Nios® II
UART の活用術
DMA との結合でソフトウェア負荷軽減

Ver.17.1

2020 年 5 月 Rev.2

Nios® II – UART の活用術　DMA との結合でソフトウェアの負荷軽減

目次

1. はじめに

Nios® II による UART 通信を UART Core を使って行う場合、送受信の処理はソフトウェアで行わなければなりません。しかし、UART 通信が頻繁に行われるようなアプリケーションの場合、この送受信処理の負荷が本来行わなければいけない処理を圧迫したり、受信処理が間に合わず受信データの取りこぼしなどの問題が発生する場合があります。そこで、送受信処理をハードウェアに実装することで、ソフトウェアによる送受信処理を最小限に抑え、受信データの取りこぼしなどの問題を解決する手段をご提案します。

本資料は、UART Core に Avalon® Streaming Interface（以降、 Avalon-ST）を接続できるアダプタ・モジュールを作成し、それをどのように構成、制御するかを説明した資料になります。設定済のモジュールやソース・ファイルなどをそのまま利用する事で簡単に確認ができますので、下記のサイトから入手してお試しください。

Nios® II はじめてガイド - UART 活用術 DMA との結合でソフトウェア負荷軽減

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。

※ Quartus® Prime Standard Edition Version 17.1.1

※ Nios® II Software Build Tools for Eclipse Version 17.1.1

※ 17.1.1 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）

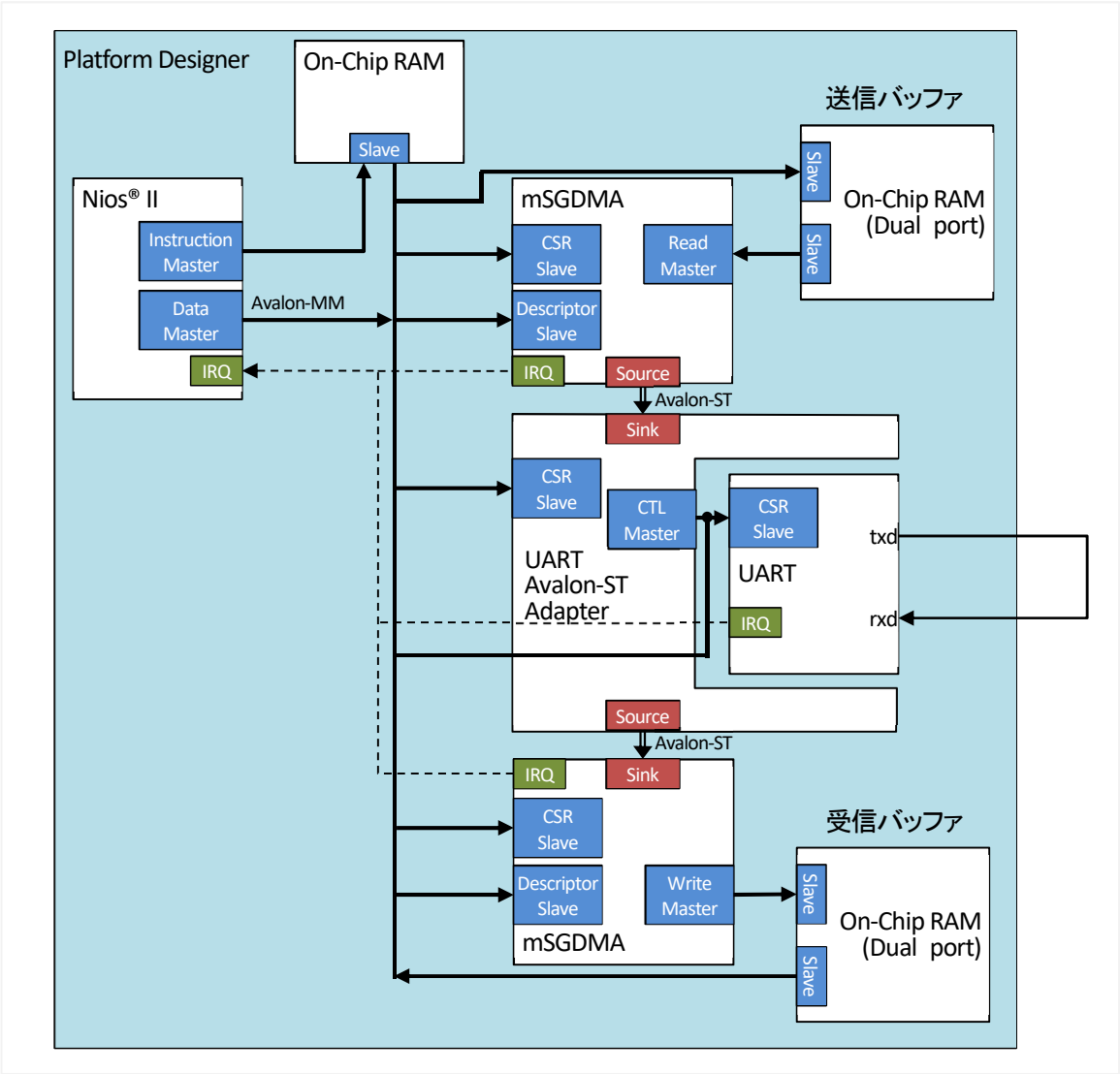
3. 仕様

3-1. 機能

UART Core には Avalon® Memory Mapped Interface（以降、 Avalon-MM）の Slave ポートが用意されており、このポートのレジスタにアクセスする事でデータの送受信を行います。本資料では、このレジスタにハードウェアからアクセスし、送信データと受信データを Avalon-ST ポートにストリーム・データとして入出力できるようにする事で、DMA や FIFO などを利用する事を可能にしています。

3-2. UART to Avalon-ST アダプタの使用例

本資料で作成するアダプタ・コアは、Nios® II の動作環境に加え、UART Core（以降、UART ）および Modular Scatter-Gather DMA Core（以降、mSGDMA）、On-Chip Memory (RAM and ROM) Core（以降、On-Chip RAM）の 3 つで、Platform Designer 上に実装します。

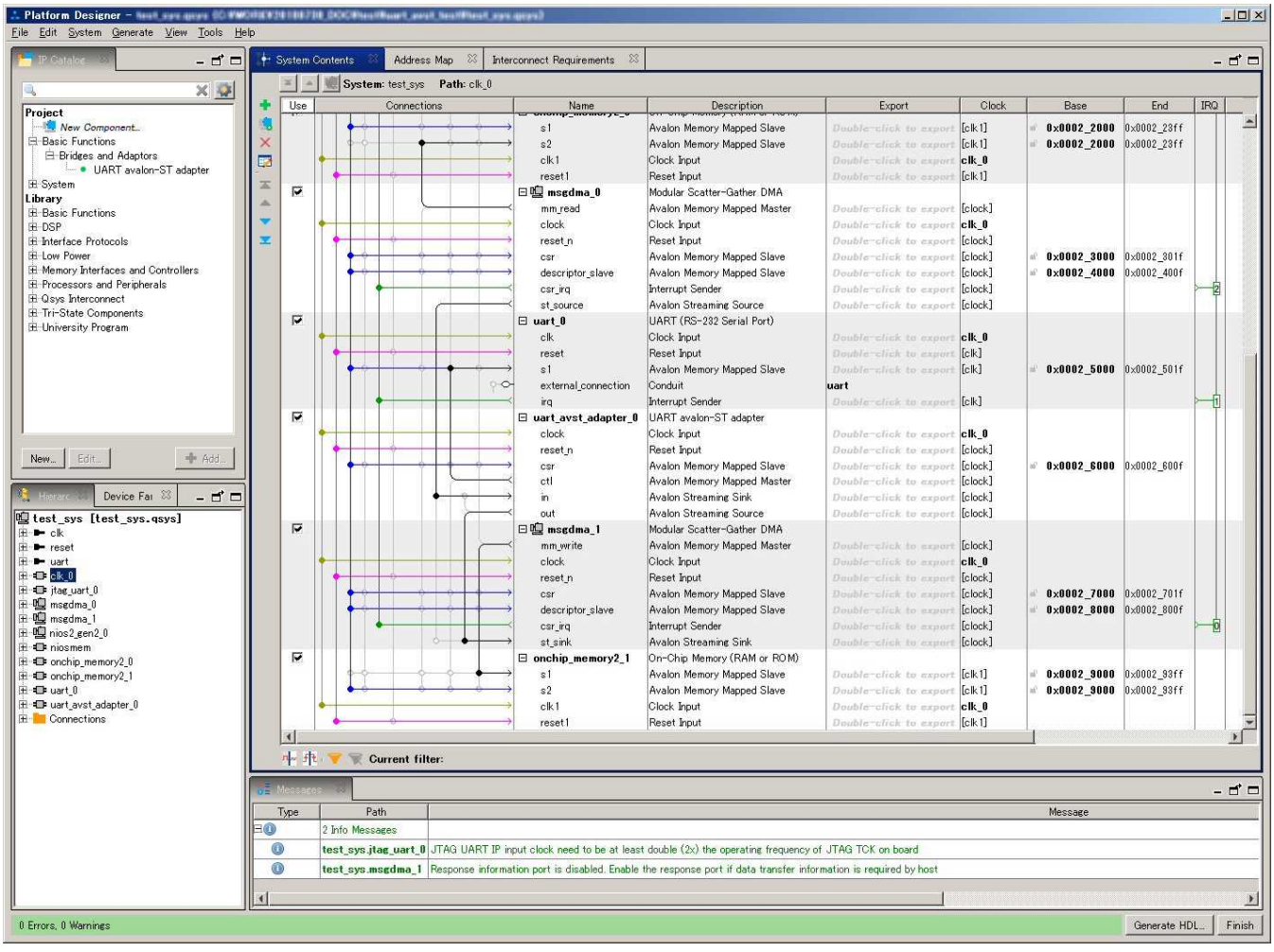


※ UART からの txd および rxd は I/O Pin に接続し、基板の外でジャンパ線等で繋ぐようにし、このジャンパ線を切断する事でエラー発生時の動作が確認できるようにしています。

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

```
module uart_test(
    input           FPGA_CLK1_50,
    input   [1:0]   KEY,
    output          txd,
    input           rxd
);

wire    clk_100;
wire    reset_n;

pll pll_inst (
    .refclk     (FPGA_CLK1_50),     // Input Clock(50MHz)
    .rst        (!KEY[0]),          // Reset
    .outclk_0   (clk_100),          // Output Clock(100MHz)
    .locked     (reset_n)           // PLL Locked
);

test_sys u_test_sys (
    .clk_clk        (clk_100),
    .reset_reset_n  (reset_n),

    .uart_rxd       (rxd),
    .uart_txd       (txd)
);

endmodule
```

Platform Designer で接続した例です。詳細については次章以降で説明しています。



4. 実装

4-1. UART Avalon-ST adapter モジュールの組み込み

UART Avalon-ST Adapter のサンプル・コードを下記に示します。

```
module uart_avst_adapter
#(
    parameter DATA_WIDTH    = 8,    // Avalon-ST Data bit width
    parameter SLAVE_ADDRESS = 0     // UART module CSR address
)
(
    input   wire                        clk             ,   // clock
    input   wire                        reset_n         ,   // reaset_n
    // IN:Avalon-ST Sink
    output  wire                        in_ready        ,   // Sink ready
    input   wire                        in_valid        ,   // Sink valid
    input   wire    [DATA_WIDTH-1:0]    in_data         ,   // Sink data
    // OUT:Avalon-ST Source
    input   wire                        out_ready       ,   // Source ready
    output  wire                        out_valid       ,   // Source valid
    output  wire    [DATA_WIDTH-1:0]    out_data        ,   // Source data
    // CTL:Avalon-MM Master
    output  reg     [31:0]              ctl_address     ,   // Control port address
    output  reg                         ctl_write       ,   // Control port write
    output  reg     [15:0]              ctl_writedata   ,   // Control port writedata
    output  reg                         ctl_read        ,   // Control port read
    input   wire    [15:0]              ctl_readdata    ,   // Control port readdata
    input   wire                        ctl_waitrequest ,   // Control port wait request
    // CSR:Avalon-MM Slave
    input   wire    [1:0]               csr_address     ,   // Register address
    input   wire                        csr_write       ,   // Register write
    input   wire    [31:0]              csr_writedata   ,   // Register writedata
    input   wire                        csr_read        ,   // Register write
    output  reg     [31:0]              csr_readdata        // Register writedata
);

    // Local parameters.
    localparam INTERVAL     = 32'd100;  // clk / 100

    localparam REG_RX_DATA  = 32'h0 + SLAVE_ADDRESS;
    localparam REG_TX_DATA  = 32'h4 + SLAVE_ADDRESS;
    localparam REG_STATUS   = 32'h8 + SLAVE_ADDRESS;

    localparam ST_IDLE      = 4'd0;     // Idle
    localparam ST_RD_REG    = 4'd1;     // Read status register
    localparam ST_RD_CMP    = 4'd2;     // Read status register is completed

    localparam ST_TX_REG    = 4'd3;     // Receive transmit data from avalon-ST sink
    localparam ST_TX_WAIT   = 4'd4;     // Write transmit data register
    localparam ST_TX_CMP    = 4'd5;     // Wait write latency

    localparam ST_RX_REG    = 4'd6;     // Read receive data register
    localparam ST_RX_WAIT   = 4'd7;     // Wait read latency
    localparam ST_RX_CMP    = 4'd8;     // Transmit receive data to avalon-ST source

    reg     [1:0]               r_csr_state;
    reg     [31:0]              r_csr_timeout;
    reg     [31:0]              r_timer;
    reg     [3:0]               r_state;
    reg                         r_rrdy;
    reg     [DATA_WIDTH-1:0]    r_tx_data;
    reg     [DATA_WIDTH-1:0]    r_rx_data;
    reg     [31:0]              r_tx_count;
    reg     [31:0]              r_rx_count;

    wire                        w_tx_enable;
    wire                        w_rx_enable;
    wire    [15:0]              w_tx_data;
    wire    [DATA_WIDTH-1:0]    w_rx_data;

    assign  in_ready = (r_state == ST_TX_REG)? 1'b1 : 1'b0;
    assign  out_valid = ((r_state == ST_RX_CMP) & out_ready)? 1'b1 : 1'b0;
    assign  out_data = r_rx_data;
    assign  w_tx_enable = r_csr_state[0];
    assign  w_rx_enable = r_csr_state[1];

    generate begin
    if (DATA_WIDTH > 8) begin
        assign w_tx_data = {r_tx_data[7:0],    r_tx_data[15:8]};    // Write transmit data(9-bit ~ 16-bit) Little endian
        assign w_rx_data = {ctl_readdata[7:0], ctl_readdata[15:8]}; // Latch receive data(9-bit ~ 16-bit) Little endian
    end else begin
        assign w_tx_data = {{(16 - DATA_WIDTH){1'b0}}, r_tx_data};  // Write transmit data(7,8-bit)
        assign w_rx_data = ctl_readdata[DATA_WIDTH-1:0];            // Latch receive data(7,8-bit)
    end
    end endgenerate
```

インターフェースに関しては次の章で説明します

周期タイマーの初期値
clk を INTERVAL で分周したものがレジスタのポーリング周期となります

レジスタへのアクセスは、パラメータの SLAVE_ADDRESS を加算したアドレスとなります

ステート・マシン・コードの定義

CSR の Reg'-0 の Bit-0 が 1 で Tx 有効、Bit-1 が 1 で Rx 有効

UART の Data Bits が 9-bit の時、Avalon-ST のデータ幅は 16-bit となりますが、その場合、データは Little endian となりますので、byte 反転

```
    // CSR(Avalon-MM Slave)
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            r_csr_state   <= 2'b00;
            r_csr_timeout <= INTERVAL;
        end else begin
            if(csr_write) begin
                r_csr_state   <= (csr_address == 2'd0)? csr_writedata[1:0] : r_csr_state;   // Status/Control reg'
                r_csr_timeout <= (csr_address == 2'd1)? csr_writedata      : r_csr_timeout; // Interval reg'
            end
            if(csr_read) begin
                csr_readdata <= (csr_address == 2'd0)? {30'b0, r_csr_state} :  // Status/Control reg'
                                (csr_address == 2'd1)? r_csr_timeout :         // Interval reg'
                                (csr_address == 2'd2)? r_tx_count :            // Transmit counter
                                (csr_address == 2'd3)? r_rx_count : 32'b0;     // Receive counter
            end
        end
    end

    // Interval counter
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            r_timer  <= 32'd0;
        end else begin
            if(r_timer >= r_csr_timeout) begin
                r_timer  <= 32'd0;
            end else begin
                r_timer <= r_timer + 32'd1;
            end
        end
    end

    // State machine
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            r_state         <= ST_IDLE;
            ctl_address     <= REG_STATUS;
            ctl_read        <= 1'b0;
            ctl_write       <= 1'b0;
            ctl_writedata   <= 16'h0000;
            r_rrdy          <= 1'b0;
            r_rx_data       <= {(DATA_WIDTH){1'b0}};
            r_tx_data       <= {(DATA_WIDTH){1'b0}};
        end else begin
            // Clear Tx/Rx counted if disable at Rx or Tx
            r_tx_count <= (w_tx_enable)? r_tx_count : 32'b0;
            r_rx_count <= (w_rx_enable)? r_rx_count : 32'b0;
            // The processing execution is canceled.
            if(~w_tx_enable & ~w_rx_enable) begin
                r_state <= ST_IDLE;
            end else
            case(r_state)
                // Idle[0]
                ST_IDLE : begin
                    r_state <= (r_timer == 32'd0)? ST_RD_REG : ST_IDLE; // Wait time-out
                end

                //=============================================================
                // The status register of a communication module is read.
                //=============================================================
                // Read status register[1]
                ST_RD_REG : begin
                    if(!ctl_waitrequest) begin      // waitreq:Lo?
                        ctl_address <= REG_STATUS;  // STATUS reg address
                        ctl_read <= 1'b1;           // read=Hi
                        r_state <= ST_RD_CMP;       // Next state
                    end
                end

                // Wait read latency[2]
                ST_RD_CMP : begin
                    if(!ctl_waitrequest) begin      // waitreq:Lo?
                        ctl_read <= 1'b0;           // read=Lo
                        r_rrdy  <= (w_rx_enable & ctl_readdata[7])? 1'b1 : 1'b0;     // La
                        r_state <= (w_tx_enable & ctl_readdata[6])? ST_TX_REG :      // Tx enable & trdy:Hi?
                                   (w_rx_enable & ctl_readdata[7])? ST_RX_REG :      // Rx
                                                                    ST_IDLE;        // if both are Lo return to IDLE
                    end
                end

                //=============================================================
                // Transmit process
                //=============================================================
                // Receive transmit data from avalon-ST sink[4]
                ST_TX_REG : begin
                    if(in_valid) begin              // sink valid enable
                        r_tx_data <= in_data;       // Latch data
                        r_state <= ST_TX_WAIT;      // Next state
                    end else begin
                        r_state <= (r_rrdy)? ST_RX_REG : ST_IDLE; // rredy=Hi? Receive pr
                    end
                end

                // Write transmit data register[5]
                ST_TX_WAIT : begin
                    if(!ctl_waitrequest) begin      // waitreq:Lo?
                        ctl_address <= REG_TX_DATA; // Transmit Data reg address
                        ctl_write <= 1'b1;          // write=Hi
                        ctl_writedata <= w_tx_data; // Write transmit data
                        r_state <= ST_TX_CMP;       // Next state
                    end
                end

                // Wait write latency[7]
                ST_TX_CMP : begin
                    if(!ctl_waitrequest) begin      // waitreq:Low?
                        ctl_write <= 1'b0;          // write=Lo
                        r_tx_count <= r_tx_count + 32'd1;          // Count tx data
                        r_state <= (r_rrdy)? ST_RX_REG : ST_IDLE;  // rredy=Hi? Receive process / Lo IDLE
                    end
                end
```

CSR アクセス
Reg'-0: 制御レジスタ
Reg'-1: インターバル・カウント
Reg'-2: 送信データ数カウンタ
Reg'-3: 受信データ数カウンタ
※Reg'-0,1 は R/W、Reg'-2,3 は Read Only

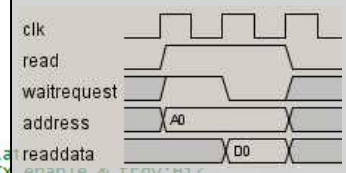
ポーリング周期カウント
Reg'-1 のインターバル・カウント値になるまでカウント・アップ、インターバル・カウント値になると 0 クリア

状態遷移図は次の章で説明しています

Tx および Rx が無効になると送信/受信データカウントをクリア

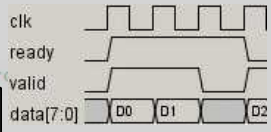
カウンタが 0 になると ST_RD_REG に遷移

UART のステータス・レジスタを読み出して、TRDY（送信可能）と RRDY（受信完了）により送信データ書込みのステートに遷移するか、受信データ読出しのステートに遷移

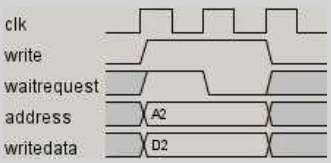


[Avalon-MM Timing Chart]

送信処理のステート
Avalon-ST Sink からデータを入力し、UART の 送信データレジスタに書き込む



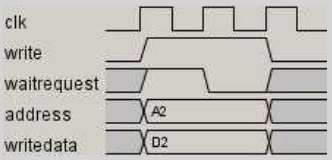
[Avalon-ST Timing Chart]



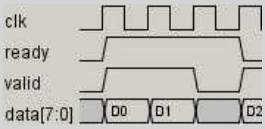
[Avaon-MM Timing Chart]

```
//=============================================================
// Receive process
//=============================================================
// Read receive data register[8]
ST_RX_REG : begin
    if(!ctl_waitrequest) begin      // waitreq:Low?
        ctl_address <= REG_RX_DATA; // Receive Data reg address
        ctl_read <= 1'b1;           // read=Hi
        r_state <= ST_RX_WAIT;      // Next state
    end
end

// Wait read latency[9]
ST_RX_WAIT : begin
    if(!ctl_waitrequest) begin      // waitreq:Low?
        ctl_read <= 1'b0;           // read=Lo
        r_rx_data <= w_rx_data;     // Latch receive data
        r_state <= ST_RX_CMP;       // Next state
    end
end

// Transmit receive data to avalon-ST source[11]
ST_RX_CMP : begin
    if(out_ready) begin             // source ready:Hi?
        r_rx_count <= r_rx_count + 32'd1;// Count rx data
        r_state <= ST_IDLE;         // Return to IDLE
    end
end

default : begin
    r_state <= ST_IDLE;
end
endcase
end
end
endmodule
```

受信処理のステート
UART の受信データレジスタから読出し、Avalon-ST Source にデータを出力



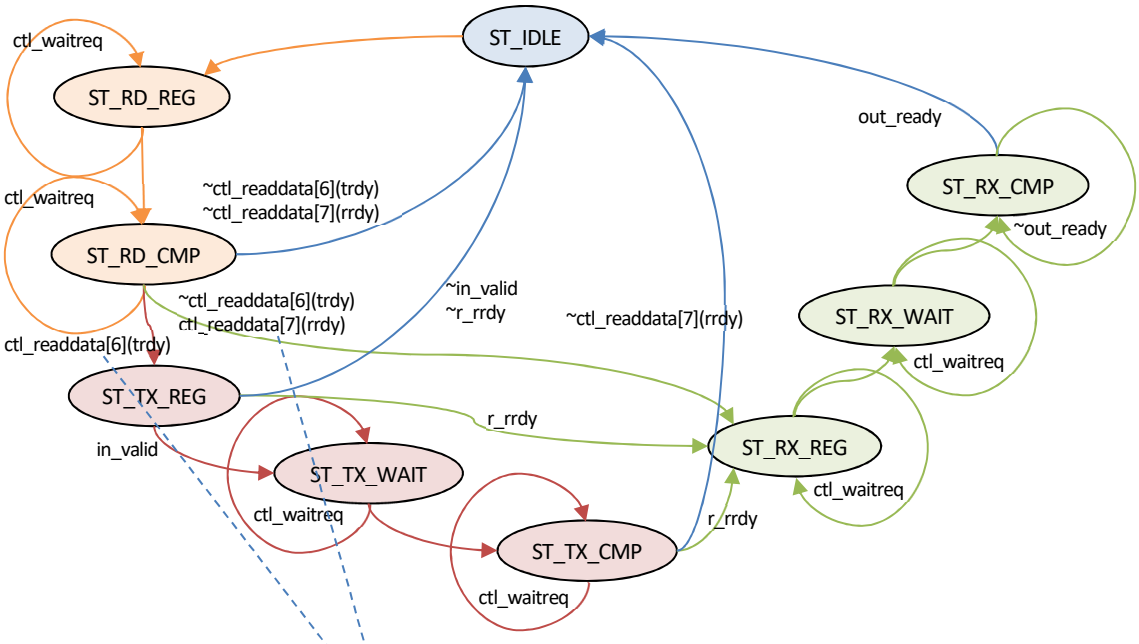
[Avalon-MM Timing Chart]



[Avalon-ST Timing Chart]

4-1-1. 状態遷移図

UART Avalon-ST Adapter の中のステート・マシンの状態遷移を下記に示します。



Offset	Register Name	R/W	Description/Register Bits													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	Reserved					(10)	(10)	Receive Data						
1	txdata	WO	Reserved					(10)	(10)	Transmit Data						
2	status (9)	RW	Reserved	eop		cts	dcts	e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	Reserved	ieop		rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor (11)	RW	Baud Rate Divisor													
5	endof-packet (11)	RW	Reserved					(10)	(10)	End-of-Packet Value						

UART Code レジスタ・マップ

4-1-2. レジスタ・マップ

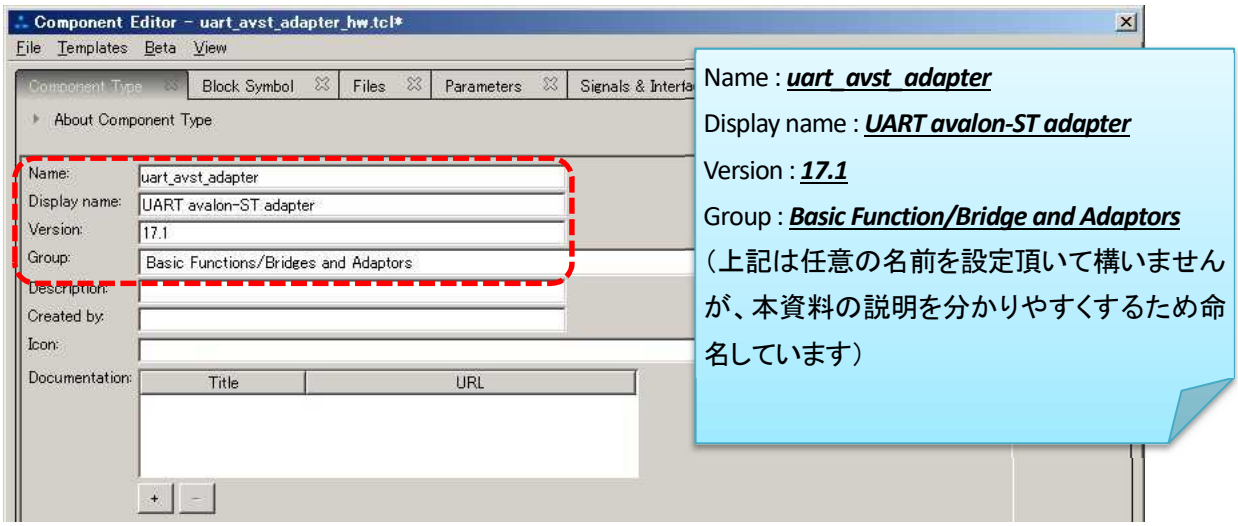
UART Avalon-ST Adapter のレジスタ・マップを下記に示します。

オフセット	レジスタ名	R/W	ビット・アサイン	使用方法
0	Control	RW	31‥2 / 1 / 0 Reserved / Rx / Tx	Bit-0 に 1 を設定すると、送信に本モジュールが動作し、0 を設定すると動作を停止します。 Bit-1 に 1 を設定すると、受信に本モジュールが動作し、0 を設定すると動作を停止します。
1	Interval count	RW	31‥0 Count	動作周期を設定します。 本モジュールに供給しているクロックの周波数を、本レジスタで設定した値で分周して起動します。
2	Tx data count	RO	31‥0 Count	送信データ数を示します。 本レジスタをクリアするには、Control レジスタの bit-0 に 0 を設定してください。
3	Rx data count	RO	31‥0 Count	受信データ数を示します。 本レジスタをクリアするには、Control レジスタの bit-1 に 0 を設定してください。

4-2. IP Catalog への組み込み

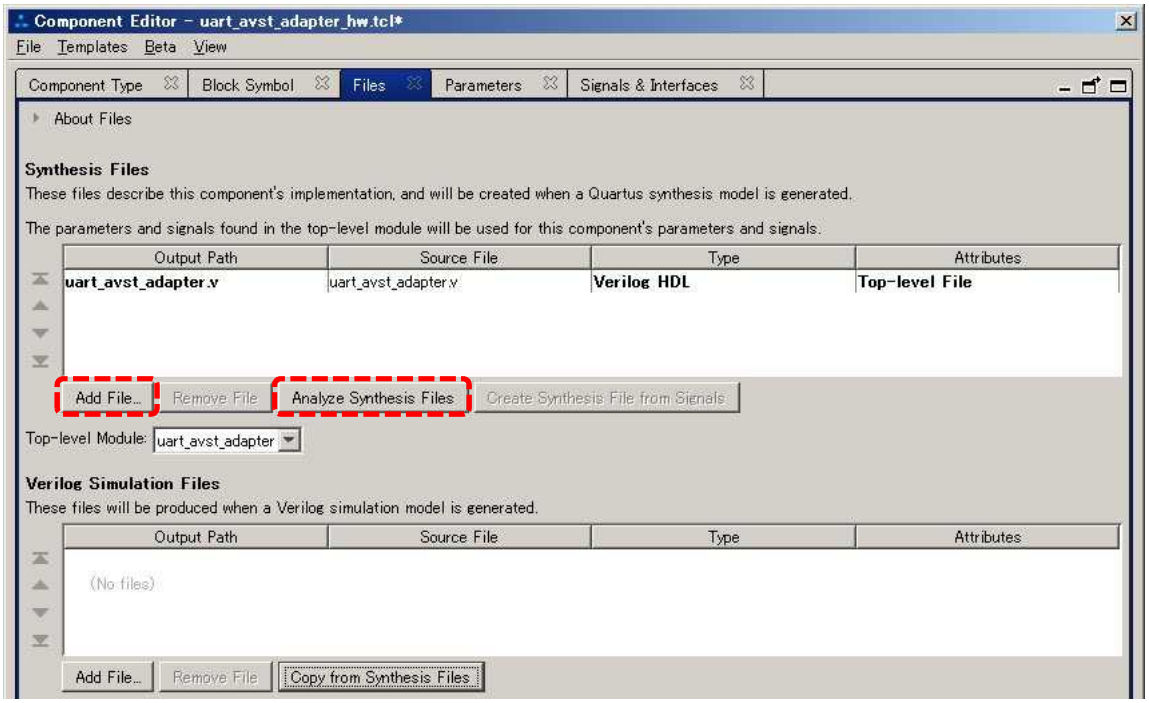
Platform Designer の IP Catalog へは、Component Editor を使って組み込みます。

1. New Component Editor を起動して、**Component Type** タブで **Name** および **Display name** 、**Version**※1 **Group** を設定します。

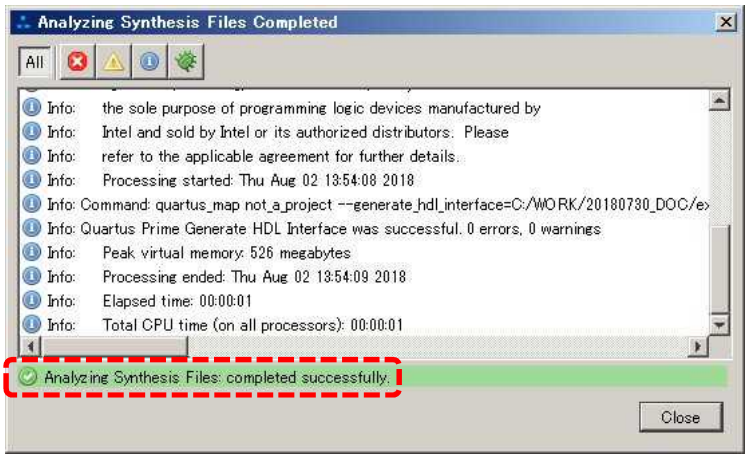


※1 Nios® II Software Build Tools for Eclipse（以降、Nios II SBT ）でプロジェクトを作成する際 BSP Editor での Generate 時、本モジュール用のヘッダファイルを自動で読み込むために uart_avst_adapter_sw.tcl を使用しますが、ツール・バージョンが 7.1 未満の場合エラーとなりますので、7.1 以上にする必要があります。

2. **Files** タブで **Synthesis Files** を設定します。



[Add File...] をクリックしてファイルを登録し、**[Analyze Synthesis Files]** をクリックしてコードのチェックを行います。

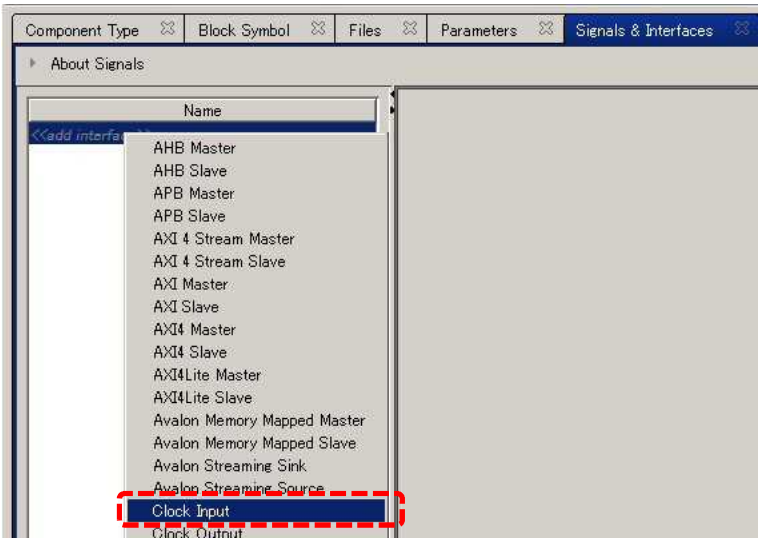


"**Completed successfully**"が表示されれば正常です。

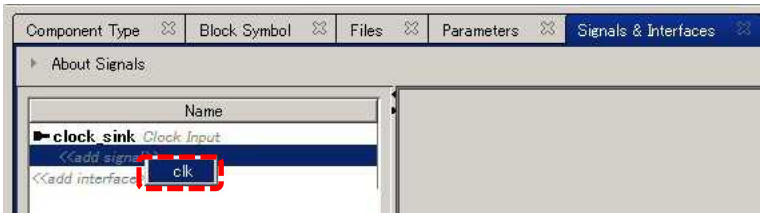
※ シミュレーションを行う場合は **[Copy from Synthesis Files]** をクリックして、 **Verilog Simulation Files** のリストに同じファイルを追加してください。

3. **Signal & Interface** タブで **<<add interface>>** をクリックして **Clock Input** を選択します。

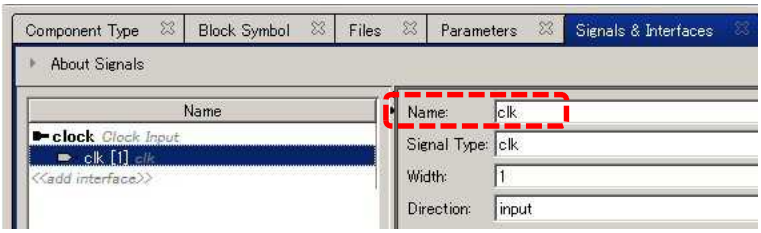
※ Component Editor は前項 2. の操作で、自動でインターフェースを作成しますが、正しく割り当てられない場合がありますので、"Remove" で、一旦全てのインターフェースを削除してから作業を進める事をお勧めします。



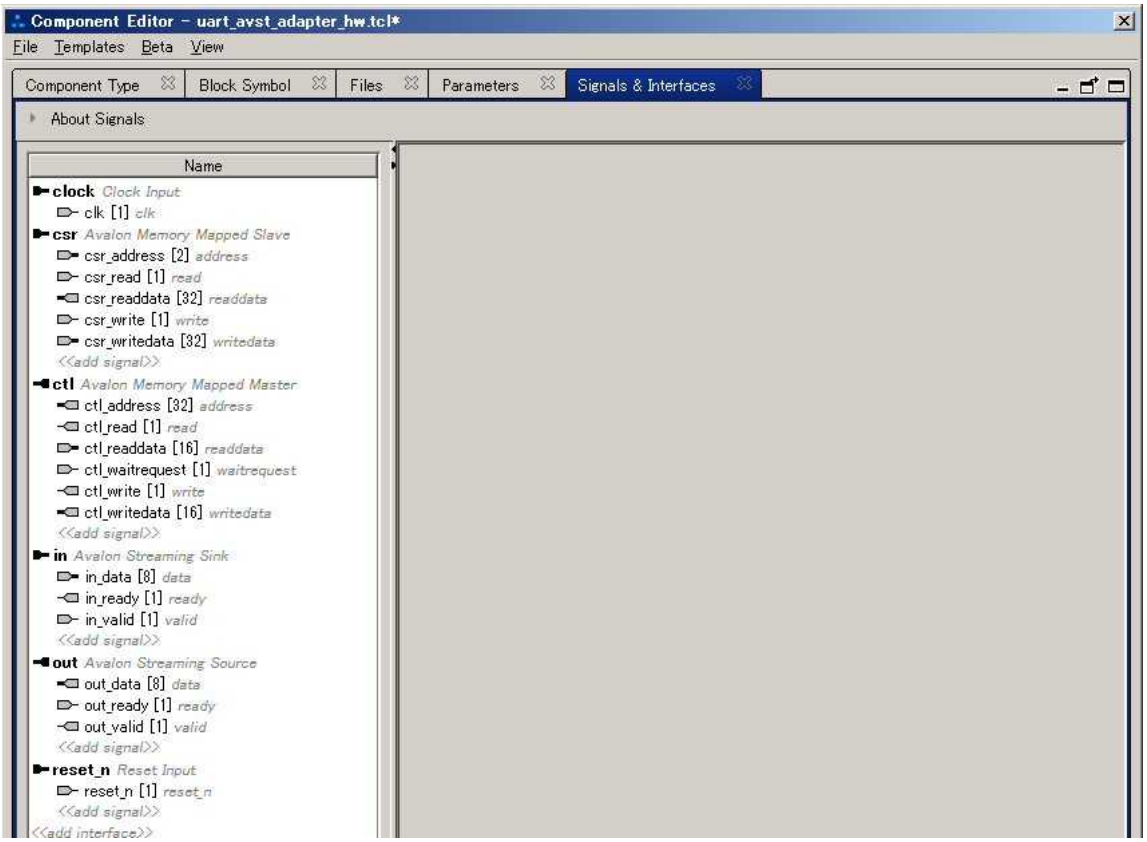
4. 作成された **clock_sink** の下の **<<add signal>>** をクリックして **clk** を選択します。



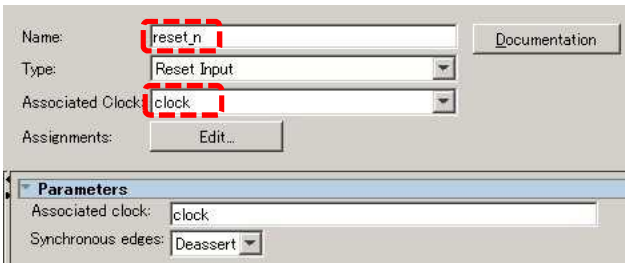
5. 作成された **clock_sink** や **clock_sink_clk** の名前は変更可能です。**Name** をそれぞれ ”clock” と ”clk” に変更してください。



6. 3. から 5. の手順で下記のように、 reset_n , csr , ctl , in , out インターフェースを作成します。



A) reset_n (Reset Input) の設定



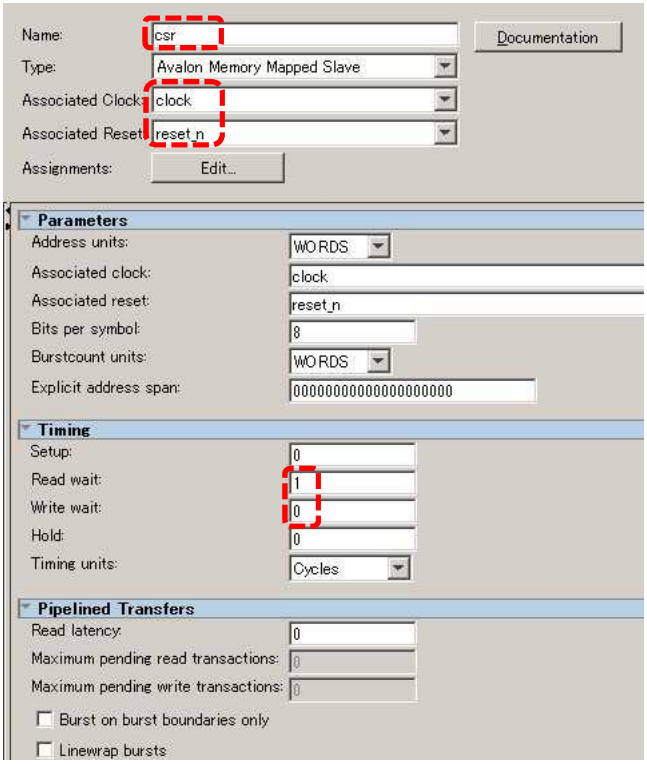
Signal Type	Name	Width	Direction
reset_n	reset_n	1	input

Name : ***reset_n***

Associated Clock : ***clock***

上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

B) csr (Avalon Memory Mapped Slave) の設定



Name : ***csr***

Associated Clock : ***clock***

Associated Reset : ***reset_n***

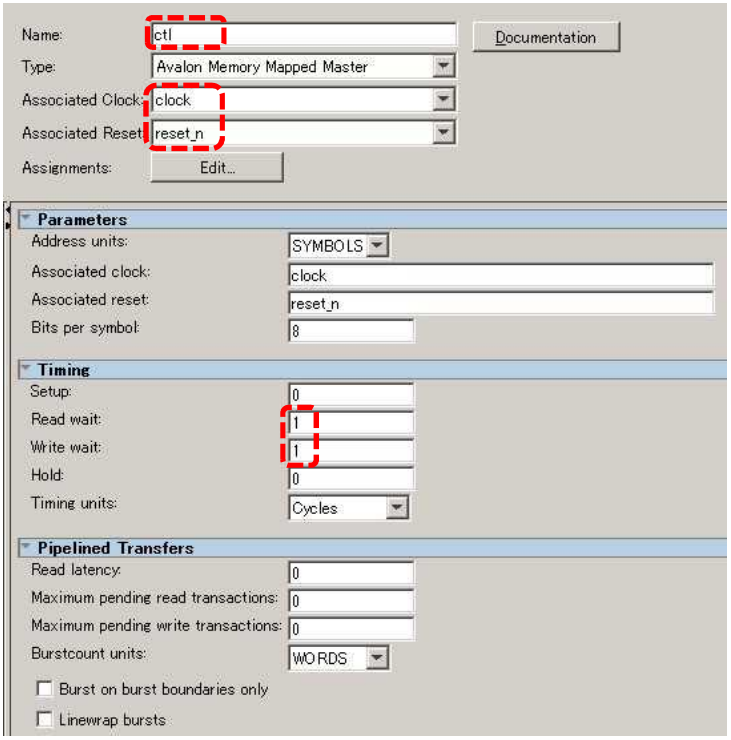
Read wait : ***1***

Write wait : ***0***

上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

Signal Type	Name	Width	Direction
address	csr_address	2	input
write	csr_write	1	input
writedata	csr_writedata	32	input
read	csr_read	1	input
readdata	csr_readdata	32	output

C) ctl (Avalon Memory Mapped Master) の設定



Name : ***ctl***

Associated Clock : ***clock***

Associated Reset : ***reset_n***

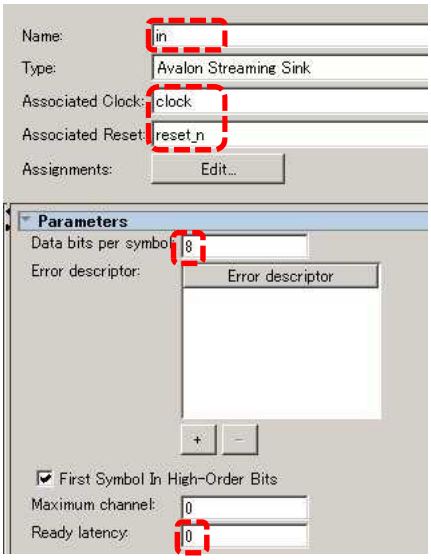
Read wait : ***1***

Write wait : ***1***

上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

Signal Type	Name	Width	Direction
address	ctl_address	32	output
write	ctl_write	1	output
writedata	ctl_writedata	16	output
read	ctl_read	1	output
readdata	ctl_readdata	16	input
wairequest	ctl_waitrequest	1	input

D) in (Avalon-Streaming Sink) の設定



Name : ***in***

Associated Clock : ***clock***

Associated Reset : ***reset_n***

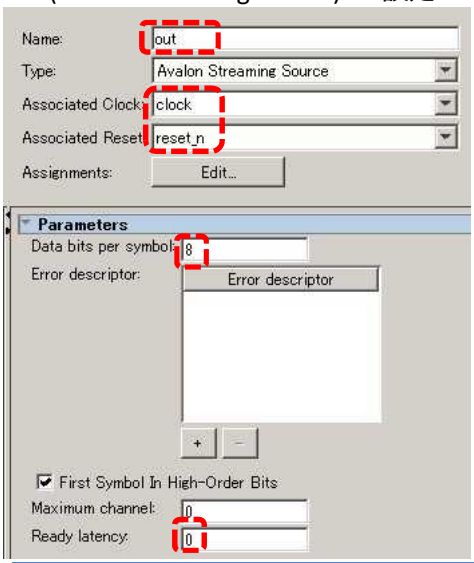
Data bits per symbol : ***8***

Ready latency : ***0***

上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

Signal Type	Name	Width	Direction
ready	in_ready	1	output
valid	in_valid	1	input
data	in_data	8	input

E) out (Avalon Streaming Source) の設定



Name : ***out***

Associated Clock : ***clock***

Associated Reset : ***reset_n***

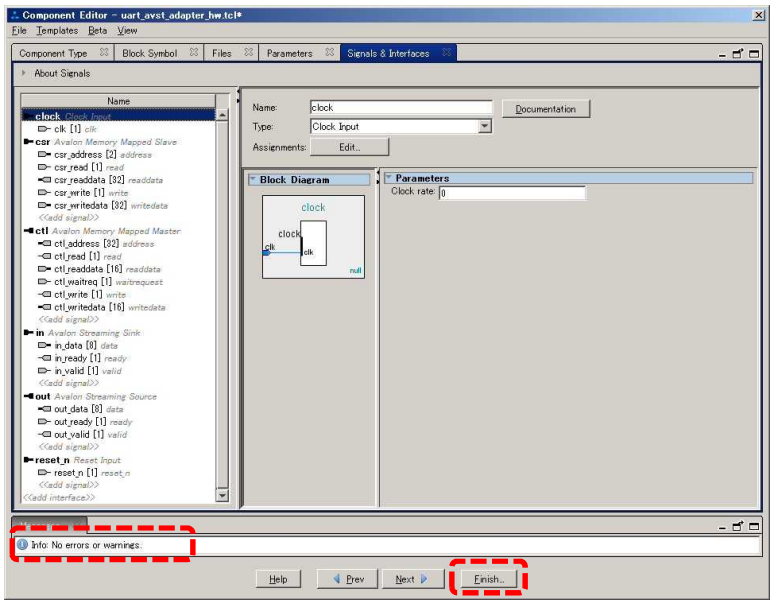
Data bits per symbol : ***8***

Ready latency : ***0***

上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

Signal Type	Name	Width	Direction
ready	out_ready	1	input
valid	out_valid	1	output
data	out_data	8	output

7. **Message** にエラーやワーニングが無い事を確認して **[Finish]** をクリックします。



8. **Platform Designer** の **IP Catalog** に追加されたモジュールが表示されれば **Component Editor** での編集は完了です。

9. プロジェクト・フォルダに作成された uart_avst_adapter_hw.tcl ファイルをテキスト・エディタで開き、下記のセクションを修正してください。

A) parameters

1. ***set_parameter_property DATA_WIDTH ALLOWED_RANGES {8 16}*** ・・・(追加または変更)

※ DATA_WIDTH パラメータを、数値の直接入力から 8 または 16 のコンボボックスで選択できるようになります。

2. ***set_parameter_property SLAVE_ADDRESS DISPLAY_HINT hexadecimal*** ・・・(追加)

※ SLAVE_ADDRESS パラメータを、整数入力から 16 進 (0x0 表現)入力にする事ができます。

B) connection point in

3. add_interface_port in in_data data Input ***8*** ⇒ ***DATA_WIDTH*** ・・・(変更)

※ in(Avalon Streaming Sink)の data を、DATA_WIDTH パラメータで設定された bit 幅で定義します。

C) connection point out

4. add_interface_port out out_data data Output ***8*** ⇒ ***DATA_WIDTH*** ・・・(変更)

※ out(Avalon Streaming Source)の data を、DATA_WIDTH パラメータで設定された bit 幅で定義します。

下記に変更後の uart_avst_adapter_hw.tcl ファイルの内容を示します。

```
#
# uart_avst_adapter "UART avalon-ST adapter" v17.1
#
#
#


#
# request TCL package from ACDS 16.1
#
package require -exact qsys 16.1


#
# module uart_avst_adapter
#
set_module_property DESCRIPTION ""
set_module_property NAME uart_avst_adapter
set_module_property VERSION 17.1
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP "Basic Functions/Bridges and Adaptors"
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME "UART avalon-ST adapter"
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false


#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL uart_avst_adapter
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file uart_avst_adapter.v VERILOG PATH uart_avst_adapter.v TOP_LEVEL_FILE

add_fileset SIM_VERILOG SIM_VERILOG "" ""
set_fileset_property SIM_VERILOG TOP_LEVEL uart_avst_adapter
set_fileset_property SIM_VERILOG ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property SIM_VERILOG ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file uart_avst_adapter.v VERILOG PATH uart_avst_adapter.v


#
# parameters
#
add_parameter DATA_WIDTH INTEGER 8
set_parameter_property DATA_WIDTH DEFAULT_VALUE 8
set_parameter_property DATA_WIDTH DISPLAY_NAME DATA_WIDTH
set_parameter_property DATA_WIDTH TYPE INTEGER
set_parameter_property DATA_WIDTH UNITS None
set_parameter_property DATA_WIDTH ALLOWED_RANGES {8 16}
set_parameter_property DATA_WIDTH HDL_PARAMETER true
add_parameter SLAVE_ADDRESS INTEGER 0
set_parameter_property SLAVE_ADDRESS DEFAULT_VALUE 0
set_parameter_property SLAVE_ADDRESS DISPLAY_NAME SLAVE_ADDRESS
set_parameter_property SLAVE_ADDRESS TYPE INTEGER
set_parameter_property SLAVE_ADDRESS UNITS None
set_parameter_property SLAVE_ADDRESS ALLOWED_RANGES -2147483648:2147483647
set_parameter_property SLAVE_ADDRESS DISPLAY_HINT hexadecimal
set_parameter_property SLAVE_ADDRESS HDL_PARAMETER true


#
# display items
#


#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""

add_interface_port clock clk clk Input 1


#
# connection point reset_n
#
add_interface reset_n reset end
set_interface_property reset_n associatedClock clock
set_interface_property reset_n synchronousEdges DEASSERT
set_interface_property reset_n ENABLED true
set_interface_property reset_n EXPORT_OF ""
set_interface_property reset_n PORT_NAME_MAP ""
set_interface_property reset_n CMSIS_SVD_VARIABLES ""
set_interface_property reset_n SVD_ADDRESS_GROUP ""

add_interface_port reset_n reset_n reset_n Input 1
```

```
#
# connection point csr
#
add_interface csr avalon end
set_interface_property csr addressUnits WORDS
set_interface_property csr associatedClock clock
set_interface_property csr associatedReset reset_n
set_interface_property csr bitsPerSymbol 8
set_interface_property csr burstOnBurstBoundariesOnly false
set_interface_property csr burstcountUnits WORDS
set_interface_property csr explicitAddressSpan 0
set_interface_property csr holdTime 0
set_interface_property csr linewrapBursts false
set_interface_property csr maximumPendingReadTransactions 0
set_interface_property csr maximumPendingWriteTransactions 0
set_interface_property csr readLatency 0
set_interface_property csr readWaitTime 1
set_interface_property csr setupTime 0
set_interface_property csr timingUnits Cycles
set_interface_property csr writeWaitTime 0
set_interface_property csr ENABLED true
set_interface_property csr EXPORT_OF ""
set_interface_property csr PORT_NAME_MAP ""
set_interface_property csr CMSIS_SVD_VARIABLES ""
set_interface_property csr SVD_ADDRESS_GROUP ""

add_interface_port csr csr_address address Input 2
add_interface_port csr csr_write write Input 1
add_interface_port csr csr_writedata writedata Input 32
add_interface_port csr csr_read read Input 1
add_interface_port csr csr_readdata readdata Output 32
set_interface_assignment csr embeddedsw.configuration.isFlash 0
set_interface_assignment csr embeddedsw.configuration.isMemoryDevice 0
set_interface_assignment csr embeddedsw.configuration.isNonVolatileStorage 0
set_interface_assignment csr embeddedsw.configuration.isPrintableDevice 0


#
# connection point ctl
#
add_interface ctl avalon start
set_interface_property ctl addressUnits SYMBOLS
set_interface_property ctl associatedClock clock
set_interface_property ctl associatedReset reset_n
set_interface_property ctl bitsPerSymbol 8
set_interface_property ctl burstOnBurstBoundariesOnly false
set_interface_property ctl burstcountUnits WORDS
set_interface_property ctl doStreamReads false
set_interface_property ctl doStreamWrites false
set_interface_property ctl holdTime 0
set_interface_property ctl linewrapBursts false
set_interface_property ctl maximumPendingReadTransactions 0
set_interface_property ctl maximumPendingWriteTransactions 0
set_interface_property ctl readLatency 0
set_interface_property ctl readWaitTime 1
set_interface_property ctl setupTime 0
set_interface_property ctl timingUnits Cycles
set_interface_property ctl writeWaitTime 1
set_interface_property ctl ENABLED true
set_interface_property ctl EXPORT_OF ""
set_interface_property ctl PORT_NAME_MAP ""
set_interface_property ctl CMSIS_SVD_VARIABLES ""
set_interface_property ctl SVD_ADDRESS_GROUP ""

add_interface_port ctl ctl_address address Output 32
add_interface_port ctl ctl_read read Output 1
add_interface_port ctl ctl_readdata readdata Input 16
add_interface_port ctl ctl_write write Output 1
add_interface_port ctl ctl_writedata writedata Output 16
add_interface_port ctl ctl_waitrequest waitrequest Input 1


#
# connection point in
#
add_interface in avalon_streaming end
set_interface_property in associatedClock clock
set_interface_property in associatedReset reset_n
set_interface_property in dataBitsPerSymbol 8
set_interface_property in errorDescriptor ""
set_interface_property in firstSymbolInHighOrderBits true
set_interface_property in maxChannel 0
set_interface_property in readyLatency 0
set_interface_property in ENABLED true
set_interface_property in EXPORT_OF ""
set_interface_property in PORT_NAME_MAP ""
set_interface_property in CMSIS_SVD_VARIABLES ""
set_interface_property in SVD_ADDRESS_GROUP ""

add_interface_port in in_ready ready Output 1
add_interface_port in in_valid valid Input 1
add_interface_port in in_data data Input DATA_WIDTH
```

```
#
# connection point out
#
add_interface out avalon_streaming start
set_interface_property out associatedClock clock
set_interface_property out associatedReset reset_n
set_interface_property out dataBitsPerSymbol 8
set_interface_property out errorDescriptor ""
set_interface_property out firstSymbolInHighOrderBits true
set_interface_property out maxChannel 0
set_interface_property out readyLatency 0
set_interface_property out ENABLED true
set_interface_property out EXPORT_OF ""
set_interface_property out PORT_NAME_MAP ""
set_interface_property out CMSIS_SVD_VARIABLES ""
set_interface_property out SVD_ADDRESS_GROUP ""

add_interface_port out out_ready ready Input 1
add_interface_port out out_valid valid Output 1
add_interface_port out out_data data Output DATA_WIDTH
```

10. 関連ファイルを纏めるため、プロジェクトのルート・フォルダの下に新たにフォルダを追加し、そこに uart_avst_adapter.v と修正した uart_avst_adapter_hw.tcl を移動してください。

11. Nios II SBT でプロジェクトを作成した際、本モジュールのヘッダファイルを自動で読み込ませるための作業を行います。

    下記の uart_avst_adapter_sw.tcl ファイルを作成し、10. で作成したフォルダに格納します。

```
# Create a new driver
create_driver uart_avst_adapter_driver

# Associate it with some hardware known as "uart_avst_adapter"
set_sw_property hw_class_name uart_avst_adapter

# The version of this driver
set_sw_property version __VERSION_SHORT__

# This driver may be incompatible with versions of hardware less
# than specified below. Updates to hardware and device drivers
# rendering the driver incompatible with older versions of
# hardware are noted with this property assignment.
#
# Multiple-Version compatibility was introduced in version 7.1;
# prior versions are therefore excluded.
set_sw_property min_compatible_hw_version 7.1

# Initialize the driver in alt_sys_init()
set_sw_property auto_initialize false

# Location in generated BSP that above sources will be copied into
set_sw_property bsp_subdirectory drivers

# Interrupt properties: This driver supports both legacy and enhanced
# interrupt APIs, as well as ISR preemption.
set_sw_property isr_preemption_supported true
set_sw_property supported_interrupt_apis "legacy_interrupt_api enhanced_interrupt_api"

#
# Source file listings...
#

# C/C++ source files

# Include files
add_sw_property include_source inc/uart_avst_adapter_regs.h

# This driver supports HAL & UCOSII BSP (OS) types
add_sw_property supported_bsp_type HAL
add_sw_property supported_bsp_type UCOSII

# End of file
```

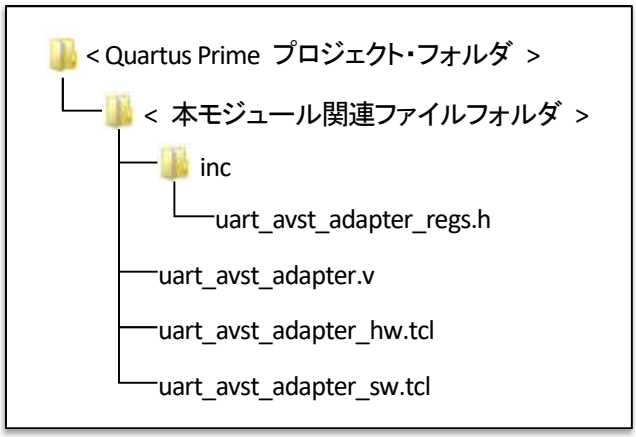
※ このファイルを追加する事で、本モジュールを組み込んだ .sopcinfo を使って Nios II SBT でプロジェクトを作成すると、自動的に **inc/uart_avst_adapter_regs.h** が BSP プロジェクトの drivers フォルダに組み込まれるようになります。

12. 下記の uart_avst_adapter_regs.h（C ヘッダ）ファイルを作成し、10. で作成したフォルダの下に ”inc” フォルダを新たに作成し、そこに格納します。

```
// file  uart_avst_adapter_reg.h

#ifndef __UART_AVST_ADAPTER_REGS_H__
#define __UART_AVST_ADAPTER_REGS_H__

#include <io.h>

#define UART_ADAPTER_CONTROL_REG                0

#define IOADDR_UART_ADAPTER_CONTROL(base)        \
        __IO_CALC_ADDRESS_NATIVE(base, UART_ADAPTER_CONTROL_REG)
#define IORD_UART_ADAPTER_CONTROL(base)          \
        IORD(base, UART_ADAPTER_CONTROL_REG)
#define IOWR_UART_ADAPTER_CONTROL(base, data)    \
        IOWR(base, UART_ADAPTER_CONTROL_REG, data)

#define UART_ADAPTER_CTL_TX_ENABLE              0x01
#define UART_ADAPTER_CTL_RX_ENABLE              0x02

#define UART_ADAPTER_INTERVAL_REG               1

#define IOADDR_UART_ADAPTER_INTERVAL(base)       \
        __IO_CALC_ADDRESS_NATIVE(base, UART_ADAPTER_INTERVAL_REG)
#define IORD_UART_ADAPTER_INTERVAL(base)         \
        IORD(base, UART_ADAPTER_INTERVAL_REG)
#define IOWR_UART_ADAPTER_INTERVAL(base, data)   \
        IOWR(base, UART_ADAPTER_INTERVAL_REG, data)

#define UART_ADAPTER_TX_COUNT_REG               2

#define IOADDR_UART_ADAPTER_TX_COUNT(base)       \
        __IO_CALC_ADDRESS_NATIVE(base, UART_ADAPTER_TX_COUNT_REG)
#define IORD_UART_ADAPTER_TX_COUNTL(base)        \
        IORD(base, UART_ADAPTER_TX_COUNT_REG)

#define UART_ADAPTER_RX_COUNT_REG               3

#define IOADDR_UART_ADAPTER_RX_COUNT(base)       \
        __IO_CALC_ADDRESS_NATIVE(base, UART_ADAPTER_RX_COUNT_REG)
#define IORD_UART_ADAPTER_RX_COUNT(base)         \
        IORD(base, UART_ADAPTER_RX_COUNT_REG)

#endif /* __UART_AVST_ADAPTER_REGS_H__ */
```

※ 本ファイルは uart_avst_adapter_sw.tcl の下位の inc フォルダに格納されている必要があります。

13. フォルダとファイルの構成は下記のようになります。



4-3. Nios II システムの作成

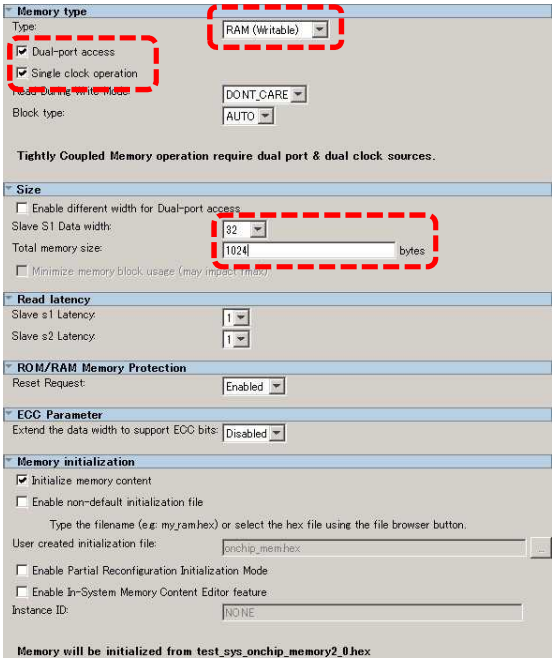
4-3-1. ベース・プロジェクトの準備

Nios II が動作する Quartus Prime プロジェクトを用意し、Platform Designer で UART 、mSGDMA 、On-Chip RAM と、今回作成した uart_avst_adapter を追加します。

4-3-2. Platform Designer の編集

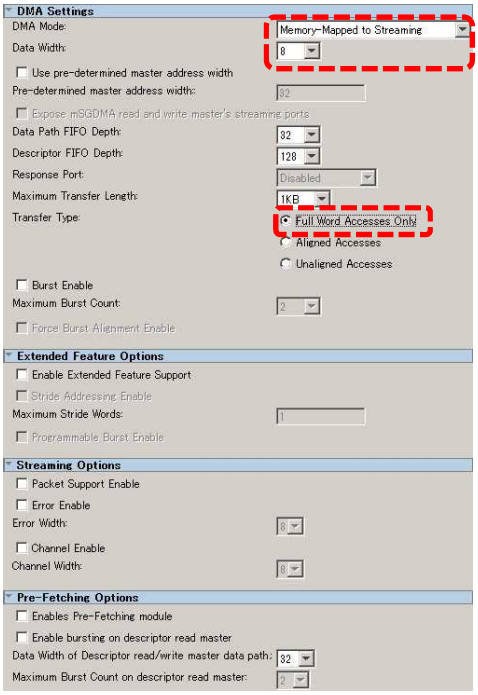
1. Platform Designer を開き、下記のモジュールを順に追加します。
   - On-Chip RAM (On-Chip Memory (RAM or ROM))
   - mSGDMA (Modular Scatter-Gather DMA)
   - UART (UART Core)
   - uart_avst_adapter (UART avalon-ST adaper)
   - mSGDMA (Modular Scatter-Gather DMA)
   - On-Chip RAM (On-Chip Memory (RAM or ROM))

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。
   - On-Chip RAM



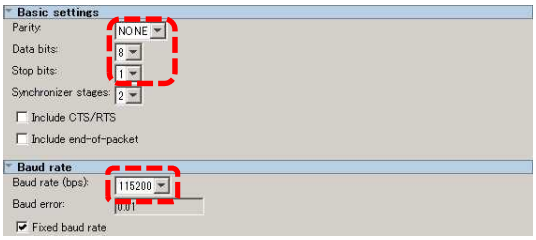
Type ： ***RAM(Writable)***
Dual-port access ： ***On***
Single clock operation ： ***On***
Slave S1 Data width ： ***32***
Total memory size ： ***1024***
上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

・ mSGDMA



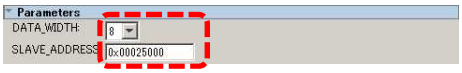
DMA Mode ： ***Memory-Mapped to Streaming***
Data Width ： ***8***
Transfer Type ： ***Full Word Access Only***
上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

・ UART



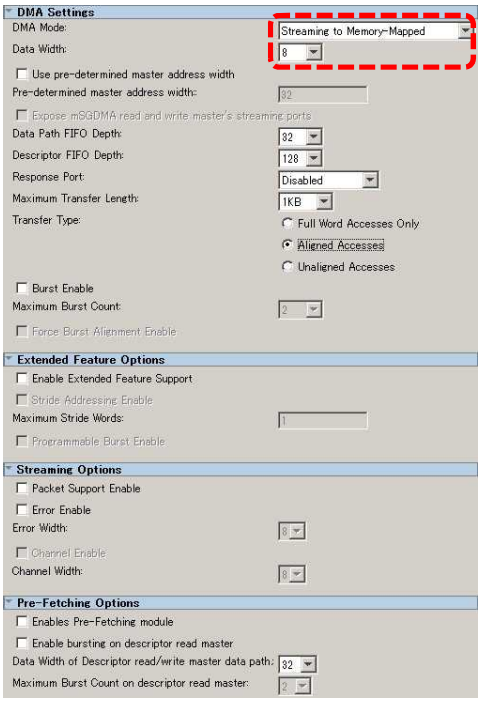
Parity ： ***NONE***
Data bits ： ***8***
Stop bits ： ***1***
Baud rate (bps) ： ***115200***
上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

・ uart_avst_adapter



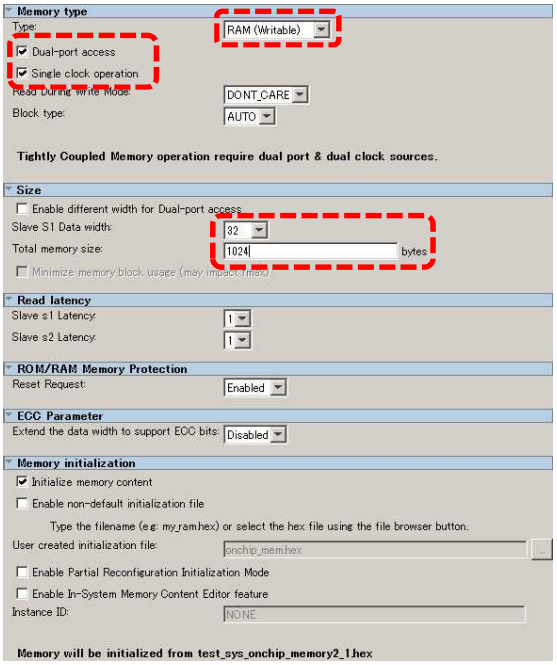
DATA_WIDTH ： ***8***
SLAVE_ADDRESS ： ***0x00025000***
DATA_WIDTH ： UART の Data bits を 7 または 8 と設定した場合は 8 、Data bits を 9 と設定した場合は 16 を選択してください。
SLAVE_ADDRESS ： UART の CSR ベース・アドレスを入力します。

・ mSGDMA



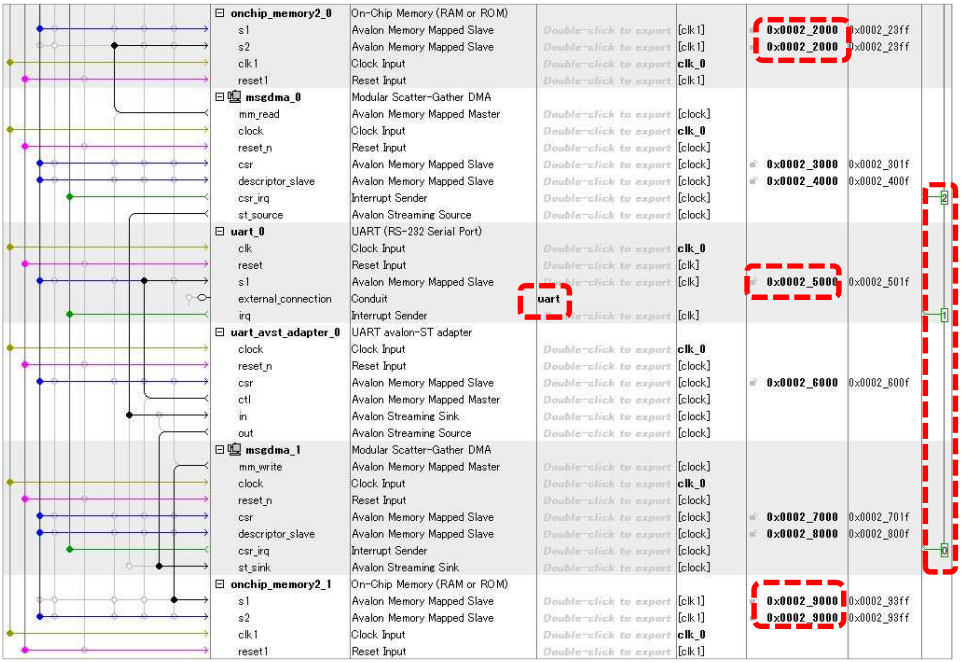
DMA Mode ： ***Streaming to Memory-Mapped***
Data Width ： ***8***
上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

・ On-Chip RAM



Type ： ***RAM(Writable)***
Dual-port access ： ***On***
Single clock operation ： ***On***
Slave S1 Data width ： ***32***
Total memory size ： ***1024***
上記は重要なパラメータです。上記以外はデフォルトのままで構いません。

3. 追加したモジュールを下記の通り接続し、ベース・アドレスや IRQ 番号、UART の Conduit を設定します。



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。

4-4. 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 uart_test(
    input           FPGA_CLK1_50,
    input   [1:0]   KEY,
    output          txd,
    input           rxd
);

wire    clk_100;
wire    reset_n;

pll pll_inst (
    .refclk     (FPGA_CLK1_50),     // Input Clock(50MHz)
    .rst        (!KEY[0]),          // Reset
    .outclk_0   (clk_100),          // Output Clock(100MHz)
    .locked     (reset_n)           // PLL Locked
);

test_sys u_test_sys (
    .clk_clk        (clk_100),
    .reset_reset_n  (reset_n),

    .uart_rxd       (rxd),
    .uart_txd       (txd)
);

endmodule
```

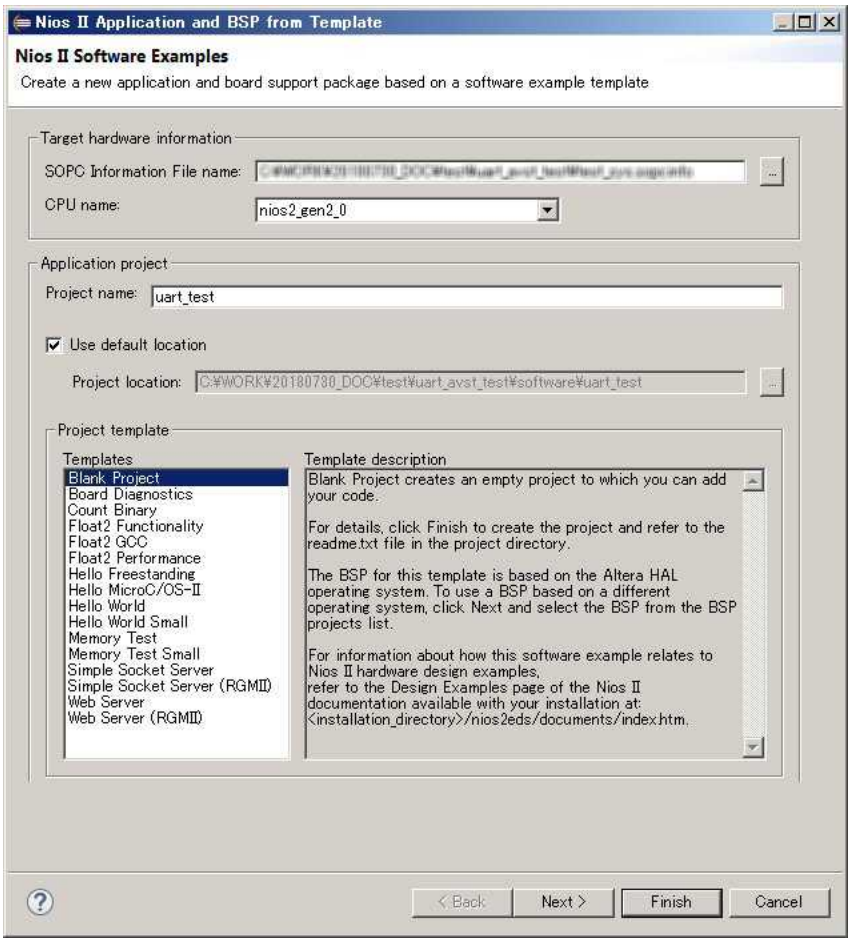
2. Assignment Editor でピン配置してください。

	tatu	From	To	Assignment Name	Value	Enabled
1	✔		FPGA_CLK1_50	Location	PIN_V11	Yes
2	✔		KEY[0]	Location	PIN_AH17	Yes
3	✔		KEY[1]	Location	PIN_AH16	Yes
4	✔		FPGA_CLK1_50	I/O Standard	3.3-V LVTTL	Yes
5	✔		KEY[0]	I/O Standard	3.3-V LVTTL	Yes
6	✔		KEY[1]	I/O Standard	3.3-V LVTTL	Yes
7	✔		txd	Location	PIN_AF7	Yes
8	✔		rxd	Location	PIN_AF8	Yes
9	✔		txd	I/O Standard	3.3-V LVTTL	Yes
10	✔		rxd	I/O Standard	3.3-V LVTTL	Yes

3. Quartus Prime でコンパイルしてください。

4-5. ソフトウェア検証プロジェクトの作成

1. Nios II SBT を開き、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。

```
// file  main.c

/*****************************************************************************
 *  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_uart.h>
#include <altera_avalon_uart_regs.h>
#include <altera_msgdma_descriptor_regs.h>
#include <altera_msgdma_csr_regs.h>
#include <altera_msgdma.h>
#include <uart_avst_adapter_regs.h>

/*****************************************************************************
 *  re-definitions
 *  ※このセクションは"system.h"で定義された定数を、
 *    本ソースファイル内で使用するために再定義したものです。
 *    ご使用の際は右側の定数を"system.h"を参照の上、変更してお使いください。
 *****************************************************************************/
#define UART_ADAPTER        (UART_AVST_ADAPTER_0_BASE)      // UART->Avalon-ST アダプタのCSRアドレス

#define UART_ADRS           (UART_0_BASE)                   // UART のCSRアドレス
#define UART_IRQ_ID         (UART_0_IRQ_INTERRUPT_CONTROLLER_ID)  // UART の割り込みコントローラID
#define UART_IRQ_NO         (UART_0_IRQ)                    // UART の割り込み番号
#define UART_BITS           (UART_0_DATA_BITS)              // UART のデータ bit 幅

#define TX_MSGDMA_NAME      (MSGDMA_0_CSR_NAME)             // Tx 用 mSGDMAのレジスタ名
#define TX_BUFF_ADRS        (ONCHIP_MEMORY2_0_BASE)         // Tx 用送信バッファのアドレス
#define TX_BUFF_SIZE        (ONCHIP_MEMORY2_0_SIZE_VALUE)   // Tx 用送信バッファのサイズ

#define RX_MSGDMA_NAME      (MSGDMA_1_CSR_NAME)             // Rx 用mSGDMAのレジスタ名
#define RX_BUFF_ADRS        (ONCHIP_MEMORY2_1_BASE)         // Rx 用送信バッファのアドレス
#define RX_BUFF_SIZE        (ONCHIP_MEMORY2_1_SIZE_VALUE)   // Rx 用送信バッファのサイズ

/*****************************************************************************
 *  definitions (define, enum, typedef, etc...)
 *****************************************************************************/
#define TX_DMA                                      // UART TxをDMA実行させる場合
#define RX_DMA                                      // UART RxをDMA実行させる場合

#define TRUE                -1
#define FALSE               0

#define UART_STAT_MSK       (ALTERA_AVALON_UART_STATUS_PE_MSK  | \
                             ALTERA_AVALON_UART_STATUS_FE_MSK  | \
                             ALTERA_AVALON_UART_STATUS_BRK_MSK | \
                             ALTERA_AVALON_UART_STATUS_ROE_MSK | \
                             ALTERA_AVALON_UART_STATUS_TOE_MSK | \
                             ALTERA_AVALON_UART_STATUS_E_MSK)

#define UART_CTRL_MSK       (ALTERA_AVALON_UART_CONTROL_PE_MSK  | \
                             ALTERA_AVALON_UART_CONTROL_FE_MSK  | \
                             ALTERA_AVALON_UART_CONTROL_BRK_MSK | \
                             ALTERA_AVALON_UART_CONTROL_ROE_MSK | \
                             ALTERA_AVALON_UART_CONTROL_TOE_MSK | \
                             ALTERA_AVALON_UART_CONTROL_E_MSK)

#define TRANS_LENGTH        128

// UART の Data bitsに合せて処理を変更
#if (UART_BITS == 9)
    typedef unsigned short  UART_DATA;  // データ幅は16bit
    #define DMA_MULT        2           // DMAでの転送データを2倍
#elif (UART_BITS == 8)
    typedef unsigned char   UART_DATA;  // データ幅は8bit
    #define DMA_MULT        1           // DMAでの転送データはそのまま
#else
    typedef unsigned char   UART_DATA;  // データ幅は8bit
    #define DMA_MULT        1           // DMAでの転送データはそのまま
#endif

/*****************************************************************************
 *  variables
 *****************************************************************************/
volatile int rx_cmp = FALSE;    // 受信完了フラグ
volatile int rx_err = FALSE;    // 受信エラーフラグ
volatile int tx_err = FALSE;    // 送信エラーフラグ
volatile int uart_err = FALSE;  // UARTエラーフラグ

#ifndef RX_DMA
int nRxCount = 0;                               // UART Rxを割り込みを使って受信する場合のカウンタ
UART_DATA *pRxData = (UART_DATA*)(RX_BUFF_ADRS);    // UART Rxを割り込みを使って受信する場合のバッファアドレス
#endif

/*****************************************************************************
 *  proto types
 *****************************************************************************/
void dump(UART_DATA *adr, int size);                    // メモリダンプ
int check(UART_DATA *adr0, UART_DATA *adr1, int size);  // メモリチェック
```

uart_avst_adapter のレジスタ定義ヘッダファイル

"system.h" から再定義しています

TX_DMA は送信処理に DMA を使用します
RX_DMA は受信処理に DMA を使用します

UART の Data bits の設定に合わせて、送受信データのタイプを切り替えています。
7-bit : unsigned char
8-bit : unsigned char
9-bit : unsigned short

```c
/*******************************************************************************
 *  interrupt handler
 ******************************************************************************/
// UART 割り込みハンドラ
static void uart_callback(void * context)
{
    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();

    // コントロール Reg' とステータス Reg' の値を取得
    int nCtl = IORD_ALTERA_AVALON_UART_CONTROL(csr);
    int nSta = IORD_ALTERA_AVALON_UART_STATUS(csr);

#ifdef RX_DMA
    // 割り込みクリア
    IOWR_ALTERA_AVALON_UART_STATUS(csr, UART_STAT_MSK);

#else
    // 割り込みクリア
    IOWR_ALTERA_AVALON_UART_STATUS(csr, UART_STAT_MSK | ALTERA_AVALON_UART_STATUS_RRDY_MSK);

    // RRDY が立っていれば RX Data 取り出し
    if(nSta & ALTERA_AVALON_UART_STATUS_RRDY_MSK)
    {
        // RX Bufferに書き込んでカウンタをインクリメント
        pRxData[nRxCount ++] = IORD_ALTERA_AVALON_UART_RXDATA(csr);
        if(nRxCount >= TRANS_LENGTH)
        {
            nRxCount = 0;
            rx_cmp = TRUE;
        }
    }
#endif
    // エラーが発生か?
    if(nSta & UART_STAT_MSK)
    {
        // エラーフラグを立てる
        uart_err = TRUE;
        printf("UART_IRQ:%X.%X\n", nSta, nCtl);
    }

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

#ifdef TX_DMA
// Tx DMA 割り込みハンドラ
static void tx_dma_callback(void * context)
{
    struct alt_msgdma_dev *tx_dma = (alt_msgdma_dev*)context;

    // コントロール Reg' とステータス Reg' の値を取得
    int nCtl = IORD_ALTERA_MSGDMA_CSR_CONTROL(tx_dma->csr_base);
    int nSta = IORD_ALTERA_MSGDMA_CSR_STATUS(tx_dma->csr_base);

    // エラーが発生か?
    if(nSta & ALTERA_MSGDMA_CSR_STOPPED_ON_ERROR_MASK)
    {
        // エラーフラグを立てる
        tx_err = TRUE;
        printf("TX_IRQ:%X.%X\n", nSta, nCtl);
    }
}
#endif

#ifdef RX_DMA
// Rx DMA 割り込みハンドラ
static void rx_dma_callback(void * context)
{
    struct alt_msgdma_dev *tx_dma = (alt_msgdma_dev*)context;

    // コントロール Reg' とステータス Reg' の値を取得
    int nCtl = IORD_ALTERA_MSGDMA_CSR_CONTROL(tx_dma->csr_base);
    int nSta = IORD_ALTERA_MSGDMA_CSR_STATUS(tx_dma->csr_base);

    // 指定 byte 数分受信完了?
    if(nSta & ALTERA_MSGDMA_CSR_DESCRIPTOR_BUFFER_EMPTY_MASK)
    {
        // 受信完了フラグを立てる
        rx_cmp = TRUE;
    }else
    {
        // それ以外の割り込みはエラーとする
        rx_err = TRUE;
        printf("RX_IRQ:%X.%X\n", nSta, nCtl);
    }
}
#endif
```

UART の割り込みハンドラ
通信エラー発生時と、受信処理をソフトウェアで実行した場合の受信割り込みに対応

受信割り込みにより、受信データをレジスタから読み出してバッファに書き込み

送信用 DMA 割り込みハンドラ

受信用 DMA 割り込みハンドラ

```
/*****************************************************************************
 *  main function
 *****************************************************************************/
int main()
{
    int i;
    int tx_length;  // 送信データ数

    // Tx/Rx 用バッファアドレスを非キャッシュ領域で取得
    UART_DATA *tx_buff = (UART_DATA*)(TX_BUFF_ADRS + 0x8000000);
    UART_DATA *rx_buff = (UART_DATA*)(RX_BUFF_ADRS + 0x8000000);

#if defined(RX_DMA) && defined(TX_DMA)                                  // TxもRxもDMA転送
    // UART->Avalon-ST アダプタの初期化
    IOWR_UART_ADAPTER_CONTROL(UART_ADAPTER, UART_ADAPTER_CTL_TX_ENABLE | UART_ADAPTER_CTL_RX_ENABLE);   // Tx/Rx Enable
    IOWR_UART_ADAPTER_INTERVAL(UART_ADAPTER, 10000);                    // Interval(100u
#endif
#if defined(RX_DMA) && !defined(TX_DMA)                                 // RxのみDMA転送
    // UART->Avalon-ST アダプタの初期化
    IOWR_UART_ADAPTER_CONTROL(UART_ADAPTER, UART_ADAPTER_CTL_RX_ENABLE);    // Rx Enable
    IOWR_UART_ADAPTER_INTERVAL(UART_ADAPTER, 10000);                    // Interval(100u
#endif
#if !defined(RX_DMA) && defined(TX_DMA)                                 // TxのみDMA転送
    // UART->Avalon-ST アダプタの初期化
    IOWR_UART_ADAPTER_CONTROL(UART_ADAPTER, UART_ADAPTER_CTL_TX_ENABLE);    // Tx Enable
    IOWR_UART_ADAPTER_INTERVAL(UART_ADAPTER, 30000);                    // Interval(300us
#endif
#if !defined(RX_DMA) && !defined(TX_DMA)                                // TxもRxもDMA転送
    // UART->Avalon-ST アダプタの初期化
    IOWR_UART_ADAPTER_CONTROL(UART_ADAPTER, 0);                         // Tx/Rx Disable
    IOWR_UART_ADAPTER_INTERVAL(UART_ADAPTER, 10000);                    //
#endif

#ifdef RX_DMA
    // UART 割り込みマスクを設定
    IOWR_ALTERA_AVALON_UART_CONTROL(UART_ADRS, UART_CTRL_MSK);
#else
    // UART 割り込みマスクを設定(RRDYを割り込みで受けて受信データを取り出す)
    IOWR_ALTERA_AVALON_UART_CONTROL(UART_ADRS, UART_CTRL_MSK | ALTERA_AVALON_UART_CONTROL_RRDY_MSK);
#endif
    // UART 割り込みハンドラの登録
    alt_ic_isr_register(UART_IRQ_ID, UART_IRQ_NO, uart_callback, (void*)UART_ADRS, 0x0);

#ifdef TX_DMA
    // Tx mSGDMA
    alt_msgdma_dev *tx_dma;
    alt_msgdma_standard_descriptor tx_desc;
    int tx_status;

    // Tx 用 mSGDMA をオープン
    tx_dma = alt_msgdma_open(TX_MSGDMA_NAME);
    if(NULL == tx_dma)
    {
        printf("Error:TX mSGDMA Open Fail\n");
        return FALSE;
    }
    // Tx 用 mSGDMA の割り込みハンドラを登録
    alt_msgdma_register_callback(tx_dma,
                                 tx_dma_callback,
                                 ALTERA_MSGDMA_CSR_GLOBAL_INTERRUPT_MASK |
                                 ALTERA_MSGDMA_CSR_STOP_ON_ERROR_MASK |
                                 ALTERA_MSGDMA_CSR_STOP_ON_EARLY_TERMINATION_MASK,
                                 tx_dma);
#endif

#ifdef RX_DMA
    // Rx mSGDMA
    alt_msgdma_dev *rx_dma;
    alt_msgdma_standard_descriptor rx_desc;
    int rx_status;

    // Rx 用 mSGDMA をオープン
    rx_dma = alt_msgdma_open(RX_MSGDMA_NAME);
    if(NULL == rx_dma)
    {
        printf("Error:RX mSGDMA Open Fail\n");
        return FALSE;
    }

    // Rx 用 mSGDMA の割り込みハンドラを登録
    alt_msgdma_register_callback(rx_dma,
                                 rx_dma_callback,
                                 ALTERA_MSGDMA_CSR_GLOBAL_INTERRUPT_MASK |
                                 ALTERA_MSGDMA_CSR_STOP_ON_ERROR_MASK |
                                 ALTERA_MSGDMA_CSR_STOP_ON_EARLY_TERMINATION_MASK,
                                 rx_dma);
#endif
```

uart_avst_adapter の初期設定
・送受信に DMA を使用する場合
・受信のみに DMA を使用する場合
・送信のみに DMA を使用する場合
・送受信共に DMA を使用しない場合

INTERVAL の設定値
mSGDMA は転送開始から 5ms 以内に完了しない場合、タイムアウト・エラーとなるため、高速に転送する必要があるが、受信処理をソフトウェアで実行する場合、あまり高速にすると割り込みハンドラの実行が追い付かずオーバーラン・エラーとなる場合があるため、この設定値には注意が必要

UART 割り込みハンドラの登録

送信用 DMA の初期設定
割り込みハンドラの登録

受信用 DMA の初期設定
割り込みハンドラの登録

```
    // 送受信ループ
    while(1)
    {
#ifdef RX_DMA
        // 転送サイズ最大 255 + 1
        tx_length = (rand() & 255) + 1;
#else
        // 転送サイズ256固定(受信データが割り込みの中では分からないため固定とする)
        tx_length = TRANS_LENGTH;
#endif
        // Tx 送信用バッファをクリア
        memset(tx_buff, 0xFF, TX_BUFF_SIZE);
        // Tx 送信用バッファに乱数データを書き込み
        for(i = 0;i < tx_length;i ++)
        {
            tx_buff[i] = (UART_DATA)rand();
        }
        // RX 受信用バッファをクリア
        memset(rx_buff, 0xFF, RX_BUFF_SIZE);

        printf("Trans length:%d\n", tx_length);

        // 受信完了フラグを落とす
        rx_cmp = FALSE;

#ifdef RX_DMA
        // Rx DMAのディスクリプタの作成
        rx_status = alt_msgdma_construct_standard_st_to_mm_descriptor(rx_dma, &rx_desc, (alt_u32*)rx_buff, tx_length * DMA_MULT,
                                                                      ALTERA_MSGDMA_DESCRIPTOR_CONTROL_TRANSFER_COMPLETE_IRQ_MASK);
        if(0 != rx_status)
        {
            printf("Error:RX DMA descriptor Fail[%d]\n", rx_status);
            return FALSE;
        }
        // Rx DMAの受信起動(non blocking)
        rx_status = alt_msgdma_standard_descriptor_async_transfer(rx_dma, &rx_desc);
        if(0 != rx_status)
        {
            printf("Error:RX DMA async trans Fail[%d]\n", rx_status);
            return FALSE;
        }
#else
        // 受信処理はUART割込みの中で行う
#endif

        // 送信前にキャッシュフラッシュ
        alt_dcache_flush_all();

#ifdef TX_DMA
        // Tx DMAのディスクリプタの作成
        tx_status = alt_msgdma_construct_standard_mm_to_st_descriptor(tx_dma, &tx_desc, (alt_u32*)tx_buff, tx_length * DMA_MULT,
                                                                      ALTERA_MSGDMA_DESCRIPTOR_CONTROL_TRANSFER_COMPLETE_IRQ_MASK);
        if(0 != tx_status)
        {
            printf("Error:TX DMA descriptor Fail[%d]\n", tx_status);
            return FALSE;
        }
        // Tx DMAの送信起動(blocking)
        tx_status = alt_msgdma_standard_descriptor_sync_transfer(tx_dma, &tx_desc);
        if(0 != tx_status)
        {
            printf("Error:TX DMA sync trans Fail[%d]\n", tx_status);
            return FALSE;
        }
#else
        // 転送サイズ分 UART 送信
        for(i = 0;i < tx_length;)
        {
            //  Tx Readyなら
            if(IORD_ALTERA_AVALON_UART_STATUS(UART_ADRS) & ALTERA_AVALON_UART_STATUS_TRDY_MSK)
            {
                // 送信データ書込み
                IOWR_ALTERA_AVALON_UART_TXDATA(UART_ADRS, tx_buff[i ++]);
                // ※RxをDMAで受けた場合、オーバーランエラーが発生するためlus程度Wait
                usleep(1000);
            }
        }
#endif

        // RX DMA 転送完了待ちループ
        while(1)
        {
            // Rx DMA が Buffer Empty になれば受信完了
            if(rx_cmp == TRUE) break;
            // DMA と UART でエラーが検出されていれば終了
            if(tx_err || rx_err || uart_err) break;
        }

        // 送信バッファと受信バッファの値を比較し、異なっていればダンプ出力して終了
        if(FALSE == check(tx_buff, rx_buff, tx_length))
        {
            // 転送データのダンプ
            printf("[ TX Buffer ]\n");
            dump(tx_buff, TX_BUFF_SIZE);

            printf("[ RX Buffer ]\n");
            dump(rx_buff, RX_BUFF_SIZE);

            return FALSE;
        }else
        {
            // 転送OK
            printf("Verify OK!\n");
        }
    }
    return TRUE;
}
```

送信データ数の初期化
受信に DMA を使用しない場合、割り込みハンドラの中で受信データ数が判別できないため TRANS_LENGTH で固定。
それ以外は 256 未満の乱数で設定

送信/受信バッファの初期化
送信バッファは乱数で、受信バッファは 0xFF を設定

受信用 DMA のディスクリプタテーブルを作成

受信用 DMA の起動
（ノンブロッキング・モード）

受信に DMA を使用しない場合、受信処理は割り込みハンドラの中で実行

送信用 DMA のディスクリプタテーブルを作成

送信用 DMA の起動
（ブロッキング・モード）

送信に DMA を使用しない場合、送信バッファから読み出したデータを UART のレジスタに書き込む

転送完了待ちポーリング処理
割り込みハンドラで設定されたフラグを判断

送受信データのベリファイを行い、違いがあればコンソールにメモリ・ダンプを出力して終了

```
/*****************************************************************************
 *  sub function
 *****************************************************************************/
#define PRINT_ADRS      "%04X:"
#define PRINT_CRLF      "\n"
#if (UART_BITS == 9)
#define PRINT_DATA      " %04X"
#define PRINT_ERR       "Error[%04X]: %04X:%04X\n"
#define PRINT_LINE      8
#define UART_DATA_MSK   0x1FF
#elif (UART_BITS == 8)
#define PRINT_DATA      " %02X"
#define PRINT_ERR       "Error[%04X]: %02X:%02X\n"
#define PRINT_LINE      16
#define UART_DATA_MSK   0xFF
#else
#define PRINT_DATA      " %02X"
#define PRINT_ERR       "Error[%04X]: %02X:%02X\n"
#define PRINT_LINE      16
#define UART_DATA_MSK   0x7F
#endif

void dump(UART_DATA *adr, int size)
{
    int i;

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

    for(i = 0;i < (size / sizeof(UART_DATA));i ++)
    {
        // アドレス
        if((i % PRINT_LINE == 0) && (i < size - 1))
        {
            printf(PRINT_ADRS, (i * sizeof(UART_DATA)));
        }
        // データ
        printf(PRINT_DATA, adr[i]);
        // CR,LF
        if((i % PRINT_LINE == (PRINT_LINE - 1)) && (i < size - 1))
        {
            printf(PRINT_CRLF);
        }
    }
    printf("\n");
}

int check(UART_DATA *adr0, UART_DATA *adr1, int size)
{
    int i;
    int res = TRUE;

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

    for(i = 0;i < (size / sizeof(UART_DATA));i ++)
    {
        if((adr0[i] & UART_DATA_MSK) != adr1[i])
        {
            printf(PRINT_ERR, (i * sizeof(UART_DATA)), (adr0[i] & UART_DATA_MSK) ,adr1[i]);
            res = FALSE;
        }
    }
    return res;
}
```

コンソールにメモリの値をダンプ出力

送受信バッファの値をベリファイ

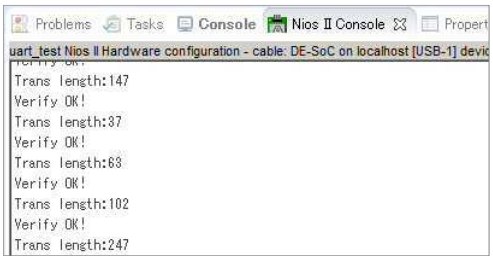
3. Build してエラーが無い事を確認してください。

5. 検証

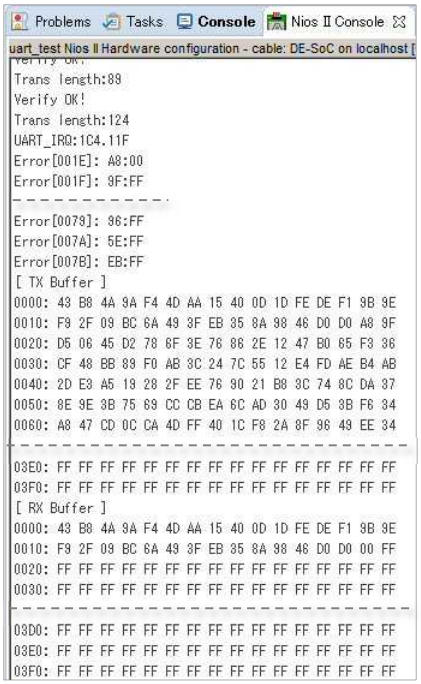
5-1. 動作の確認

1. GPIO-0 の Pin-2 と Pin-4 をジャンパ線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。



4. 動作中にジャンパ線を引き抜くと、UART 通信エラーが発生し、ベリファイ結果がエラーとなります。



6. 補足

6-1. 注意事項

- mSGDMA の Hardware Abstraction Layer Application Program Interface（以降、HAL API ）を使用してデータ転送を行う場合、転送に時間が掛かると関数によってはタイムアウト・エラー（ -ETIME ）を出力して停止する場合がありますので、その場合は下記の方法をお試しください。
  - Interval Count (Reg'-1) の値を小さくして動作周期を短くする。ただし、Tx もしくは Rx のどちらかをソフトウェア処理で実装した場合、Nios II からのレジスタ・アクセスに遅延が発生する可能性がありますので、設定値の選定は注意してください。
  - mSGDMA と uart_avst_adapter の間の Avalon-ST に On-Chip FIFO Memory Core を挿入する事で、DMA の動作時間を短くする。
  - HAL API を使用せず、レジスタに直接アクセスして mSGDMA を制御する。
- 受信に DMA を使用した場合、コマンドやレスポンスが固定長のものは扱い易いものの、不定長のものは扱い難い場合があります。そのような場合は、受信処理をソフトウェアで実装する事も可能ですが、mSGDMA の代わりに On-Chip FIFO Memory Core (Avalon FIFO Memory) を使う事で、取りこぼしの可能性の低い自由度の高い受信処理を行う事ができます。

7. 参考資料

- Nios® II まとめページ
- Embedded Peripherals IP User Guide
- インテル® FPGA の開発フロー／FPGA トップページ
- マクニカ・ホームページ の Nios® II 関連 技術情報ページ
- マクニカ・ホームページ の Nios® II 関連 FAQ ページ
- アルティマ技術サポート Nios® II 関連 技術コンテンツページ
- アルティマ技術サポート Nios® II 関連 FAQ ページ

改版履歴

Revision	年月	概要
1	2018 年 8 月	初版
2	2020 年 5 月	① リンク URL 修正（p3、p32） ②「免責およびご利用上の注意」内のリンク修正（p33）