ＡＶＲ２３１３－テキスト画面表示プログラム　オリジナル版　マニュアル

－目次－

１．機能概要

・ＵＡＲＴ信号から文字キャラクターのＡＳＣＩＩコードを入力し、ＮＴＳＣコンポジット信号（の類似信号）を生成、出力する。

・文字１個のサイズは５ドット×６ドットのオリジナルフォント。白黒のみの表示。アルファベット、英数字、その他記号を合わせて７０種のビットマップをプログラムＲＯＭ上に格納。

・横13文字×縦7行の合計９１文字を表示。1文字表示するごとにカーソルが自動的に移動。画面右端まで書くとカーソルが自動的に1行下の左端に移動する。一番下の行の右端まで書くと、カーソルが自動的に左上に移動する。

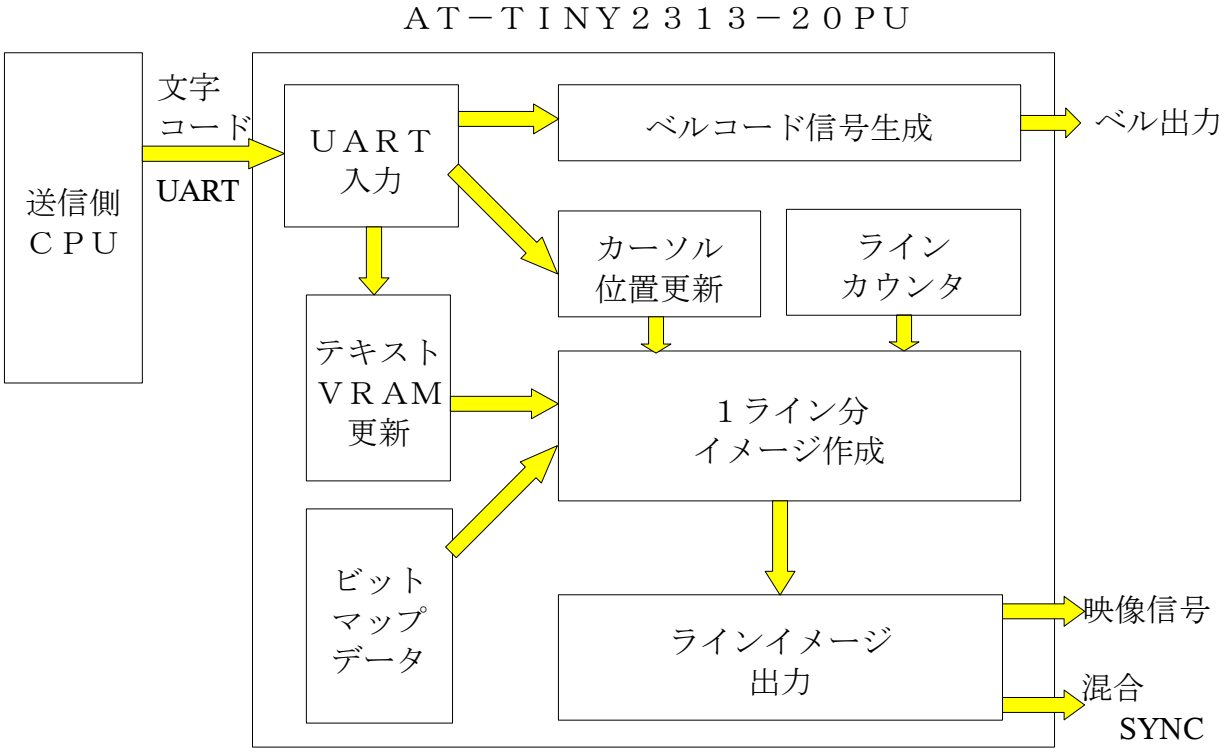
・座標系は、左上（０，０）～右下（１２，６）の範囲。カーソル座標を直接指定してカーソル移動することが可能。

・全画面クリア、ベルコードの特殊コードを搭載。

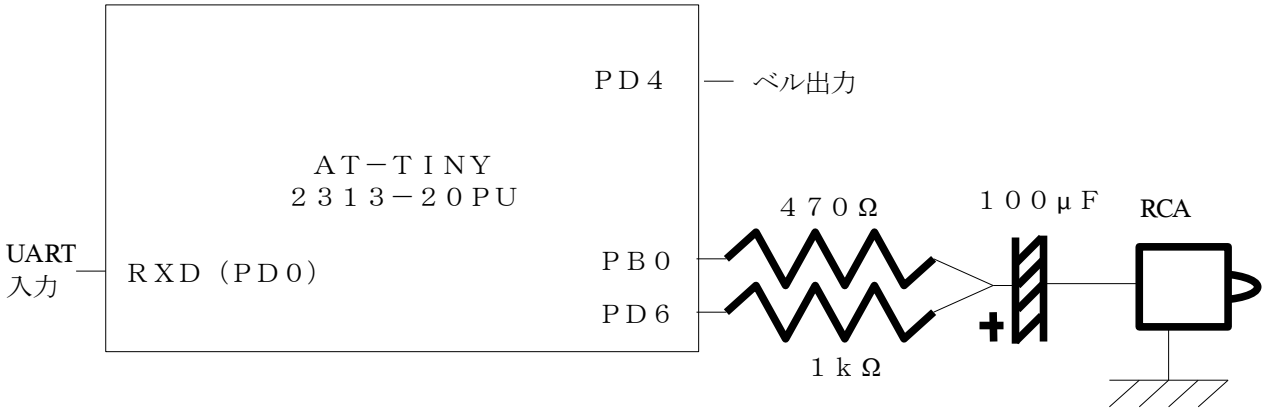
・カーソル位置にカーソルは表示されない。

２．動作環境

（１）ブロック図



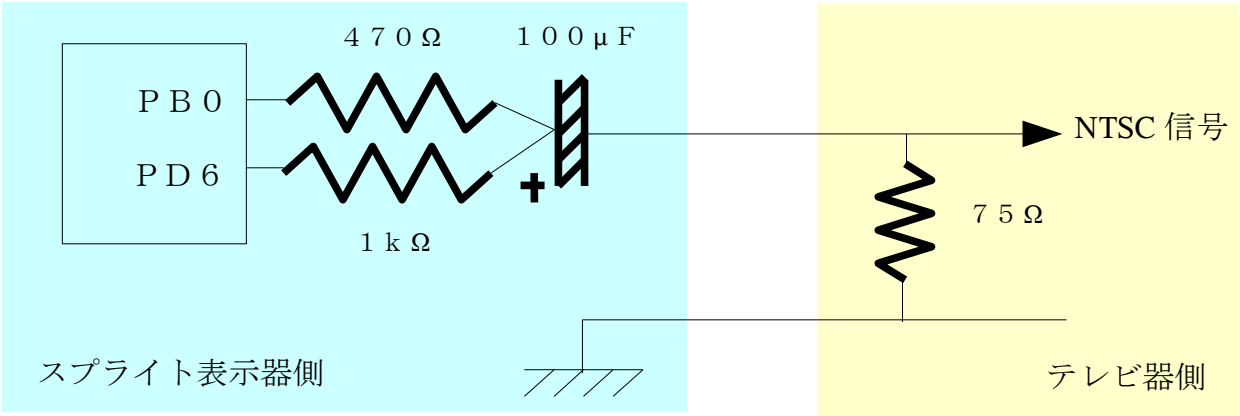
（２）使用回路図



・ＵＡＲＴ制御信号入力　：　ＲＸＤ端子（ＰＤ０）
・映像信号　：　ＰＢ０端子
・合成ＳＹＮＣ　：　ＰＤ６端子
・ベル出力　：　ＰＤ４端子

（注１）ポートＢのＰＢ０以外の端子は、常時ノイズが出力されているためＮＣとする。
（注２）ＲＣＡ端子と抵抗の間にカップリングコンデンサを置く。

（３）抵抗値について



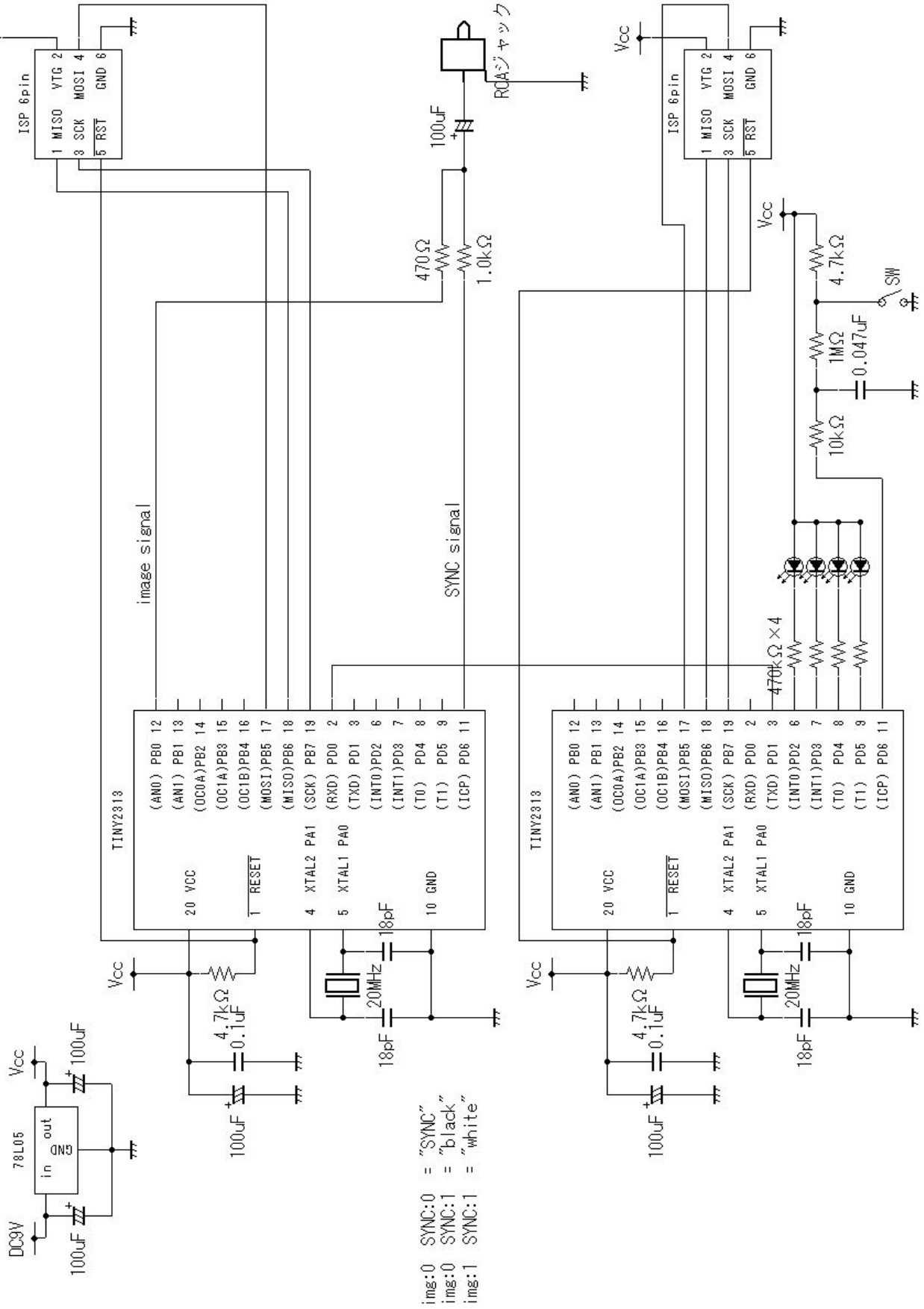
・ＳＭＰＴＥ－１７０Ｍで規定された電圧（ＳＹＮＣ信号から白までが１Ｖｐｐで、ＳＹＮＣ信号から黒までが０．３Ｖｐｐ）に合わせるために、５Ｖの電圧を抵抗で分圧して実現する。その際の抵抗は上図のとおり４７０Ω、１ｋΩ、７５Ωの３つを利用する。出力電圧は以下のとおり。（合成抵抗値と電圧の計算式は割愛する）

	ＳＹＮＣ	*灰色*	*黒*	*白*
ＰＢ０出力（映像）	０	１	０	１
ＰＤ６出力（SYNC）	０	０	１	１
出力電圧	０Ｖ	０．６４６Ｖ	０．３Ｖ	０．９４９Ｖ

（４）それ以外の外部回路

・動作クロック：２０ＭＨｚ。セラロック発信子、クリスタル発振子、クリスタルオシレータのいずれかを使用。その際、フューズビットでＣＬＫＤＩＶ８を指定してはならない。（クロックプリスケールレジスタが１／１倍でなければならない）
・ＵＡＲＴ接続端子は５Ｖｐｐ（通常のＴＴＬレベルの電圧）とする。（ＲＳ２３２Ｃレベルの電圧ではない）

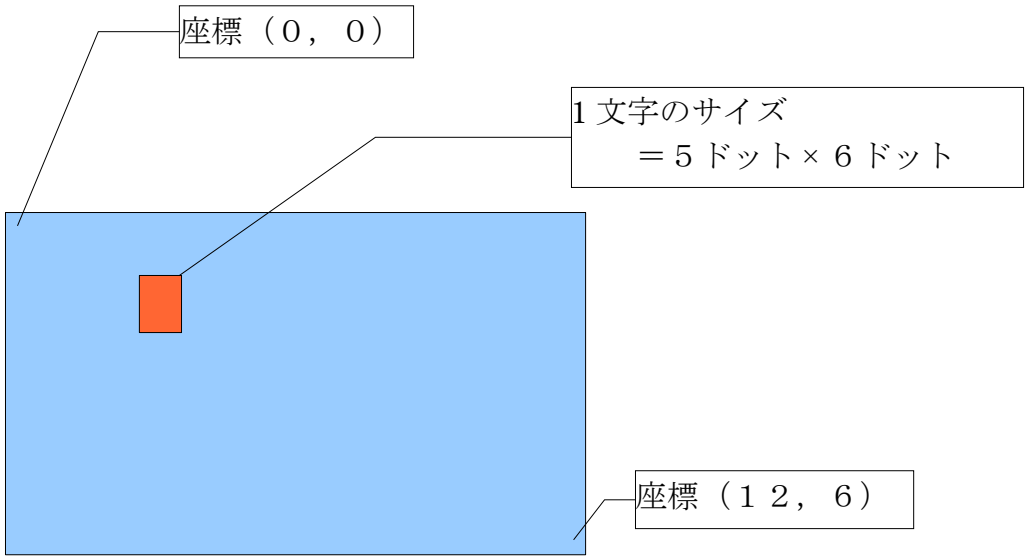
（5）サンプル回路図


Vcc
ISP 6pin
1 MISO
VTG 2
3 SCK
MOSI 4
5 RST
GND 6
RCAジャック
100uF
470Ω
1.0kΩ
4.7kΩ
SW
1MΩ
0.047uF
10kΩ
470kΩ×4
image signal
SYNC signal
TINY2313
20 VCC
1 RESET
4 XTAL2
5 XTAL1
10 GND
(AN0) PB0 12
(AN1) PB1 13
(OC0A)PB2 14
(OC1A)PB3 15
(OC1B)PB4 16
(MOSI)PB5 17
(MISO)PB6 18
(SCK) PB7 19
(RXD) PD0 2
(TXD) PD1 3
(INT0)PD2 6
(INT1)PD3 7
(T0) PD4 8
(T1) PD5 9
(ICP) PD6 11
PA1
PA0
18pF
20MHz
4.7kΩ
0.1uF
78L05
in
out
GND
DC9V
img:0 SYNC:0 = "SYNC"
img:0 SYNC:1 = "black"
img:1 SYNC:1 = "white"

３．仕様

（１）表示可能領域

・文字が配置可能な座標は、左上（０，０）～右下（１２，６）の範囲。
・カーソルが画面右端にある時に表示可能文字コードを受信すると、その位置に文字を表示した後にカーソルは１行下の左端に移動する。
・最下行の最右端＝（１２，６）にカーソルがあるときに表示可能文字コードを受信すると、その位置に文字を表示した直後にカーソルが（０，０）に移動する。
・白黒表示のみ対応。
・カーソル位置にはカーソルは表示されない。



・文字１個のサイズは５ドット×６ドット。以下にそのサンプルとしてキャプチャー画面の部分拡大図を示す。（2 ラインで 1 ドットを表現している）

（２）使用可能文字種数

・アルファベット、英数字、その他記号を合わせて７０種のビットマップをプログラムＲＯＭ上に格納。（ただし、アルファベットの小文字フォントは搭載しておらず、大文字に変換して表示している。）
・それ以外に特殊文字コードが存在する。特殊文字コードについては（３）を参照。
・文字コード表に存在しないコードを受信しても無視される。（何も起こらない）
・文字コードの一覧については、項番６付録（１）キャラクターコード表を参照。

（３）特殊文字コード

・ベルコード（０ｘ０７）：ＰＤ４端子に約０．５秒間だけ５Ｖを出力する。（この０．５秒の間にさらにベルコードを受信した場合、２回目のベルコード受信から再度０．５秒をカウントしなおす。この際ＰＤ４端子の出力は途切れない。以降、０．５秒以内にベルコードを再受信するたびにカウンター残量は０．５秒に巻き戻される。）　なお、ベルコード出力だけではカーソル位置は影響を受けない。
・画面クリア（０ｘ０Ｃ）：画面全体を空白コードで埋める。
・Ｘ軸指定コード（０ｘ８０～８Ｃ）：カーソルのＸ軸を直接指定する。
・Ｙ軸指定コード（０ｘ９０～９６）：カーソルのＹ軸を直接指定する。

（４）ＵＡＲＴ接続仕様

・１２００ｂｐｓ、データ長８ビット、ノンパリ、ストップビット１。
・連続してデータを受信することが可能。（送信側ＣＰＵで１バイト送信する毎にウェイトを入れる必要なし）
・ＵＡＲＴの初期化処理が完了してデータ受信が可能になるまでに４０．１５μ秒を要する。（フューズビットで指定するスタートアップタイムは含まない。）

４．使用方法

（１）初期化

・電源ＯＮ時もしくはリセット時に、テキストＶＲＡＭに立ち上げメッセージを自動的に表示する。カーソル位置は（０，０）に初期化される。適宜、ユーザープログラム側から画面クリアコード（０ｘ０Ｃ）を送信して立ち上げメッセージを消去する。
・また、立ち上げ時の回路上のノイズがＵＡＲＴに乗って不要データが取り込まれる恐れがある場合は、ユーザープログラムの先頭で画面クリアを行うことによってクリアすること。
・テキスト画面表示器表示器がＵＡＲＴの初期化を終えるまでの必要時間は、シミュレータによると４０．１５秒（フューズビットで指定するスタートアップタイムは含まない）。よってスタートアップタイム＋４０．１５μ秒以内にＵＡＲＴの受信データが届くと正常に受信することが出来ないため、起動直後は送信側ＣＰＵからデータを送信する前に適宜タイミング待ちを行うこと。

（２）文字表示、およびカーソル制御

・表示可能文字のコードをＵＡＲＴにて送ることで、カーソル位置にその文字コードが表示される。表示可能な文字コードの一覧は項番６付録（１）を参照。アルファベットの小文字については、内部で大文字に変換されて表示されることに留意。
・カーソルを特定位置に移動するには、Ｘ軸、Ｙ軸を指定することで実現する。Ｘ軸の指定は０ｘ８０～０ｘ８Ｃ、Ｙ軸の指定は０ｘ９０～０ｘ９６を送信することで実現する。
・全画面をクリアするには０ｘ０Ｃを、ベル出力は０ｘ０７を送信する。
・以下にアセンブラ用のサンプルを示す。

```
   ;******************
   ;*****  定義類  *****
   ;******************

   ;*****  定義ファイル  *****

.include "tn2313def.inc"


   ;*****  レジスタ  *****

.def data1 = R16                ;超局所用途の汎用レジスタ
.def out_data = R17                     ;UART 出力用データ（サブ引渡し用）
.def l_count1 = R18                     ;汎用カウンタ１
.def l_count2 = R19                     ;汎用カウンタ２
.def l_count3 = R20                     ;汎用カウンタ３



   ;*****  定数  *****
```

```
   ;************************
   ;*****  リセット処理  *****
   ;************************

.org 0x00                           ;リセット時のスタート番地
   rjmp    RESET            ;リセットがかかったときに0番地からRESETに相対ジャンプ

RESET:

   ;************************
   ;*****  初期化処理  *****
   ;************************

   ;*****  スタックポインタ初期設定  *****

   ldi             data1,RAMEND            ;data1にスタックポインタのアドレスをセット
   out             SPL,data1                       ;スタックポインタにdata1をセット

   ;*****  初期化処理呼び出し  *****

   rcall   INITIALIZE



MAIN_LOOP:

   ;************************
   ;*****  メインループ  *****
   ;************************


   ;*****  送信処理  *****


   ldi             out_data,0x31   ;1
   rcall   PUSH_AND_OUT

   ldi             out_data,0x41   ;A
   rcall   PUSH_AND_OUT

   ldi             out_data,0x26   ;&
   rcall   PUSH_AND_OUT

   ldi             out_data,0x0c   ;cls
   rcall   PUSH_AND_OUT

   ldi             out_data,0x20   ;space
   rcall   PUSH_AND_OUT


   ldi             l_count1,0
MA_LOOP1:
   mov             out_data,l_count1
   rcall   PUSH_AND_OUT
   inc             l_count1
   cpi             l_count1,0x80
   brne    MA_LOOP1                ;0から0x7fまで続けて出力
```

```
   ldi             out_data,0x32   ;2
   rcall   PUSH_AND_OUT

   ldi             out_data,0x42   ;B
   rcall   PUSH_AND_OUT

   ldi             out_data,0x24   ;$
   rcall   PUSH_AND_OUT


   ldi             out_data,0x0c   ;cls
   rcall   PUSH_AND_OUT


   ldi             l_count1,0
MA_LOOP2:
   ldi             out_data,0x30
   rcall   UART_OUT
   inc             l_count1
   cpi             l_count1,91
   brne    MA_LOOP2                ;91 文字連続して数字の 0 を出力

   ldi             l_count1,0
MA_LOOP3:
   ldi             out_data,0x31
   rcall   UART_OUT
   inc             l_count1
   cpi             l_count1,91
   brne    MA_LOOP3                ;91 文字連続して数字の 1 を出力

   ldi             l_count1,0
MA_LOOP4:
   ldi             out_data,0x32
   rcall   UART_OUT
   inc             l_count1
   cpi             l_count1,91
   brne    MA_LOOP4                ;91 文字連続して数字の 2 を出力

   ldi             l_count1,0
MA_LOOP5:
   ldi             out_data,0x33
   rcall   UART_OUT
   inc             l_count1
   cpi             l_count1,91
   brne    MA_LOOP5                ;91 文字連続して数字の 3 を出力

   ldi             l_count1,0
MA_LOOP6:
   ldi             out_data,0x34
   rcall   UART_OUT
   inc             l_count1
   cpi             l_count1,91
   brne    MA_LOOP6                ;91 文字連続して数字の 4 を出力


   ldi             out_data,0x8B   ;Ｘ軸を 11 に
   rcall   PUSH_AND_OUT

   ldi             out_data,0x5A   ;Z
   rcall   PUSH_AND_OUT
```

```
ldi		out_data,0x94	;Ｙ軸を４に
rcall	PUSH_AND_OUT

ldi		out_data,0x52	;R
rcall	PUSH_AND_OUT


ldi		out_data,0x0d	;crlf
rcall	PUSH_AND_OUT

ldi		out_data,0x41	;a
rcall	PUSH_AND_OUT


ldi		out_data,0x0d	;crlf
rcall	PUSH_AND_OUT

ldi		out_data,0x41	;a
rcall	PUSH_AND_OUT


ldi		out_data,0x0d	;crlf
rcall	PUSH_AND_OUT

ldi		out_data,0x41	;a
rcall	PUSH_AND_OUT


ldi		out_data,0x0d	;crlf
rcall	PUSH_AND_OUT

ldi		out_data,0x41	;a
rcall	PUSH_AND_OUT


ldi		out_data,0x0d	;crlf
rcall	PUSH_AND_OUT

ldi		out_data,0x41	;a
rcall	PUSH_AND_OUT


ldi		out_data,0x80	;Ｘ軸を０に
rcall	UART_OUT
ldi		out_data,0x90	;Ｙ軸を０に
rcall	UART_OUT
ldi		out_data,0x2A	;*
rcall	UART_OUT
rcall	BUTTON_PUSH

ldi		out_data,0x81	;Ｘ軸を１に
rcall	UART_OUT
ldi		out_data,0x91	;Ｙ軸を１に
rcall	UART_OUT
ldi		out_data,0x2A	;*
rcall	UART_OUT
rcall	BUTTON_PUSH

ldi		out_data,0x82	;Ｘ軸を２に
rcall	UART_OUT
ldi		out_data,0x92	;Ｙ軸を２に
rcall	UART_OUT
```

```
    ldi             out_data,0x2A   ;*
    rcall   UART_OUT
    rcall   BUTTON_PUSH

    ldi             out_data,0x83   ;X軸を3に
    rcall   UART_OUT
    ldi             out_data,0x93   ;Y軸を3に
    rcall   UART_OUT
    ldi             out_data,0x2A   ;*
    rcall   UART_OUT
    rcall   BUTTON_PUSH

    ldi             out_data,0x84   ;X軸を4に
    rcall   UART_OUT
    ldi             out_data,0x94   ;Y軸を4に
    rcall   UART_OUT
    ldi             out_data,0x2A   ;*
    rcall   UART_OUT
    rcall   BUTTON_PUSH

    ldi             out_data,0x85   ;X軸を5に
    rcall   UART_OUT
    ldi             out_data,0x95   ;Y軸を5に
    rcall   UART_OUT
    ldi             out_data,0x2A   ;*
    rcall   UART_OUT
    rcall   BUTTON_PUSH

    ldi             out_data,0x86   ;X軸を6に
    rcall   UART_OUT
    ldi             out_data,0x96   ;Y軸を6に
    rcall   UART_OUT
    ldi             out_data,0x2A   ;*
    rcall   UART_OUT
    rcall   BUTTON_PUSH


    ldi             out_data,0x0c   ;cls
    rcall   PUSH_AND_OUT


    ldi             l_count1,0
MA_LOOP7:
    mov             out_data,l_count1
    rcall   UART_OUT
    inc             l_count1
    cpi             l_count1,0x80
    brne    MA_LOOP7                ;0から0x7fまで続けて出力
    rcall   BUTTON_PUSH

    ldi             out_data,0x0c   ;cls
    rcall   PUSH_AND_OUT



    rjmp    MAIN_LOOP               ;LOOPに相対ジャンプして無限ループ
```

```
   ;************************
   ;*****  サブルーチン  *****
   ;************************


   ;*****                  *****
   ;*****  ＜初期化処理＞  *****
   ;*****                  *****
INITIALIZE:

   ;*****  入出力ポートの設定  *****

   ldi             data1,0b10111111        ;data1を第６ビット以外立てる
   out             DDRD,data1                      ;ＰＤ６以外すべて出力にセット

   ldi             data1,0b11111111        ;data1をすべて立てる
   out             DDRB,data1                      ;ＰＢをすべて出力にセット


   ;*****  初期値の設定  *****




   ;*****  RS232C  初期化  *****

   ldi             data1,0
   out             UCSRB,data1                     ;まず、一旦ディゼーブルに設定
   out             UCSRA,data1                     ;すべてクリア

   ldi             data1,((1<<UCSZ1)|(1<<UCSZ0))
   out             UCSRC,data1                     ;8ビット、ノンパリ、ストップ１

   ldi             data1,0x03
   out             UBRRH,data1                     ;上位４ビット
   ldi             data1,0xfc
   out             UBRRL,data1                     ;下位８ビット
                                                           ;UBRR = 0x3f7 = 1015 (=1230BAUD @20MHz)
                                                           ;UBRR = 0x411 = 1041 (=1200BAUD @20MHz)

   ldi             data1,(1<<TXEN)
   out             UCSRB,data1                     ;設定完了後に、受信イネーブルを設定



   ;*****  立ち上げまでにタイムラグを作る  *****

   ldi             l_count1,255
IN_LOOP1:
   ldi             l_count2,255
IN_LOOP2:
   nop
   nop
   nop
   nop
   nop

   dec             l_count2
   cpi             l_count2,0
```

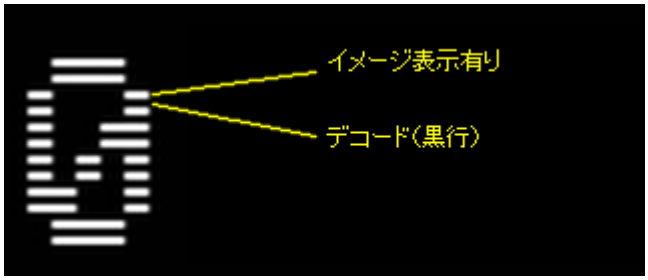
```
   brne    IN_LOOP2

   dec             l_count1
   cpi             l_count1,0
   brne    IN_LOOP1                        ;1/100程度のタイムラグ



   ret                                                     ;初期化終わり



   ;*****                                  *****
   ;*****  ＜キー押下＆ｏｆｆチェック＞  *****
   ;*****                                  *****
BUTTON_PUSH:
                                                           ;ボタンが押下されるまで待ち、
                                                           ;その後離されるまで待つ

   ;*****  〈step1〉  *****

BP_LOOP1:
   in              data1,PIND
   andi    data1,0b01000000        ;6ビット目が立っているかチェック
   brne    BP_LOOP1                        ;ボタンが押されてなければ繰り返す
                                                           ;ボタンは負理論：押されなければ１が入力

   ;*****  〈step2〉  *****

   sbi             PORTD,2

BP_LOOP2:
   in              data1,PIND
   andi    data1,0b01000000        ;6ビット目が立っているかチェック
   breq    BP_LOOP2                        ;ボタンが押されていれば繰り返す
                                                           ;ボタンは負理論：押されていれば０が入力

   cbi             PORTD,2

   ret                                                     ;戻る



   ;*****                      *****
   ;*****  ＜UART出力サブ＞  *****
   ;*****                      *****
UART_OUT:
                                                           ;UDREが立つまで待ってからデータを出力

UO_LOOP1:
   in              data1,UCSRA                     ;UCSRAを読み込む
   andi    data1,(1<<UDRE)         ;UDREが立っているかチェック
   breq    UO_LOOP1                        ;立つまで繰り返す

   out             UDR,out_data            ;データを出力


   ret                                                     ;戻る
```

```
   ;*****                    *****
   ;*****  ＜ボタン＋出力＞  *****
   ;*****                    *****
PUSH_AND_OUT:
                                                    ;ボタン押下を待ってからデータ出力

   rcall   BUTTON_PUSH                  ;ボタンチェック　呼び出し
   rcall   UART_OUT                     ;データ出力　呼び出し


   ret                                              ;戻る
```

５．注意点など

（１）表示方式・信号について（ＮＴＳＣとの差異など）

・当表示器にて出力するビデオ信号は、ＮＴＳＣ信号（ＳＭＰＴＥ１７０Ｍ）に類似する信号となっているが、以下の点で異なるので注意を要する。（ビデオ表示機器、録画機器によっては表示が正常に行われなかったり、全く表示されなかったり、場合によっては機器にダメージを与える恐れがあるかもしれないことを注意）
・座標データとビットマップデータから画面表示のイメージに変換するためのデコード処理には現実問題かなりの時間を要する。加えて、ＡＴ－ＴＩＮＹ２３１３はＳＲＡＭが１２８バイトと小さいため、１画面全体のビットイメージをＳＲＡＭに展開することはできない。現実解として、１ライン分だけのビットイメージをＳＲＡＭ上に生成（デコード処理）し、そのイメージを表示する、ということをライン数分繰り返すことによって画面全体の表示を行っている。
この１行分のイメージを作成する処理時間を確保するために、走査線を２本１組にして１ラインはデコード（この間は黒行の表示）、１ラインはイメージ表示という具合に処理を行っている。以下に実際のキャプチャー画面を示す。
（なお黒行についても他の行と同じように水平同期信号は出力されている。）



（２）インターレースについて

・ＳＭＰＴＥ１７０Ｍでは、１フィールドは２６２．５本の走査線、２フィールドで１フレームのインターレース表示を行うと定義されているが、当表示器においては１フィールドを２６２本の走査線で構成するため、インターレース表示が行われない。一般に市販されているテレビ受像機やビデオ機器等では問題なく表示可能であることをおおよそ確認しているが、機器によっては表示などに支障が起こる可能性がある。

（３）カラー信号について

・当表示器ではカラーバースト信号等のカラー画像生成の信号が含まれておらず、表示は常にモノクロのみとなる。

（４）信号のタイミングについて

・全般的にＳＭＰＴＥ１７０Ｍに近いタイミングで信号を生成しているが、回路に組み込む発振器の精度や、ソフトウェアで擬似的に信号を生成している都合、厳密には一致していないという点と、処理の都合で部分的に信号のタイミングがず

れている部分が存在する。およそ以下の点で信号のタイミングにズレがあることを認識。

＜信号のズレその１：走査線１本の長さについて＞

走査線１本あたり６３．４０μ秒で表示しており、厳密なＳＭＰＴＥ１７０Ｍとは若干のズレがある。

＜信号のズレその２：走査線間のズレについて＞

走査線と走査線の間に一部ベルコードの出力時間計測等の追加処理を行っているところなどがあり、他の走査線より１～２クロック分ほど水平同期期間が長いところが生じる。一般的なテレビ表示器では、水平同期期間が延びた場合その走査線の１本～数本下側において表示位置のズレが生じることが多い（このズレは数本内で収束される）。これを加味して、このような追加処理の下側には表示物が存在しない（黒行）ように設計してあり、基本的には表示内容に悪影響が及ばないように配慮している。ただし、画像表示器によっては表示がズレる可能性もあると考えられるので留意。

（５）出力電圧について

・出力されるコンポジット信号は、ＳＭＰＴＥ１７０Ｍに相当するよう１Ｖｐｐのうち同期信号が０．３Ｖｐｐ、映像信号分が０．７Ｖｐｐとなっている。ただし、既成品の抵抗による電圧の分圧なので、厳密な電圧値とは若干誤差があることに留意。

６．付録

（１）キャラクターコード表

・以下にキャラクターコード表を示す。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	-	sp	0	@	P	‘	p	x0	y0	-	-	-	-	-	-
1	-	-	!	1	A	Q	a	q	x1	y1	-	-	-	-	-	-
2	-	-	”	2	B	R	b	r	x2	y2	-	-	-	-	-	-
3	-	-	#	3	C	S	c	s	x3	y3	-	-	-	-	-	-
4	-	-	$	4	D	T	d	t	x4	y4	-	-	-	-	-	-
5	-	-	%	5	E	U	e	u	x5	y5	-	-	-	-	-	-
6	-	-	&	6	F	V	f	v	x6	y6	-	-	-	-	-	-
7	BL	-	’	7	G	W	g	w	x7	-	-	-	-	-	-	-
8	-	-	(	8	H	X	h	x	x8	-	-	-	-	-	-	-
9	-	-	)	9	I	Y	i	y	x9	-	-	-	-	-	-	-
A	-	-	*	:	J	Z	j	z	x10	-	-	-	-	-	-	-
B	-	-	+	;	K	[	k	{	x11	-	-	-	-	-	-	-
C	CL	-	,	<	L	¥	l	|	x12	-	-	-	-	-	-	-
D	-	-	−	=	M	]	m	}	-	-	-	-	-	-	-	-
E	-	-	.	>	N	^	n	→	-	-	-	-	-	-	-	-
F	-	-	/	?	O		o	←	-	-	-	-	-	-	-	-

：表示可能文字（ｓｐはスペース）
：制御文字
：表示可能文字（大文字に変換して表示）