試作検討 2B

RFID モジュール・マイコン間通信

創造設計第二 TA：張 辰樹・金 賢鍾

平成 18 年度 10 月 23・26 日

試作検討 2 の目的

今回の試作検討では、RFID モジュールの使い方、マイコン間の通信の仕方を学ぶことを目的とする。本日はまず、RFID モジュールを用いた RFID タグへの読み書きを行う。RFID モジュールはマイコン・PC・その他の機器と非同期式シリアル通信を行うことが出来る。これを使って RFID モジュールとマイコンの間の通信を行い、RFID タグの読み書きを行う。
次に 2 台のマイコンの間で通信を行う。マイコン間の通信にも非同期式シリアル通信を用いる。2 台のマイコンを用いて、互いの状態を通信する。

1. 通信方式

ここでは、コンピュータ間の通信に使われている方式を紹介する。

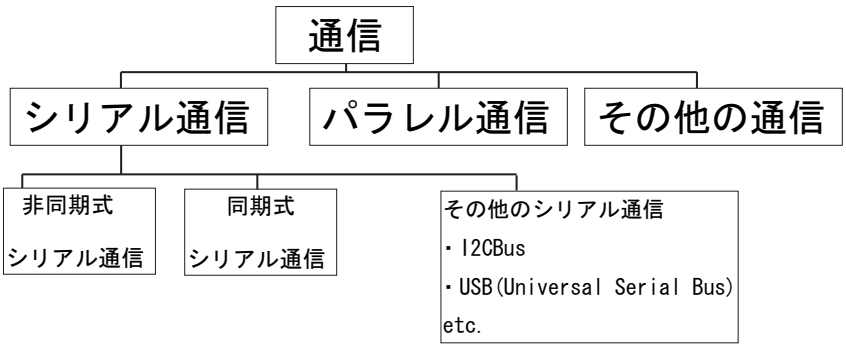


Fig. 1: 通信

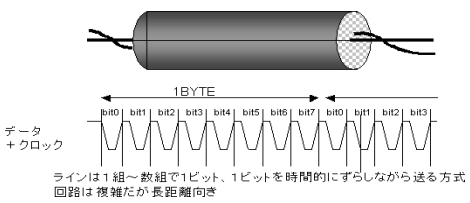


Fig. 2: シリアル通信

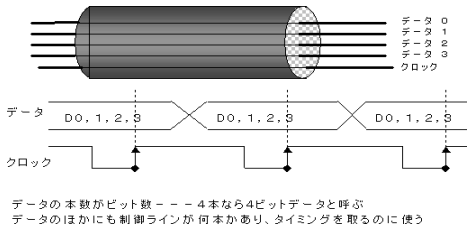


Fig. 3: パラレル通信

1.1　シリアル通信

Fig. 1 にあるように、通信は大きく分けてシリアル通信とパラレル通信に分かれる。

シリアル通信は基本的に送受信に各 1 本のデータ線を使い、全てのビットを順番にデータ線に流すことにより通信が行われる。このため、TxD,RxD の最低 2 本の線を繋げば通信できる (**Fig.** 2)。

これに対して、パラレル通信では、送信したいビットの数だけ通信線を用意し、一度に全ビットを通信する方式となる (**Fig.** 3)。

1.2　非同期式シリアル通信 (UART)

シリアル通信には、さらに非同期式・同期式などの通信方式がある。

同期式のシリアル通信は、互いに共通のクロックを持ち、このクロックにあわせて通信を行う。このため、RxD や TxD 以外にもクロック線が必要になるなど面倒な部分が多いが、速い速度での通信を行うことが出来る。

非同期式のシリアル通信では、通信を開始する前に、転送レートをお互いに設定しておき、その転送レートに従ってデータを転送する。共通のクロックは必要ない。この方式ではあまり早い通信を行うことは出来ないが、簡単であるため広く利用されている。最も簡単な場合 RxD(受信)、TxD(送信) の 2 本の信号線があれば実現できる。

今回の RFID モジュールとの通信では RxD、TxD に加え 5[V]、GND の合計 4 つの信号線を使う。また、2 台のマイコン間の通信では RxD、TxD に加え、GND の合計 3 つの信号線を使う。

2.　OAKS16-mini のシリアル通信用ポート

創造設計第二で使用するマイコン OAKS16-mini は、UART0、UART1、UART2 の 3 個のシリアル通信用のポートを搭載している。このうち UART1 はデバッグや焼き込みなどの際のホスト PC との通信に使われている。UART0 は、マイコンの 5[V] 系を RS232C 電圧レベルに変換するバッファや、正論理と負論理を入れ替えるインバータを持った RS2332C ドライバ (**Fig.** 4) を通った信号線が来ている。このため、このドライバを搭載していない RFID モジュールとの通信には向いていない。従って、今回の試作検討では RFID モジュールとの通信を UART2 を使って行い、マイコン間通信は UART0 を使って行う。

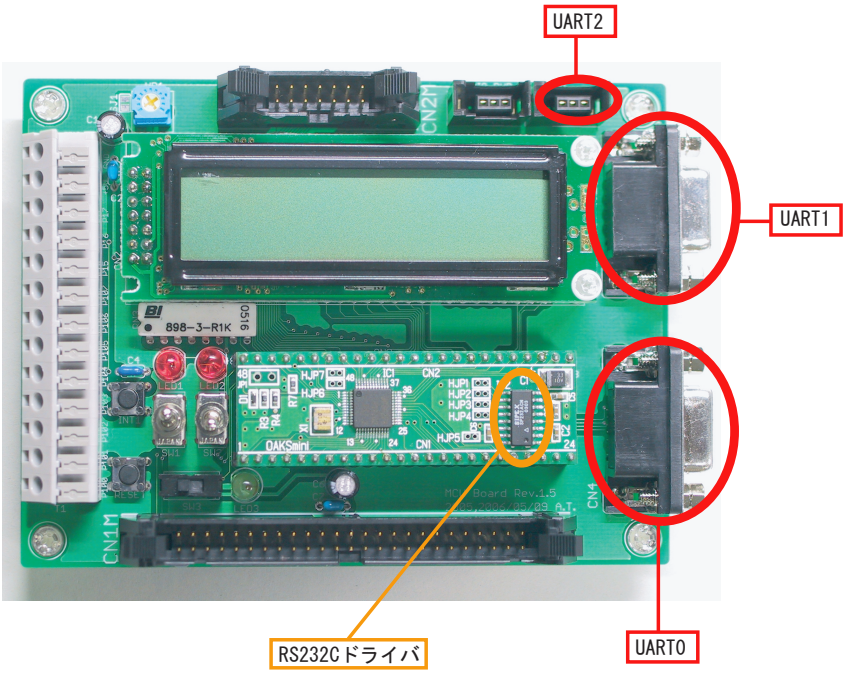


Fig. 4: MCU ボード

3. RFID モジュールと RFID タグ

3.1 RFID とは

JR の Suica などの非接触 IC カードは所定の場所に近づくだけで，情報のやり取りができます．これらは，図 5 のようなものを，カードに内蔵することにより実現しているシステムです．IC チップは，大きなコイルアンテナを用いて外部からの電磁波を変換し電源としています．このコイルアンテナを使用して，外部と通信を行います．さらに，IC チップは不揮発性のメモリを内蔵しており，外部からデータを読んだり，書き込んだりできるようになっています．

多くの場合，このチップとアンテナをシール状にしたり，プラスチックのカードの中に内蔵したりして使用します．これを IC タグ（または RFID タグ）と呼びます．

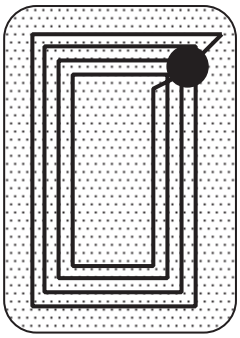


Fig. 5: RFID タグ

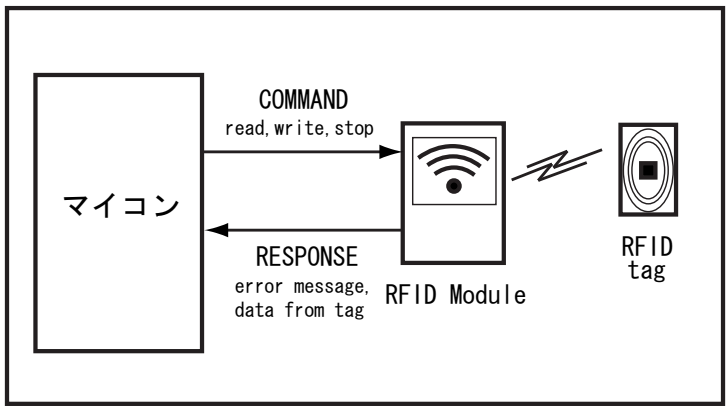


Fig. 6: RFID システムの構成及び関係

3.2 マイコン - RFID モジュール - タグの関係

今回使う RFID システムを構成するマイコン，RFID モジュール，そしてタグの関係を図 6 で示してあります．まずマイコンからタグに必要な情報を読むか，または書くかのコマンドを RFID モジュールに送ります．そのコマンドを RFID モジュールが受信し，コマンドに応じてタグと情報のやり取りを行います．情報のやり取りが終わった後，RFID モジュールはタグとのやり取りがうまくできたかについてマイコン側に報告（送受信時のエラー発生有無，及びタグから読み出した情報）し，一通りの仕事が終了します．

⚠ 注意	マイコンからコマンドを出してからその結果がマイコンに戻るまで時間がかかるので，リアルタイムプログラムの構成としてはコマンド関数と受信データ取る関数を別々に設定すべき

3.3 RFID リーダライタモジュール V720S-HMC73

競技時に貸与されるオムロン社製 RFID リーダライタモジュール V720S-HMC73（以下，V720）には様々な機能がサポートされ．これらの機能を活用することで，より効率の良い V720 との通信や V720 を介したタグとの交信が実現できるようになります．また，V720 をタグ検出センサとして用いるなど，新たな活用法も見えてくるはずです．

3.3.1 交信モード

V720 では，様々な交信モードでタグとの交信を行うことができます．交信モードは，更新後のタグに対する処理の違いによりシングルモードと FIFO モードの 2 種類に，コマンドの処理手順，実行タイミングの違いによりトリガモード，オートモード，リピートモードの 3 種類に分けられ，これらの組み合わせにより計 6 種類（実際は FIFO 連続モードがあるので 7 種類）があります．（試作検討時に使用した RFID ライブラリでは，シングルトリガモードを使用.）以下に，各モードについて簡単に説明しておきます．なお，各モードの動作シー

ケンスについては V720 のデータシート（4-7，4-8）を参照して下さい．

- 交信後のタグに対する処理の違い
  1. **シングルモード（Single Mode）**
     一度交信（Read/Write コマンド処理）を終了したタグに対して特別な処理を施さない．タグとの交信時間は FIFO モードに比べて短い．
  2. **FIFO モード（FIFO Mode）**
     タグとの交信（Read/Write コマンド処理）後，無変調発振を継続することで一度交信を終了したタグにアクセス禁止処理を施す．アクセス禁止されたタグは，アンテナの交信範囲外に出ると（もしくは，Stop コマンドを受信して発振を停止した後）再び交信可能となる．（アクセス禁止中のタグは，モジュールからはタグ不在と見なされるので注意．）
- コマンドの処理手順，実行タイミングの違い
  1. **トリガモード（Trigger Mode）**
     Read/Write コマンド受信後，すぐにタグと交信してレスポンス（交信範囲内にタグがなければタグ不在エラー）を送信．レスポンス送信後，新たなコマンドを受信するまで待機．
  2. **オートモード（Auto Mode）**
     Read/Write コマンド受信後，タグが交信範囲内に入ってくるのを待ってタグと交信し，レスポンスを送信．送信後（タグを待っている状態では Stop コマンドを受信後），新たなコマンドを受信するまで待機．
  3. **リピートモード（Repeat Mode）**
     Read/Write コマンド受信後，タグが交信範囲内に入ってくるのを待ってタグと交信し，レスポンスを送信．送信後は再びタグの検出を開始し，Stop コマンドを受信するまで新たなコマンドは受け付けずに ↪ タグ検出 → 交信 → レスポンス送信 ↩ のサイクルを繰り返す．

トリガモードだけでは，複数の V720 を切り替えて使用しても結局はそれぞれを単独で使っていることになりますが，オートモード，リピートモードを用いれば，複数の V720 にコマンドを送信して並行して同時に使うことができるようになります．また，タグ検出時にレスポンスが返ってくるので，V720 をタグ検出センサとして使うこともできます．

3.3.2　シリアル通信速度

V720 では，上位制御ユニットとの通信速度（転送レート）を 9600bps/38400bps から選択することができます．通信速度を 38400bps とした場合には，9600bps の場合と比較して V720 側のコマンド受信時，もしくは上位制御ユニット側のレスポンス受信時にオーバーラン・エラーが若干発生しやすくなりますが，コマンド・レスポンスの通信時間は短縮されます．（V720 との通信速度なので，タグとの交信時間はシリアル通信速度とは関係ありません．交信時間を短縮するためにはシングルモードを用いること．）

V720 側の通信速度を変更するには，通信速度設定スイッチを変更（OFF：9600bps，ON：38400bps）する必要があります．詳しくはデータシート（2-2）を参照して下さい．

3.3.3　同バンクデータの一括読み書き

競技フィールドに貼付される RFID タグには，1024 バイトのメモリを持つフィリップス社製の IC チップ SL2-ICS20 が搭載されていますが，この IC チップのメモリマップにおける「ページ」と「バンク」という区分は，本来 V720 にとっての区分です．「ページ」とは V720 の最小アクセス単位 4 バイト（ASCII コード 4 文字分）であり，「バンク」とは同時にアクセス可能な最大ページ数 16 ページのことです．つまり，同バンクのデータであれば一度のコマンド処理で任意のページを一括で読み書きすることができます．図 7 でタグのメモリマップの構成と情報の領域の予定図を示しております．

3.4　V720S ライブラリ（ver1.0beta）

RFID ライブラリに加え，以上の V720 の未使用機能を中心に，新たな機能を追加した C 言語ライブラリが V720S ライブラリです．T:¥ss2¥LIB06¥V720S_LIB より ver1.0beta を入手することができます．

Bank	Page	Byte0	Byte1	Byte2	Byte3
00h	0h				
	1h				
	2h				
	3h				
	4h				
	5h				
	6h				
	7h		**Freespace**		
	8h				
	9h				
	Ah				
	Bh				
	Ch				
	Dh				
	Eh				
	Fh				

Bank	Page	Byte0	Byte1	Byte2	Byte3
01h	0h		**Position**		
	1h				
	2h				
	3h				
	4h				
	5h		**Unused**		
	6h				
	7h				
	8h				
	9h				
	Ah				
	Bh				

Read / Write Area
Read-only Area
Unused Area

Fig. 7: メモリマップの定義とアクセス規約 (予定)

3.4.1 ライブラリ構成

V720S ライブラリの構成は以下のようになっています．本ライブラリのインターフェース（3.4.2 節参照）を使用するためには，ユーザー側のアプリケーションプログラムで**公開ヘッダファイル v720s.h をインクルード**する必要があります．なお，UART2 ライブラリは，V720 を制御する際のハードウェア依存部分である OAKS16mini の UART2 [*1]（OAKS16mini のマニュアル第 13 章参照）を用いたシリアル通信のための関数群を提供するもので，V720S ライブラリ内で使われています．

ライブラリ構成	
v720s.c（V720S Library C Source File）	上位ライブラリ関数の定義
v720s.h（V720S Library Public Header File）	上位ライブラリ関数（公開部）に関連する定義・宣言
uart2.c（UART2 Library C Source File）	下位ライブラリ関数の定義
uart2.h（UART2 Library Private Header File）	下位ライブラリ関数（非公開）に関連する定義・宣言

ライブラリのヘッダファイルは通常，インターフェースを提供するためのもの（公開ヘッダファイル）と，ライブラリ自身を作るために必要なもの（非公開ヘッダファイル）とに分けられます．これは，バージョンアップ（インターフェース以外の中身の変更）の際のトラブルを避けるために，アプリケーションプログラムではライブラリの中身に依存した書き方をして欲しくないからです．今回の V720S ライブラリを使ったマシンのソフトウェア開発でも，今後のバージョンアップによるサポートを受けたい場合には，**公開ヘッダファイル v720s.h 以外に依存したコーディングはやめましょう**．（非公開とは「ユーザ」ではなく「アプリケーションプログラム」に対して非公開という意味です．きちんと中身を理解してライブラリを使えるように，他の非公開ヘッダファイルやライブラリのソースにも必ず一度は目を通しておいて下さい．）ただし，V720S ライブラリは完全な

[*1] **UART（Universal Asynchronous Receiver Transmitter）**シリアルポートなどに使われる通信回路で，送信時にパラレルなバイトデータをシリアルのビットストリームに変換，受信時にその逆に復元する

オープンソースであるため，自分たちで独自にカスタマイズしたい場合にはこの限りではありません．もちろん，ライブラリのソース自体を変更して使ってもらっても構いません．

3.4.2 インターフェース

V720S ライブラリでは，RFID リーダライタモジュール V720S-HMC73 を制御するための以下のライブラリ関数，グローバル変数，およびマクロ関数を提供します．

ライブラリ関数	
v720s_init	UART2 の初期化・V720 との通信テスト
v720s_putcmd_read	V720 への Read（タグデータ読みだし）コマンドの送信
v720s_putcmd_write	V720 への Write（タグデータ書き込み）コマンドの送信
v720s_putcmd_stop	V720 への Stop（実行中のコマンド停止，発振停止）コマンドの送信
v720s_getdata	V720 からのレスポンス（V720 の終了コードと読み出しデータ）受信

グローバル変数	
uart2_received	UART2 側へ V720 のレスポンス受信完了通知
v720s_busy	一通りの仕事が終わったことを通知

3.4.3 ライブラリ関数

各ライブラリ関数の形式，機能，および返値は以下のようになっています．

v720s_init	
形式	int v720s_init(void)
機能	UART2 初期化と V720 との通信テスト (TEST コマンド送信，レスポンス受信) を行う．
返値	正常処理時 0，それ以外の時エラーコード 999，998，110〜114 を返す．

v720s_putcmd_read	
形式	int v720s_putcmd_read(unsigned char *mode_type*, unsigned char *data_type*, unsigned char *pagenum_fs_st*, unsigned char *pagenum_fs_end*)
機能	*mode_type* の交信モードで *data_type* で指定されるタグデータを読み出す要求を V720 に出す．データにフリースペースが指定された時は *pagenum_fs_st* からの *pagenum_fs_end* ページまで読み出しを要求する．
返値	正常処理時 0，それ以外の時エラーコード 997〜993，991 を返す．

v720s_putcmd_write	
形式	int v720s_putcmd_write(unsigned char *mode_type*, unsigned int *pagenum_fs_st*, char* *data_write*)
機能	*mode_type* の交信モードでフリースペースのタグデータ領域に *data_write* で指定されるデータを *pagenum_fs_st* のページから書き始める要求を V720 に出す．タグデータにフリースペースが指定され，かつデータがページ単位に満たない時はデータの終端にはスペースが挿入される．
返値	正常処理時 0，それ以外の時エラーコード 997〜992 を返す．

v720s_putcmd_stop	
形式	int v720s_putcmd_stop(void)
機能	実行中のコマンド停止（発振状態の場合は発振停止）の要求を V720 に出す．
返値	正常処理時 0，それ以外の時エラーコード 997，996 を返す．

v720s_getdata	
形式	int v720s_getdata(char* *data_read*)
機能	V720 からのレスポンス（要求に対する終了コード，読み出しデータ）を共有メモリに格納する．読み出しデータは *data_read* で指定される char 型配列に文字データとして格納される．読み出しデータがない場合は *data_read* にはヌル文字が格納される．
返値	正常処理時は V720 の終了コード，それ以外の時エラーコード 995，110〜114 を返す．

上記ライブラリ関数に関するいくつかの注意事項をまとめておきます．

- v720s_init 関数は「OAKS16mini の初期化」関数です．「V720 の初期化」は接続さえきちんとしていれば必要ありません．そのため，**複数の V720 を使う場合も v720s_init 関数は一度だけ実行すれば十分**です．
- **v720s_putcmd_xxx 関数の返値は「V720S ライブラリのエラーコード」**で，単にライブラリ関数をその仕様通りに呼び出せているかを確認するためのものです．従って，この返値が正常処理を示す 0 であっても V720 がコマンド処理を完了したことにはなりません．V720 がコマンド処理を完了したかどうかを確認するためには，**v720s_getdata の返値のうちの「V720 の終了コード」**を確認して下さい．（v720s_getdata の返値には V720S，UART2 ライブラリのエラーコードも含まれています．）
- v720s_getdata 関数では，ポインタで指定された char 型配列に受信した UART1 の受信バッファデータをコピーします．この時，バッファ・オーバーフローを起こして他のメモリを破壊しないようにするためには，**char 型配列のサイズを 65（1 バンク分のデータ + ヌル文字）以上にしておく**必要があります．（1 バンク以下の読み出しの場合はそれ以下．）残念ながら C 言語ではパラメータとして渡された文字配列のサイズを呼ばれた関数側で求める方法はないので，ライブラリ内でのエラーチェックは不可能です．

3.4.4 グローバル変数

各グローバル変数の形式，初期値，および機能は以下のようになっています．

uart2_received	
形式	unsigned int uart2_received
初期値	0
機能	V720 からのレスポンス受信完了 (UART2 側) を通知する．UART2 の受信許可を出してから，受信が終了したときに 1 となり新たな受信を許可した時に 0 にセットされる．

v720s_busy	
形式	unsigned int v720s_busy
初期値	0
機能	コマンドを出してもいいことを通知する．V720S ライブラリ関数 v720s_getdata の呼出しにより前回受信したレスポンスが共有メモリに格納された時 0 でクリア，次のコマンドを出してからレスポンス受信完了時に 1 にセットされる．

上記グローバル変数の使い方について簡単にまとめておきます．

- 送信データは V720 との間の共通の転送レートで 1 バイト分ずつ V720 へシリアル送信されます．従って，コマンドは v720s_putcmd_xxx 関数の呼び出しから復帰した時に送信完了している訳ではありません．複数の V720 を同時に使う（オート，リピートモードでコマンドを送信して，レスポンスを受信する前に別の V720 に切り替えてコマンドを送信するなど）場合，**コマンド送信完了通知が届いてから切り替えを行う**と良いでしょう．
- v720s_getdata 関数は，V720 からのレスポンスが届いた時に呼び出さなければ意味がありません．**v720s_getdata 関数の実行タイミングはレスポンス受信通知より判断**して下さい．

3.4.5　ライブラリ関数引数

ライブラリ関数の引数 *mode_type*，*data_type*，*fs_pagenum* の指定方法を以下に示します.

交信モード *mode_type*	
V720S_MODE_ST	シングルトリガモード：Single Trigger
V720S_MODE_SA	シングルオートモード：Single Auto
V720S_MODE_SR	シングルリピートモード：Single Repeat（読みだしのみ可）
V720S_MODE_FT	FIFO トリガモード：FIFO Trigger
V720S_MODE_FA	FIFO オートモード：FIFO Auto
V720S_MODE_FR	FIFO リピートモード：FIFO Repeat

アクセスデータ *data_type*	
V720S_DATA_UID	シリアルナンバー：Unique Identification（読み出しのみ可）
V720S_DATA_FS	フリースペース：Freespace（読み書き可）
V720S_DATA_POS	位置座標：Position（読み出しのみ可）

フリースペース読み出し，書き込みページ指定 *pagenum_fs_st or _end*	
0〜15	フリースペースから読み出す時は最初と最後ページ（0〜15）を指定 書き込む時は書き込む初期ページ (0〜15) を指定

3.4.6　エラーコード一覧

V720S ライブラリ関数の返値である V720S，UART ライブラリのエラーコード，V720 の終了コードの一覧を以下に示します.（V720 の終了コードは一部のみなので，詳しくはデータシート（4-9）を参照すること.）

V720S ライブラリエラーコード（〜999）	
999	V720 と通信できません.
998	TEST コマンド実行後の V720 のレスポンスの終了コードが正しくありません.
997	UART の初期化（v720s_init 関数の呼び出し）が行なわれていません.
996	前回コマンドの送信が完了していません.（コマンド送信完了通知がまだ届いていません.）
995	V720 からのレスポンスは未受信です.（レスポンス受信完了通知がまだ届いていません.）
994	交信モード *mode_type* の指定が正しくありません.
993	アクセスするタグデータ *data_type* の指定が正しくありません.
992	書き込みデータの文字数が正しくありません.（フリースペース：最大 64 文字）
991	フリースペースの読み出しページ数 *fs_pagenum_st or _end* の指定が正しくありません.

UART ライブラリ受信エラーコード（100〜）	
102	UART 送受信中に受信バッファにバッファ・オーバーフローが発生しました.
111	UART 受信中にオーバーラン・エラーが発生しました.
112	UART 受信中にフレーミング・エラーが発生しました.
114	UART 受信中にパリティ・エラーが発生しました.

V720 終了コード（0〜）	
0	コマンドを正常に実行しました.
70	タグとの交信中にノイズ等の障害が発生し，正常に完了できませんでした.
72	コマンド処理時に交信範囲内にタグが存在しません.（Trigger mode だけ有効）

3.4.7　通信速度の変更方法

シリアル通信速度（転送レート）はデフォルトでは 9600bps となっています．転送レートを 38400bps に変更するためには以下を行なって下さい．

- V720 側の設定：V720 の通信速度設定スイッチを変更（3.3.2 節参照）
- OAKS16mini 側の設定：uart2.h の「#define UART2_BOWRATE 129」を「#define UART2_BOWRATE 31」に変更

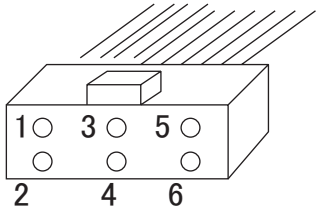
3.5　マイコンと RFID モジュールと接続

3.5.1　RFID コンポーネント

RFID リードライトモジュール V720S は，固定して使うことを目的として作られているため，ケーブルを着脱したり，競技ごとにはがしたりするような使い方は想定されていません．このような使い方をすると，コネクタ部分が破損するなど，耐久性に不安がでてきます．

そこで，これらの耐久性の向上と，保守性のため，V720S，ケースやケーブルを含めて，「RFID コンポーネント」とします．

RFID リードライトモジュールからの信号も，７番から 10 番までのピンの信号を省いています．信号と，コネクタ形状について，図 8 に示します．



Pin No.	記号	機能
1	5V	+5V電源
2	GND	グランド
3	RxD	シリアル入力
4	TxD	シリアル出力
5	GND	グランド
6	GND	グランド

Fig. 8: RFID からのケーブルのコネクタ

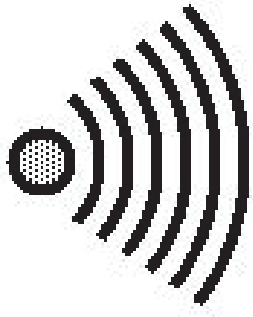


Fig. 9: アンテナ面を示すシール

RFID コンポーネントでは，図 9 のシールが張ってある方が，RFID リードライトモジュールのアンテナ面になっています。**(図 9 の丸が交信範囲の中心となります．)** タグには，シールの着いている面を向けるようにします．

RFID コンポーネントのマシンへの取り付けは，マジックテープにて行います．詳細については後日発表します．

3.5.2　マイコンボードとの接続

実際にロボットを作る際には，ユーザーボード上などに，外部回路を組んでください．RFID を切り替えるための推奨回路とフィールド上のタグの位置情報については後日発表します．

今回の試作検討では，外部回路は作成せず，一番上についているデバッグボードをはがして，SS2 ボードから直接ポートにつなぐことで通信を行います．

注意	SS2 ボード上の IC 類は非常に高価なので，静電気や不注意によるショートなどで破壊しないように気をつけてください

OAKS16mini では，P70 が UART2 の送信ポート，P71 が UART2 の受信ポートになっています．したがって，RFID モジュールの受信ポートと P70 を，RFID モジュールの送信ポートと P71 を接続します．この際，**V850/KG1 と RFID モジュールの送受信ポートをそれぞれクロスでつなぐことに気をつけて下さい．**また，TXD2 は N チャンネルオープンドレインなので **RFID モジュールの RXD と接続際にはプールアップ抵抗を付ける必要があります．**(RFID モジュールの通信方式は CMOS レベル)

図 10 に，SS2 ボードから 5V，GND，P70 および P71 をとる方法を示します．**電源と GND を間違えないように気をつけてください．**

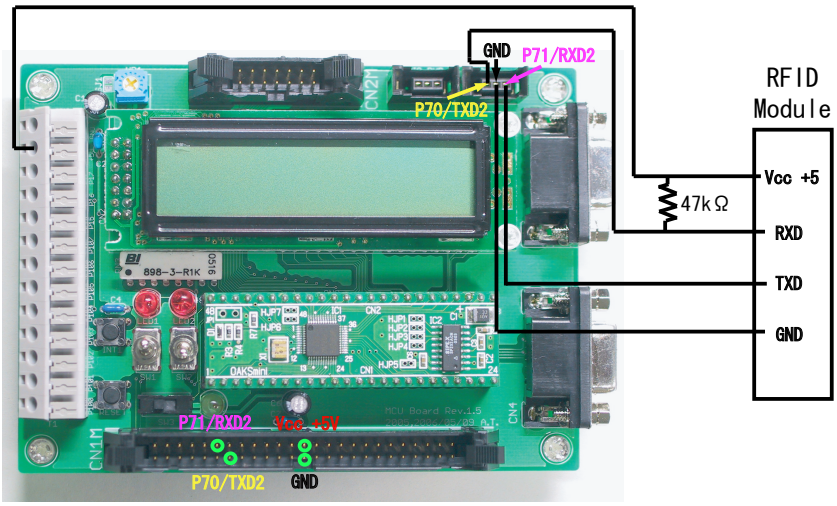


Fig. 10: SS2 ボードから 5V，GND，P70 および P71 をとる方法

3.6 課題（V720S と C 言語ライブラリの使い方）

3.6.1 実験準備

実験準備

1. T:¥ss2¥shisaku¥shisaku02¥rfid 以下にあるサンプルプログラムをコピーする．
2. main.c をエディタで開いてプログラム全体を確認した後，以下の点を修正する．
   - グロバールで定義された，フリースペース書き込み用の char 型ポインタの配列 message に，自分たちのオリジナルメッセージを書く．
3. TM でビルド後，図 10 を参考して V720S とマイコンを接続する．その後，デバッガ KD30 の起動して watch ウィンドウを起動．watch には，以下の message のグローバル変数を追加しておく．

3.6.2 実験手順

実験手順

1. 実験準備ができたらタグを机から 2cm 以上浮かせて（マイコンボードの箱の上に乗せるなど）してプログラムを実行し，タグを近づけた時に液晶の画面の数字が 0 になることを確認した上でタグとの交信範囲を測ってください．**初期化で失敗した際は V720S とマイコンの接続を再度確認すること．**この時，RFID コンポーネントの蓋をはずしておくと V720S のアンテナ位置，および V720 の LED の色から通信状態が分かってやりやすい．
   （蓋を開ける時は十分に注意し，この手順が終わったら確実に戻しておく）
2. トグルスイッチにより簡単にコマンドを変えることができます．CPU 隣にトグルスイッチが 2 つあり，0-3 までのコマンドが選択できます．2 個のトグルスイッチの状態が（液晶の部分を上として）
   - 上上→ mode=0 となりタグ固有のシリアルナンバー読み出し
   - 上下→ mode=1 となりフリースペースに message の内容書き込み
   - 下上→ mode=2 となりフリースペース読み出し
   - 下下→ mode=3 となり位置情報の読み出し

   となります．まずは読み取りコマンドを実行してもともとタグに入っている情報を読んだ後で，書き込みコマンドを実行して自分たちの情報を書き込み，その後, 情報が書き換わっているかを確認する．
3. 最後に V720S の交信モードを変えながら，それぞれの交信モードを熟知する．(Single & FIFO Tirgger, FIFO Repeat は重要)
   (ここで交信モードの選択として default で　V720S_MODE_ST → Single Trigger となっています．一通り 4 つのコマンドをやってみて，理解したら　V720S_MODE_ST → V720S_MODE_FT(FIFO Trigger) に変えて Single との違う点を調べてください．
   試作検討としては Single&FIFO Trigger モードだけの試しとなりますが，RFID モジュールを使う予定の班は必ず Repeat モード，特に V720S_MODE_FR(FIFO Repeat) モードはやってみてください．)
4. 一通り V720S の使い方を理解したらあらためてもう一度ソースを眺め，班のメンバーでプログラムの流れを確認する..
   - ライブラリ関数の返り値（send_error_code，receive_error_code）の意味
   - データ受信コマンド v720s_getdata(data) の送信タイミングと data に格納される中身
   - コマンドの送信タイミングの制御の仕方（送信許可フラグ send_flag の使用法）

追加
スイッチ文の中のコマンドは皆さんが RFID モジュールを使いやすくするために用意されたライブラリです．ライブラリを使いこなすなめには資料のライブラリの説明のところをしっかり読んでください．もし v720s.c の内容が分からない人は TA に質問するか，T:¥ss2¥data_sheet の中の v720 のデータシートファイルを読んでください．

4. マイコン ↔ マイコン間通信

シリアル通信についてのレジスタ設定についてはハードウェアマニュアルの 13 章 (特に 13.1 節と 13.3 節) を参照すること。

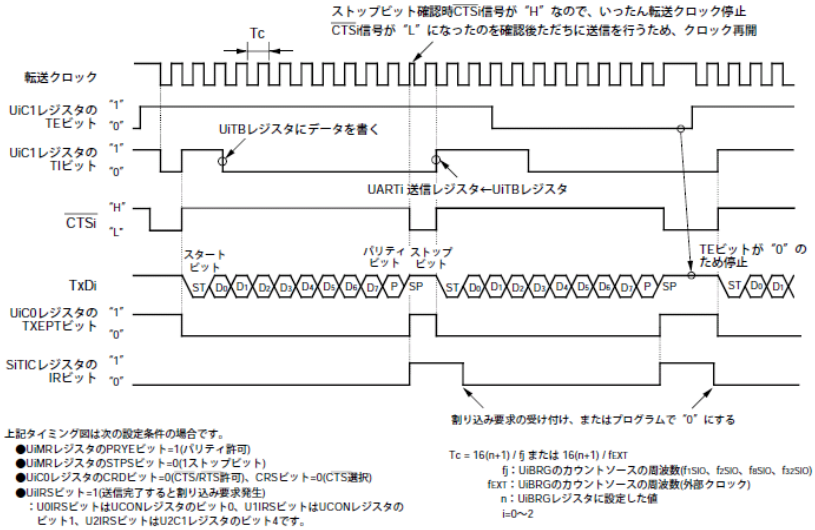


Fig. 11: シリアル通信のタイミングチャート (**今回利用する設定とは違うので注意!!**)

4.1 なぜマイコン間通信が必要か

今回の競技では、1 班につき複数台ロボットの製作・出場が認められている。複数台のロボットを動かすときにはマイコン間通信が必要になるかもしれない。

また、動かすロボットが 1 台であっても、マイコン 1 台のポート数で動かせるアクチュエータやセンサの数には厳しい制限がある。これをマイコン間通信を使って 1 台のロボットに 2 枚のマイコンを搭載してポートを増やし、多くのアクチュエータやセンサを使えるようにすることも目的となる。

4.2 マイコン間通信ライブラリ

マイコン間通信に利用できるライブラリを紹介する。下位関数を集めたファイルである uart0.c や uart0.h は基本的に RFID モジュール通信と同じものを使っている。

ここでは mcu_comunication.c と mcu_comunication.h に含まれている上位関数について説明する。

マイコン間通信初期化	
形式	int init_mcu_comu(void)
機能	UART0 関連レジスタの初期化処理を行い、関連する変数を初期化する。
返値	正常処理時 0，それ以外の時エラーコード 100～117, 210～230 を返す。

マイコン間通信・送信	
形式	int mcu_comu_tra_str(const unsigned char* send_mess);
機能	相手マイコンに対して文字列 send_mess にターミネータを付加したものを送信する。
返値	正常処理時 0，または一つ前の送信動作のエラーコード 100～117, 210～230 を返す。

マイコン間通信・受信	
形式	int mcu_comu_rec_str(unsigned char* receive_mess);
機能	相手のマイコンからターミネータが届いていたら、受信バッファにある文字列のターミネータをヌル文字に変換して receive_mess ポインタが指す場所に格納する。
返値	正常処理時 0，それ以外の時エラーコード 100〜117, 210〜230 を返す。

4.3 実験・文字列送受信

1. 1 班あたり、パソコンを 2 台とマイコンを 2 つずつ用意する。
2. 2 台のパソコンとそれぞれのマイコンを繋げ、マイコン同士は UART0 同士で接続する (**Fig.** 12)。このケーブルは 9pin シリアルのクロスケーブルである。このケーブルは競技会に向けて各班で製作することになる。

> **注意** 2 台の間で RxD と TxD が入れ替わることになるので、**クロスケーブル**を使う。ケーブルを製作する際には接続が**クロス**になるように注意する。

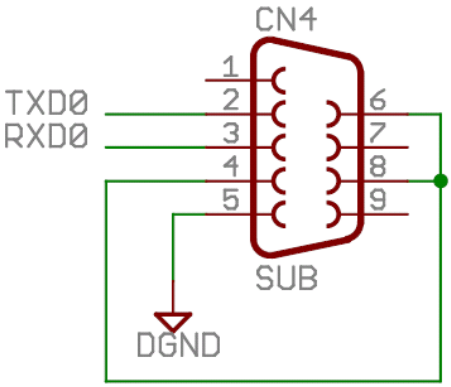


Fig. 12: UART0/D-sub コネクタ

3. サンプルプログラムを各々のディレクトリにコピーする。(各班で 2 つ。a と b)
   サンプルプログラムの場所は
   `T:\SS2\shisaku\shisaku02\mcu_comunication\`
   の中にある。
4. ビルドしてマイコンにダウンロードして実行。
5. 正常に通信できていれば、リモコンボードの赤いボタンを押したときに、そのマイコンの LCD 下段に ON、他方のマイコンの LCD 上段に ON と表示される。また赤いボタンを押していないときには各マイコンの同じ場所に OFF と表示されるはずである。

解説

このサンプルプログラムではマイコンの動作は以下のようになっている。

- 50[ms] に 1 回、赤いボタンが押されているかをサンプリングする (ポーリング)。
- 50[ms] に 1 回、赤いボタンの状態を LCD 下段に表示する。
- 50[ms] に 1 回、赤いボタンが押されていたら文字列” ON”を、押されていない場合は文字列”OFF”を UART0 を通して送信する。
- UART0 に終端文字 (<CR>) を受信したら送られてきたメッセージを LCD 上段に表示する。

6. ときどきエラーコードが表示されることを確認する。(IE は初期化エラー・RE は受信エラー・TE は送信エラー)

エラーコード一覧

- 101：初期化時に送受信可能文字数を越えたバッファが用意された。
- 102：バッファオーバーフロー (バッファを越える文字数の送受信が行われた)
- 111：バッファオーバーラン (受信文字を読み出す前に次の受信が行われた)
- 112：フレーミングエラー (ストップビット不整合)
- 113：バッファオーバーラン + フレームエラー
- 114：パリティエラー
- 115：バッファオーバーラン + パリティエラー
- 116：フレーミングエラー + パリティエラー
- 117：オーバーランエラー + フレーミングエラー + パリティエラー
- 210：初期化不良のため送受信不能
- 220：1 つ前の文字列送信が未完了
- 230：文字列受信が未完了

4.4 ソースコードをいじってみよう

1. 相手マイコンに送る文字列を変えてみよう。
   ただし半角英数字のみ。実際には送受信バッファに用意した分だけ文字数を特に制約なく通信できるが、今回は LCD 表示の制約により送受信できる文字数は 10 文字までで、なおかつ赤いスイッチの状態が ON の時と OFF の時の文字数が一致するように書いてほしい。文字数をあわせにくいときは少ない方の文字数の左側にスペースをつめて文字数が同じになるように調節すると良い。
2. 通信速度を下げてみよう
   ・uart0.h 内の`#define UART0_BOWRATE` の値を大きくすると通信速度は遅くなる。この値を 255 に設定してみよう
   ・0〜255 まで設定可能
   ・255 に設定した時よりもさらに通信レートを遅くしたい場合は uart0.c 内にある u0c0 レジスタの内容を書き換える
   ・2 台のマイコンの値を合わせる必要がある
   ・通信ビットレート [bps] = 20[MHz] / 16 / (`UART0_BOWRATE` + 1)
   ・エラーの出方に変化はあるか？

> **警告** .h ファイル (ヘッダーファイル) を変更したときは、**リビルド**を行う必要がある。普通のビルドを行うと.h ファイルの変更は反映されない。

4.5 考察

次のことを考えてみる。TA を呼んで説明する。

- エラーの原因は主としてなにか。これを回避したりこのエラーと上手に付き合うためにはどのような方法が考えられるか。
- 各マイコンが持つレジスタの種類とその内容の両方を送受信したいときはどのように送信すればいいか。直接レジスタの内容の 16 進数を 1 バイトとして送受信すれば良いか。
- 実際の競技に使うときはどのような通信を用いれば良いか。
- マイコンが 2 台ではなく 3 台以上になる場合はどうすれば良いか。

4.6 補足

試作検討では利用していない機能などについて補足する。

4.6.1 フロー制御

今回の試作検討では用いていないが、RTS、CTS を用いたフロー制御を行うことが出来る。これは、さらに 2 つのポートを用いて、受信レジスタがいっぱいの時に相手のマイコンに送信停止を要求し、受信レジスタが空になったら送信再開を要求するものである。これによりバッファーオーバーランエラーが起こらなくなる。

フロー制御については実際にデモストレーションをやってみる。
フロー制御をやってみたい場合はスタッフに相談すると良い。

4.6.2　ストップビット

U0MR レジスタの STPS ビットを変更することによりストップビットの数を 1 個と 2 個に変更できる。これは、今回は同じ仕様のマイコン同士の通信であるため関係ない (2 台のマイコンを同じ設定にする必要はある) が、他の機器と接続する場合などには適宜設定する。

4.6.3　パリティビット

パリティビットを用いることによって、通信の失敗を検出できる。パリティビットとは、送信したデータ中の”H”の個数が奇数か偶数かを表すビットで、データに続いて送受信される。このパリティビットと、データの内容を照合して合致しないときはエラービットを立てるものである。これにより、通信エラーを検出し、再送信を要求することも出来る。パリティビットの送受信を変更するには U0MR レジスタの PRY ビット、PRYE ビットを変更する。また、パリティエラーが起こったときは受信側の U0RB レジスタの PER ビットが”H”となる。これらの設定も必ず通信を行うマイコン同士で同じ設定を用いる必要がある。なお、マイコン ↔ マイコン間通信や、RFID モジュールとの通信には偶数パリティが利用されている。

4.6.4　シリアルポートの変更

何らかの理由でマイコン ↔ マイコン間通信を UART0 以外のポートで行いたいときはスタッフと相談すると良い。

4.6.5　UART2 を用いた特殊通信

UART2 には特殊通信機能が搭載されており、I^2C Bus モードでの通信や 3 台以上のマイコンの通信が行える。詳しくはハードウェアマニュアル 13.4 節や 13.5 節を参照。

4.7　競技会などで使うために

見てきたように、シリアル通信は全くエラーを起こさないとは言い切れない。そこで、このマイコン間通信が黒ヤギさんと白ヤギさんにならないようにするための工夫が必要である。

実際にロボットコンテストで使う際にはレジスタの中身や、相手マイコンへの命令を送受信することになると考えられる。この際に、その通信内容は、

通信の種類 + データ

にすると良い。通信の種類とはその通信が何を表しているかを示すものであり、例えば「この通信はこちらのマイコンの p1 レジスタの中身を示しているものだ。」とか、「相手マイコンのモータに対する命令を示しているものだ。」とかを先頭につけることで通信の信頼性が増すと考えられる。データには実際のレジスタの中身の値を書けば良い。

さらに、マイコン A がマイコン B に対して何かを送信し、その返事をマイコン B がマイコン A に送信するという問答方式と、マイコン A はマイコン B が必要とする全ての情報を繰り返し流し続け、マイコン B はマイコン A が必要とする情報を繰り返し流し続けるという垂れ流し方式が考えられる。前者の方はシンプルだがひとつの通信エラーでその後の問答が全てダメになる可能性をはらんでいる。その点、後者は一つの通信エラーが起きても比較的被害が少ないはずである。

また、シリアル通信では基本的に ASCII コードしか送ることが出来ない。そこで ASCII コードを使ってどのような通信が出来るかを考える必要もある。

例えば (これはあくまで例でこれを用いないといけないわけではない)、上記で挙げた通信の種類を表す通信種部のコードを 3 桁の数字として、さらにデータ部は 3 桁の数字として、それを ASCII 文字列に変換して送ることが出来る。以下にこの方法を用いた場合の、「こちらのマイコンの P1 レジスタの中身は 0xA9 だ」と送りたい場合の通信例を示してみる。

1. P1 レジスタの中身を送ることを示す通信の種類コードを 111 と仮定し、P1 レジスタの中身が 0xA9 だと仮定すると送信するメッセージ (send_mess) は次のようにして得ることが出来る。
   `sprintf(send_mess, "%ld", (unsigned long)111*1000 + p1);`
2. こうすると send_mess には**文字列**”111169”が入る。この文字列を相手マイコンに送信する。
3. この**文字列**を受信側マイコンでは receive_mess というポインタの指す領域に受け取る。
4. 受信側マイコンで long 型の receive_code を用意して
   `receive_code=atol(receive_mess);`
   (atol() は ASCII コードの数字をを long 型に変換する。int 型に変換する場合は atoi() を使う。共に stdlib.h に定義されている。)
   とすることで、long 型の receive_code に 111169 という**数字**を受け取ることが出来る。
5. さらに受信側マイコンでは int 型の receive_cmd(通信種部を格納) と receive_data(データ部を格納) を用意し、
   `receive_cmd = (receive_code - (receive_code % 1000)) / 1000;//`**つまり** `receive_cmd = 111`
   `receive_data = receive_code % 1000;//`**つまり** `receive_data = 169 = 0xA9`
   とすることで、通信を通信種部とデータ部に分離することが出来る。

このような処理をする関数を作成したものを
`T:\SS2\shisaku\shisaku02\omake\`
に用意しているので使ってみると便利かもしれない。

4.8 最後に重要なこと・シリアル通信速度とマイコン処理速度の差

【重要】シリアル通信速度とマイコンの処理速度には大きな差がある。これをしっかり頭に入れておかないと、オーバーランエラーの原因となる。たとえば、プログラム中で while 文を回している中で、毎回送信を行うような事をしようとすると、前の通信が終わっていないのに次の通信を始めようとするようなことになる。逆に毎回読み出しを行おうとすると、通信が終わってないのにメッセージを取り出すことになり不完全なメッセージとなる。従って、常に通信速度とマイコンの処理の速度の差がどの程度あるかを意識してプログラムを書くようにすることが重要である。