



Laboratory 2: Addings Solution




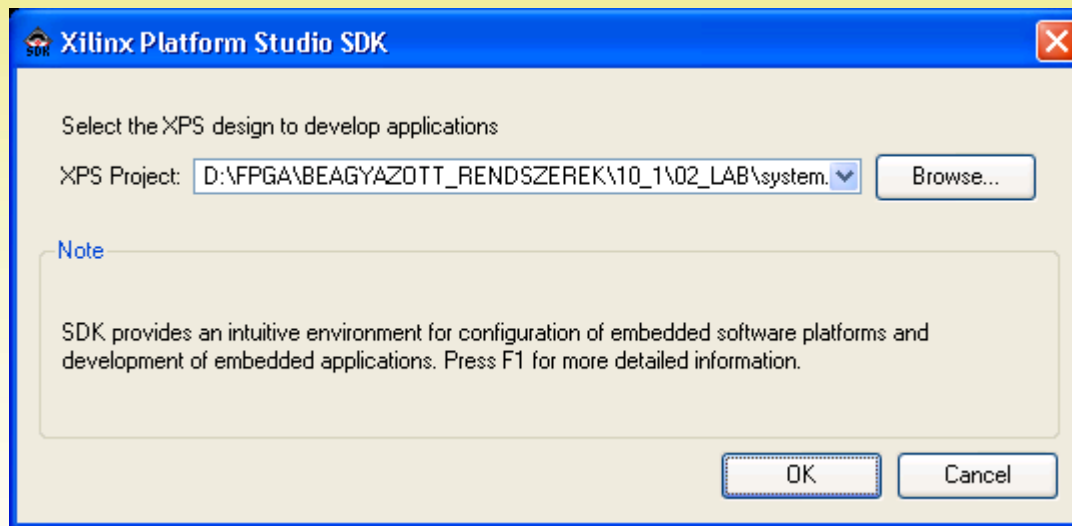
Appendix for using Xilinx EDK/SDK 10.1 SP3

Instructor: Zsolt Vörösházi, PhD.



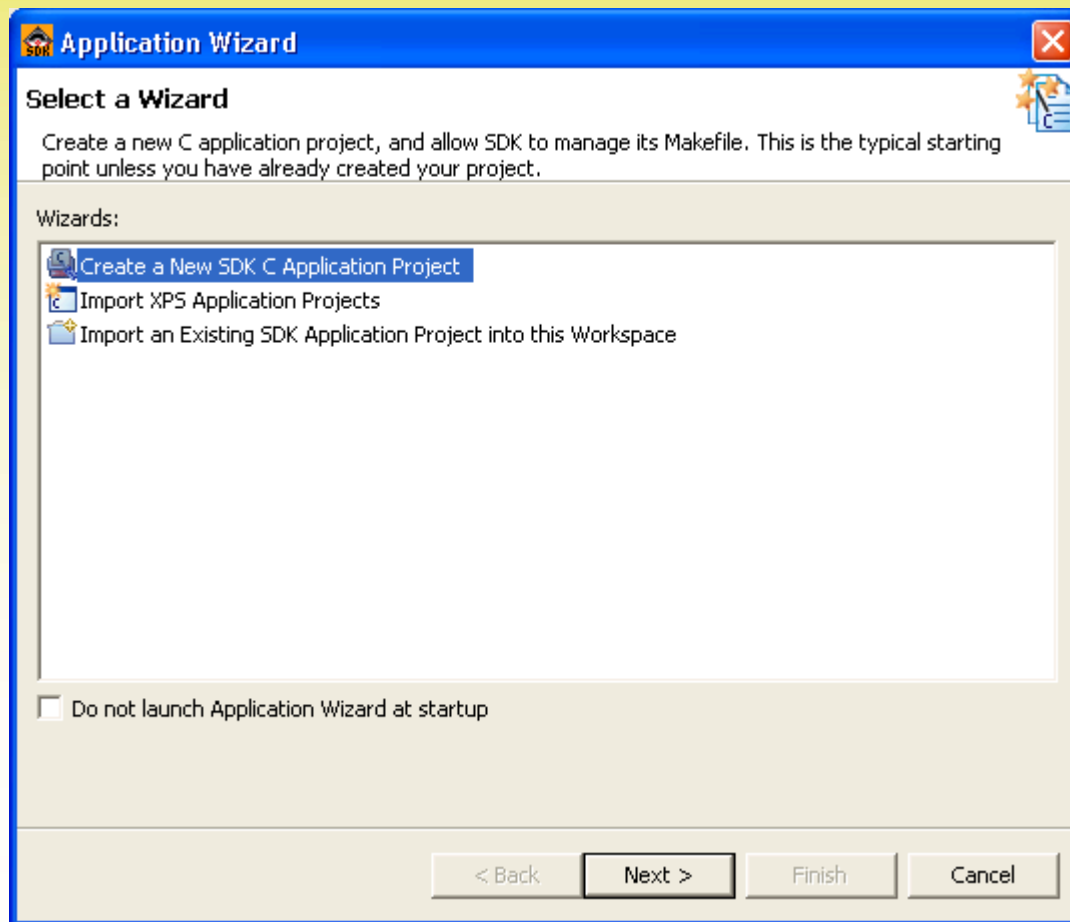
SDK 10.1 – Step 1)

- Run Xilinx Platform Studio SDK 
- Select workplace / XPS Project directory



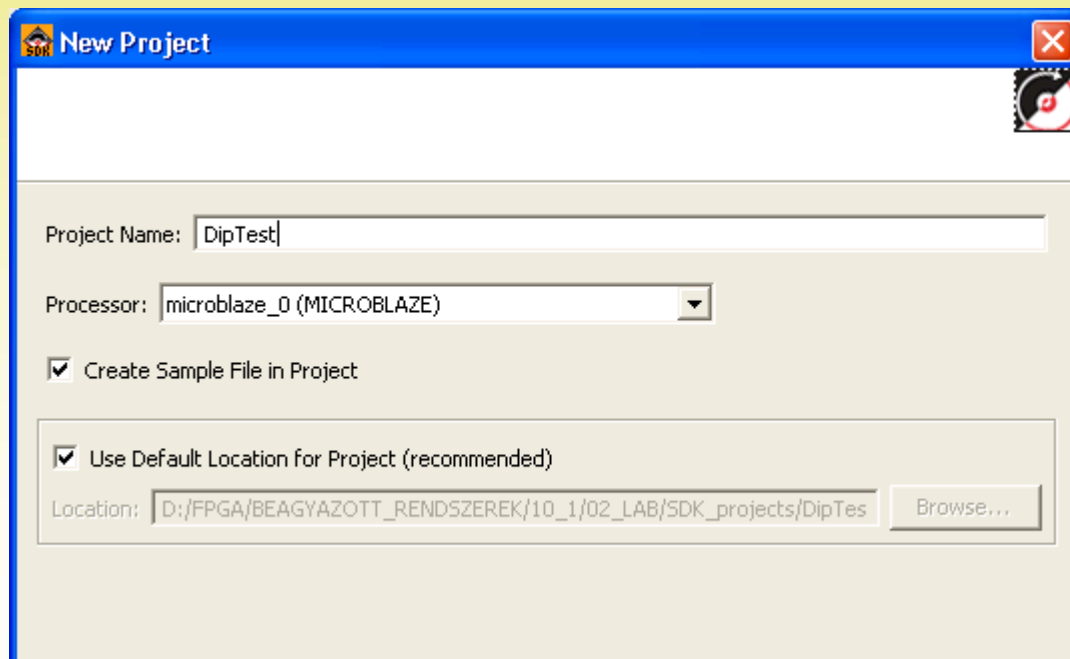
Select Application Wizard

- Create a new SDK Application project



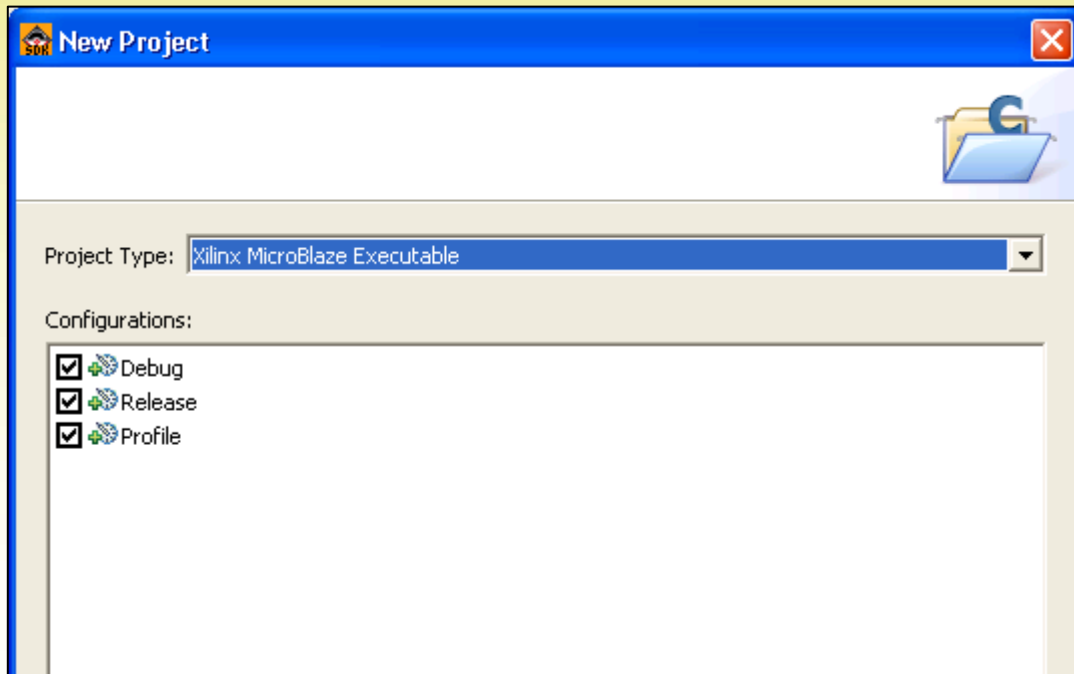
Add project name

- Add project name: „DipTest” and processor instance „MicroBlaze_0”



Add project type

- Project type: Xilinx MicroBlaze Executable
- Configurations: Debug / Release / Profile



DipTest – main.c

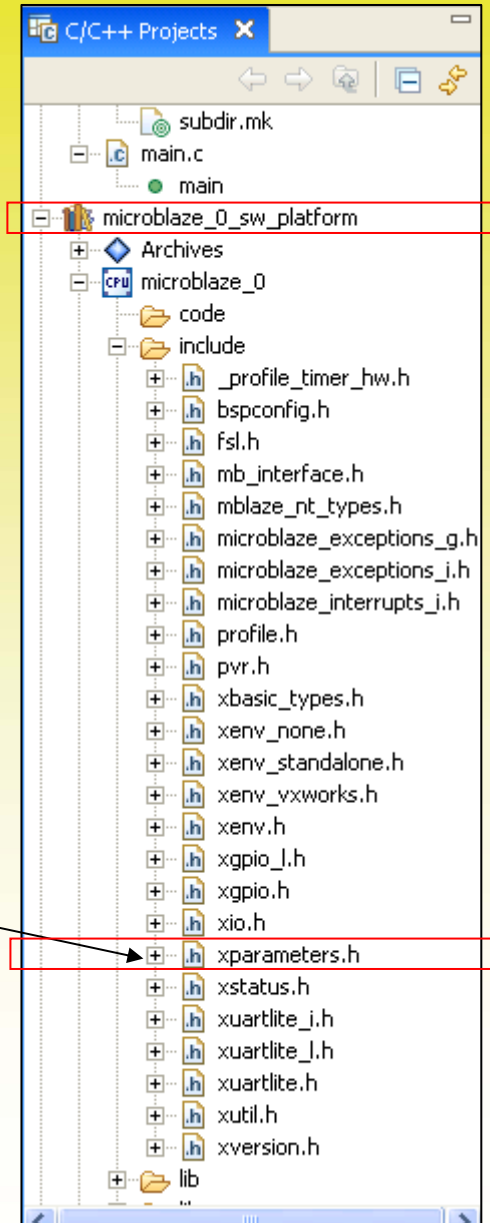
- Simple DipTest main.c application created
- Automatic build is set by default (Project → Build Automatically)
- SW platform created (generated from .MSS file)

SW platform

Microblaze_0_sw_platform

(right click -> generate Libraries and BSP or LIBGen icon)

- Archives: .a (binary)
- Microblaze_0
 - Code
 - Include*
 - See xparameters.h (generated from .MHS)
 - Lib
 - LibSrc

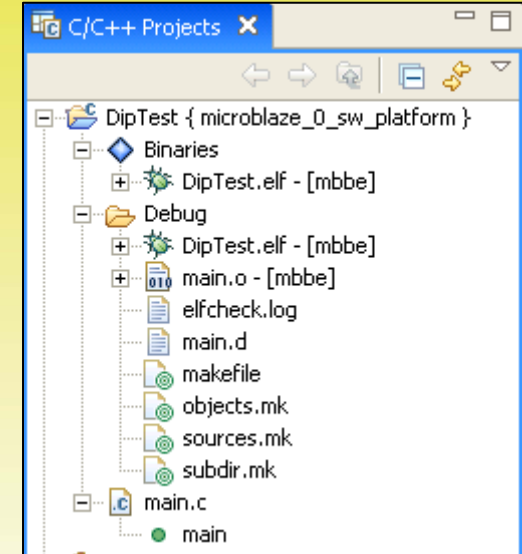


DipTest SW application

DipTest

{microblaze_0_sw_platform}

- Binaries (.elf)
- Debug (.elf)
- main.c
- Additional headers and sources



GPIO drivers and applications

- c:\Xilinx\10.1\EDK\sw\XilinxProcessorIPLib\drivers\
 - **gpio_v2_12_a**: GPIO v2.12 driver functions (low- and high-level driver functions [cpp, h])
 - /Build: OS dependent Makefiles
 - /Data: `gpio_header.h` + `.tcl` + `.mdd` (declares `GPIOInput/OutputExample()` functions)
 - /Doc: API in html form (**see index.html**)
 - /Examples: simple example applications (use drivers)
 - e.g. `xgpio_tapp_example.c`
 - /Src: sources of low-, and higher-level drivers

gpio_header.h

- Declares `GpioInputExample()` function for prototyping
- **XStatus** `GpioInputExample (Xuint16 DeviceId, Xuint32 *DataRead);`
 - `#include "xbasic_types.h"`
 - `#include "xstatus.h"`

xgpio_tapp_example.c

- Declares `GpioInputExample()` function
 - This performs a test on the GPIO driver/ device with the GPIO configured as INPUT
 - Invokes low level drivers

```
XStatus GpioInputExample (Xuint16 DeviceId,  
Xuint32 *DataRead);      /*Function  
Prototype*/
```

```
XGpio GpioInput;      /* The driver instance  
for GPIO Device configured as I/P */
```

xgpio_tapp_example.c (cont.)

Defines GpioInputExample () function as follows:

```
Status GpioInputExample (Xuint16 DeviceId, Xuint32 *DataRead) {
    XStatus Status;

    /*
     * Initialize the GPIO driver so that it's ready to use,
     * specify the device ID that is generated in xparameters.h
     */

    Status = XGpio_Initialize(&GpioInput, DeviceId);
    if (Status != XST_SUCCESS)    {
        return XST_FAILURE;
    }

    /*
     * Set the direction for all signals to be inputs    */
    XGpio_SetDataDirection(&GpioInput, GPIO_CHANNEL,
0xFFFFFFFF);

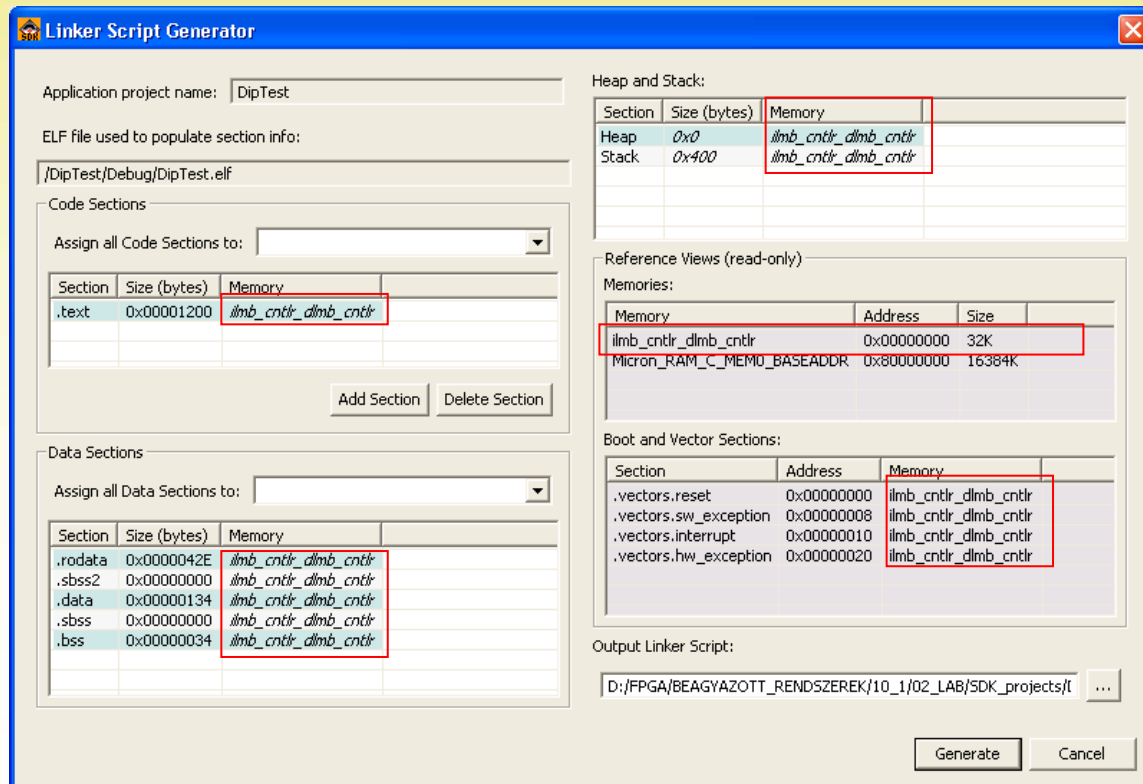
    /*
     * Read the state of the data so that it can be verified    */
    *DataRead = XGpio_DiscreteRead(&GpioInput, GPIO_CHANNEL);
    return XST_SUCCESS;
}
```

Dependencies

- **#define** GPIO_INPUT_DEVICE_ID
XPAR_DIP_DEVICE_ID
 - Comes from xparameters.h (generated from .MHS)
- **#define** DIP_CHANNEL 1
 - Comes from DIP IP core (remark: GUI settings)
- **#define** GPIO_BITWIDTH 8
 - Comes from 8 DIPSwitches are located on Nexys2 board
- Important note: xil_printf() function must be used instead of normal printf() because the consumes less memory
 - **#include** stdio.h

Step 2.) Generate Linker Script

- If necessary, set all sections of the .elf file into the internal BRAM memory
 - Select [ilmb_cntlr_dlmb_cntlr] -> Generate



SDK: Custom program segments (compile sw application)

- **.text** — the **executable** code
- **.rodata** — any **read-only** data used in the execution of the code
- **.data** — where **read-write** variables and **pointers** are stored
- **.bss** — a part of the **data segment** containing **statically-allocated** variables

- **.heap** — where **dynamically allocated** memory is located
- **.stack** — where **function-CALL** parameters and other temporary data is stored

Step 3.) Build SW application

- After building the `DipTest` sw application the size of the generated, downloadable `DipTest.elf` file as follows:

```
***** Determining Size of ELF File *****  
  
mb-size DipTest.elf  
text    data    bss      dec    hex      filename  
5758    332     1080     7170   1c02     DipTest.elf
```

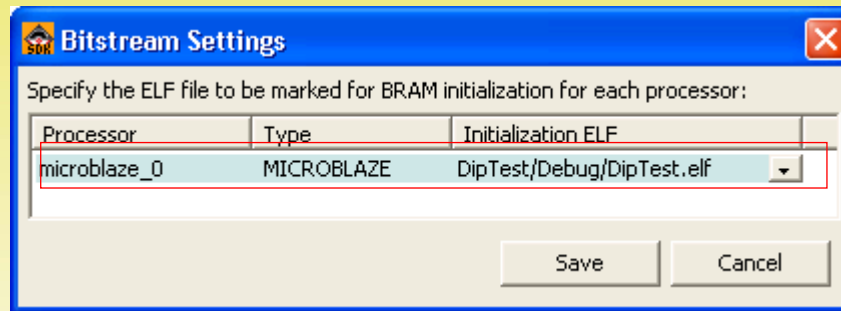
OK. It is (7170 bytes of total program code) fitted to the 32KByte BRAM internal memory.

Step 4. Terminal Program

- Set the following parameters **properly** (see the parameters of `xps_uartlite` in the `.mhs` file!)
 - Com port: COMX
 - Baud Rate: 9600
 - Data Bits: 8
 - Stop Bits: 1
 - Parity Bit: None
 - Flow control: none

Step 5. Method a.) Programming the FPGA via Xilinx Impact

- **Select Device Configuration menu -> Bitstream settings**
 - Select compiled DipTest.elf file for running MicroBlaze sw codes



- **Connect the Xilinx JTAG-Platform USB cable to Nexys-2 board's JTAG interface**
- **Select Device Configuration menu -> Program FPGA**
 - Bitstream (**system.bit**)
 - BRAM Memory Map (**.bmm**) + DipTest.elf
 - -> **D:\FPGA\BEAGYAZOTT_RENDSZEREK\10_1\02_LAB\SDK\SDK_projects\implementation\download_sdk.bit**

Step 5. Method b.) Programming the FPGA via Digilent Adept



- Instead of using the **Xilinx iMPact**, we **use** **Digilent Adept Suite!** programmer provided by DigilentInc (vendor of the FPGA board).
- Browse your `SDK_project\implementation\` directory for „download_sdk.bit” bitstream file.
- Use and set properly the terminal program (e.g. Windows Hyperterminal, Teraterm Pro, or Putty etc.)
- At the final step Program the FPGA!
- **At now the Lab 2/A is completed in SDK 10.1 SP3 ☺**

Questions

- Open the **system.mhs** file, study its contents, and answer the following questions
- Number of external ports: _____
- Number of external ports that are output (O): _____
- Number of external ports that are input (I): _____
- Num. of external ports that are bidirectional (IO): _____
- Number of clock ports: _____ Freq: _____
- Number of reset ports: _____ Polarity: _____

Questions

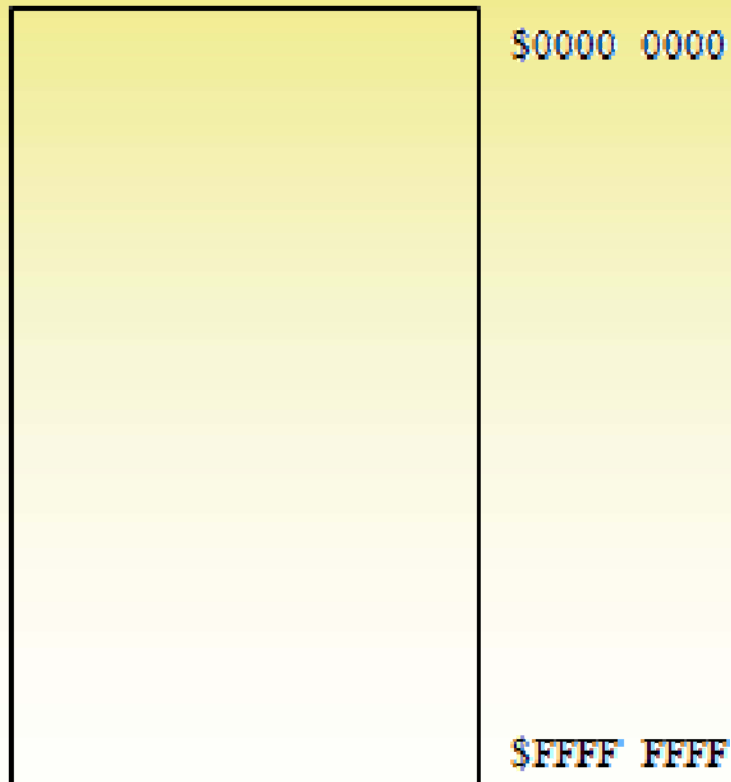
- List the **instances** to which the `clk_s` is connected:

- List the **instances** connected to the `mb_plb` bus:



Questions

- Draw the **address map** of the system, providing instance names:



Questions

- **Check Report files (system.par) or log messages in Consol window after the placement process step):**

Logic Utilization:

Number of Slice Flip Flops: _____ out of 17,344 11%

Number of 4 input LUTs: _____ out of 17,344 19%

Logic Distribution:

Number of occupied Slis: _____ out of 8,672 29%

Number of External IOBs _____ out of 250 22%

Number of External Input IOBs _____

Number of External Output IOBs _____

Number of External Bidir IOBs _____

Number of BSCANs _____ out of 1 100%

Number of BUFGMUXs _____ out of 24 8%

Number of DCMs _____ out of 8 12%

Number of MULT18X18SIOS _____ out of 28 10%

Number of RAMB16s _____ out of 28 71%

Number of Slices _____ out of 8672 29%

Number of SLICEMs _____ out of 4336 6%

Laboratory task 2/B: Push Buttons

- In XPS/EDK:
 - Similar to the Task 2/A add Push Buttons (4 bit) as new peripheral into the elaborated embedded system design
 - Rename it: „push”, and add a particular address
 - Generate netlist and bitstream
- In XPS/SDK:
 - Similar to the Task 2/A, create modify the previous *DipTest* sw application in SDK
 - Create *PeripheralTestsApp_bsp*
 - Implement BSP, generate Linker Script (.ld)
 - Compile MB codes (.elf file)
 - Generate Bitstream (.bit)
- Download bitstream (.bit), and analyze the desing on the Digilent Nexys2 FPGA board.
- Try to answer the questions according to the TASK 2/A.

Question

- What is the size of **.elf** program, and the different program sections?
- Which is the `base_address` and `high_address` (or address size) of the push button GPIO peripheral?
- Which header `.h` file contains the MicroBlaze system parameters for various peripherals?