

Nios® II I2C マスターの活用術 Avalon®-ST インターフェースによる通信

Ver.17.1

Nios® II – I2C マスターの活用術 Avalon-ST インターフェースによる通信

目次

1. はじめに	3
2. 適用条件	3
2-1. 対応バージョン	3
2-2. 検証ハードウェア	3
3. 仕様	3
3-1. 機能	3
3-2. I2C Master の使い方	4
4. 実装	6
4-1. ベース・プロジェクトの準備	6
4-2. Platform Designer の編集	6
4-3. Quartus Prime の編集	9
4-4. Nios II Software Build Tool (以降 Nios II SBT) の編集	11
5. 検証	15
5-1. 動作の確認	15
6. 補足	16
6-1. 注意事項	16
7. 参考資料	17
改版履歴	18

1. はじめに

Platform Designer には標準で Avalon® I2C (Master) Core が準備されており、この IP コアは Nios® II による制御で I2C 通信を行う事ができます。この IP コアは、デフォルトでは Avalon Memory Mapped Interface (Avalon-MM) ポートと接続し、データも含めレジスタ操作で通信をするように実装されますが、送受信データを Avalon Streaming Interface (Avalon-ST) で入出力する事ができ、その Avalon-ST ポートに DMA や FIFO 等を接続する事で、Nios II によるレジスタ操作を最小限に抑えた制御が可能です。本資料では、その手法について説明しています。

2. 適用条件

2-1. 対応バージョン

本資料では、下記のツール、バージョンを使用しています。

- Intel® Quartus® Prime Standard Edition Version 17.1.0
- Nios II Software Build Tools (SBT) for Eclipse Version 17.1.0

※ 17.1 以外のバージョンでも同様の方法で実装することは可能ですが、一部の機能や操作方法が異なる場合がありますのでご注意ください。

2-2. 検証ハードウェア

- Atlas-SoC Kit (DE0-Nano-SoC Kit)
(FPGA: Cyclone® V SE 5CSEMA4U23C6N)

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=941&PartNo=4>

3. 仕様

3-1. 機能

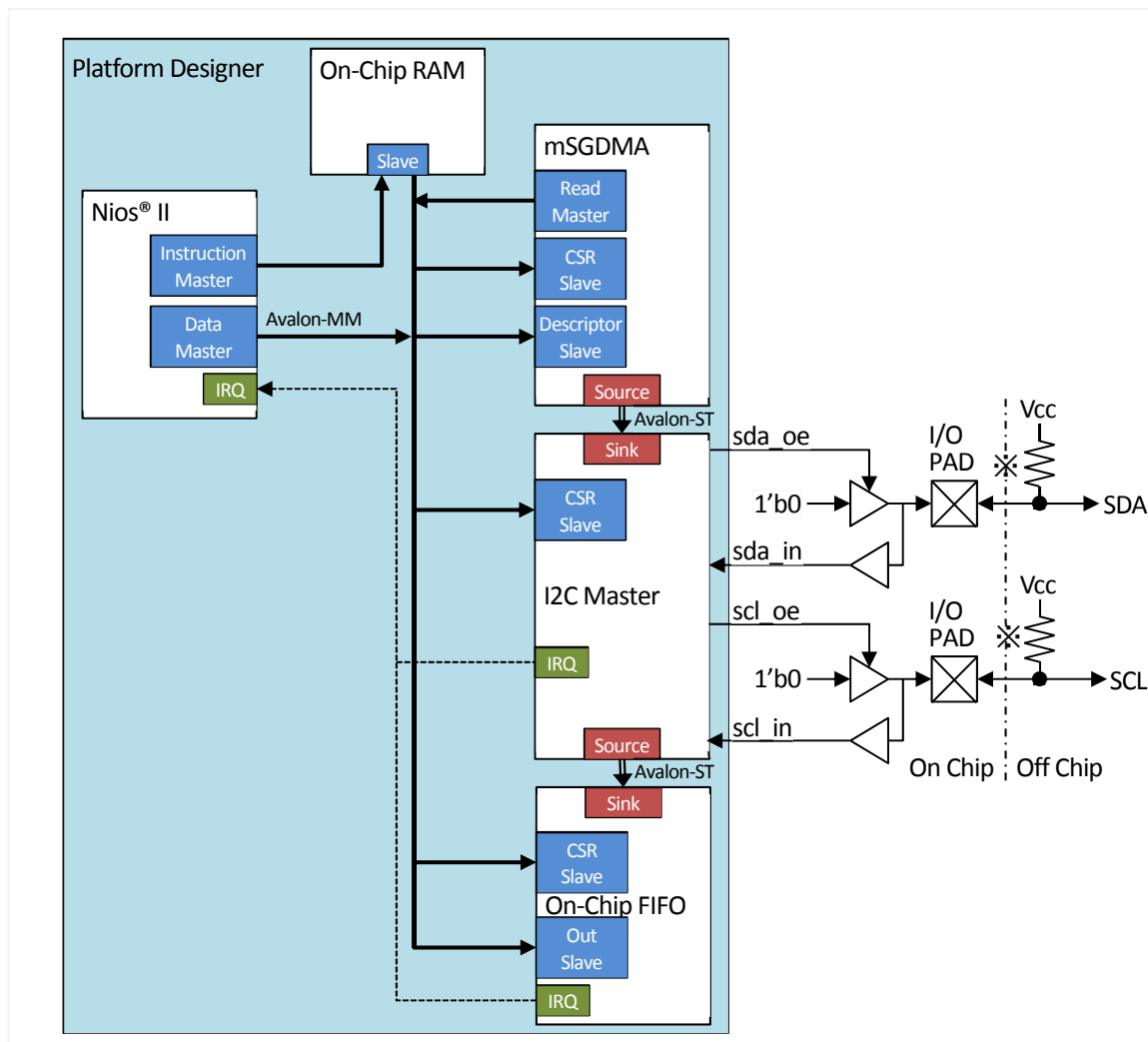
本資料では、Avalon I2C (Master) Core (以降 I2C Master)の Sink 側の Avalon-ST ポートに Modular Scatter-Gather DMA Core (以降 mSGDMA)を、Source 側の Avalon-ST ポートに On-Chip FIFO Memory Core (以降 On-Chip FIFO)を接続します。

なお、I2C Master は下記の機能を持っています。

- I2C 標準モード (100kbs) および高速モード (400kbs) に対応
- マルチ・マスターおよびクロック・ストレッチに対応
- 7-bit および 10-bit デバイス・アドレスに対応

3-2. I2C Master の使い方

本実装で使用する IP コアは、Nios II の動作環境に加え、I2C Master および mSGDMA、On-Chip FIFO の 3 つで、Platform Designer 上で Avalon-ST インターフェースで接続します。



※Pull-up 抵抗の代わりに、Quartus Prime の Assignment Editor で "Weak Pull-Up Resistor" を "On" に設定する事でも可能です。

✓	i2c_master_sda	Weak Pull-Up Resistor	On	Yes			
✓	i2c_master_scl	Weak Pull-Up Resistor	On	Yes			

I/O ポートに接続する際の上記の回路を Verilog-HDL で記載した例です。

```

module top(
    input    clock,
    input    reset_n,
    inout    i2c_master_sda,
    inout    i2c_master_scl
);

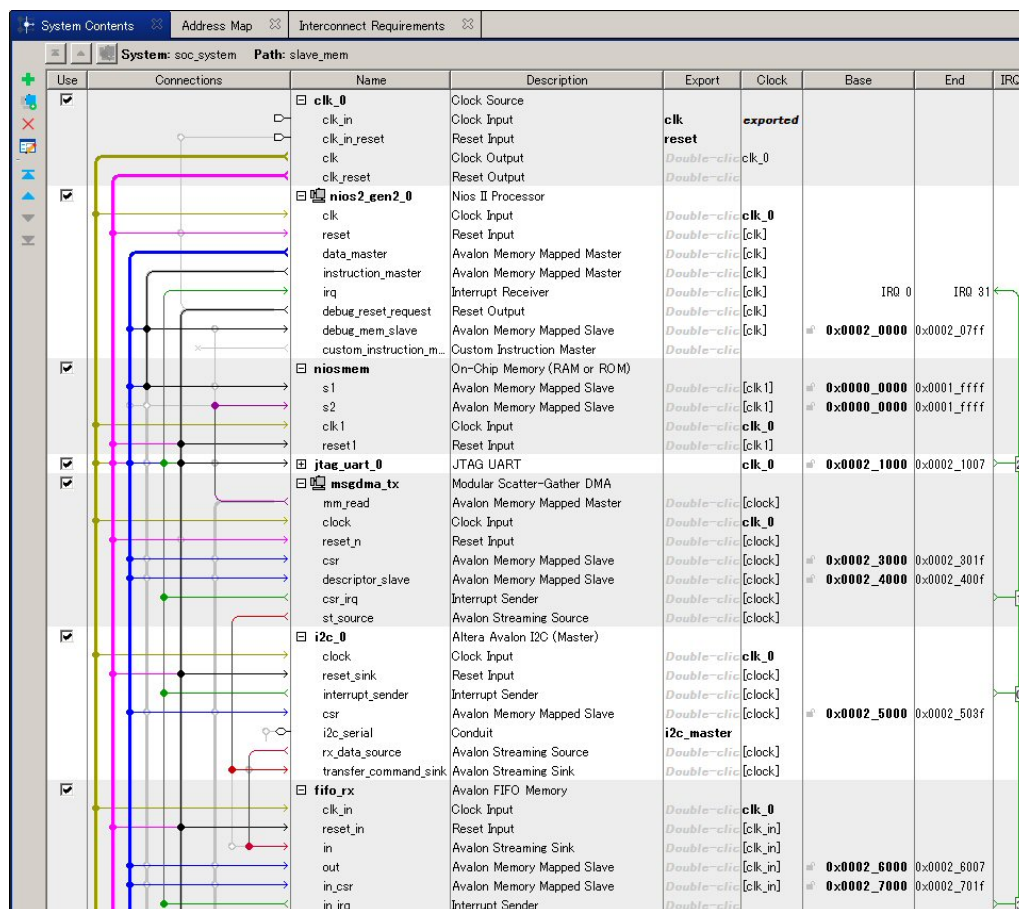
wire scl_in;
wire sda_in;
wire scl_oe;
wire sda_oe;

assign scl_in = i2c_master_scl;
assign sda_in = i2c_master_sda;
assign i2c_master_scl = (scl_oe)? 1'b0 : 1'bz;
assign i2c_master_sda = (sda_oe)? 1'b0 : 1'bz;

pd_project u0 (
    .clk_clk      (clock),
    .reset_reset_n (reset_n),
    .i2c_master_sda_in (sda_in), // i2c_master.sda_in
    .i2c_master_scl_in (scl_in),  // i2c_master.scl_in
    .i2c_master_sda_oe (sda_oe),  // i2c_master.sda_oe
    .i2c_master_scl_oe (scl_oe),  // i2c_master.scl_oe
);

endmodule
  
```

Platform Designer で接続した例です。



4. 実装

4-1. ベース・プロジェクトの準備

Nios II が動作する Quartus プロジェクトを用意し、Platform Designer で I2C Slave、mSGDMA、On-Chip FIFO を追加します。動作確認のための I2C スレーブは、I2C Slave to Avalon-MM Master Bridge Core（以降 I2C Slave）を使用します。

4-2. Platform Designer の編集

1. Platform Designer を開き、下記のモジュールを追加します。

- ・ mSGDMA (Modular Scatter-Gather DMA)
 - ・ I2C Master (Altera® Avalon- I2C (Master))
 - ・ On-Chip FIFO (Avalon-FIFO Memory)
- また、検証に使用する I2C スレーブも追加します。
- ・ I2C Slave (Altera I2C Slave To Avalon MM Master Bridge)
 - ・ On-Chip RAM (On-Chip Memory (RAM or ROM))

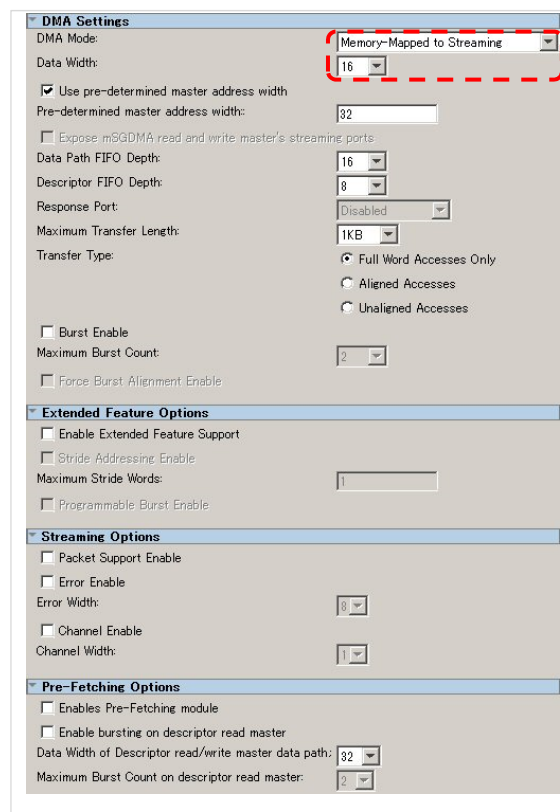
2. 追加したモジュールのパラメータを下記の設定に変更します。

【 mSGDMA 】

DMA Mode : Memory-Mapped to Streaming

Data Width : 16

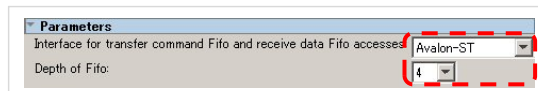
上記以外は、接続するメモリの種類、送信データの種類や長さなどを考慮して設定してください。



【 I2C Master 】

Interface for transfer command FIFO and receive data FIFO accesses : Avalon-ST

Depth of FIFO : 4



【 On-Chip FIFO 】

Allow backpressure : On

Create status interface for input : On

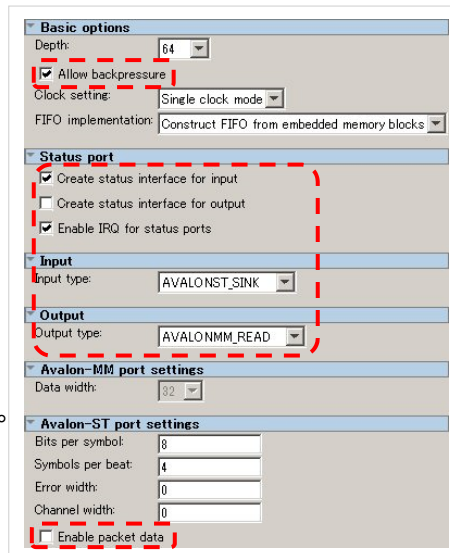
Enable IRQ for status ports : On

Input type : AVALONST_SINK

Output type : AVALONMM_READ

Enable packet data : Off

上記以外は、必要に応じて変更してください。



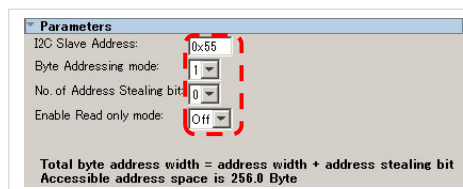
【 I2C Slave 】

I2C Slave Address : 0x55

Byte Addressing mode : 1

No. of Address Stealing bit : 0

Enable Read only mode : Off



【 On-Chip RAM 】

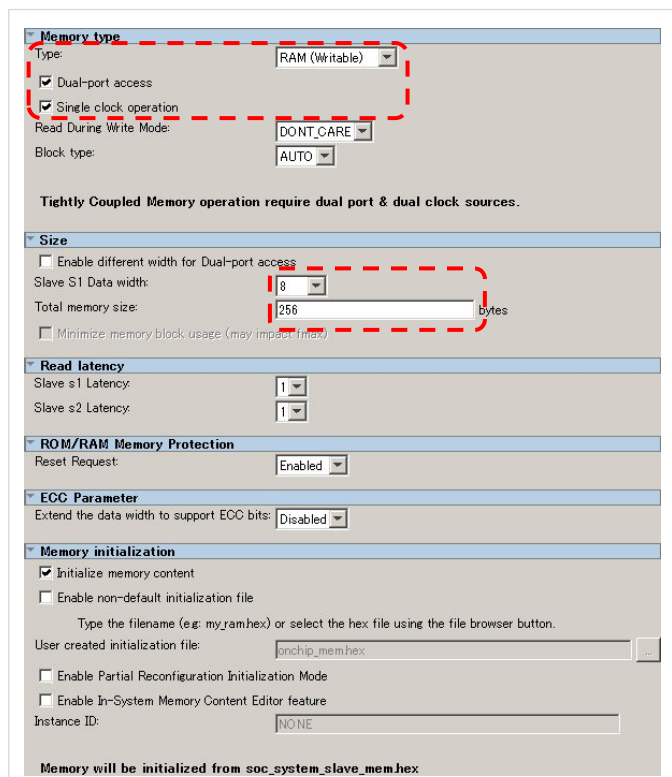
Type : RAM(Writable)

Dual-port access : On

Single clock operation : On

Slave S1 Data width : 8

Total memory size : 256



3. 各モジュールを接続し、スレーブ・アドレスと IRQ 番号を設定します。なお、必要に応じてアドレスや IRQ 番号、構成などは変更する事も可能です。

【 mSGDMA 】

ポート	接続先	ポート	スレーブ・アドレス / IRQ / Export
mm_read	Nios II のプロゲ ラムメモリ	s2 (Dual-port でない場合 s1)	
csr	Nios II	data_master	0x0002_3000
descriptor_slave	Nios II	data_master	0x0002_4000
csr_irq	Nios II	irq	2
st_source	I2C Master	transfer_command_sink	

【 I2C Master 】

ポート	接続先	ポート	スレーブ・アドレス / IRQ / Export
interrupt_sender	Nios II	irq	1
csr	Nios II	data_master	0x0002_5000
i2c_serial	外部		i2c_master
rx_data_source	On-Chip FIFO	in	
transfer_command_sink	mSGDMA	st_source	

【 On-Chip FIFO 】

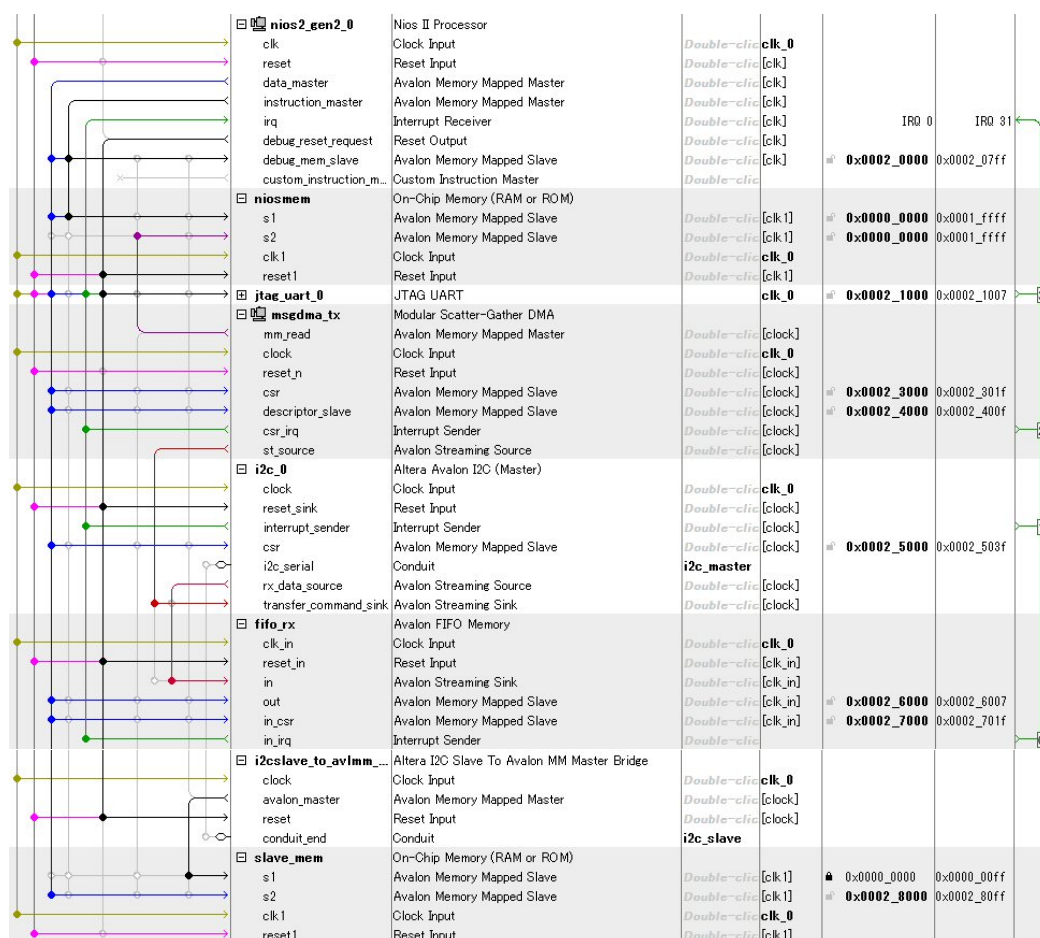
ポート	接続先	ポート	スレーブ・アドレス / IRQ / Export
in	I2C Master	rx_data_source	
out	Nios II	data_master	0x0002_6000
in_csr	Nios II	data_master	0x0002_7000
in_irq	Nios II	irq	0

【 I2C Slave 】

ポート	接続先	ポート	スレーブ・アドレス / IRQ / Export
avalon_master	On- Chip RAM	s1	
conduit_end	外部		i2c_slave

【 On-Chip RAM 】

ポート	接続先	ポート	スレーブ・アドレス / IRQ / Export
s1	i2c_slave	avalon-master	0x0000_0000
s2	Nios II	data_master	0x0002_8000



4. Platform Designer の Generate HDL を実行し、エラーが発生しない事を確認してください。

4-3. Quartus Prime の編集

1. トップ・レベル・モジュールを下記のように記述してください。

```

module i2c_test(
    input          FPGA_CLK1_50,
    input          [1:0] KEY,
    inout          i2c_master_sda,
    inout          i2c_master_scl,
    inout          i2c_slave_sda,
    inout          i2c_slave_scl
);

wire m_sda_in;
wire m_scl_in;
wire m_sda_oe;
wire m_scl_oe;

wire s_sda_in;
wire s_scl_in;
wire s_sda_oe;
wire s_scl_oe;

assign i2c_master_sda = (m_sda_oe)? 1'b0 : 1'bz;
assign m_sda_in = i2c_master_sda;
assign i2c_master_scl = (m_scl_oe)? 1'b0 : 1'bz;
assign m_scl_in = i2c_master_scl;

assign i2c_slave_sda = (s_sda_oe)? 1'b0 : 1'bz;
assign s_sda_in = i2c_slave_sda;
assign i2c_slave_scl = (s_scl_oe)? 1'b0 : 1'bz;
assign s_scl_in = i2c_slave_scl;

soc_system soc_inst (
    .clk_clk          (FPGA_CLK1_50), // Clock 50MHZ
    .reset_reset_n   (KEY[0]),        // Reset Switch

    .i2c_master_sda_in (m_sda_in),    // i2c_master.sda_in
    .i2c_master_scl_in (m_scl_in),    // .scl_in
    .i2c_master_sda_oe (m_sda_oe),    // .sda_oe
    .i2c_master_scl_oe (m_scl_oe),    // .scl_oe

    .i2c_slave_conduit_data_in (s_sda_in), // i2c_slave.conduit_data_in
    .i2c_slave_conduit_clk_in  (s_scl_in), // .conduit_clk_in
    .i2c_slave_conduit_data_oe (s_sda_oe), // .conduit_data_oe
    .i2c_slave_conduit_clk_oe  (s_scl_oe), // .conduit_clk_oe
);

endmodule

```

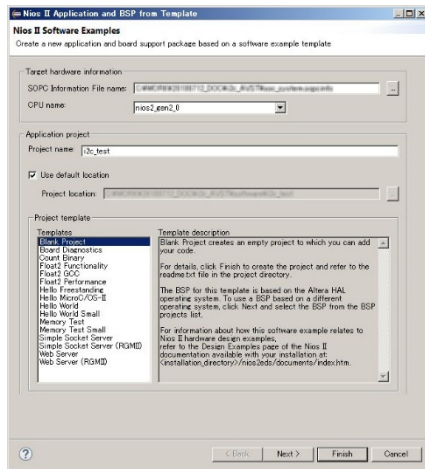
Assignment Editor でピン配置してください。

	statu	From	To	Assignment Name	Value	Enabled	Entity
1	✓		FPGA_CLK1_50	Location	PIN_V11	Yes	
2	✓		FPGA_CLK1_50	I/O Standard	3.3-V LVTTL	Yes	i2c_test
3	✓		KEY[0]	Location	PIN_AH17	Yes	
4	✓		KEY[0]	I/O Standard	3.3-V LVTTL	Yes	i2c_test
5	✓		KEY[1]	Location	PIN_AH16	Yes	
6	✓		KEY[1]	I/O Standard	3.3-V LVTTL	Yes	i2c_test
7	✓		i2c_master_scl	Location	PIN_V12	Yes	
8	✓		i2c_master_scl	I/O Standard	3.3-V LVTTL	Yes	i2c_test
9	✓		i2c_master_sda	Location	PIN_AF7	Yes	
10	✓		i2c_master_sda	I/O Standard	3.3-V LVTTL	Yes	i2c_test
11	✓		i2c_slave_scl	Location	PIN_W12	Yes	
12	✓		i2c_slave_scl	I/O Standard	3.3-V LVTTL	Yes	i2c_test
13	✓		i2c_slave_scl	Weak Pull-Up Resistor	On	Yes	i2c_test
14	✓		i2c_slave_sda	Location	PIN_AF8	Yes	
15	✓		i2c_slave_sda	I/O Standard	3.3-V LVTTL	Yes	i2c_test
16	✓		i2c_slave_sda	Weak Pull-Up Resistor	On	Yes	i2c_test
17		<<new>>	<<new>>	<<new>>			

2. Quartus Prime でコンパイルしてください。

4-4. Nios II Software Build Tool (以降 Nios II SBT) の編集

1. 新規にプロジェクトを作成してください。テンプレートは Blank Project を使用してください。



2. ソース・ファイルを新規に作成し、メイン関数を下記のように記述してください。

```

/*****
 * includes
 *****/

#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/alt_irq.h>
#include <sys/alt_cache.h>
#include <altera_avalon_i2c.h>
#include <altera_avalon_fifo.h>
#include <altera_avalon_fifo_util.h>
#include <altera_msgdma_descriptor_regs.h>
#include <altera_msgdma_csr_regs.h>
#include <altera_msgdma.h>

/*****
 * definitions (define, enum, typedef, etc..)
 *****/

#define TRUE -1
#define FALSE 0
#define I2C_SLAVE_MEM (SLAVE_MEM_BASE) // I2C Slaveのメモリアドレス
#define I2C_SLAVE_SIZE (SLAVE_MEM_SIZE_VALUE) // I2C Slaveのメモリサイズ
#define I2C_MASTER_NAME (I2C_0_NAME) // I2C Masterのデバイス名
#define FIFO_CSR (FIFO_RX_IN_CSR_BASE) // On-Chip FIFOのレジスタアドレス
#define FIFO_DATA (FIFO_RX_OUT_BASE) // On-Chip FIFOのメモリアドレス
#define FIFO_IRQ_CTRL_ID (FIFO_RX_IN_CSR_IRQ_INTERRUPT_CONTROLLER_ID) // On-Chip FIFOのIRQコントローラID
#define FIFO_IRQ (FIFO_RX_IN_CSR_IRQ) // On-Chip FIFOのIRQ番号
#define MSGDMA_CSR (MSGDMA_TX_CSR_NAME) // MSGDMAのレジスタアドレス

/*****
 * variables
 *****/

volatile int stop = FALSE; // I2C 通信停止フラグ

/*****
 * proto types
 *****/

void dump(unsigned char *adr, int size); // メモリダンプ

```

本ソース・コード内で使用する定数を再定義しています。“system.h”内の必要な定義を右側に記述してください。

```

/*****
 * interrupt handler
 *****/
// On-Chip FIFO 割り込みハンドラ
static void fifo_callback(void * context)
{
    int status;
    alt_u32 csr = (alt_u32)context;
    alt_irq_context cpu_sr;

    // 全割り込みをディセーブル
    cpu_sr = alt_irq_disable_all();

    // FIFO のステータスを読み出す
    status = altera_avalon_fifo_read_status(csr, ALTERA_AVALON_FIFO_IENABLE_ALL);

    // FULL か ALMOSTFULL なら I2C 通信停止フラグを立てる
    if(status & (ALTERA_AVALON_FIFO_STATUS_AF_MSK | ALTERA_AVALON_FIFO_STATUS_F_MSK))
    {
        stop = TRUE;
    }

    // イベントのクリア
    altera_avalon_fifo_clear_event(csr, (alt_u32)status);
    // 全割り込みをイネーブル
    alt_irq_enable_all(cpu_sr);
}

// I2C Master 割り込みハンドラ
static void i2c_callback(void * context)
{
    ALT_AVALON_I2C_DEV_t *i2c_dev = (ALT_AVALON_I2C_DEV_t *) context;
    alt_u32 status;
    alt_irq_context cpu_sr;

    // 全割り込みをディセーブル
    cpu_sr = alt_irq_disable_all();

    // I2C Masterのステータスを読み出す
    alt_avalon_i2c_int_status_get(i2c_dev, &status);

    // ステータスを出力(テスト用)
    printf("I2C Master Error Interrupt:%X\n", (int)status);

    // I2C 割り込みをディセーブル
    alt_avalon_i2c_int_disable(i2c_dev, ALT_AVALON_I2C_ISR_ALLINTS_MSK);

    // I2C 割り込みをクリア
    alt_avalon_i2c_int_clear(i2c_dev, ALT_AVALON_I2C_ISR_ALL_CLEARABLE_INTS_MSK);

    // I2C 割り込みをイネーブル
    alt_avalon_i2c_enable(i2c_dev);

    // 全割り込みをイネーブル
    alt_irq_enable_all(cpu_sr);
}

/*****
 * main function
 *****/
int main()
{
    int i;

    // mSGDMA
    alt_msgdma_dev *tx_dma;
    alt_msgdma_standard_descriptor wr_desc, rd_desc;
    int dma_status;

    // I2C Master
    ALT_AVALON_I2C_DEV_t *i2c_dev;
    ALT_AVALON_I2C_STATUS_CODE i2c_status;

    // FIFO
    int fifo_status;

    // I2C Slave memory(非キャッシュ領域でポインタ生成)
    unsigned char *slv_buff = (unsigned char*)(I2C_SLAVE_MEM | 0x80000000);

    // I2C Command
    // アドレス 0x00 に 4-byte を書き込む
    unsigned char wr_cmd[][2] = {{0x02, 0xAA}, // Start , Device Address, W/R=0
                                {0x00, 0x00}, // Write address
                                {0x00, 0xAB}, // Write Data[0]
                                {0x00, 0xBC}, // Write Data[1]
                                {0x00, 0xCD}, // Write Data[2]
                                {0x01, 0xDE}}; // Write Data[3], Stop

    // アドレス 0x00 から 4-byte を読み出す
    unsigned char rd_cmd[][2] = {{0x02, 0xAA}, // Start , Device Address, W/R=0
                                {0x01, 0x00}, // Read Address, Stop
                                {0x02, 0xAB}, // Start, Device Address, W/R=1
                                {0x00, 0x00}, // Read Data[0]
                                {0x00, 0x00}, // Read Data[1]
                                {0x00, 0x00}, // Read Data[2]
                                {0x01, 0x00}}; // Read Data[3], Stop

    // I2C Slave メモリを 0xFF でクリア(書き込みの状態を分かりやすくするため 0xFF で Fill)
    memset(slv_buff, 0xFF, I2C_SLAVE_SIZE);
}

```

On-Chip FIFO に貯まったデータの数が 10-word (main 関数内の altera_avalon_fifo_init() 関数にて ALMOSTFULL の値を 10 と設定)以上になると割り込みが生成され、本ハンドラがコールされます。本ハンドラでは FULL か ALMOSTFULL の検出で stop 変数に TRUE を設定し、main 関数内のループを中断させます。

I2C Master でエラーが発生した場合等にコールされます。本実装では、I2C の送受信は mSGDMA にて行われ、データ単位でのステータスの監視は行っておりませんので、エラーの発生は割り込みハンドラで検出します。

内部変数の初期化を行っています。
I2C のコマンドは、データ 1-byte あたり 2-byte 必要で、上位 byte には Start/Stop Condition を出力する bit データ、下位 byte に送信データを設定します。

I2C Slave のメモリは 0xFF で Fill する事で、書き込まれたデータが分かりやすくしています。


```
// === I2C Master 関連の初期化 ===
// I2C Master をオープン
i2c_dev = alt_avalon_i2c_open(I2C_MASTER_NAME);
if (NULL == i2c_dev)
{
    printf("Error:I2C Master Open Fail\n");
    return FALSE;
}

// I2C Master 初期化
alt_avalon_i2c_init(i2c_dev);
// I2C Master 割り込みハンドラの登録と割り込み有効
alt_ic_isr_register(I2C_DEV->irq_controller_ID, i2c_dev->irq_ID, i2c_callback, i2c_dev, 0x0);
alt_avalon_i2c_int_enable(i2c_dev, ALT_AVALON_I2C_ISR_ALL_CLEARABLE_INTS_MSK);
// I2C Master 起動
i2c_status = alt_avalon_i2c_enable(i2c_dev);
if (ALT_AVALON_I2C_SUCCESS != i2c_status)
{
    printf("Error:I2C Master Enable Fail\n");
    return FALSE;
}

// === On-Chip FIFO 関連の初期化 ===
// On-Chip FIFO の初期化 (10-word 蓄積されたら ALMOSTFULL 割り込み発生)
fifo_status = altera_avalon_fifo_init(FIFO_CSR, (ALTERA_AVALON_FIFO_IENABLE_AF_MSK | ALTERA_AVALON_FIFO_IENABLE_F_MSK), 1, 10);
if (ALTERA_AVALON_FIFO_OK != fifo_status)
{
    printf("Error:FIFO init Fail[%d]\n", fifo_status);
    return FALSE;
}
// On-Chip FIFO 割り込みハンドラの登録
alt_ic_isr_register(FIFO_IRQ_CTRL_ID, FIFO_IRQ, fifo_callback, (void*)FIFO_CSR, 0x0);

// === mSGDMA 関連の初期化 ===
// mSGDMA をオープン
tx_dma = alt_msgdma_open(MSGDMA_CSR);
if (NULL == tx_dma)
{
    printf("Error:TX mSGDMA Open Fail\n");
    return FALSE;
}

for(i = 1; i++)
{
    // キャッシュフラッシュ
    alt_dcache_flush_all();

    // I2C Write用ディスクリプタの登録
    dma_status = alt_msgdma_construct_standard_mm_to_st_descriptor(tx_dma, &wr_desc, (alt_u32*)wr_cmd, sizeof(wr_cmd),
        ALTERA_MSGDMA_DESCRIPTOR_CONTROL_TRANSFER_COMPLETE_IRQ_MASK);

    if(0 != dma_status)
    {
        printf("Error:DMA descriptor Fail[%d]\n", dma_status);
        return FALSE;
    }
    // I2C Master による Write コマンドの DMA 起動
    dma_status = alt_msgdma_standard_descriptor_sync_transfer(tx_dma, &wr_desc);
    if(0 != dma_status)
    {
        printf("Error:DMA async trans Fail[%d]\n", dma_status);
        return FALSE;
    }

    // I2C Read用ディスクリプタの登録
    dma_status = alt_msgdma_construct_standard_mm_to_st_descriptor(tx_dma, &rd_desc, (alt_u32*)rd_cmd, sizeof(rd_cmd),
        ALTERA_MSGDMA_DESCRIPTOR_CONTROL_TRANSFER_COMPLETE_IRQ_MASK);

    if(0 != dma_status)
    {
        printf("Error:DMA descriptor Fail[%d]\n", dma_status);
        return FALSE;
    }
    // I2C Master による Read コマンドの DMA 起動
    dma_status = alt_msgdma_standard_descriptor_sync_transfer(tx_dma, &rd_desc);
    if(0 != dma_status)
    {
        printf("Error:DMA async trans Fail[%d]\n", dma_status);
        return FALSE;
    }

    // 書き込みアドレスと値を変更
    wr_cmd[1][1] = i * 0x10; // 書き込みアドレス = i * 0x10
    wr_cmd[2][1] = i + 1; // 書き込みデータ[0] = i + 1
    wr_cmd[3][1] = i + 2; // 書き込みデータ[0] = i + 2
    wr_cmd[4][1] = i + 3; // 書き込みデータ[0] = i + 3
    wr_cmd[5][1] = i + 4; // 書き込みデータ[0] = i + 4

    // 読み出しアドレスを変更
    rd_cmd[1][1] = i * 0x10; // 読み出しアドレス = i * 0x10

    // On-Chip FIFO の割り込みで FULL か ALMOSTFULL(10) なら終了
    if(stop == TRUE)
    {
        break;
    }
}
}
```

I2C Slave の初期化と割り込みハンドラの登録を行っています。
alt_avalon_i2c_enable() の実行で、mSGDMA からデータが送り出される事で、I2C コマンドを送信します。

On-Chip FIFO の初期化と割り込みハンドラの登録を行います。

mSGDMA の初期化を行います。

I2C の送受信処理ループです。stop 変数が TRUE になるまで繰り返し処理します。

I2C Write コマンドを送信するため、ディスクリプタ・テーブルの設定と mSGDMA の起動を行います。

I2C Read コマンドを送信するため、ディスクリプタ・テーブルの設定と mSGDMA の起動を行います。

Write コマンドを変更しています。
アドレスは 0x10 単位に加算しており、データにも昇順の値を設定しています。

Read コマンドを変更しています。
アドレスを 0x10 単位に加算しています。

stop 変数が TRUE に変化したらループを抜けます。

```
// On-Chip FIFO のレベルを取得
int lev = altera_avalon_fifo_read_level(FIFO_CSR);
printf("\nRead Count:%d\n", lev);

printf("Read Data:");
// On-Chip FIFO からデータを読み出してコンソール出力
for(i = 0; i < lev; i++)
{
    int data;
    altera_avalon_read_fifo(FIFO_DATA, FIFO_CSR, &data);
    printf("%08X", data);
}

printf("\n\n=== I2C Slave Dump ===");
// I2C Slave メモリのダンプ
dump(slv_buff, I2C_SLAVE_SIZE);

return TRUE;
}
```

On-Chip FIFO に貯まっている word 数を取得して、FIFO から読み出してコンソールに出力します。

I2C Slave メモリのダンプを出力します。

```
/* sub function
=====
void dump(unsigned char *adr, int size)
{
    int i;
    unsigned char ucData;

    // キャッシュフラッシュ
    alt_dcache_flush_all();

    printf("\n0000: ");
    for(i = 0; i < size; i++)
    {
        ucData = adr[i];
        if((i % 16 == 15) && (i < size - 1))
        {
            printf("%02X \n%04X: ", ucData, i + 1);
        }else
        {
            printf("%02X ", ucData);
        }
    }
    printf("\n");
}
```

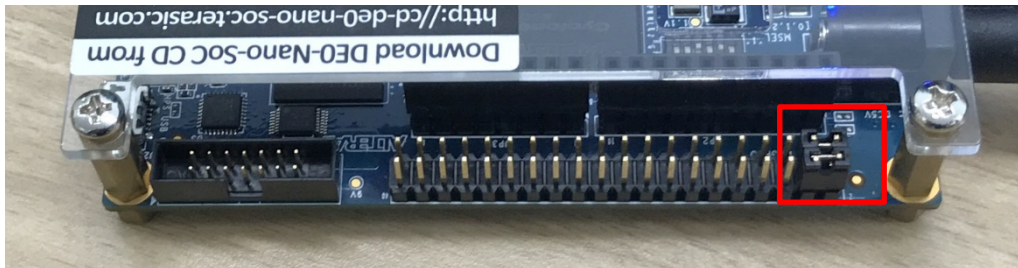
メモリ・ダンプ用の関数です。

3. Buildしてエラーが無い事を確認してください。

5. 検証

5-1. 動作の確認

1. GPIO-0 の Pin-1 と Pin-3 、Pin-2 と Pin-4 をジャンパ等で接続してください。



2. ダウンロード・ケーブルで、SOF ファイルを書き込み、Nios II を Run させてください。
3. Nios II SBT の Nios II Console に下記が出力されれば正常に動作しています。

```

Problems Tasks Console Nios II Console Properties
I2c_test Nios II Hardware configuration - cable: DE-SoC on localhost [USB-1] device ID: 2 instance ID: 0 name: #aguart_0

Read Count:10
Read Data:DECDBACAB,05040302,06050403,07060504,08070605,09080706,0A090807,0B0A0908,0C0B0A09,0D0C0B0A,

=== I2C Slave Dump ===
0000: AB BC CD DE FF FF FF FF FF FF FF FF FF FF
0010: 02 03 04 05 FF FF FF FF FF FF FF FF FF FF
0020: 03 04 05 06 FF FF FF FF FF FF FF FF FF FF
0030: 04 05 06 07 FF FF FF FF FF FF FF FF FF FF
0040: 05 06 07 08 FF FF FF FF FF FF FF FF FF FF
0050: 06 07 08 09 FF FF FF FF FF FF FF FF FF FF
0060: 07 08 09 0A FF FF FF FF FF FF FF FF FF FF
0070: 08 09 0A 0B FF FF FF FF FF FF FF FF FF FF
0080: 09 0A 0B 0C FF FF FF FF FF FF FF FF FF FF
0090: 0A 0B 0C 0D FF FF FF FF FF FF FF FF FF FF
00A0: 0B 0C 0D 0E FF FF FF FF FF FF FF FF FF FF
00B0: 0C 0D 0E 0F FF FF FF FF FF FF FF FF FF FF
00C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF
  
```

6. 補足

6-1. 注意事項

- On-Chip FIFO の Avalon-MM ポートは 32-bit のため 4-byte 単位でのアクセスとなります。4-byte 未満のデータの読出しには注意が必要です。
- I2C Master の Avalon-ST Sink ポートには、Start of Packet や End of Packet 信号がありませんので、そのままでは Start of Packet や End of Packet 信号を出力するモジュールには接続できません。しかし、これらの信号を、Avalon-ST Sink の 16-bit の任意の bit に割り当てる回路を挿入する事で、それらのモジュールに接続する事が可能です。

7. 参考資料

- Embedded Peripherals IP User Guide
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf
- インテル® FPGA の開発フロー／FPGA トップページ
<https://service.macnica.co.jp/library/109705>
- Nios II 技術資料
https://service.macnica.co.jp/library/list?sort%5Blibrary_publish_at2_d%5D=desc&tag=Nios+II
- Nios II FAQ
https://service.macnica.co.jp/support/faq/list?sort%5Bfaq_publish_at2_d%5D=desc&tag=Nios+II

改版履歴

Revision	年月	概要
1	2018 年 8 月	初版

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。
株式会社マクニカ アルティマ カンパニー <https://www.alt.macnica.co.jp/> 技術情報サイト アルティマ技術データベース <http://www.altima.jp/members/>
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。