイルシステムをメモリ上に展開し、OS を起動する役目をもつ。CPU に SH4 を使ったものについては、ブートローダーの有名なものとして RedBoot と IPL+g の２つが存在する。

RedBoot は Red Hat 社が開発し、配布している μITRON Ver.3 系の OS である eCos を流用したもので、SH 系の CPU の他、ARM、68K、PPC、MIPS、H8 にも対応している。
IPL+g は SH 系の CPU にしか対応していないブートローダーであるが、機能的には RedBoot とほとんど変わらない。

今回使用する MS104-SH4 のフラッシュロムには、最初から RedBoot が書き込まれているので、それを利用する。フラッシュロムから起動するか、コンパクトフラッシュから起動するかの指示は、RedBoot に対するコマンドで行う。

# 第４章　構成検討

## ４－１　装備する機能とプログラム

① シリアルカメラからの画像データを MS104-SH4 ボードで受信する。
② 受信した画像データ（YUV422 形式）をビットマップ画像形式に変換する。
③ インターネットを通して http プロトコルでアクセスしてきた場合監視画像を見ることができる。

上記機能を実現するために必要な MS104-SH4 ボード上のプログラム
① データ受信プログラム
② 画像データ変換プログラム
③ WEB サーバプログラム

①と②に関しては汎用のものが存在しないため、クロスコンパイル環境で新規に作成する。③の機能を実現するために、boa（小型 WEB サーバプログラム）を使用する。boa はボードメーカーの配布 OS に組み込まれているため、新たにコンパイルして組み込む必要はなく、テキストファイルで画像を表示するための HTML ファイルを記述するだけでよい。

## ４－２　システム構成の検討

シリアルカメラと MS104-SH4 ボードの接続インターフェースが最大の難関である。シリアルカメラはクロック同期方式でデータを送信するが、MS104-SH4 のシリアルポートは調歩同期方式（非同期方式）となっている。MS104-SH4 からはクロックを出力する機能は存在しないため、工夫が必要となる。３つの方法を考案し検討を行った。

① SH4 は CPU の機能として調歩同期方式とクロック同期方式の両方が使える。CPU とシリアルポートコネクタを接続する信号線は途中 CPLD を通過しているため、結線を変更してフロー制御の信号線をクロックラインとして利用する。
（回路図による確認が必要）
② シリアルポートには電源ラインが存在しないため、電源線のある I/O ポートを利用し、プログラムでクロック信号を作成し、読み出しを行う。
③ 調歩同期方式のシリアルポートでデータを受信するため、シリアルカメラと MS104-SH4 ボードの間にマイコンを設置し、シリアルカメラ出力のクロック同期方式を調歩同期方式に変換して出力し、MS104-SH4 ボードでデータを受信する。

各方法についての検討結果

① SH4 は CPU の機能として SCI と SCIF という２つのシリアルポートを持ち、SCI はクロック同期方式と調歩同期方式の２つのモードを選択可能、SCIF は調歩同期方式のみでともに全二重通信が可能となっている。

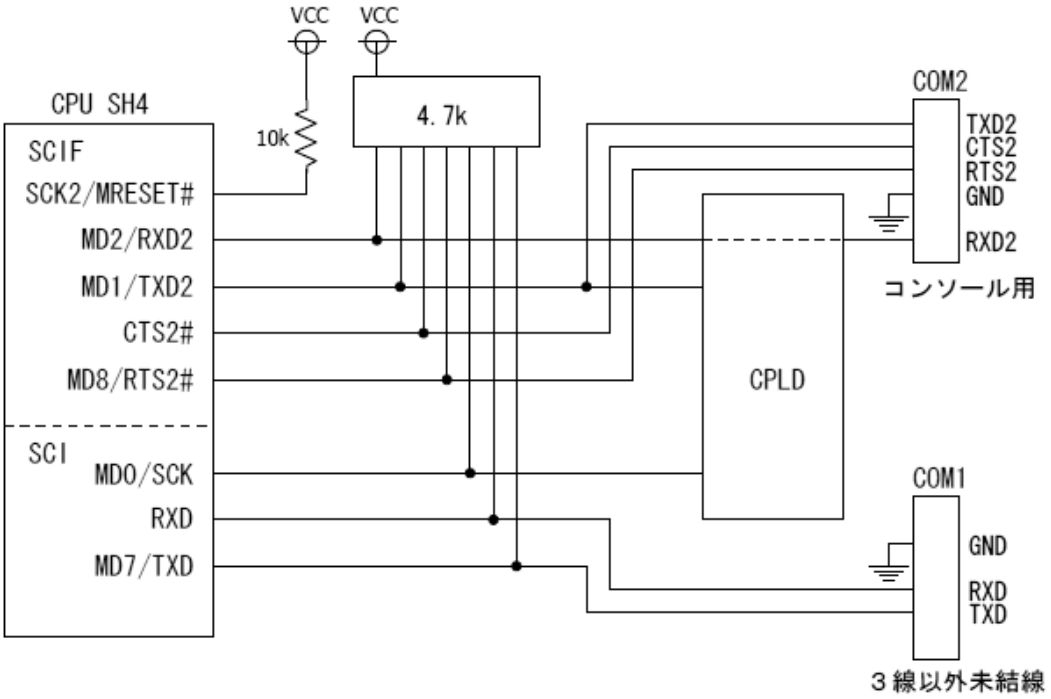


図 5：SH4 と COM ポートの接続回路

クロック同期方式に使用できるのは COM1 に接続された SCI インターフェースであるが、回路図を見ると、クロックを出力する SCK 信号は CPLD に入力されているが CPLD と COM1 端子の接続がされていない。そのため、COM1 端子でクロック同期方式の通信を行うためには、SCK の信号線と COM1 の空きピンを空中配線する必要がある。また、電源線が存在しないため、電源も別途空中配線しなければならない。

OS に関しても、標準では調歩同期方式の通信しかサポートしていないため、シリアルポートを扱うデバイスドライバに変更を行う必要がある。

以上のことから、COM ポートによりクロック同期方式で直接通信を行う方法は難易度が高く、避けたほうがよい。

② I/O ポートは、CPU に直接つながっている。使用する場合は、SH4 の I/O ポート A を直接操作する必要がある。Linux のユーザー空間から I/O ポートにアクセスするためにはデバイスドライバが必要になるため、この場合にもデバイスドライバの作成が必須条件となる。I/O ポート操作のサンプルプログラムがあるのでそれを改変して利用する方法もあるが、OS 上でプログラムによりポート操作を行う場合、タイミングは 10ms 程度の周期が限界であり、今回の高速読み出しには対応できないと判断した。

③ クロック同期方式と調歩同期方式を変換するため、PIC マイコンを使用することにする。この場合、読み出しクロックは PIC から内蔵タイマーを使って供給して１ビットずつ読むことになる。秋葉原の秋月電子通商で販売されている AKI-PIC877 モジュールを使用すれば、RS-232C のドライバも付いているので COM ポートへの接続も簡単である。この方式で行くことを前提に検討を進める。

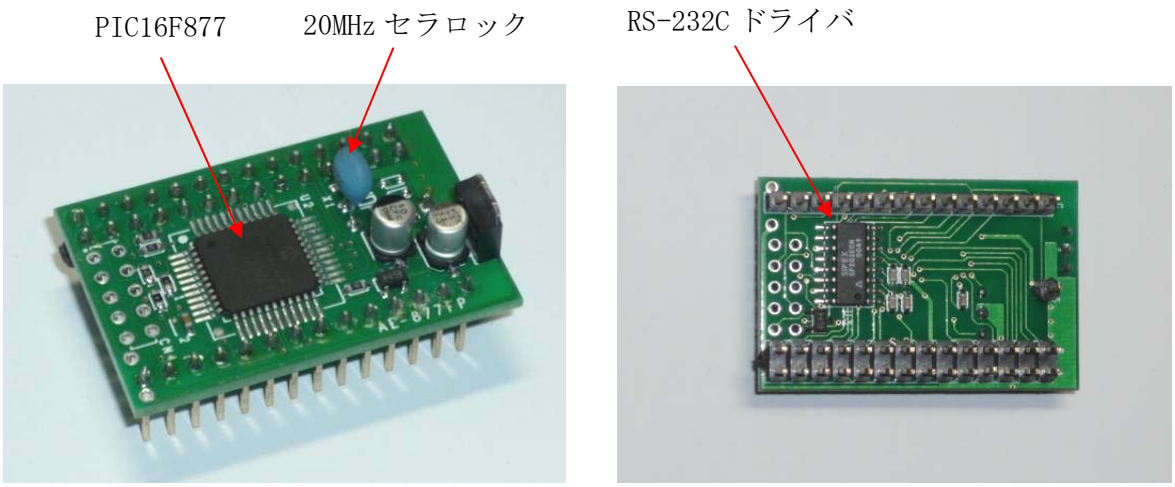


図 6：AKI-PIC877 モジュール（秋月電子）

以後、AKI-PIC877 モジュールは PIC16F877 モジュールと記述する。

表 1：調歩同期方式とクロック同期方式の違い

| | データ速度 | データ長 | ブロック同期 |
|---|---|---|---|
| 調歩同期方式 | ～921.6kbps | 7 or 8 ビット | スタートビット |
| ｸﾛｯｸ同期方式 | クロック依存 | 8 ビット固定 | 特定信号列による |

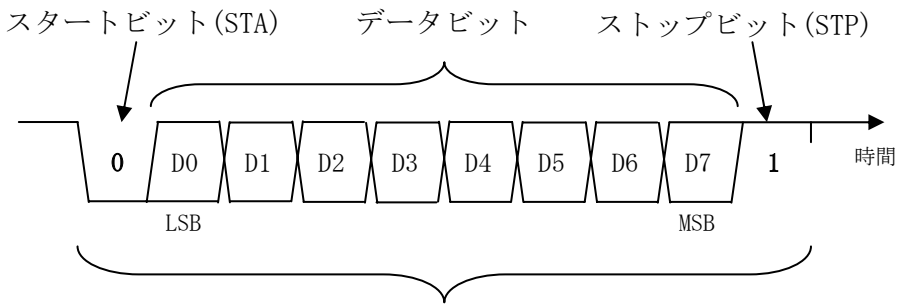


図 7：調歩同期方式のデータシーケンス
（スタート/ストップビット各１、パリティなしの場合）

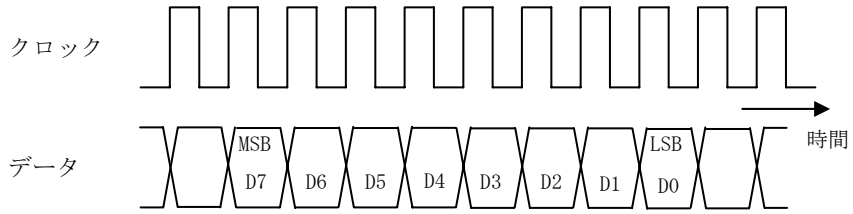


図 8：クロック同期方式のデータシーケンス
（クロック立ち上がりでデータラッチする場合）

調歩同期方式では、図７のように、スタートビットとストップビットを加え、１バイトの送信に 10 ビットの信号変化が必要となる。クロック同期方式では、調歩同期方式のようにデータ以外の付加的なビットが無いため、伝送効率は良いが、どこがバイトの先頭か判断できない。これを解決するために、特徴のある信号ブロックを受信することによってバイトデータの先頭を判断する方法が用いられる。合致した時に次からの８ビット毎が各バイトデータと判断できる。

電源については、空きピンに配線するか、モジュールの電源端子に直接配線すればよい。ただし、20MHz クロックで動作させるためには、5V の供給が必要となる。この 5V は PC104 バスコネクタから別配線で供給する。

PIC16F877 は PICNIC にも使われており、CAI で使用している。パッケージは違うが初めて使うマイコンではないので開発環境もすでに整っている。その点では問題ない。A/D コンバータに温度センサーを接続することにより、急激な温度変化も検出できるといった拡張性もあるので有利である。

続いて、クロック同期方式の読み出し速度と調歩同期方式の送信速度について検討する。使用するシリアルカメラはデータシートが存在せず、有志によりインターネット上で公開されているデータだけが頼りとなっている。幸い必要なデータ速度とフォーマットは公開されているため、それを使って検討する。

○読み出しデータクロック　　182kHz 以上（44 マイクロ秒/byte 以下）
○データフォーマット
・ヘッダスタートマーカ　　AA 55（2 バイト）
　　ヘッダ情報　28 バイト
・データスタートマーカ　　**55 AA**（2 バイト）
　　画像データ　202752 バイト（＝352×288×2）
・データエンドマーカ　　**FF D9**（2 バイト）

データスタートマーカとデータエンドマーカに挟まれた部分が画像データであり、YUV422 方式で 352×288×2(=202,752)バイトで構成されている。そのほかの情報部分としてヘッダスタートマーカとデータスタートマーカの前に 28 バイトあるが、この部分は共通なので受信する必要はない。
画像データの 202,752 バイトは PIC 内に保存できないため、データスタートマーカを検出した後の 8 ビットずつを順次出力しなければならない。

読み出し速度を 250kbps(32 マイクロ秒／byte)とすると、クロックのデューティー比を 50％として 500kHz でクロック信号を変化させなければならない。250kbps で読み出すと、8 ビット読み出しに 28 マイクロ秒、次のビットまで 4 マイクロ秒なので、この 4 マイクロ秒の間に RS-232C 経由で送信すると、スタートビット 1、ストップビット 1 を付けて 10 ビットなので 2.5Mbps で送信しなければならない。

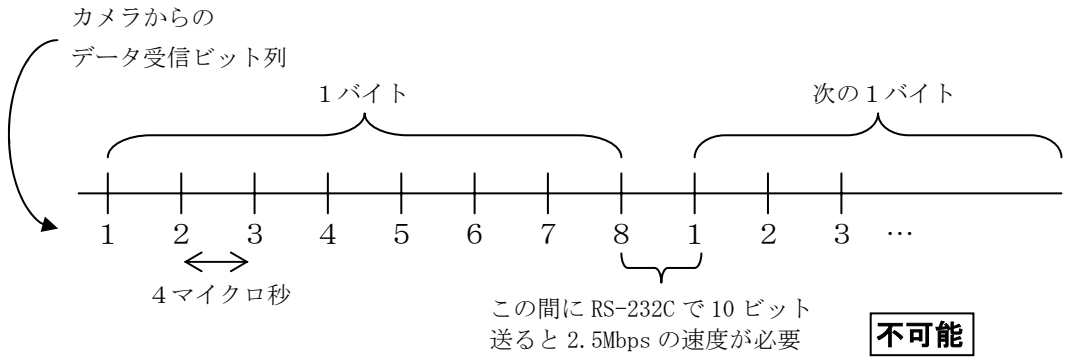


図 9：データバイト間で RS-232C 通信を行う場合

これは不可能であるので、シリアルカメラからのデータ受信を行っている空き時間に送信する必要がある。

データを受信している空き時間に送信する場合は、図 8 で示すようなスケジュールとなる。受信・送信ともに縦の線はタイマーによる割り込みで動作させる。

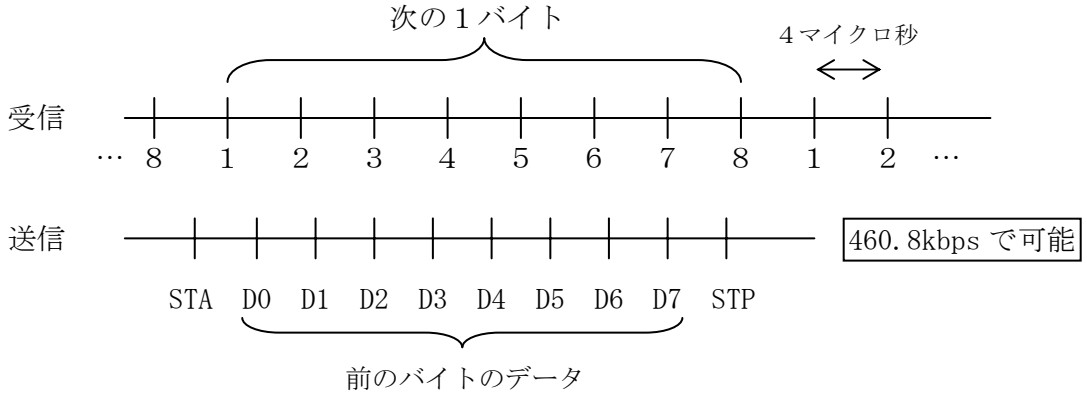


図 10：データ受信の空き時間に送信

受信速度は 250kbps を仮定しているため､20MHz の動作クロックでは各ビットの受信間隔に 80 クロック存在する。受信時に行う動作としては

1. 読み出しクロック生成のためのポートのビット反転動作
2. ポートの読み込み
3. ビット比較
4. ビット列格納

送信時の動作としては

1. ビット列読み出し
2. ポートにデータをセット

が考えられるが、実際にプログラムを書いてみないと正確にはわからないが、80 クロックで可能と判断する。

以上の検討より、方法として③のクロック同期方式→調歩同期方式の変換によるデータ受信方法で可能と判断する。

## ４－３　構成の概略図

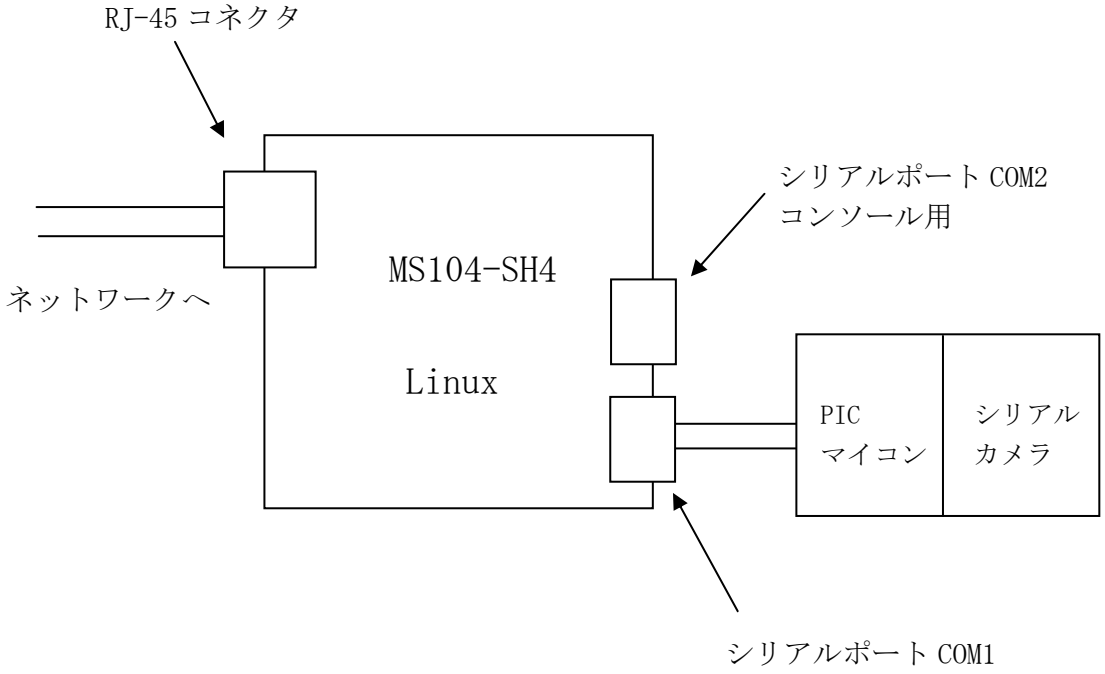


図 11：システム構成図

## ４－４　回路図

シリアルカメラと途中に PIC マイコンを入れることを考え、次のページに示すような回路を考えた。

シリアルカメラの動作電源が+3.0V であることから、レベル変換のためのオープンドレインタイプのバッファを２個（U2, U3）使い、それぞれ入力側の H レベルの電圧で動作させ、出力側をプルアップしてレベル変換をしている。+3.0V の電源電圧は、LDO（U4）により作成している。

U2 と U3 は各 IC とも１回路分しか使用していないので、無駄に思われるが、ポート入力電圧が電源電圧を超えてはいけないという制約により 3V 動作と 5V 動作の IC を使っている。

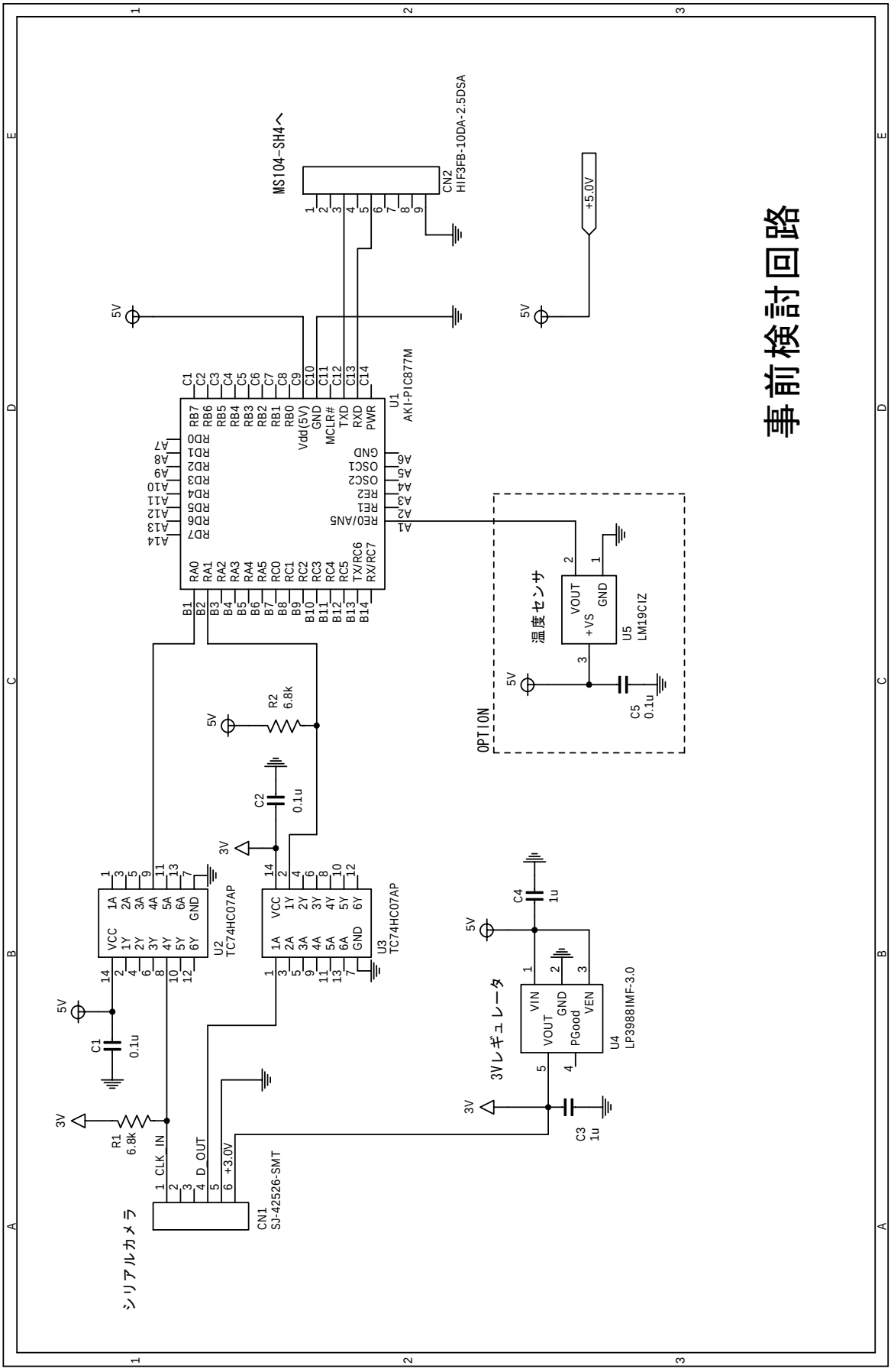


図 12：クロック同期方式→調歩同期方式変換基板回路図

# 第５章　製作

## ５－１　基板製作

回路図に従い、基板に回路を実装した。

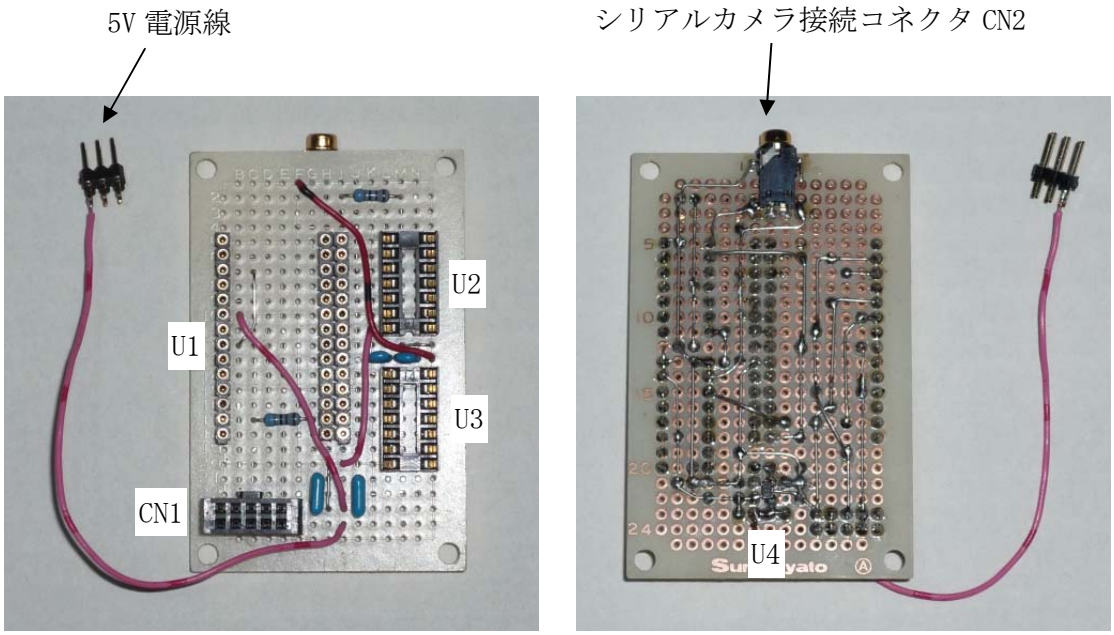


図 13：基板実装

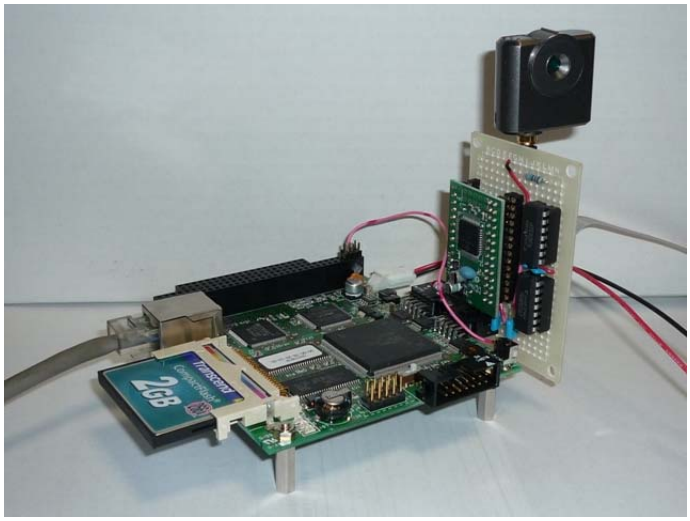

図 14：監視システム全体

## ５－２　コード実装（PIC16F877 モジュール）

RS-232C 経由での通信にビットレートのずれが起こらないようにコーディングした。命令に対する説明は、適宜コメントにて記載してあるのでそちらを参照されたい。プログラムリストを示す。

リスト 1：クロック同期 → 調歩同期変換プログラム

```
 1: ; クロック同期　－> 調歩同期変換プログラム
 2:
 3:     LIST            P=PIC16F877
 4:     INCLUDE <P16F877.INC>
 5:
 6: ;-----------------------------------------------------------------------------
 7:
 8:     __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC & _LVP_OFF
 9:
10: ;-----------------------------------------------------------------------------
11:
12: RXB0        EQU             20H             ; 送信用バッファ
13: RXB1        EQU             21H             ; 受信用バッファ
14: SCIF        EQU             23H             ; クロック同期受信ステータス保存用
15: BITC        EQU             24H             ; ビットカウンタ
16: COUNT       EQU             25H             ; ヘッダ送り用
17:
18: ;-----------------------------------------------------------------------------
19:
20:     org     0
21:     goto    START
22:
23:     org     8
24:
25: START:
26: ; 各種ポートの設定
27:     ; 入力ピンが浮いていると誤動作の原因となるため必要なもの以外は出力に変更
28:     ; A/D の設定      A port:アナログ　E port:デジタル
29:     movlw   B'00000010'
30:     bsf     STATUS,RP0      ; [Bank1]
31:     movwf   ADCON1
32:     bcf     STATUS,RP0      ; [Bank0]
33:     ; A port 温度センサー接続時 RA0 をアナログ入力ピンで使用するがそれ以外出力固定
34:     clrf    PORTA
35:     movlw   B'00000000'
36:     bsf     STATUS,RP0      ; [Bank1]
37:     movwf   TRISA
38:     bcf     STATUS,RP0      ; [Bank0]
39:     ; E port 汎用 I/O ポート, RE1:カメラクロック出力, RE0:カメラデータ入力,
40:     ; その他:出力固定
41:     clrf    PORTE
42:     movlw   B'00000001'
43:     bsf     STATUS,RP0      ; [Bank1]
44:     movwf   TRISE
45:     bcf     STATUS,RP0      ; [Bank0]
46:     ; B port 使用しないので出力固定
47:     clrf    PORTB
48:     bsf     STATUS,RP0      ; [Bank1]
49:     clrf    TRISB
50:     bcf     STATUS,RP0      ; [Bank0]
```

リスト１：クロック同期 → 調歩同期変換プログラム（続き）

```
51:     ; C port  RC6,RC7 は TX,RX として使用するときはビットセットしておく
52:     clrf      PORTC
53:     movlw     B'11000000'
54:     bsf       STATUS,RP0          ; [Bank1]
55:     movwf     TRISC
56:     bcf       STATUS,RP0          ; [Bank0]
57:     ; D port  使用しないので出力固定
58:     clrf      PORTD
59:     bsf       STATUS,RP0          ; [Bank1]
60:     clrf      TRISD
61:     bcf       STATUS,RP0          ; [Bank0]
62:
63:     movlw     B'00000000'         ; 割込み禁止
64:     movwf     INTCON              ; 多少早くなるらしい
65:
66:     clrf      SCIF                ; クロック同期受信ステータスクリア
67:     clrf      RXB0                ; 受信バッファクリア
68:     clrf      RXB1
69:     clrf      BITC                ; 受信ビットカウンタクリア
70:
71: ;--------------------------------------------------------------------------
72:
73:     ;goto     START1              ; 最初に RS-232C 経由でのスタート信号を受け取らない場合
74:                                   ; 受け取る場合はコメントにする
75: ;--------------------------------------------------------------------------
76: ; RC-232C 経由で、クロック同期と調歩同期の変換スタート信号を待つ(460.8kbps)ための設定
77: RS232C:
78:     bsf       STATUS,RP0          ; [Bank1]
79:     movlw     B'00000001'
80:     movwf     SPBRG                       ; 460.8kbps のために 1 を代入(SPBRG=99h)
81:     bsf       TXSTA,BRGH                  ; 高速モード(TXSTA=98h)
82: ;   bcf       TXSTA,SYNC                  ; 非同期モード（デフォルト）
83:     bcf       STATUS,RP0          ; [Bank0]
84:     bsf       RCSTA,SPEN                  ; シリアルポート有効(RCSTA=18h)
85:
86: RXLOOP:
87: ; PIR1.RCIF を監視し、フラグがセットされた場合、RCREG(=1Ah)を確認し、スタート信号「ST」
88: ; だった場合にこのループを抜け次の処理へ。それ以外はそのままループ
89:     bsf       RCSTA,CREN          ; 受信を有効
90:                                   ; 受信すると PIR1.RCIF(PIR1=0Ch)がセットされる
91:     movf      RCSTA,W
92:     andlw     B'00000110'         ; フレームエラー、オーバーランエラーの検出
93:                                   ; （発生したら一度受信を無効にする）
94:     btfsc     STATUS,Z            ; エラーが発生していなければ Z=0
95:     bcf       RCSTA,CREN
96:     btfss     PIR1,RCIF
97:     goto      RXLOOP
98:
99: ; RCREG の確認
100:    movf      RCREG,W                     ; 受信データを W レジスタにロード
101:    xorlw     H'53'                       ; 文字「S」と比較　一致すれば Z=0
102:    btfsc     STATUS,Z
103:    goto      RXLOOP                      ; 一致しない場合再び受信
104:    movf      RCREG,W                     ; 受信データ２文字目を W レジスタにロード
105:    xorlw     H'54'                       ; 文字「T」と比較　一致すれば Z=0
106:    btfsc     STATUS,Z
107:    goto      RXLOOP                      ; 一致しない場合再び受信
108:
109: ;--------------------------------------------------------------------------
110:
111: START1:
112: ; 「ST」が受信できた場合、RS-232C での受信を停止 RC6,RC7 を I/O ポートに変更
113:    bcf       RCSTA,CREN                  ; 受信停止
114:    bcf       RCSTA,SPEN                  ; シリアルポート無効(RCSTA=18h)
115:    bsf       STATUS,RP0          ; [Bank1]
```

リスト１：クロック同期 → 調歩同期変換プログラム（続き）

```
116:      bcf       TRISC,6                         ; RC6 ポートを出力へ設定
117:      bcf       STATUS,RP0          ; [Bank0]
118:      bsf       PORTC,6                         ; RC-232C の信号線は無信号時は H レベル
119:
120: ; シリアルカメラのクロック同期の受信開始（RE1：クロック出力，RE0：データ入力）
121: ; クロック周期は約 4 マイクロ秒（正確でなくて良い）　一方、SH4 ボードへの
122: ; RS-232C（ポート ON/OFF による）出力は 460.8kbps を維持しなければならない
123:
124: ; 460.8kbps では
125: ; RE0 ポート H -> RE0 ポート L，RE1 ポート読み込み -> RE0 ポート H ・・・
126: ; ２マイクロ秒ごとに RE1 状態遷移　その間に置ける命令数は７（2.17 マイクロ秒）
127: ; クロックが H->L->(H)が 16 命令ならば細かいことは気にしない
128: ; 最短でクロックを変化させると周期が２命令ステップ（1.8MHz）
129: ; この半分くらいの速度ならカメラから読み出せるらしい
130:
131: ; 最初のデータ受信では受信済みのデータが無いので RS-232C 送信はなし
132: ; 単にカメラからのクロック同期受信のみ
133:
134:; マーカ読み出し周期 18 命令ステップ
135:
136:      ;goto     CLKIN_N
137:
138: CLKIN1:                  ; ヘッダスタートマーカ１文字目判定
139:      bsf       PORTE,1                                       ; CLK_H
140:      rlf       RXB1,F              ; 受信バッファ左シフト      ;1
141:      NOP                                                     ;2
142:      NOP                                                     ;3
143:      NOP                                                     ;4
144:      NOP                                                     ;5
145:      NOP                           ; 時間調整                ;6
146:      bcf       PORTE,1                                       ; CLK_L
147:      movf      PORTE,W             ; RE->W                   ;1
148:      addwf     RXB1,F                                        ;2
149:      NOP                                                     ;3
150:      NOP                                                     ;4
151:      movf      RXB1,W                                        ;5
152:      xorlw     H'AA'                                         ;6
153:      btfss     STATUS,Z                                      ;7
154:      goto      CLKIN1              ; not AA                  ;8,9 goto:2clock
155:      clrf      RXB1                                          ;8,9 skip:2clock
156:
157: CLKIN2:                  ; ヘッダスタートマーカ２文字目判定
158:      bsf       PORTE,1                                       ; CLK_H
159:      rlf       RXB1,F              ; 受信バッファ左シフト      ;1
160:      decf      BITC,F                                        ;2
161:      NOP                                                     ;3
162:      NOP                           ; 時間調整                ;4
163:      bcf       PORTE,1                                       ; CLK_L
164:      movf      PORTE,W             ; RE->W                   ;1
165:      addwf     RXB1,F                                        ;2
166:      movf      BITC,W                                        ;3
167:      andlw     B'00000111'                                   ;4
168:      btfss     STATUS,Z                                      ;5
169:      goto      CLKIN2                                        ;6,7 goto:2clock
170:      ; CLKIN1 につづいて 8 ビットを受信
171:      movf      RXB1,W                                        ;6,7 skip:2clock
172:      xorlw     H'55'                                         ;8
173:      btfss     STATUS,Z                                      ;9
174:      goto      CLKIN1              ; not 55                  ;10,11
175:      clrf      RXB1                                          ;10,11
176:
177:      movlw     D'224'              ; 28 バイト X8 ビット     ;12
178:      movwf     COUNT                                         ;13
179:
180: HEADER:                 ; 28 バイト送りループ
```

リスト１：クロック同期 → 調歩同期変換プログラム（続き）

```
181:      bsf       PORTE,1                                    ; CLK_H
182:      NOP                                                  ;1
183:      NOP                                                  ;2
184:      NOP                                                  ;3
185:      NOP                                                  ;4
186:      bcf       PORTE,1                                    ; CLK_L
187:      decf      COUNT,F                                    ;5
188:      btfss     STATUS,Z                                   ;6
189:      goto      HEADER                                     ;7,8
190:                                                           ;7
191: CLKIN3:                ; データスタートマーカ１文字目判定
192:      bsf       PORTE,1                                    ; CLK_H
193:      rlf       RXB1,F            ; 受信バッファ左シフト   ;1
194:      decf      BITC,F                                     ;2
195:      NOP                                                  ;3
196:      NOP               ; 時間調整                         ;4
197:      bcf       PORTE,1                                    ; CLK_L
198:      movf      PORTE,W           ; RE->W                  ;1
199:      addwf     RXB1,F                                     ;2
200:      movf      BITC,W                                     ;3
201:      andlw     B'00000111'                                ;4
202:      btfss     STATUS,Z                                   ;5
203:      goto      CLKIN3                                     ;6,7
204:      ; HEADERにつづいて8ビットを受信
205:      movf      RXB1,W                                     ;6,7
206:      xorlw     H'55'                                      ;8
207:      btfss     STATUS,Z                                   ;9
208:      goto      CLKIN1            ; not 55                 ;10,11
209:      clrf      RXB1                                       ;10,11
210:
211: CLKIN4:                ; データスタートマーカ２文字目判定
212:      bsf       PORTE,1                                    ; CLK_H
213:      rlf       RXB1,F            ; 受信バッファ左シフト   ;1
214:      decf      BITC,F                                     ;2
215:      NOP                                                  ;3
216:      NOP               ; 時間調整                         ;4
217:      bcf       PORTE,1                                    ; CLK_L
218:      movf      PORTE,W           ; RE->W                  ;1
219:      addwf     RXB1,F                                     ;2
220:      movf      BITC,W                                     ;3
221:      andlw     B'00000111'                                ;4
222:      btfss     STATUS,Z                                   ;5
223:      goto      CLKIN4                                     ;6,7
224:                          ; CLKIN3につづいて8ビットを受信
225:      movf      RXB1,W                                     ;6,7
226:      xorlw     H'AA'                                      ;8
227:      btfss     STATUS,Z                                   ;9
228:      goto      CLKIN1            ; not AA                 ;10,11
229:      clrf      RXB1                                       ;10,11
230:
231:      ; カメラから送られてくるデータのうち、画像データ部分が検出
232:      ; された場合に、PORTDの７ビット目につないだLEDが光る
233:      bsf       PORTD,7           ; データブロック検出信号
234:
235: ; まずヘッダスターとマーカ AA55を 受信し、28バイト送って
236: ; データスタートマーカ 55AA を受信できた場合にだけここに到達する
237: ; 以上の動作により、バイトの同期もとっている。
238: ; この後受信するデータが画像データ本体
239:
240: ; クロック同期で受信したカメラからのデータはRXB1の0ビット目に入れ
241: ; 左シフトすることによりMSB側へ送る
242: ; 8ビット受信したらRXB1->RXB0へ
243: ; ポート操作によるRS-232CはRXB0のLSB側から送り出す
244: ; 送信するデータの１バイト目はデータが入っていないため00が送られるが
245: ; 受信側で無視することにする。
```

リスト１：クロック同期 → 調歩同期変換プログラム（続き）

```
246:
247: ; RS-232C 送信のビット変化は１命令ステップのずれは許容とする
248:
249:;    bcf     STATUS,C ; キャリーをクリアしておく
250:
251: CLKIN_N:
252:     bsf     PORTE,1                                 ;7  CLK_H7
253:     bcf     PORTC,6         ; スタートビット        ;0 TX
254:     clrf    RXB1                                    ;1
255:     NOP                                             ;2
256:     bcf     PORTE,1                                 ;3  CLK_L7
257:     movf    PORTE,W         ; RE->W                 ;4  Bit7 読み出し
258:     addwf   RXB1,F          ; マスク省略            ;5
259:     bsf     PORTE,1                                 ;6  CLK_H6
260:     btfsc   RXB0,0                                  ;7      H       L
261:     bsf     PORTC,6         ; 0 ビット目が H        ; TX    0
262: ;   NOP             ;ダミー ; 0 ビット目が L        ; TX            0
263:     rlf     RXB1,F          ; RXB1 左シフト         ;1
264:     bcf     PORTE,1                                 ;2  CLK_L6
265:     movf    PORTE,W         ; RE->W                 ;3  Bit6 読み出し
266:     addwf   RXB1,F          ; マスク省略            ;4  退避
267:     bsf     PORTE,1                                 ;5  CLK_H5
268:     btfsc   RXB0,1          ; 1 ビット目検査        ;6
269:     bsf     PORTC,6                                 ; TX    7
270:     btfss   RXB0,1                                  ;       0       7,0
271:     bcf     PORTC,6                                 ; TX    1
272:     rlf     RXB1,F          ; RXB1 左シフト         ;       1,2     2
273:     bcf     PORTE,1                                 ;3  CLK_L5
274:     movf    PORTE,W         ; RE->W                 ;4  Bit5 読み出し
275:     addwf   RXB1,F          ; マスク省略            ;5  退避
276:     btfsc   RXB0,2          ; 2 ビット目検査        ;6
277:     bsf     PORTC,6                                 ; TX    7
278:     btfss   RXB0,2                                  ;       0       7,0
279:     bcf     PORTC,6                                 ; TX              1
280:     bsf     PORTE,1                                 ;  CLK_H4 1,2   2
281:     rlf     RXB1,F          ; RXB1 左シフト         ;3
282:     bcf     PORTE,1                                 ;4  CLK_L4
283:     movf    PORTE,W         ; RE->W                 ;5 Bit4 読み出し
284:     btfsc   RXB0,3          ; 3 ビット目検査        ;6
285:     bsf     PORTC,6                                 ; TX    7
286:     btfss   RXB0,3                                  ;       0       7,0
287:     bcf     PORTC,6                                 ; TX            1
288:     NOP                                             ;       1,2     2
289:     addwf   RXB1,F          ; マスク省略            ;3 退避
290:     bsf     PORTE,1                                 ;4  CLK_H3
291:     rlf     RXB1,F          ; RXB1 左シフト         ;5
292:     btfsc   RXB0,4          ; 4 ビット目検査        ;6
293:     bsf     PORTC,6                                 ; TX    7
294:     btfss   RXB0,4                                  ;       0       7,0
295:     bcf     PORTC,6                                 ; TX            1
296:     bcf     PORTE,1                                 ; CLK_L3 1,2    2
297:     movf    PORTE,W         ; RE->W                 ;3 Bit3 読み出し
298:     addwf   RXB1,F          ; マスク省略            ;4 退避
299:     bsf     PORTE,1                                 ;5  CLK_H2
300:     btfsc   RXB0,5          ; 5 ビット目検査        ;6
301:     bsf     PORTC,6                                 ; TX    7
302:     btfss   RXB0,5                                  ;       0       7,0
303:     bcf     PORTC,6                                 ; TX            1
```

リスト１：クロック同期 → 調歩同期変換プログラム（続き）

```
304:     bcf     PORTE,1                                 ; CLK_L2 1,2 2
305:     movf    PORTE,W         ; RE->W                 ;3 Bit2 読み出し
306:     rlf     RXB1,F          ; RXB 左シフト          ;4
307:     bsf     PORTE,1                                 ;5  CLK_H1
308:     btfsc   RXB0,6          ; 6 ビット目検査        ;6
309:     bsf     PORTC,6                                 ; TX     7
310:     btfss   RXB0,6                                  ;        0        7,0
311:     bcf     PORTC,6                                 ; TX             1
312:     addwf   RXB1,F          ; マスク省略            ; 退避 1,2       2
313:     rlf     RXB1,F          ; RXB 左シフト          ;3
314:     bcf     PORTE,1                                 ;4  CLK_L1
315:     movf    PORTE,W         ; RE->W                 ;5 Bit1 読み出し
316:     btfsc   RXB0,7          ; 7 ビット目検査        ;6
317:     bsf     PORTC,6                                 ; TX     7
318:     btfss   RXB0,7                                  ;        0        7,0
319:     bcf     PORTC,6                                 ; TX             1
320:     bsf     PORTE,1                                 ; CLK_H0 1,2     2
321:     addwf   RXB1,F          ; マスク省略            ;3 退避
322:     rlf     RXB1,F          ; RXB 左シフト          ;4
323:     bcf     PORTE,1                                 ;5  CLK_L0
324:     movf    PORTE,W         ; RE->W                 ;6 Bit0 読み出し
325:     addwf   RXB1,F          ; マスク省略            ;7 退避
326:     bsf     PORTC,6         ; ストップビット        ;0 TX
327:
328:     movf    RXB1,W                                  ;1
329:     movwf   RXB0            ; RXB1->RXB0            ;2
330:     NOP                                             ;3
331:     NOP                                             ;4
332:     goto    CLKIN_N                                 ;5,6
333:
334: ; 止める手段を上記  CLKIN_N  のループの中に埋め込むことは
335: ; ステップ数から考えて不可能
336:
337:     END
```

プログラムをコーディングするにあたり、当初検討した内容では実現できない項目があり、次のように変更した。

１．PIC16F877 モジュールからのデータ送信開始

MS104-SH4 の起動時にシリアルポートからデータ入力があるとフリーズする場合があるらしいので、データ送信開始はまず、MS104-SH4 側からデータ送信をスタートする信号「ST」を送信し、PIC 側で「ST」を受信できた場合にクロック同期方式でカメラから得られた信号を調歩同期方式に変換して送信を開始するような方式とした。ただし、命令ステップ数の関係からデータ送信を止める判定式がかけないため、一度ブロック同期を取ったら、その後の８ビットずつを１バイトとして連続送信するようにしてある。

２．回路への変更

コーディング途中、ポートＥをアナログ、その他のポートをデジタルに設定することが出来ないことが判明、ポートＡとポートＥを逆にした。しかし、温度センサーの処理は余裕が無いため削除。また、カメラからのデータ入力はポートＥのビット０となるようにした。これにより、命令数の削減をすることが出来る。

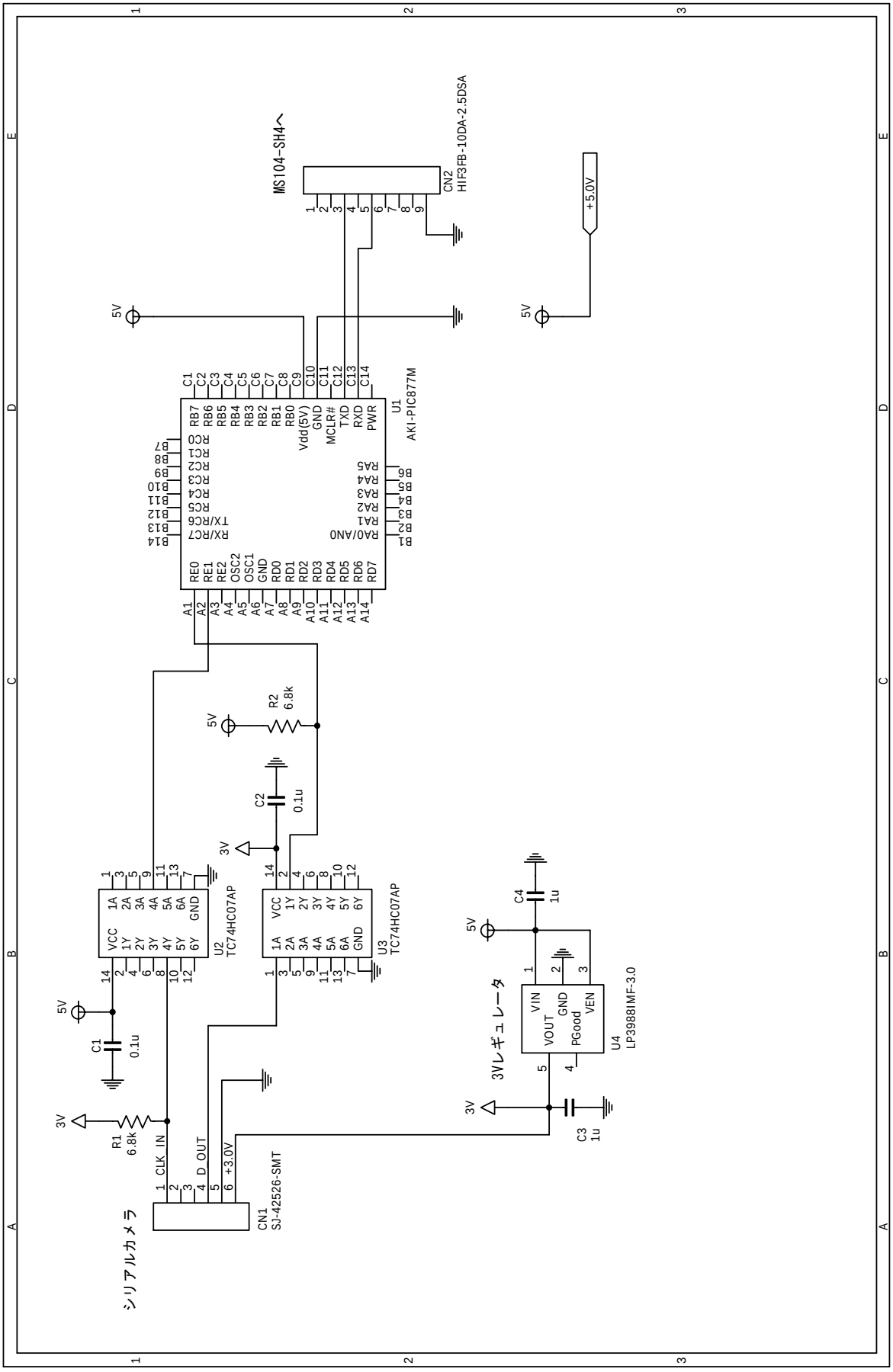


図 15:クロック同期方式→調歩同期方式変換基板回路図(確定版)

３．PIC16F877 の動作クロックについて

標準で実装されている水晶は 20MHz であるが、460.8kbps の速度で通信を行う場合には、このままではエラー率が 9.58%となり、これは実用に耐えうる値ではない。代わりに 14.7456MHz の振動子を使用することによりエラー率を 0%とすることができる。今回は、水晶振動子ではなく温度補償型水晶発振器（TCXO）を使用することにした。(図 16, 図 17)

ボーレート計算式

$$\text{希望のボーレート} = F_{osc} / (16 \times (X + 1))$$

ただし、$X$ はボーレート発生器レジスタ SPBRG に設定する値で整数でなければならない。

・動作クロック $F_{osc}$＝20MHz の場合

$$460800 = 20 \times 10^6 / (16 \times (X + 1)) \qquad X = [1.7126] = 2$$

$$\text{算出ボーレート} = 20 \times 10^6 / (16 \times (2 + 1)) = 416667$$

$$\text{エラー率} = \frac{460800 - 416667}{460800} \times 100 = 9.58\ \%$$

・動作クロック $F_{osc}$＝14.7456MHz の場合

$$460800 = 14.7456 \times 10^6 / (16 \times (X + 1)) \qquad X = 1$$

整数となるので、エラー率は 0%となる。

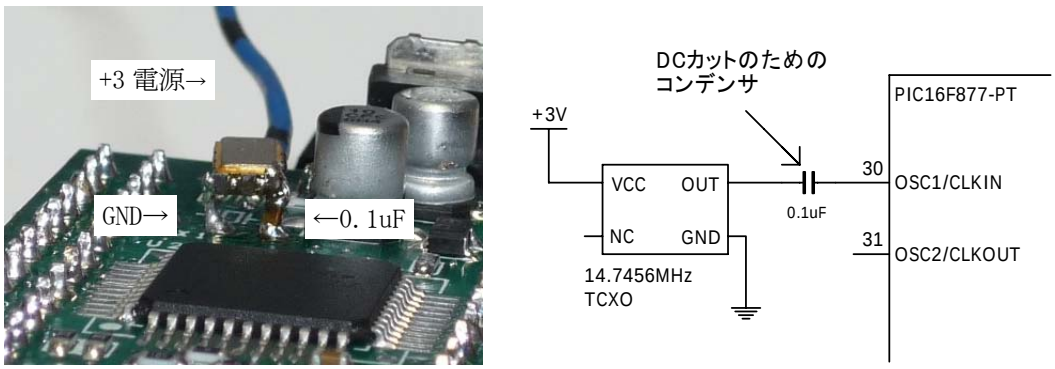


図 16：発振器の実装図と周辺の回路図

４．MS104-SH4 への RS-232C 経由でのデータ送信タイミング

タイミングは PIC16F877 内蔵のタイマーモジュールで行う予定であったが、シリアルカメラのクロック同期方式でのデータ読み出しのタイミングと周期が近いため、タイマー割り込みのタイミングがずれる可能性がある。そのため、タイマーを使用せずに命令ステップ数を利用したデータ送信方式とした。各命令のステップ数を勘案した結果、正確に 460.8kbps の信号変化が不可能なため、各ビットの変化を理想的な信号変化間隔（1/460800 秒）を中心に前後１命令ステップの誤差を許容とし、１ブロックの送信では（1/460800 秒 × 10 ビット）にずれが無いように工夫した。

# 第６章　MS104-SH4 の設定

## ６－１　シリアルポートの設定

MS104-SH4 で Linux を動作させた場合、シリアルポートの速度は標準では 115.2kbps までしか対応していないため、460.8kbps の通信を実現するためには Linux カーネルのシリアルポートドライバに変更を加え、カーネルを再構築しなければならない。以下、その手順を示す。

１．シリアルポートドライバへの変更

SH4 の使用するシリアルポートドライバは kernel/drivers/char/sh-sci.c であり、次のようにソースに変更を加える。

sh-sci.c 480 行付近（行左端に＋がついている行を追加）

```
        case 38400:
                t = BPS_38400;
                break;
        case 57600:
                t = BPS_57600;
                break;
+       case 230400:
+               t = SCBRR_VALUE(230400);
+               break;
+       case 460800:
+               t = SCBRR_VALUE(921600);
+               break;
        defaults:
                printk(KERN_INFO "sci: unsupported baud rate: %d, using 115200 instead.¥n", baud);
        case 115200:
                t = BPS_115200;
                break;
        }
```

２．カーネルの再コンパイル

カーネルソースのトップディレクトリで make menuconfig を実行し、メニューの一番下から２番目の Load Alternate Configurations File を選択する。コンフィギュレーションファイルは arch/sh/def-configs/ms104-sh4/ms104sh4.config を使用する。メニュー画面の Exit を選択し、設定を保存し終了。

続いてコンパイルを行う。

```
make dep; make clean; make zImage
```

と入力してしばらく待つと、arch/sh/boot 以下に、zImage というファイルが出来る。このファイルがカーネルイメージとなる。コンパイラは自動的に sh4-linux-gcc が使用される。

３．カーネルのインストールと再起動

作成されたzImageをMS104-SH4ボードのファイルシステムの/boot以下にvmlinuzと名前を変更し保存後、rebootコマンドで再起動する。再起動の途中でmodprobeがエラーを吐くが、今回カーネルドライバをモジュールで組み込んでいないので、無視してよい。ログインプロンプトが表示されれば再構築されたカーネルで起動したことになる。

４．460.8kbpsシリアル通信の動作確認

Windowsマシンで確認を行う。通信ソフトはTeraTermを使用する。古いTeraTermでは、460.8kbpsに対応していないので、最新のものを使用するとよい。Windowsを使用した場合、オンボードの標準的なシリアルポートドライバは115.2kbpsまでしか対応していないため、USBシリアル変換ケーブルを使用する。

USB経由でMS104-SH4のコンソールを接続し、38.4kbps（起動時初期設定）に設定し起動する。rootでログイン後、

```
stty 460800
```

とコマンドを打つと、460.8kbpsに設定できる。このままでは、文字化けをするので、TeraTermの設定を460.8kbpsに設定しなおす。数回Enterキーを押してコマンドプロンプトが正常に表示されれば、成功である。確認のため、sttyとだけ打つと

```
speed 460800 baud; line = 0;
kill = ^X;
-brkint ixany -imaxbel
-iexten -echok
```

と表示され、確かに460.8kbpsとなっていることが確認できる。

## ６－２　シリアルポート受信プログラム（MS104-SH4）

Linux では、シリアルポートなどの入出力デバイスは、デバイスドライバが組み込まれれば、通常のファイルを扱うように操作することが出来る。MS104-SH4 では、シリアルポートに相当するファイルは、COM1:/dev/ttySC0 と COM2:/dev/ttySC1 が該当する。今回カメラからのデータを受信するのは COM1 なので、/dev/ttySC0 を使用する。

１．受信テスト①

実際に、シリアルポートでカメラからのデータが受信できるかテストプログラムを作成し、PIC から送られてくるデータを表示してみる。

リスト 2：RS-232C 受信テストプログラム

```
 1: #include <sys/types.h>
 2: #include <sys/stat.h>
 3: #include <fcntl.h>
 4: #include <termios.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7:
 8: #define BAUDRATE B460800
 9: #define MODEMDEVICE "/dev/ttySC0"
10:
11: int main(void)
12: {
13:     int fd, res;
14:     struct termios oldtio,newtio;
15:     unsigned char buf[255];
16:
17:     fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);
18:     if (fd <0) {perror(MODEMDEVICE);  exit(-1);}
19:
20:     tcgetattr(fd,&oldtio); /* 現在のポート設定を待避 */
21:
22:     bzero(&newtio, sizeof(newtio));
23:     newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
24:     newtio.c_iflag = IGNPAR;
25:     newtio.c_oflag = 0;
26:
27:     /* set input mode (non-canonical, no echo,...) */
28:     newtio.c_lflag = 0;
29:
30:     newtio.c_cc[VTIME]    = 0;   /* キャラクタ間タイマは未使用 */
31:     newtio.c_cc[VMIN]     = 1;   /* 1 文字受け取るまでブロックする */
32:     tcflush(fd, TCIFLUSH);            /* バッファをクリアする */
33:
34:     tcsetattr(fd,TCSANOW,&newtio);    /* 設定 */
35:
36:     while (1) {       /* 入力ループ */
37:             res = read(fd,buf,255);
38:             printf("%X¥n", buf[0]);
39:     }
40:
41:     tcsetattr(fd,TCSANOW,&oldtio);    /* 設定を元に戻す */
42:     close(fd);
43:     return 0;
44: }
```

まずはじめに、PIC16F877 モジュールから、460.8kbps の調歩同期方式で一定の値を送信し、正しく受信できるか確認を行う。そのために PIC のプログラムに変更を加えた。変更点は、各種ポートの設定が終わった後の命令、リスト 5 の 73 行目と 136 行目のコメントをはずし、最後のループにジャンプする設定に変更する。また、330,331 行目の NOP の部分を以下のように変更した。

```
328:    movf    RXB1,W
329:    movwf   RXB0
330:    NOP
331:    NOP
332:    goto    CLKIN_N
```

⇒

```
328:    movf    RXB1,W
329:    movwf   RXB0
330:    movlw   H'41'
331:    movwf   RXB0
332:    goto    CLKIN_N
```

変更することにより、PIC16F877 モジュールからは 16 進数で 0x41（文字 'A'）が 460.8kbps の調歩同期方式で連続送信されることになる。

シリアルポート COM1 に 5-1 で製作した基板を取り付け、上記プログラム実行するとコンソール画面に２桁の 16 進数 '41' が表示されるはずであったが、表示されたのは '5' の文字の連続であった。PIC16F877 モジュールも見たところ実装されている RS-232C ドライバ IC が 460.8kbps の高速度通信に使えるのか疑問に思ったのでデータシートを検索してみた。

使用されている RS-232C ドライバ IC：SIPEX SP202ECN
対応速度：MAX 230kbps

であり、正しく受信できなかった原因と考えられた。急遽 460.8kbps に対応している RS-232C ドライバ IC を探したところ、簡単に入手でき、そのまま載せ替えができる AnalogDevices 社の ADM3202ARN があり、その RS-232C ドライバに変更することにした。（秋月電子通商で購入）

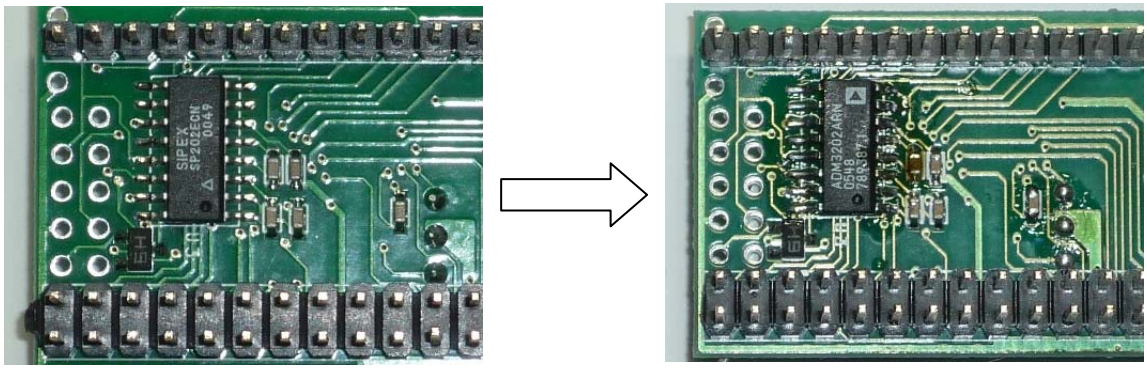


図 17：RS-232C ドライバ IC の換装

写真ではちょっときたないが、変更後、連続して '41' が表示されたため、受信は正しく行われていると確認できた。文字化けの原因は予想どおり RS-232C ドライバ IC が 460.8kbps に対応していなかったことであった。

２．受信テスト②

PIC16F877 モジュールのプログラムの最後から２～３行目を NOP に戻し、カメラからのデータを受信し、標準出力からファイルにリダイレクトで書き込む。画像データは連続して送られてくるはずなので、しばらく受信して Ctrl-c で停止する。

```
# ./a.out > data
```

コンパイルした実行ファイルは a.out である。作成されたデータを確認してみると途中連続して 'FF' が存在することが確認できた。使用したシリアルカメラ DMR-C1 に類似した Treva という携帯電話に接続して使用するカメラがあるが、その Treva を解析したホームページ「Treva 解析のページ」によると、それぞれの画像ごとの間（フレーム間）には連続して 'FF' が送信されるとのことであった。

受信したデータの 'FF' が連続していない部分を確認したところ、880 バイトであり、想定される画像データ 202786 バイト（ヘッダ 32＋画像データ 352×288×2＋エンドマーカ 2）よりも非常に小さい値となっている。受信の設定がおかしいと考え、いろいろ設定を変更して受信を試みたが、うまくいかなかった。

試しに、cat コマンドを使って、/dev/ttySC0 を直接ファイルにダンプしてみることにした。受信停止は ctrl-c で停止する。

```
# cat /dev/ttySC0 > data
```

受信したデータをバイナリエディタで表示したところ、同様に 'FF' が連続している部分が確認でき、データに相当する部分が約 202800 バイトあったので、正しく受信できていると思われた。しかし、'FF' が連続している部分の前にデータエンドマーカである 'FF D9' が確認できなかった。

続いて cat コマンドを実行した後にシリアルカメラ側の電源を入れて受信したデータを確認したところ、最初の 'FF の連続している部分の直前に 'FF D9' が確認できたが、２つ目は '1F FE CF FF FF …' ３つ目は '7F EC FF FF …' でありデータエンドマーカが確認できなかった。

しかし、'FF D9' が確認できなかった２つ目と３つ目のデータ部分について

・２つ目の連続 'FF' の直前部分

```
  1    F    F    E    C    F    F    F
0001 1111 1111 1110 1100 1111 1111 1111…
     └──┘ └──┘ └──┘ └──┘
      F    F    D    9
```

・３つ目の連続 'FF' の直前部分

```
  7    F    E    C    F    F    F
0001 1111 1111 1110 1100 1111 1111 …
     └──┘ └──┘ └──┘ └──┘
      F    F    D    9
```

２進数に変換してみると、データエンドマーカである 'FF D9' が確認でき、受信データがビットずれを起こしているらしいことが確認できた。

## ６－３　画像データのビットマップ変換と確認

cat コマンドを実行してからシリアルカメラの電源を入れ受信したデータの最初の連続 'FF D9' が現れるまでの間のデータについて、'FF D9' よりも前の部分 202752 バイト（352×288×2）をバイナリエディタで抽出し、image.dat というファイルに保存した。このファイルは YUV422 形式（UYVY 形式）画像データが保存されている。このデータをビットマップ形式のイメージファイルに変換し、正しい画像データなのか確認する。

YUV422 形式（UYVY 形式）は、多くある YUV 形式の中でも最も多く使われる形式で、２ピクセルを４バイトで表現する。YUV の Y は輝度を表し、U は青色差（青－輝度)、V は赤色差（赤－輝度）を表す。422 は１ピクセルあたりのデータの比率で、Y:U:V＝4:2:2(=8:4:4)であることを示す。データの並びを U0, Y0, V0, Y1 とすると

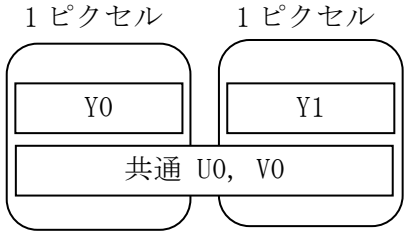


図 18：YUV422 のデータ形式

図 18 のように、最初の１ピクセルは、Y0, U0, V0 の３つのデータから構成され、続く１ピクセルは Y1, U0, V0 の３つのデータから構成される。

一方、ビットマップ形式は、ヘッダ部分、情報ヘッダ部分に続いて画像データ部分が続く形式をしており、画像データ部分は画像の高さが正の値の場合、画像の最下辺のデータから右上に向かって RGB データが BGR の順に並んでいる構造をもつ。

表 2：ビットマップファイルの構造

| | | | |
|---|---|---|---|
| ファイルヘッダ | ファイルタイプ | ２バイト | 'BM' |
| | ファイルサイズ[byte] | ４バイト | 今回は 304182 |
| | 予約領域１ | ２バイト | 常に 0 |
| | 予約領域２ | ２バイト | 常に 0 |
| | ファイル先頭から画像データまでのオフセット[byte] | ４バイト | 54 |
| 情報ヘッダ | 情報ヘッダサイズ[byte] | ４バイト | 40 |
| | 画像の幅[ピクセル] | ４バイト | 今回は 352 |
| | 画像の高さ[ピクセル] | ４バイト | 今回は-288(0xFFFFFEE0)<br>負数の場合は画像データが左上から右下に記録（一般的でない） |
| | プレーン数 | ２バイト | 常に１ |
| | 色ビット数[bit] | ２バイト | 今回は 24 |
| | 圧縮形式 | ４バイト | 非圧縮＝0 |
| | 画像データサイズ[byte] | ４バイト | 今回は 304128 |
| | 水平解像度[dot/m] | ４バイト | 2835（0 の場合もある） |

| | | | |
|---|---|---|---|
| | 垂直解像度[dot/m] | ４バイト | 2835（0 の場合もある |
| | 格納パレット数[使用色数] | ４バイト | 0 |
| | 重要色数 | ４バイト | 0 |
| | パレットデータ | ０バイト | 使用しない |
| | 画像データ | 今回は 304128 バイト | 各ピクセルのデータは行単位で記録され、行データは４バイト境界に揃える。 |

各データは事前に Windows のペイントツールで 352×288 サイズの 24 ビットカラーのビットマップ形式のファイルを作成し、それに使われているデータを流用する。ファイルタイプと画像データを除いて、各データはリトルエンディアン、つまり後ろのバイトデータの方が上位にある形式で記録されていることに注意する。

YUV422 形式（UYVY 形式）からビットマップの画像データ RGB への変換式は

$$
\begin{aligned}
R &= Y0 + 1.402\times(V0-128) \\
G &= Y0 - 0.344\times(U0-128) - 0.714\times(V0-128) \\
B &= Y0 + 1.772\times(U0-128)
\end{aligned}
$$

上式は１ピクセル目の変換式であるが、続く１ピクセルの変換式は、上式で Y0 を Y1 に変換すればよい。R, G, B の各データは１バイトなので 0～255 までしか表せないため、RGB に変換した値が 0 よりも小さい場合は 0 とし、255 よりも大きい場合は 255 としなければならない。以上の手順により YUV422 画像データから４バイトずつ読み込んで変換式で RGB 画像データを作成することにより、YUV422 形式からビットマップ形式への変換が可能となる。

リスト 3：YUV422 形式→ビットマップ形式変換プログラム yuv2bmp.c

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3:
 4: int main(void)
 5: {
 6:     FILE *fout;
 7:     FILE *fin;
 8:     int i, c;
 9:     double u0,y0,v0,y1,r,g,b;
10:     unsigned char header[] = {0x42,0x4D,0x36,0xA4,0x04,0,0,0,0,0,0x36,0,0,0};
11:     unsigned char infohead[] = {0x28,0,0,0,0x60,1,0,0,0xE0,0xFE,0xFF,0xFF,
12:     1,0,0x18,0,0,0,0,0,0,0xA4,4,0,0x13,0x0B,0,0,0x13,0x0B,0,0,0,0,0,0,0,0,0,0};
13:
14:     if((fout = fopen ("/root/test.bmp","wb+")) == NULL) {
15:             printf("file open error!!¥n");
16:             exit(EXIT_FAILURE);
17:     };
18:
19:     /* BMP ヘッダ書き出し */
20:     for(i=0;i<14;i++) {
21:             fputc(header[i],fout);
22:     }
23:     for(i=0;i<40;i++) {
24:             fputc(infohead[i],fout);
25:     }
```

リスト 3：YUV422 形式→ビットマップ形式変換プログラム yuv2bmp.c（続き）

```
26:     /* YUV422 データファイルオープン */
27:     if((fin = fopen ("/root/image.dat","rb")) == NULL) {
28:             perror(NULL);
29:             printf("data file open error!!¥n");
30:             exit(EXIT_FAILURE);
31:     };
32:
33:     for(i=1;i<=50688;i++) {           /* (352x288x2)/4=50688 回ループ */
34:             u0 = fgetc(fin);
35:             y0 = fgetc(fin);
36:             v0 = fgetc(fin);
37:             y1 = fgetc(fin);
38:
39:             r = (256*y0 +                   359*(v0 - 128))/256;
40:             g = (256*y0 -  88*(u0 - 128) - 183*(v0 - 128))/256;
41:             b = (256*y0 + 454*(u0 - 128)                  ) /256;
42:
43:             if(255 < r) r = 255;
44:             if(r < 0) r = 0;
45:             if(255 < g) g = 255;
46:             if(g < 0) g = 0;
47:             if(255 < b) b = 255;
48:             if(b < 0) b = 0;
49:
50:             // BMP パレットは BGR の順
51:             // first pixel
52:             fputc((unsigned char)b, fout);
53:             fputc((unsigned char)g, fout);
54:             fputc((unsigned char)r, fout);
55:
56:             r = (256*y1 +                   359*(v0 - 128))/256;
57:             g = (256*y1 -  88*(u0 - 128) - 183*(v0 - 128))/256;
58:             b = (256*y1 + 454*(u0 - 128)                  ) /256;
59:
60:             if(255 < r) r = 255;
61:             if(r < 0) r = 0;
62:             if(255 < g) g = 255;
63:             if(g < 0) g = 0;
64:             if(255 < b) b = 255;
65:             if(b < 0) b = 0;
66:
67:             // second pixel
68:             fputc((unsigned char)b, fout);
69:             fputc((unsigned char)g, fout);
70:             fputc((unsigned char)r, fout);
71:     }
72:
73:
74:     fclose(fin);
75:     fclose(fout);
76: }
```

プログラムを実行することにより、YUV422 形式のデータファイル(/root/image.dat)からビットマップ形式の画像(/root/test.bmp)に変換される。

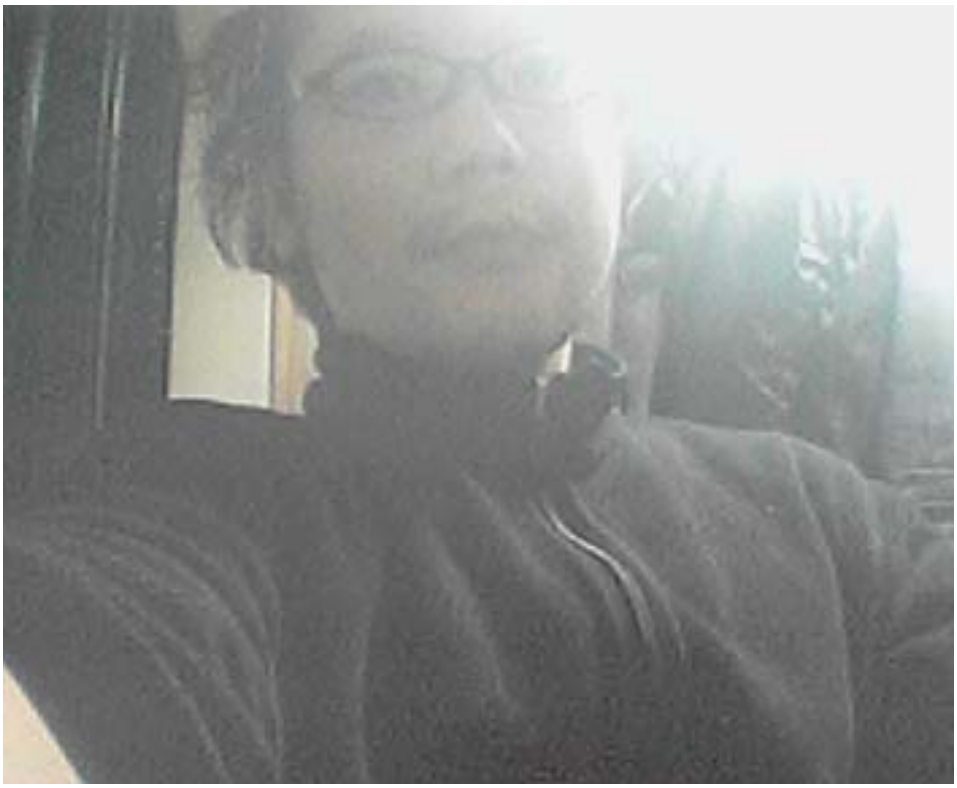

図 19：シリアルカメラで撮影された画像第 1 号

ビットマップ画像に撮影画像が変換できたので、ここで初めてシリアルカメラからの画像データが正しく受信できていることが確認できた。

## ６－４　ビットずれ対策の検討

画像データの連続受信の際に発生するビットずれの対策として、考えられる方法が２つある。１つは、受信側の MS104-SH4 で受信データを２進数に分解し、ビットずれ量を検出して正しいデータに修正する方法。もう１つは、送信側の PIC16F877 モジュールでビットずれを修正し送信する方法である。

受信側でデータを修正する場合は受信データの扱いが煩雑になるため、送信側でずれを修正し送信する方法を検討する。

ビットずれを検出するには、正しい画像データの範囲を特定できなければならない。PIC のプログラム リスト 5 において CLKIN1～CLKIN4 はデータのブロック同期をとるための部分であるから、ここを再度利用できればよい。また、いくつか受信した画像データの部分を確認したところ、'FF' というデータが無く、またデータエンドマーカが 'FF D9' というように 'FF' で始まっているため、'FF' のデータを見つけたらデータの終わりと判断し、CLKIN1 へ戻るようにプログラムが変更できればよい。必要な命令はWレジスタにクロック同期で受信したデータが入っているとして次の３行である。

```
        xorlw   H'FF'
        btfsc   STATUS,Z
        goto    CLKIN1
```

この命令を、最後の部分に追加しなければならないのだが、最後の部分に空いている行は NOP 命令の２行しかない。しかし、リスト 5 の 288 行目の NOP 命令をなくし、順に上に詰めることにより、３行分を確保することが出来る。

なお、実験の途中に MS104-SH4 起動時にシリアルポートからデータ入力があってもフリーズしないことが確認できたため、スタート信号を RS-232C で受信する部分は必要ないと判断し、削除した。

リスト 4：クロック同期 → 調歩同期変換プログラム（ビットずれ対策版）

```
 1: ; クロック同期　－＞ 調歩同期変換プログラム（ビットずれ対策版）
 2:
 3:      LIST            P=PIC16F877
 4:      INCLUDE  <P16F877.INC>
 5:
 6: ;-----------------------------------------------------------------------------
 7:
 8:      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC & _LVP_OFF
 9:
10: ;-----------------------------------------------------------------------------
11:
12: RXB0        EQU             20H             ; 送信用バッファ
13: RXB1        EQU             21H             ; 受信用バッファ
14: SCIF        EQU             23H             ; クロック同期受信ステータス保存用
15: BITC        EQU             24H             ; ビットカウンタ
16: COUNT EQU           25H             ; ヘッダ送り用
17:
18: ;-----------------------------------------------------------------------------
19:
20:      org     0
21:      goto    START
22:
23:      org     8
24:
```

リスト４：クロック同期 → 調歩同期変換プログラム（ビットずれ対策版）（続き）

```
START:
; 各種ポートの設定
        ; 入力ピンが浮いていると誤動作の原因となるため必要なもの以外は出力に変更
        ; A/D の設定          A port:アナログ     E port:デジタル
        movlw     B'00000010'
        bsf       STATUS,RP0          ; [Bank1]
        movwf     ADCON1
        bcf       STATUS,RP0          ; [Bank0]
        ; A port  使用しないので出力に固定する
        clrf      PORTA
        movlw     B'00000000'
        bsf       STATUS,RP0          ; [Bank1]
        movwf     TRISA
        bcf       STATUS,RP0          ; [Bank0]
        ; E port  汎用 I/O ポート, RE1:カメラクロック出力, RE0:カメラデータ入力,
        ; その他:出力固定
        clrf      PORTE
        movlw     B'00000001'
        bsf       STATUS,RP0          ; [Bank1]
        movwf     TRISE
        bcf       STATUS,RP0          ; [Bank0]
        ; B port  使用しないので出力固定
        clrf      PORTB
        bsf       STATUS,RP0          ; [Bank1]
        clrf      TRISB
        bcf       STATUS,RP0          ; [Bank0]
        ; C port  RC6 を出力、RC7 を入力
        clrf      PORTC
        movlw     B'10000000'
        bsf       STATUS,RP0          ; [Bank1]
        movwf     TRISC
        bcf       STATUS,RP0          ; [Bank0]
        ; D port  使用しないので出力固定
        clrf      PORTD
        bsf       STATUS,RP0          ; [Bank1]
        clrf      TRISD
        bcf       STATUS,RP0          ; [Bank0]

        movlw     B'00000000'         ; 割込み禁止
        movwf     INTCON              ; 多少早くなるらしい

        clrf      SCIF                ; クロック同期受信ステータスクリア
        clrf      RXB0                ; 受信バッファクリア
        clrf      RXB1
        clrf      BITC                ; 受信ビットカウンタクリア

        bsf       PORTC,6             ; RS-232C は無信号時は H レベル

; シリアルカメラのクロック同期の受信開始（RE1：クロック出力, RE0：データ入力）
; クロック周期は約 4 マイクロ秒（正確でなくて良い）　一方、SH4 ボードへの
; RS-232C（ポート ON/OFF による）出力は 460.8kbps を維持しなければならない

; 460.8kbps では
; RE0 ポート H -> RE0 ポート L, RE1 ポート読み込み -> RE0 ポート H ・・・
; ２マイクロ秒ごとに RE1 状態遷移　その間に置ける命令数は７（2.17 マイクロ秒）
; クロックが H->L->(H)が 16 命令ならば細かいことは気にしない
; 最短でクロックを変化させると周期が２命令ステップ（1.8MHz）
; この半分くらいの速度ならカメラから読み出せるらしい

; 最初のデータ受信では受信済みのデータが無いので RS-232C 送信はなし
; 単にカメラからのクロック同期受信のみ

;; マーカ読み出し周期 18 命令ステップ

```

リスト４：クロック同期 → 調歩同期変換プログラム（ビットずれ対策版）（続き）

```
 89: CLKIN1:                ; ヘッダスタートマーカ１文字目判定
 90:      bsf       PORTE,1                                  ; CLK_H
 91:      rlf       RXB1,F             ; 受信バッファ左シフト      ;1
 92:      NOP                                                ;2
 93:      NOP                                                ;3
 94:      NOP                                                ;4
 95:      NOP                                                ;5
 96:      NOP                          ; 時間調整            ;6
 97:      bcf       PORTE,1                                  ; CLK_L
 98:      movf      PORTE,W            ; RE->W               ;1
 99:      addwf     RXB1,F                                   ;2
100:      NOP                                                ;3
101:      NOP                                                ;4
102:      movf      RXB1,W                                   ;5
103:      xorlw     H'AA'                                    ;6
104:      btfss     STATUS,Z                                 ;7
105:      goto      CLKIN1             ; not AA              ;8,9 goto:2clock
106:      clrf      RXB1                                     ;8,9 skip:2clock
107:
108: CLKIN2:                ; ヘッダスタートマーカ２文字目判定
109:      bsf       PORTE,1                                  ; CLK_H
110:      rlf       RXB1,F             ; 受信バッファ左シフト      ;1
111:      decf      BITC,F                                   ;2
112:      NOP                                                ;3
113:      NOP                ; 時間調整                      ;4
114:      bcf       PORTE,1                                  ; CLK_L
115:      movf      PORTE,W            ; RE->W               ;1
116:      addwf     RXB1,F                                   ;2
117:      movf      BITC,W                                   ;3
118:      andlw     B'00000111'                              ;4
119:      btfss     STATUS,Z                                 ;5
120:      goto      CLKIN2                                   ;6,7 goto:2clock
121:      ; CLKIN1につづいて８ビットを受信
122:      movf      RXB1,W                                   ;6,7 skip:2clock
123:      xorlw     H'55'                                    ;8
124:      btfss     STATUS,Z                                 ;9
125:      goto      CLKIN1             ; not 55              ;10,11
126:      clrf      RXB1                                     ;10,11
127:
128:      movlw     D'224'             ; 28バイトX8ビット    ;12
129:      movwf     COUNT                                    ;13
130:
131: HEADER:                ; 28バイト送りループ
132:      bsf       PORTE,1                                  ; CLK_H
133:      NOP                                                ;1
134:      NOP                                                ;2
135:      NOP                                                ;3
136:      NOP                                                ;4
137:      bcf       PORTE,1                                  ; CLK_L
138:      decf      COUNT,F                                  ;5
139:      btfss     STATUS,Z                                 ;6
140:      goto      HEADER                                   ;7,8
141:                                                         ;7
142: CLKIN3:                ; データスタートマーカ１文字目判定
143:      bsf       PORTE,1                                  ; CLK_H
144:      rlf       RXB1,F             ; 受信バッファ左シフト      ;1
145:      decf      BITC,F                                   ;2
146:      NOP                                                ;3
147:      NOP                ; 時間調整                      ;4
148:      bcf       PORTE,1                                  ; CLK_L
149:      movf      PORTE,W            ; RE->W               ;1
150:      addwf     RXB1,F                                   ;2
151:      movf      BITC,W                                   ;3
152:      andlw     B'00000111'                              ;4
153:      btfss     STATUS,Z                                 ;5
```

リスト４：クロック同期 → 調歩同期変換プログラム（ビットずれ対策版）（続き）

```
154:      goto      CLKIN3                                        ;6,7
155:      ; HEADER につづいて 8 ビットを受信
156:      movf      RXB1,W                                        ;6,7
157:      xorlw     H'55'                                         ;8
158:      btfss     STATUS,Z                                      ;9
159:      goto      CLKIN1              ; not 55                  ;10,11
160:      clrf      RXB1                                          ;10,11
161:
162: CLKIN4:                    ; データスタートマーカ２文字目判定
163:      bsf       PORTE,1                                       ; CLK_H
164:      rlf       RXB1,F              ; 受信バッファ左シフト    ;1
165:      decf      BITC,F                                        ;2
166:      NOP                                                     ;3
167:      NOP                 ; 時間調整                        ;4
168:      bcf       PORTE,1                                       ; CLK_L
169:      movf      PORTE,W             ; RE->W                   ;1
170:      addwf     RXB1,F                                        ;2
171:      movf      BITC,W                                        ;3
172:      andlw     B'00000111'                                   ;4
173:      btfss     STATUS,Z                                      ;5
174:      goto      CLKIN4                                        ;6,7
175:                          ; CLKIN3 につづいて 8 ビットを受信
176:      movf      RXB1,W                                        ;6,7
177:      xorlw     H'AA'                                         ;8
178:      btfss     STATUS,Z                                      ;9
179:      goto      CLKIN1              ; not AA                  ;10,11
180:      clrf      RXB1                                          ;10,11
181:
182:      ; 以上の動作により、バイトの同期をとっている。
183:      ; カメラから送られてくるデータのうち、画像データ部分が検出
184:      ; された場合に、PORTD の７ビット目につないだ LED が光る
185:      bsf       PORTD,7             ; データブロック検出信号
186:
187:      ; 送信用バッファ RXB0 にはデータ区切りの目印 FF を入れておく
188:      movlw     H'FF'
189:      movwf     RXB0
190:
191: ; クロック同期で受信したカメラからのデータは RXB1 の 0 ビット目に入れ
192: ; 左シフトすることにより MSB 側へ送る
193: ; 8 ビット受信したら RXB1->RXB0 へ
194: ; ポート操作による RS-232C は RXB0 の LSB 側から送り出す
195: ; 送信するデータは、FF に続いて画像データが送られる。
196: ; この FF は受信側でデータの区切りの目印として使用する。
197:
198: ; RS-232C 送信のビット変化は１命令ステップのずれは許容とする
199:
200: CLKIN_N:
201:      bsf       PORTE,1                                       ;7  CLK_H7
202:      bcf       PORTC,6             ; スタートビット          ;0 TX
203:      clrf      RXB1                                          ;1
204:      NOP                                                     ;2
205:      bcf       PORTE,1                                       ;3  CLK_L7
206:      movf      PORTE,W             ; RE->W                   ;4  Bit7 読み出し
207:      addwf     RXB1,F              ; マスク省略              ;5  Bit7 退避
208:      bsf       PORTE,1                                       ;6  CLK_H6
209:      btfsc     RXB0,0                                        ;7       H         L
210:      bsf       PORTC,6             ; 0 ビット目が H          ; TX     0
211: ;    NOP       ダミー              ; 0 ビット目が L          ; TX               0
212:      rlf       RXB1,F              ; RXB1 左シフト           ;1
213:      bcf       PORTE,1                                       ;2  CLK_L6
214:      movf      PORTE,W             ; RE->W                   ;3  Bit6 読み出し
215:      addwf     RXB1,F              ; マスク省略              ;4  Bit6 退避
216:      bsf       PORTE,1                                       ;5  CLK_H5
217:      btfsc     RXB0,1              ; 1 ビット目検査          ;6
218:      bsf       PORTC,6                                       ; TX     7
```

リスト４：クロック同期 → 調歩同期変換プログラム（ビットずれ対策版）（続き）

```
219:    btfss   RXB0,1                                  ;       0       7,0
220:    bcf     PORTC,6                                 ; TX            1
221:    bcf     PORTE,1                                 ;  CLK_L5  1,2      2
222:    rlf     RXB1,F          ; RXB1 左シフト         ;3
223:    movf    PORTE,W         ; RE->W                 ;4  Bit5 読み出し
224:    bsf     PORTE,1                                 ;5 CLK_H4
225:    btfsc   RXB0,2          ; 2 ビット目検査        ;6
226:    bsf     PORTC,6                                 ; TX      7
227:    btfss   RXB0,2                                  ;       0       7,0
228:    bcf     PORTC,6                                 ; TX            1
229:    addwf   RXB1,F          ; マスク省略            ;  Bit5 退避 1,2    2
230:    rlf     RXB1,F          ; RXB 左シフト          ;3
231:    bcf     PORTE,1                                 ;4  CLK_L4
232:    movf    PORTE,W         ; RE->W                 ;5 Bit4 読み出し
233:    btfsc   RXB0,3          ; 3 ビット目検査        ;6
234:    bsf     PORTC,6                                 ; TX      7
235:    btfss   RXB0,3                                  ;       0       7,0
236:    bcf     PORTC,6                                 ; TX            1
237:    bsf     PORTE,1                                 ; CLK_H3 1,2        2
238:    addwf   RXB1,F          ; マスク省略            ;3 Bit4 退避
239:    rlf     RXB1,F          ; RXB 左シフト          ;4
240:    bcf     PORTE,1                                 ;5 CLK_L3
241:    btfsc   RXB0,4          ; 4 ビット目検査        ;6
242:    bsf     PORTC,6                                 ; TX      7
243:    btfss   RXB0,4                                  ;       0       7,0
244:    bcf     PORTC,6                                 ; TX            1
245:    movf    PORTE,W         ; RE->W                 ; Bit3 読み出し 1,2 2
246:    bsf     PORTE,1                                 ;3 CLK_H2
247:    addwf   RXB1,F          ; マスク省略            ;4 Bit3 退避
248:    bcf     PORTE,1                                 ;5 CLK_L2
249:    btfsc   RXB0,5          ; 5 ビット目検査        ;6
250:    bsf     PORTC,6                                 ; TX      7
251:    btfss   RXB0,5                                  ;       0       7,0
252:    bcf     PORTC,6                                 ; TX            1
253:    rlf     RXB1,F          ; RXB 左シフト          ;       1,2       2
254:    movf    PORTE,W         ; RE->W                 ;3 Bit2 読み出し
255:    bsf     PORTE,1                                 ;4  CLK_H1
256:    addwf   RXB1,F          ; マスク省略            ;5 Bit2 退避
257:    btfsc   RXB0,6          ; 6 ビット目検査        ;6
258:    bsf     PORTC,6                                 ; TX      7
259:    btfss   RXB0,6                                  ;       0       7,0
260:    bcf     PORTC,6                                 ; TX            1
261:    rlf     RXB1,F          ; RXB 左シフト          ;       1,2       2
262:    bcf     PORTE,1                                 ;3  CLK_L1
263:    movf    PORTE,W         ; RE->W                 ;4 Bit1 読み出し
264:    bsf     PORTE,1                                 ;5 CLK_H0
265:    btfsc   RXB0,7          ; 7 ビット目検査        ;6
266:    bsf     PORTC,6                                 ; TX      7
267:    btfss   RXB0,7                                  ;       0       7,0
268:    bcf     PORTC,6                                 ; TX            1
269:    addwf   RXB1,F          ; マスク省略            ; Bit1 退避 1,2     2
270:    rlf     RXB1,F          ; RXB 左シフト          ;3
271:    bcf     PORTE,1                                 ;4  CLK_L0
272:    movf    PORTE,W         ; RE->W                 ;5 Bit0 読み出し
273:    addwf   RXB1,F          ; マスク省略            ;6 Bit0 退避
274:    movf    RXB1,W          ; RXB1->W               ;7
275:    bsf     PORTC,6         ; ストップビット        ;0 TX
276:    movwf   RXB0            ; W->RXB0               ;1
277:    xorlw   H'FF'                                   ;2
278:    btfsc   STATUS,Z                                ;3
279:    goto    CLKIN1                                  ;4      同期取り直しへ
280:    goto    CLKIN_N                                 ;5,6    4,5,6
281:
282:    END
```

PIC から送られてくるデータは

FF
YUV422 画像データ 202752 バイト
FF
YUV422 画像データ 202752 バイト
……

というようなデータ形式となる。これに合わせて受信プログラムを変更した。今回は、受信したデータをビットマップファイルにまで変換するところまで組み込んだ。

RS-232C の受信は cat コマンドではうまく取り出せたのを踏まえ、シリアルポートの設定を関数 init_serial()で行い、/dev/ttySC0 は open 文ではなく fopen 文デバイスをオープンし、fgetc 命令で読み出すことに成功した。getImage()関数は、FF と FF の間にあるデータを一時ファイルとして書き出し、データのバイト数を返す関数で、202752 バイトでない場合は 202752 バイトになるまで受信するようにしてある。

リスト 5：RS-232C 受信プログラム（ビットずれ対策版）getbmp.c

```
 1: #include <sys/types.h>
 2: #include <sys/stat.h>
 3: #include <sys/signal.h>
 4: #include <fcntl.h>
 5: #include <termios.h>
 6: #include <stdio.h>
 7: #include <stdlib.h>
 8:
 9: #define BAUDRATE B460800
10: #define MODEMDEVICE "/dev/ttySC0"
11:
12: int int_serial(void);
13: int getImage(void);
14: void makebmp(void);
15:
16: int main(void)
17: {
18:     int count = 0;
19:
20:     init_serial();
21:     while(1) {
22:             if((count = getImage()) == 202752) break;
23:     }
24:     makebmp();
25:
26:     return 0;
27: }
28:
29:
30: int init_serial(void)
31: {
32:     int fd;
33:     struct termios newtio;
34:     // シリアルポート初期設定
35:     // これをやっておかないとうまく受信できない
36:     fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);
37:     if (fd <0) {perror(MODEMDEVICE); exit(-1); }
38:
```

リスト５：RS-232C 受信プログラム（ビットずれ対策版）getbmp.c （続き）

```
39:     bzero(&newtio, sizeof(newtio));
40:     newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
41:     newtio.c_iflag = IGNPAR;
42:     newtio.c_oflag = 0;
43:
44:     /* set input mode (non-canonical, no echo,...) */
45:     newtio.c_lflag = 0;
46:
47:     newtio.c_cc[VTIME]    = 0;   /* キャラクタ間タイマは未使用 */
48:     newtio.c_cc[VMIN]     = 1;   /* ５文字受け取るまでブロックする */
49:
50:     tcflush(fd, TCIFLUSH);
51:     tcsetattr(fd,TCSANOW,&newtio);
52:
53:     close(fd);
54: }
55:
56:
57: int getImage(void)
58: {
59:     int c, cnt, find;
60:     FILE *fin, *fout;
61:
62:     cnt =0;
63:
64:     if((fin = fopen(MODEMDEVICE, "rb")) == NULL) {
65:             printf("device open error!!¥n");
66:             exit(EXIT_FAILURE);
67:     };
68:
69:     if((fout = fopen("/root/image", "w+")) == NULL) {
70:             printf("file open error!!¥n");
71:             exit(EXIT_FAILURE);
72:     };
73:
74:     while(1) {              /* データスタート検出 */
75:             c = fgetc(fin);
76:             if ((unsigned char)c == 0xFF) {
77:                     break;
78:             }
79:     }
80:
81:     while(1) {              /* データ部分書き出し */
82:             c = fgetc(fin);
83:             if((unsigned char)c != 0xFF) {
84:                     find = 1;
85:                     fputc((unsigned char)c, fout);
86:                     cnt++;
87:             } else if(find == 1) {
88:                     break;
89:             }
90:     }
91:     fclose(fout);
92:     fclose(fin);
93:
94:     return cnt;
95: }
```

リスト５：RS-232C 受信プログラム（ビットずれ対策版）getbmp.c （続き）

```
 96:
 97:
 98: void makebmp(void)
 99: {
100:     FILE *fout;
101:     FILE *fin;
102:     int i, c;
103:     double u0,y0,v0,y1,r,g,b;
104:     unsigned char header[] = {0x42,0x4D,0x36,0xA4,0x04,0,0,0,0,0,0x36,0,0,0};
105:     unsigned char infohead[] = {0x28,0,0,0,0x60,1,0,0,0xE0,0xFE,0xFF,0xFF,
106:     1,0,0x18,0,0,0,0,0,0,0xA4,4,0,0x13,0x0B,0,0,0x13,0x0B,0,0,0,0,0,0,0,0,0,0};
107:
108:     if((fout = fopen ("/root/image.bmp","wb+")) == NULL) {
109:             printf("file open error!!¥n");
110:             exit(EXIT_FAILURE);
111:     };
112:
113:     // BMP ヘッダ書き出し
114:     for(i=0;i<14;i++) {
115:             fputc(header[i],fout);
116:     }
117:     for(i=0;i<40;i++) {
118:             fputc(infohead[i],fout);
119:     }
120:
121:     if((fin = fopen ("/root/image.dat","rb")) == NULL) {
122:             perror(NULL);
123:             printf("data file open error!!¥n");
124:             exit(EXIT_FAILURE);
125:     };
126:
127:     for(i=1;i<=50688;i++) {           /* (352x288x2)/4=50688 回ループ */
128:             u0 = fgetc(fin);
129:             y0 = fgetc(fin);
130:             v0 = fgetc(fin);
131:             y1 = fgetc(fin);
132:
133:             r = (256*y0 +                  359*(v0 - 128))/256;
134:             g = (256*y0 -  88*(u0 - 128) - 183*(v0 - 128))/256;
135:             b = (256*y0 + 454*(u0 - 128)                  )/256;
136:
137:             if(255 < r) r = 255;
138:             if(r < 0) r = 0;
139:             if(255 < g) g = 255;
140:             if(g < 0) g = 0;
141:             if(255 < b) b = 255;
142:             if(b < 0) b = 0;
143:
144:             // BMP パレットは BGR の順
145:             // first pixel
146:             fputc((unsigned char)b, fout);
147:             fputc((unsigned char)g, fout);
148:             fputc((unsigned char)r, fout);
149:
150:             r = (256*y1 +                  359*(v0 - 128))/256;
151:             g = (256*y1 -  88*(u0 - 128) - 183*(v0 - 128))/256;
152:             b = (256*y1 + 454*(u0 - 128)                  )/256;
```

リスト５：RS-232C 受信プログラム（ビットずれ対策版）getbmp.c （続き）

```
153:
154:            if(255 < r) r = 255;
155:            if(r < 0) r = 0;
156:            if(255 < g) g = 255;
157:            if(g < 0) g = 0;
158:            if(255 < b) b = 255;
159:            if(b < 0) b = 0;
160:
161:            // second pixel
162:            fputc((unsigned char)b, fout);
163:            fputc((unsigned char)g, fout);
164:            fputc((unsigned char)r, fout);
165:    }
166:
167:    fclose(fin);
168:    fclose(fout);
169: }
```

以上の getbpm プログラムを実行すると、そのときにシリアルカメラから送られてきているデータを使って、ビットマップ画像を作成することが出来る。

## ６－５　画像取得の自動化

シリアルカメラからの画像を受信しビットマップ画像を作成するプログラムが完成したので、これを一定時間間隔で実行することを考える。

Linux には cron とよばれる、ジョブを自動実行するためのサービスが存在する。この設定ファイル（crontab）に時間とコマンドを記載しておくと定期的に実行することが出来る。設定ファイルの書き込みには、crontab -e というコマンドを使用する。コマンドを実行すると vi エディタが立ち上がる。次のように記述する。

```
MAILTO=“”

*/1 * * * *    /root/getbmp
```

リスト 5 のプログラムを/root/getbmp としておいておく場合を仮定している。
これにより、コマンド getbmp が１分ごとに実行することができる。

１行目の MAILTO 文は、実行をメールで知らせるユーザを記述するのだが、メール通知はいらないので“”として記述しておく。

# 第７章　Web サーバでの表示するための設定

## ７－１　設定

Web サーバによってホームページとして表示する場合は､デフォルトのホームページのファイルのあるところに、index.html というファイルに表示するために必要な文を HTML で記入しておいて置けばよい。MS104-SH4 の場合、/var/www/index.html が相当する。index.html と同じ場所に画像ファイルを置く。

index.html 記載例

```
<HTML>
<HEADER>
<TITLE>PHOTO BY SERIAL CAMERA</TITLE>
</HEADER>
<BODY>
<IMG SRC="xxxx.xxx"> <!--ファイル名を記入する-->
</BODY>
</HTML>
```

以上で、Web アクセスしてきたときに撮影された画像が表示される。

## ７－２　Web ブラウザ，携帯電話での表示の注意点

Web ブラウザで､インターネット越しに作成した bmp ファイルを表示しようとしたが、Internet Explorer では表示されなかった。また、携帯電話でも同様に画像が表示されないのを確認した。JPEG 形式では正常に表示されるため、ビットマップ形式から JPEG 形式へ変換する必要がある。また、JPEG 形式のほうが画像サイズが小さいので、通信の面から考えると経済的である。

JPEG 形式に変換するためのコマンドがあるのでそれを利用する。

○ビットマップ形式から JPEG 形式への変換コマンド

```
cjpeg –quality 90 A.bmp > B.jpg
```

クオリティはデフォルトで 75 であるが、少々粗くなるので、90 を指定している。このコマンドは、ビットマップ画像の高さ指定の数値が負の値だとうまく動作しないので、リスト 5 の 105 行目、高さの値を 0xE0,0xFE,0xFF,0xFF ⇒ 0x20,1,0,0 としておく。(この場合、ビットマップの画像は上下が逆になっている)

○JPEG 画像の上下を反転させるコマンド

```
jpegtran –flip vertical B.jpg > C.jpg
```

C.jpg を index.html で表示するファイルに指定すればよい。

この２つのコマンドをリスト 6 に入れることによって、自動的にカメラから画像を取得し、ホームページを通して画像を監視できるシステムが完成する。

リスト 6 に次のように変更を加える。

・105 行目

```
unsigned char infohead[] = {0x28,0,0,0,0x60,1,0,0,0xE0,0xFE,0xFF,0xFF,
```

⇩

```
unsigned char infohead[] = {0x28,0,0,0,0x60,1,0,0,0x20,1,0,0,
```

・24 行目の makebmp()の次の行に次の４行を追加する。

```
    // bmp -> jpeg
    system("cjpeg -quality 90 /root/image.bmp > /root/tmp.jpg");
    // 上下反転
    system("jpegtran -flip vertical /root/tmp.jpg > /var/www/photo.jpg");
```

また、これに伴い index.html での画像ファイルの指定は、photo.jpg とする。

## 第８章　結論

シリアルカメラからの画像データを高速シリアル通信で受信し、Web ページで画像を確認するシステムが完成した。インターネットにつながっていて、ルーターの設定を適切に行えば、次の画像のように、携帯電話で画像が確認できる。


図 20：携帯電話での確認画面

当初、通信速度の関係から PIC で読み出すなど無理かと思われたが、回路にも命令を少なくするように変更を加え、アセンブラソースを切り詰め、どうにか完成までたどり着くことが出来た。

## 第９章　応用アプリケーション例

受信側が Linux で動作しているため、プログラムを書けばいくらでも応用がきく。考えうるアプリケーションとしては、次のようなものが挙げられる。

① 火災通報システム
受信画像データの輝度信号分布を毎回記録しておき、大きく変化した場合に異常事態として携帯電話にメールで通報を行う。もちろんそのときの画像も確認可能。輝度分布の他に、MS104-SH4 の I/O ポートを利用して、温度センサが特定の温度以上になったら通報を行うことも可能。

② セキュリティ監視システムの応用
MS104-SH4 の I/O ポート経由で、赤外線焦電センサの信号を入力し、人体検知と同時に撮影した画像を電子メールで送付することが可能。

## 第１０章　参考にさせていただいたサイト、お世話になったサイト

・MS104-SH4
「株式会社アルファプロジェクト」
http://www.apnet.co.jp/
・各種パーツ購入
「地質時代 Treva、DMR-C1 用コネクタ」
http://blog.micwire.net/article/23534805.html
「マルツパーツ館 WebShop」
https://www.marutsu.co.jp/
「秋月電子通商」
http://akizukidenshi.com/
・シリアルカメラ DMR-C1 の解析
http://www.mujirushi.org/ews/dmrc1/index.html
・CMOS カメラユニット「Treva」の解析
http://www.paken.org/aaf/treva/
・YUV フォーマット
http://vision.kuee.kyoto-u.ac.jp/~hiroaki/firewire/yuv.html
http://www.nnet.ne.jp/~hi6/lab/pixel/
・ビットマップ画像フォーマット
http://www.umekkii.jp/data/computer/file_format/bitmap.cgi
・Serial－Programming－HOWTO [JF]
http://www.linux.or.jp/JF/JFdocs/Serial-Programming-HOWTO.html