

Nios II SBT によるソフトウェア開発 セクション 3

ver.14



2015年5月 Rev.1

ELSENA,Inc.





Nios II SBT によるソフトウェア開発 セクション 3

<u>目次</u>

1.	はじめに	3
2.	JTAG デバッグ・コア	3
2	-1. Nios II CPU の設定	3
2	-2. JTAG デバッグ・コア	ł
2	-3. リモート・デバッグ	5
3.	Nios II デバッグ機能	5
3	-1. Nios II デバッグ・オプション	5
	3-1-1. Nios II SBT 起動オプション	5
	3-1-2. スタック・オーバーフロー・チェックのオプション	3
3	-2. マルチ・プロセッサ・システムのデバッグ)
	3-2-1. マルチ・プロセッサ用ソフトウェア・プロジェクトの作成)
	3-2-2. マルチ・プロセッサの起動10)
4.	コードのパフォーマンス検証11	l
4	-1. パフォーマンス・カウンタ11	l
	4-1-1. パフォーマンス・カウンタの概要11	l
	4-1-2. パフォーマンス・カウンタの使用例12	2
4	-2. プロファイラ14	ł
	4-2-1. プロファイリング・データの生成方法14	ļ
	4-2-2. プロファイリング・データの解析14	ļ
4	-3. パフォーマンス・カウンタ vs プロファイラ15	5
改覑	反履歴16	5

1. <u>はじめに</u>

この資料は、Nios[®] II Software Build Tool(Nios II SBT)によるソフトウェア開発について紹介しています。セクション 3 で取り上げている内容は、以下のとおりです。

- □ JTAG デバッグ・コア
- □ Nios II デバッグ機能
- □ コードのパフォーマンス検証

2. JTAG デバッグ・コア

この章では、Nios II をデバッグするために使用する JTAG デバッグ・コアについて説明します。

2-1. Nios II CPU の設定

JTAG デバッグ・コアはシステムの中に実装され、ホスト PC から Nios II とターゲットとなる Qsys システムをコ ントロールすることを可能にします。JTAG デバッグ・コアを使用する際には、Qsys の Nios II CPU のウィザードの JTAG Debug Module タブでデバッグ・レベルを選択します。

デバッグ・レベルは、Level 1 から 4 まで選択することができ、デバッグ・レベルを上げると使用できるサポート機能が増えます。No debugger は JTAG デバッグ・コアをシステムの中に組み込まない時に設定します。

🔓 Nios II Processor - nios2_qsys_1 🔀							
Nios II Processor altera_nios2_gsys							
Block Diagra Block Diagra Core Nos II Caches and Memory Interfaces Advanced Features MMJ and MPU Settings JTAG Debug Module Show signals Select a Debugging Level Debug level: No Debugger clk C Level 1 reset: n C Level 2 d ing C Level 3 tag debug mo C Level 4							
	No Debugger	Level 1	Level 2	Level 3	Level 4		
JTAG Target Connection Download Software Software Breakpoints 2 Hardware Breakpoints 2 Data Triggers 2 Hardware Breakpoints 2 Data Triggers 2 Hardware Breakpoints 2 Data Triggers 2 Hardware Breakpoints 2 Data Triggers 2 Hardware Breakpoints 2 Hardware				JTAG Target Connection Download Software Software Breakpoints 4 Hardware Breakpoints 4 Data Triggers Instruction Trace Orr-Chip Trace Data Trace Off-Chip Trace			
	No M9Ks	One M9K	One M9K	One M9K + trace	One M9K + trace		
	☐ Include debugack signals These signals appear on the top-level Qeys system. You must manually connect these signals to logic external to the Qeys system.						
	▼ Break Vector						
Break vector memory: nios2_csys[1].tag.debug.module Break vector offset: 0x0000020 Break vector: 0x0000020							
	Advanced Debug Settings						
CL Onchip Frace: 128 Frames - Image: The second state internal 2x clock signal							
	Advance debug licenses can be purchased from MPS Technologies, Inc. http://www.mips.com/fs2redirect.htm						
Error: nios2_qsys_1: Reset slave is not specified. Please select the reset slave Error: nios2_qsys_1: Exception slave is not specified. Please select the exception slave							
Cancel Finish							

2-2. JTAG デバッグ・コア

ホスト PC は、JTAG デバッグ・コアを介して Nios II プロセッサと Qsys システム(ターゲット)に接続し、Nios II SBT デバッガにてデバッグを行います。

JTAG デバッグ・コアには、2つのインタフェースがあります。

- ◆ JTAG インタフェースでホストに接続(JTAG HUB ロジック)
 - JTAG プログラミング・ケーブルを使用
- ◆ デバッグの対象プロセッサのアバロン・スレーブ・インタフェース
 - プロセッサから JTAG デバッグ・コアのメモリとコントロール・レジスタへアクセス

JTAG デバッグ・コアのメモリはオンチップ・メモリを使用し、1024 バイト(256 ワード(32 ビット))です。JTAG 経由 でリード・ライトを行うことができ、アバロンのインタフェースからは上位 256 バイトのみ書き込みできます。

JTAG HUB モジュールで JTAG ノードと JTAG ピンの切り替えを行います。JTAG HUB モジュールは、 Quartus[®] II 開発ソフトウェアの合成時に自動で挿入されます。JTAG HUB モジュールによって JTAG デバッグ・コ ア、Signal Tap[®] II、JTAG UART 等のノードを共有します。





上図のシステム例には Nios II プロセッサと JTAG UART コアが含まれています。ホスト PC 上のデバッガ、 JTAG ターミナルとの接続は1つのダウンロード・ケーブルを使用します。JTAG サーバ・ソフトウェアによって各ホス ト・アプリケーションは独立してターゲットに接続することができます。ここで言う、JTAG ターミナルとは、System Console や Nios II Console(Eclipse 上のターミナル)、SignalTap II などのホストPC 上で動作する各種ソフトウェアを 指します。

2-3. リモート・デバッグ

JTAG サーバはリモート・デバッグをサポートしており、遠隔地のホストを JTAG デバッグ・サーバとして設定し、 Quartus II 開発ソフトウェアのプログラマと Nios II SBT により遠隔からターゲットをコントロールすることができま す。

ボードが接続された PC を遠隔クライアントから操作できるようにするために、リモート JTAG サーバの設定を行います。

Quartus II 開発ソフトウェアのプログラマ・ウィンドウにて、リモート・クライアントを有効にし、接続するためのパス ワードを設定します。

Programmer – C:/workshop File Edit View Processing Tools	/141128_sw_v140/BeMicro_140/BeMicro_AdvBtCpr - BeMicro_AdvBtCpr - [Chain1.cdf] Window Help 👂	_ 🗆 🗙
Hardware Setup No Hardw Enable real-time ISP to allow ba Stop Auto Detect Change File Change File Add Device Change File Add Device Change File Chang	Hardware Setup TAG Settings TAG Settings Specify JTAG servers to add and remove from the JTAG Servers list. Specify the password used by remote clients to access the local JTAG server Y Erase ISS JTAG Servers Configure Local JTAG Servers d Server JTAG Servers Configure Local JTAG Server d Server Password: OK Cancel OK Configure Local JTAG Server Close Close	2 P

遠隔の PC と手元の PC を接続するために、手元の PC にて Quartus II 開発ソフトウェアのプログラマでリモート・サーバの名前を入力し、サーバ側で設定したパスワードを入力します。

サーバが登録されますと、手元の PC の Quartus II 開発ソフトウェアのプログラマからリモートのサーバに接続 されている使用可能な USB-Blaster™ が認識されるので、使用するプログラミング・ハードウェアに設定して書き込 みを行います。

Programmer – C:/workshop File Edit View Processing Tools	p/141128_sw_v140/BeMicro_140/BeMicro_AdvBtCpr - BeMicro_AdvBtCpr - [Chain1.cdf]
File Edit View Processing Tools	Window Help Search altera.c.
M Save File Mdd Device ↑ th Up ↓ th Down	Configure Local JTAG Server

🔥 ALTIMA

3. <u>Nios II デバッグ機能</u>

この章では、Nios II SBT のデバッグ機能のより高度なデバッグにて使用できるオプションを説明します。

3-1. Nios II デバッグ・オプション

3-1-1. Nios II SBT 起動オプション

Run メニュー ⇒ Debug Configurations... の Target Connection タブと Debugger タブにて、デバッガ起動オプションの設定を行います。下記のオプションの設定を行うことができます。

◆ System ID checks(Target Connection タブにて設定)

ホストPCにある実行コード(.elf)の System ID と接続先の FPGA にある System ID のチェックに関する設定です。

Ignore mismatched system ID

チェックを入れると、Qsys で設定した System ID のチェックを行いません。

- Ignore mismatched system timestamp
 チェックを入れると、System ID のタイムスタンプのチェックを行いません。
- Download (Target Connection タブにて設定)

デバッグ時のエントリ・ポイントを設定します。

• Download ELF to selected target system

ホスト PC から実行コード(.elf)を実行領域に設定されている RAM にダウンロードします。チェックが無い場合、実行中のプログラムにアタッチします。

Reset the selected target system
 実行コード(.elf)をダウンロードした後に、Nios II をリセットします。

◆ Stop on startup at (Debugger タブにて設定)

デバッグ時にエントリ・ポイントを設定するかの設定になります。チェックを入れた場合、デバッグ時に指定した 関数(default: main)の先頭で停止します。

<u>デバッガ設定画面 (Target Connection タブ)</u>

🖨 Debug Configurations		×			
Create, manage, and run configurations The expected Stdout device name does not match the selected target byte stream device name.					
Ype filter text C C/C++ Application C C/C++ Attach to App C C/C++ Postmortem I C C/C++ Remote Applic Launch Group Nios II Hardware Nios II Hardware v2 (Name: New_configuration				
Filter matched 8 of 8 items	Apply	Revert			
?	Debug	Close			

<u>デバッガ設定画面 (Debugger タブ)</u>

🖨 Debug Configurations		×
Create, manage, and run Nios II Hardware Tab Group	n configurations	1 A
Image: Second system Image: Second system Image: Secon	Name: New_configuration	
Filter matched 8 of 8 items		Revert
?	<u>D</u> e bug	Close



3-1-2. スタック・オーバーフロー・チェックのオプション

デバッグ実行時にメモリのスタック・オーバーフロー・チェックを行うことができます。

BSP Editor の Main タブの Advanced にある enable_runtime_stack_checking のオプションをオンにすることによって、関数のエントリ前にメモリのスタック・ポインタがスタック・リミットを超えていないかを比較し、超えた場合にはプログラムを停止するための break の命令が挿入されます。

スタック・オーバーフローが発生した場合には、プログラムは break 命令が実行され停止します。この break はデ バッガの逆アセンブラビューで表示され、確認することができます。

hal —					
	max_file_descriptors:	32			
	enable_instruction_related_exceptions_	api			
	log_port:	none 💌			
	🔽 enable_exit				
	🔽 enable_clean_exit				
	enable_runtime_stack_checking				
	🗖 enable_c_plus_plus				
	✓ enable_lightweight_device_driver_api				
	<pre>enable_mul_div_emulation</pre>				
	<pre>enable_sopc_sysid_check</pre>				
	custom_newlib_flags:	none			
	log_flags:	0			

3-2. マルチ・プロセッサ・システムのデバッグ

Qsys システムでは複数の Nios II を組み込んだシステムを構築することができます。

Nios II SBT はマルチ・プロセッサ・デバッグに対応しており、プロセッサ毎にデバッグを行うことができます。ター ゲットとなるプロセッサを操作している時に、その他のプロセッサも同時にデバッグすることができます。

3-2-1. マルチ・プロセッサ用ソフトウェア・プロジェクトの作成

マルチ・プロセッサ用のソフトウェア・プロジェクトを作成する場合には、プロセッサ毎にプロジェクトを作成します。 新規プロジェクト・ウィザードにて選択する Qsys システムの .sopcinfo ファイルは共通ですが、システム内のどのタ ーゲットとなるプロセッサかを CPU: の項目にて設定し、ソフトウェア・プロジェクトを作成します。

Nios II Application and BSP from Template			
Nios II Software Examples			
Create a new application and boa template	rd support package based on a software example		
Target hardware information SOPC Information File name:	C:¥work¥BeMicro_140¥qsys_top.sopcinfo		
CPU name:	nios2_1		
Project name: hello_world			

※ 複数のプロセッサのプログラム・メモリを共有のメモリに割り当てる際には、それぞれのプロセッサ用にパーティションを分ける必要があります。プロセッサ毎のメモリのパーティションは、それぞれのプロセッサの設定の エクセプション・アドレスを先頭アドレスとして分けることができます。 3-2-2. マルチ・プロセッサの起動

各プロセッサをひとつのグループとして登録し一括で実行、デバッグを Nios II SBT 上から行うことができます。

複数のプロジェクトを一括起動するために、複数のプロジェクトをグループにまとめます。

Run メニュー ⇒ Debug Configurations... Nios II Hardware より、プロセッサ毎にデバッグ時のオプションを設定し ます。Target Connection タブの Connections の Processors にて、ターゲットとなるプロセッサのインスタンス ID を 選択します。マルチ・プロセッサ・システムの場合には必ずインスタンス ID を選択する必要があります。

次に Launch Group で一括起動するプロジェクトを選択しグループを作成し、Debug で実行します。

E Debug Configurations		×
Create, manage, and run c	onfigurations	- AC
Ype filter text C /C++ Application C /C++ Attach to Application C /C++ Postmortem Debi C /C++ Remote Application C /C++ Remote Application Launch Group New_configuration (1) Nios II Hardware nios2_1 nios2_2 Nios II Hardware v2 (beta	Name: New_configuration (1) Launches © @ommon Nos II Hardware::nios2_1 debug Mios II Hardware::nios2_2 debug	Up Down Edit Add Remove
Filter matched 10 of 10 items	Apply	Revert
?	Deb	ug Close

4. コードのパフォーマンス検証

🛆 ALTIMA

この章では、プログラムのパフォーマンスを測定する方法を紹介します。パフォーマンス・カウンタを使用する方法 とデバッガのプロファイリング機能を使用する方法の2つを主に紹介します。

4-1. パフォーマンス・カウンタ

4-1-1. パフォーマンス・カウンタの概要

Qsys の Performance Counter Unit を使用してコード内の指定したセクションで費やされた時間をシステム・クロックの精度で測定することができます。

パフォーマンス・カウンタは内部に2つのカウンタを持ちます。

- ◆ Time Counter : 64 ビットカウンタ、指定したコード・セクションの実行時間をシステム・クロックの精度で測定
- ◆ Event Counter : 32 ビットカウンタ、指定したコード・セクションの実行回数を測定

パフォーマンス・カウンタは、測定開始から終了までのすべての実行時間、および実行回数を測定するカウンタ(グ ローバル・カウンタ)のほかに、各セクションを測定するカウンタ(セクション・カウンタ)を複数持ち、異なるコード・セク ションの実行時間、回数を同時に測定することが可能です。

測定するセクション数を指定することによって、それぞれのセクション毎に Time Counter と Event Counter が生成 され測定に使用されます。

👃 Performance Cou	nter Unit – performance_counter_0					
Megecore Performance altera_avalon	ce Counter Unit _performance_counter Documentation					
Block Diagram Configuration						
	Number of simultaneously-measured sections: 3 」 測定するセクション数を指定					
clk cloc reset res	Description 2 This peripheral and associated software macro w minimally-intrusive, real-time hardware profiling of your software p n 5					
control_slave ava	You can simultaneously measure several <i>sectil</i> 6 f your program. Each measured section uses both a 64-bit tin <mark>7</mark> unter and a 32-bit occurrence counter. The time counter measures the total time spe <u>ht in</u> a section of code with single-clock resolution. The occurrence-counter measures how many times a section of code is entered.					
	Macros declared in this peripheral's header file make it easy to start and stop counters when entering and exiting sections of C-code. C library routines allow you to retrieve and analyze the results.					
	See the datasheet in the Documentation links for more information.					

🔥 ALTIMA

4-1-2. パフォーマンス・カウンタの使用例

パフォーマンス・カウンタを制御するためのマクロが altera_avalon_performance_counter.h ファイル内で定義されて おり、使用することができます。パフォーマンス測定には、下記のマクロをコード中の測定したいセクションの前後に 挿入します。

使用方法の概要は以下のとおりです。

- ◆ 全てのカウンタを停止し、0にリセットします(PERF_RESET())
- ◆ グローバル・カウンタを開始し、セクション・カウンタをイネーブルにします (PERF_START_MEASURING())
- ◆ コード・セクションの測定を開始します (PERF_BEGIN()、PERF_END())
- ◆ グローバル・カウンタを停止し、セクション・カウンタをディセーブルにします (PERF_STOP_MEASURING())

パフォーマンス・カウンタ使用例



パフォーマンス・カウンタ マクロ

マクロ	概要
PERF_RESET(hw_base_address)	すべてのカウンタを停止し無効にします。リセット時 には0に設定されます
PERF_START_MEASURING(hw_base_address)	グローバル・カウンタをスタートさせ、セクション・カウ ンタを有効にします
PERF_STOP_MEASURING(hw_base_address)	グローバル・カウンタを停止し、セクション・カウンタ を無効にします
PERF_BEGIN(hw_base_address, which_section)	コード・セクションの開始位置
PERF_END(hw_base_address, which_section)	コード・セクションの終了位置

※ hw_base_address : パフォーマンス・カウンタ・ユニットのベース・アドレス

which_section : カウンタ・セクション番号

さらに、パフォーマンス・カウンタを使用した測定結果をコンソールに出力するための関数やカウンタの値を帰す関数が用意されていますので、下記のように使用することができます。perf_print_formatted_report 関数はコンソールにカウンタの測定結果を表示します。



Performance Counter Report							
Total Time: 11.0817 seconds (941946244 clock-cycles)							
+	++	+		++			
Section	8	Time (sec)	Time (clocks)	Occurrences			
+	++	+	+	++			
test section 1	85.5	9.47664	805514087	1			
+	++	+	+	++			
test section 2	14.5	1.60508	136432127	1			
+	++	+		++			

HAL	概要
perf_print_formatted_report()	フォーマット化されたプロファイル結果を stdout に出力します
perf_get_total_time()	グローバル・カウンタの合計値をクロック・サイクル数で返します
perf_get_section_time()	指定したセクション・カウンタの合計値をクロック・サイクル数で返します
perf get num starts()	指定したセクションの実行回数を返します
alt_get_cpu_freq()	CPU の動作周波数を Hz で返します

4-2. プロファイラ

4-2-1. プロファイリング・データの生成方法

Nios II SBT では GNU プロファイラを使用することができます。以下の設定をすることで、GNU プロファイラは、 実行時にホスト PC の gmon.out ファイルにプロファイル結果を出力します。

使用方法の概要は以下のとおりです。

- ◆ アプリケーション / BSP プロジェクトの Properties ⇒ Nios II Application / BSP Properties の設定
 - ▶ デバッグ・レベルを "On" に設定
- ◆ BSP Editor の Main タブ ⇒ Common で、sys_clk_timer を設定
- BSP Editor \mathcal{O} Main $\forall \mathcal{I} \Rightarrow$ Common \mathcal{C} , enable_gprof $arepsilon \mathcal{I} = \mathcal{I} + \mathcal{I}$
- ◆ BSP Editor の Main タブ ⇒ Common で、enable_reduced_device_drivers にチェックが入っていないこと
- ◆ プロジェクトを Clean 後、再度プロジェクトをビルドし、プログラムを実行
- ◆ 出力された gmon.out ファイルでプロファイル結果を確認
 - Nios II SBT

gmon.out はアプリケーション・プロジェクトに生成

生成された gmon.out をダブルクリックで開き内容を確認

> コマンド・シェル

gmon.out をコマンド・ラインから生成

例) nios2-download -g --write-gmon gmon.out xxxxxxx.elf

nios2-elf-gprof コマンドを使用して測定結果を確認します

- 例) gmon.out ファイルをテキスト・ファイルに変換して確認する nios2-elf-gprof profiler_gnu.elf gmon.out > report.txt
- 例) すべてのソースにプロファイラからの追加情報が挿入されたファイルを出力 nios2-elf-gprof profiler_gnu.elf gmon.out > report.txt

4-2-2. プロファイリング・データの解析

gmon.out ファイルを開きプロファイル結果を確認します。gmon.out ファイルにはフラット・プロファイル、コール・グラフの項目があり、下記の内容が記述されています。

◆ フラット・プロファイル

各関数の実行時間、合計のコールされた回数

◆ コール・グラフ

関数毎に、コールされた親関数とコールされた回数、コールしている子関数とコールした回数

また、Nios II SBT の Call Hierarchy ビューや Task ビューを利用して各関数の実行時間等のプロファイル結果を 確認することができます。 <u>フラット・プロファイル 出力例</u>

Flat profile:						
Each sam	ople counts	as 0.01	seconds.			
* C1	umulative	self		self	total	
time	seconds	seconds	calls	s/call	s/call	name
97.93	11.08	11.08	1	11.08	11.10	main
1.09	11.20	0.12				alt_dcache_flush
0.85	11.30	0.10	9	0.01	0.01	alt_busy_sleep
0.09	11.31	0.01	3	0.00	0.00	_malloc_r
0.00	11.31	0.00	1148	0.00	0.00	alt_irq_handler
0.00	11.31	0.00	1135	0.00	0.00	alt_avalon_timer_sc_irq
0.00	11.31	0.00	1135	0.00	0.00	alt_tick
0.00	11.31	0.00	325	0.00	0.00	unpack_d
0.00	11.31	0.00	258	0.00	0.00	udivmodsi4
0.00	11.31	0.00	171	0.00	0.00	pack_d
0.00	11.31	0.00	160	0.00	0.00	muldi3
0 00	44 04	0 00	104	0 00	0 00	

<u>コール・グラフ 出力例</u>

Call graph (explanation follows)					
granul	larity:	each sam	ple hit cov	ers 32 byte	e(s) for 0.09% of 11.31 seconds
index	% time	self	children	called	name
		0.00	11.18	1/1	_start [2]
[1]	98.9	0.00	11.18	1	alt_main [1]
		11.08	0.02	1/1	main [3]
		0.00	0.09	1/1	alt_sys_init [6]
		0.00	0.00	1/1	alt_io_redirect [54]
		0.00	0.00	1/1	alt_irq_init [55]
		0.00	0.00	1/1	close [66]
		0.00	0.00	1/1	_do_ctors [190]
		0.00	0.00	1/1	atexit [65]
		0.00	0.00	1/1	exit [68]
		0.00	0.00	1/4	alt_release_fd [39]
					<pre><spontaneous></spontaneous></pre>
[2]	98.9	0.00	11.18		start [2]
		0.00	11.18	1/1	alt_main [1]
		 11.08	 0.02	1/1	alt main [1]
[3]	98.1	11.08	0.02	1	main [3]
[0]	20.1	0.01	0.00	1/9	alt_busy_sleep [5]

4-3. パフォーマンス・カウンタ vs プロファイラ

パフォーマンス・カウンタは、指定した特定コードの測定区間の実行クロック・サイクル数を正確に測定します。

プロファイラではプログラム・カウンタをサンプリングし、それぞれの関数にどれだけ実行時間がかかっているかー 定期間内のヒット数を表示します。プロファイラを使用しますと、プロファイラの分のプログラムがオーバー・ヘッドとな り、システム全体の実行時間が遅くなりますので注意が必要です。

<u> 改版履歴</u>

Revision	年月	概要
1	2015 年 5 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。
 株式会社アルティマ ホームページ: http://www.altima.co.jp
 技術情報サイト EDISON: http://www.altima.jp/members/index.cfm
 株式会社エルセナ ホームページ: http://www.elsena.co.jp
 技術情報サイト ETS : http://www.elsena.co.jp/elspear/members/index.cfm
- 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。