

# Nios II ペリフェラル PIO (Parallel I/O) 簡易ユーザ・ガイド

ver.14

# Nios II ペリフェラル PIO (Parallel I/O) 簡易ユーザ・ガイド

## 目次

1. はじめに.....	3
2. PIO 概要.....	3
2-1. PIO 概要.....	3
2-2. PIO レジスタ・マップ.....	3
3. PIO 設定項目.....	4
3-1. Basic Settings.....	4
3-2. Input Options.....	5
3-3. Simulation.....	6
4. PIO 使用例.....	7
4-1. シンプルな Input / Output として使用する.....	7
4-1-1. Input 使用.....	7
4-1-2. Output 使用.....	8
4-2. 割り込みポートとして使用する.....	10
4-2-1. 割り込みポートとして使用する.....	10
4-2-2. 割り込みにマスクをかける.....	12
4-3. Input と Output 両方持つインタフェースとして使用する.....	13
4-4. Bidirectional (tristate) ports として使用する.....	14
改版履歴.....	15

## 1. はじめに

この資料は、Qsys に提供されている PIO (Parallel I/O) を使用して、Qsys システム外部のロジックやデバイスの制御を行う方法を簡易的に説明しています。

この資料の内容は、Quartus® II 14.0、Nios II 14.0 Software Build Tools for Eclipse を使用して動作確認済みです。

## 2. PIO 概要

### 2-1. PIO 概要

PIO は Avalon Memory-Mapped スレーブ・ポートと General I/O とのインタフェースを提供します。また、FPGA 内のロジックと接続する際にも、FPGA 外のロジックと接続する際にも使用することができます。

PIO は、ひとつのコンポーネントにつき 32 本までポートを持つことができます。Nios® II 等のホストからの制御によってレジスタへのデータの書き込み、読み込みを行います。

### 2-2. PIO レジスタ・マップ

下表は PIO のレジスタ・マップを表しています。

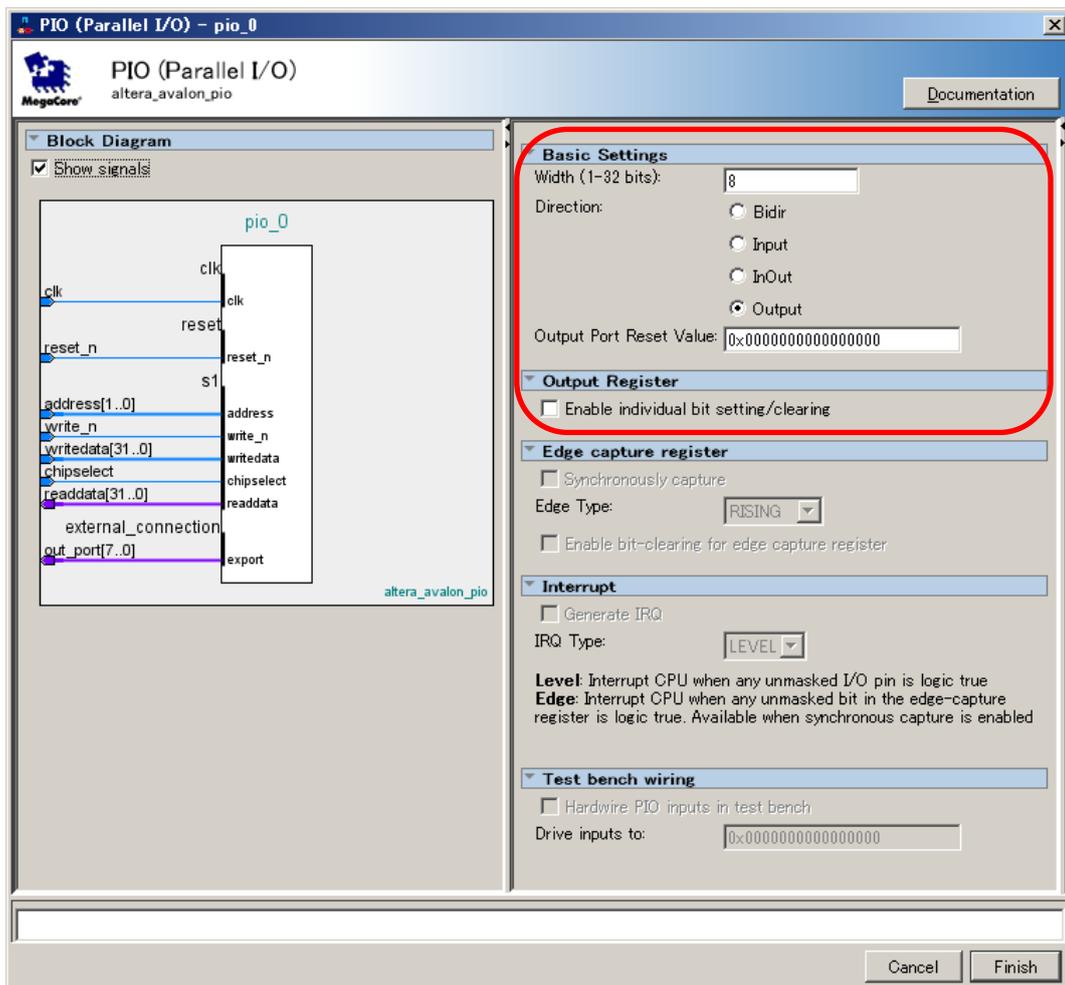
Offset	レジスタ名		Read/Write	(n-1)	....	2	1	0
0	data	Read access	R	現在 PIO にあるデータ値				
		Write access	W	PIO に出力するデータ値				
1	direction		R/W	各 I/O ポートに対する個別の入出力方向の制御。入力の場合は 0、出力の場合は 1 に設定します。				
2	interruptmask		R/W	各 I/O ポートに対する個別の割り込みのイネーブル/ディセーブル。ビットを 1 に設定すると対応するポートの割り込みが有効になります。				
3	edgecapture		R/W	各入力ポートのエッジ検出				
4	outset		W	特定ビットに値をセットします。(※1)				
5	outclear		W	特定ビットをクリアします。(※1)				

(※1) [4] outset、[5] outclear のレジスタは、Output Register を有効にした場合にのみ有効です。

### 3. PIO 設定項目

#### 3-1. Basic Settings

- ◆ Width : ポートの幅を設定します。1～32 ビットを指定することができます。
- ◆ Direction : ポートの制御方向を設定します。
  - ▶ Bidi : それぞれのビットがひとつのピンを使用して、駆動とデータの取り込みを行います。それぞれのビットごとに入力と出力を選択することができます。ポートを入力に設定した場合には、FPGA の I/O ピンはトライステートに設定されます。
  - ▶ Input : 入力ポートとしてのみ使用することができます。
  - ▶ InOut : 入力と出力のポートがそれぞれ別に生成されます。
  - ▶ Output : 出力ポートとしてのみ使用することができます。
- ◆ Out Port Reset Value : ポートを出力に設定した際のリセット値を指定します。デフォルトは 0 です。
- ◆ Output Register : オンにすると、ポートの特定のビットの制御が有効になります。デフォルトの設定はオフです。(レジスタ [4] outset、[5] outclear のレジスタを操作することによって制御します。)



### 3-2. Input Options

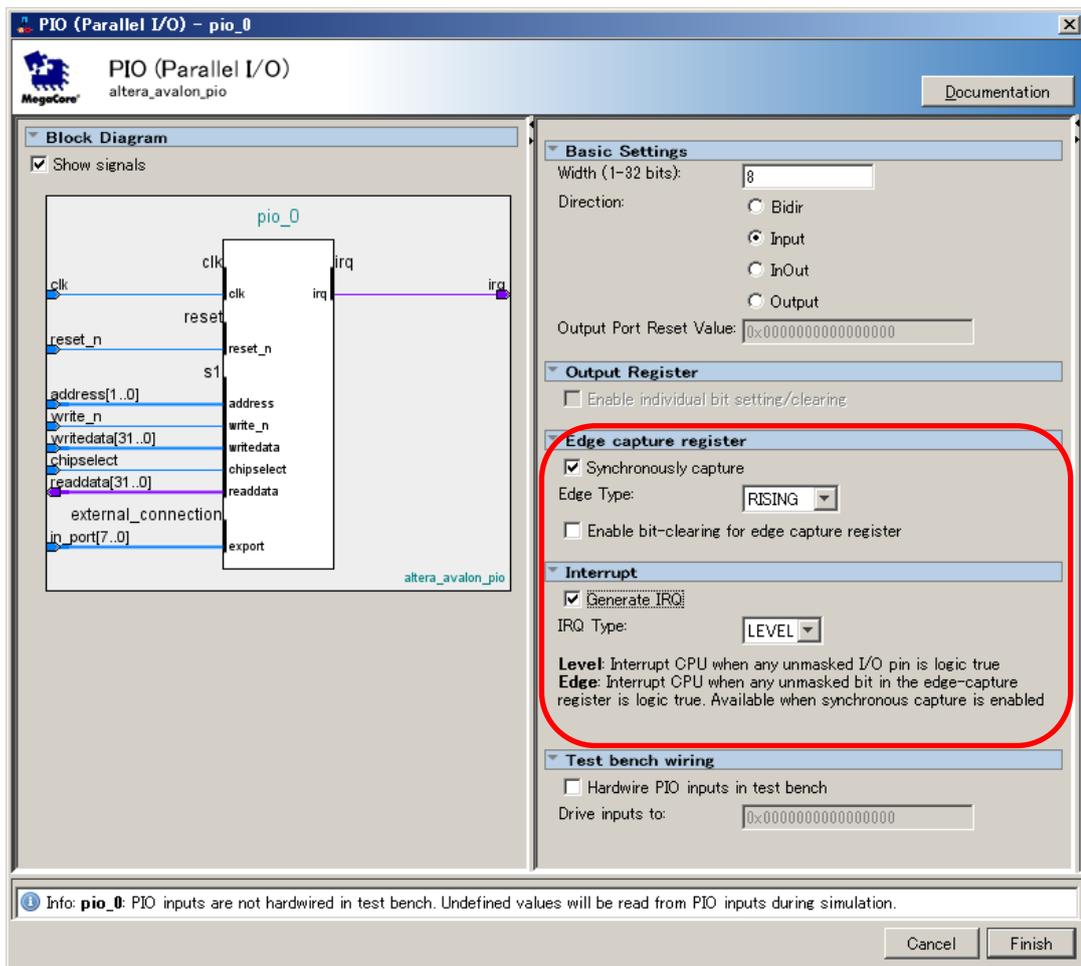
Input Options では Direction を入力に設定した場合の、エッジ・キャプチャ・レジスタの設定と割り込みの設定を行います。

#### ■ Edge capture register

- ▶ Synchronously capture : オンにすると、ポートの入力信号のエッジを取り込むことができます。
  - ◇ RISING : 立ち上がりエッジを取り込みます。
  - ◇ FALLING : 立下りエッジを取り込みます。
  - ◇ ANY : 両エッジを取り込みます。
- ▶ Enable bit-clearing for edge capture register : オンにすると、特定ビットのみを操作することができます。

#### ■ Interrupt

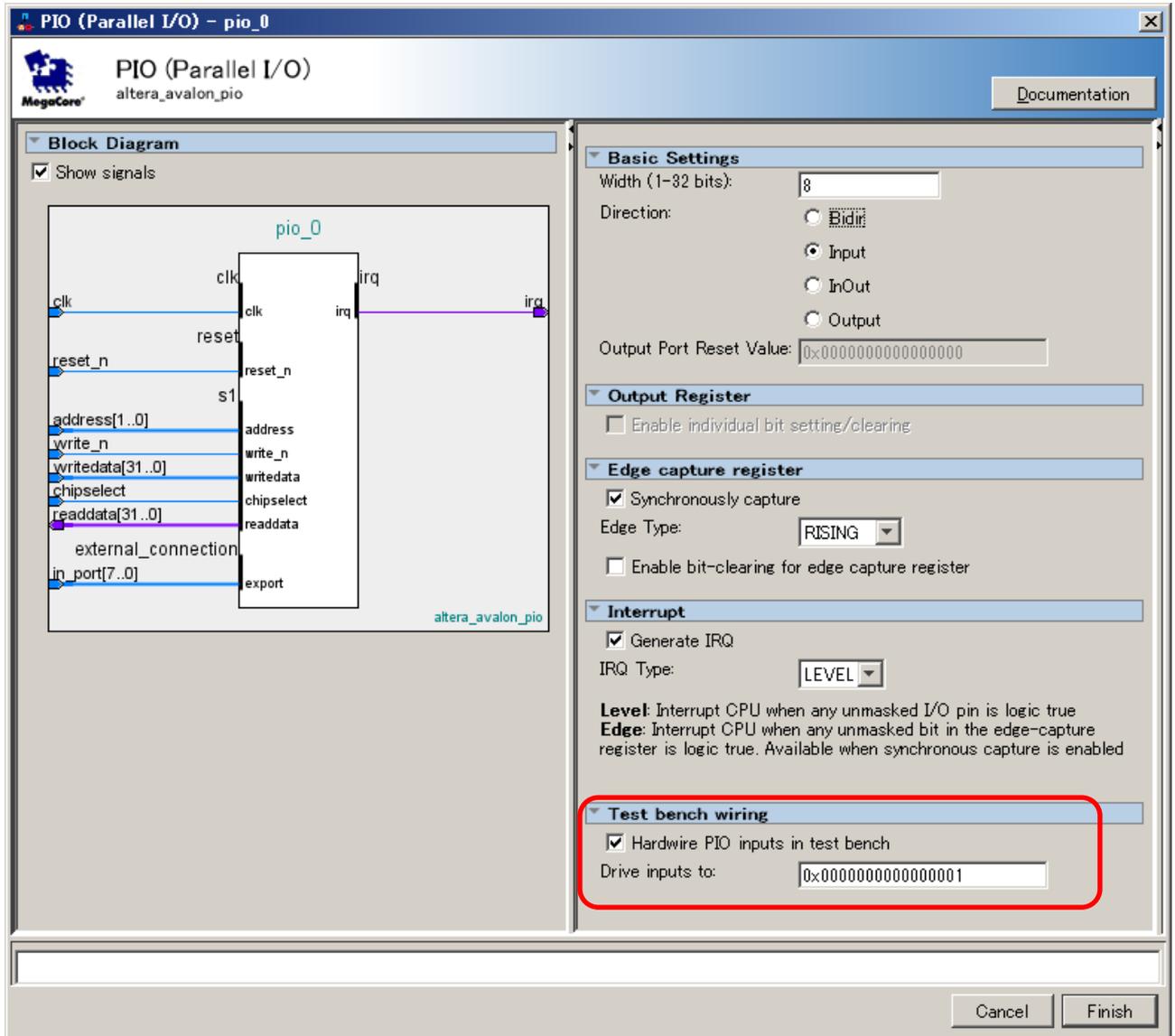
- ▶ Generate IRQ : オンにすると、入力信号から割り込みを発生させます。
  - ◇ LEVEL : 入力ポートのレベルを読んで割り込みを発生します。
    - ※ 入力信号の High を認識します。入力信号が Low アクティブの場合には、外部に NOT ゲートを挿入する必要があります。
  - ◇ EDGE : 入力ポートのエッジを取り込んで割り込みを発生します。
    - ※ 上記の“Edge capture register”で設定したエッジを認識します。



The screenshot shows the PIO (Parallel I/O) configuration window for the 'pio\_0' component. The 'Basic Settings' section shows Width (1-32 bits) set to 8, Direction set to Input, and Output Port Reset Value set to 0x0000000000000000. The 'Output Register' section has 'Enable individual bit setting/clearing' unchecked. The 'Edge capture register' section is highlighted with a red circle and contains the following settings: 'Synchronously capture' is checked, 'Edge Type' is set to 'RISING', and 'Enable bit-clearing for edge capture register' is unchecked. The 'Interrupt' section has 'Generate IRQ' checked and 'IRQ Type' set to 'LEVEL'. Below this, there are explanatory notes: 'Level: Interrupt CPU when any unmasked I/O pin is logic true' and 'Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled'. The 'Test bench wiring' section has 'Hardwire PIO inputs in test bench' unchecked and 'Drive inputs to:' set to 0x0000000000000000. An information message at the bottom states: 'Info: pio\_0: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.' Buttons for 'Cancel' and 'Finish' are at the bottom right.

### 3-3. Simulation

Test bench wiring では、シミュレーション実行時に入力信号として与える初期値を設定します。設定された値は、ツールが生成するテストベンチに記述されます。



## 4. PIO 使用例

### 4-1. シンプルな Input / Output として使用する

#### 4-1-1. Input 使用

PIO と接続したボタン(4ビット)を読み込む場合の例です。PIO の設定は下記のとおりです。

ソフトウェアからボタンを接続した PIO の data レジスタの値を読み込んで、ボタンの ON/OFF を判断する場合は、割り込みを使用しない場合の例です。

The screenshot shows the configuration interface for the PIO peripheral. The following settings are highlighted with red boxes and annotated with yellow callouts:

- Basic Settings:**
  - Width (1-32 bits):** Set to 4. Annotation: [ポート幅を設定] 4ビットのボタンを接続する場合
  - Direction:** Input selected. Annotation: [ポートの入出力設定] 入力ポート
- Output Register:**
  - Enable individual bit setting/clearing:** Unchecked. Annotation: 入力ポートなので使用しません
- Edge capture register:**
  - Synchronously capture:** Unchecked. Annotation: エッジ・キャプチャ・レジスタと割り込み入力を使用しない場合
  - Edge Type:** RISING
  - Enable bit-clearing for edge capture register:** Unchecked
- Interrupt:**
  - Generate IRQ:** Unchecked. Annotation: エッジ・キャプチャ・レジスタと割り込み入力を使用しない場合
  - IRQ Type:** LEVEL
- Test bench wiring:**
  - Hardwire PIO inputs in test bench:** Unchecked
  - Drive inputs to:** 0x0000000000000000

4-1-2. Output 使用

PIO を使用して、ボード上の LED (8ビット)を制御する場合の例です。PIO の設定は下記のとおりです。この場合は出力ポートに設定しますので、Input Options と Simulation の設定は必要ありません。

**Basic Settings**

Width (1-32 bits): 8

Direction:

- Bidir
- Input
- InOut
- Output

Output Port Reset Value: 0x0000000000000000

**Output Register**

Enable individual bit setting/clearing

**Edge capture register**

Synchronously capture

Edge Type: RISING

Enable bit-clearing for edge capture register

**Interrupt**

Generate IRQ

IRQ Type: LEVEL

**Test bench wiring**

Hardwire PIO inputs in test bench

Drive inputs to: 0x0000000000000000

[ポート幅を設定]  
8ビット LED を接続する場合

[ポートの入出力設定]  
出力ポート

[リセット値を設定]  
0x0 に設定した場合

[特定ビットの操作]  
OFF の場合

下記は PIO 経由で接続した外部のボタンと LED を制御するシンプルなプログラムです。

ボタンからの入力を IORD を使用して読み込みます。ボタンが押された際の操作の中で IOWR で LED を操作し点灯させます。

#### ソフトウェア例) ボタンの入力を読み込んで LED を点灯

```
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"

#define NONE_PRESSED 0xF
#define DEBOUNCE 1000

int main(void)
{
    int buttons;    // ボタンの値を保持するために使用
    int led = 0;    // LED に値を書き込むために使用

    printf("Simple\n"); // 実行時にコンソールに" Simple" の文字出力

    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, led); // PIO に初期値を書き込む

    while(1)
    {
        buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE); // PIO 経由でボタンの値を読み込む

        if(buttons != NONE_PRESSED) // if ボタンが押されていたら
        {
            led++;

            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, led); // PIO に新しい値を書き込み

            /* スイッチのデバウンス処理
             ボタンが最初に押されてから少し時間をおく
             ボタンが離されるまで待ち、その後時間をおく */
            usleep(DEBOUNCE);
            while(buttons != NONE_PRESSED) // ボタンが離されるのを待つ
                buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE); // buttons の値をアップデート
            usleep(DEBOUNCE);
        }
    }
}
```

## 4.2. 割り込みポートとして使用する

PIO からの入力信号を Nios II の割り込み信号として使用する方法です。

### 4.2-1. 割り込みポートとして使用する

下記の設定は割り込みを PIO の入力信号の立ち上がりエッジで発生させる場合の例です。Basic Setting のページは“4-1-1. Input 使用”と同様に設定します。

Edge capture register で Synchronously capture にチェックを入れてイネーブルにし、RISING を選択します。

Interrupt で割り込みをイネーブルにします。割り込みはレベル割り込みとエッジ割り込みの一方を選択することができます。レベル割り込みは High を認識します。エッジ割り込みは Edge capture register で設定したエッジを検出し、割り込みを発生します。

The screenshot shows the configuration interface for the PIO peripheral. The following settings are highlighted with red boxes and annotated with yellow callouts:

- Basic Settings:** Width (1-32 bits): 4; Direction: Input (selected).
- Edge capture register:**
  - Synchronously capture: エッジ・キャプチャ・レジスタをイネーブル
  - Edge Type: RISING: [エッジの設定]立ち上がりエッジの場合
  - Enable bit-clearing for edge capture register: [特定ビットの操作]ディセーブルの場合
- Interrupt:**
  - Generate IRQ: 割り込み信号生成をイネーブル
  - IRQ Type: EDGE: [割り込みレベル or エッジ]エッジ割り込み設定の場合

Edge capture register は、割り込みサービス・ルーチンの中でクリアする必要があります。クリアしないまま割り込みサービス・ルーチンから戻ると、Edge capture register は一度取り込んだエッジを保持したままなので、次に発生したエッジを検出することができません。

以下のサンプル・ソースは、ボタン割り込みを使用した場合のソフトウェア・サンプルです。このサンプルでは、ボタン4つを接続した PIO の Edge capture register を使用して、ボタンの押された位置を検出します。同時にボタンからの割り込みを登録しているため、ボタンが押されると Nios II に対して割り込みが発生し、登録した割り込みサービス・ルーチンが実行されます。

割り込みサービス・ルーチンでは、Edge capture register の値を読み込んでボタンの位置に対応した LED を点灯します。

### ソフトウェア例) 割り込みを使用してボタンの入力を読み込む

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"

/* ボタン割り込みのサービス・ルーチン */

static void button_interrupts(void* context, alt_u32 id)
{
    int button_position; // ボタンが押された場所の保存に使用
    button_position = IORD_ALTERA_AVALON_PIO_EDGE_CAP (BUTTON_PIO_BASE);
                    // ボタンPIOのエッジ・キャプチャ・レジスタを読み込み
                    // どのポジションのボタンが押されたかを検出

    IOWR_ALTERA_AVALON_PIO_DATA (LED_PIO_BASE, button_position);
                    // 押されたボタンに対応したLEDのビットを点灯

    /* エッジ・キャプチャ・レジスタのリセット */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP (BUTTON_PIO_BASE, 0x0);
    button_position = 0x0;
}

/* メインルーチン */

int main(void)
{
    printf("Interrupt test\n");

    /* PIOの初期化 */
    // 特定のボタンのみマスクをかけて有効にする (ボタンが4つの場合)
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK (BUTTON_PIO_BASE, 0xF);

    // エッジ・キャプチャ・レジスタのリセット
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP (BUTTON_PIO_BASE, 0x0);

    // 割り込みハンドラの登録
    alt_irq_register (BUTTON_PIO_IRQ, 0, button_interrupts);

    while(1)
    {
        // ボタンの割り込みを待つ
    }

    return 0;
}
```

#### 4-2-2. 割り込みにマスクをかける

上記のソフトウェア・サンプルでは、

```
IOWR_ALTERA_AVALON_PIO_IRQ_MASK (BUTTON_PIO, 0xF);
```

の記述で割り込みにマスクを設定しています。これは、4ビットのボタンを想定し、すべてのビットからの入力を有効にします。

このマクロを使用して、特定ビット入力のみを割り込みとしてイネーブルすることができます。例えば、32ビットのPIO (INPUT\_PIO)のビット0とビット1のみを割り込み信号としてイネーブルする場合には、

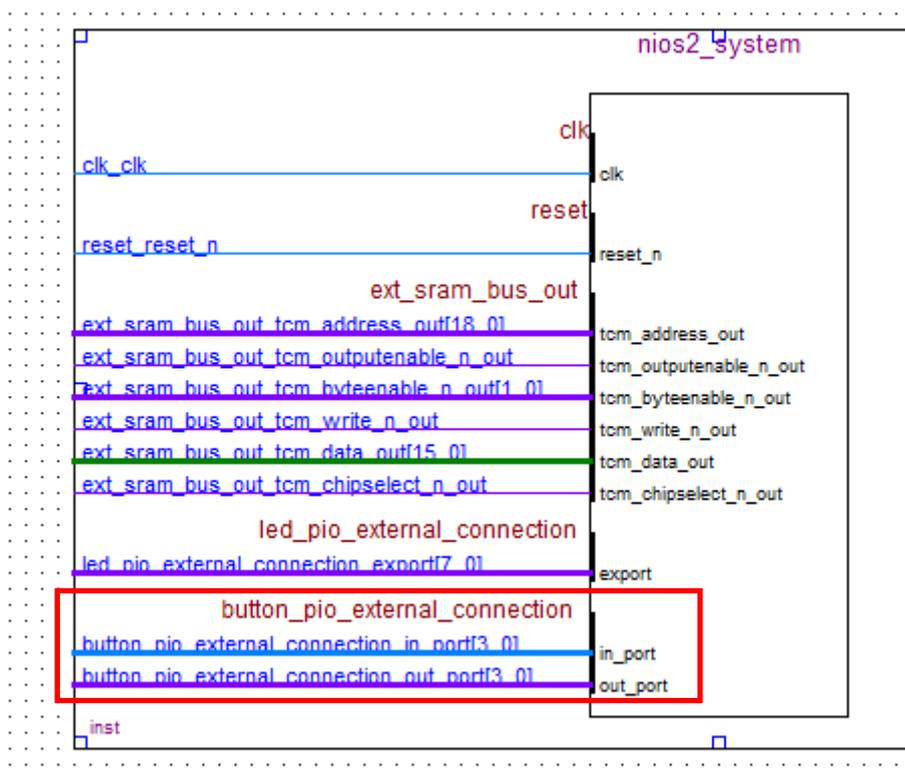
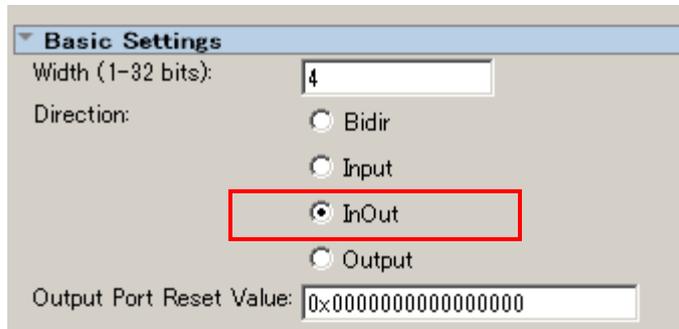
```
IOWR_ALTERA_AVALON_PIO_IRQ_MASK (INPUT_PIO, 0x3);
```

と設定します。

### 4-3. Input と Output 両方持つインタフェースとして使用する

ひとつの PIO を別々の入力ポートと出力ポートを持つインタフェースとして使用する方法です。

PIO の Basic Settings の Direction で InOut を選択します。下記に示すブロック図のように、ひとつの PIO に入力ポートと出力ポートが生成されます。



入力と出力の切り替えはソフトウェアから、PIO の direction レジスタへ入力の場合は 0、出力の場合は 1 を書き込んで切り替えます。“altera\_avalon\_pio\_regs.h” ファイルの中に direction レジスタへのアクセスのマクロが用意されていますので、ファイルをインクルードして使用することができます。

例えば下記のように記述して、入力と出力を切り替えます。(PIO の名前; BOTH\_IN\_OUT\_PIO)

■ 入力にする場合

```
IOWR_ALTERA_AVALON_PIO_DIRECTION (BOTH_IN_OUT_PIO_BASE, 0x0);
```

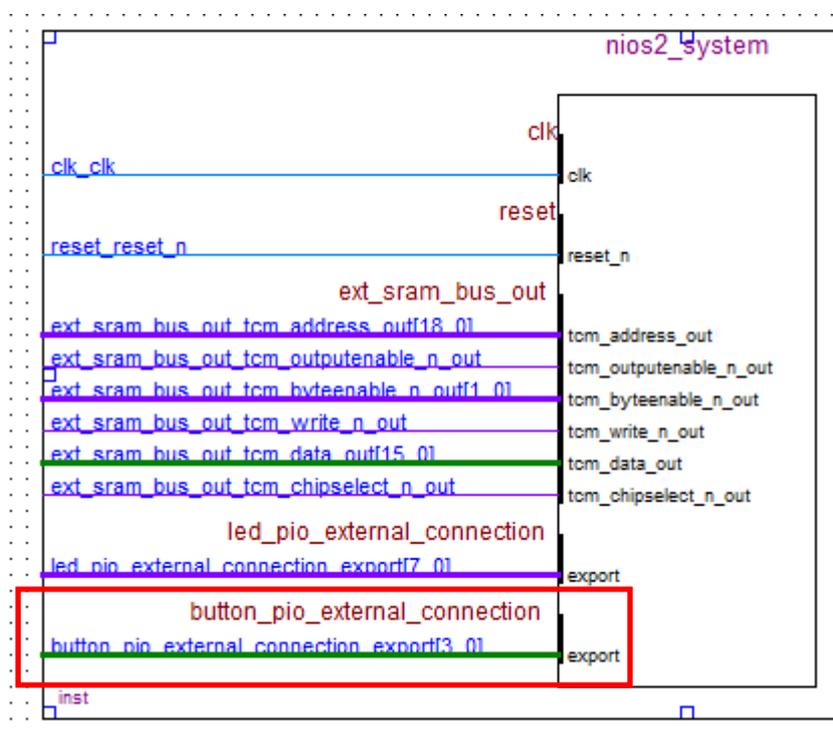
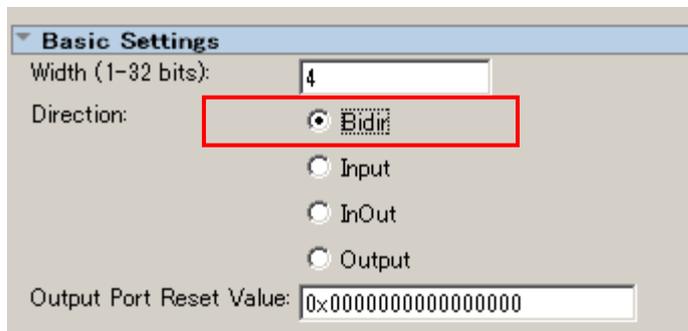
■ 出力にする場合

```
IOWR_ALTERA_AVALON_PIO_DIRECTION (BOTH_IN_OUT_PIO_BASE, 0x1);
```

#### 4.4. Bidirectional (tristate) ports として使用する

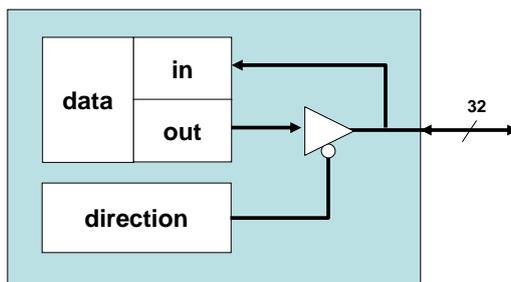
ひとつのポートをトライ・ステート制御して双方向ピンとして使用する方法です。

PIO の Basic Settings の Direction で Bidir を選択します。下記に示すブロック図のように双方向のポートが生成されます。



入出力の切り替えは、InOut のポートの際と同様に、PIO の direction レジスタへ入力の場合は 0、出力の場合は 1 を書き込んで切り替えます。

Bidirectional ポートを使用した場合には、下図のように自動でトライ・ステート制御の回路が挿入されます。



## 改版履歴

Revision	年月	概要
1	2014 年 8 月	新規作成

### 免責およびご利用上の注意

弊社より資料を入手されましたお客様におかれましては、下記の使用上の注意を一読いただいた上でご使用ください。

1. 本資料は非売品です。許可無く転売することや無断複製することを禁じます。
2. 本資料は予告なく変更することがあります。
3. 本資料の作成には万全を期していますが、万一ご不明な点や誤り、記載漏れなどお気づきの点がありましたら、本資料を入手されました下記代理店までご一報いただければ幸いです。  
 株式会社アルティマ ホームページ: <http://www.altima.co.jp> 技術情報サイト EDISON: <https://www.altima.jp/members/index.cfm>  
 株式会社エルセナ ホームページ: <http://www.elsena.co.jp> 技術情報サイト ETS : <https://www.elsena.co.jp/elspear/members/index.cfm>
4. 本資料で取り扱っている回路、技術、プログラムに関して運用した結果の影響については、責任を負いかねますのであらかじめご了承ください。
5. 本資料は製品を利用する際の補助的な資料です。製品をご使用になる際は、各メーカー発行の英語版の資料もあわせてご利用ください。