

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

<u>目次</u>

1. はじめに		3
2. 適用条件		3
2-1. 対応バージョン		
2-2. 検証ハードウェア		
3. 仕様		3
3-1. 機能		
3-2. I2C Master の使い	ヽ方	
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	
4-3. Quartus Prime $ {\cal O}$	編集	9
4-4. Nios II Software Bu	uild Tool (以降 Nios II SBT) の編集	
5. 検証		15
5-1. 動作の確認		
6. 補足		16
6-1. 注意事項		
7. 参考資料		17
改版履歴		

1. <u>はじめに</u>

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. <u>適用条件</u>

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.<u>仕様</u>

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 (400kbps) 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。

<pre>module top(input clock, input reset_n, inout i2c_master_sda, inout i2c_master_scl);</pre>	
wire scl_in; wire sda_in; wire scl_oe; wire sda_oe;	
assign scl_in = i2c_master, assign sda_in = i2c_master, assign i2c_master_scl = (s assign i2c_master_sda = (s	_scl; _sda; c1_oe)? 1'b0 : 1'bz; da_oe)? 1'b0 : 1'bz;
□pd_project u0 (.clk_clk .reset_reset_n	(clock), (reset_n),
.i2c_master_sda_in .i2c_master_scl_in .i2c_master_sda_oe .i2c_master_scl_oe	(sda_in), // i2c_master.sda_in (scl_in), // .scl_in (sda_oe), // .sda_oe (scl_oe), // .scl_oe
); endmodule	

Platform Designer で接続した例です。

3 † : 8	bystem (Contents 🐰 Address Map 🕅	Interconnect Requirements	8					
	* *	System: soc_system Path:	slave_mem				-		
	Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
			⊟ clk_0	Clock Source					
×			clk_in	Clock Input	clk	exported			
			clk_in_reset	Reset Input	reset				
			clk	Clock Output	Double-clic	clk_0			
X			clk_reset	Reset Output	Double-clic				
^			曰 멜 nios2_gen2_0	Nios II Processor		12.100.0100			
w.		• · · · · · · · · · · · · · · · · · · ·	clk	Clock Input	Double-clic	clk_0			
X		I + →	reset	Reset Input	Double-clic	[clk]			
			data_master	Avalon Memory Mapped Master	Double-clic	[clk]			
			instruction_master	Avalon Memory Mapped Master	Double-clic	[clk]			
			irq	Interrupt Receiver	Double-clic	[clk]	IRQ 0	IRQ 31	5
			debug_reset_request	Reset Output	Double-clic	[clk]			
		$ \uparrow \uparrow \circ \longrightarrow$	debug_mem_slave	Avalon Memory Mapped Slave	Double-clic	[clk]	• 0×0002_0000	0×0002_07ff	
		×	custom_instruction_m	Custom Instruction Master	Double-clic				
			🗆 niosmem	On-Chip Memory (RAM or ROM)	2 100 00				
			sl	Avalon Memory Mapped Slave	Double-clic	[clk1]	0x0000_0000	0×0001_ffff	
			s2	Avalon Memory Mapped Slave	Double-clic	[clk1]	0x0000_0000	0×0001_ffff	
			clk1	Clock Input	Double-clic	clk_0			
		$ \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet$	reset1	Reset Input	Double-clic	[clk1]			
		₽ ₽ ₽ ♀ ₽ 	🗄 jtag_uart_0	JTAG UART		clk_0		0×0002_1007	2
			🗆 🖳 msgdma_tx	Modular Scatter-Gather DMA					
			mm_read	Avalon Memory Mapped Master	Double-clic	[clock]			
		∲ 	clock	Clock Input	Double-clic	clk_0			
			reset_n	Reset Input	Double-clic	[clock]			
			csr	Avalon Memory Mapped Slave	Double-clic	[clock]	= 0x0002_3000	0×0002_301f	
			descriptor_slave	Avalon Memory Mapped Slave	Double-clic	[clock]	= 0x0002_4000	0×0002_400f	
			csr_irq	Interrupt Sender	Double-clic	[clock]			
			st_source	Avalon Streaming Source	Double-clic	[clock]			
			⊟ i2c_0	Altera Avalon I2C (Master)					
		▶ Ⅰ → ↓ ↓ →	clock	Clock Input	Double-clic	clk_0			
			reset_sink	Reset Input	Double-clic	[clock]			
			interrupt_sender	Interrupt Sender	Double-clic	[clock]			Þ-Ó
			csr	Avalon Memory Mapped Slave	Double-clic	[clock]	= 0x0002_5000	0×0002_503f	
		00	i2c_serial	Conduit	i2c_master				
			rx_data_source	Avalon Streaming Source	Double-clic	[clock]			
			transfer_command_sink	Avalon Streaming Sink	Double-clic	[clock]			
	•		🖯 fifo_rx	Avalon FIFO Memory					
		♦ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	clk_in	Clock Input	Double-clic	clk_0			
		↓ ↓ ↓ ↓	reset_in	Reset Input	Double-clic	[clk_in]			
			in	Avalon Streaming Sink	Double-clic	[clk_in]			
			out	Avalon Memory Mapped Slave	Double-clic	[clk_in]	= 0x0002_6000	0×0002_6007	
			in_csr	Avalon Memory Mapped Slave	Double-clic	[clk_in]	0x0002_7000	0x0002_701f	
			in_irq	Interrupt Sender	Double-clic				3



4. <u>実装</u>

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。

* DMA Settings	2.
DMA Mode:	Memory-Mapped to Streaming
Data Width:	16 -
☑ Use pre-determined master address width	</td
Pre-determined master address width:	32
Expose mSGDMA read and write master's stread	ming ports
Data Path FIFO Depth:	16
Descriptor FIFO Depth:	
Response Port:	Disabled
Maximum Transfer Length:	
Transfer Type:	
Transfer Type	Full word Accesses Unly
	C Aligned Accesses
	C Unaligned Accesses
Eurst Enable	
Maximum Burst Count:	2 💌
🔲 Force Burst Alignment Enable	
* Extended Feature Options	
Enable Extended Feature Support	
Stride Addressing Enable	
Maximum Stride Words:	1
Programmable Burst Enable	J.
* Streaming Options	
Packet Support Enable	
Error Enable	
Error Width:	8 💌
🗖 Channel Enable	
Channel Width:	
Pre-Fetching Options	
Enables Pre-Fetching module	
Enable bursting on descriptor read master	
Data Width of Descriptor read/write master data pa	th: 22 💌
Maximum Burst Count on descriptor read master:	

ALTIMA

[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

Parameters	and the sud-sector data fits and the fits
iteriace for transfe	Avalon-ST
Depth of Fifo:	4 -
Basic options	
Depth.	64
Allow backpress	sure
Clock setting:	Single clock mode 💌
FIFO implementatio	Construct FIFO from embedded memory blocks
Status port	
🔽 Create status ir	terface for input
Create status in	terface for output
Enable IBQ for	status ports
Input	
input type:	AVALONST_SINK
Output	
Output Output type:	AVALONIMM DEAD
	AVALONMIM_READ
Avalon-MM port	settings
Data width:	32 💌
Avalon-ST port	settings
Bits per symbol:	8
Symbols per beat:	4
Error width:	0
Channel width:	
I Enable packet d	ata
Parameters	
I2C Slave Address:	10x55
Byte Addressing mo	
No. of Address Stan	
NO. OF MUURESS OTEA	
nable Read only mo	ide: Off 💌

Off 💌

= address width a is 256.0 Byte

Total byte address width

Туре:	RAM (Writable)
Dual-port access	
Circle electron	
Pand During White Made	<u></u> -
Nedu During write mode.	
Block type:	
Tightly Coupled Memory ope	eration require dual port & dual clock sources.
Size	
Enable different width for Dua	al-port access
Slave S1 Data width:	8
Total memory size:	256 bytes
Minimize memory block usage	e (may impact imax)
Read latency	
Slave s1 Latency	1
Slave s2 Latency	1 -
Slave s2 Latency ROM/RAM Memory Protecti	on The second se
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request:	n Enabled
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request: ECC Parameter	on Enabled
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support	I I
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request: ECC Parameter Extend the data width to support Memory initialization	I I
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support Memory initialization Imitialize memory content	Enabled
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support Memory initialization initialize memory content Enable non-default initializati	I I
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request: ECC Parameter Extend the data width to support Memory initialization imigation for the field of the second second imigation for the field of the second sec	
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support Memory initialization Initialize memory content Initialize memory content Initialize memory content Type the filename (e.g. m) User created initialization file:	
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support Memory initialization ✓ Initialize memory content C Enable non-default initializati Type the filename (e.g: my User created initialization file:	
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support Memory initialization Imitialize memory content Enable non-default initializati Type the filename (e.g: my User created initialization file: Enable Partial Reconfiguration	
Slave s2 Latency: ROM/RAM Memory Protecti Reset Request ECC Parameter Extend the data width to support Memory initialization Image: Initialize memory content Enable non-default initializati Type the filename (e.g. my User created initialization file Enable Partial Reconfiguration Enable In-System Memory Co	

+ address stealing bit



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 のプログ	s2			
	ラムメモリ	(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-	s1	
	Chip		
	RAM		
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

		🗆 🖳 nios2_gen2_0	Nios II Processor					
+		clk	Clock Input	Double-clic	clk_0			
		reset	Reset Input	Double-clic	[clk]			
		data_master	Avalon Memory Mapped Master	Double-clic	[clk]			
		instruction_master	Avalon Memory Mapped Master	Double-clic	[clk]			
	$ \longrightarrow $	irq	Interrupt Receiver	Double-clic	[clk]	IRQ 0	IRQ 31	5
		debug reset request	Reset Output	Double-clic	[clk]		-	
	$\bullet \bullet \circ \circ \to$	debug mem slave	Avalon Memory Mapped Slave	Double-clic	[clk]	0x0002 0000	0×0002 07ff	
	×	custom instruction m	Custom Instruction Master	Double-clic				
		🗆 niosmem	On-Chip Memory (RAM or ROM)					
	$\bullet \bullet \circ \circ \circ \to$	sl	Avalon Memory Mapped Slave	Double-clic	[clk1]	0x0000 0000	0×0001 ffff	
	\diamond \diamond \diamond \diamond \rightarrow	\$2	Avalon Memory Mapped Slave	Double-clic	[clk1]	0×0000 0000	0×0001 ffff	
+		clk1	Clock Input	Double-clic	clk0	_	-	
	↓ →	reset1	Reset Input	Double-clic	[clk1]			
++	$\bullet \diamond \bullet \bullet \bullet \to$	⊞ itag uart 0	JTAG UART		clk 0	0×0002 1000	0×0002 1007	-3
		🖂 🖳 msedma tx	Modular Scatter-Gather DMA		-	_	-	Ĭ
		mm read	Avalon Memory Mapped Master	Double-clic	[clock]			
+	`	clock	Clock Input	Double-click	clk 0			
		reset n	Beset Input	Double-clic	[clock]			
		CST	Avalon Memory Manned Slave	Double-clic	[clock]	0×0002 3000	0×0002 301f	
		descriptor slave	Avalon Memory Mapped Slave	Double-clic	[clock]	0×0002_0000	0×0002_400f	
		est in	Interrupt Sender	Double-clic	[clock]			
		st source	Avalon Streaming Source	Double~clic	[clock]			l l
		E i2c 0	Altera Avalon I2C (Master)	Double cite	[olocid]			
		clock	Clock Input	Double-clic	-11-0			
		recet cink	Reset Input	Double-clic	feleck1			
		interrupt sender	Interrupt Sender	Double-clic	[clock]			
		eer	Avalon Memory Manned Slave	Double-clic	[clock]	0,0002 5000	0∨0002 503f	Ĩ
		i)o cerial	Conduit	i2c master	LCIOCK1	- 020002_0000	0,0002_0001	
		rx data cource	Ausion Streaming Source	Deublerelie	[alaok]			
		transfer command cink	Auston Streaming Source	Double circ	[clock]			
		E fife ry	Avalon ETEO Memory	Double che	LCIOCK			
			Clock Input	Daublanatia				
T L		reset in	Popot Input	Double cite	Colk_in]			
		in	Austan Streaming Sink	Double chc	[CIK_III] [CIK_in]			
		out	Avalon Memory Mapped Slave	Double clic	Colk_in]		0~0002 6007	
		in cor	Auston Memory Mapped Stave	Double clic	[CIK_IN]	0x0002_0000	0x0002_0007	
		in ing	Intervent Sender	Double che	ICIK_III	- 0,0005_1000	0,0002_7011	
		in_ing □ inealaus te sulmm	Alterna IOC Slave Te Avales MM Master Bridge	Double clic				
		E I2CSIAVE_to_avimm	Clearly Travet	0.41.5				
			Clock input	Double-clic	CIK_U			
		avaion_master	Product memory mapped master	Double-clic	CIUCK]			
		reset	neset input	Double-clic	UUUCK J			
	00	conauit_ena	Conduit	12C_Slave				
		⊡ slave_mem	On-Only Memory (KAWI or KOW)	0.11	-u. 11	A 0.0000 0000	0.0000.0055	
		51	Avaluar Memory Mapped Slave	Double-chc	[CIK I] I		0.0000_0011	
		S2	Avaion Memory Mapped Slave	Double-clic		0X0002_8000	UXUUU2_8011	
		CIK I	Diock input	Double-clic	CIK_0			
		recett	Reset Indut	12 martial in more large	ICIK III			

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。

1'bz; 1'bz;
'bz; 'bz;
// Clock 50MHz // Reset Switch
// i2c_master.sda_in // scl_in // sda_oe // .scl_oe
// i2c_slave.conduit_data_in // .conduit_clk_in // .conduit_data_oe // .conduit_clk_oe



Assignment Editor でピン配置してください。

ta	tu From	То	Assignment Name	Value	Enabled	Entity
1 🗸	•	- FPGA_CLK1_50	Location	PIN_V11	Yes	
2 🗸	 International 	- FPGA_CLK1_50	I/O Standard	3.3-V LVTTL	Yes	i2c_test
3 🗸	1	🖳 КЕҮ[0]	Location	PIN_AH17	Yes	
4 🗸	·	ів_ КЕУ[0]	I/O Standard	3.3-V LVTTL	Yes	i2c_test
5 🗸	1	💁 КЕҮ[1]	Location	PIN_AH16	Yes	
6 🗸	·	🖳 КЕҮ[1]	I/O Standard	3.3-V LVTTL	Yes	i2c_test
7 🗸	 International 	i2c_master_scl	Location	PIN_V12	Yes	
8 🗸	 International 	峯 i2c_master_scl	I/O Standard	3.3-V LVTTL	Yes	i2c_test
9 🗸	1	迄 i2c_master_sda	Location	PIN_AF7	Yes	
10 🗸	 International 	峯 i2c_master_sda	I/O Standard	3.3-V LVTTL	Yes	i2c_test
11 🗸	1	i2c_slave_scl	Location	PIN_W12	Yes	
12 🗸	·	i2c_slave_scl	I/O Standard	3.3-V LVTTL	Yes	i2c_test
13 🗸	·	峯 i2c_slave_scl	Weak Pull-Up Resistor	On	Yes	i2c_test
14 🗸	1	迄 i2c_slave_sda	Location	PIN_AF8	Yes	
15 🗸	·	峯 i2c_slave_sda	I/O Standard	3.3-V LVTTL	Yes	i2c_test
16 🗸	1	迄 i2c_slave_sda	Weak Pull-Up Resistor	On	Yes	i2c_test
17	< <new>></new>	< <new>></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。



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



/ On-Chip FIFO 割り込みハンドラ	
atic void fifo_callback(void * context)	
int status;	_/ On-Chip FIFO に貯まったデータの数が 10-word
alt_u32 csr = (alt_u32)context;	~ (main 問数内の altera avalon fifo init() 閉数に
att_itt_concert cpu_st,	
// 全書的込みをディセーブル	て ALMOSTFULL の値を 10 と設定)以上になる
<pre>cpu_sr = alt_irq_disable_all();</pre>	
// FIFO のステータスを読み出す	と割り込みか生成され、本ハントフかコールされ
<pre>status = altera_avalon_fifo_read_status(csr, ALTERA_AVALON_FIF0_IENABLE_ALL);</pre>	ます 木ハンドラでは FUIL か ALMOSTFULL の
// 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)	⁾ 検出で stop 変数に TRUE を設定し、main 関数
i stop = TRUE;	
}	内のルーノを中断させま9。
<pre>// 1٨>F00000 altera avalon fifo clear event(csr, (alt u32)status);</pre>	
// 全書的込みをイネーブル	
<pre>alt_irq_enable_all(cpu_sr);</pre>	
I2C Master 割り込みハンドラ	」 I2C Master でエラーが発生した場合等にコールる
atic void i2c_callback(void * context)	
ALT AVAION T2C DEV + *i2c dev = (ALT AVALON I2C DEV t *) context;	れよ9。本夫宏では、12Cの区文信は IIISODIVI
alt_u32 status;	→ にて行われ、データ単位でのステータスの監視に
<pre>alt_irq_context cpu_sr;</pre>	
// 全害的込みをディセーブル	行っておりませんので、エラーの発生は割り込み
<pre>cpu_sr = alt_irq_disable_all();</pre>	
// TOC Masterのフラーカフを読み出す	ハントフで検出しま9。
alt_avalon_i2c_int_status_get(i2c_dev, &status);	
// ステータスを出力(テスト用) nvintf("720 Macter Engen Integrunt"致いの", (int)status):	
printry ize mater error interruption (it's (intersectory)	
// 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);	
alf_avalon_itcf_int_arsabic(itc_ace,bci_intern_itcf_intern_itcf_intern_itcf_intern_itcf_intern_itcf_interneitcfinterneitc	
// I2C 割0込みをクリア al+ avalon i2c int clear(i2c dev,ALT AVALON_I2C_ISR_ALL_CLEARABLE_INTS_MSK);	
// T2C 実的込みをイネーブル	
alt avalor i2c enable(i2c dev):	
<pre>alt_avalon_i2c_enable(i2c_dev);</pre>	
alt_avalon_i2c_enable(i2c_dev); // 全書的込みをイネーブル	
alt_avlp_nitc_enable(i2c_dev); // 全書的込みをイネーブル alt_irq_enable_all(cpu_sr);	
alt_avln_i2c_enable(i2c_dev); // 全書的込みをイネーブル alt_irq_enable_all(cpu_sr);	
alt_asinofferproduction; // 全書的込みをイネーブル alt_irq_enable_all(cpu_sr);	
alt_avalon_i2c_enable(i2c_dev); // 全部D込みをイネーブル alt_irq_enable_all(cpu_sr); main function	
alt_avalon_i2c_enable(i2c_dev); // 全期D込みを(ネーブル alt_irq_enable_all(cpu_sr); main function t main()	
alt_avalon_i2c_enable(i2c_dev); // 全即)込みを(ネーブル alt_irq_enable_all(cpu_sr); main function t main()	r
alt_avalon_i2c_enable(i2c_dev); // 全即)込みを(ネーブル alt_irq_enable_all(cpu_sr); main function t main() int i;	
alt_avalon_i2c_enable(i2c_dev); // 全部)込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA	
alt_avalon_i2c_enable(i2c_dev); // 全部)込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dev *tx_dma; 	(
alt_avalom_izc_enable(i2c_dev); // 全即込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dtav *tx_dma; alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma status;	(
alt_avalom_izc_enable(i2c_dev); // 全期の込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma_status;	
alt_avalom_izc_enable(i2c_dev); // 全部D込みを(ネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma_status; // I2C Master Alt AVAION I2C DEV t *i2c dev;	1
alt_avalon_i2c_enable(i2c_dev); // 全期の込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_dev *tx_dma; alt_msgdma_dev *tx_dma; alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma_status; // I2C Master ALT_AVALOM_I2C_DEV t *i2c_dev; ALT_AVALOM_I2C_STATUS_CODE i2c_status;	
alt_avalon_i2c_enable(i2c_dev); // 全期の込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_mSgdma_dev *tx_dma; alt_mSgdma_dev *tx_dma; alt_mSgdma_dev *tx_dma; alt_mSgdma_dev *tx_dma; alt_mSgdma_dev *tx_dma; alt_mSgdma_dev *tx_dma; alt_mSgdma_tev *tx_dma; alt_MSGMA Alt_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_STATUS_CODE i2c_status; // crec	
<pre>alt_avalon_izc_enable(i2c_dev); // 全期Di2a%t(A-J)k alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_dev *tx_dma; 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_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_status; // FIF0 int fifo status;</pre>	
alt_avalon_i2c_enable(i2c_dev); // 全即込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dev *tx_dma; 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_DEV_t *i2c_dev;	
alt_avalon_i2c_enable(i2c_dev); // 全割的込みをイネーブル alt_irq_enable_all(cpu_sr); * main function * main function * main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_tendard_descriptor wr_desc, rd_desc; int dma_status; // I2C Master ALT_AVALOM_I2C_DEV_t *i2c_dev; ALT_AVALOM_I2C_STATUS_CODE i2c_status; // FIFO int fifo_status; // I2C Slave memory(非やヤジュ領域でポイン攻生成) unsigned char *slv buff = (unsigned char*)(I2C_SLAVE_MEM 0x8000000);	
alt_avalom_izc_enable(i2c_dev); // 全即込みをイネーブル alt_irq_enable_all(cpu_sr); main function t main() int i; // mSGDMA alt_msgdma_dtandard_descriptor wr_desc, rd_desc; int dma_standard_descriptor wr_desc, rd_desc; int fing_stands; // I2C Master ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_STATUS_CODE i2c_status; // FIFO int fifo_status; // I2C Slave memory(排音やジン領域でポインタ生成) unsigned char *slv_buff = (unsigned char*)(I2C_SLAVE_MEM 0x80000000);	中部亦物の知問化を行っています。
alt_avalom_izc_enable(i2c_dev); // 全即込みをイネーブル alt_irq_enable_all(cpu_sr); // amain function tt main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma_standard_descriptor wr_desc, rd_desc; int fifo_standard_descriptor wr_desc, rd_desc; // I2C Master ALT_AVALON_I2C_DEV_t *12c_dev; ALT_AVALON_I2C_DEV_t *12c_dev; ALT_AVALON_I2C_STATUS_CODE 12c_status; // FIF0 int fifo_status; // I2C Slave memory(排物+voyl编版で未分少生成) unsigned char *slv_buff = (unsigned char*)(I2C_SLAVE_MEM 0x8000000); // I2C Comman f_abuta ###%jdb;	内部変数の初期化を行っています。
alt_avalon_i2c_enable(i2c_dev); // 全部D込みをイネーブル alt_irq_enable_all(cpu_sr); * main function * main funct	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte y
alt_avalon_i2c_enable(i2c_dev); // 全部Di2A5e(ネーブル alt_irq_enable_all(cpu_sr); * main function * main func	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte y
<pre>alt_avalon_i2c_enable(i2c_dev); // 全割Di2b&tA-Jル alt_irq_enable_all(cpu_sr); * main function t main() int i; // mSGOMA alt_msgdma_dev *tx_dma; alt_msgdma_dev *tx_dma; // I2C Master Alt_AVALOM_I2C_DEV_t *i2c_dev; Alt_AVALOM_I2C_DEV_t *i2c_d</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte y 要で、上位 byte には Start/Stop Condition を出;
<pre>// drawalon_i2c_enable(i2c_dev); // 全部D込みをイネーブル alt_irq_enable_all(cpu_sr); // alt_irq_enable_all(cpu_sr); // main function tf main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_dev *tx_dma; alt_msgdma_totad_descriptor wr_desc, rd_desc; int dma_status; // I2C Master ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_STATUS_CODE i2c_status; // FIFO int fifo_status; // I2C Slave memory(排标v9/JI報版で未行少生成) unsigned char *slv_buff = (unsigned char*)(I2C_SLAVE_MEM 0x80000000); // I2C Command // アドレス 0x00[[4-byte を書き込む] unsigned char wr_cmd[][2] = {{0x00, 0x00}; // Write Data[0] {0x00, 0x00}; // Write Data[1] {0x00, 0x00}; // Write Data[1] {0x00, 0x00}; // Write Data[2]</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 4 要で、上位 byte には Start/Stop Condition を出ま する bit データ 下位 byte に详信データを設定
alt_avalon_i2c_enable(i2c_dev); // 全部的込みをイネーブル alt_irq_enable_all(cpu_sr); // main function int i; // mSGDMA alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma_status; // IZC Master ALT_AVALOM_IZC_DEV_t *i2c_dev; ALT_AVALOM_IZC_DEV_t *i2c_dev; ALT_AVALOM_IZC_STATUS_CODE i2c_status; // FIFO int fifo_status; // IZC 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); // IZC Command // TFL2 &xx00(2.4-byte を書参込5) unsigned char wr_cmd[][2] = {{0x82, 0xAA}, // Start , Device Address, W/R=0 {0x00, 0xAB}, // Write Data[0] {0x00, 0xDB}, // Write Data[1] {0x00, 0xDE}; // Write Data[3], Stop	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte y 要で、上位 byte には Start/Stop Condition を出 する bit データ、下位 byte に送信データを設定
<pre>alt_avalon_i2c_enable(i2c_dev); // 全部D込みを(ネーブル alt_irq_enable_all(cpu_sr); * main function tf main() int i; // mSGDMA alt_msgdma_standard_descriptor wr_desc, rd_desc; int dma_status; // I2C Master AlT_AVALOM_i2C_DEV_t *i2c_dev; AlT_AVALOM_i2C_DEV_t *i2c_dev; AlT_AVALOM_i2C_DEV_t *i2c_dev; AlT_AVALOM_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(2 4-byte を書意込む) unsigned char wr_cmd[][2] = {0x02, 0xAA}, // Start , Device Address, W/R=0 {0x00, 0x02}, // Write Data[3] {0x00, 0x02}, // Write Data[3], Stop // 7FLX 0x000が6,4-byte を読み出す (0x01, 0x02); // Write Data[3], Stop // 7FLX 0x000が6,4-byte を読み出す unsigned char rd_cmd[][2] = {0x02, 0xAA}, // Start , Device Address, W/R=0</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 如 要で、上位 byte には Start/Stop Condition を出 する bit データ、下位 byte に送信データを設定 ます。
alt_avalon_i2c_enable(i2c_dev); // 全部D込みを{木-ブル alt_irq_enable_all(cpu_sr); * main function * main functi	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 如 要で、上位 byte には Start/Stop Condition を出た する bit データ、下位 byte に送信データを設定 ます。
<pre>alt_avalon_i2c_enable(i2c_dev); // 全割Di2b&d/A-J/L alt_irq_enable_all(cpu_sr); * main function * main() int i; // mSGDMA alt_msgdma_dev *tx_dma; alt_msgdma_tandard_descriptor wr_desc, rd_desc; int dma_status; // I2C Master Alt_AVALOM_I2C_DEV_t *i2c_dev; ALT_AVALOM_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 0x8000000); // I2C Command // アドレス 0x00(2 + byte を書参込む unsigned char wr_cmd[][2] = {{0x00, 0x00}; // Write Data[0] {0x00, 0x00}; // Write Data[1] {0x00, 0x00}; // Write Data[2] {0x00, 0x00}; // Write Data[3], Stop // アドレス 0x00/b64-byte を読み出す unsigned char rd_cmd[][2] = {{0x02, 0xA}; // Start, Device Address, W/R=0 {0x00, 0x00}; // Katrt, Device Address, W/R=0 {0x00, 0x00}; // Read Address, Stop {0x02, 0xA0}; // Start, Device Address, W/R=1 {0x00, 0x00}; // Start, Device</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 如 要で、上位 byte には Start/Stop Condition を出た する bit データ、下位 byte に送信データを設定し ます。
<pre>alt_avalon_i2c_enable(i2c_dev); // 全部D込為を{A-ブル alt_irq_enable_all(cpu_sr); // amain function * main func</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 如 要で、上位 byte には Start/Stop Condition を出た する bit データ、下位 byte に送信データを設定し ます。
<pre>alt_avalon_i2c_enable(i2c_dev); // 全部D込みをイネーブル alt_irq_enable_all(cpu_sr); * main function * main funct</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 如 要で、上位 byte には Start/Stop Condition を出 する bit データ、下位 byte に送信データを設定し ます。
<pre>alt_avalon_i2c_enable(i2c_dev); // 全部D込みをイネーブル alt_irq_enable_all(cpu_sr); * main function t main() int i; // mSGDMA alt_msgdma_eve *tx_dma; alt_msgdma_eve *tx_dma; alt_msgdma_etandard_descriptor wr_desc, rd_desc; int dma_status; // I2C Master ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; ALT_AVALON_I2C_DEV_t *i2c_dev; MIT_AVALON_I2C_DEV_t *i2c_d</pre>	内部変数の初期化を行っています。 12C のコマンドは、データ 1-byte あたり 2-byte 娘 要で、上位 byte には Start/Stop Condition を出た する bit データ、下位 byte に送信データを設定し ます。 12C Slave のメモリは OxFF で Fill する事で、書き



// === I2C Master 間違の分別明化 === // T2C Master 間違の分別明化 ===	
<pre>// Inter Master alt_avalon_izc_open(IZC_MASTER_NAME); if (NULL == i2c_dev) { printf("Error:I2C Mater Open Fail\n"); return FALSE; }</pre>	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。
<pre>// 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, 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)</pre>	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。
<pre>printf("Error:I2C Mater Enable Fail\n"); return FALSE; }</pre>	
// === On-Chip FIFO 関連の初期化 === // On-Chip FIFO の初期期化(10-word 審積されたち ALMOSTFULL 期的込み発生) fifo_status = altera_avalon_fifo_init(FIFO_CSR, (ALTERA_AVALON_FIFO_IENABLE_AF_MSK ALT if(ALTERA_AVALON_FIFO_OK != fifo_status)	<pre>ERA_AVALON_FIFO_IENABLE_F_MSK), 1, 10);</pre>
{ printf("Error:FIFO init Fail[%d]\n", fifo_status); return FALSE; } // On-Chip FIFO 割込み/\と片を登録 alt is is register(FIFO IED CTEL ID ETED IED fifo callback (woid#\ETED CSE @x@);	On-Chip FIFO の初期化と割り込みハンドラの 登録を行います。
al_le_ss_egate()// of all condition () / / / / / / / / / / / / / / / / / /	
<pre>if(NULL == tx_dma) { printf("Error:TX mSGDMA Open Fail\n"); return FALSE;</pre>	mSGDMA の初期化を行います。
} for(i = 1;;i ++) { // ችላማንጋቢ7ንምንጊ alt deache fluch all();	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用ディスクリブタの登録 dma_status = alt_msgdma_construct_standard_mm_to_st_descriptor(tx_dma, ≀_desc, (a ALTERA_MSGDMA_DESCRI	<pre>lt_u32*)wr_cmd, sizeof(wr_cmd), PTOR_CONTROL_TRANSFER_COMPLETE_IRQ_MASK);</pre>
if(@ != dma_status) { printf("Error:DMA descriptor Fail[派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, ≀_desc); if(@ != dma_status) { printf("Error:DMA async_trans_Fail[%d]\n", dma_status);	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。
return FALSE; } // I2C Read用ディスクリブタの登録 dma_status = alt_msgdma_construct_standard_mm_to_st_descriptor(tx_dma, &rd_desc, (a ALTERA_MSGDMA_DESCRI	lt_u32*)rd_cmd, sizeof(rd_cmd), PTOR_CONTROL_TRANSFER_COMPLETE_IRQ_MASK);
if(@ != 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(@ != dma_status) { printf("Error:DMA async_trans Fail[%d]\n", dma_status); return EAVES.	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。
<pre>} // 書込みアドレス/値を変更 wr_cmd[1][1] = i * 0x10; // 書込みアドレス = i * 0x10 wr_cmd[2][1] = i + 1; // 書込みデータ[0] = i + 1 wr_cmd[3][1] = i + 2; // 書込みデータ[0] = i + 2 wr_cmd[4][1] = i + 3; // 書込みデータ[0] = i + 3 wr_cmd[5][1] = i + 4; // 書込みデータ[0] = i + 4</pre>	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。
// 読出しアドレスを変更 rd_cmd[1][1] = i * 0×10; // 読出しアドレス = i * 0×10 // On-Chip FIFO の書的込みで FULL か ALMOSTFULL(10) なら終了 if(stop == TRUE) { break; }	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);	
<pre>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("%08X,", data); } printf("\n\n=== I2C Slave Dump ==="); // I2C Slave メモリの珍ンプ dumo(club buff C SLAVE SUTE);</pre>	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。
return TRUE; }	I2C Slave メモリのダンプを出力します。
/*************************************	メモリ・ダンプ用の関数です。
<pre>Void dump(unsigned char "adr, int size) { int i; unsigned char ucData; // ギャッジュフラッシュ 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); } } }</pre>	
<pre>printf("\n"); }</pre>	

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。 woorojseJatros-oueu-Oap-po//:dtty woj OD DOS-OUEN-OBO PEOJUMOO



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 2c_test Nios II Hardware configuration - cable: DE-SoC on localhost [USB-1] device ID: 2 instance ID: 0 name; jtaguart_0

Read Count:10

Read Data:DECDBCAB,05040302,06050403,07060504,08070605,09080706,0A090807,0B0A0908,0C0B0A09,0D0C0B0A,



6. <u>補足</u>

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. <u>参考資料</u>

- Embedded Peripherals IP User Guide <u>https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf</u>
- 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 <u>https://service.macnica.co.jp/library/list?sort%5Blibrary_publish_at2_d%5D=desc&tag=Nios+II</u>
- Nios II FAQ <u>https://service.macnica.co.jp/support/faq/list?sort%5Bfaq_publish_at2_d%5D=desc&tag=Nios+II</u>



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。
- 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 <u>https://www.alt.macnica.co.jp/</u> 技術情報サイト アルティマ技術データベース <u>http://www.altima.jp/members/</u>
- 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、各メーカ発行の英語版の資料もあわせてご利用ください。