デジタル入出力システム用サンプルソフトウェア ユーザーガイド

1 概 要

本ソフトウェアは、図 1 のハード構成の各装置に組み込むことにより簡易的な DI、DO システムとして動作します。

ソフトウェアは、FPGA 用(system verilog)、Raspberry Pi 用(python3)のいずれもソースコードの形で提供します。ライセンスは、Apache License Version 2.0 としていますので、改変して生成物を配布する場合でもソースコードの公開義務はありません。

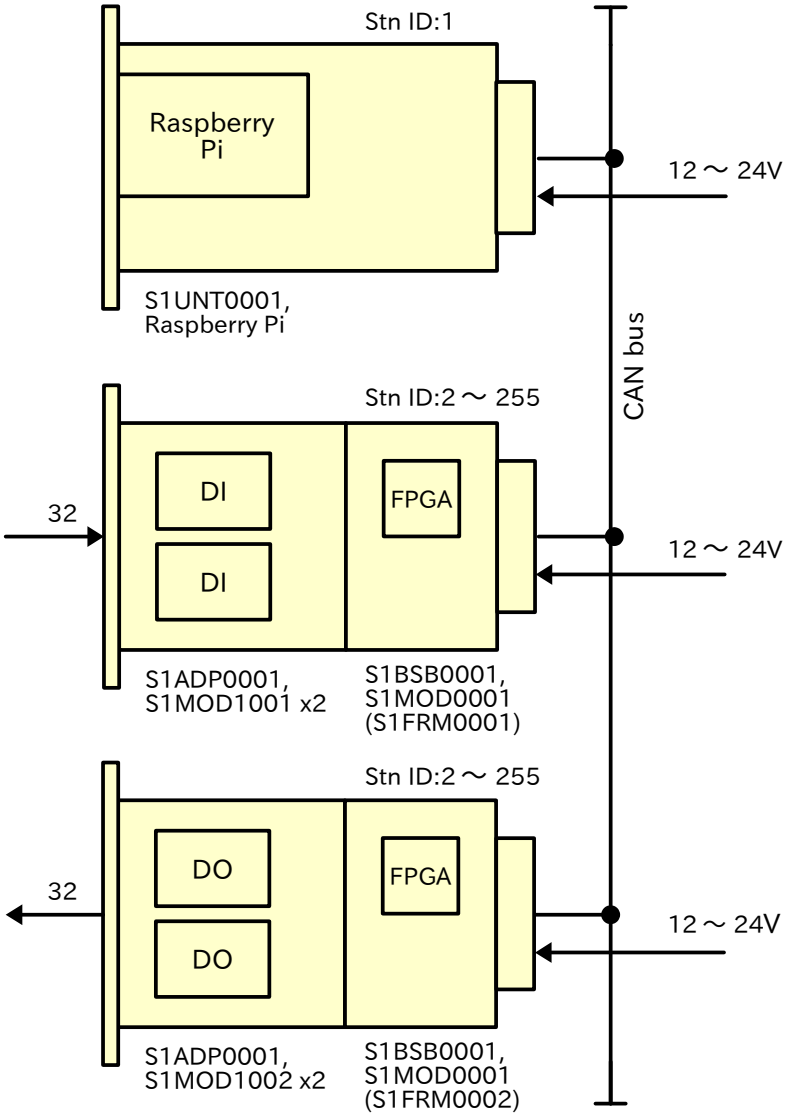


図 1 ハードウェアの構成

2 FPGA ファームウェア

2.1 セットアップ

Windows PC へのセットアップを例に説明します。開発ツールは Quartus Prime 18.1.0 Lite Edition を使用しています。

（1）製品情報ページから S1DIDOExampleFPGAFirmware.zip をダウンロードします。
（2）任意の場所に解凍します。
（3）プロジェクトフォルダ内のソースフォルダは解凍した時点では作成されていません。２種類のファームウェアでソースツリーを共通にしているためです。
実体は S1SampleFirmware\verilog ですので、プロジェトフォルダ内にリンクを作成します。
（コピーでも動作します。）

```
S1DIDOExampleFPGAFirmware
  ├──S1FRM0001_SCE_3V
  │      └──verilog（リンクを作成)
  ├──S1FRM0002_SCE_3V
  │        └──verilog（リンクを作成)
  └──S1SampleFirmware
          └──verilog（リンクのソース）
```

付録１、２にファームウェアのブロック図を添付しています。ソースコード上のモジュール名などを併記していますので、ソースコード内のテキスト検索などにご利用ください。

2.2 ビットファイルの生成

開発ツールの操作方法については、開発ツールのヘルプなどを参照願います。開発ツール、プロジェクトファイルが正しく配置されていれば、 Processing -> Start Compilation の操作でビットファイルが生成されます。

2.3 テスト

簡易的なテストコードを S1SampleFirmware\verilog\test に配置しています。ひとつのテストコードでひとつの結果を判定して、passed, failed を含む文字列を出力します。

（1）S1SampleFirmware\test_work でコマンドプロンプトを開きます。
（2）do_test.bat 内の 開発ツールの path 設定を環境に合わせて修正します。

(3) 以下のコマンドでテストを実行します。処理の詳細は、テキストエディタでバッチファイルを確認してください。

check.bat　　：各テストの実行結果を１行ずつ出力します。実行するテストを指定しない場合はTest＊にマッチする全てのフォルダでテストが実行されます。

check_fail.bat　：check.bat とほぼ同じ処理ですが、結果が failed のテストのみ表示されます。

(4) GUI でシミュレーションを行いたい場合は、test_work から個別のフォルダへ移動して

```
test_work\Test00x_xxxx> test.bat GUI
```

を実行すれば GUI モードで起動します。

3 Python アプリケーション

3.1 セットアップ

Raspberry Pi へのセットアップを例に説明します。python はバージョン 3.6 以降を使用しています。Windows でも同様に設定できますが CAN インターフェースの実体が無いため、CAN ライブラリのインスタンスを生成する部分では仮想インターフェースを使用する必要があります。

(1) サンプルアプリケーションで使用しているライブラリをインストールします。 pip3 がインストールされていない場合は、まず pip3 をインストールします。

```
$ sudo apt install python3-pip
```

(2) CAN 通信のライブラリをインストールします。

```
$ pip3 install python-can
```

(3) 製品情報ページから S1DIDOExamplePython3App.tgz をダウンロードします。

(4) Raspberry Pi に転送して任意の場所に解凍します。

(5) S1SampleApp/src へ移動します。プログラムは下記のように実行します。

```
$ python3 {アプリケーション名} {option}
```

動作の内容については「4.2.2 アプリケーションの動作」を参照してください 。コマンドの書式については、「付録 3 python アプリケーションの help メッセージ」をご覧ください。

3.2 テスト

簡易的なテストコードを S1SampleApp\tests に配置しています。 python 標準の unittest を使用していますので特定のテスト用ライブラリには依存していません。

テストは下記のコマンドで実行します。

```
$ cd S1SampleApp/src
$ python3 -m unittest discover ../tests
```

4 システムの動作

4.1 ボード間の通信

各ボードは CAN バスに接続し、メッセージを送受信することにより処理を行います。 本システムでは次のように通信仕様となっています。

・通信速度：1Mbps
・アービトレーションフィールド：拡張フォーマットのみ使用
・Station ID：0x01～0xFF （Raspberry Pi ： 0x01、ベースボード：0x02～0xFF）
（ベースボードは HexSW で設定。SW1 が MSB 側）
（0x00 は、全 Station 宛てになる。）
・ノード数：CAN ドライバ IC の能力による。本システムでは 120 ノード以下。
・メッセージフォーマット：製品情報ページの「CAN メッセージフォーマット」を参照。

図 1 のハード構成ではノード数は３になっていますが、Station ID を個別に設定することでノード数を増やすことができます。

CAN バスのノード数はバックプレーン（S1BKP1001）の実装数に制限されません。電源、CAN バスが各ボードに接続されていれば、バックプレーン、サブラックは必要ありません。

4.2 ソフトウェアの動作

4.2.1 CAN コントローラのアクティブ化

ハードウェア、ソフトウェアの設定の完了後、以下のコマンドで CAN コントローラをアクティブ化します。通信を行うためには、CAN コントローラがアクティブ状態である必要があります。

```
$ sudo ip link set can0 up type can bitrate 1000000
```

4.2.2 アプリケーションの動作

システムの操作は Raspberry Pi の python アプリケーションで行います。python アプリケーションから送出されたメッセージに FPGA ファームウェアが応答する形で動作します。

（1）status_request.py

・status request message を出力（宛先 ID が 0x00 の場合、全 station が応答する。）
・該当 station から固有のメッセージを出力
・受信されたメッセージを画面に表示

（2）config_read.py

・config read message を出力（宛先 ID が 0x00 の場合、全 station が応答する。）
・該当 station から config data が出力される。(FRAM の先頭８バイト)
・受信されたメッセージを画面に表示

（3）config_write .py

・config write message を出力（宛先 ID が 0x00 の場合、全 station にライトする。）
・該当 station の config data エリアにデータが書き込まれる。(FRAM の先頭８バイト)
（DI の場合、８バイト目が検出遅延時間(ms)の設定になる。）

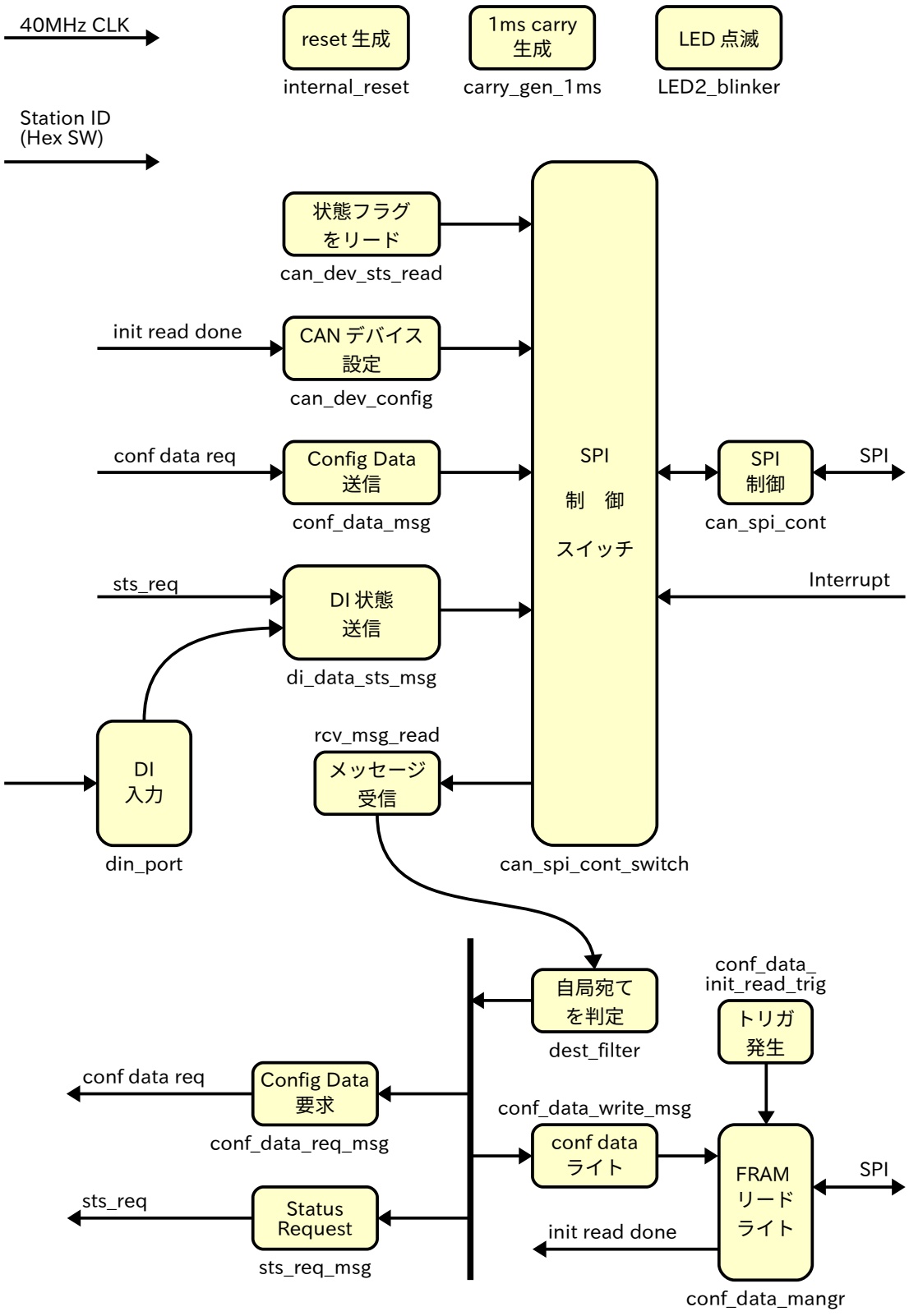
（4）do_cont.py

・do data set message を出力（宛先 ID が 0x00 の場合、全 station にライトする。）
・該当 station のデジタル出力レジスタにデータが書き込まれる。

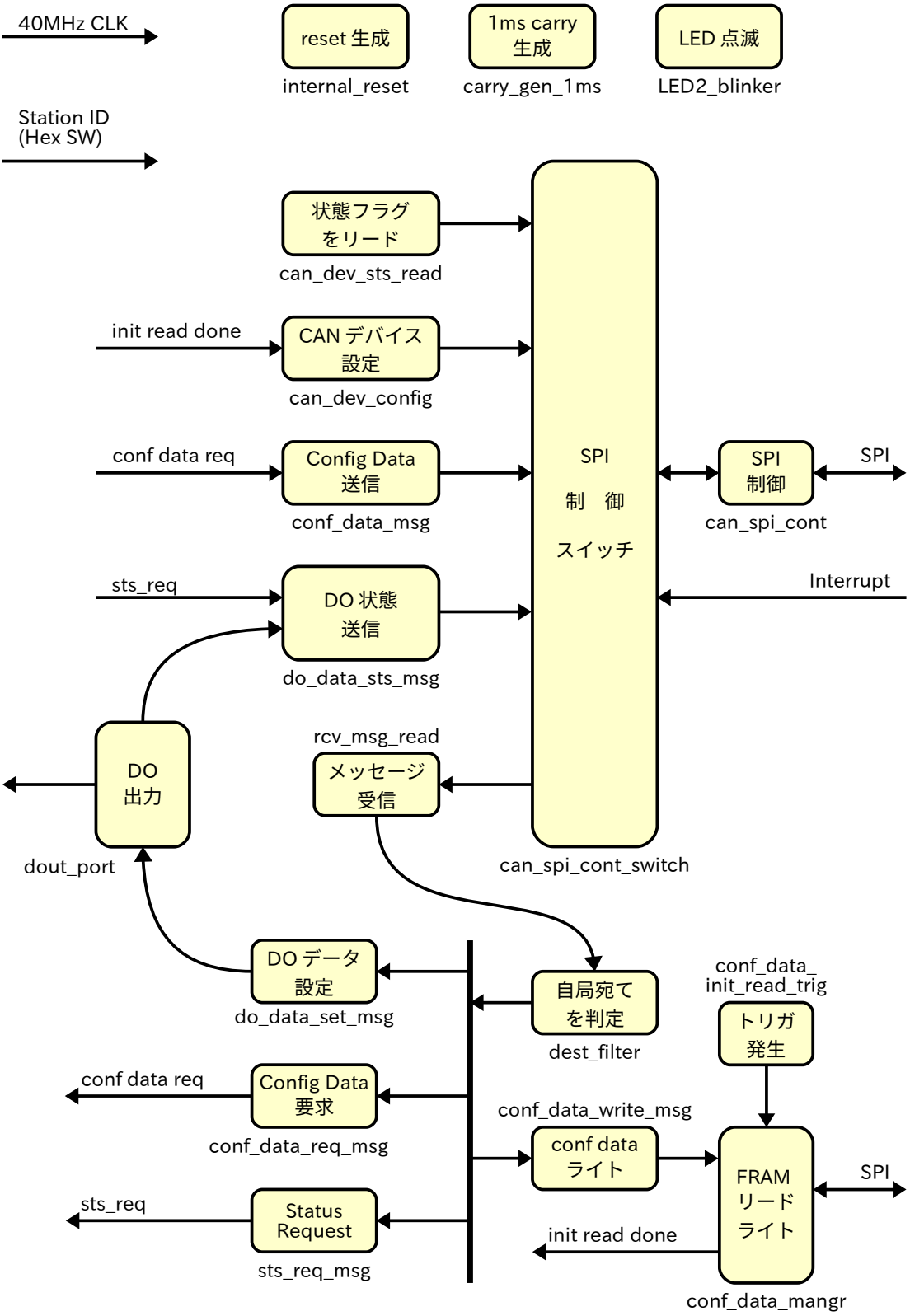
（5）message_monitor .py

・最初に status request message を出力（宛先 ID 0x00、全 station 宛て）
・各 station から出力される CAN メッセージを受信して、画面に出力する処理を繰り返す。

付録１　デジタル入力サンプルファーム(S1FRM0001)ブロック図



付録 2　デジタル出力サンプルファーム(S1FRM0002)ブロック図



付録３　python アプリケーションの help メッセージ

```
------------------------------------------------------------------------
usage: status_request.py [-h] [--src_stn_id SRC_STN_ID] [--dest_stn_id DEST_STN_ID]

status request message を送出し、受信した message を表示する.

optional arguments:
 -h, --help          show this help message and exit
 --src_stn_id SRC_STN_ID
                送信元 station id.(省略時は 0x01)
 --dest_stn_id DEST_STN_ID
                宛先 station id.(省略時は 0x00)

------------------------------------------------------------------------
usage: config_read.py [-h] [--src_stn_id SRC_STN_ID] [--dest_stn_id DEST_STN_ID]

config data request message を送出し、受信した message を表示する.

optional arguments:
 -h, --help          show this help message and exit
 --src_stn_id SRC_STN_ID
                送信元 station id.(省略時は 0x01)
 --dest_stn_id DEST_STN_ID
                宛先 station id.

------------------------------------------------------------------------
usage: config_write.py [-h] [--src_stn_id SRC_STN_ID] dest_stn_id data

config data write message を送出する.

positional arguments:
 dest_stn_id         宛先 station id.
 data              config data. ex:0x1122334455667788

optional arguments:
 -h, --help          show this help message and exit
 --src_stn_id SRC_STN_ID
                送信元 station id.(省略時は 0x01)

------------------------------------------------------------------------
usage: do_cont.py [-h] [--src_stn_id SRC_STN_ID] dest_stn_id data

do data set message を送出する.

positional arguments:
 dest_stn_id         宛先 station id.
 data              config data. ex:0x11223344

optional arguments:
 -h, --help          show this help message and exit
 --src_stn_id SRC_STN_ID
                送信元 station id.(省略時は 0x01)

------------------------------------------------------------------------
usage: message_monitor.py [-h] [--src_stn_id SRC_STN_ID]

各 station が送出する message をモニターする.

optional arguments:
 -h, --help          show this help message and exit
 --src_stn_id SRC_STN_ID
                送信元 station id.(省略時は 0x01)
```

改訂履歴

日付 バージョン	変 更 内 容
2020-01-27 ver 1.0	初版発行