

# Lista 4: Writing Basic Software Applications

Przemysław Dragańczuk, Marcin Serafin, Aksel Stankiewicz

33INF–SSI/A

# 1 Wstęp

## 1.1 Cel ćwiczenia

Celem ćwiczenia jest pokazanie, jak można napisać prostą aplikację softwareową dla płytki Spartan-3E. Aplikacja ta ma za zadanie włączanie odpowiednich diod LED w zależności od stanu przycisków i przełączników. Sekcja .text oprogramowania będzie przechowywana w pamięci BRAM.

## 2 Przebieg ćwiczenia

### 2.1 Modyfikacja projektu z laboratorium 3

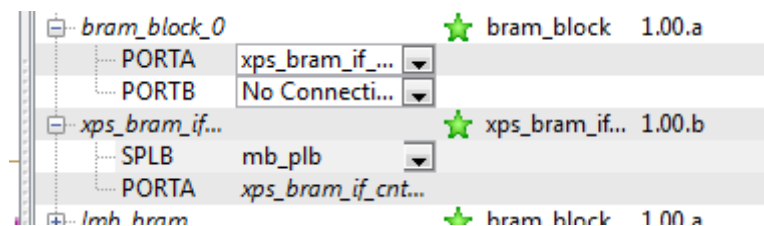
Skopiowano folder lab3 do nowego folderu lab4, a następnie otwarto go w programie XPS.

### 2.2 Dodanie BRAM i kontroler BRAM

Z katalogu IP dodano następujące IP do projektu systemu:

- XPS BRAM Controller 1.00.b
- Block RAM (BRAM) block 1.00.a

Następnie połączono kontroler BRAM do PLB i BRAM do kontrolera BRAM.



Rysunek 1: Dodane i połączone IP BRAM i BRAM controller

W zakładce "Adresses" ustawiono rozmiar pamięci BRAM na 8K i kliknięto na przycisk "Generate Adresses"

W menu "Hardware" kliknięto na "Generate Bitstream". Później w menu "Project" kliknięto "Export Hardware Design to SDK" a następnie "Export & Launch SDK".

## 2.3 Tworzenie aplikacji

Po otwarciu SDK w odpowiednim folderze, utworzono nowy projekt typu "Xilinx C Project". Wybrano szablon "Empty Application" a jako nazwę podano "TestApp". Następnie wybrano odpowiedni BSP. Zaimportowano plik lab4.c.

```
#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h"

//=====

int main (void)
{
    XGpio dip, push, led;
    int i, psb_check, dip_check;
    // define instance pointer for LEDs_8Bit device

    xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID);
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_PUSH_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xffffffff);
    // initialize and set data direction for LEDs_8Bit device
    XGpio_Initialize(&led, XPAR_LEDS_8BIT_DEVICE_ID);
    XGpio_SetDataDirection(&led, 1, 0xffffffff);

    while (1)
    {
        psb_check = XGpio_DiscreteRead(&push, 1);
        xil_printf("Push Buttons Status %x\r\n", psb_check);
        dip_check = XGpio_DiscreteRead(&dip, 1);
        xil_printf("DIP Switch Status %x\r\n", dip_check);

        // output dip switches value on LEDs_8Bit device
        XGpio_DiscreteWrite(&led, 1, psb_check);

        for (i=0; i<999999; i++);
    }
}
```

Rysunek 2: Zaimporowany program

Po przeczytaniu dokumentacji zmodyfikowano kod tak, aby na diodach LED wyświetlany był stan przycisków i przełączników.

```
1 #include "xparameters.h"
2 #include "xgpio.h"
3 #include "xutil.h"
4
5 int main (void)
6 {
7     XGpio dip, push, led;
8     int i, psb_check, dip_check;
```

```

9
10 xil_printf("-- Start of the Program --\r\n");
11
12 XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID);
13 XGpio_SetDataDirection(&dip, 1, 0xffffffff);
14
15 XGpio_Initialize(&push, XPAR_PUSH_DEVICE_ID);
16 XGpio_SetDataDirection(&push, 1, 0xffffffff);
17
18 XGpio_Initialize(&led, XPAR_LEDS_8BIT_DEVICE_ID);
19 XGpio_SetDataDirection(&led, 1, 0xffffffff);
20
21 while (1)
22 {
23     psb_check = XGpio_DiscreteRead(&push, 1);
24     xil_printf("Push Buttons Status %x\r\n", psb_check);
25     dip_check = XGpio_DiscreteRead(&dip, 1);
26     xil_printf("DIP Switch Status %x\r\n", dip_check);
27
28     int status = (psb_check << 4) | dip_check;
29
30     XGpio_DiscreteWrite(&led, 1, status);
31
32     for (i=0; i<999999; i++);
33 }
34 }

```

## 2.4 Analiza zbudowanych obiektów

W menu "Xilinx Tools" kliknięto na "Generate Linker Script". Następnie zmieniono rozmiar stosu i stogu na 400 bajtów i kliknięto "Generate".

Zrekompileowano plik lab4.c, a następnie w menu "Xilinx Tools" wybrano "Launch shell". Komenda `cd` została użyta aby przejść do folderu `TestApp/Debug`.

Użyto komendy `mb-objdump -h TestApp.elf` do wyświetlenia poszczególnych sekcji programu.

```

C:\Windows\system32\cmd.exe
D:\33inf-ssiMA\lab4\SDK\SDK_Export\TestApp\Debug>nb-objdump -h TestApp.elf
TestApp.elf:      file format elf32-microblaze

Sections:
Idx Name          Size      UMA      LMA      File off  Algn
 0 .vectors.reset 00000004 00000000 00000000 000000f4 2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 1 .vectors.sw_exception 00000004 00000008 00000008 000000f8 2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 2 .vectors.interrupt 00000004 00000010 00000010 000000fc 2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 3 .vectors.hw_exception 00000004 00000020 00000020 00000100 2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 4 .text           00001160 00000050 00000050 00000104 2**2
   CONTENTS, ALLOC, LOAD, CODE
 5 .init           00000034 000011b0 000011b0 00001264 2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 6 .fini           0000001c 000011e4 000011e4 00001278 2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 7 .ctors          00000008 00001200 00001200 000012b4 2**2
   CONTENTS, ALLOC, LOAD, DATA
 8 .dtors          00000008 00001208 00001208 000012bc 2**2
   CONTENTS, ALLOC, LOAD, DATA
 9 .rodata         0000045a 00001210 00001210 000012c4 2**2
   CONTENTS, ALLOC, LOAD, READONLY, DATA
10 .sdata2         00000006 0000166a 0000166a 0000171e 2**0
   ALLOC
11 .sbss2          00000000 00001670 00001670 00001870 2**0
   CONTENTS
12 .data           00000148 00001670 00001670 00001720 2**2
   CONTENTS, ALLOC, LOAD, DATA
13 .eh_frame       00000004 000017b8 000017b8 00001868 2**2
   CONTENTS, ALLOC, LOAD, DATA
14 .jcr            00000004 000017bc 000017bc 0000186c 2**2
   CONTENTS, ALLOC, LOAD, DATA
15 .sdata          00000000 000017c0 000017c0 00001870 2**0
   CONTENTS
16 .sbss           00000000 000017c0 000017c0 00001870 2**0
   CONTENTS
17 .tdata          00000000 000017c0 000017c0 00001870 2**0
   CONTENTS
18 .tbss           00000000 000017c0 000017c0 00001870 2**0
   CONTENTS
19 .bss            0000002c 000017c0 000017c0 00001870 2**2
   ALLOC
20 .heap           00000194 000017ec 000017ec 00001870 2**0
   ALLOC
21 .stack          00000190 00001980 00001980 00001870 2**0
   ALLOC
22 .debug_line     000012e9 00000000 00000000 00001870 2**0
   CONTENTS, READONLY, DEBUGGING
23 .debug_info     00001617 00000000 00000000 00002b59 2**0
   CONTENTS, READONLY, DEBUGGING
24 .debug_abbrev   0000087a 00000000 00000000 00004170 2**0
   CONTENTS, READONLY, DEBUGGING
25 .debug_aranges  00000240 00000000 00000000 000049f0 2**3
   CONTENTS, READONLY, DEBUGGING
26 .debug_macinfo  00000686 00000000 00000000 00004c30 2**0
   CONTENTS, READONLY, DEBUGGING
27 .debug_frame    00000294 00000000 00000000 0000d2b8 2**2
   CONTENTS, READONLY, DEBUGGING
28 .debug_loc      0000089f 00000000 00000000 0000d54c 2**0
   CONTENTS, READONLY, DEBUGGING
29 .debug_pubnames 000002d4 00000000 00000000 0000ddeh 2**0
   CONTENTS, READONLY, DEBUGGING
30 .debug_ranges   00000018 00000000 00000000 0000e0bf 2**0
   CONTENTS, READONLY, DEBUGGING
31 .debug_str       00000693 00000000 00000000 0000e0d7 2**0
   CONTENTS, READONLY, DEBUGGING

D:\33inf-ssiMA\lab4\SDK\SDK_Export\TestApp\Debug>

```

Ponownie otwarto okno "Generate Linker Script". Rozmiary stosu i stogu zostały zmienione na 1KB. Następnie w liście rozwijanej "Place Code Sections in:" wybrano kontroler BRAM.

Po ponownej rekompilacji pliku lab4.c i uruchomieniu komendy `mb-objdump` zauważono, że sekcja `.text` przesunęła się do pamięci BRAM.

## 2.5 Weryfikacja w sprzęcie

Używając narzędzia "Program FPGA" wgrano aplikację na płytkę i zweryfikowano, że działa

```
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
```

## 3 Wnioski

Oprogramowanie Xilinx Platform Studio pozwala w bardzo prosty sposób projektować oprogramowanie, które można wgrać na płytki FPGA. Pozwala również na zobaczenie wszystkich informacji o zainstalowanym sprzęcie, jak połączenia między urządzeniami czy przypisane adresy pamięci.