

Nios II - PIO を使用した I²C-Bus (2 ワイヤ) マスタの実装

ver.1.0

2010 年 6 月

Nios II - PIO を使用した I²C-Bus (2 ワイヤ)マスタの実装

目次

| | |
|------------------------------------|---|
| 1. はじめに..... | 3 |
| 2. 適用条件..... | 3 |
| 3. システムの構成..... | 3 |
| 3-1. SOPC Builder の設定..... | 3 |
| 3-2. PIO の設定..... | 4 |
| 3-2-1. シリアル・クロック・ライン用 PIO の設定..... | 4 |
| 3-2-2. シリアル・データ・ライン用 PIO の設定..... | 4 |
| 3-2-3. Nios II を含むその他のペリフェラル..... | 4 |
| 4. Nios II IDE の実装..... | 5 |
| 4-1. START Condition の出力処理..... | 5 |
| 4-2. STOP Condition の出力処理..... | 5 |
| 4-3. 1bit の送信処理..... | 6 |
| 4-4. 1bit の受信処理..... | 6 |
| 4-5. 1byte の送信処理..... | 7 |
| 4-6. 1byte の受信処理..... | 7 |
| 4-7. EEPROM 書き込み処理..... | 8 |
| 4-8. EEPROM 読み出し処理..... | 8 |
| 5. 補足..... | 9 |
| 5-1. SignalTap II での波形観測..... | 9 |
| 5-1-1. 書き込み動作波形..... | 9 |
| 5-1-2. 読み出し動作波形..... | 9 |
| 5-1-3. データ・レート..... | 9 |

1. はじめに

この資料は、マイクロ・プロセッサとデバイス・チップとの通信で一般的に使用されている I²C-Bus (2 ワイヤ)通信を、PIO を使用して実現する方法を紹介しています。

I²C-Bus は、シリアル・クロック・ライン (SCLK) とシリアル・データ・ライン (SDAT) の 2 本のバス・ラインだけで構成されており、比較的低いビット・レートの通信を想定したバスのため、ソフトウェアで信号を制御して通信を行うことが可能です。

※ I²C はフィリップス社の商標です。

※ I²C-Bus の詳細はメーカーの資料をご参照ください。

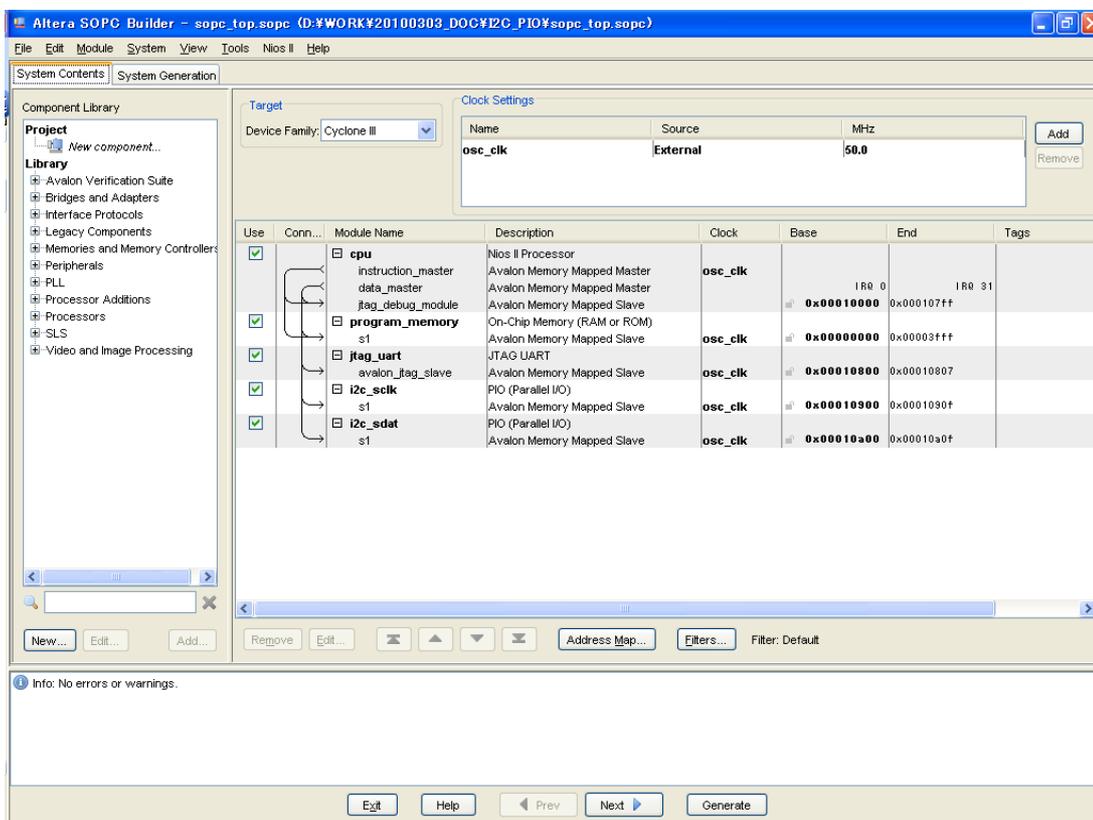
2. 適用条件

本資料では Nios[®] II をマスタ・デバイス、EEPROM をスレーブ・デバイスとした通信を紹介していますが、複数のマスタが混在する場合や、Nios II をスレーブ・デバイスとした実装については、この資料の中では紹介しておりません。

3. システムの構成

3-1. SOPC Builder の設定

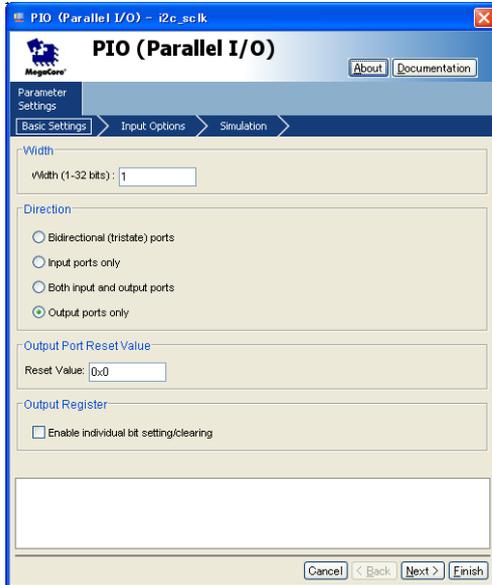
SOPC Builder では、シリアル・クロック・ラインの接続する i2c_sclk と、シリアル・データ・ラインに接続する i2c_sdat を PIO で作成し、Nios II の data_master に接続してください。



3-2. PIO の設定

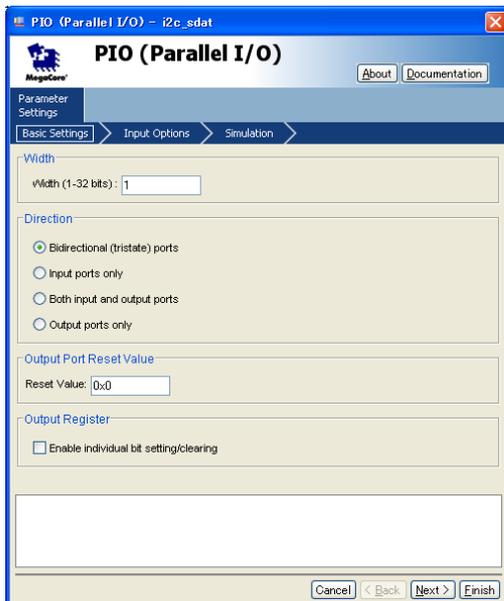
3-2-1. シリアル・クロック・ライン用 PIO の設定

- Width : 1bit
- Direction : Output ports Only



3-2-2. シリアル・データ・ライン用 PIO の設定

- Width : 1bit
- Direction : Bidirection(tristate) ports
- Input Option : 全て Desable



3-2-3. Nios II を含むその他のペリフェラル

本実装のために設定を変更する箇所はありません。

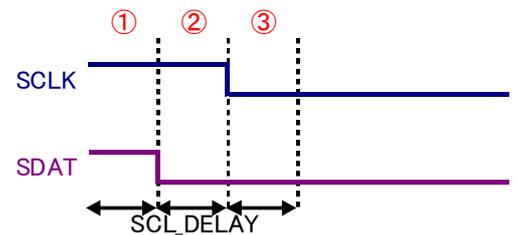
4. Nios II IDE の実装

4-1. START Condition の出力処理

I²C-Bus では、一連のデータを送信する直前に START Condition を定義しています。この状態は、SCLK が “H” の区間で SDAT が “H” から “L” に変化することで生成されます。

```

inline void i2c_start(long sclk, long sdat)
{
    SCL_DELAY;
    // SDAT を出力ポートに
    IOWR_ALTERA_AVALON_PIO_DIRECTION(sdat,1);
    // SDAT=1..SCLK=1
    IOWR_ALTERA_AVALON_PIO_DATA(sdat, 1); -①
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 1); -①
    SCL_DELAY;
    // SDAT=0..SCLK=0
    IOWR_ALTERA_AVALON_PIO_DATA(sdat, 0); -②
    SCL_DELAY;
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 0); -③
}
    
```

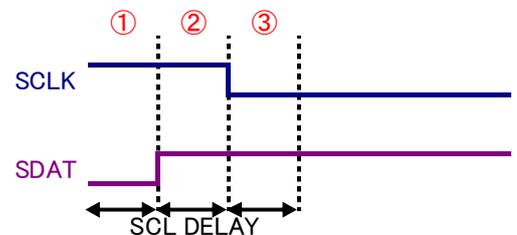


4-2. STOP Condition の出力処理

I²C-Bus では、一連のデータの終端を STOP Condition を定義しています。この状態は、SCLK が “H” の区間で SDAT が “L” から “H” に変化することで生成されます。

```

inline void i2c_stop(long sclk, long sdat)
{
    SCL_DELAY;
    // SDAT を出力ポートに
    IOWR_ALTERA_AVALON_PIO_DIRECTION(sdat,1);
    // SDAT=0..SCLK=1
    IOWR_ALTERA_AVALON_PIO_DATA(sdat, 0); -①
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 1); -①
    SCL_DELAY;
    // SDAT=1..SCLK=0
    IOWR_ALTERA_AVALON_PIO_DATA(sdat, 1); -②
    SCL_DELAY;
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 0); -③
}
    
```

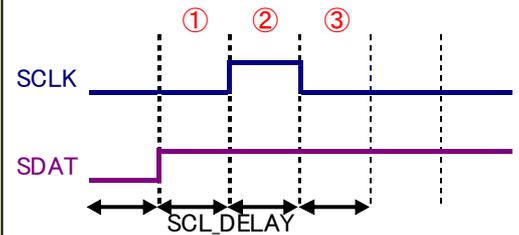


4-3. 1bit の送信処理

I²C-Bus では、データの送信に SCLK が “H” の期間は SDAT の状態は一定でなければなりません。したがって、データを送信する場合は、SCLK が “L” の期間に SDAT を変化させる必要があります。

```

inline void i2c_send_bit(long sclk, long sdat, int Data)
{
    // SCLK=0
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 0);
    // SDAT を出力ポートに
    IOWR_ALTERA_AVALON_PIO_DIRECTION(sdat,1); -①
    // SDAT にデータ送出
    IOWR_ALTERA_AVALON_PIO_DATA(sdat, (Data)?1:0); -①
    SCL_DELAY;
    // SDAT=1..SCLK=0
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 1); -②
    SCL_DELAY;
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 0); -③
}
    
```



4-4. 1bit の受信処理

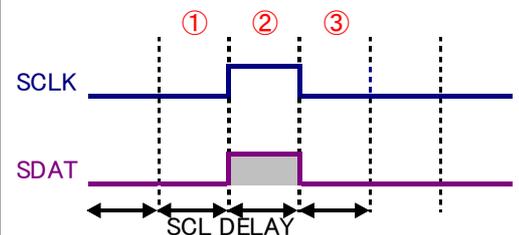
I²C-Bus では、データの受信では SCLK が “H” の期間に SDAT の値を読み出す必要があります。

```

inline int i2c_recv_bit(long sclk, long sdat)
{
    int Data;

    // SCLK=0
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 0); -①
    SCL_DELAY;
    // SCLK=1
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 1); -②
    // SDAT を入力ポートに
    IOWR_ALTERA_AVALON_PIO_DIRECTION(sdat,0); -②
    // SDAT からデータを取得
    Data = IORD_ALTERA_AVALON_PIO_DATA(sdat); -②
    SCL_DELAY;
    // SCLK=0
    IOWR_ALTERA_AVALON_PIO_DATA(sclk, 0); -③

    return Data;
}
    
```



4-5. 1byte の送信処理

『1bit 送信処理』を使用し 1byte のデータを送信します。全ての bit の送信が完了した後、『1bit 受信処理』を使用しスレーブからのアクリッジが正しく受信されたかを確認します。

```

int i2c_send_byte(long sclk, long sdat, unsigned short Data)
{
    unsigned short Mask = 1 << (BIT_WIDTH - 1);
    int i;

    // ビット幅分データ送信
    for(i = 0; i < BIT_WIDTH; i++)
    {
        // 1bit 送信
        i2c_send_bit(sclk, sdat, (Data & Mask));
        // bit マスクをシフト
        Mask >>= 1; // there is a delay in this command
    }

    // ACK の受信
    return i2c_rcv_bit(sclk, sdat);
}
    
```

4-6. 1byte の受信処理

『1bit 受信処理』を使用し 1byte 分のデータを受信します。全ての bit の受信が完了した後、『1bit 送信処理』を使用しスレーブへの ACK を送信します。

```

unsigned short i2c_rcv_byte(long sclk, long sdat)
{
    unsigned short Data = 0;
    int i;

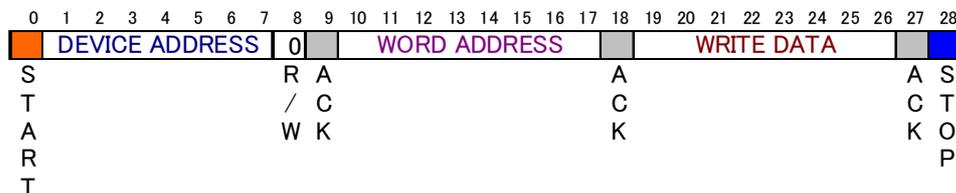
    // ビット幅分データ送信
    for(i = 0; i < BIT_WIDTH; i++)
    {
        // 受信データのシフト
        Data <<= 1;
        // 1bit 受信
        if(i2c_rcv_bit(sclk, sdat))
            Data |= 0x01;
    }

    // ACK の送信
    i2c_send_bit(sclk, sdat, 1);

    return Data;
}
    
```

4-7. EEPROM 書き込み処理

EEPROM への書き込みは、下記のフォーマットのデータを送信することで行います。



```

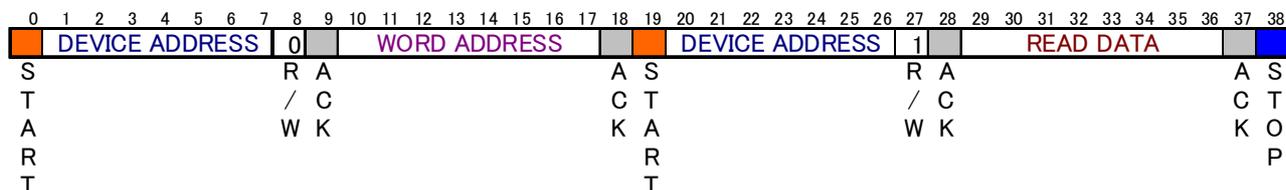
void i2c_send(long sclk, long sdat, unsigned char address, unsigned char reg, unsigned char data)
{
    i2c_start(sclk, sdat);

    i2c_send_byte(sclk, sdat, address << 1);
    i2c_send_byte(sclk, sdat, reg);
    i2c_send_byte(sclk, sdat, data);

    i2c_stop(sclk, sdat);
}
    
```

4-8. EEPROM 読み出し処理

EEPROM からの読み出しは、下記のフォーマットのデータを送受信することで行います。



```

unsigned char i2c_recv(long sclk, long sdat, unsigned char address, unsigned char reg)
{
    unsigned char data;

    i2c_start(sclk, sdat);

    i2c_send_byte(sclk, sdat, (address << 1));
    i2c_send_byte(sclk, sdat, reg);

    i2c_start(sclk, sdat);

    i2c_send_byte(sclk, sdat, (address << 1) | 0x01);
    data = i2c_recv_byte(sclk, sdat);

    i2c_stop(sclk, sdat);

    return data;
}
    
```

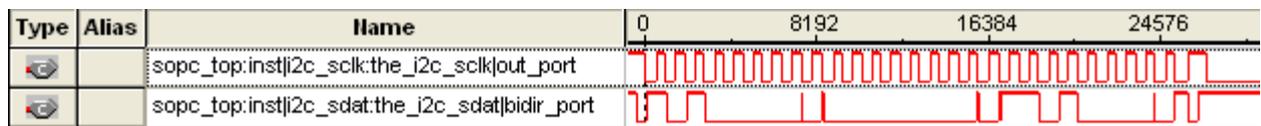
5. 補足

5-1. SignalTap II での波形観測

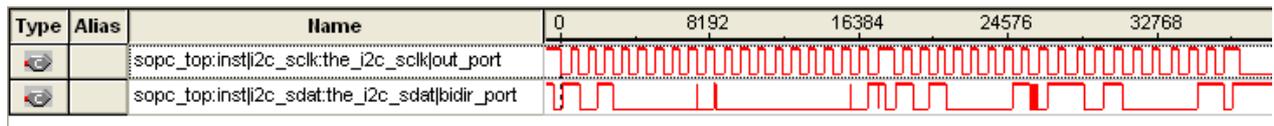
評価ボードでの信号を SignalTap[®] II で計測した波形を下記に示します。

- Device : Cyclone III EP3C25F324C8
- Processor core : Nios II/e 50Hz
- Program Memory : On-Chip RAM(16Kbyte)
- Optimize option : -O3

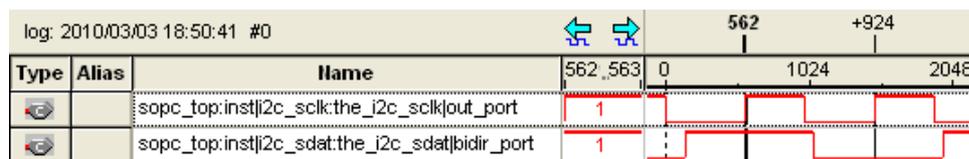
5-1-1. 書き込み動作波形



5-1-2. 読み出し動作波形



5-1-3. データ・レート



サンプリング・クロック : 50MHz = 20ns

パルス周期 : 924clock

データ・レート : $1 / (20\text{ns} * 924\text{clock}) = 54.113\text{Kbp}$

免責、及び、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 : 〒222-8563 横浜市港区新横浜 1-5-5 マクニカ第二ビル TEL: 045-476-2155 HP: <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 : 〒163-0928 東京都新宿区西新宿 2-3-1 新宿モノリス 28F TEL: 03-3345-6205 HP: <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場合は、英語版の資料もあわせてご利用ください。