Programowanie procesora Microblaze w środowisku SDK

Zespół Rekonfigurowalnych Systemów Obliczeniowych AGH Kraków http://www.fpga.agh.edu.pl/

9 kwietnia 2010

# 1.Wstęp

Celem niniejszego ćwiczenia jest:

- zapoznanie z narzędziami umomożliwościami debuggowania kodu uruchomionego na platformie MicroBlaze
- zapoznanie ze środowiskiem graficznym wspomagającym prace programisty: Xilinx Platform SDK opartym na narzędziu Eclipse.

Wymagania:

- Xilinx ISE Design Suite 11.2.
- Płytka uruchomieniowa Spartan3E-200 Digilent Board.

## 2.Zapoznanie z platformą sprzętowa

Do wykonania poniższego ćwiczenia należy posłużyć się platformą sprzętową Microblaze wykonaną w poprzednich ćwiczenia:

"EDK MicroBlaze Tutorial Documentation z wykorzystaniem platformy FPGA z rodziny SPARTAN3E"

#### a) Komponenty sprzętowe

W skład platformy uruchomieniowej powinny wchodzić następujące komponenty

PLL	🛃 Bus Interfaces	Ports Address	ses
	Name	Bus Name	ІР Туре
	<u>■ microblaze_0</u>		📩 microblaze
	dimb		☆ lmb_v10
	imb		📩 lmb_v10
	mb_plb		🌟 plb_v46
	😟 dlmb_cntlr		🚖 lmb_bram_if
6-0-0-	😟 ilmb_cntir		👷 lmb_bram_if
	😟 Imb_bram		👷 bram_block
<u></u>	😟 mdm_0		👷 mdm
	😟 DIP_Switche		👷 xps_gpio
<u> </u>	🗄 LED_75EGME		👷 xps_gpio
<b>_</b>	🕒 LEDs_88it		👷 xps_gpio
<b></b>	🕒 Push_Button		👷 xps_gpio
<b></b>	🗈 R5232		☆ xps_uartlite
	dock_genera		👷 clock_gener
	proc_sys_res		🐈 proc_sys_re
	Deserve als 1 Maderk		

Rysunek 1 Moduły sprzętowe

- microblaze\_0 procesor MicroBlaze
- **mb\_plb** magistrala i arbiter OPB
- mdm0 moduł interfejsu umożliwiający sprzętowe debuggowanie poprzez JTAG
- **RS232** interfejs UART
- dcm\_0 moduł generatrora zegara
- dlmb\_cntlr interfejs danych BRAM
- ilmb\_cntlr interfejs instrukcji BRAM
- lmb\_bram pamięć Block RAM
- peryferia GPIO

#### b) Eksport ustawień do środowiska SDK

Aby możliwe było korzystanie ze środowiska programistycznego GUI należy dokonać eksportu parametrów i ustawień utworzonej w Xilinx Platform Studio platformy sprzętowej. Ustawienia te są eksportowane w formacie XML

W celu wygenerowania pliku z ustawieniami należy w środowisku XPS w którym otwarty jest nasz projekt hardwarowy wykonać:

**Projekt**  $\rightarrow$  **Export Hardware Design To SDK** 

Pojawi się okno

Export to SDK / Launch SDK			
This dialog allows you to export hardware platform information to be used in Xilinx Software Development Kit			
Include bitstream and BMM file			
(XPS will regenerate bitstream if necessary, and it may take some time to finish.)			
Directory location for hardware description files			
SDK\SDK_Export			
Export Only Export & Launch SDK Cancel Help			

Wybrać polecenie "Export Only"

# 4. Środowisko Eclipse

Eclipse jest rozbudowanym środowiskiem programistycznym napisanym w Javie. Projekt został stworzony przez firmę IBM a następnie udostępniony społeczności Open Source. W naszym przypadku jest ono dostosowane przez firmę Xilinx m.in. w celu łatwiejszego tworzenia i debuggowania oprogramowania na platformę MicroBlaze.

#### a) Tworzenie projektu

Otworzyć z menu START Windows aplikacje Xilinx Platform SDK.

Na początku należy wskazać miejsce na dysku gdzie będzie zapisywany projekt. Można wybrać dowolne miejsce, ale w podanej ścieżce **nie może być spacji.** 

Workspace Launcher	
Select a workspace Xilinx Software Development Kit stores your projects in a fe Choose a workspace folder to use for this session.	older called a workspace.
Workspace: D:\Project\SDK_Workspace	<u>B</u> rowse
$\square$ Use this as the default and do not ask again	
	OK Cancel

Również podczas otwierania aplikacja zapyta o plik XML z ustawieniami platformy sprzętowej. Należy podąć ścieżkę do wygenerowanego wcześniej pliku XML:

New Hardware Specification File	×
Hardware Specification File	ï
8 File D:\Project\SDK\SDK_Export\hw\system.xml does not exist.	
Hardware Specification File: D:\Projecti,SDK\SDK_Export\hw\system.xml	Browse
OK.	Cancel

Wybrana platforma sprzętowa będzie widoczna w oknie nawigacji projektu:



Dla wyeksportowanej z XPS platformy sprzętowej musimy w SDK utworzyć odpowiadającą platformę softwarową:

Wybieramy w SDK **File→ New → Software Platform** 

🐵 New Software Platform Project 🛛 🔀
Create a Software Platform Project Create a Software Platform project
Project name: SW_Platform
Processor: microblaze_0 (microblaze)
Platform Type: standalone
Standalone is a simple, low-level software platform. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.
Project Location:
IV Use <u>d</u> efault
Directory; D:\_Work\EDK\spartan1600\SW_Platform
<u> </u>

Należy wybrać nazwę platformy softwarowej i typ platformy "*standalone*". Wciskamy "Finish"

Powstaje platforma sprzętowa ze wszystkimi bibliotekami i driverami potrzebnymi do programowania platformy.



Listę dołączonych bibliotek i sterowników można zobaczyć wybierając "Software Platform Settings" w menu kontekstowym wpisu z nazwą platformy (prawy klawisz myszy na nazwie platformy).

🐵 Software Platform	Settings - SW_Platform {	microblaze_0 }				
Software Platform 9	Settings					n-la
Customize the libraries ar	nd drivers for the Software Platfo	erm.				0
Software Platform	Processor Settings					
Drivers	CRU Data and Land					
	CPU Driver: jcpu	<u> </u>		CPO Driver version: 1	1.12.a	
	Processor Parameters:					
	Name	Current Value	Default Value	Туре	Description	^
	🖃 microblaze_0					
	compiler	mb-gcc	mb-gcc	string	Compiler used to compile both BSP/Libraries a	and Applicat
	archiver	mb-ar	mb-ar	string	Archiver used to archive libraries for both BS	P generatio
	extra_compiler_flags	-g	-g	string	Extra compiler flags used in BSP and library g	eneration.
	xmdstub_peripheral	none	none	peripheral_instance	Debug peripheral to be used with xmdstub	IN CONTRACTOR INC.
	<					>
	<ul> <li>OS Version and Libraries Select</li> </ul>	ion				
	OS: standalone		Version:	2.00.a	•	
	Standalone is a simple, low-le the basic features of a hoste	vel software platforr d environment, such	n, It provides acce as standard input	ess to basic processor feature and output, profiling, abort	es such as caches, interrupts and exceptions a and exit.	is well as
	Use Library Version D	escription				
	□ xilmfs 1.00.a • X □ xilisf 1.00.a • X	ilinx Memory File Syst ilinx In-system and Se	em erial Flash Library			
	xilflash 1.01.a X	ilinx Flash library for I	Intel/AMD CFI con	npliant parallel flash	a such as . Describes Such as ACE laws a law of drive	
	□ xinatis 1.00.a ♥ P □  wip 1.00.b ♥  v	vIP TCP/IP Stack libra	ry: IwIP v1.3.0, X	ilinx adapter v1.00.b	system, Requires systemace lower level unive	515.
	Download Partner Library Defin	ition Files				
						-
					OK	Cancel

#### b) Pisanie i kompilacja kodu

W następnym kroku tworzymy projekt aplikacji: File→ New → Managed Make C Project

🐵 New Managed Make C Application	on Project 🛛 🛛 🛛
Managed Make C Project Create a new Managed Make C project.	G
Project Name:       SW_Project          Software Platform:       SW_Platform         Project Location       Image: Construction for Project (reconstruction)         Image: Construction for Project (reconstruction)       D:/_Work/EDK/spartan1600/S         Sample Applications       Sample Application         Software Platform       Second	ommended) W_Project Browse  Description An application with no contents.
< <u>B</u> ack <u>N</u> e	ext > Einish Cancel

Wybieramy nazwę naszego projektu i klikamy Finish.

#### c) Programowanie układu FPGA

Aby możliwe było uruchamianie kodu na platformie FPGA musi ona zostać wstępnie zaprogramowana konfiguracją sprzętową zdolna do komunikacji z PC poprzez JTAG Wykonujemy komendę:

File→ Program FPGA

🐵 Progra	ım FPGA					
Program	FPGA					
🔇 File D:/	Project/SDk	(/SDK_Export/hv	v/system_bd.bmm	does n	ot exist	<b>6</b> -0
⊢Bit and Br	nm Files					 
Bit File:	D:/Project	:/system.bit				Browse
Bmm File:	D:/Project	:/SDK/SDK_Expo	rt/hw/system_bd.	bmm		Browse
Specify the	ELF file to b	e initialized to e	ach processor's BR	AM me	mory:	
Processor		Туре	Initialization	ELF		
microblaze	_0	microblaze	BootLoop	-		
1						
		Save an	nd Program	Sav	e	Cancel

Automatycznie wybrany jest plik konfiguracyjny naszej platformy hardwarowej i plik konfiguracji pamięci dla niej. W pliku .**bmm** zawarte są wszystkie informacje potrzebne do prawidłowego umieszczenia skompilowanego kodu w pamięciach Block RAM FPGA. We wgranej konfiguracji procesor wykonywać będzie program "branch-to-itself": bootloop. Można ten program zmienić wybierając inny skompilowany kod czyli inny plik **.elf**. Przy pierwszym uruchomieniu nie zmieniamy nic. Naciskamy "Save and Program"

# d) Pisanie i kompilacja kodu

Do utworzonego projektu dodajemy nowy plik źródłowy **File→ New → Source File** 

🐵 New Sourc	e File		×
Create a new s	ource file.		C
Source <u>F</u> older: Source File:	SW_Platform first.cpp		Browse
		<u>E</u> inish	Cancel

Plik pojawi się w oknie projektu



Teraz możemy przystąpić do pisania naszej aplikacji. W oknie edytora wpisujemy poniższy kod i zapisujemy projekt.

```
#include "xgpio.h"
 1
    #include "xparameters.h"
 2
 3
 4
    main()
 5
    {
        Xuint8 a,b;
 6
 7
 8
        while(1)
 9
        {
           a=-1;b=-1;
10
11
           while(a>9){
12
              xil printf("\nPodaj a=");
13
              a=XUartLite RecvByte(XPAR RS232 BASEADDR);
14
15
              a-=0x30;
           }
16
17
           while(b>9) {
18
              xil printf("\nPodaj b=");
19
              b=XUartLite RecvByte(XPAR RS232 BASEADDR);
20
              b-=0x30;
21
           }
22
23
24
25
           xil printf("\nSuma=%d", a+b);
        }
26
27
28
        return 0;
29
    }
```

Projekt zostanie przekompilowany a w oknie konsoli pojawi się komunikat: "Build complete for project SW\_Project"

#### e) Uruchomienie debuggera

Z menu kontekstowego projektu wybieramy DebugAs→ Debug on Hardware

C(C++ Projects × D:/_Work/EDK/spa @ mirroblace_0 ( @ \$ SW_Platfo @ Archiv @ mirroblace_1 @ Makefi @ blogen.	rtan1600/5DK/5DK_Export/hw microblaze) rm as laze_0 le log mk	<pre>first.cpp 1 int 2 { 3 4 4 </pre>	n X main	() rn O;		
🕀 😥 SW_Projec	New		F)			
	Go Into					
	Open in New Window					
	Build Project Rebuild Project					
	Copy Raste Rename Relete					
	import					
	Refresh Close Project		Co.	insole 🛛	Properties	
	Generate Linker Script Board Support Package S Software Platform Settin View Design Report Run As	Settings	nc DH 1s	iir sec bheader K/spart aze_0 S ssed.	-hw an1600/ W_appli	uress SDK/S catio
<	Debug As		2	1 Debug	on Hardwar	e
/SW_Project	Team Compare With Restore from Local Histo	ī¥	3	2 Debug	on Simulato	ć
	Properties		,		A K	411

## f) Wgrywanie i debuggowanie programu

Przed uruchomieniem programu płytkę Spartan3E200 należy połączyć z terminalem np.: Hyperterminal Windows.

Aplikacja automatycznie załaduje nasz program do pamięci (plik \*.elf) zaczynając od lokacji 0x00000000. Następnie przejdzie do okna debuggowania.

Debug - first.cpp - Xilinx Software Development Kit		
Elle Edit Refactor Navigate Search Project Tools Hardware Design	n <u>R</u> un <u>Window</u> <u>Help</u>	_
] 📬 • 🖾 🗁 🗋 🛛 📓 ] 🛍 🗉 🔂 ] 🎎 💆 ] 🐓 🛝 🤗	‰ 🏙 ] 🖸 🔯 ] 🏇 • Ø • Q • ] 🥭 🖋 ] 🖉 + 🖏 - 🏷 🗢 -	
Debug C/C++ Resource		
🍄 Debug ⊠ 👘 🗆	Variables X Breakpoints XMD Console Modules Registers 🖓 🗆 Welcome 🚺 Memory 🛛	
🕩 🗉 📕 🖉 🍇 🚴 👁 .e 🗮 💓 🖬 🗸	2.46日 \$* \$ % ▼ 13 14 18 1	
Cereminated >>W proj_HW [Dirk C(C++ E)F]           off         cterminated >>N0 target debug apent (0:09.03.10 15:20)           cterminated, ext value: 0:0ebugger Process (0:09.03.10 15: cterminated, ext value: 0:0ebugger Process (0:09.03.10 15: cterminated, ext value: 0:00_W0(200;E) startini.00(5W)           SW_application_HW [Oline: C(C++ E)F]           W_application_HW [Oline: C(C++ E)F]           W Thread (0) (Suspended)           Imain() & w. HWR: copis 0.00000H           Debugger Process (0:09.3.10 15:30)	King and Angel and An	×
R first.cop 🐰 🗖 belo.cop		
1 int main() 2(	• • man	
Console 🕅 Tasks		
SW_application_HW [Dilro: C/C++ ELF] D:\_Work[EDK]spartan1600[SW_Proje	lect/Debug/SM_application.elf (09:03:10 15:36)	

Debugger środowiska SDK umożliwia wszystkie czynności, jakie dostępne są w innych środowiskach programistycznych np. Visual C++. Możliwa jest praca krokowa., ustawianie pułapek ( **breakpoint** ), podgląd zmiennych, pamięci rejestrów procesora i wiele innych.

# Nie należy zapominać że program wykonuje się w rzeczywistym procesorze na naszej platformie sprzętowej, a środowisko SDK komunikuje się z procesorem przy interfejsu JTAG

Program należy przekrokować obserwując zachowanie płytki.

# 5. Zaliczenie

Napisać i uruchomić program spełniający następujące funkcje:

- Wyświetlający na diodach led wartość wpisana z terminala
- Wypisujący na terminalu wartość ustawioną na przełącznikach DIP
- Reagujący zdefiniowanym komunikatem po wciśnięciu przycisku "push button" np.: "Wciśnięto przycisk A".

Wszystkie wymienione funkcje mogą być realizowane równocześnie – w jednej pętli programu.