http://zs.utia.cas.cz

Application Note

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

UTIA EdkDSP Platform Demonstrator on Xilinx SP605 Board – PLB Bus

Jiří Kadlec
kadlec@utia.cas.cz
phone: +420 2 6605 2216
UTIA AV CR, v.v.i.

Revision history:

Rev.	Date	Author	Description
1	23.8.2014	Jiří Kadlec	First draft
2	27.8.2014	Jiří Kadlec	Update of text, References.
3	30.12.2014	Jiří Kadlec	EdkDSP C compiler is part of the SDK project edkdsp_cc, Description of firmware C code.

Acknowledgements:

This work has been partially supported by the ENIAC JU project "Interactive Power Devices for Efficiency in Automotive with Increased Reliability and Safety" (IDEAS). The IDEAS project is an **ENIAC JU** project number 304603. This work has been also partially supported by the Ministry of Education Youth and Sports of the Czech Republic project No. 7H12004. This project is providing the Czech Republic part of funding for the ENIAC JU project IDEAS.

Table of contents

1. Summary

1.1 Key features

This application note describes the EdkDSP platform demonstrator package provided by UTIA for the Xilinx SP605 designs with PLB bus. It explains how to install and use the demonstrator on Windows7, (32 or 64 bit).

- Implementation of adaptive acoustic noise cancellation: Demonstrated accelerators compute the recursive adaptive LMS algorithm for regression filter with 2000 coefficients with the sampling frequency 80 KHz in single precision floating point arithmetic. This corresponds to the sustained performance of 580 MFLOP/s.
- The accelerators can be reprogrammed by the firmware. The programming is possible in C with the use of the UTIA EDKDSP C compiler. Accelerators can be programmed with two firmware programs. Designs can swap in the real time the firmware in only few clock cycles in the runtime.
- The alternative firmware can be downloaded to the EdkDSP accelerators in parallel with the execution of the current firmware. This is demonstrated by swap of the firmware for the FIR filter room response to the firmware for adaptive LMS identification of the filter coefficients in the acoustic noise cancellation demo.
- The EdkDSP accelerator is providing bit-exact identical single-precision floating point results as the reference software implementation running on MicroBlaze with the Xilinx HW single precision floating point unit.
- The EdkDSP accelerator is 140x - 160x faster than computation on area optimized 83,3 MHz MicroBlaze with HW floating point unit, in presented case of the 2000 tap adaptive LMS filter.
- The floating point 2000 tap coefficients FIR filter (acoustics room model) is computed with the floating point performance 820 MFLOP/s. The peak performance (only theoretical) is 1.6 GFLOP/s.
- The EdkDSP accelerators are internally organised with 8xSIMD data paths.
- This evaluation package presents 2 EdkDSP accelerator families: one family without pipelined floating point divider data path and one family with a single pipelined floating point divider data path. The members of both families differ by size and by supported vector floating point operations.
- The floating point applications can be scheduled inside of the EdkDSP accelerator by the Xilinx PicoBlaze6 processor. Each firmware program has maximal size of 4096 (18 bit wide words).
- All designs include Xilinx display controller 1280x792p60 for digital DVI or analogue RGB 24 bit display.

1.2 What is included

This EdkDSP platform evaluation package includes these deliverables for the Windows 7 (32 or 64bit):

- 8 evaluation versions of EdkDSP accelerators with 8xSIMD floating point data paths integrated in 8 precompiled designs (MicroBlaze 83,3 MHz, Accelerator 95,2 MHz) and in 8 precompiled designs (MicroBlaze 83,3 MHz, Accelerator 83,3 MHz) with PLB bus. Designs are compiled in Xilinx XPS 14.5.
- All 16 precompiled designs include the Xilinx PLB bus video controller and an evaluation version of the Xilinx xps_Soft_TEMAC 1Gb Ethernet controller with SDMA data path to the external DDR3 memory.
- SW demos are using the Xilinx Xilkernel OS, LwIP package and the memory file system xilmfs.
- UTIA is providing source code for the demo applications and SW projects for the Xilinx SDK 14.5. These source code projects are compiled with the UTIA libraries libwal.a, libjpg.a, librgb.a and libmfsimage.a.
- The evaluation versions of the UTIA EdkDSP accelerators compiled in the designs and the Xilinx TEMAC 1Gb Ethernet controller compiled in the designs run only for a limited time. The time limit for the Xilinx TEMAC is several hours. Evaluation versions of the UTIA EdkDSP accelerators are related to maximal number of performed vector operations.
- The UTIA EdkDSPC C compiler is provided as 3 executable applications for Ubuntu in the VMware Player.
- The firmware is also provided in precompiled files to enable testing of accelerators without C compiler.
- A release version of the EdkDSP package for SP605, PLB bus is offered by UTIA. It provides EdkDSP accelerators for the SP605 in form of PLB netlist pcores with main limitations of the free evaluation package removed. See sections 4-6 of this application note for specification of deliverables and license details.

2. Demonstrator of EdkDSP platform on SP605 board with PLB

2.1 Description of EdkDSP accelerators and evaluation designs

This application note describes how to set-up and use collection of 8 HW designs on Xilinx SP605 board with PLB bus. The demonstrator serves to evaluation of parameters of two floating point accelerator families:

- **bce_fp11_1x8_0_plbw_v1_[10|20|30|40]_a** is a family of four floating point accelerators with 8 SIMD data paths.
- **bce_fp12_1x8_0_plbw_v1_[10|20|30|40]_a** is similar family of 4 floating point accelerators with 8 SIMD data paths extended by pipelined floating point division (FPDIV) in single data path.

The four grades [10|20|30|40] of EdkDSP accelerator differ in HW-supported vector computing capabilities.

The area optimized accelerators **bce_fp11_1x8_0_plbw_v1_10_a** and **bce_fp12_1x8_0_plbw_v1_10_a** perform vector floating point operations FPADD, FPSUB in 8 SIMD data paths.

The accelerators **bce_fp11_1x8_0_plbw_v1_20_a** and **bce_fp12_1x8_0_plbw_v1_20_a** perform vector floating point operations FPADD, FPSUB in 8 SIMD data paths plus the vector floating point MAC operations in 8 SMD data paths for length of the vector 1 up to 10. These accelerators can be used in applications like floating point matrix multiplication with row and column dimensions <= 10.

The accelerators **bce_fp11_1x8_0_plbw_v1_30_a** and **bce_fp12_1x8_0_plbw_v1_30_a** support identical operations as the bce_fp11_1x8_0_plbw_v1_20_a and bce_fp12_1x8_0_plbw_v1_20_a plus the floating point vector by vector dot products performed in 8 SIMD data paths. It is optimized for parallel computation of up to 8 FIR or LMS filters, each with size up to 250 coefficients. It is also effective in case of floating point matrix by matrix multiplications, where one of the dimensions is large (in the range from 11 to 255).

Finally, the accelerators **bce_fp11_1x8_0_plbw_v1_40_a** and **bce_fp12_1x8_0_plbw_v1_40_a** support identical operations as the bce_fp11_1x8_0_plbw_v1_30_a and bce_fp12_1x8_0_plbw_v1_30_a plus an additional HW support of dot product into 8 sections computed in 8 data paths in parallel with HW supported wind-up into scalar result in the end of the batch computation.

The **bce_fp11** versions of 8x SIMD accelerators has no support for pipelined vector floating point division and it is suitable for applications like FIR filters or adaptive LMS filters with no need for floating point division.

The **bce_fp12** versions of 8x SIMD accelerators are larger in comparison to the bce_fp11 equivalents and support in single data path the pipelined vector floating point division. Accelerators are suitable for applications like adaptive normalised NLMS filters and the square root free versions of adaptive RLS QR filters and adaptive RLS LATTICE filters.

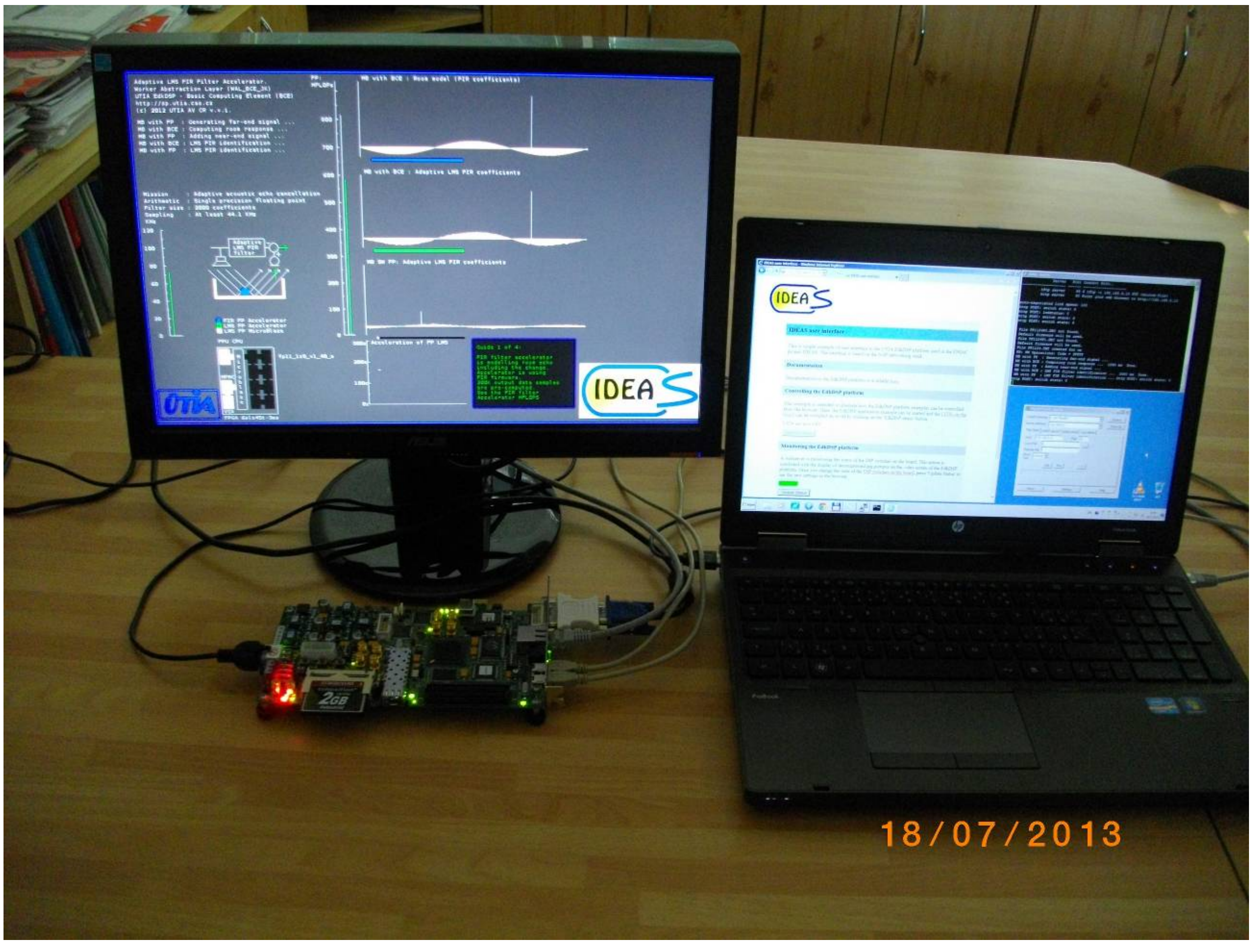


Figure 1: EdkDSP demonstrator; FIR filter 820 MFLOPS; adaptive LMS filter 580 MFLOPS

Eight precompiled HW designs demonstrate use of single instance of UTIA EdkDSP floating point accelerator on 32bit PLB_v46 bus of the Xilinx MicroBlaze soft-core processor on the Xilinx Spartan 6 SP605 FPGA board with system clock of accelerator 83,3 MHz. See Figure 2.

Another set of eight precompiled HW designs also demonstrate use of single instance of UTIA EdkDSP floating point accelerator on the same board, but with system clock of accelerator 95,2 MHz.

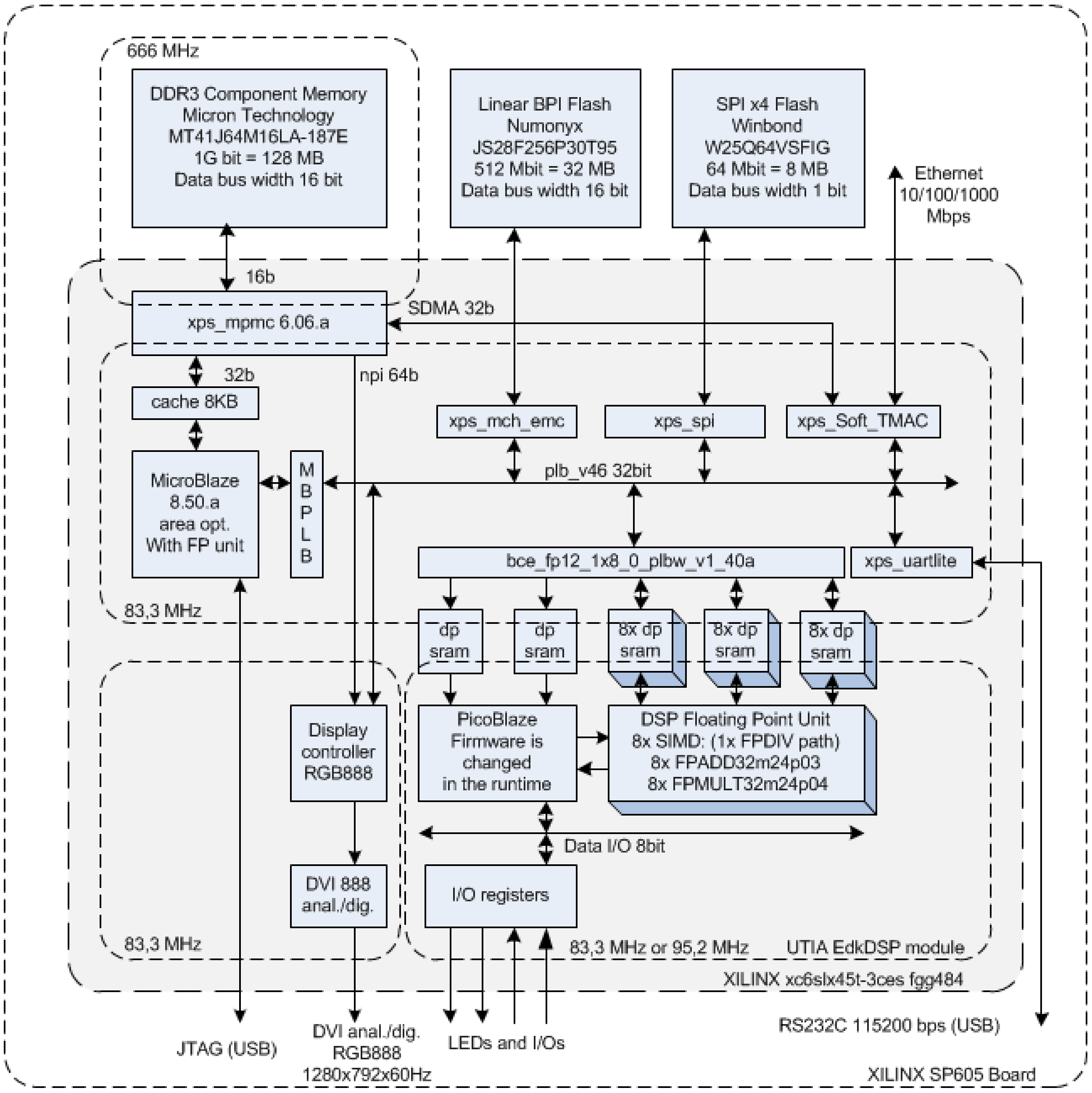
Common properties of all 16 precompiled evaluation designs:

- The EdkDSP floating point accelerators are reconfigurable during runtime by change of firmware.
- 1 Gbit Ethernet point-to point connectivity with the PC.
- MicroBlaze system.
- Designed in Xilinx EDK/ISE 14.5 tools.
- Video display with fixed resolution 1280x792 60Hz with progressive scan. The video clock is 83,3 MHz
- Video display is supporting:
    - The digital DVI (or HDMI) flat panel monitors with (RGB, 8 bit per pixel).
    - The legacy flat panel monitors with analogue VGA input (RGB, 8 bit per pixel).

Presented HW accelerators can results in better POWER per MFLOPS ratio for certain class of DSP applications in comparison to the computation on standard CPU with HW floating point support.

The demonstrator includes source code of set of SW demos prepared for compilation in the Xilinx SDK 14.5.

Xilinx display controller is used to visualize the FIR RLS system parameter tracking by the recursive LMS algorithm. Results from accelerators are identical (bit-exact) with the results computed by the MicroBlaze (with HW floating point support). Video display is also demonstrating the measured acceleration of computation by this set of EdkDSP accelerators. See Figure 1.



666 MHz
DDR3 Component Memory Micron Technology MT41J64M16LA-187E 1G bit = 128 MB Data bus width 16 bit
Linear BPI Flash Numonyx JS28F256P30T95 512 Mbit = 32 MB Data bus width 16 bit
SPI x4 Flash Winbond W25Q64VSFIG 64 Mbit = 8 MB Data bus width 1 bit
Ethernet 10/100/1000 Mbps
16b
xps_mpmc 6.06.a
SDMA 32b
32b
npi 64b
cache 8KB
xps_mch_emc
xps_spi
xps_Soft_TMAC
MicroBlaze 8.50.a area opt. With FP unit
MBPLB
plb_v46 32bit
bce_fp12_1x8_0_plbw_v1_40a
xps_uartlite
83,3 MHz
dp sram
dp sram
8x dp sram
8x dp sram
8x dp sram
Display controller RGB888
PicoBlaze Firmware is changed in the runtime
DSP Floating Point Unit 8x SIMD: (1x FPDIV path) 8x FPADD32m24p03 8x FPMULT32m24p04
Data I/O 8bit
DVI 888 anal./dig.
I/O registers
83,3 MHz
83,3 MHz or 95,2 MHz
UTIA EdkDSP module
XILINX xc6slx45t-3ces fgg484
JTAG (USB)
DVI anal./dig. RGB888 1280x792x60Hz
LEDs and I/Os
RS232C 115200 bps (USB)
XILINX SP605 Board

Figure 2: Simplified architecture of all 16 EdkDSP evaluation designs; SP605 board; PLB bus.

2.2 Resources used by the designs

The resources used by the 16 presented designs are summarised in Figure 3 - Figure 6.

6slx45-3c	fp	fp	fp	fp	fp			Design size		Performance	
	Add		Dot	S8		ff	Lut	Bram	Slices	LMS	FIR
	Mul	Mac	Prod	Prod	div	%	%	no(of)	%	Mflop/s	Mflop/s
fp11_1x8_10	8x					27	64	97(116)	90	3,6	3,6
fp11_1x8_20	8x	8x				29	67	97(116)	89	3,6	3,6
fp11_1x8_30	8x	8x	8x			31	77	97(116)	96	3,6	3,6
fp11_1x8_40	8x	8x	8x	1x		31	81	97(116)	98	**520**	**740**

Figure 3: MicroBlaze 83.3 MHz; 8xSIMD EdkDSP 83.3 MHz; no FP division

6slx45-3c	fp	fp	fp	fp	fp			Design size		Performance	
	Add		Dot	S8		ff	Lut	Bram	Slices	LMS	FIR
	Mul	Mac	Prod	Prod	div	%	%	no(of)	%	Mflop/s	Mflop/s
fp12_1x8_10	8x				1x	29	66	97(116)	92	3,6	3,6
fp12_1x8_20	8x	8x			1x	31	74	97(116)	94	3,6	3,6
fp12_1x8_30	8x	8x	8x		1x	33	78	97(116)	95	3,6	3,6
fp12_1x8_40	8x	8x	8x	1x	1x	33	76	97(116)	97	**520**	**740**

Figure 4: MicroBlaze 83.3 MHz; 8xSIMD EdkDSP 83.3 MHz with FP division

6slx45-3c	fp	fp	fp	fp	fp			Design size		Performance	
	Add		Dot	S8		ff	Lut	Bram	Slices	LMS	FIR
	Mul	Mac	Prod	Prod	div	%	%	no(of)	%	Mflop/s	Mflop/s
fp11_1x8_10	8x					27	64	97(116)	88	3,6	3,6
fp11_1x8_20	8x	8x				29	67	97(116)	93	3,6	3,6
fp11_1x8_30	8x	8x	8x			31	77	97(116)	97	3,6	3,6
fp11_1x8_40	8x	8x	8x	1x		31	80	97(116)	97	**580**	**820**

Figure 5: MicroBlaze 83.3 MHz; 8xSIMD EdkDSP 95.2 MHz; no FP division

6slx45-3c	fp	fp	fp	fp	fp			Design size		Performance	
	Add		Dot	S8		ff	Lut	Bram	Slices	LMS	FIR
	Mul	Mac	Prod	Prod	div	%	%	no(of)	%	Mflop/s	Mflop/s
fp12_1x8_10	8x				1x	29	66	97(116)	91	3,6	3,6
fp12_1x8_20	8x	8x			1x	31	74	97(116)	98	3,6	3,6
fp12_1x8_30	8x	8x	8x		1x	33	78	97(116)	96	3,6	3,6
fp12_1x8_40	8x	8x	8x	1x	1x	33	76	97(116)	97	**580**	**820**

Figure 6: MicroBlaze 83.3 MHz; 8xSIMD EdkDSP 95.2 MHz with FP division

2.3 Use of external DDR3 memory

Presented FPGA designs are running in the Xilinx SP605 development board. It is using the Micron DDR3 component memory MT41J64M16LA-187E with clock signal 666 MHz. The DDR3 component is connected to Xilinx Spartan xc6slx45t-3 FPGA. The 16 data path to DDR3 memory can support up to 1332 MB/s RD and up to 1332 MB/s WR at 666 MHz. It is actually supporting in all 16 designs:

- 32bit wide cache interface to MicroBlaze processor with floating point unit operating at 83,3 MHz (max 333 MB/s RD or 333 MB/s WR)
- 32bit wide SDMA data path of the TEMAC 1Gbit Ethernet unit operating at 83,333 MHz (max 333 MB/s RD or 333 MB/s WR)
- 64bit wide video display interface with the resolution 1280x792x60Hz operating at 83,333 MHz (average bandwidth 333 MB/s RD)

Peak of the requested data transfer rate:

- RD from the DDR3 memory: (83,3 MHz*4B + 83,3 MHz*4B + 83,3 MHz*8B) = 1322 MB/s (100%)
- WR to the DDR3 memory: (83,3 MHz*4B + 83,3 MHz*4B) = 666 MB/s (50%)

Average data transfer rate:

- RD from the DDR3 memory: (83,3 MHz*4B + 83,3 MHz*4B + 83,3 MHz*4B) < 991 MB/s (< 75% of max theoretical bandwidth).
- WR to the DDR3 memory: (83,3 MHz*4B + 83,3 MHz*4B) < 666 MB/s (< 50% of max theoretical bandwidth).

2.4 Re-programmability of EdkDSP accelerators

Each of The 83MHz/95MHz EdkDSP floating point accelerator subsystem contains one reprogrammable Xilinx PicoBlaze6 processor and the floating point 8xSIMD DSP unit. It is delivering sustained 740/820 MFLOP/s in case of 2000 tap FIR filter computation and 520/580 MFLOP/s in case of the adaptive 2000 tap LMS filter identification.

The Xilinx PicoBlaze6 processor has fixed configuration with size of the program memory 4096 (18 bit wide) words, 64 Bytes scratch pad RAM memory and the interrupt vector in the address 1023.

All EdkDSP accelerator designs present in this package instantiate 2 program memories, each with the 4096 (18bit wide) words. Both program memories are accessible by MicroBlaze processor via PLB bus. The PicoBlze6 can execute program from each of these memories. The MicroBlaze application can write new firmware to the currently unused program memory, while the PicoBlaze6 is executing firmware from the other program memory.

2.5 Debug of EdkDSP accelerators in the evaluation package

All EdkDSP accelerators can communicate with MicroBlaze program. The communication is using the Worker Abstraction Layer (WAL) library API. This API is used for support of writing of the debug information from the worker to the MicroBlaze terminal, memory based file system or graphical display.

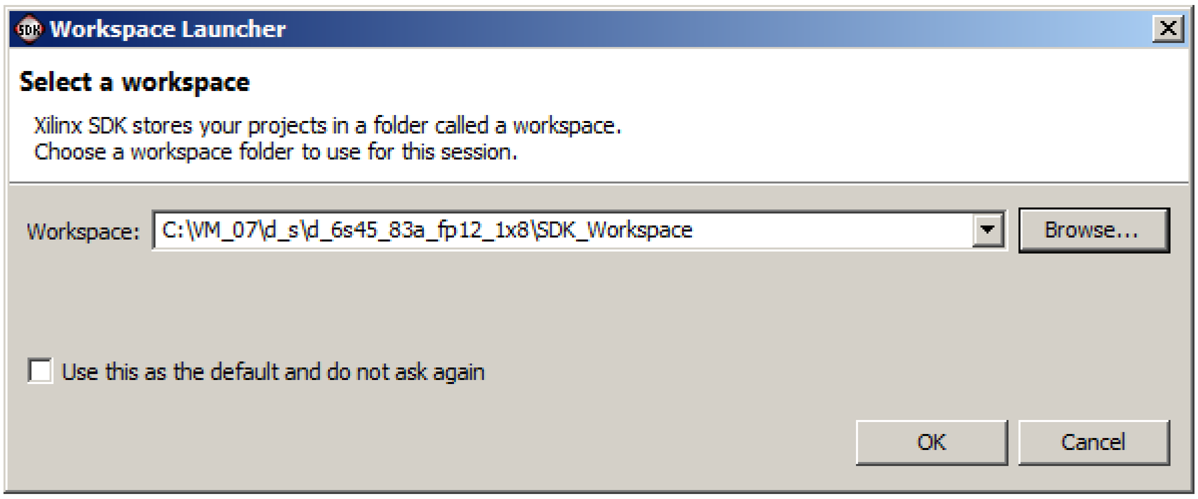
3. Installation of EdkDSP platform on SP605 board with PLB

3.1 Import of precompiled projects and SW into Xilinx SDK 14.5

Unzip the evaluation package to directory of your choice. The directory c:\VM_07 will be used in this application note. You will get these directories:

```
c:\VM_07\d_145_6s_plb
22.12.2014  18:00   <DIR>       .
22.12.2014  18:00   <DIR>       ..
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp11_1x8_v1_10a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp11_1x8_v1_20a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp11_1x8_v1_30a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp11_1x8_v1_40a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp12_1x8_v1_10a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp12_1x8_v1_20a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp12_1x8_v1_30a
22.12.2014  18:00   <DIR>       d_6s45_83a_95_fp12_1x8_v1_40a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp11_1x8
22.12.2014  18:00   <DIR>       d_6s45_83a_fp11_1x8_IMPORT
22.12.2014  18:00   <DIR>       d_6s45_83a_fp11_1x8_v1_10a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp11_1x8_v1_20a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp11_1x8_v1_30a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp11_1x8_v1_40a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp12_1x8
22.12.2014  18:00   <DIR>       d_6s45_83a_fp12_1x8_IMPORT
22.12.2014  18:00   <DIR>       d_6s45_83a_fp12_1x8_v1_10a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp12_1x8_v1_20a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp12_1x8_v1_30a
22.12.2014  18:00   <DIR>       d_6s45_83a_fp12_1x8_v1_40a
```

Select SDK 14.5 workspace in c:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace.


Workspace Launcher

Select a workspace

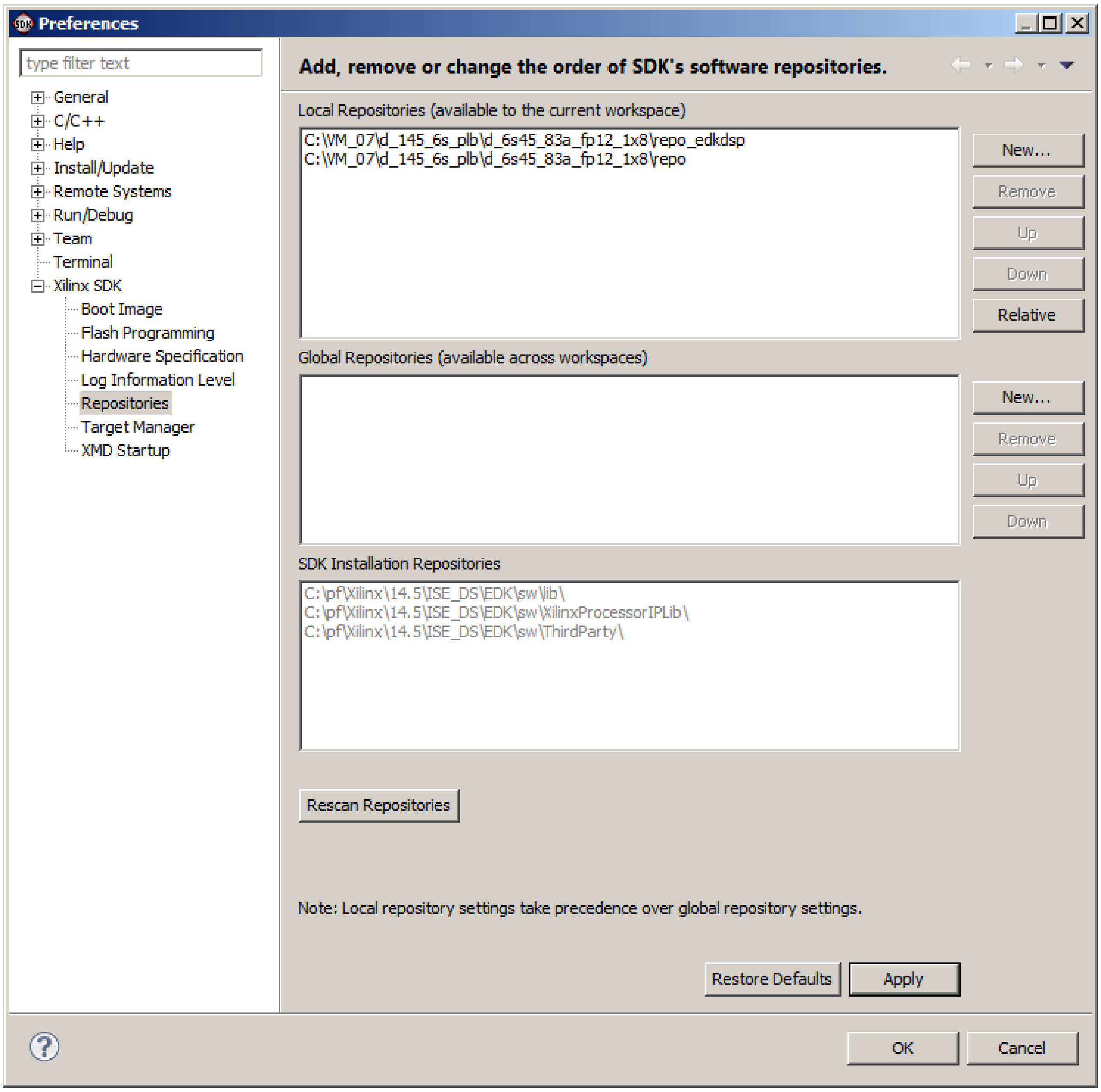
Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\VM_07\d_s\d_6s45_83a_fp12_1x8\SDK_Workspace

☐ Use this as the default and do not ask again

Figure 7: Select the SDK 14.5 Workspace

Add path to the UTIA EdkDSP repository of SW drivers and path to the Xilinx repository of SW drivers c:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\repo_edkdsp.
c:\VM_07\ d_145_6s_plb \d_6s45_83a_fp12_1x8\repo.



Add, remove or change the order of SDK's software repositories.

Local Repositories (available to the current workspace)

C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\repo_edkdsp
C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\repo

Global Repositories (available across workspaces)

SDK Installation Repositories

C:\pf\Xilinx\14.5\ISE_DS\EDK\sw\lib\
C:\pf\Xilinx\14.5\ISE_DS\EDK\sw\XilinxProcessorIPLib\
C:\pf\Xilinx\14.5\ISE_DS\EDK\sw\ThirdParty\

Note: Local repository settings take precedence over global repository settings.

Figure 8: Include the Xilinx SW repository and the UTIA EdkDSP SW repository

Click on the “Rescan Repositories” button. Click on the “Apply button”, and finally click on the OK button. The paths to the SW drivers have been defined.

In SDK, select File -> New -> Project … -> Xilinx -> Hardware Platform Specification.
Click on the Next button.

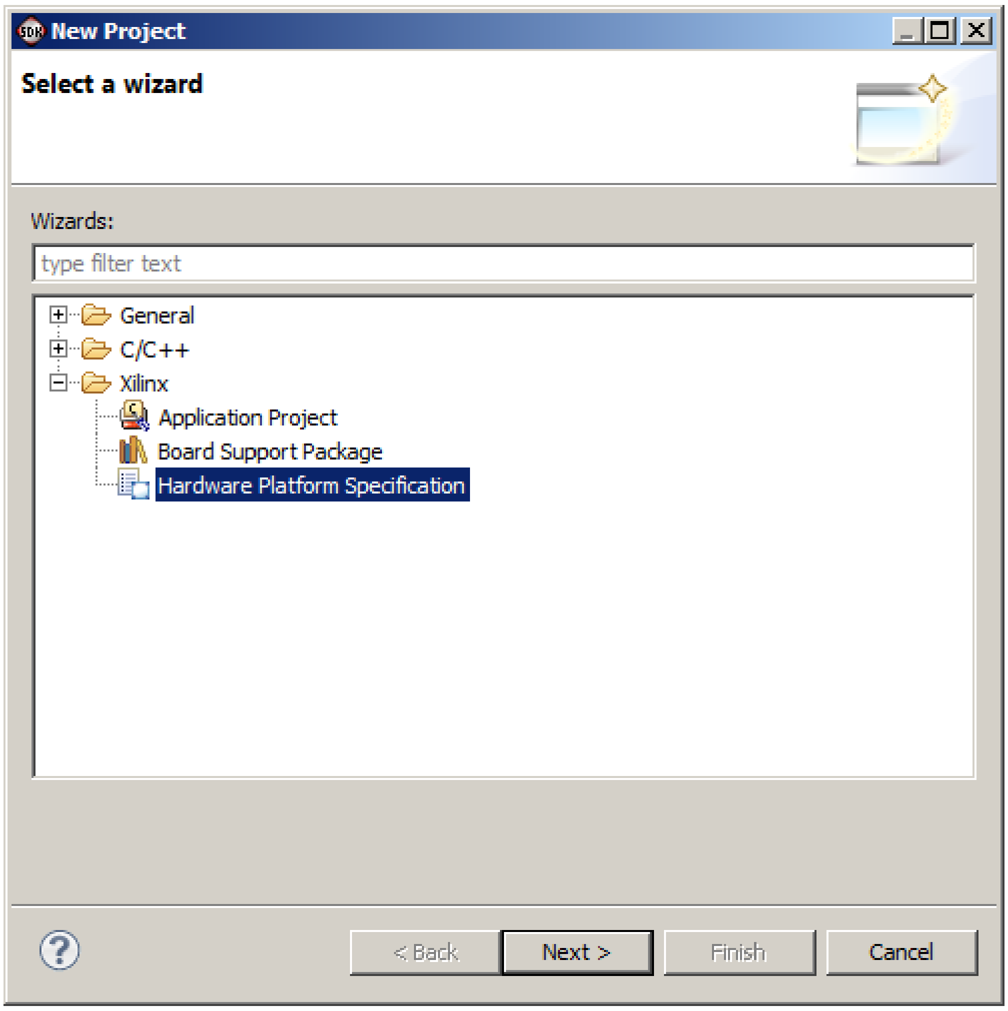


Figure 9: Specify the Hardware Platform

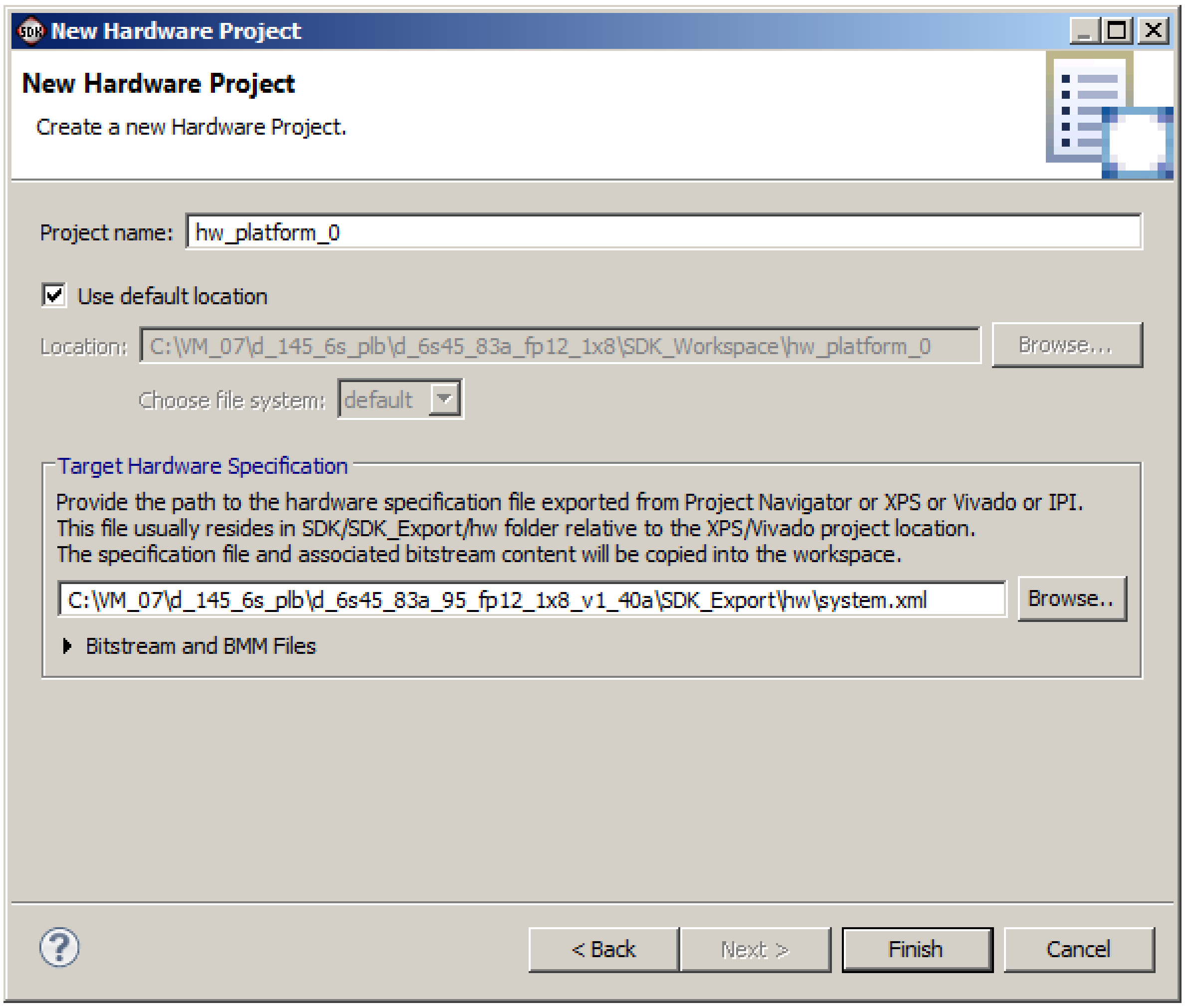
In the “New Hardware Project” screen, fill into the Project name:
hw_platform_0

In the New Hardware Project screen, fill into the Target Hardware Specification:

C:\VM_07\d_145_6s_plb\d_6s45_83a_95_fp12_1x8_v1_40a\SDK_Export\hw\system.xml

This will specify one of the 16 precompiled HW designs present in the evaluation package.
We have selected design demonstrating the UTIA EdkDSP accelerator with 8xSIMD data path, with floating point single data path division, with 95,2 MHz clock bce_fp12_1x8_0_plbw_v1_40_a.

Click on “Finish” button to finalize the selection of the precompiled HW design.


New Hardware Project

New Hardware Project

Create a new Hardware Project.

Project name: hw_platform_0

[✓] Use default location

Location: C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform_0

Choose file system: default

Target Hardware Specification

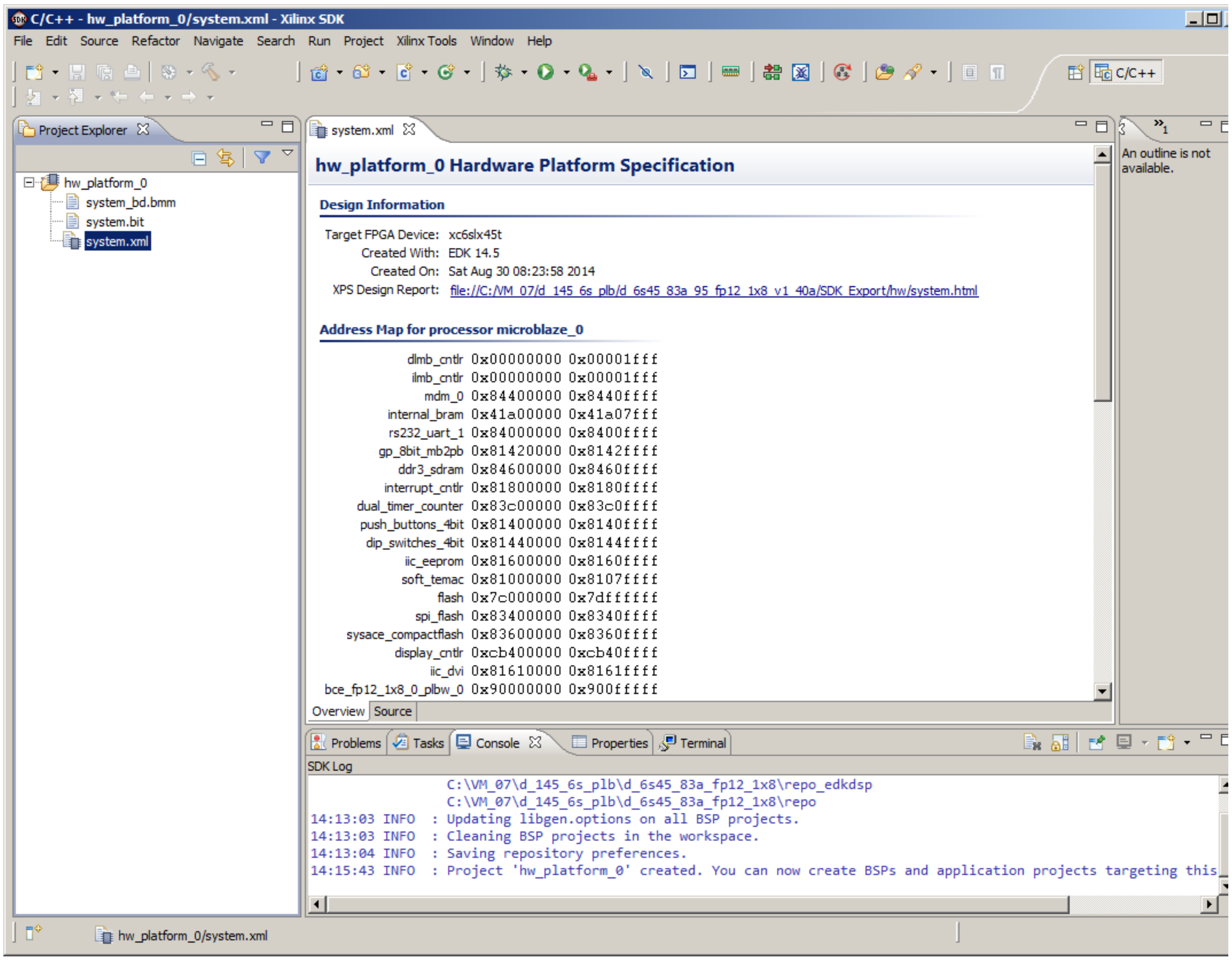
Provide the path to the hardware specification file exported from Project Navigator or XPS or Vivado or IPI.
This file usually resides in SDK/SDK_Export/hw folder relative to the XPS/Vivado project location.
The specification file and associated bitstream content will be copied into the workspace.

C:\VM_07\d_145_6s_plb\d_6s45_83a_95_fp12_1x8_v1_40a\SDK_Export\hw\system.xml

Figure 10: Use the name “hw_platform_0” and selet one of the provided xml design descriptions

The hardware platform “hw_platform” has been created.

SDK is interpreting the system.xml and presents HW cores of in the design. See Figure 11.


C/C++ - hw_platform_0/system.xml - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
C/C++
Project Explorer
hw_platform_0
system_bd.bmm
system.bit
system.xml
system.xml
An outline is not available.
Overview Source
Problems Tasks Console Properties Terminal
SDK Log
hw_platform_0/system.xml

hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device:	xc6slx45t
Created With:	EDK 14.5
Created On:	Sat Aug 30 08:23:58 2014
XPS Design Report:	file://C:/VM_07/d_145_6s_plb/d_6s45_83a_95_fp12_1x8_v1_40a/SDK_Export/hw/system.html

Address Map for processor microblaze_0

dlmb_cntlr	0x00000000	0x00001fff
ilmb_cntlr	0x00000000	0x00001fff
mdm_0	0x84400000	0x8440ffff
internal_bram	0x41a00000	0x41a07fff
rs232_uart_1	0x84000000	0x8400ffff
gp_8bit_mb2pb	0x81420000	0x8142ffff
ddr3_sdram	0x84600000	0x8460ffff
interrupt_cntlr	0x81800000	0x8180ffff
dual_timer_counter	0x83c00000	0x83c0ffff
push_buttons_4bit	0x81400000	0x8140ffff
dip_switches_4bit	0x81440000	0x8144ffff
iic_eeprom	0x81600000	0x8160ffff
soft_temac	0x81000000	0x8107ffff
flash	0x7c000000	0x7dffffff
spi_flash	0x83400000	0x8340ffff
sysace_compactflash	0x83600000	0x8360ffff
display_cntlr	0xcb400000	0xcb40ffff
iic_dvi	0x81610000	0x8161ffff
bce_fp12_1x8_0_plbw_0	0x90000000	0x900fffff

```
                      C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\repo_edkdsp
                      C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\repo
14:13:03 INFO  : Updating libgen.options on all BSP projects.
14:13:03 INFO  : Cleaning BSP projects in the workspace.
14:13:04 INFO  : Saving repository preferences.
14:15:43 INFO  : Project 'hw_platform_0' created. You can now create BSPs and application projects targeting this
```

Figure 11: Hardware platform with the MicroBlaze address map

SW projects can be imported into SDK now. Select:

File -> Import -> General -> Existing Projects into Workspace
Click on Next button. See Figure 12.

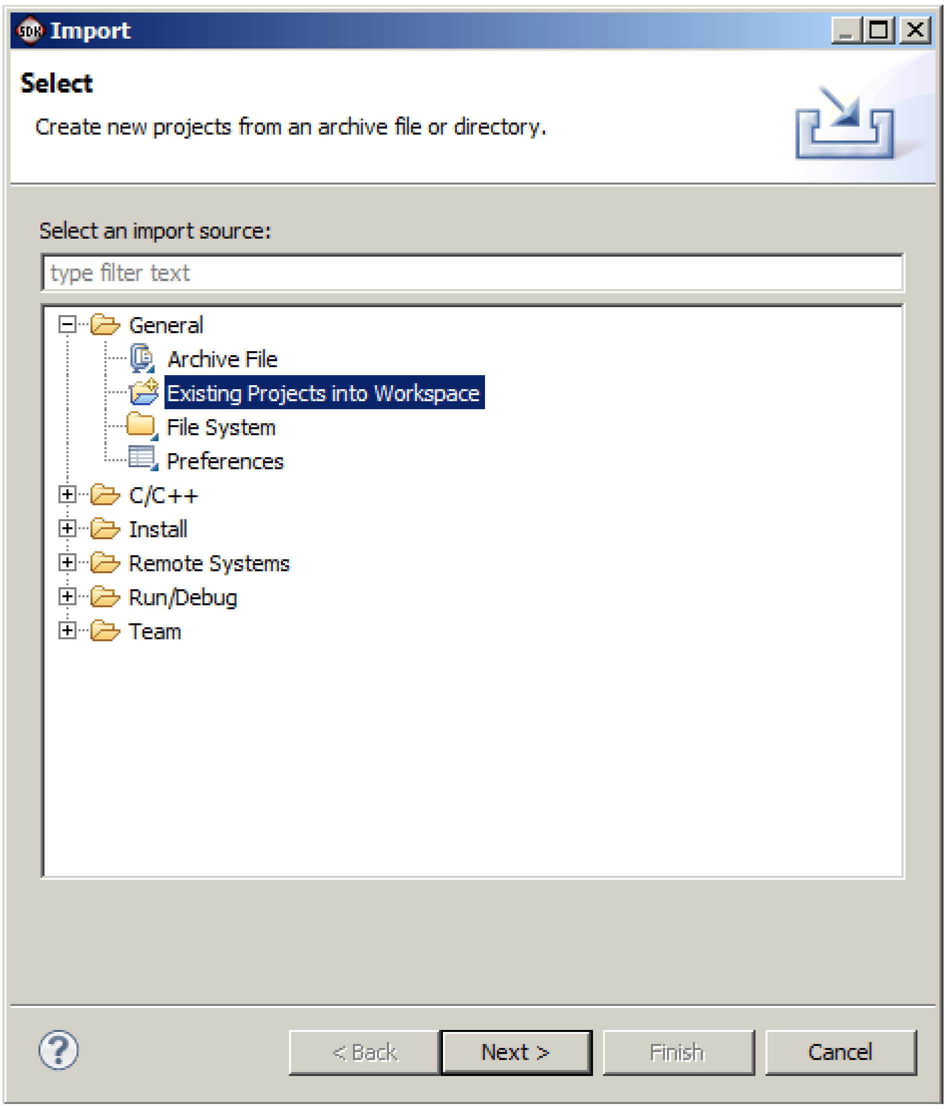
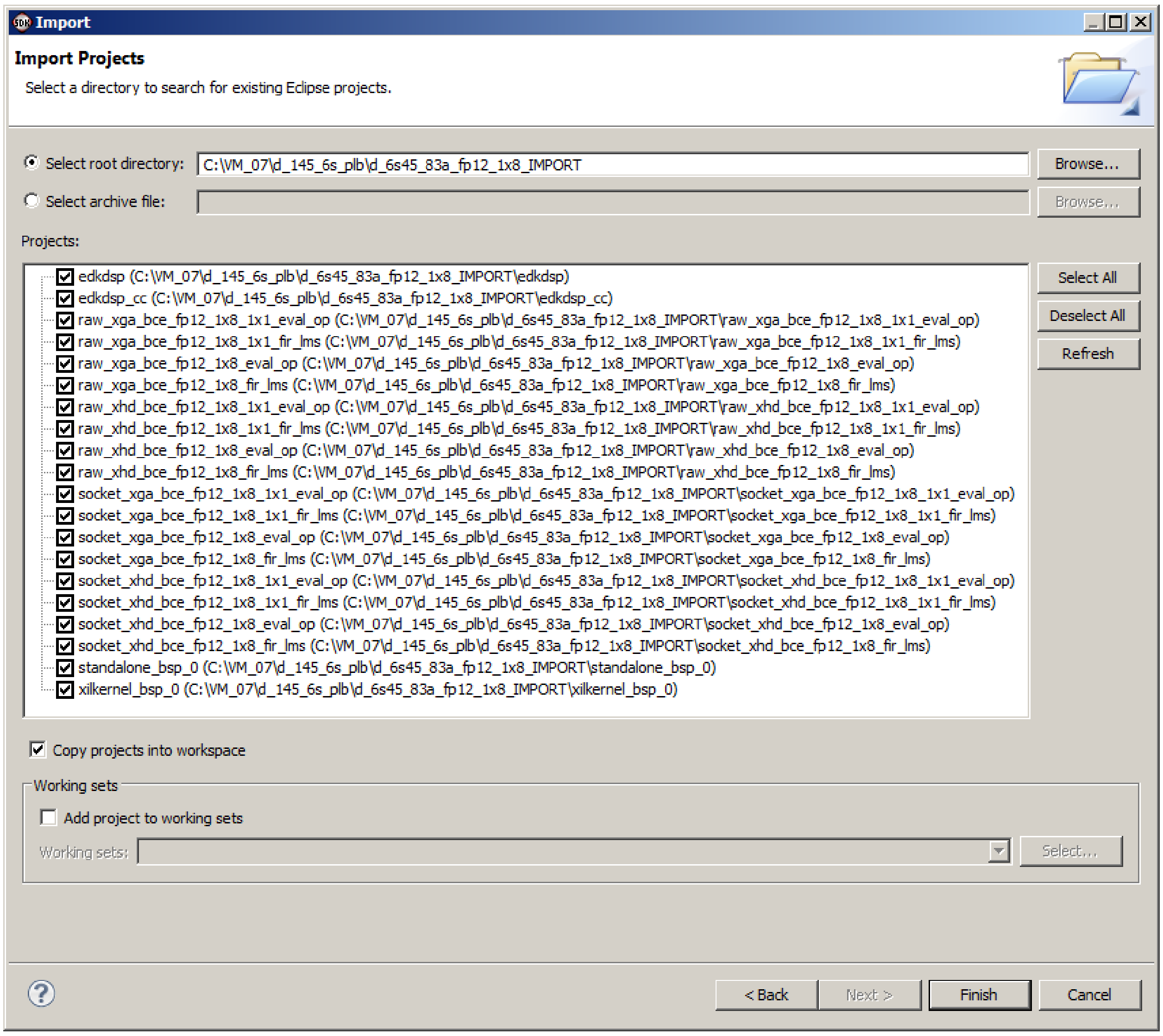


Figure 12: Import existing projects into Workspace

Type directory with projects to be imported. See Figure 13.

C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT

Set the "Copy projects into workspace" check box.
Click on Finish button. See Figure 13.



(● Select root directory:) C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT

Select archive file:

Projects:

- ☑ edkdsp (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\edkdsp)
- ☑ edkdsp_cc (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\edkdsp_cc)
- ☑ raw_xga_bce_fp12_1x8_1x1_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xga_bce_fp12_1x8_1x1_eval_op)
- ☑ raw_xga_bce_fp12_1x8_1x1_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xga_bce_fp12_1x8_1x1_fir_lms)
- ☑ raw_xga_bce_fp12_1x8_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xga_bce_fp12_1x8_eval_op)
- ☑ raw_xga_bce_fp12_1x8_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xga_bce_fp12_1x8_fir_lms)
- ☑ raw_xhd_bce_fp12_1x8_1x1_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xhd_bce_fp12_1x8_1x1_eval_op)
- ☑ raw_xhd_bce_fp12_1x8_1x1_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xhd_bce_fp12_1x8_1x1_fir_lms)
- ☑ raw_xhd_bce_fp12_1x8_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xhd_bce_fp12_1x8_eval_op)
- ☑ raw_xhd_bce_fp12_1x8_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\raw_xhd_bce_fp12_1x8_fir_lms)
- ☑ socket_xga_bce_fp12_1x8_1x1_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xga_bce_fp12_1x8_1x1_eval_op)
- ☑ socket_xga_bce_fp12_1x8_1x1_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xga_bce_fp12_1x8_1x1_fir_lms)
- ☑ socket_xga_bce_fp12_1x8_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xga_bce_fp12_1x8_eval_op)
- ☑ socket_xga_bce_fp12_1x8_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xga_bce_fp12_1x8_fir_lms)
- ☑ socket_xhd_bce_fp12_1x8_1x1_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xhd_bce_fp12_1x8_1x1_eval_op)
- ☑ socket_xhd_bce_fp12_1x8_1x1_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xhd_bce_fp12_1x8_1x1_fir_lms)
- ☑ socket_xhd_bce_fp12_1x8_eval_op (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xhd_bce_fp12_1x8_eval_op)
- ☑ socket_xhd_bce_fp12_1x8_fir_lms (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\socket_xhd_bce_fp12_1x8_fir_lms)
- ☑ standalone_bsp_0 (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\standalone_bsp_0)
- ☑ xilkernel_bsp_0 (C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT\xilkernel_bsp_0)

☑ Copy projects into workspace

Working sets

☐ Add project to working sets

Working sets:

Figure 13: Select Copy projects into workspace and finish the Import of projects.

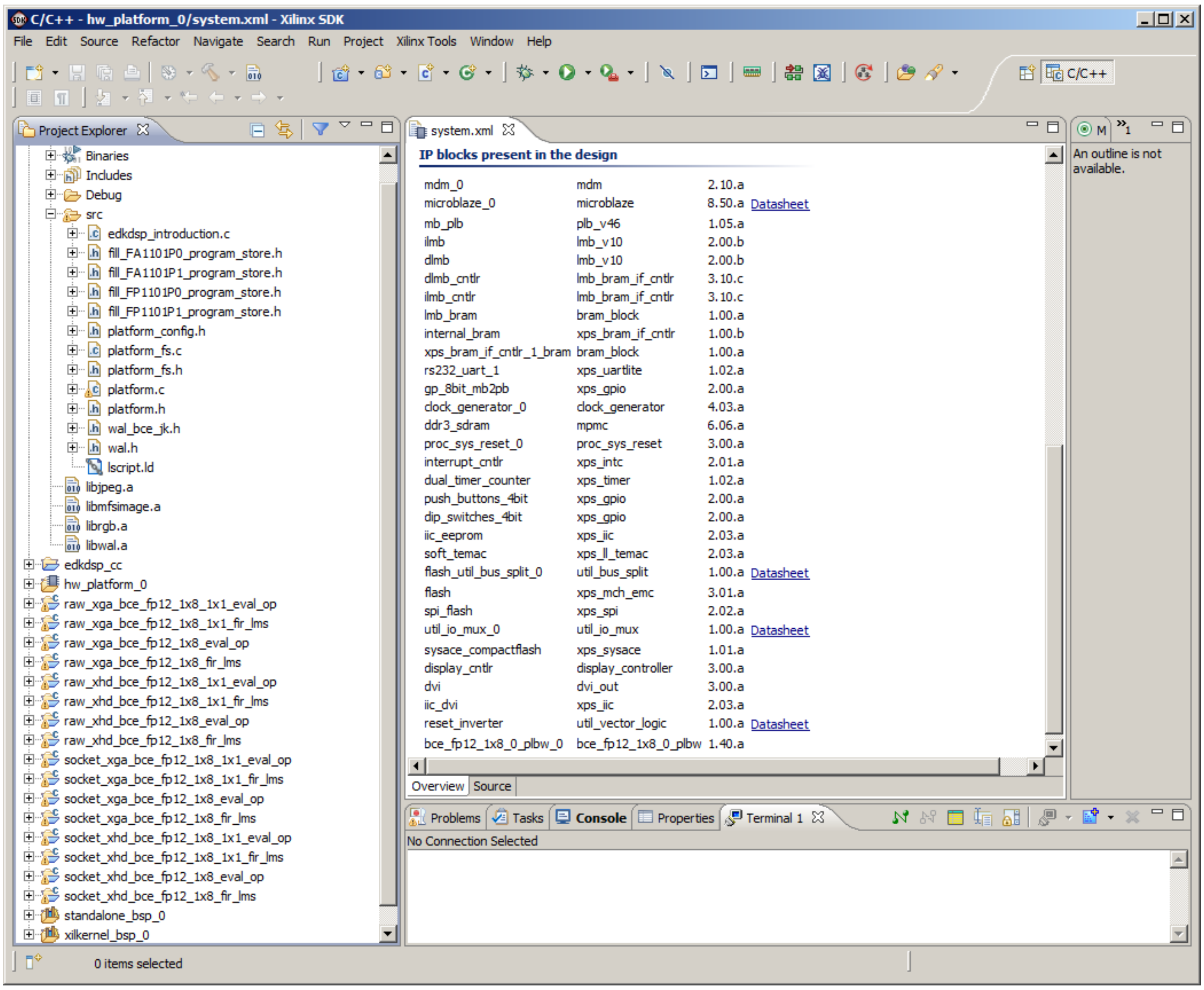
All the UTIA EdkDSP SW projects are imported into SDK workspace from the directory
C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8_IMPORT

Process of compilation will start automatically.

This first compilation of all SDK SW projects can take several minutes to finish.
It should finish without errors. See Figure 14.

3.2 Description of EdkDSP project

See content of the introductory "edkdsp" project in the "Project Explorer" window of the SDK. Inspect also the list of IP blocks and related driver versions present in the evaluation design. See Figure 14.


C/C++ - hw_platform_0/system.xml - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
C/C++
Project Explorer
system.xml
An outline is not available.
Overview Source
Problems Tasks Console Properties Terminal 1
No Connection Selected
0 items selected

Project Explorer:

```
Binaries
Includes
Debug
src
    edkdsp_introduction.c
    fill_FA1101P0_program_store.h
    fill_FA1101P1_program_store.h
    fill_FP1101P0_program_store.h
    fill_FP1101P1_program_store.h
    platform_config.h
    platform_fs.c
    platform_fs.h
    platform.c
    platform.h
    wal_bce_jk.h
    wal.h
    lscript.ld
libjpeg.a
libmfsimage.a
librgb.a
libwal.a
edkdsp_cc
hw_platform_0
raw_xga_bce_fp12_1x8_1x1_eval_op
raw_xga_bce_fp12_1x8_1x1_fir_lms
raw_xga_bce_fp12_1x8_eval_op
raw_xga_bce_fp12_1x8_fir_lms
raw_xhd_bce_fp12_1x8_1x1_eval_op
raw_xhd_bce_fp12_1x8_1x1_fir_lms
raw_xhd_bce_fp12_1x8_eval_op
raw_xhd_bce_fp12_1x8_fir_lms
socket_xga_bce_fp12_1x8_1x1_eval_op
socket_xga_bce_fp12_1x8_1x1_fir_lms
socket_xga_bce_fp12_1x8_eval_op
socket_xga_bce_fp12_1x8_fir_lms
socket_xhd_bce_fp12_1x8_1x1_eval_op
socket_xhd_bce_fp12_1x8_1x1_fir_lms
socket_xhd_bce_fp12_1x8_eval_op
socket_xhd_bce_fp12_1x8_fir_lms
standalone_bsp_0
xilkernel_bsp_0
```

IP blocks present in the design

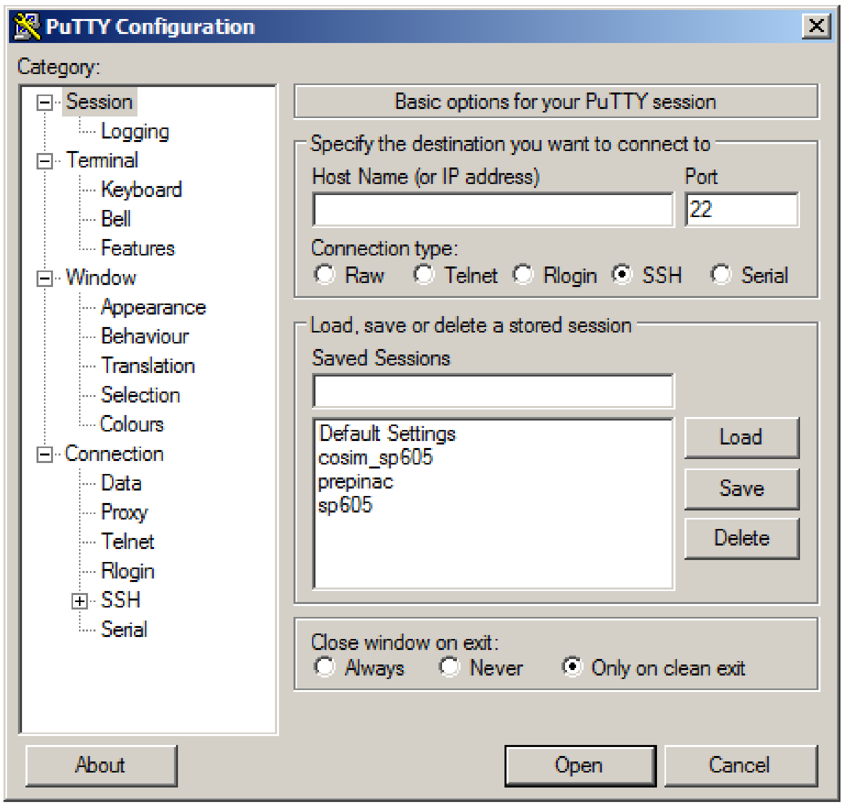
mdm_0	mdm	2.10.a	
microblaze_0	microblaze	8.50.a	Datasheet
mb_plb	plb_v46	1.05.a	
ilmb	lmb_v10	2.00.b	
dlmb	lmb_v10	2.00.b	
dlmb_cntlr	lmb_bram_if_cntlr	3.10.c	
ilmb_cntlr	lmb_bram_if_cntlr	3.10.c	
lmb_bram	bram_block	1.00.a	
internal_bram	xps_bram_if_cntlr	1.00.b	
xps_bram_if_cntlr_1_bram	bram_block	1.00.a	
rs232_uart_1	xps_uartlite	1.02.a	
gp_8bit_mb2pb	xps_gpio	2.00.a	
clock_generator_0	clock_generator	4.03.a	
ddr3_sdram	mpmc	6.06.a	
proc_sys_reset_0	proc_sys_reset	3.00.a	
interrupt_cntlr	xps_intc	2.01.a	
dual_timer_counter	xps_timer	1.02.a	
push_buttons_4bit	xps_gpio	2.00.a	
dip_switches_4bit	xps_gpio	2.00.a	
iic_eeprom	xps_iic	2.03.a	
soft_temac	xps_ll_temac	2.03.a	
flash_util_bus_split_0	util_bus_split	1.00.a	Datasheet
flash	xps_mch_emc	3.01.a	
spi_flash	xps_spi	2.02.a	
util_io_mux_0	util_io_mux	1.00.a	Datasheet
sysace_compactflash	xps_sysace	1.01.a	
display_cntlr	display_controller	3.00.a	
dvi	dvi_out	3.00.a	
iic_dvi	xps_iic	2.03.a	
reset_inverter	util_vector_logic	1.00.a	Datasheet
bce_fp12_1x8_0_plbw_0	bce_fp12_1x8_0_plbw	1.40.a	

Figure 14: All projects are compiled. Open the "edkdsp" introduction SW project.

The introductory "edkdsp" project is using only the serial link terminal and the Xilinx memory based file system. It is not using the LwIP Ethernet libraries and it is not using the video display.

Connect the jtag and serial line USB cables to your SP605 board and switch ON the board.

Start serial line terminal on your PC. See configuration of terminal PuTTY on Figure 15.



Specify the destination you want to connect to

Host Name (or IP address):

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

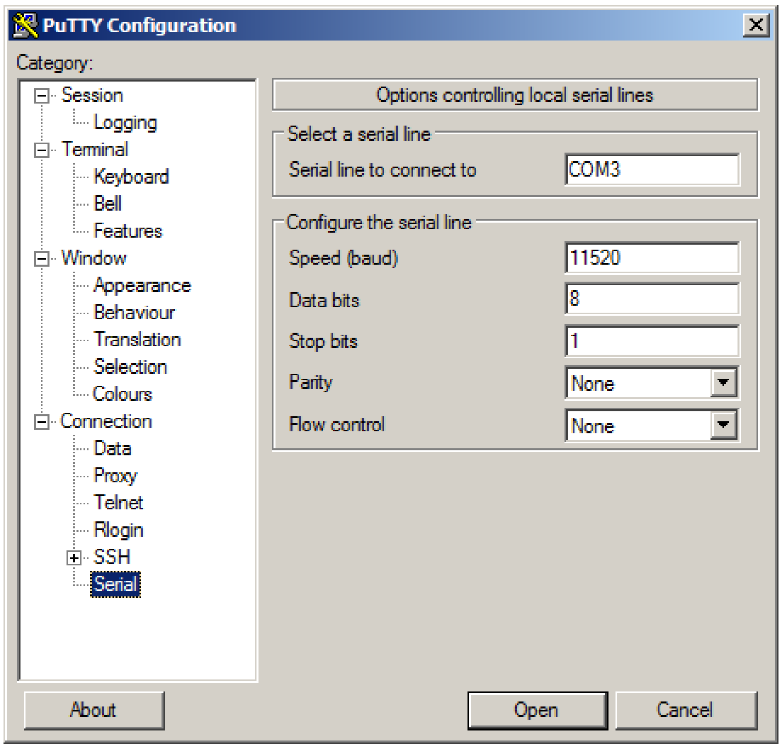
Saved Sessions

Default Settings
cosim_sp605
prepinac
sp605

Close window on exit: Always Never [✓ Only on clean exit]

Figure 15: Start the serial line terminal (PuTTY)

Select 115200 baud and “Flow control” to None. See Figure 16 and Figure 17.



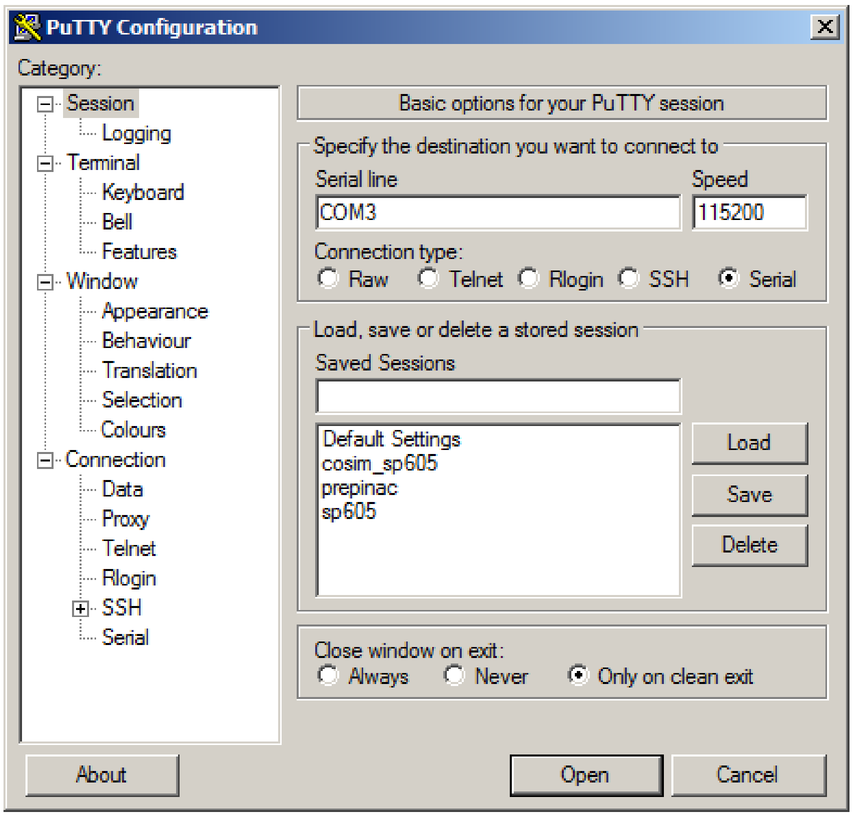
Select a serial line

Serial line to connect to	COM3

Configure the serial line

Speed (baud)	11520
Data bits	8
Stop bits	1
Parity	None
Flow control	None

Figure 16: Setup your COM port. Speed 115200 baud, and Flow control ”None”



Specify the destination you want to connect to

Serial line: COM3

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

Default Settings
cosim_sp605
prepinac
sp605

Close window on exit: Always Never [✓ Only on clean exit]

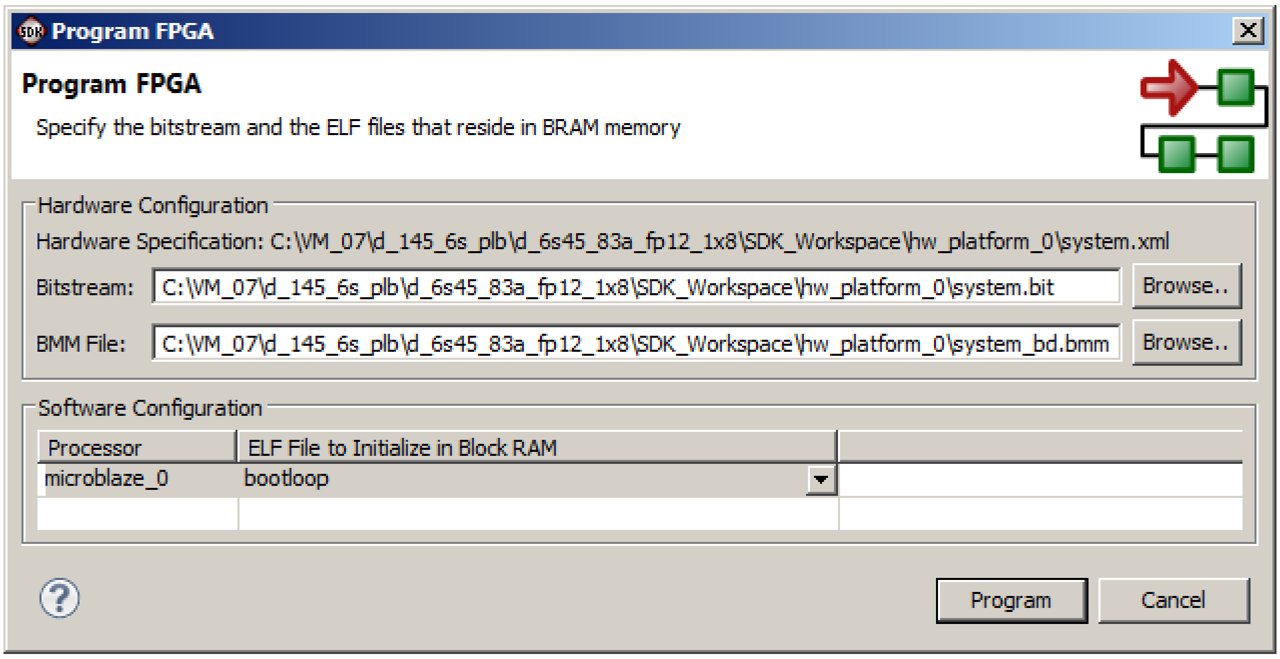
Figure 17: Select "Serial" Session and the terminal window will open.

In SDK, program the SP605 board by selecting:
Xilinx Tools -> Program FPGA

Type or browse to the "system.bit" file and the "system_bd.bmm" file:

C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform\system.bit
C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform\system_bd.bmm

Click on the "Program" button.


Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Specification: C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform_0\system.xml

Bitstream: C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform_0\system.bit

BMM File: C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform_0\system_bd.bmm

Software Configuration

Processor	ELF File to Initialize in Block RAM
microblaze_0	bootloop

Figure 18: Specify the .bit and the .bmm file with location the initial MB program.

The SP605 board is programmed with the .bit file now. The MicroBlaze is running in the initial bootloop.

The SW application from the edkdsp project has to be downloaded to the DDR3 memory, now.
The “edkdsp_introduction.c” application is performing these steps:

- Initialize cache and serial line controller.
- Initialize memory based file system from data linked in the .elf binary file.
- Initialize data structures describing the edk_fp12_1x8_v1_40_a accelerator.
- Write firmware to both program memories of the accelerator.
- Read properties and the license status of the accelerator.
- MicroBlaze application can identify which member of the accelerator family is actually present (10|20|30|40) and if the floating point pipelined divider data path is present or not.
- Test floating point vector addition computed in the accelerator with 8xSIMD data path.
    - Communicate from the accelerator data about input and led I/O to MicroBlaze.
    - Write these data to an ascii text file in the ram based file system and to the terminal.
- Verify the floating point results in MicroBlaze.
- Print to the terminal the number of free and used data blocks.
- Print list of top level directories present in the file system.

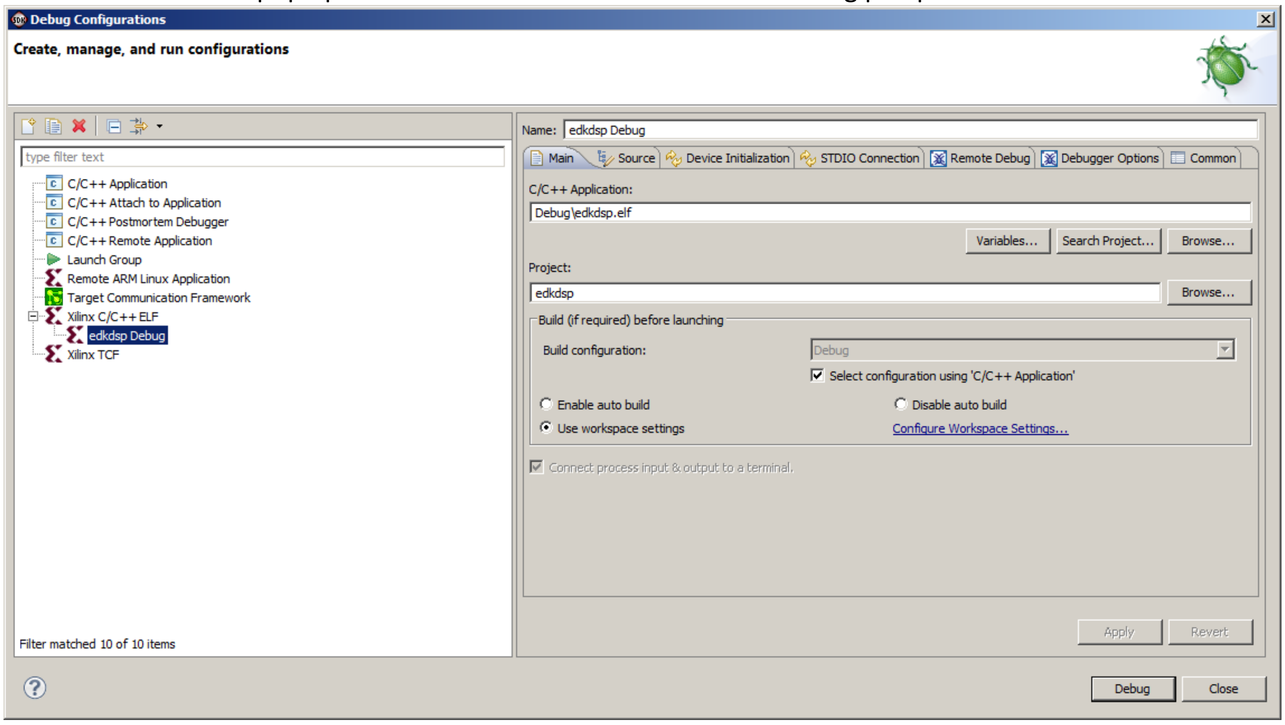
Select the “edkdsp” project by clicking on it in the SDK Project Explorer Window.

In SDK, select:
Run -> Debug Configuration ->Xilinc C/C++ ELF

Click on the “New launch configuration” in the Run configuration screen and the “edkdsp” project executable Debug\edkdsp.elf Is ready for download to DDR3 on the SP605 board via the jtag cable.

Click on “Debug” button to download the executable and start the debugger. See Figure 19.

Click on “Yes” in the pop up window related to the switch to the Debug perspective.


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Launch Group
- Remote ARM Linux Application
- Target Communication Framework
- Xilinx C/C++ ELF
  - edkdsp Debug
- Xilinx TCF

Filter matched 10 of 10 items

Name: edkdsp Debug

Main | Source | Device Initialization | STDIO Connection | Remote Debug | Debugger Options | Common

C/C++ Application: Debug\edkdsp.elf

Project: edkdsp

Build (if required) before launching

Build configuration: Debug

☑ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

☑ Connect process input & output to a terminal.

Figure 19: Debug “edkdsp.elf” code on MicroBlaze

The Debug\edkdsp.elf is executed up to the first breakpoint. See Figure 20.

Select next breakpoint as indicated in Figure 21 and click on Resume (F8) icon to run up to this new breakpoint located inside of the triple loop on the call to the function

```
bce_fp12_1x8_VADD_introduction(worker1, fw, capabilities1, ah, bh, zh);
```

See the terminal indicating properties of the accelerator.

Click on the on Resume (F8) icon to execute the first call of the test function with parameters ah=0, bh=0, zh=0. See the console output.

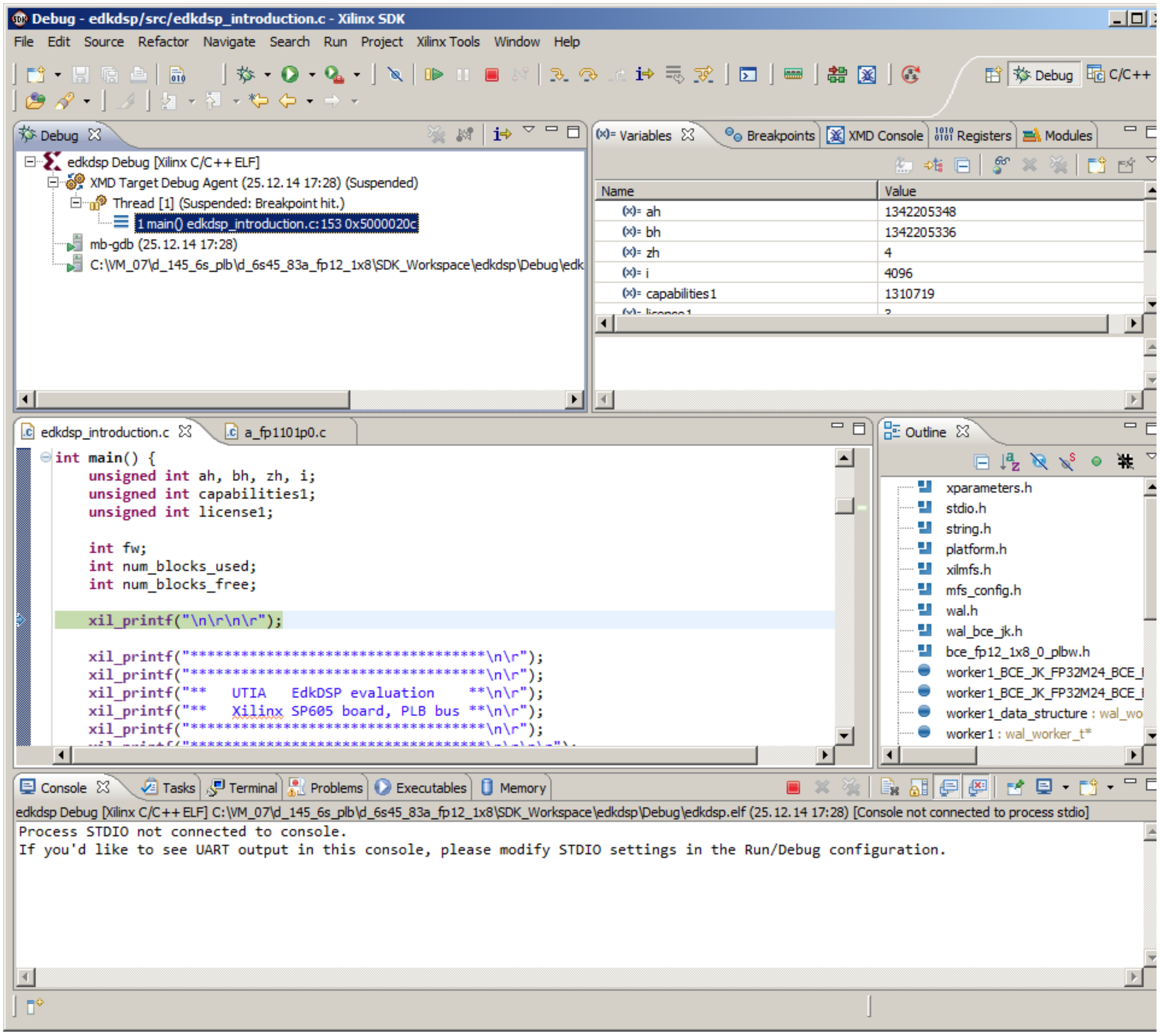
Click again the on Resume (F8) icon to execute the first call of the test function with parameters ah=0, bh=0, zh=1. See the console output as indicated in the left part of Figure 23.

Remove the breakpoint and click on the on Resume (F8) icon to execute the program up to the end.
The debugger stops at the end of the program with screen presented in Figure 22.

The terminal output is presented in the right part of Figure 23. It indicates results of verification of floating point computation up to the final combination of parameters (ah=3 bh=3 zh=3) compared to the floating point results in MicroBlaze. OK corresponds to identical results.

Terminal finally displays number of free and used data blocks in the RAM based file system of MicroBlaze and top level directories present in the file system. See the right part of Figure 23.

Terminate the debugger and close the debug perspective and return to the default SDK C/C++ perspective.



```
edkdsp Debug [Xilinx C/C++ ELF]
  XMD Target Debug Agent (25.12.14 17:28) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() edkdsp_introduction.c:153 0x5000020c
  mb-gdb (25.12.14 17:28)
  C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp\Debug\edk
```

Name	Value
ah	1342205348
bh	1342205336
zh	4
i	4096
capabilities1	1310719

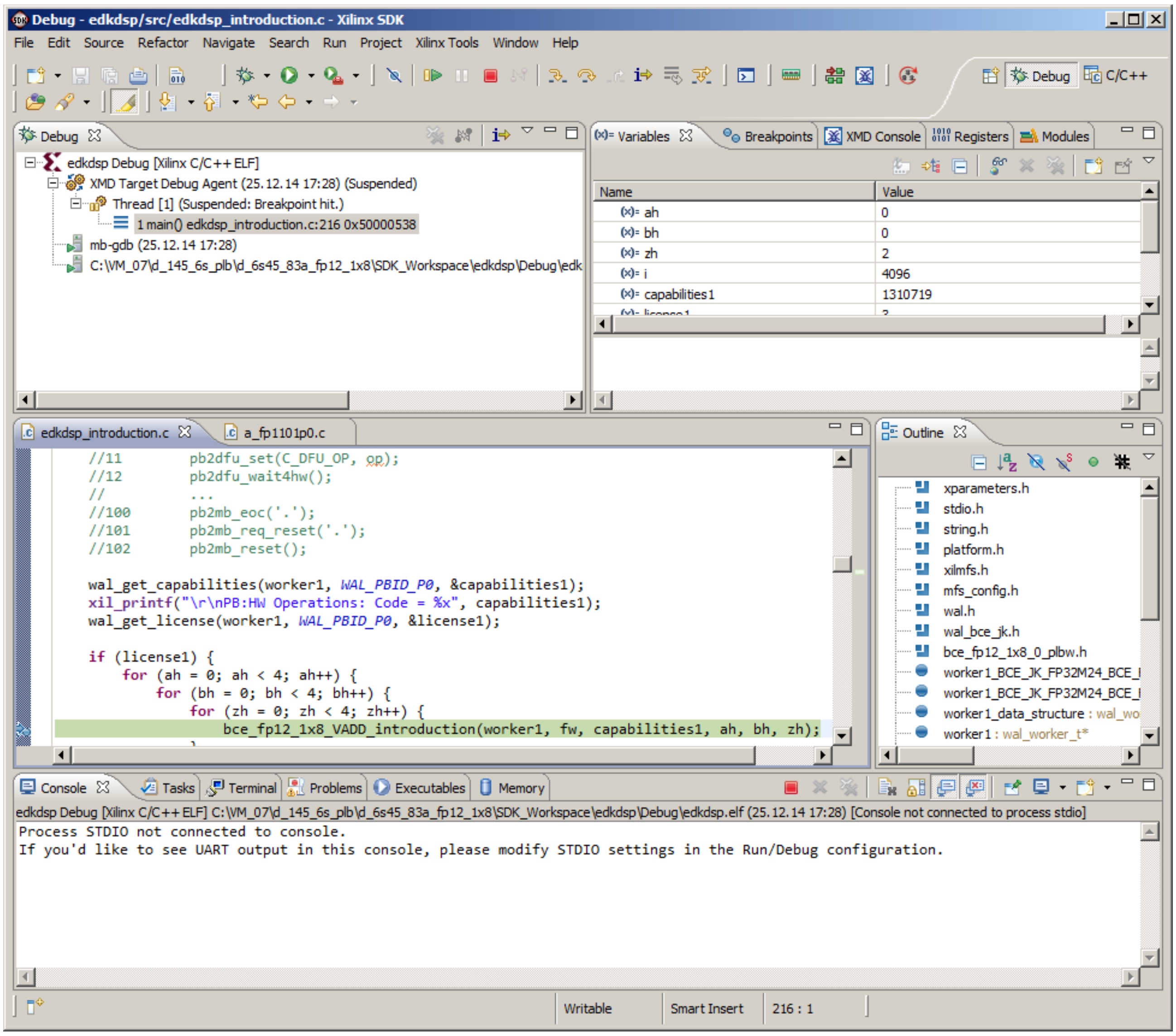
```c
int main() {
    unsigned int ah, bh, zh, i;
    unsigned int capabilities1;
    unsigned int license1;

    int fw;
    int num_blocks_used;
    int num_blocks_free;

    xil_printf("\n\r\n\r");

    xil_printf("************************************\n\r");
    xil_printf("************************************\n\r");
    xil_printf("**   UTIA   EdkDSP evaluation    **\n\r");
    xil_printf("**   Xilinx SP605 board, PLB bus **\n\r");
    xil_printf("************************************\n\r");
```

```
xparameters.h
stdio.h
string.h
platform.h
xilmfs.h
mfs_config.h
wal.h
wal_bce_jk.h
bce_fp12_1x8_0_plbw.h
worker1_BCE_JK_FP32M24_BCE_I
worker1_BCE_JK_FP32M24_BCE_I
worker1_data_structure : wal_wo
worker1 : wal_worker_t*
```

```
edkdsp Debug [Xilinx C/C++ ELF] C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp\Debug\edkdsp.elf (25.12.14 17:28) [Console not connected to process stdio]
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDIO settings in the Run/Debug configuration.
```

Figure 20: The debugger stops on the first executable line of MicroBlaze C code .

The Debug\edkdsp.elf is running until you click on the “Remove launch” (X) in the SDK. See Figure 20.


Debug - edkdsp/src/edkdsp_introduction.c - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
Debug C/C++
Debug
Variables Breakpoints XMD Console Registers Modules
edkdsp_introduction.c a_fp1101p0.c
Outline
Console Tasks Terminal Problems Executables Memory
Writable Smart Insert 216 : 1

```
edkdsp Debug [Xilinx C/C++ ELF]
  XMD Target Debug Agent (25.12.14 17:28) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() edkdsp_introduction.c:216 0x50000538
  mb-gdb (25.12.14 17:28)
  C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp\Debug\edk
```

Name	Value
ah	0
bh	0
zh	2
i	4096
capabilities1	1310719

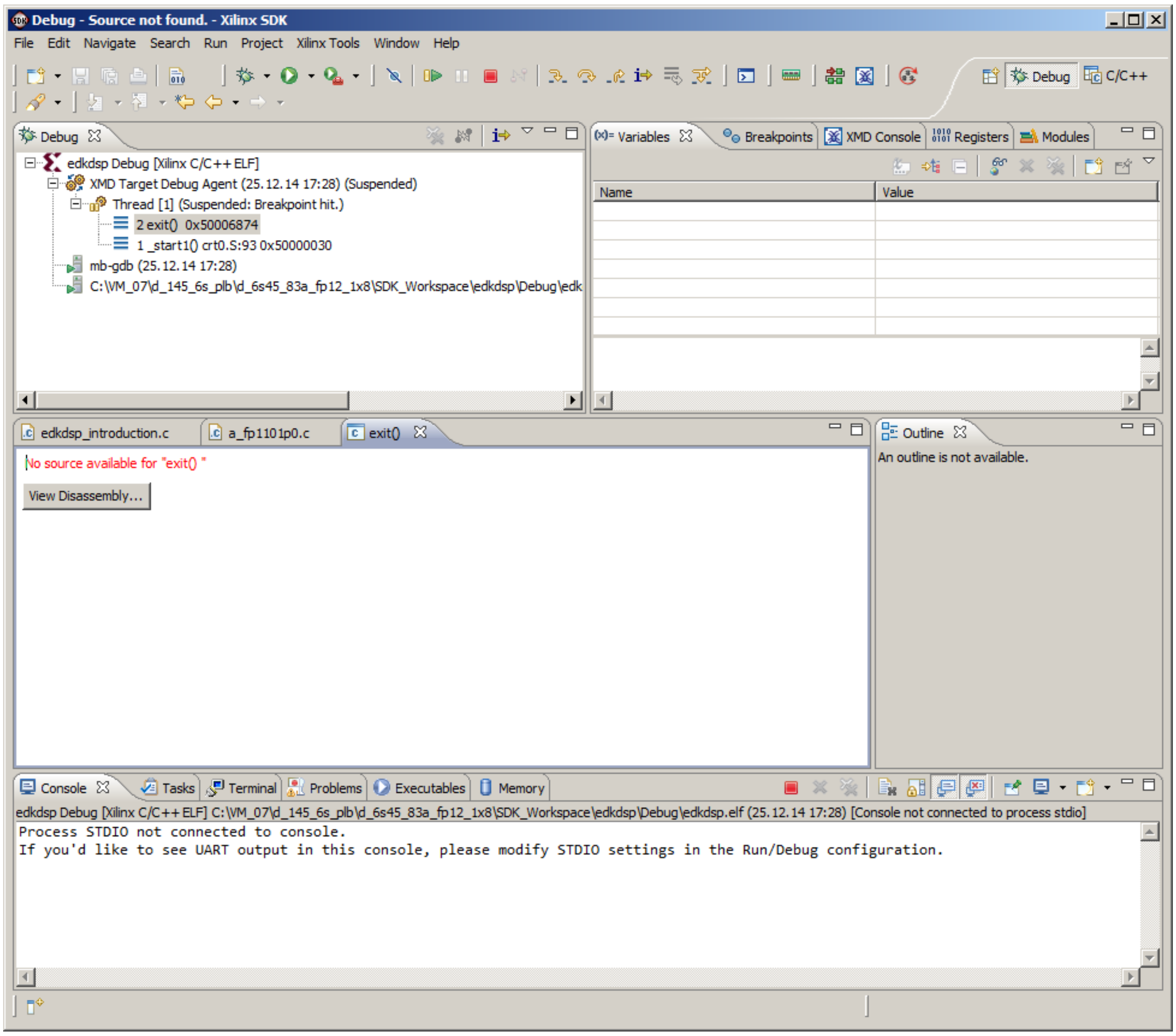
```
    //11        pb2dfu_set(C_DFU_OP, op);
    //12        pb2dfu_wait4hw();
    //          ...
    //100       pb2mb_eoc('.');
    //101       pb2mb_req_reset('.');
    //102       pb2mb_reset();

    wal_get_capabilities(worker1, WAL_PBID_P0, &capabilities1);
    xil_printf("\r\nPB:HW Operations: Code = %x", capabilities1);
    wal_get_license(worker1, WAL_PBID_P0, &license1);

    if (license1) {
        for (ah = 0; ah < 4; ah++) {
            for (bh = 0; bh < 4; bh++) {
                for (zh = 0; zh < 4; zh++) {
                    bce_fp12_1x8_VADD_introduction(worker1, fw, capabilities1, ah, bh, zh);
```

```
xparameters.h
stdio.h
string.h
platform.h
xilmfs.h
mfs_config.h
wal.h
wal_bce_jk.h
bce_fp12_1x8_0_plbw.h
worker1_BCE_JK_FP32M24_BCE_I
worker1_BCE_JK_FP32M24_BCE_I
worker1_data_structure : wal_wo
worker1 : wal_worker_t*
```

```
edkdsp Debug [Xilinx C/C++ ELF] C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp\Debug\edkdsp.elf (25.12.14 17:28) [Console not connected to process stdio]
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDIO settings in the Run/Debug configuration.
```

Figure 21: Set the breakpoint in function testing the vector ADD operation.

ttt



```
edkdsp Debug [Xilinx C/C++ ELF]
  XMD Target Debug Agent (25.12.14 17:28) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      2 exit() 0x50006874
      1 _start1() crt0.S:93 0x50000030
  mb-gdb (25.12.14 17:28)
  C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp\Debug\edk
```

No source available for "exit()"

An outline is not available.

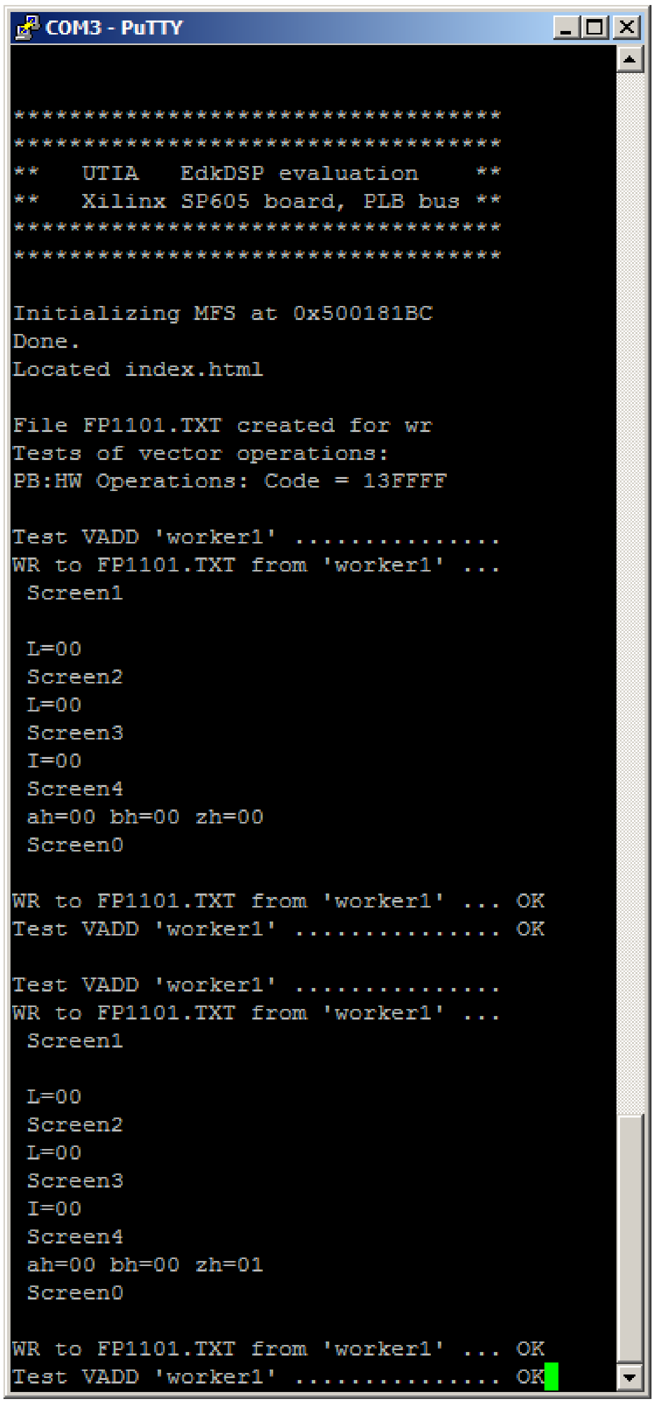
```
edkdsp Debug [Xilinx C/C++ ELF] C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp\Debug\edkdsp.elf (25.12.14 17:28) [Console not connected to process stdio]
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDIO settings in the Run/Debug configuration.
```

Figure 22: The application "edkdsp" is running.

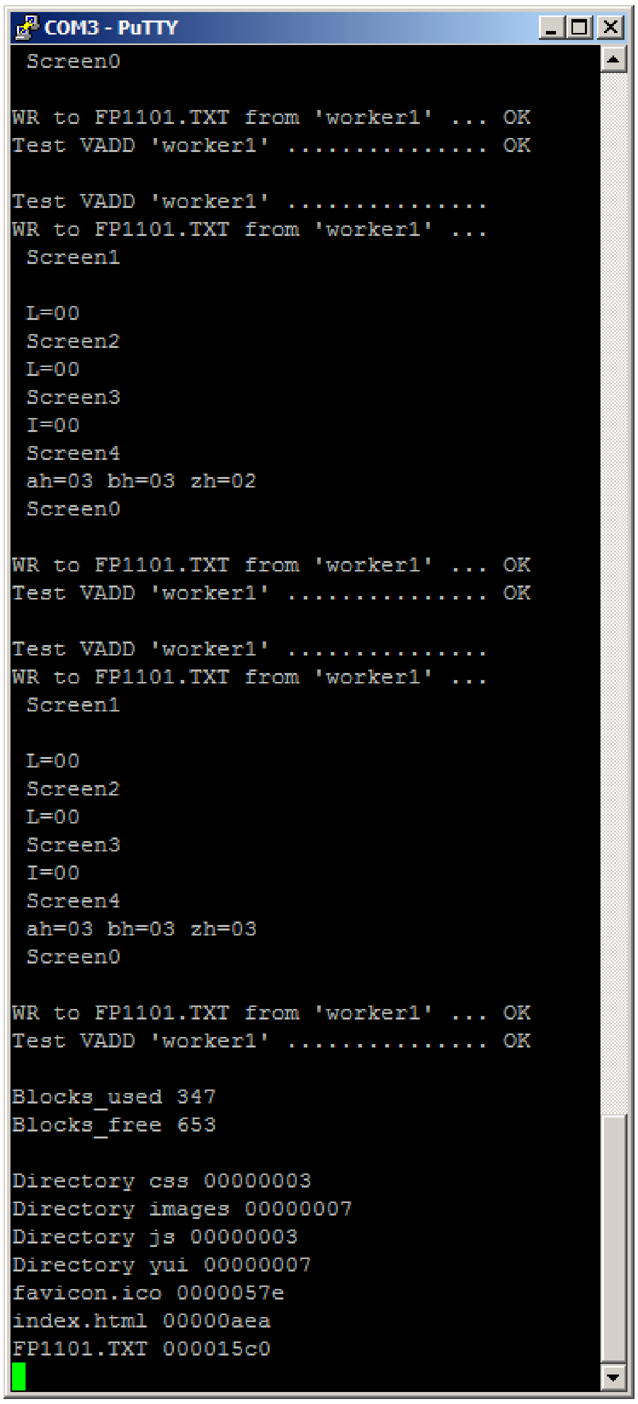
Please analyse the terminal listing on Figure 23 and compare it with the source code "edkdsp_introdiction.c" in the SDK project "edkdsp" together with the PicoBlaze6 controller firmware edkdsp_cc/a/a_fp1101p0.c in the accelerator. See Figure 24 and Figure 25.

It is demonstrating, how the MicroBlaze processor communicates with the PicoBlaze6 controller present in the accelerator. The MicroBlaze C WAL library functions support the exchange of floating point data in DDR3 (C variables and arrays) to 3x8=24 4kB memories A1, B1, Z1 … Z8, B8, Z8 of the 8xSIMD accelerator. Each memory is organised as 4 banks of 256 32bit words.

The PicoBlaze6 controller is controlling the 8xSIMD data flow unit. The data flow unit has HW supported access to these memories from the accelerator.



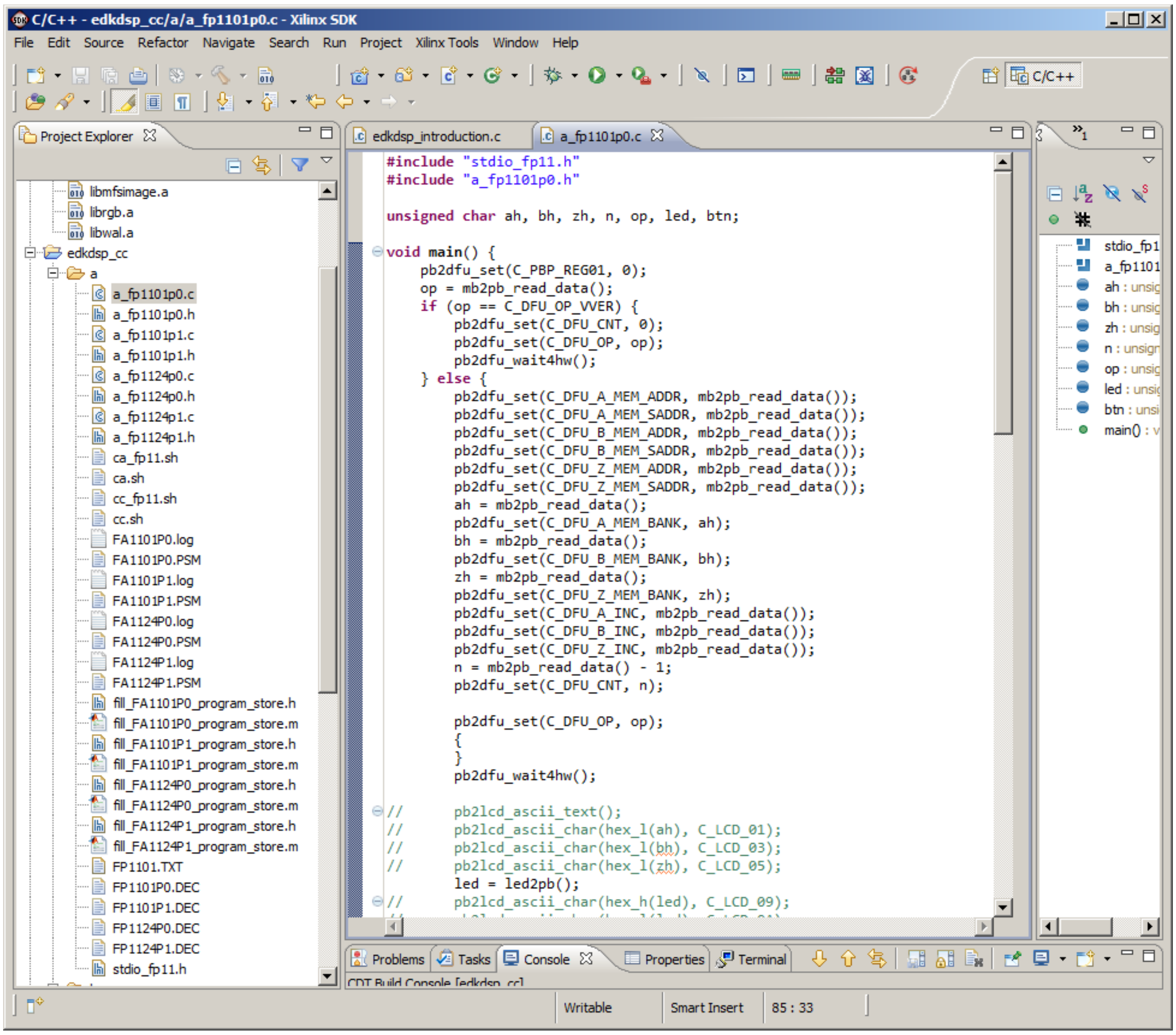
```
************************************
************************************
**   UTIA   EdkDSP evaluation    **
**   Xilinx SP605 board, PLB bus **
************************************
************************************

Initializing MFS at 0x500181BC
Done.
Located index.html

File FP1101.TXT created for wr
Tests of vector operations:
PB:HW Operations: Code = 13FFFF

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 I=00
 Screen4
 ah=00 bh=00 zh=00
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 I=00
 Screen4
 ah=00 bh=00 zh=01
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK
```



```
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 I=00
 Screen4
 ah=03 bh=03 zh=02
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 I=00
 Screen4
 ah=03 bh=03 zh=03
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Blocks_used 347
Blocks_free 653

Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1101.TXT 000015c0
```

Figure 23: Terminal output from “edkdsp”initial part left, final part right.

The listing displayed on Figure 23 indicates execution of program “edkdsp_introduction.c” cooperating with the PicoBlaze6 firmware program edkdsp_cc/a/fp1101p0.c.



```c
#include "stdio_fp11.h"
#include "a_fp1101p0.h"

unsigned char ah, bh, zh, n, op, led, btn;

void main() {
    pb2dfu_set(C_PBP_REG01, 0);
    op = mb2pb_read_data();
    if (op == C_DFU_OP_VVER) {
        pb2dfu_set(C_DFU_CNT, 0);
        pb2dfu_set(C_DFU_OP, op);
        pb2dfu_wait4hw();
    } else {
        pb2dfu_set(C_DFU_A_MEM_ADDR, mb2pb_read_data());
        pb2dfu_set(C_DFU_A_MEM_SADDR, mb2pb_read_data());
        pb2dfu_set(C_DFU_B_MEM_ADDR, mb2pb_read_data());
        pb2dfu_set(C_DFU_B_MEM_SADDR, mb2pb_read_data());
        pb2dfu_set(C_DFU_Z_MEM_ADDR, mb2pb_read_data());
        pb2dfu_set(C_DFU_Z_MEM_SADDR, mb2pb_read_data());
        ah = mb2pb_read_data();
        pb2dfu_set(C_DFU_A_MEM_BANK, ah);
        bh = mb2pb_read_data();
        pb2dfu_set(C_DFU_B_MEM_BANK, bh);
        zh = mb2pb_read_data();
        pb2dfu_set(C_DFU_Z_MEM_BANK, zh);
        pb2dfu_set(C_DFU_A_INC, mb2pb_read_data());
        pb2dfu_set(C_DFU_B_INC, mb2pb_read_data());
        pb2dfu_set(C_DFU_Z_INC, mb2pb_read_data());
        n = mb2pb_read_data() - 1;
        pb2dfu_set(C_DFU_CNT, n);

        pb2dfu_set(C_DFU_OP, op);
        {
        }
        pb2dfu_wait4hw();

//      pb2lcd_ascii_text();
//      pb2lcd_ascii_char(hex_l(ah), C_LCD_01);
//      pb2lcd_ascii_char(hex_l(bh), C_LCD_03);
//      pb2lcd_ascii_char(hex_l(zh), C_LCD_05);
        led = led2pb();
//      pb2lcd_ascii_char(hex_h(led), C_LCD_09);
```

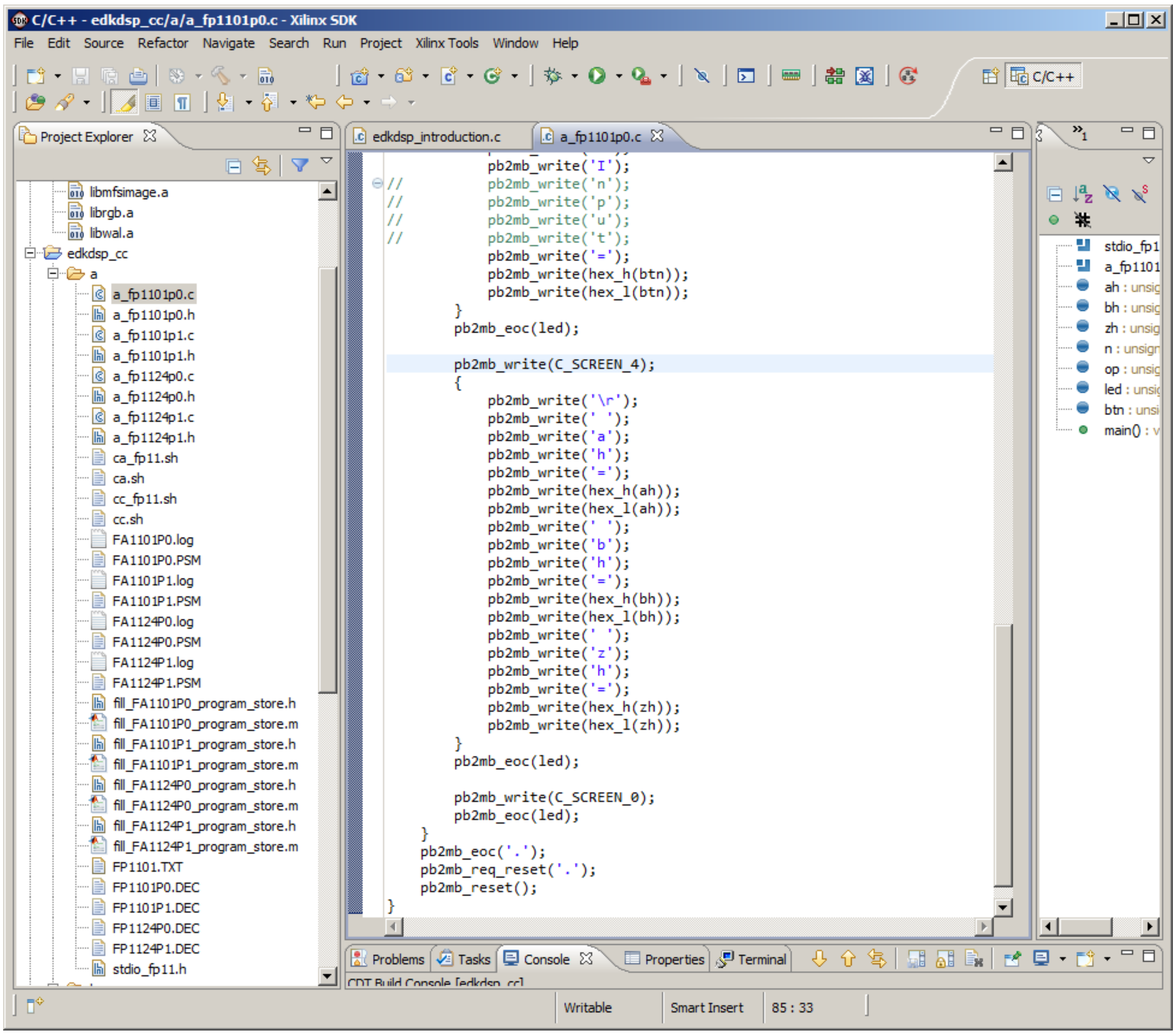
Figure 24: Initial section of the PicoBlaze6 controller firmware edkdsp_cc/a/fp1101p0.c.

Figure 24 presents the initial sequence of the PicoBlaz6 firmware program edkdsp_cc/a/a_fp1101p0.c.
It is using the EdkDSP API functions to communicate properties supported by the accelerator and parameters of tested vector floating point function under test.

The function pb2dfu_set(C_DFU_OP,op) starts the vector floating point operation of the 8xSIMD data flow unit of the accelerator. See Figure 24.

PicoBlaze6 C code is synchronised with the result of the vector floating point operation of the 8xSIMD data flow unit in function call function pb2dfu wait4hw(). See Figure 24.

Figure 25 presents the final section of the PicoBlaz6 firmware program edkdsp_cc/a/a_fp1101p0.c .
PicoBlaze6 C code is printing support information to several screens. It is served by the MicroBlaze C code. MicroBlaze prints received data to the console and to the ascii file FP1101.TXT in the RAM based file system of MicroBlaze processor.



```
            pb2mb_write('I');
//          pb2mb_write('n');
//          pb2mb_write('p');
//          pb2mb_write('u');
//          pb2mb_write('t');
            pb2mb_write('=');
            pb2mb_write(hex_h(btn));
            pb2mb_write(hex_l(btn));
        }
        pb2mb_eoc(led);

        pb2mb_write(C_SCREEN_4);
        {
            pb2mb_write('\r');
            pb2mb_write(' ');
            pb2mb_write('a');
            pb2mb_write('h');
            pb2mb_write('=');
            pb2mb_write(hex_h(ah));
            pb2mb_write(hex_l(ah));
            pb2mb_write(' ');
            pb2mb_write('b');
            pb2mb_write('h');
            pb2mb_write('=');
            pb2mb_write(hex_h(bh));
            pb2mb_write(hex_l(bh));
            pb2mb_write(' ');
            pb2mb_write('z');
            pb2mb_write('h');
            pb2mb_write('=');
            pb2mb_write(hex_h(zh));
            pb2mb_write(hex_l(zh));
        }
        pb2mb_eoc(led);

        pb2mb_write(C_SCREEN_0);
        pb2mb_eoc(led);
    }
    pb2mb_eoc('.');
    pb2mb_req_reset('.');
    pb2mb_reset();
}
```

Figure 25: Final section of the PicoBlaze6 controller firmware edkdsp_cc/a/fp1101p0.c.

3.3 Evaluation of EdkDSP C compiler

This section is describing the use of the EdkDSP C compiler to recompile the firmware for the PicoBlaze6 controller of EdkDSP accelerators.

The evaluation package includes also precompiled files with the firmware ready for download from PC to the SP605 board. These files can be used if the EdkDSP C compiler is not installed on your PC.

The UTIA EdkDSP C compiler is implemented as Ubuntu binary utility.
An “VMware player” software and a compatible Ubuntu image version is needed to run the UTIA EdkDSP C compiler on Windows 7 (64bit or 32bit) PC.

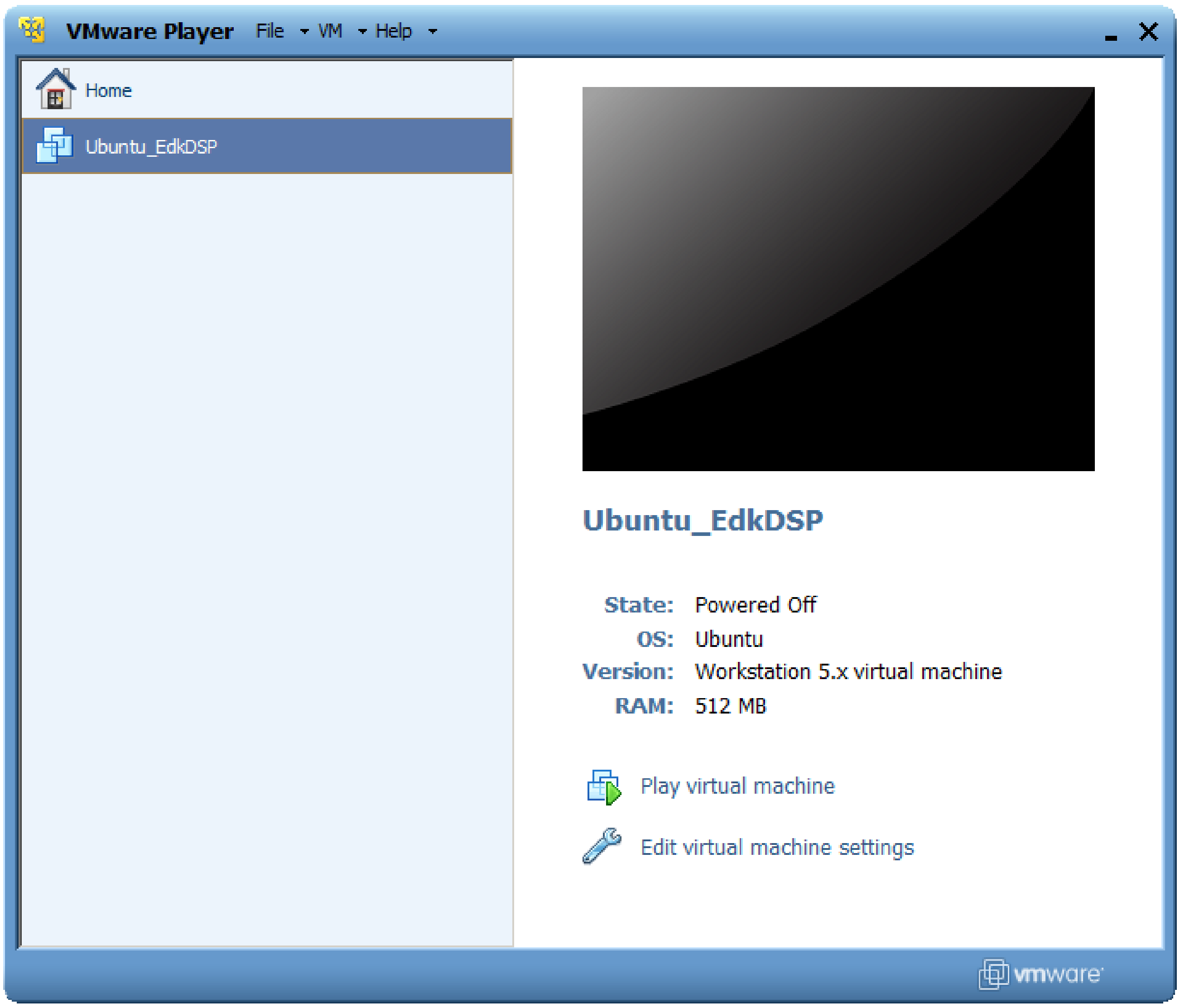
The Ubuntu image used in UTIA needs two DVD disks (8GB). That is why it is not included as part of the evaluation package.

If you would need this image, write an email request to kadlec@utia.cas.cz to get these two DVD with correct Ubuntu image from UTIA (free of charge) by standard mail.

Install from the Internet the VMware Player software (64bit or 32bit) on your PC.

Open the VMware Player and select the "Ubuntu_EdkDSP" image.
The Ubuntu will start.
Login as:

User: devel
Pswd: devuser



Ubuntu_EdkDSP

State: Powered Off
OS: Ubuntu
Version: Workstation 5.x virtual machine
RAM: 512 MB

Figure 26: Select the Ubuntu_EdkDSP image in the VMware Player and click "Play".

The VMware Player creates virtual Ethernet connections to the PC.
The PC directory c:\VM_07 needs to be shared by Windows 7 and the Ubuntu applications by Samba.

In Windows 7, set the directory c:\VM_07 as the shared directory.
In Ubuntu, open terminal and mount the PC directory c:\VM_07 to Ubuntu by Samba.

This has been done by the script samba_07.sh The script is asking for the Ubuntu user password and your Password in Windows 7.

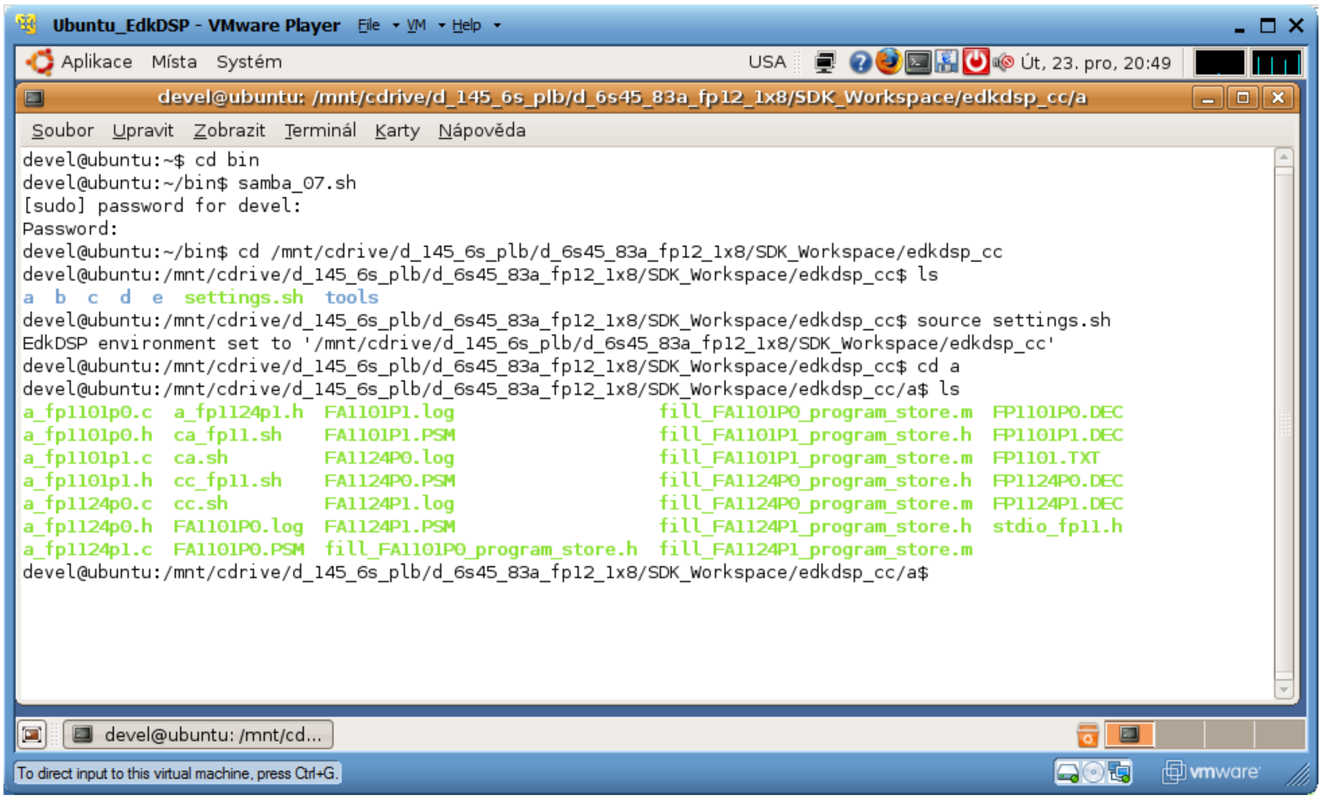
The Windows 7 c:/VM_07 directory is mounted to the Ubuntu OS as:
/mnt/cdrive

In Ubuntu, change directory to:
/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc

The EdkDSP C compiler utilities have to be on the Ubuntu PATH. This is done by sourcing the settings.sh script in this directory. Type:
source settings.sh

The EdkDSP C compiler utilities located in the directory
/mnt/cdrive/ d_145_6s_plb /e_6s45_83a_fp12_1x8_FW/tools
are on the PATH path. See Figure 32.

In Ubuntu, change directory to the example directory : ./a


Ubuntu_EdkDSP - VMware Player File VM Help
Aplikace Místa Systém
USA
Út, 23. pro, 20:49
devel@ubuntu: /mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a
Soubor Upravit Zobrazit Terminál Karty Nápověda
devel@ubuntu: /mnt/cd...
To direct input to this virtual machine, press Ctrl+G.
vmware

```
devel@ubuntu:~$ cd bin
devel@ubuntu:~/bin$ samba_07.sh
[sudo] password for devel:
Password:
devel@ubuntu:~/bin$ cd /mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc$ ls
a  b  c  d  e  settings.sh  tools
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc$ source settings.sh
EdkDSP environment set to '/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc'
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc$ cd a
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ ls
a_fp1101p0.c  a_fp1124p1.h  FA1101P1.log                   fill_FA1101P0_program_store.m  FP1101P0.DEC
a_fp1101p0.h  ca_fp11.sh    FA1101P1.PSM                   fill_FA1101P1_program_store.h  FP1101P1.DEC
a_fp1101p1.c  ca.sh         FA1124P0.log                   fill_FA1101P1_program_store.m  FP1101.TXT
a_fp1101p1.h  cc_fp11.sh    FA1124P0.PSM                   fill_FA1124P0_program_store.h  FP1124P0.DEC
a_fp1124p0.c  cc.sh         FA1124P1.log                   fill_FA1124P0_program_store.m  FP1124P1.DEC
a_fp1124p0.h  FA1101P0.log  FA1124P1.PSM                   fill_FA1124P1_program_store.h  stdio_fp11.h
a_fp1124p1.c  FA1101P0.PSM  fill_FA1101P0_program_store.h  fill_FA1124P1_program_store.m
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$
```

Figure 27: Initial setup of shared directory with Windows 7 and Sourcing of PATH

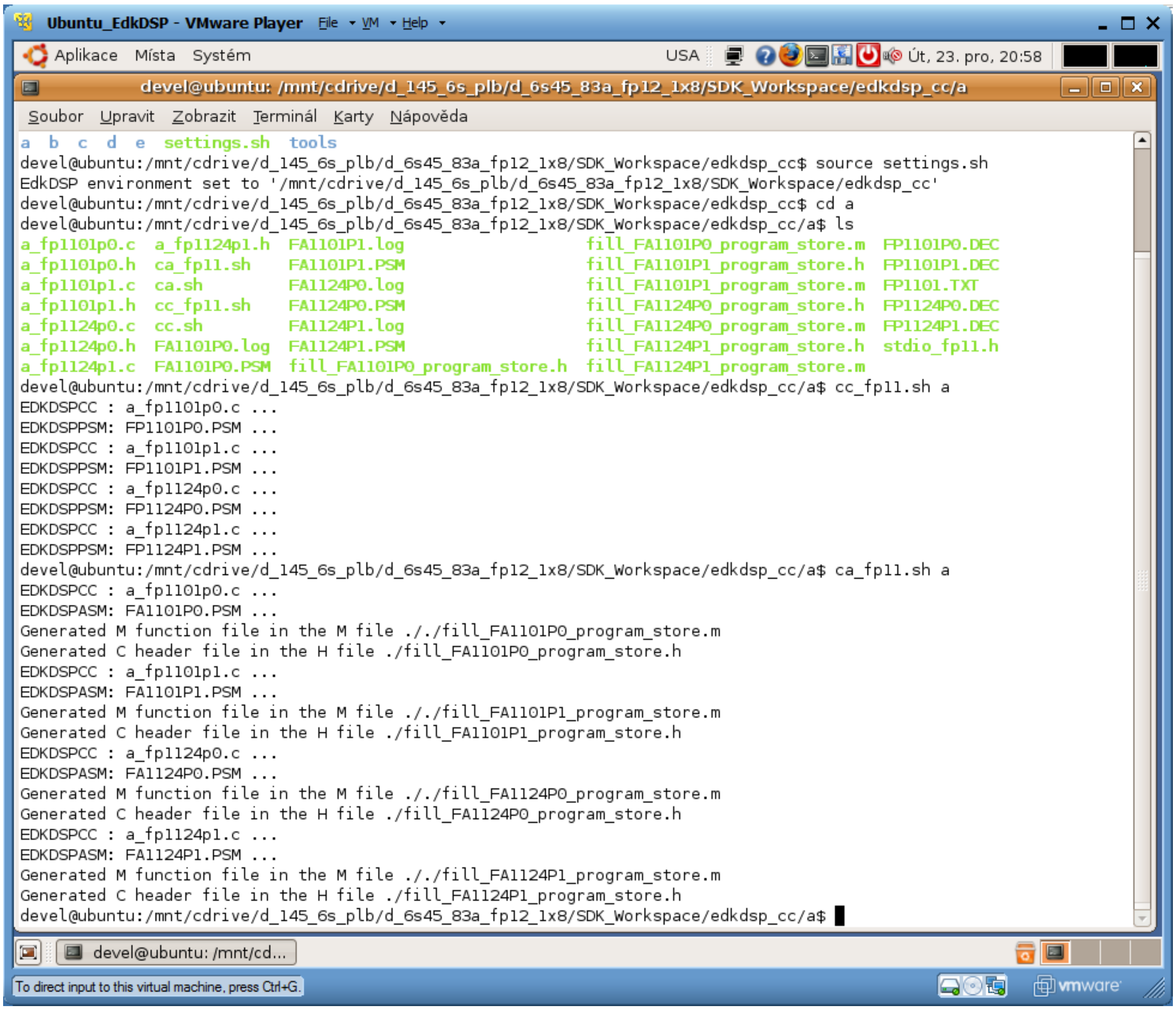
C examples can be compiled by script script cc_fp11.sh with parameter a. Type:
cc_fp11.sh a

This script will recompile and assemble all four C programs present in this directory into .DEC files, ready for download by the Windows 7 TFTP client to the ram-based file system of the SP605 board. See Figure 33.

C examples can be also compiled by the script ca_fp11.sh with parameter a. Type:
ca_fp11.sh a

This script will recompile and assemble all four C programs into .h C header files and .m Matlab scripts. The header files can be used to modify the default PicoBlaze6 firmware headers (included in the MicroBlaze source code) without need of the TFTP download of .DEC files via Ethernet. See also Figure 33.

Example of use of header files: In SDK, you can copy and paste the generated fill_FA1101P0_program_store.h and fill_FA1101P0_program._store.h files into the SDK edkdsp project directory edkdsp/src, recompile the edkdsp project and test it on board with new firmware, without the TFTP support.

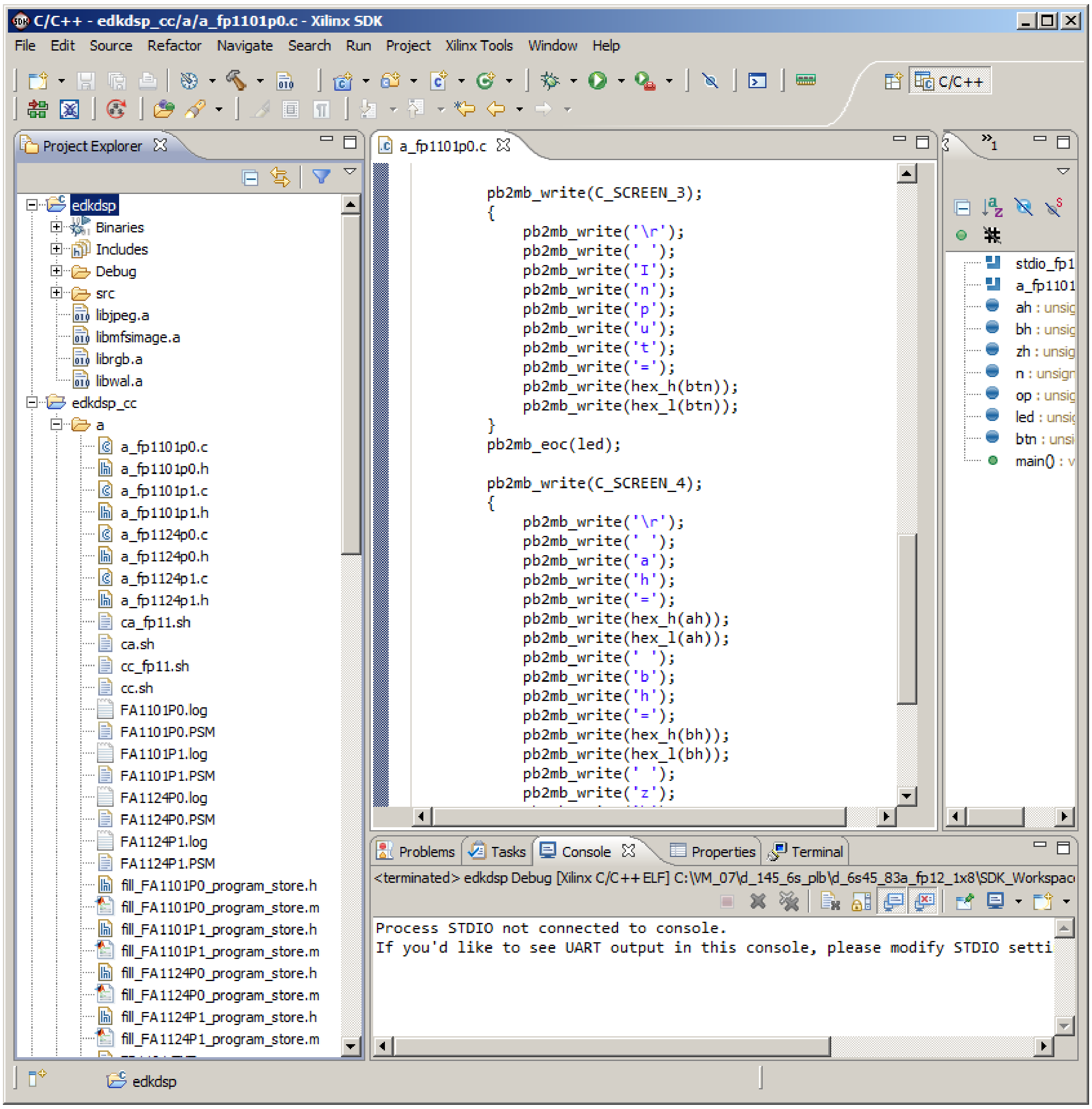

Ubuntu_EdkDSP - VMware Player
File
VM
Help
Aplikace
Místa
Systém
USA
Út, 23. pro, 20:58
devel@ubuntu: /mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a
Soubor
Upravit
Zobrazit
Terminál
Karty
Nápověda
devel@ubuntu: /mnt/cd...
To direct input to this virtual machine, press Ctrl+G.
vmware

```
a  b  c  d  e  settings.sh  tools
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc$ source settings.sh
EdkDSP environment set to '/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc'
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc$ cd a
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ ls
a_fp1101p0.c  a_fp1124p1.h  FA1101P1.log                   fill_FA1101P0_program_store.m  FP1101P0.DEC
a_fp1101p0.h  ca_fp11.sh    FA1101P1.PSM                   fill_FA1101P1_program_store.h  FP1101P1.DEC
a_fp1101p1.c  ca.sh         FA1124P0.log                   fill_FA1101P1_program_store.m  FP1101.TXT
a_fp1101p1.h  cc_fp11.sh    FA1124P0.PSM                   fill_FA1124P0_program_store.h  FP1124P0.DEC
a_fp1124p0.c  cc.sh         FA1124P1.log                   fill_FA1124P0_program_store.m  FP1124P1.DEC
a_fp1124p0.h  FA1101P0.log  FA1124P1.PSM                   fill_FA1124P1_program_store.h  stdio_fp11.h
a_fp1124p1.c  FA1101P0.PSM  fill_FA1101P0_program_store.h  fill_FA1124P1_program_store.m
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ cc_fp11.sh a
EDKDSPCC : a_fp1101p0.c ...
EDKDSPPSM: FP1101P0.PSM ...
EDKDSPCC : a_fp1101p1.c ...
EDKDSPPSM: FP1101P1.PSM ...
EDKDSPCC : a_fp1124p0.c ...
EDKDSPPSM: FP1124P0.PSM ...
EDKDSPCC : a_fp1124p1.c ...
EDKDSPPSM: FP1124P1.PSM ...
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ ca_fp11.sh a
EDKDSPCC : a_fp1101p0.c ...
EDKDSPASM: FA1101P0.PSM ...
Generated M function file in the M file ./fill_FA1101P0_program_store.m
Generated C header file in the H file ./fill_FA1101P0_program_store.h
EDKDSPCC : a_fp1101p1.c ...
EDKDSPASM: FA1101P1.PSM ...
Generated M function file in the M file ./fill_FA1101P1_program_store.m
Generated C header file in the H file ./fill_FA1101P1_program_store.h
EDKDSPCC : a_fp1124p0.c ...
EDKDSPASM: FA1124P0.PSM ...
Generated M function file in the M file ./fill_FA1124P0_program_store.m
Generated C header file in the H file ./fill_FA1124P0_program_store.h
EDKDSPCC : a_fp1124p1.c ...
EDKDSPASM: FA1124P1.PSM ...
Generated M function file in the M file ./fill_FA1124P1_program_store.m
Generated C header file in the H file ./fill_FA1124P1_program_store.h
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$
```

Figure 28: Use of UTIA EdkDSP C compiler in Ubuntu terminal under the VMware Player

3.4 Examples of firmware modifications

Use of EdkDSP C compiler will be presented in this section. We will change, recompile and execute the PicoBlaze6 firmware program edkdsp_cc/a/a_fp1101p0.c. See Figure 24 and Figure 25 for the original code and Figure 29 for the modified code. Instead of printing the accelerator external input as I= we will print from PicoBlaze6 to the MicroBlaze Input=. See the modification performed in SDK on Figure 29.

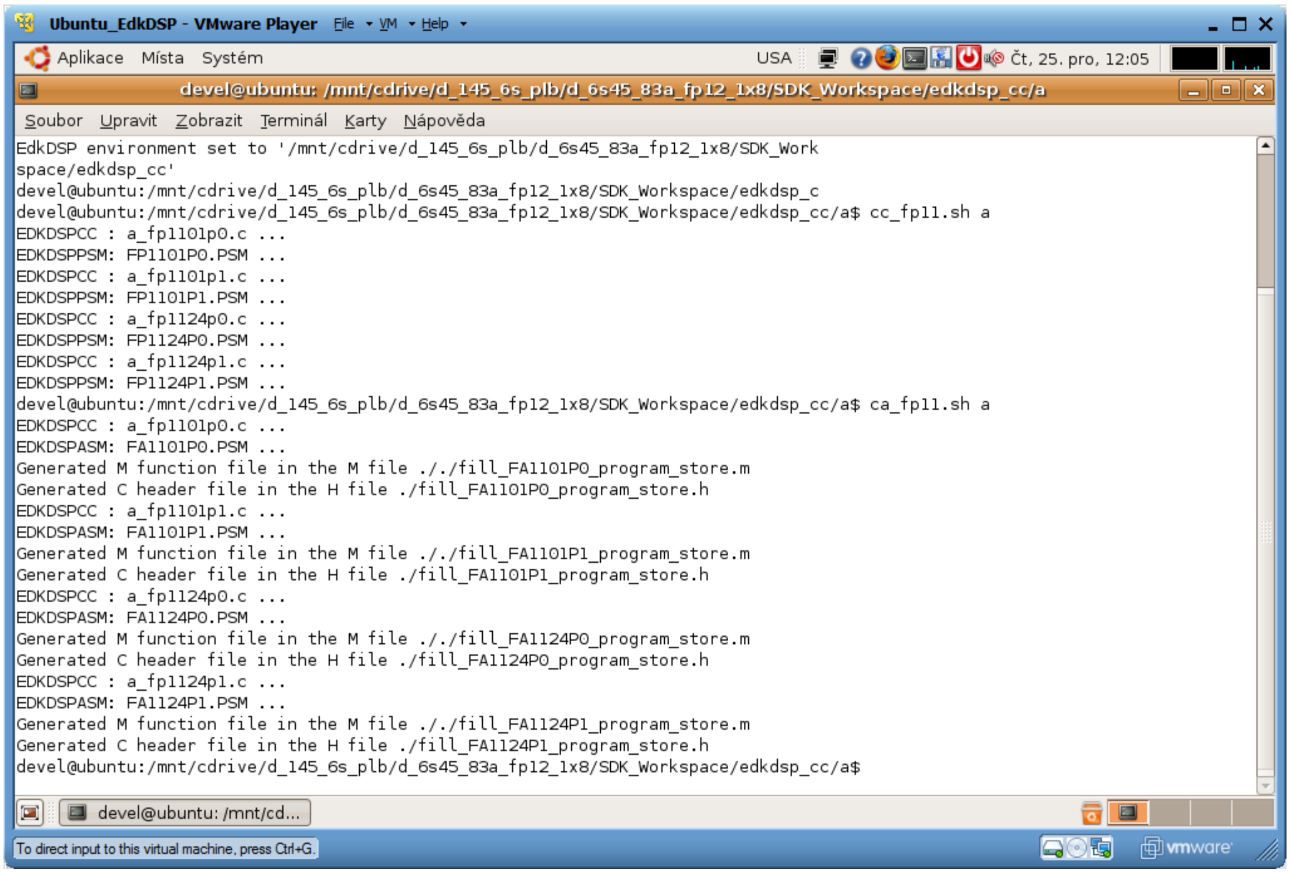


```
        pb2mb_write(C_SCREEN_3);
        {
            pb2mb_write('\r');
            pb2mb_write(' ');
            pb2mb_write('I');
            pb2mb_write('n');
            pb2mb_write('p');
            pb2mb_write('u');
            pb2mb_write('t');
            pb2mb_write('=');
            pb2mb_write(hex_h(btn));
            pb2mb_write(hex_l(btn));
        }
        pb2mb_eoc(led);

        pb2mb_write(C_SCREEN_4);
        {
            pb2mb_write('\r');
            pb2mb_write(' ');
            pb2mb_write('a');
            pb2mb_write('h');
            pb2mb_write('=');
            pb2mb_write(hex_h(ah));
            pb2mb_write(hex_l(ah));
            pb2mb_write(' ');
            pb2mb_write('b');
            pb2mb_write('h');
            pb2mb_write('=');
            pb2mb_write(hex_h(bh));
            pb2mb_write(hex_l(bh));
            pb2mb_write(' ');
            pb2mb_write('z');
```

```
<terminated> edkdsp Debug [Xilinx C/C++ ELF] C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspac
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDIO setti
```

Figure 29: Simple modification of fp110p0.c (I changed to Input)

Ttt

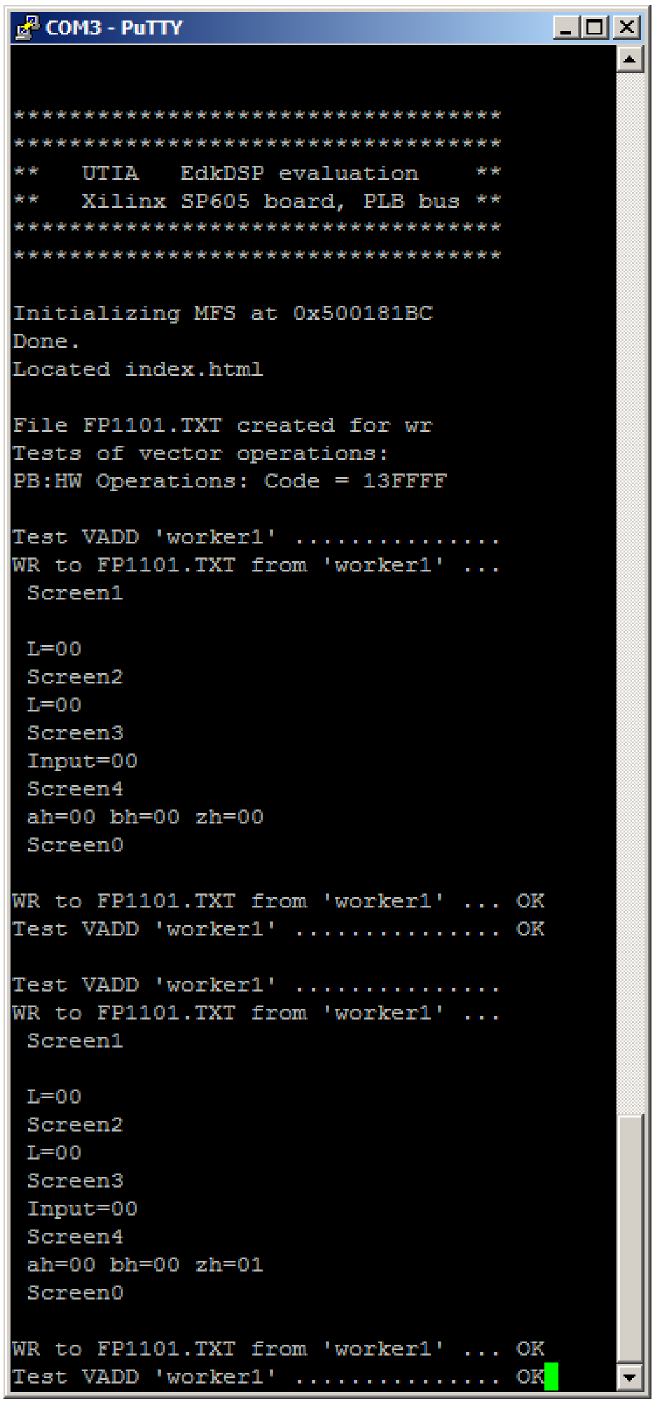

Ubuntu_EdkDSP - VMware Player
devel@ubuntu: /mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a
To direct input to this virtual machine, press Ctrl+G.

```
EdkDSP environment set to '/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Work
space/edkdsp_cc'
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_c
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ cc_fp11.sh a
EDKDSPCC : a_fp1101p0.c ...
EDKDSPPSM: FP1101P0.PSM ...
EDKDSPCC : a_fp1101p1.c ...
EDKDSPPSM: FP1101P1.PSM ...
EDKDSPCC : a_fp1124p0.c ...
EDKDSPPSM: FP1124P0.PSM ...
EDKDSPCC : a_fp1124p1.c ...
EDKDSPPSM: FP1124P1.PSM ...
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ ca_fp11.sh a
EDKDSPCC : a_fp1101p0.c ...
EDKDSPASM: FA1101P0.PSM ...
Generated M function file in the M file ././fill_FA1101P0_program_store.m
Generated C header file in the H file ./fill_FA1101P0_program_store.h
EDKDSPCC : a_fp1101p1.c ...
EDKDSPASM: FA1101P1.PSM ...
Generated M function file in the M file ././fill_FA1101P1_program_store.m
Generated C header file in the H file ./fill_FA1101P1_program_store.h
EDKDSPCC : a_fp1124p0.c ...
EDKDSPASM: FA1124P0.PSM ...
Generated M function file in the M file ././fill_FA1124P0_program_store.m
Generated C header file in the H file ./fill_FA1124P0_program_store.h
EDKDSPCC : a_fp1124p1.c ...
EDKDSPASM: FA1124P1.PSM ...
Generated M function file in the M file ././fill_FA1124P1_program_store.m
Generated C header file in the H file ./fill_FA1124P1_program_store.h
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$
```

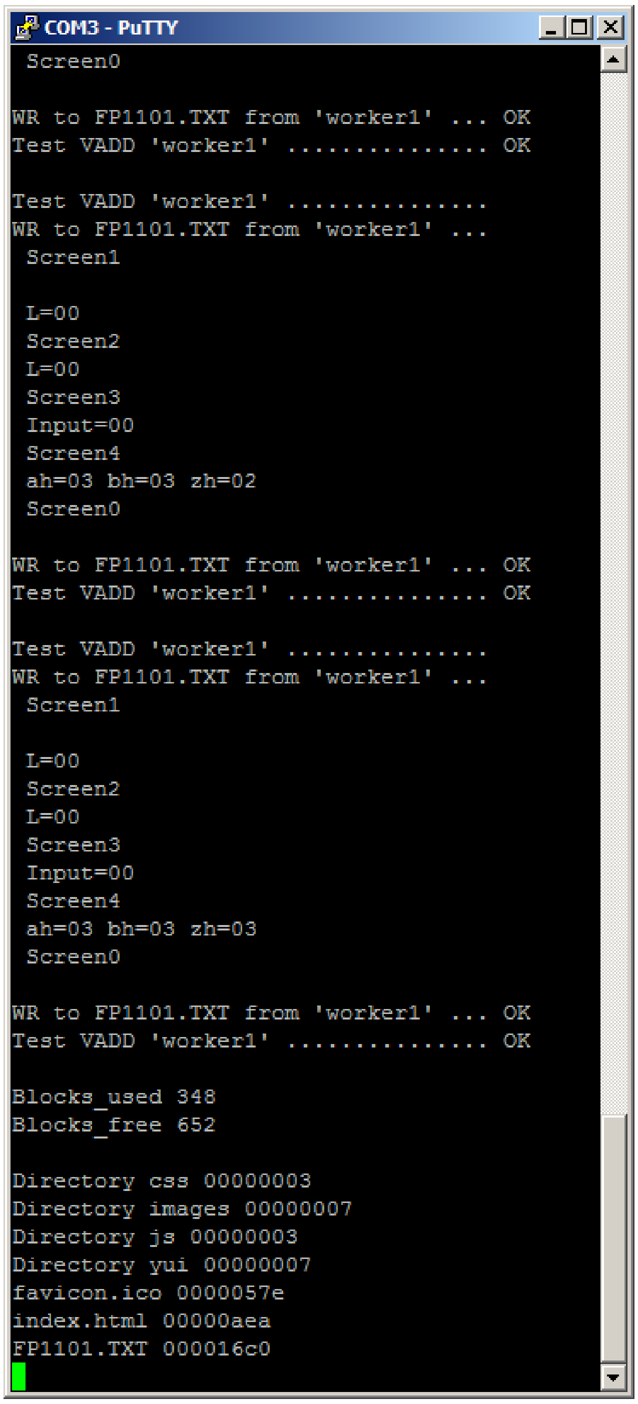
Figure 30: Simple modification of a_fp110p0.c (I changed to Input)

The modified firmware edkdsp_cc/a/a_fp1101p0.c is recompiled as indicated in Figure 30. In SDK, copy and paste the generated fill_FA1101P0_program_store.h and fill_FA1101P0_program._store.h files into the SDK edkdsp project directory edkdsp/src, recompile the edkdsp project and test it on board with new firmware.

Result is displayed on Figure 31. Please compare it with Figure 23. Screen3 string I= has been replaced with the string Input= as expected.

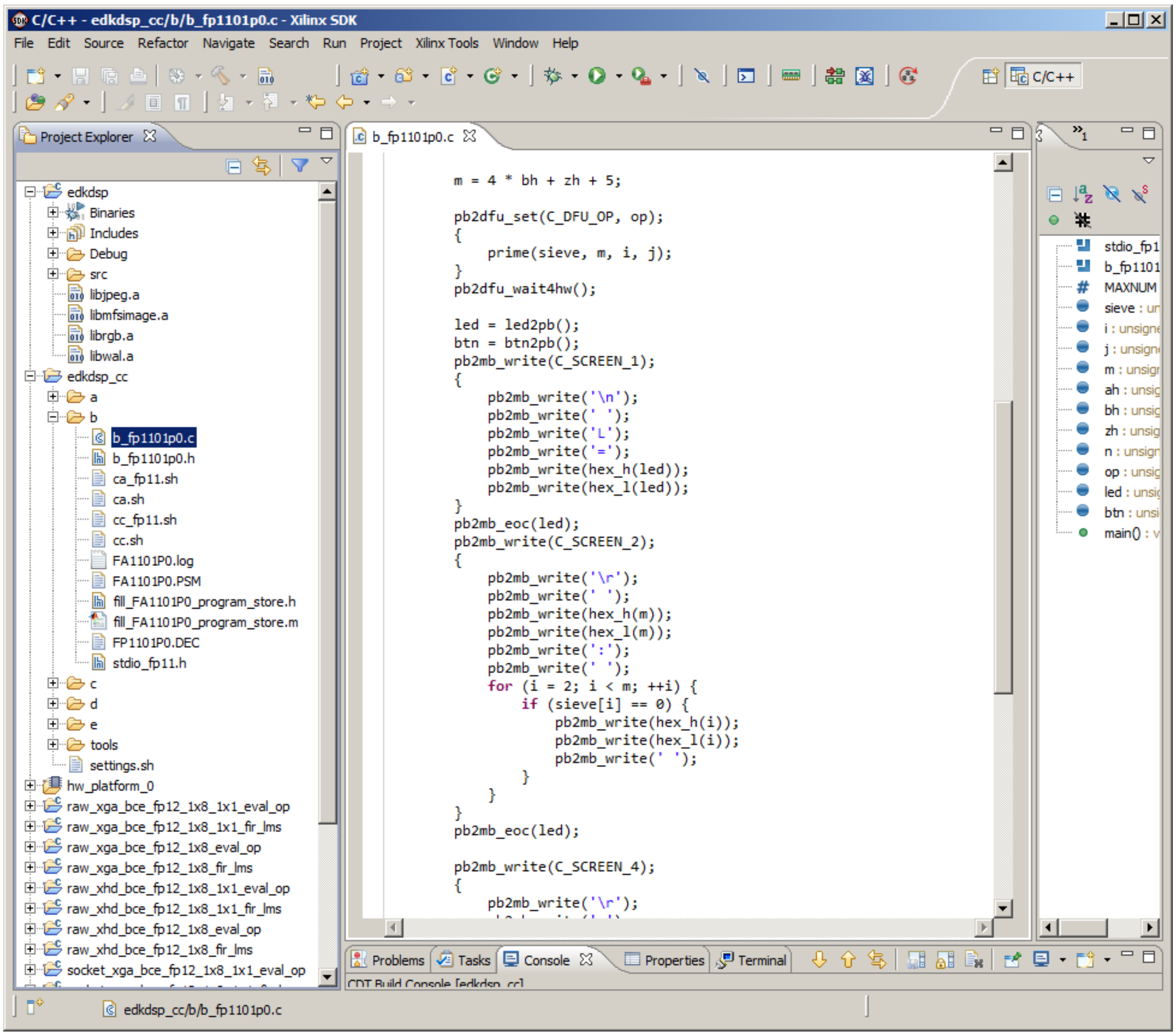


```
************************************
************************************
**   UTIA   EdkDSP evaluation    **
**   Xilinx SP605 board, PLB bus **
************************************
************************************

Initializing MFS at 0x500181BC
Done.
Located index.html

File FP1101.TXT created for wr
Tests of vector operations:
PB:HW Operations: Code = 13FFFF

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 Input=00
 Screen4
 ah=00 bh=00 zh=00
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 Input=00
 Screen4
 ah=00 bh=00 zh=01
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK
```



```
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 Input=00
 Screen4
 ah=03 bh=03 zh=02
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 L=00
 Screen3
 Input=00
 Screen4
 ah=03 bh=03 zh=03
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Blocks_used 348
Blocks_free 652

Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1101.TXT 000016c0
```

Figure 31: Terminal output from "edkdsp" with modified firmware initial part left, final part right.


C/C++ - edkdsp_cc/b/b_fp1101p0.c - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
Project Explorer
edkdsp
Binaries
Includes
Debug
src
libjpeg.a
libmfsimage.a
librgb.a
libwal.a
edkdsp_cc
a
b
b_fp1101p0.c
b_fp1101p0.h
ca_fp11.sh
ca.sh
cc_fp11.sh
cc.sh
FA1101P0.log
FA1101P0.PSM
fill_FA1101P0_program_store.h
fill_FA1101P0_program_store.m
FP1101P0.DEC
stdio_fp11.h
c
d
e
tools
settings.sh
hw_platform_0
raw_xga_bce_fp12_1x8_1x1_eval_op
raw_xga_bce_fp12_1x8_1x1_fir_lms
raw_xga_bce_fp12_1x8_eval_op
raw_xga_bce_fp12_1x8_fir_lms
raw_xhd_bce_fp12_1x8_1x1_eval_op
raw_xhd_bce_fp12_1x8_1x1_fir_lms
raw_xhd_bce_fp12_1x8_eval_op
raw_xhd_bce_fp12_1x8_fir_lms
socket_xga_bce_fp12_1x8_1x1_eval_op
b_fp1101p0.c
stdio_fp1
b_fp1101
MAXNUM
sieve : ur
i : unsigne
j : unsigne
m : unsigr
ah : unsig
bh : unsig
zh : unsig
n : unsigr
op : unsig
led : unsig
btn : unsi
main() : v
Problems Tasks Console Properties Terminal
CDT Build Console [edkdsp_cc]
edkdsp_cc/b/b_fp1101p0.c
C/C++

```
m = 4 * bh + zh + 5;

pb2dfu_set(C_DFU_OP, op);
{
    prime(sieve, m, i, j);
}
pb2dfu_wait4hw();

led = led2pb();
btn = btn2pb();
pb2mb_write(C_SCREEN_1);
{
    pb2mb_write('\n');
    pb2mb_write(' ');
    pb2mb_write('L');
    pb2mb_write('=');
    pb2mb_write(hex_h(led));
    pb2mb_write(hex_l(led));
}
pb2mb_eoc(led);
pb2mb_write(C_SCREEN_2);
{
    pb2mb_write('\r');
    pb2mb_write(' ');
    pb2mb_write(hex_h(m));
    pb2mb_write(hex_l(m));
    pb2mb_write(':');
    pb2mb_write(' ');
    for (i = 2; i < m; ++i) {
        if (sieve[i] == 0) {
            pb2mb_write(hex_h(i));
            pb2mb_write(hex_l(i));
            pb2mb_write(' ');
        }
    }
}
pb2mb_eoc(led);

pb2mb_write(C_SCREEN_4);
{
    pb2mb_write('\r');
```

Figure 32: Firmware b_fp110p0.c with computation of prime numbers.

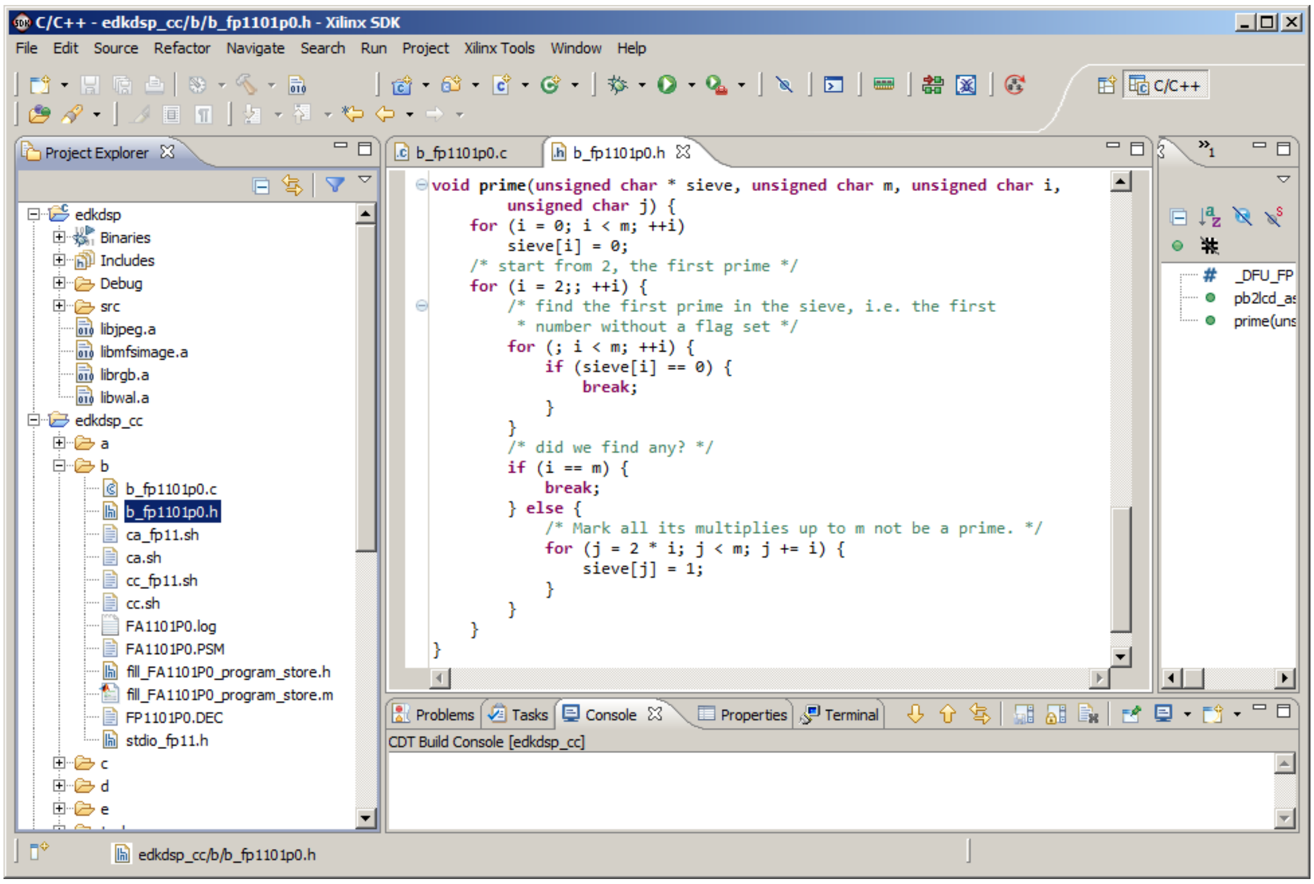
The firmware b_fp110p0.c present in edkdsp_cc/b/ directory includes simple computation of prime numbers in the range from 2 to 20.

It is computed in parallel with the vector floating point computation pf the 8xSIMD data flow unit. See Figure 32.

The PicoBlaze C function prime() is displayed in Figure 33.
Compilation of the firmware b_fp110p0.c is presented in Figure 34.
Results printed on the console are displayed on Figure 35.

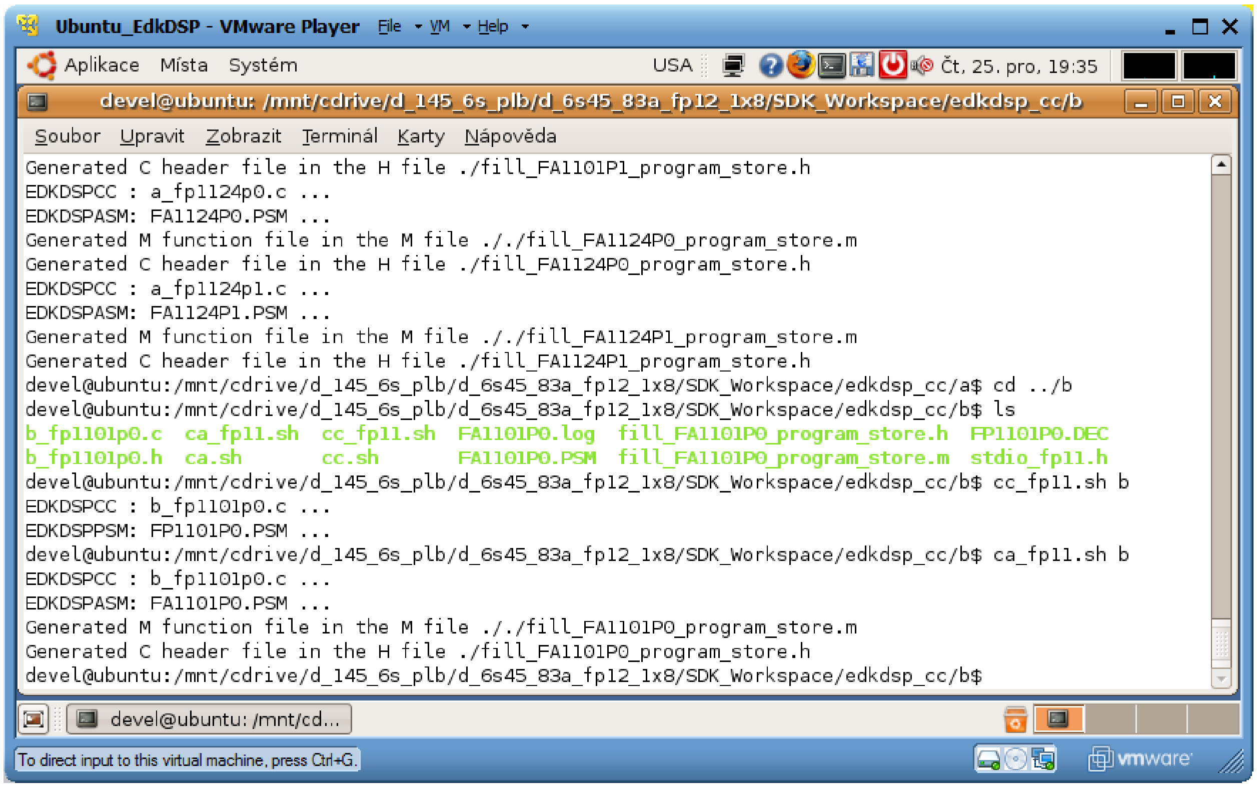
This example documents the use of user defined functions, declaration and use of single dimensional array in the EdkDSP C compiler for the PicoBlaze6 controller.

The firmware c_fp110p0.c present in edkdsp_cc/c/ directory includes identical computation of prime numbers. It is documenting effect of the declaration of variables in stack instead of the declaration as global variables.



```
void prime(unsigned char * sieve, unsigned char m, unsigned char i,
        unsigned char j) {
    for (i = 0; i < m; ++i)
        sieve[i] = 0;
    /* start from 2, the first prime */
    for (i = 2;; ++i) {
        /* find the first prime in the sieve, i.e. the first
         * number without a flag set */
        for (; i < m; ++i) {
            if (sieve[i] == 0) {
                break;
            }
        }
        /* did we find any? */
        if (i == m) {
            break;
        } else {
            /* Mark all its multiples up to m not be a prime. */
            for (j = 2 * i; j < m; j += i) {
                sieve[j] = 1;
            }
        }
    }
}
```

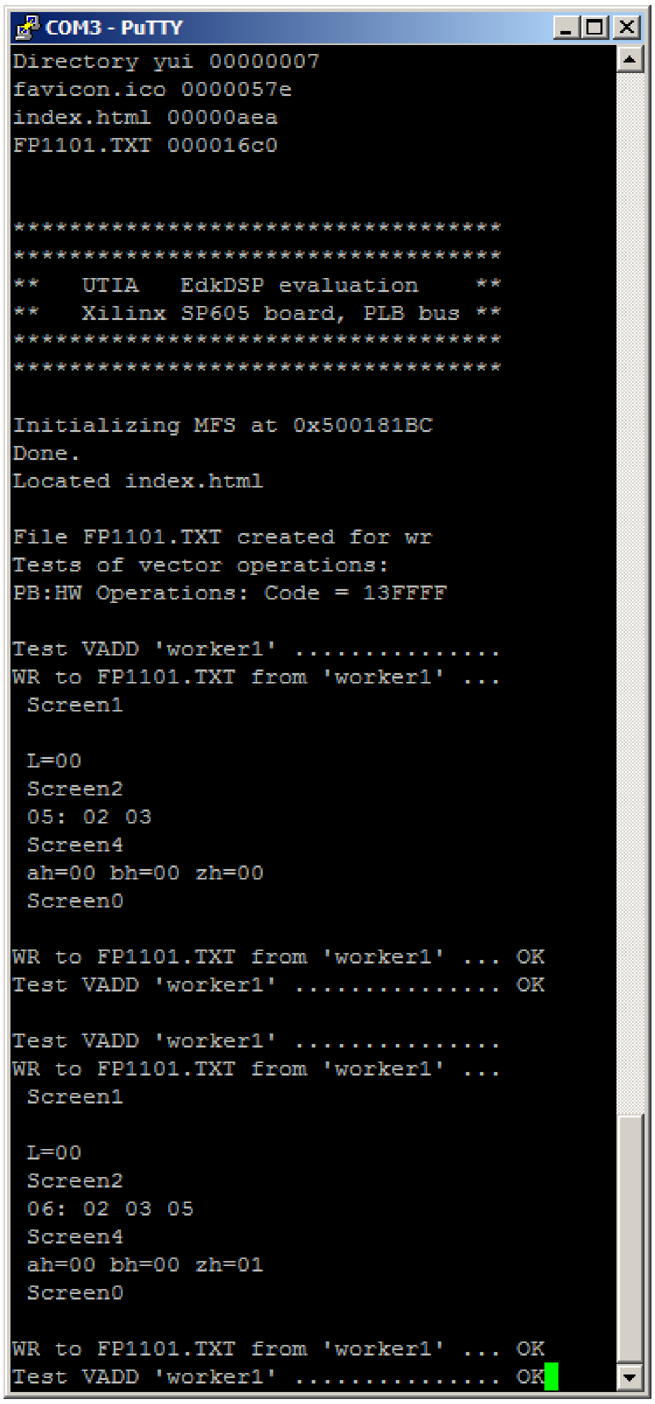
Figure 33: PicoBlaze C function prime() is computing prime numbers.


Ubuntu_EdkDSP - VMware Player File VM Help
Aplikace Místa Systém
USA
Čt, 25. pro, 19:35
devel@ubuntu: /mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/b
Soubor Upravit Zobrazit Terminál Karty Nápověda
devel@ubuntu: /mnt/cd...
To direct input to this virtual machine, press Ctrl+G.
vmware

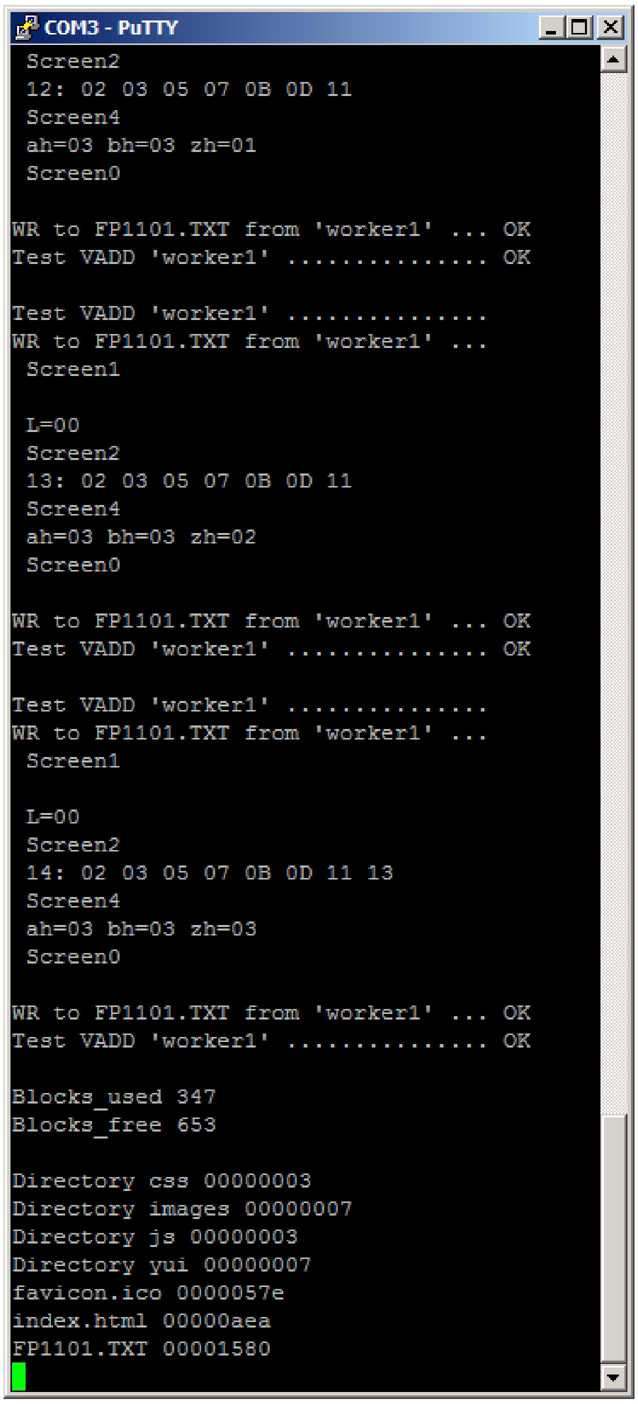
```
Generated C header file in the H file ./fill_FA1101P1_program_store.h
EDKDSPCC : a_fp1124p0.c ...
EDKDSPASM: FA1124P0.PSM ...
Generated M function file in the M file ././fill_FA1124P0_program_store.m
Generated C header file in the H file ./fill_FA1124P0_program_store.h
EDKDSPCC : a_fp1124p1.c ...
EDKDSPASM: FA1124P1.PSM ...
Generated M function file in the M file ././fill_FA1124P1_program_store.m
Generated C header file in the H file ./fill_FA1124P1_program_store.h
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a$ cd ../b
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/b$ ls
b_fp1101p0.c  ca_fp11.sh  cc_fp11.sh  FA1101P0.log  fill_FA1101P0_program_store.h  FP1101P0.DEC
b_fp1101p0.h  ca.sh       cc.sh       FA1101P0.PSM  fill_FA1101P0_program_store.m  stdio_fp11.h
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/b$ cc_fp11.sh b
EDKDSPCC : b_fp1101p0.c ...
EDKDSPPSM: FP1101P0.PSM ...
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/b$ ca_fp11.sh b
EDKDSPCC : b_fp1101p0.c ...
EDKDSPASM: FA1101P0.PSM ...
Generated M function file in the M file ././fill_FA1101P0_program_store.m
Generated C header file in the H file ./fill_FA1101P0_program_store.h
devel@ubuntu:/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/b$
```

Figure 34: Compilation of firmware b_fp1101p0.c

Ttt



```
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1101.TXT 000016c0


************************************
************************************
**   UTIA   EdkDSP evaluation     **
**   Xilinx SP605 board, PLB bus **
************************************
************************************

Initializing MFS at 0x500181BC
Done.
Located index.html

File FP1101.TXT created for wr
Tests of vector operations:
PB:HW Operations: Code = 13FFFF

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 05: 02 03
 Screen4
 ah=00 bh=00 zh=00
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 06: 02 03 05
 Screen4
 ah=00 bh=00 zh=01
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK
```

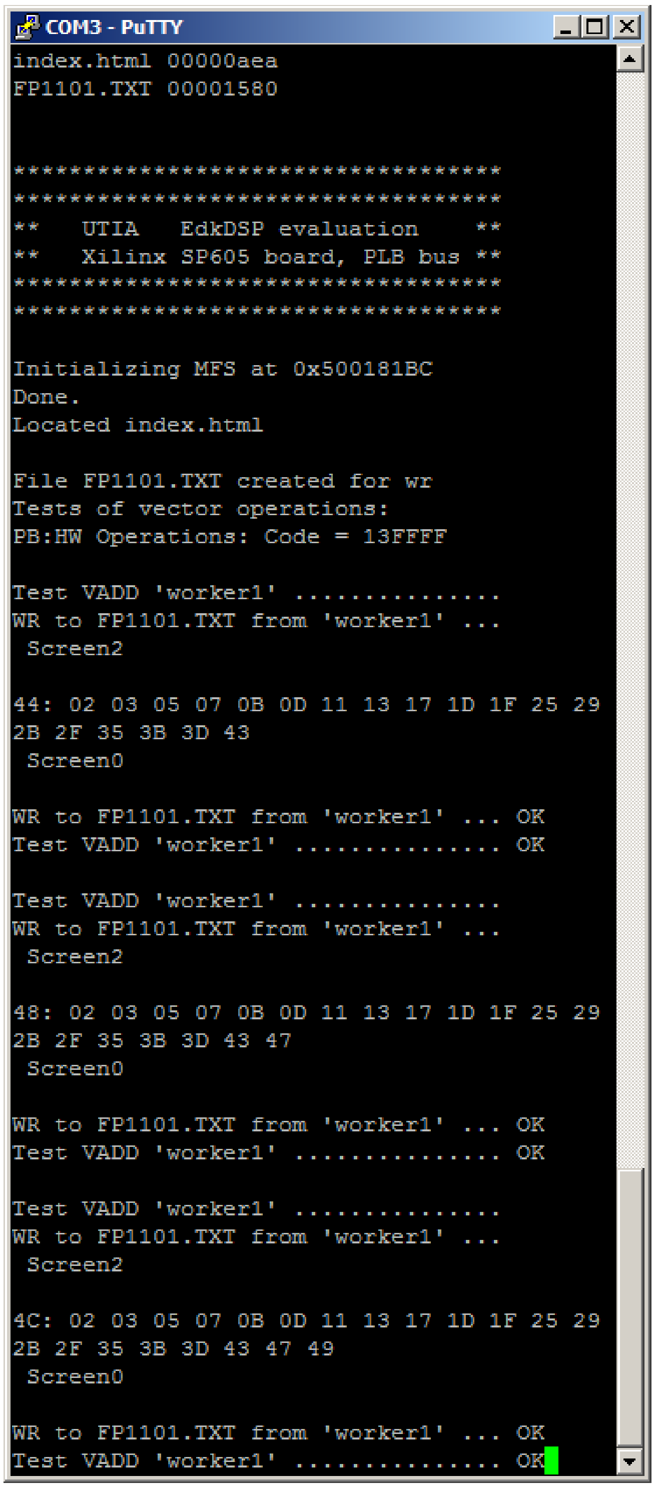


```
 Screen2
 12: 02 03 05 07 0B 0D 11
 Screen4
 ah=03 bh=03 zh=01
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 13: 02 03 05 07 0B 0D 11
 Screen4
 ah=03 bh=03 zh=02
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen1

 L=00
 Screen2
 14: 02 03 05 07 0B 0D 11 13
 Screen4
 ah=03 bh=03 zh=03
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Blocks_used 347
Blocks_free 653

Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1101.TXT 00001580
```

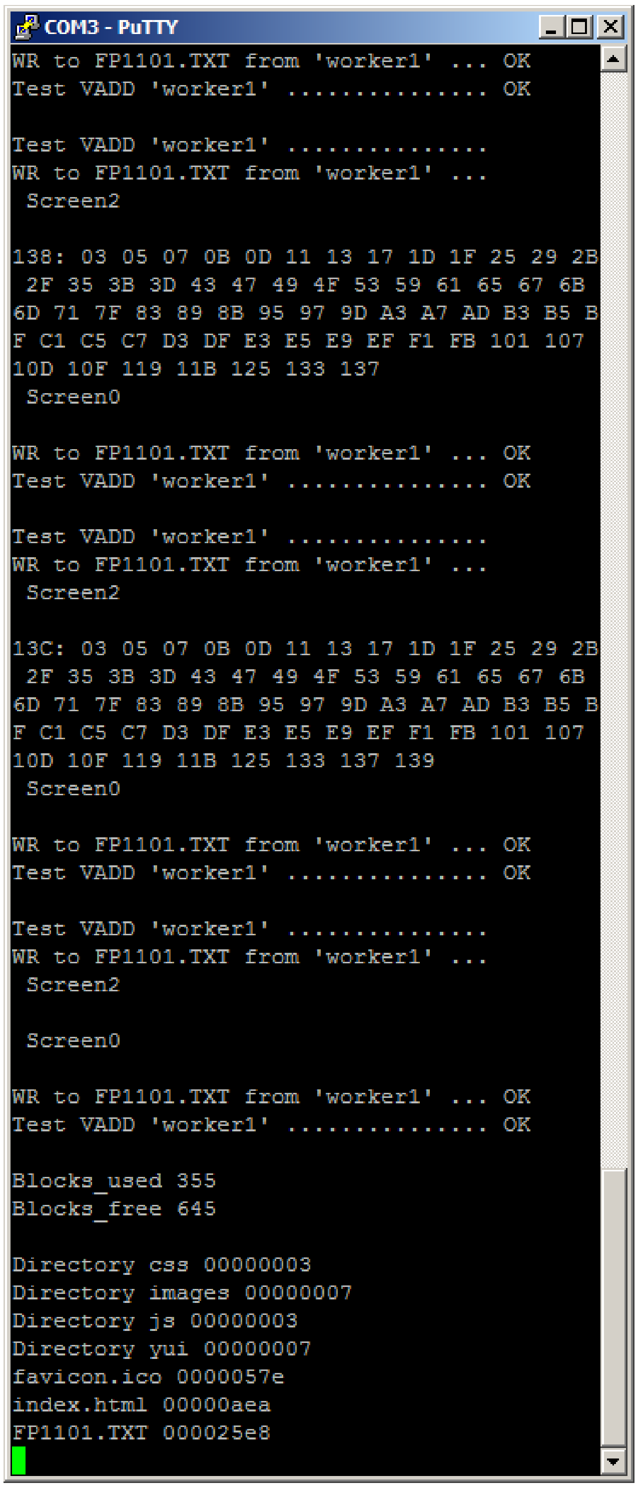
Figure 35: Terminal output from "edkdsp"initial part left, final part right.

The firmware d_fp110p0.c present in edkdsp_cc/d/ directory demontrates PicoBlaze6 computation of prime numbers in the range from 2 to 255. The algorithm is documenting the access to bits in an unsigned char array and the use of unsigned int data type (16 bit).

The firmware e_fp110p0.c present in edkdsp_cc/e/ directory demontrates PicoBlaze6 computation of prime numbers in the range from 2 to 320. The algorithm is documenting printing and display of data larger than 0xFF. See Figure 36.



```
index.html 00000aea
FP1101.TXT 00001580


***********************************
***********************************
**    UTIA    EdkDSP evaluation    **
**    Xilinx SP605 board, PLB bus **
***********************************
***********************************

Initializing MFS at 0x500181BC
Done.
Located index.html

File FP1101.TXT created for wr
Tests of vector operations:
PB:HW Operations: Code = 13FFFF

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen2

44: 02 03 05 07 0B 0D 11 13 17 1D 1F 25 29
2B 2F 35 3B 3D 43
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen2

48: 02 03 05 07 0B 0D 11 13 17 1D 1F 25 29
2B 2F 35 3B 3D 43 47
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen2

4C: 02 03 05 07 0B 0D 11 13 17 1D 1F 25 29
2B 2F 35 3B 3D 43 47 49
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK
```



```
WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen2

138: 03 05 07 0B 0D 11 13 17 1D 1F 25 29 2B
 2F 35 3B 3D 43 47 49 4F 53 59 61 65 67 6B
6D 71 7F 83 89 8B 95 97 9D A3 A7 AD B3 B5 B
F C1 C5 C7 D3 DF E3 E5 E9 EF F1 FB 101 107
10D 10F 119 11B 125 133 137
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen2

13C: 03 05 07 0B 0D 11 13 17 1D 1F 25 29 2B
 2F 35 3B 3D 43 47 49 4F 53 59 61 65 67 6B
6D 71 7F 83 89 8B 95 97 9D A3 A7 AD B3 B5 B
F C1 C5 C7 D3 DF E3 E5 E9 EF F1 FB 101 107
10D 10F 119 11B 125 133 137 139
 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Test VADD 'worker1' ...............
WR to FP1101.TXT from 'worker1' ...
 Screen2

 Screen0

WR to FP1101.TXT from 'worker1' ... OK
Test VADD 'worker1' ............... OK

Blocks_used 355
Blocks_free 645

Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1101.TXT 000025e8
```

Figure 36: Terminal output from firmware e_fp1101p0.c with prime numvers from 2 to 320.

See Figure 36 for the output from the firmware e_fp110p0.c present in edkdsp_cc/e/ directory.

3.5 EdkDSP C compiler parameters, restrictions and the PicoBlaze6 API

The EdkDSP C compiler supports these data types:

•	unsigned char	8 bit values	0x0	…	0xFF
•	char	8 bit values	0x80	…	0x7F
•	unsigned int	8 bit values	0x0	…	0xFFFF
•	int	16 bit values	0x8000	…	0x7FFF

- Single dimensional array of unsigned char, char, unsigned int or int elements is supported.
- Pointer type pointing to the array, pointing to the array element or to the variable is supported.
- Function calls with all supported data types or pointers to these data types as output and input arguments.

Restrictions of the EdkDSP C compiler.

- 32 bit integer (signed or unsigned) data type is not supported.
- Array of pointers is not supported.
- Structures are not supported.
- Global variables and arrays can be only declared. They are NOT initialised to zero and cannot be statically initialised by the compiler. Content of global variables must be defined in the program.
- Text strings are not supported.

Restriction related to size of the PicoBlaze6 controller:
The PicoBlaze6 is working with only 64 bytes of the internal scratch pad memory. This memory serves for all C program variables and arrays and it is also used by the C program stack.
Program size is limited to max 4096 assembly code instructions. Each instruction is 18 bit wide.
All assembler instructions are executed in 2 clock cycles.

Programming guidelines related to the restriction of the EdkDSP C compiler.

Use of Global variables and arrays if possible. It results in faster code.
Use software initialisation of Global variables and arrays as the first step of your program.

If two subsequent C programs use an identical set of global variables and arrays, the variables will be allocated in identical locations of the scratch pad memory. If the first program performs software initialisation of global variables and arrays, performs some computations and terminates, the second program can take the advantage of initialised global variables left from the first program in the scratch pad memory. The sequence of declaration of global variables, data types and the size of arrays must be identical to make this method valid.

Optimisations performed by EdkDSP C compiler:
Expressions with constants known in the compile time are computed by the compiler and only the resulting constant is entering into the compilation process.
The compiler is using fixed set of processor registers for transfer of parameters and results in function calls. This helps to work with mixed C and ASM language programming. The PicoBlaze6 I/O functions
are defined in the ASM language and declared as functions callable from C code.
The reserved PicoBlaze6 registers (unused by the EdkDSP C compiler) can be used by the ASM language code functions. This is possible only if the interrupt is not used and if the ASM language code functions do not call another ASM language code functions.

All EdkDSP C PicoBlaze6 programs can use these optimized I/O functions:

unsigned char mb2pb_read_data();	Read value from MicroBlaze (blocking, includes handshake with MicroBlaze SW)
void pb2mb_write(unsigned char data);	Write data value to MicroBlaze MicroBlaze (blocking, includes handshake with MicroBlaze SW)
void pb2mb_eoc(unsigned char data);	Write data value to MicroBlaze MicroBlaze and indicate the end of string flag (blocking, includes handshake with MicroBlaze SW)
void pb2mb_req_reset(unsigned char data);	Write data value to MicroBlaze MicroBlaze and indicate request from to reset PicoBlaze6 (blocking, includes handshake with MicroBlaze SW)
void pb2mb_reset();	Activate reset of PicoBlaze6 MicroBlaze (blocking, includes handshake with MicroBlaze SW)
unsigned char led2pb();	Read status of PicoBlaze6 LED port
unsigned char btn2pb();	Read status of PicoBlaze6 BTN port
unsigned char hex_h(unsigned char ch);	Write to MicroBlaze hexadecimal ascii representation of the higher 4bit of the input 8bit argument
unsigned char hex_l(unsigned char ch);	Write to MicroBlaze hexadecimal ascii representation of the lower 4bit of the input 8bit argument
void pb2dfu_set(unsigned char mem, unsigned char data);	Write 8bit data to the PicoBlaze6 port mem
void pb2dfu_wait4hw();	Wait for end of the EdkDSP data flow unit vector operation

In case of HW support for the 2-line LCD board display, the EdkDSP C programs can use this interface function:

void pb2lcd_ascii_char(unsigned char ch, unsigned char pos);	Write to local 2x16 lcd display. In case of no HW support, the function will write to unconnected ports with no effect.

User can call these optimized ASM functions.
User can define its own C functions or ASM functions and include them in .h file or directly in the C code.

User defined functions can be defined and tested in C code first. The compiled ASM code can be often included in the next stage with some additional optimisations leading to shorter code and improved performance.

Examples of this approach to the optimisation can be seen in the included EdkDSP C source code examples edkdsp_cc/a … edkdsp_cc/e.

3.6 Evaluation of all EdkDSP vector operations, use of Video display and 1Gb Ethernet

The SDK SW projects included in this evaluation package demonstrate integration of the UTIA EdkDSP accelerator together with the Xilinx display controller and the Xilinx 1 Gb Xps_Soft_TEMAC controller, connecting the board to the Ethernet. The connection to the Ethernet is based on two versions of the LwIP SW:

- Raw versions of SDK SW projects use raw version of the LwIP library without real-time OS.
- Socket versions of SW projects use the socket version of LwIP in top of Xilinx XilKernel real-time OS.
- SDK SW projects named as “raw_xga…” or “socket_xga…” will work with the analogue RGB 888 display supporting the resolution 1280x792p60 with pixel clock 83,3 MHz.
- SW projects named as “raw_xhd…” or “socket_xhd…” will work with the digital RGB 888 display supporting the resolution 1280x792p60 with pixel clock 83,3 MHz and the DVI or HDMI digital interface.

Set your PC Ethernet connection to point-to-point fixed IP address:

192.168.8.2

All included UTIA EdkDSP projects are setting the IP address of the SP605 to:

192.168.8.10

This setting enables the direct point to point Ethernet connection.

Program the SP605 board by selecting in SDK:
Xilinx Tools -> Program FPGA
Type or Browse to the “system.bit” file and the “sysrem_bd.bmm” file:
C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform\system.bit
C:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform\system_bd.bmm

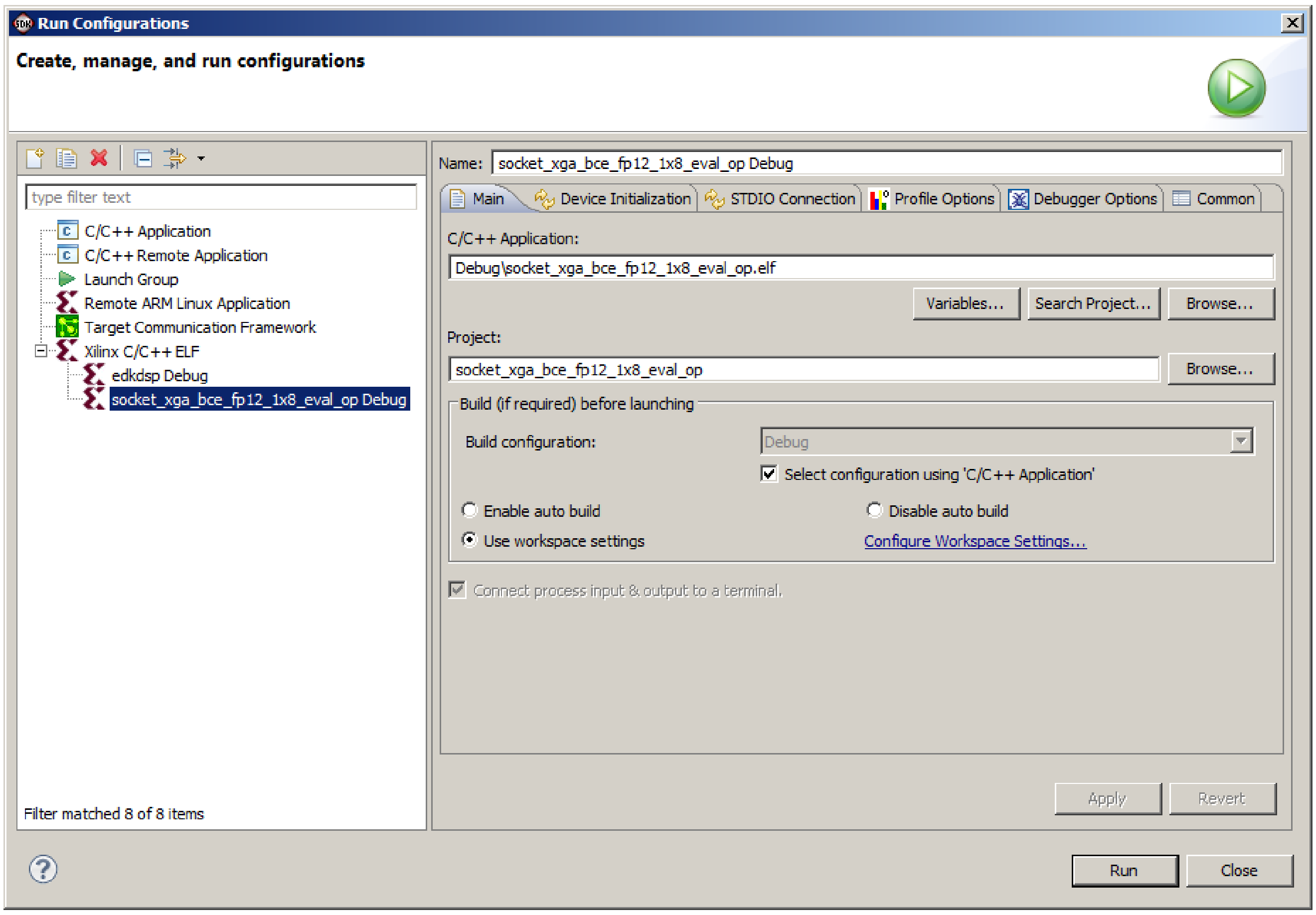
Click on the “Program” button.

The SP605 board is programmed with the .bit file now. The MicroBlaze is running in the initial bootloop.
The SW .elf application from the “socket_xhd_vce_fp12_eval_op” project has to be downloaded to the DDR3 memory, now.

Connect the SP605 to your PC with the Ethernet cable and select “socket_xhd_vce_fp12_eval_op” SDK project:
In SDK, select:
Run -> Run Configuration -> Xilinx C/C++ ELF

Click on the “New launch configuration” in the Run configuration screen and the “socket_xhd_vce_fp12_eval_op” project executable is ready for download to DDR3 on the SP605 board via the jtag cable.

Click on “Run” button to download the executable. See Figure 37.



Name: socket_xga_bce_fp12_1x8_eval_op Debug

C/C++ Application: Debug\socket_xga_bce_fp12_1x8_eval_op.elf

Project: socket_xga_bce_fp12_1x8_eval_op

Build (if required) before launching

Build configuration: Debug

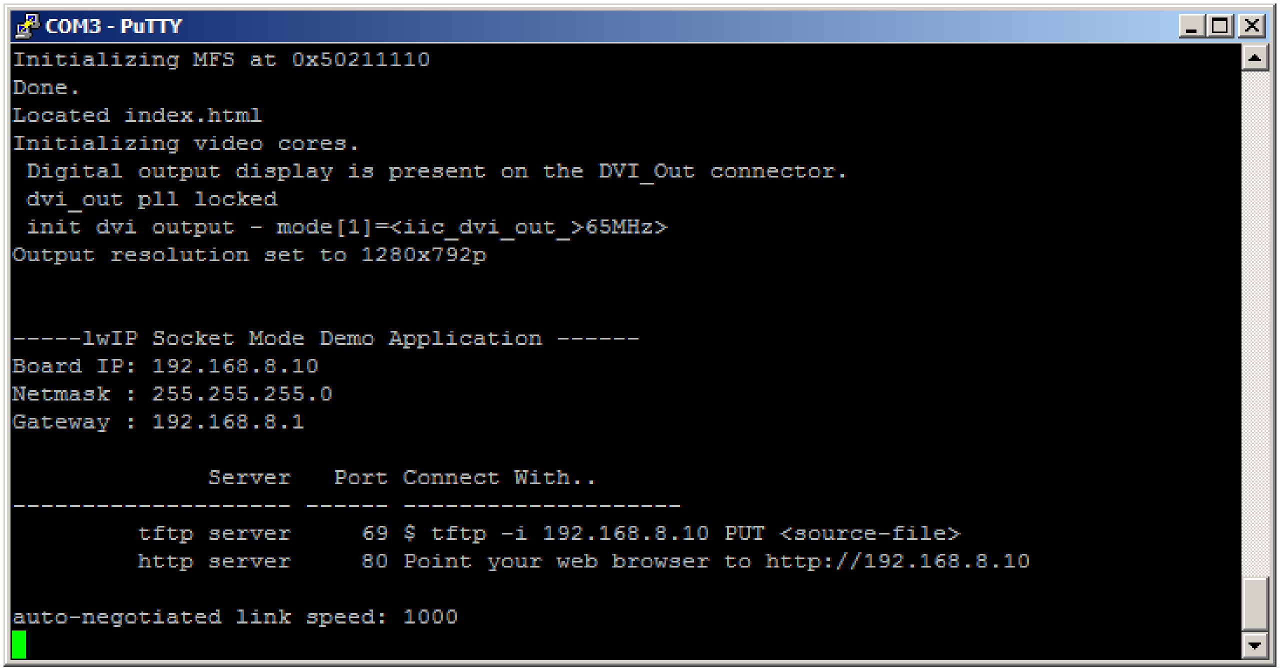
[✓] Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

[✓] Connect process input & output to a terminal.

Figure 37: Run the "socket_xhd_bce_fp12_1x8_eval_op" application.



```
Initializing MFS at 0x50211110
Done.
Located index.html
Initializing video cores.
 Digital output display is present on the DVI_Out connector.
 dvi_out pll locked
 init dvi output - mode[1]=<iic_dvi_out_>65MHz>
Output resolution set to 1280x792p


-----lwIP Socket Mode Demo Application ------
Board IP: 192.168.8.10
Netmask : 255.255.255.0
Gateway : 192.168.8.1

              Server   Port Connect With..
-------------------- ------ --------------------
         tftp server     69 $ tftp -i 192.168.8.10 PUT <source-file>
         http server     80 Point your web browser to http://192.168.8.10

auto-negotiated link speed: 1000
```

Figure 38: Initial console output from the "socket_xhd_bce_fp12_1x8_eval_op" application.

See the terminal output on Figure 38. The application is performing these steps:

- Initialize cache and serial line controller.
- Initialize memory based file system from data linked in the .elf binary file.
- The file index.html is located in the initialised ram-based file system.
- The digital dvi output to the video display is set up with the resolution 1280x792p
- The lwIP socket based Ethernet support is started
- The trivial FTP server is started as process on MicroBlaze with support from XilKernel OS.
- The http server is started as process on MicroBlaze with support from XilKernel OS.
- The xps_Soft_TEMAC controller has negotiated the Ethernet link speed 1 Gbit.

The application is waiting for events coming from the TFTP client (GET or PUT files) or events from and Ethernet Browser like Microsoft Explorer.

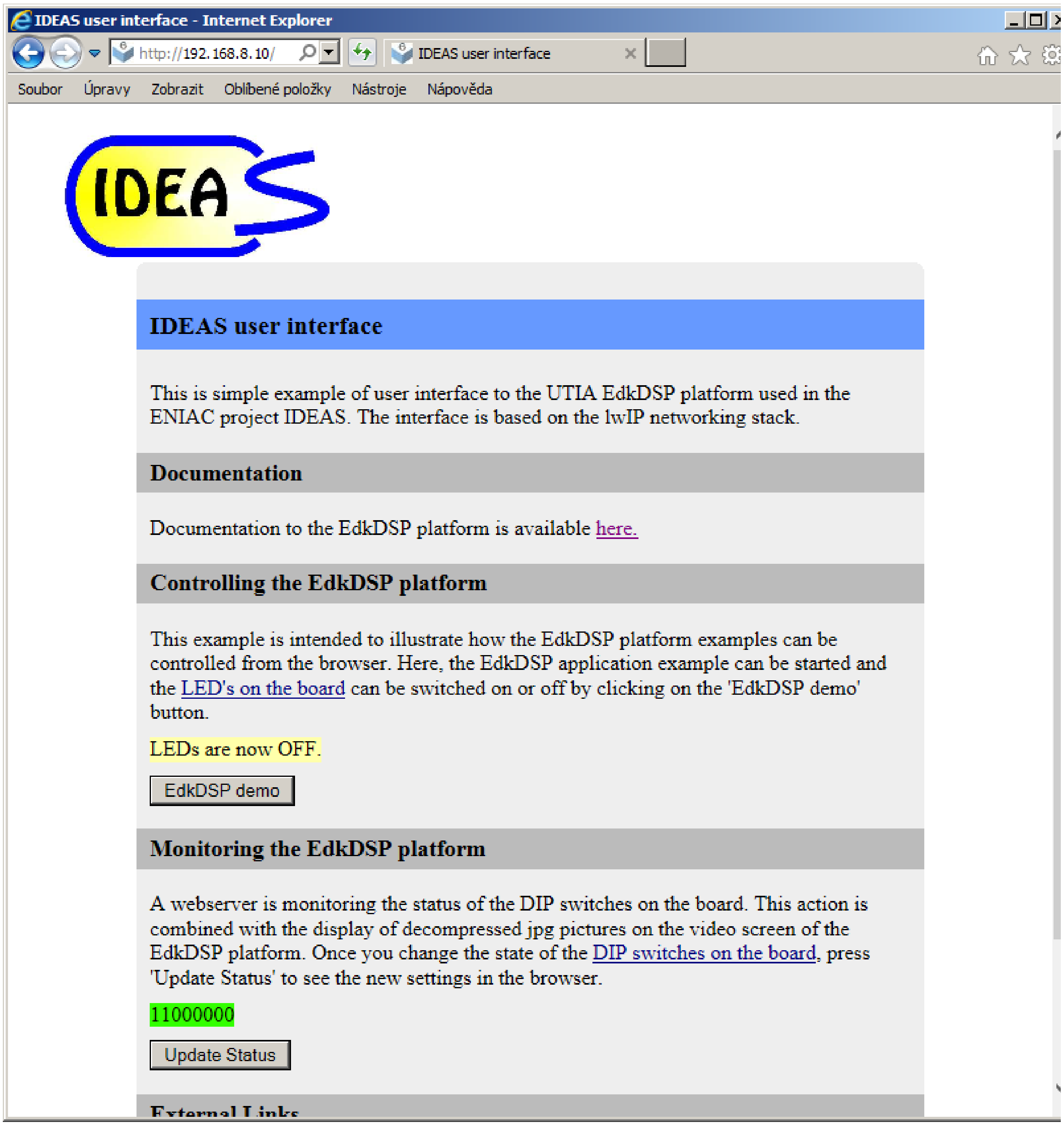
Start Microsoft Explorer on your PC.

Point to:

http://192.168.8.10

The IDEAS user interface page is open. See Figure 39. The Internet Explorer is communicating with the http server running on the MicroBlaze processor and downloads the utilities and scripts from the SP605 ram-based file system. These utilities activate the use of the buttons “EdkDSP demo” and “Update status”.

- Click on “EdkDSP demo” button to starts test of all elementary vector operations of the accelerator.
- Click on “Update status” button to Get the status of switches on the SP605 board to www and to the MicroBlaze serial terminal.
- The “Update status” button also initiates the SW jpeg decompression of ENIAC and IDEAS logos to the 1280x792p60 rgb or dvi display. Both jpeg figures are stored in the ram-based file system of the MicroBlaze processor.
- This is presenting, how the MicroBlaze processor can react to user-defined event in parallel to the evaluation of the EdkDSP accelerator vector functions.
- See the details of the simple text screens and support drawings on the display. The PicoBlaze6 controller is exchanging data with the MicroBlaze in the end of each test of vector operation.


IDEAS user interface - Internet Explorer
http://192.168.8.10/
IDEAS user interface
Soubor Úpravy Zobrazit Oblíbené položky Nástroje Nápověda
IDEAS

IDEAS user interface

This is simple example of user interface to the UTIA EdkDSP platform used in the ENIAC project IDEAS. The interface is based on the lwIP networking stack.

Documentation

Documentation to the EdkDSP platform is available here.

Controlling the EdkDSP platform

This example is intended to illustrate how the EdkDSP platform examples can be controlled from the browser. Here, the EdkDSP application example can be started and the LED's on the board can be switched on or off by clicking on the 'EdkDSP demo' button.

LEDs are now OFF.

EdkDSP demo

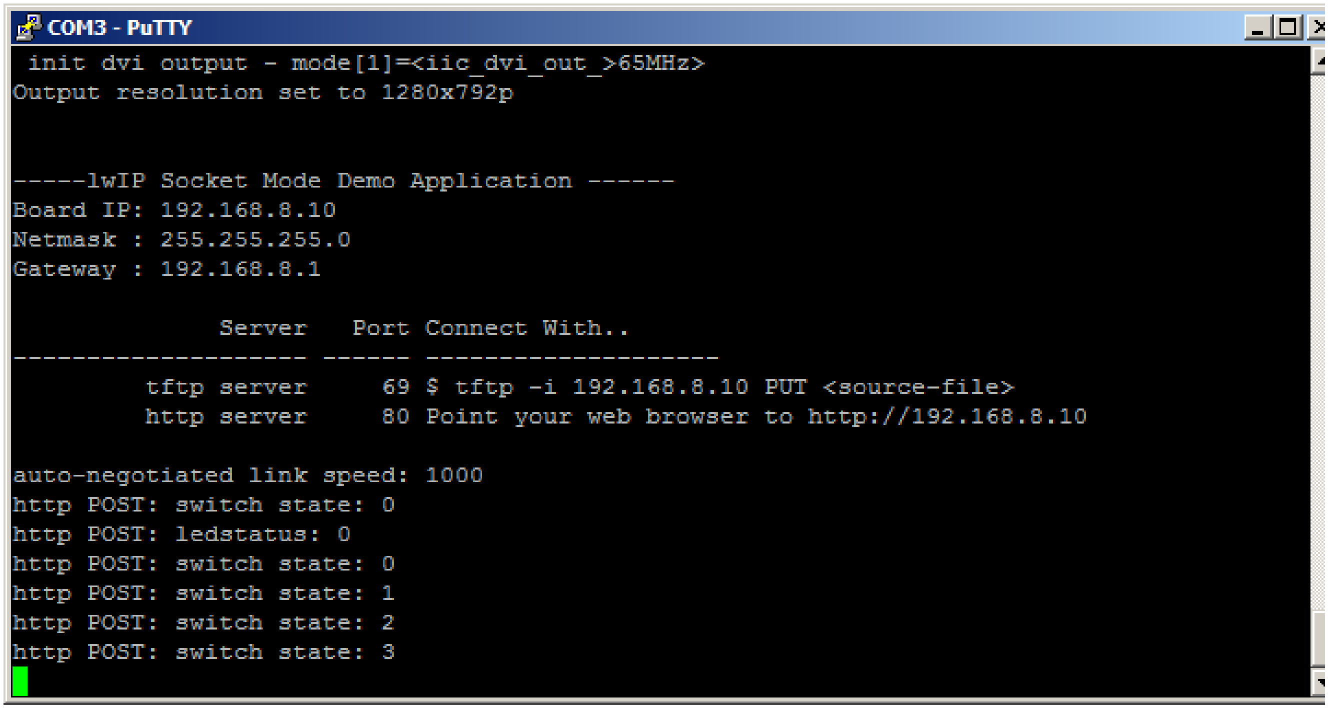
Monitoring the EdkDSP platform

A webserver is monitoring the status of the DIP switches on the board. This action is combined with the display of decompressed jpg pictures on the video screen of the EdkDSP platform. Once you change the state of the DIP switches on the board, press 'Update Status' to see the new settings in the browser.

11000000

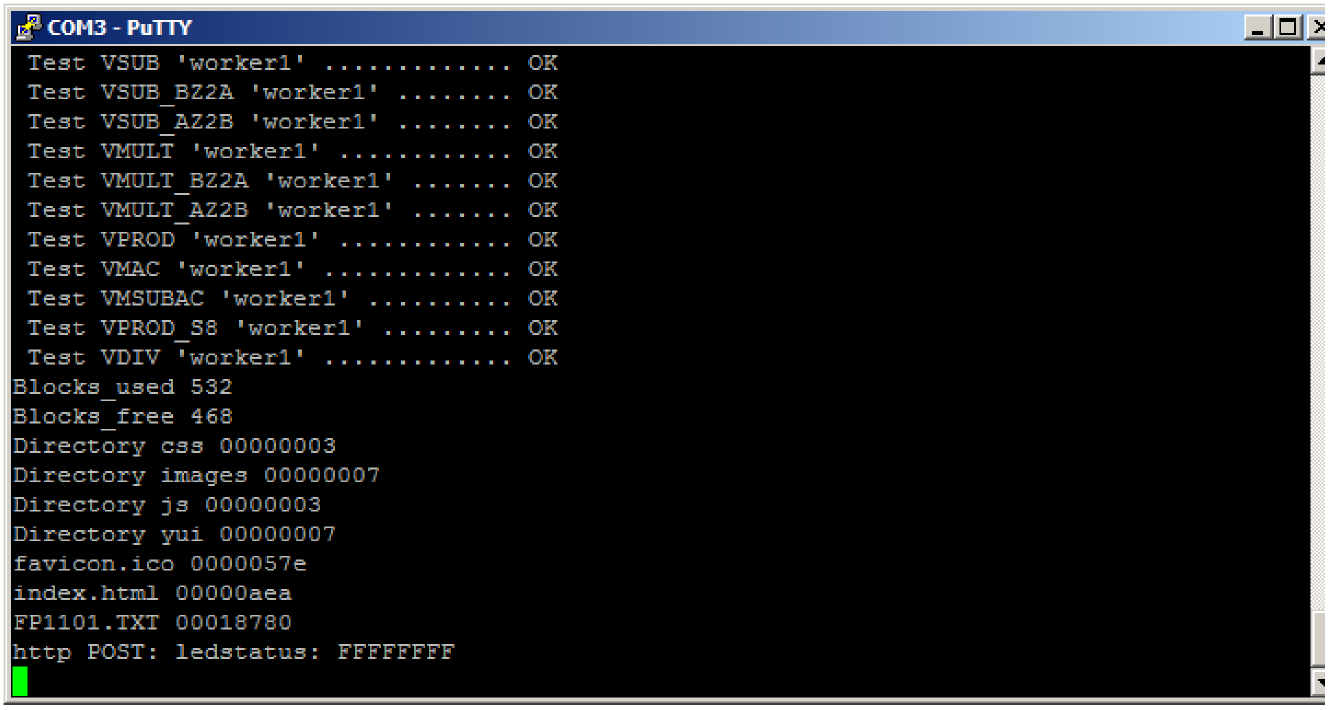
Update Status

Figure 39: Start Internet Explorer and connect to http://192.168.8.10, www page is displayed.



```
 init dvi output - mode[1]=<iic_dvi_out_>65MHz>
Output resolution set to 1280x792p


-----lwIP Socket Mode Demo Application ------
Board IP: 192.168.8.10
Netmask : 255.255.255.0
Gateway : 192.168.8.1

              Server   Port Connect With..
-------------------- ------ --------------------
         tftp server     69 $ tftp -i 192.168.8.10 PUT <source-file>
         http server     80 Point your web browser to http://192.168.8.10

auto-negotiated link speed: 1000
http POST: switch state: 0
http POST: ledstatus: 0
http POST: switch state: 0
http POST: switch state: 1
http POST: switch state: 2
http POST: switch state: 3
```

Figure 40: Press the update button and see the SP605 the terminal and the jpeg figure change.

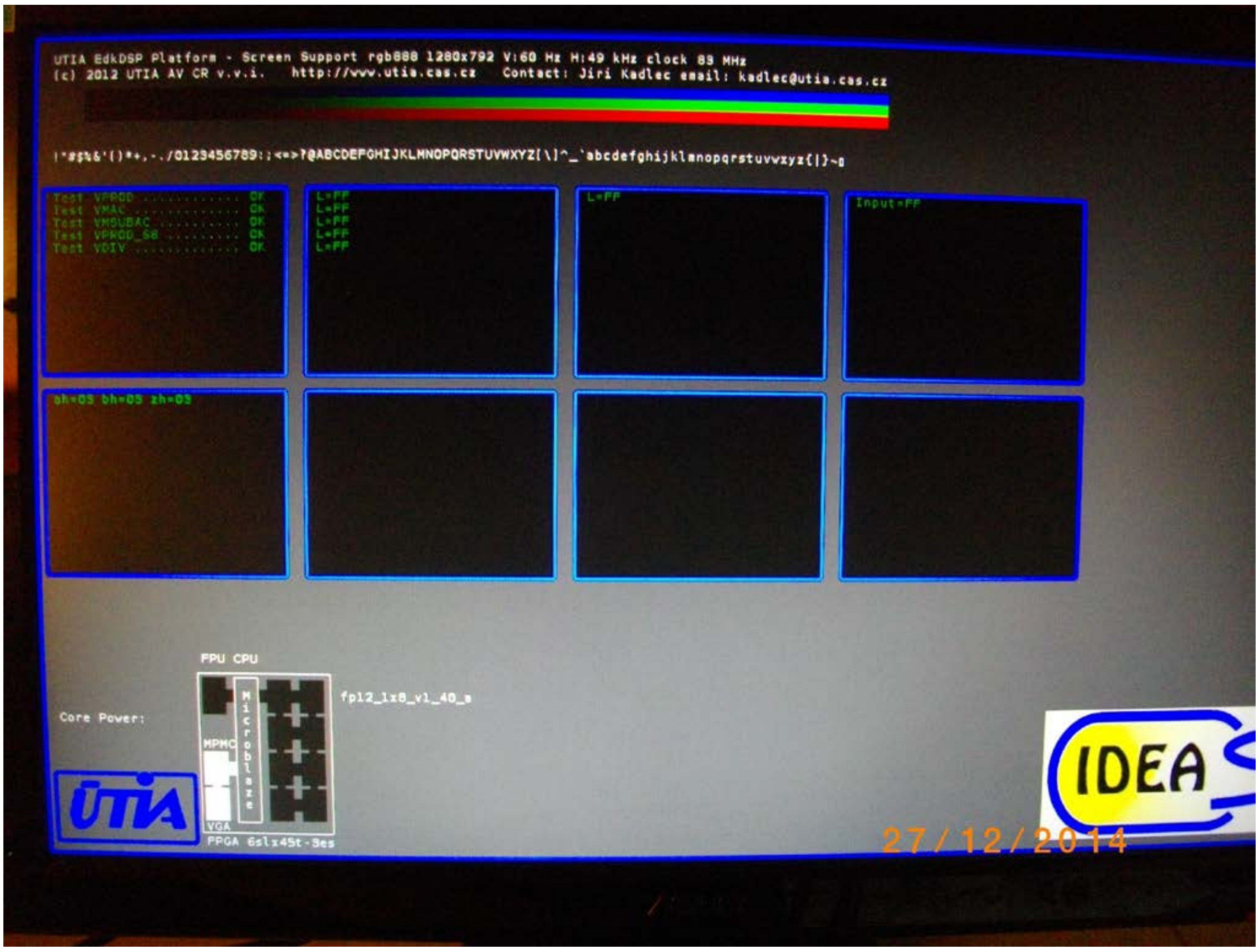


```
 Test VSUB 'worker1' ............. OK
 Test VSUB_BZ2A 'worker1' ........ OK
 Test VSUB_AZ2B 'worker1' ........ OK
 Test VMULT 'worker1' ............ OK
 Test VMULT_BZ2A 'worker1' ....... OK
 Test VMULT_AZ2B 'worker1' ....... OK
 Test VPROD 'worker1' ............ OK
 Test VMAC 'worker1' ............. OK
 Test VMSUBAC 'worker1' .......... OK
 Test VPROD_S8 'worker1' ......... OK
 Test VDIV 'worker1' ............. OK
Blocks_used 532
Blocks_free 468
Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1101.TXT 00018780
http POST: ledstatus: FFFFFFFF
```

Figure 41: Press the EdkDSP demo button, see test results in the terminal, and the file directory.

Data related to the I/O received from PicoBlaze6 are visualised and stored to the ram-based file system. The test sequence (initiated by the click on the “EdkDSP demo” button) ends by

- Print to the terminal the number of free and used data blocks.
- Print list of top level directories present in the file system.



```
UTIA EdkDSP Platform - Screen Support rgb888 1280x792 V:60 Hz H:49 kHz clock 83 MHz
(c) 2012 UTIA AV CR v.v.i.   http://www.utia.cas.cz   Contact: Jiri Kadlec email: kadlec@utia.cas.cz

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

Test VPROD ............ OK
Test VMAC ............. OK
Test VMSUBAC .......... OK
Test VPROD_S8 ......... OK
Test VDIV ............. OK

L=FF
L=FF
L=FF
L=FF
L=FF

L=FF

Input=FF

ah=03 bh=03 zh=03
```

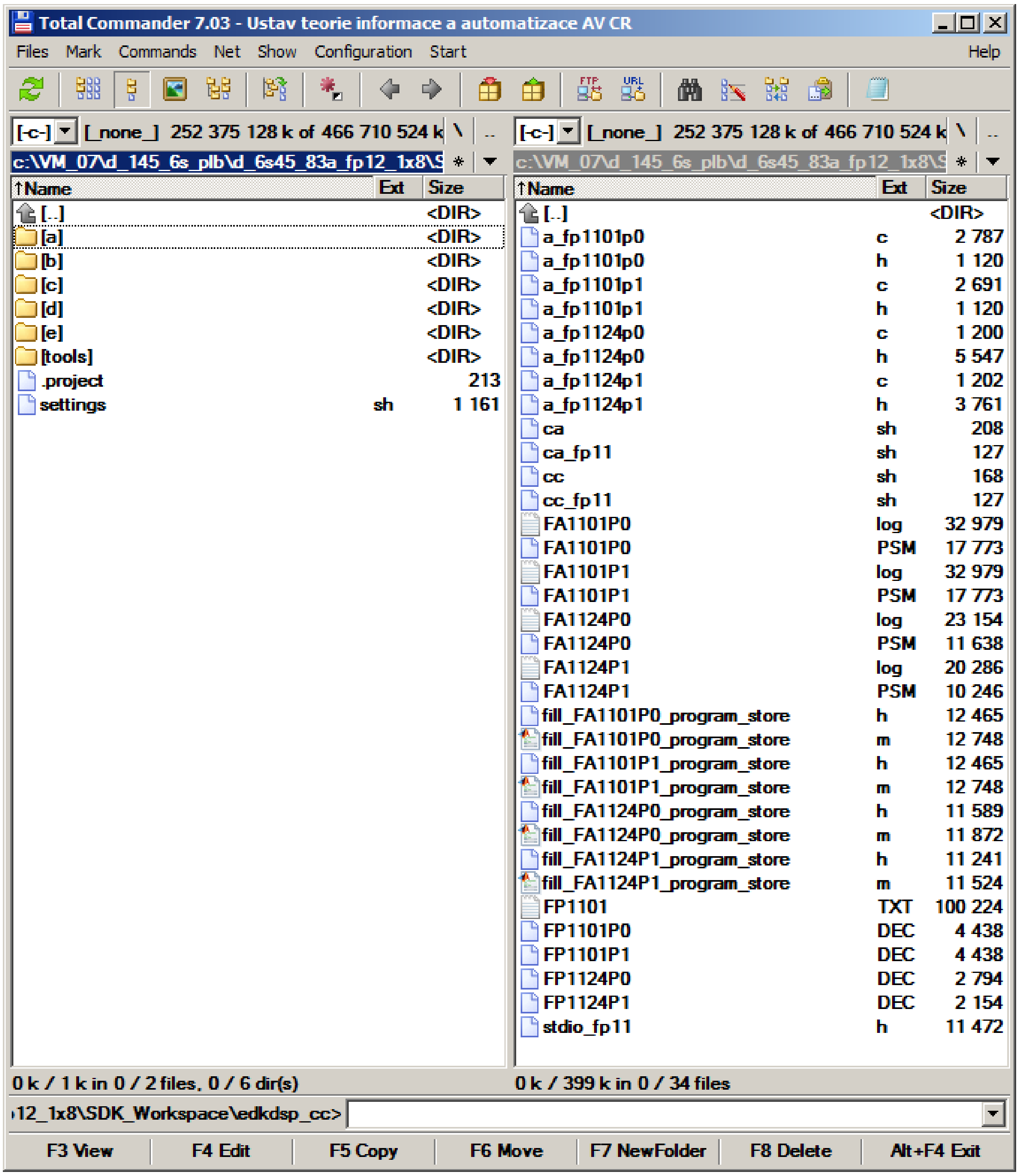
Figure 42: Display screen.

Data related to the I/O received from PicoBlaze6 are visualised and stored to the ram-based file system. The

The trivial FTP server can be evaluated at this stage. See the listed top level directory structure of the ran-based file system on the terminal. There is ascii file FP1101.TXT created by the currently running application. See Figure 41.

Open program like the Total Commander on the PC. See Figure 42. Select the PC directory where FP1101.TXT file will be stored:

c:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp_cc\a



[-c-] [_none_] 252 375 128 k of 466 710 524 k \ ..

c:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\S *

↑Name	Ext	Size
[..]		<DIR>
[a]		<DIR>
[b]		<DIR>
[c]		<DIR>
[d]		<DIR>
[e]		<DIR>
[tools]		<DIR>
.project		213
settings	sh	1 161

0 k / 1 k in 0 / 2 files, 0 / 6 dir(s)

[-c-] [_none_] 252 375 128 k of 466 710 524 k \ ..

c:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\S *

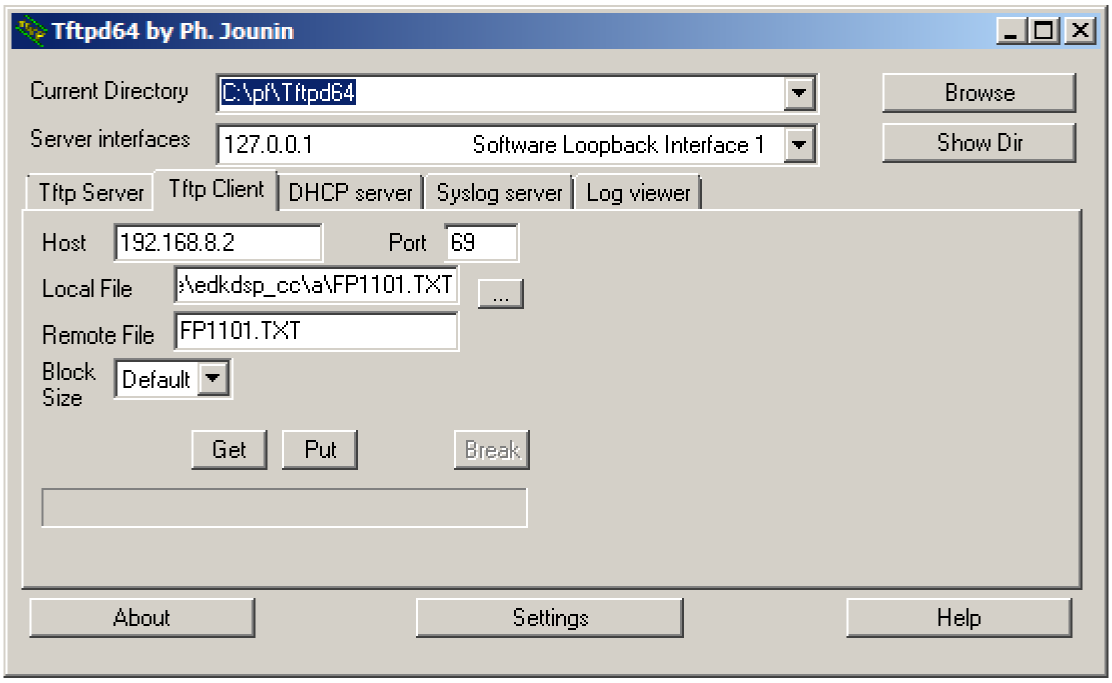
↑Name	Ext	Size
[..]		<DIR>
a_fp1101p0	c	2 787
a_fp1101p0	h	1 120
a_fp1101p1	c	2 691
a_fp1101p1	h	1 120
a_fp1124p0	c	1 200
a_fp1124p0	h	5 547
a_fp1124p1	c	1 202
a_fp1124p1	h	3 761
ca	sh	208
ca_fp11	sh	127
cc	sh	168
cc_fp11	sh	127
FA1101P0	log	32 979
FA1101P0	PSM	17 773
FA1101P1	log	32 979
FA1101P1	PSM	17 773
FA1124P0	log	23 154
FA1124P0	PSM	11 638
FA1124P1	log	20 286
FA1124P1	PSM	10 246
fill_FA1101P0_program_store	h	12 465
fill_FA1101P0_program_store	m	12 748
fill_FA1101P1_program_store	h	12 465
fill_FA1101P1_program_store	m	12 748
fill_FA1124P0_program_store	h	11 589
fill_FA1124P0_program_store	m	11 872
fill_FA1124P1_program_store	h	11 241
fill_FA1124P1_program_store	m	11 524
FP1101	TXT	100 224
FP1101P0	DEC	4 438
FP1101P1	DEC	4 438
FP1124P0	DEC	2 794
FP1124P1	DEC	2 154
stdio_fp11	h	11 472

0 k / 399 k in 0 / 34 files

12_1x8\SDK_Workspace\edkdsp_cc>

Figure 43: Locate place for the logfile "FP1101.TXT"

Use the Trivial FTP client to get the FP1101.TXT file from the ram based file system on the SP605. See Figure 43. If you would use the Windows7 64bit Tftpd64 utility (written by Ph. Jounin), use it as an Tftp Client. Set the "Server interface" to 192.168.8.2 (the PC Ethernet interface) and the "Host" to
192.168.8.10 with "Port" 69 (the SP605 Board). Set the "Remote File" to FP1101.TXT to get the file from the board file system to the PC. See Figure 44.


Tftpd64 by Ph. Jounin

Current Directory: C:\pf\Tftpd64

Server interfaces: 127.0.0.1 Software Loopback Interface 1

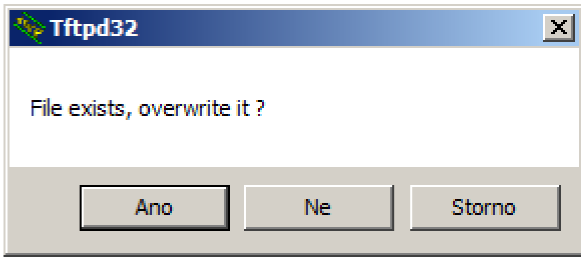
Host: 192.168.8.2 Port: 69

Local File: :\edkdsp_cc\a\FP1101.TXT

Remote File: FP1101.TXT

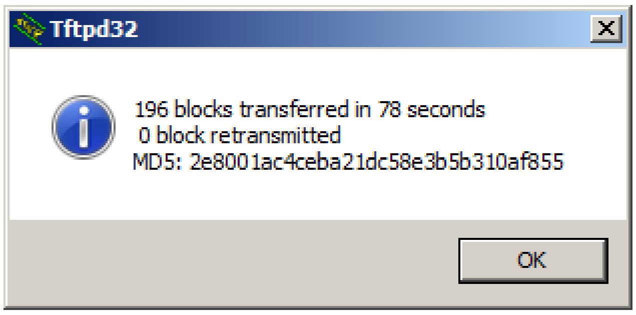
Block Size: Default

Figure 44: Locate place for the logfile “FP1101.TXT”in the TFTP client and click GET


Tftpd32

File exists, overwrite it ?

Figure 45: Confirm overwrite.


Tftpd32

```
196 blocks transferred in 78 seconds
 0 block retransmitted
MD5: 2e8001ac4ceba21dc58e3b5b310af855
```

Figure 46: TFTP is reporting number of transferred blocks from SP605 RAM file system to the PC.

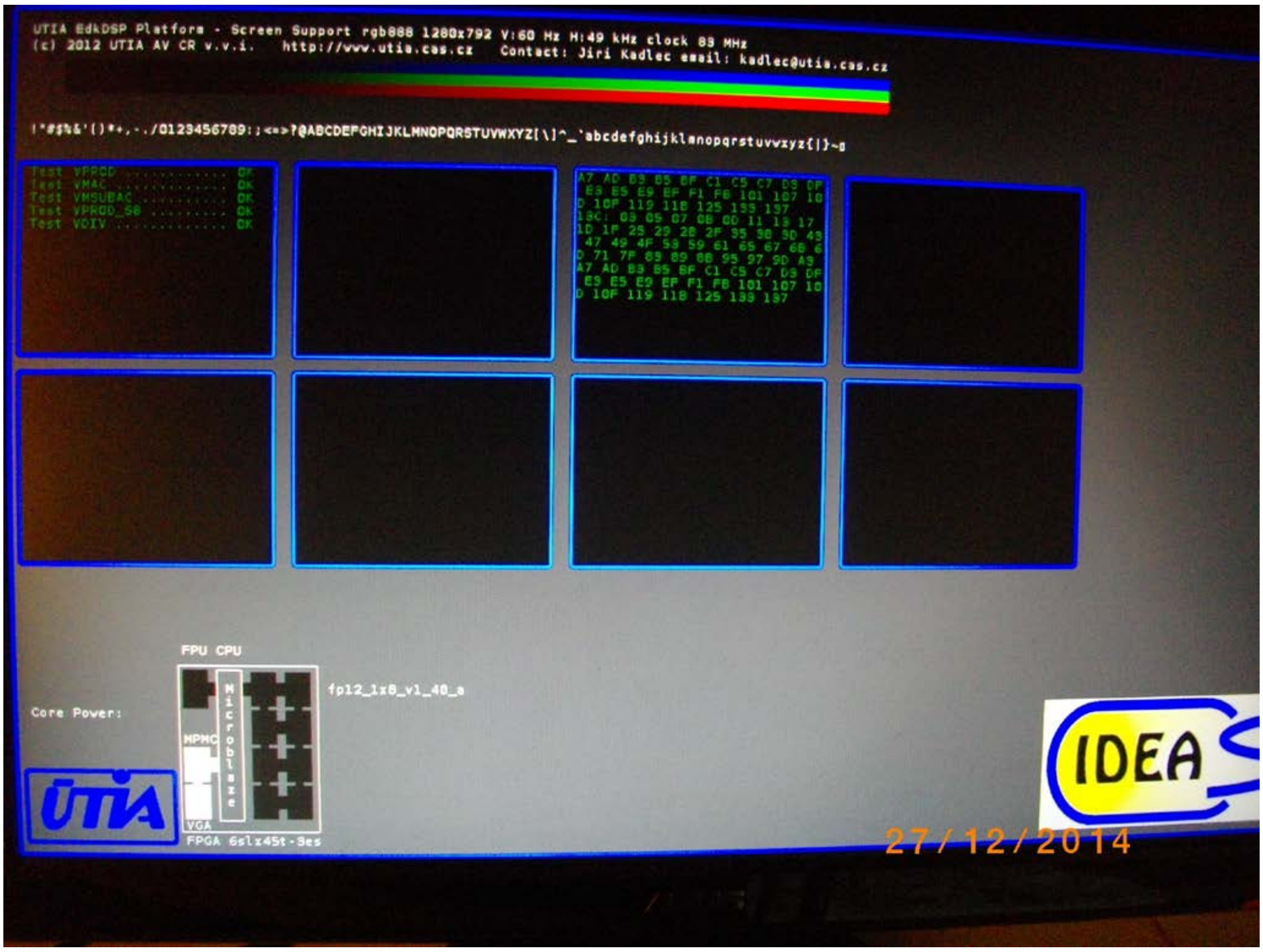
Confirm by OK, that the ascii log file FP1101.TXT is transferred from the SP605 board and it is stored on PC. See Figure 44, Figure 45 and Figure 46.

The TFTP can be used to change the EdkDSP firmware while the MicroBlaze application is running on the board.

Let us consider the firmware update to one of the prime number computation example described in previous sections.

In Windows, drag and drop the firmware file edkdsp_cc/e/FP1101P0.DEC to the Tftpd32 client. It will automatically download this file to the SP605 RAM based file system as FP1101P0.DEC file.
In Ethernet explorer click on the EdkDSP button to start the identical demo with updated firmware.

The demo is started and the EdkDSP accelerator is initialised with the firmware taken from the FP1101P0.DEC file. See the screen on Figure 47.


```
UTIA EdkDSP Platform - Screen Support rgb888 1280x792 V:60 Hz H:49 kHz clock 83 MHz
(c) 2012 UTIA AV CR v.v.i.   http://www.utia.cas.cz   Contact: Jiri Kadlec email: kadlec@utia.cas.cz

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

```
Test VPROD ............ OK
Test VMAC ............. OK
Test VMSUBAC .......... OK
Test VPROD_S8 ......... OK
Test VDIV ............. OK
```

```
A7 AD B3 B5 BF C1 C5 C7 D3 DF
 E3 E5 E9 EF F1 FB 101 107 10
D 10F 119 11B 125 133 137
13C: 03 05 07 0B 0D 11 13 17
1D 1F 25 29 2B 2F 35 3B 3D 43
 47 49 4F 53 59 61 65 67 6B 6
D 71 7F 83 89 8B 95 97 9D A3
A7 AD B3 B5 BF C1 C5 C7 D3 DF
 E3 E5 E9 EF F1 FB 101 107 10
D 10F 119 11B 125 133 137
```

Figure 47: Display with computed prime numbers in Screen2.

The downloaded firmware FP1101P0.DEC created from the source file edkdsp_cc/e/e_fp1101p0.c has been tested in the simple project edkdsp first. See Figure 36. In this case, the compiled firmware has been used without recompilation in much more complex system performing the graphical video output, html www server and TFTP file server with gigabit Ethernet. System is utilising the Xilkernel multitasking operating system.

The support of multitasking can be tested by clicking on the second button in the explorer. It instructs the Microblaze processor to decompress and display the .jpg files with logos of IDEAS project and Eniac program.

These files are stored as. jpg files in the Microblaze file system, decompressed to the RGB bitmap, stored to the DDR3 and displayed on the screen . See Figure 47.

The TFTP client can be used to upload the stored FP1101.TXT file with new content reflecting the complete sequence of prime numbers printed to the display screens in this demo run.

Close the demo www page in the Internet Explorer (see Figure 39) and close the Internet Explorer.

Close the demo application in the SDK. This will stop all parallel SW processes running on the MicroBlaze processor.

The demonstrated www user interface is similar in all SDK SW projects presented in the evaluation version of the EdkDSP platform package.

3.7 Swap of two downloaded firmware programs and evaluation of the EdkDSP performance

Swap of two firmware programs in the runtime and the floating point performance in the DSP application is demonstrated in this section. We will again download updated firmware (precompiled into .DEC file) from Windows 7 (32 or 64 bit) to SP605 board memory-based file system.

Up to now, all MicroBlaze projects have been compiled for debug without optimisation of MicroBlaze code. To measure performance of the EdkDSP accelerator in comparison to the optimized MicroBlaze code utilizing the with the floating point HW support, the MicroBlaze C code has to be compiled with C optimisation flags set to maximal optimization –O3.

In SDK project navigator, select project socket_xhd_fp12_1x8_fir_lms and select the compilation target for Release. The Debug directory inside of the socket_xhd_fp12_1x8_fir_lms project can be safely deleted. This will help to keep just single .elf executable under the Release directory.

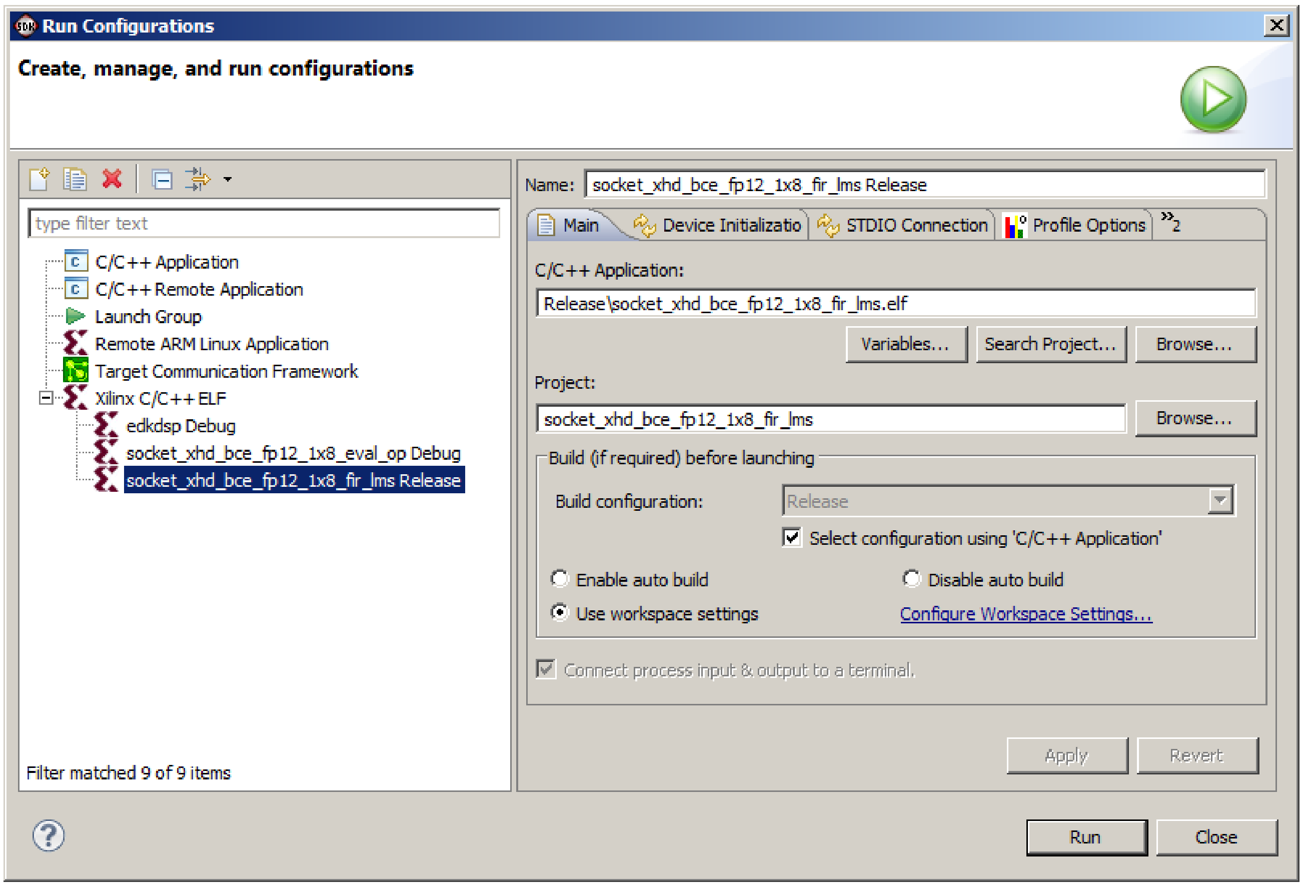
Program the SP605 board by selecting in SDK:
Xilinx Tools -> Program FPGA
Type or Browse to the system.bit file and the system_bd.bmm file:
C:\VM_07\ d_145_6s_plb \d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform\system.bit
C:\VM_07\ d_145_6s_plb \d_6s45_83a_fp12_1x8\SDK_Workspace\hw_platform\system_bd.bmm
Click on the “Program” button.

The SP605 board is programmed with the .bit file now. The MicroBlaze is running in the initial bootloop.
The SW .elf application from the “socket_xhd_fp12_1x8_fir_lms” project has to be downloaded to the DDR3 memory, now.

Connect the SP605 to your PC with the Ethernet cable and select “socket_xhd_fp12_1x8_fir_lms” SDK project:
In SDK, select:

Run -> Run Configuration ->Xilinc C/C++ ELF

Click on the “New launch configuration” in the Run configuration screen and the “socket_xhd_fp12_1x8_fir_lms” project executable is ready for download to DDR3 on the SP605 board via the jtag cable. Click on Run button to download the executable. The .elf file is downloaded via .jtag and the socket_xhd_fp12_1x8_fir_lms will be started.



Name: socket_xhd_bce_fp12_1x8_fir_lms Release

C/C++ Application: Release\socket_xhd_bce_fp12_1x8_fir_lms.elf

Project: socket_xhd_bce_fp12_1x8_fir_lms

Build (if required) before launching

Build configuration: Release

[✓] Select configuration using 'C/C++ Application'

Enable auto build   Disable auto build

[✓ Use workspace settings]   Configure Workspace Settings...

[✓] Connect process input & output to a terminal.

Figure 48: Run Release configuration of the socket_xhd_fp12_1x8_fir_lms demo application

The DVI display will present new screen layout related to the DSP active acoustic noise cancellation demo. See Figure 49.

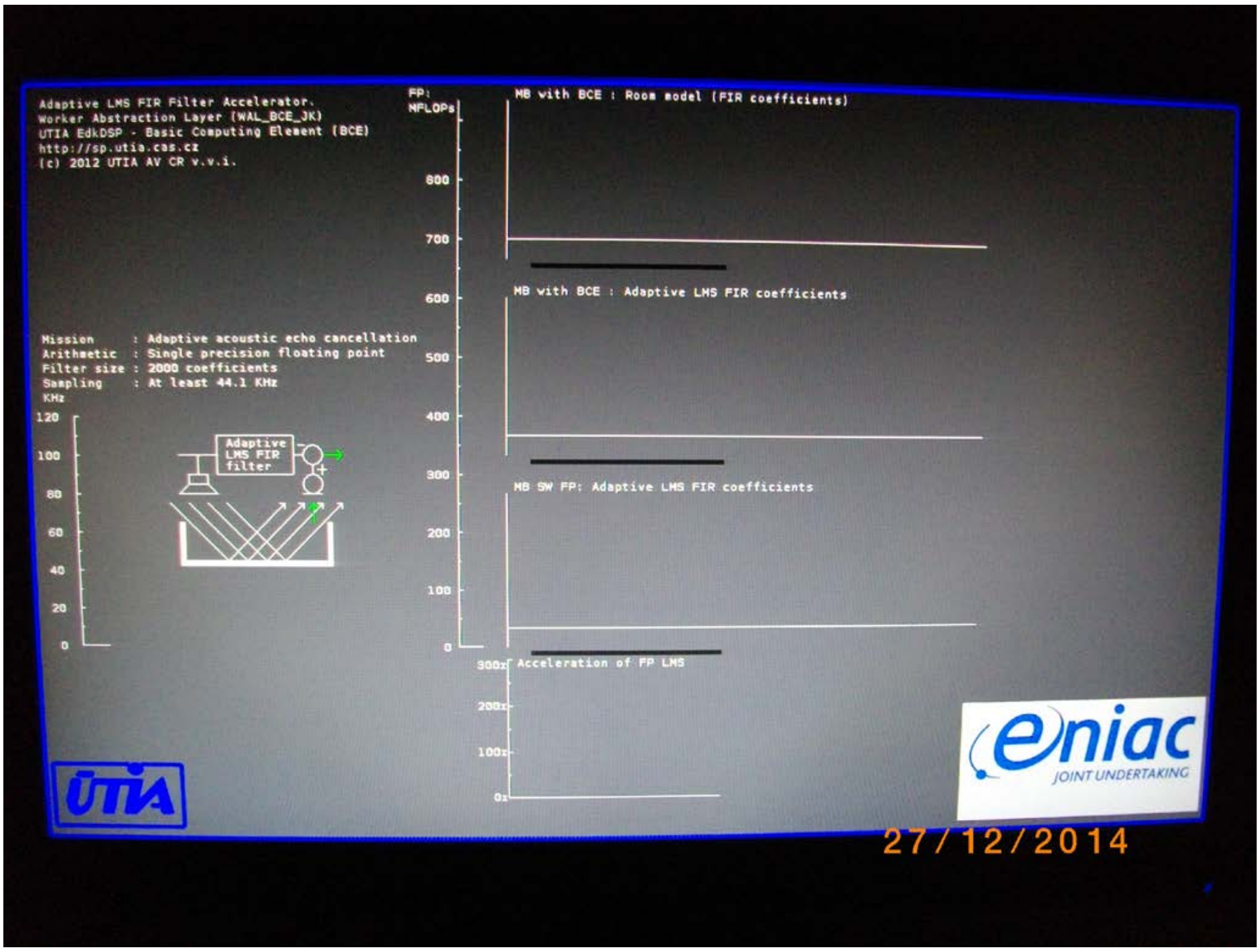

Adaptive LMS FIR Filter Accelerator.
Worker Abstraction Layer (WAL_BCE_JK)
UTIA EdkDSP - Basic Computing Element (BCE)
http://sp.utia.cas.cz
(c) 2012 UTIA AV CR v.v.i.
FP: MFLOPs
MB with BCE : Room model (FIR coefficients)
MB with BCE : Adaptive LMS FIR coefficients
MB SW FP: Adaptive LMS FIR coefficients
Mission : Adaptive acoustic echo cancellation
Arithmetic : Single precision floating point
Filter size : 2000 coefficients
Sampling : At least 44.1 KHz
KHz
Adaptive LMS FIR filter
Acceleration of FP LMS
ÚTIA
eniac
JOINT UNDERTAKING
27/12/2014

Figure 49: Run Release configuration of the socket_xhd_fp12_1x8_fir_lms demo application

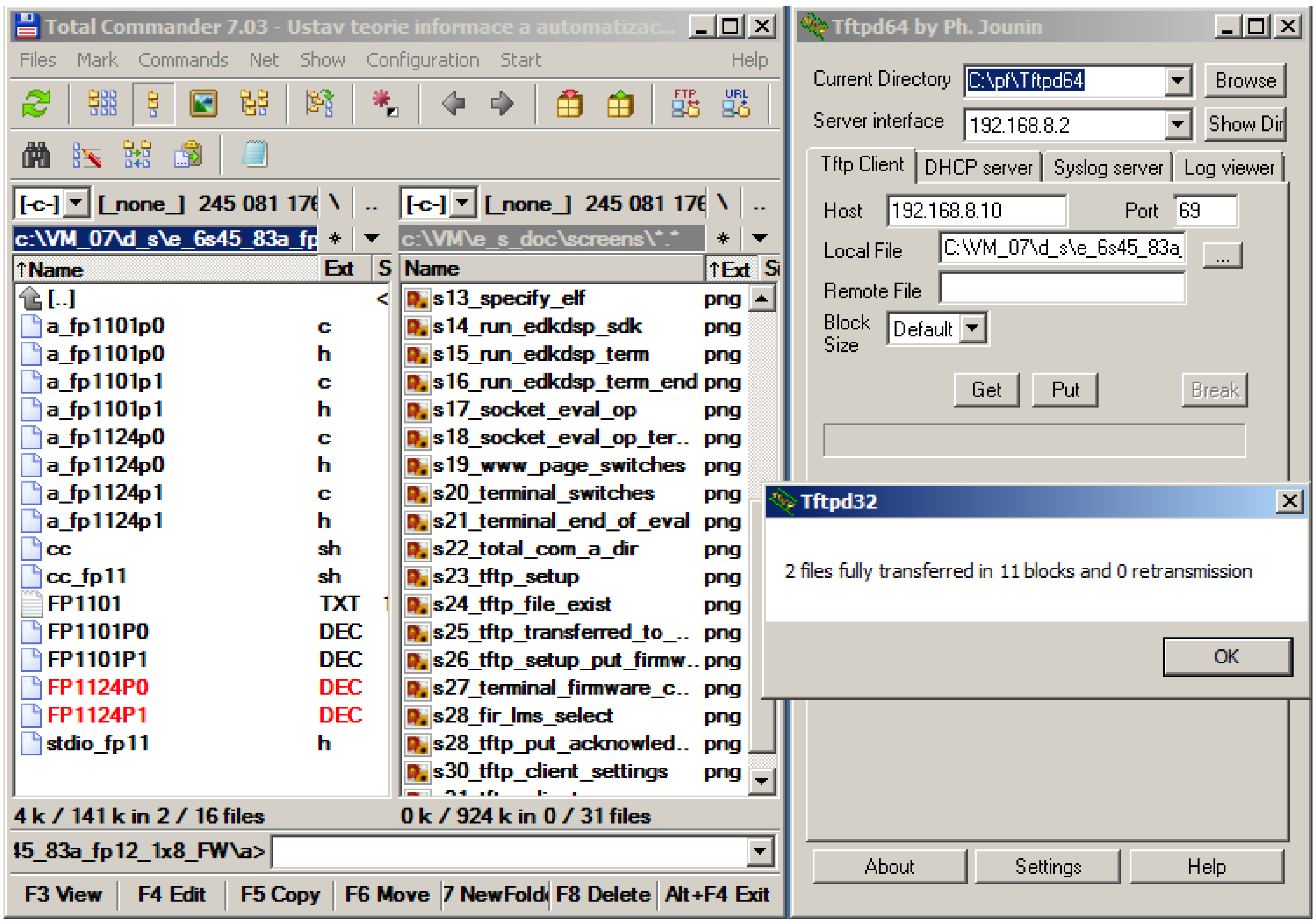
Open the Internet Explorer and test the “Update status” button to display the I/O status on the WWW and the MicroBlaze terminal. The Eniac logo (Figure 49) will be swapped to the Ideas project logo.

The socket_xhd_fp12_1x8_fir_lms is running in parallel with the http server and the TFTP server. The firmware compiled by the UTIA EdkDSP C compiler can be downloaded with the use of the TFTP client running on the PC.

In Windows, select the compiled firmware files (or use the initial .DEC files if you have not installed the compiler)
FP1124P0.DEC
FP1124P1.DEC
in the Windows directory
c:\VM_07\d_145_6s_plb\d_6s45_83a_fp12_1x8\SDK_Workspace\edkdsp_cc\a
It is corresponding to the Ubuntu directory
/mnt/cdrive/d_145_6s_plb/d_6s45_83a_fp12_1x8/SDK_Workspace/edkdsp_cc/a

In Windows, use Total Commander drag and drop them by mouse to the “Local File” section of the TFTP client. Both files are transferred to the ram-based file system of the SP605 board. Click OK on the info screen from the TFTP client. See Figure 50.

The original .DEC firmware files can be also used to avoid the installation of the UTIA EdkDSP C compiler.
Without this download, the default firmware present in the socket_xhd_fp12_1x8_fir_lms project will be used



Total Commander 7.03 - Ustav teorie informace a automatizac...

Files Mark Commands Net Show Configuration Start Help

[-c-] [_none_] 245 081 176 \ .. [-c-] [_none_] 245 081 176 \ ..

c:\VM_07\d_s\e_6s45_83a_fp *.* c:\VM\e_s_doc\screens*.*

```
↑Name              Ext        Name                          ↑Ext
[..]                          s13_specify_elf               png
a_fp1101p0         c          s14_run_edkdsp_sdk            png
a_fp1101p0         h          s15_run_edkdsp_term           png
a_fp1101p1         c          s16_run_edkdsp_term_end       png
a_fp1101p1         h          s17_socket_eval_op            png
a_fp1124p0         c          s18_socket_eval_op_ter..      png
a_fp1124p0         h          s19_www_page_switches         png
a_fp1124p1         c          s20_terminal_switches         png
a_fp1124p1         h          s21_terminal_end_of_eval      png
cc                 sh         s22_total_com_a_dir           png
cc_fp11            sh         s23_tftp_setup                png
FP1101             TXT        s24_tftp_file_exist           png
FP1101P0           DEC        s25_tftp_transferred_to_..    png
FP1101P1           DEC        s26_tftp_setup_put_firmw..    png
FP1124P0           DEC        s27_terminal_firmware_c..     png
FP1124P1           DEC        s28_fir_lms_select            png
stdio_fp11         h          s28_tftp_put_acknowled..      png
                              s30_tftp_client_settings      png
```

4 k / 141 k in 2 / 16 files   0 k / 924 k in 0 / 31 files

45_83a_fp12_1x8_FW\a>

F3 View F4 Edit F5 Copy F6 Move 7 NewFold F8 Delete Alt+F4 Exit

Tftpd64 by Ph. Jounin

Current Directory: C:\pf\Tftpd64 Browse

Server interface: 192.168.8.2 Show Dir

Tftp Client | DHCP server | Syslog server | Log viewer

Host: 192.168.8.10 Port: 69

Local File: C:\VM_07\d_s\e_6s45_83a

Remote File:

Block Size: Default

Get Put Break

About Settings Help

Tftpd32

2 files fully transferred in 11 blocks and 0 retransmission

OK

Figure 50: Download compiled firmware or the default compiled firmware to the SP605 board

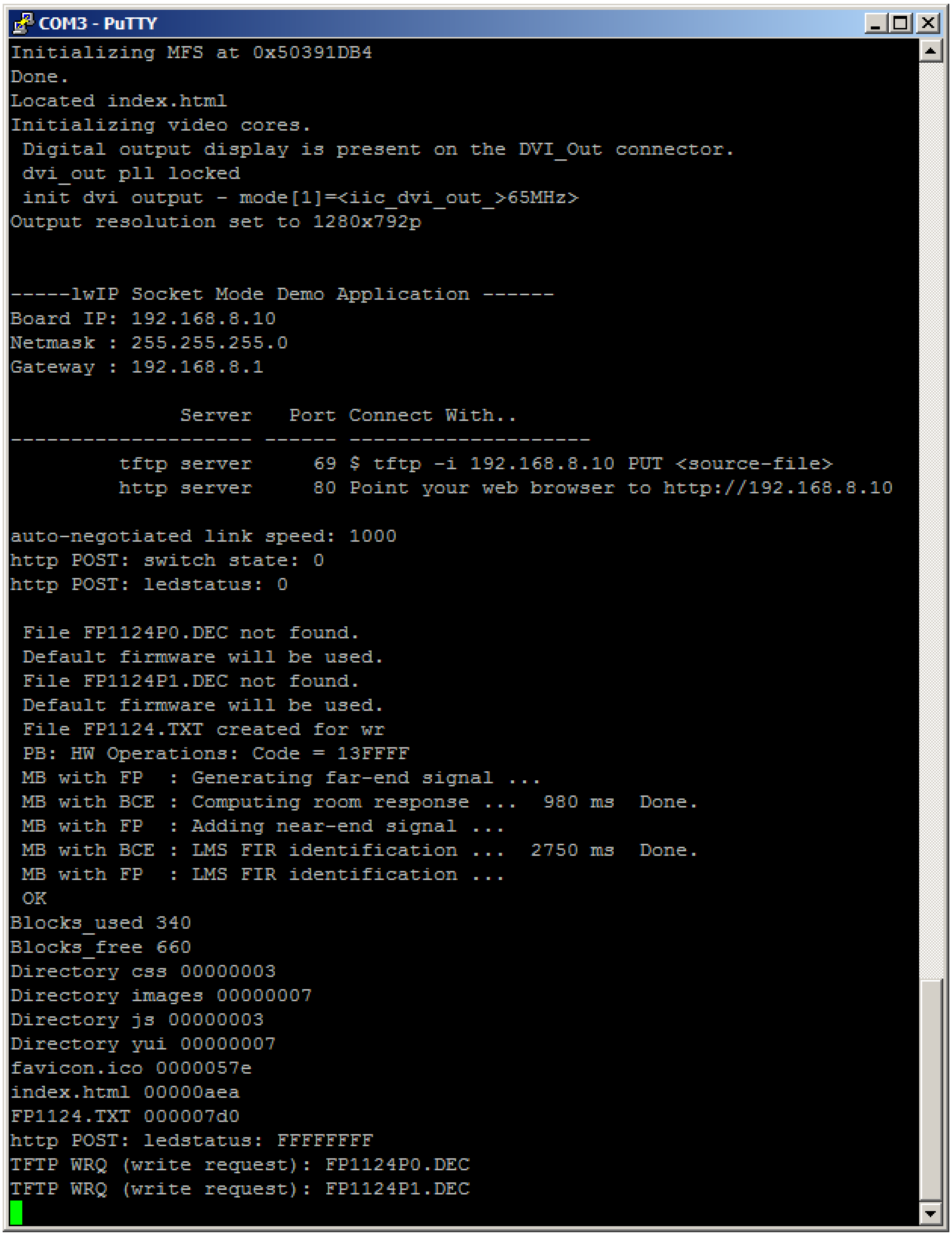
Both firmware programs correspond to 2 different modes of the accelerator.

- a_fp1124P0.c (FP1124P0.DEC) firmware is computing 2000 tap FIR filter on the 8xSIMD floating point accelerator bce_fp12_1x8_v1_40a with the MicroBlaze socket_xhd_fp12_1x8_fir_lms application support.
- a_fp1124P1.c (FP1124P1.DEC) firmware is computing 2000 tap adaptive LMS filter on the 8xSIMD floating point accelerator bce_fp12_1x8_v1_40a with the MicroBlaze socket_xhd_fp12_1x8_fir_lms application support.
- Both modes of computation (FIR/LMS) are swapped by the application in the runtime.

1. Figure 51 documents the initial execution of the project socket_xhd_fp12_1x8_fir_lms. The TFTP server has been started and demo executed with the default firmware first.

The TFTP download of firmware files FP1124P0.DEC and FP1124P1.DEC (initiated by user from Windows) as indicated in Figure 50 is reflected by the MicroBlaze TFTP server in the bottom part of the console. See Figure 51.

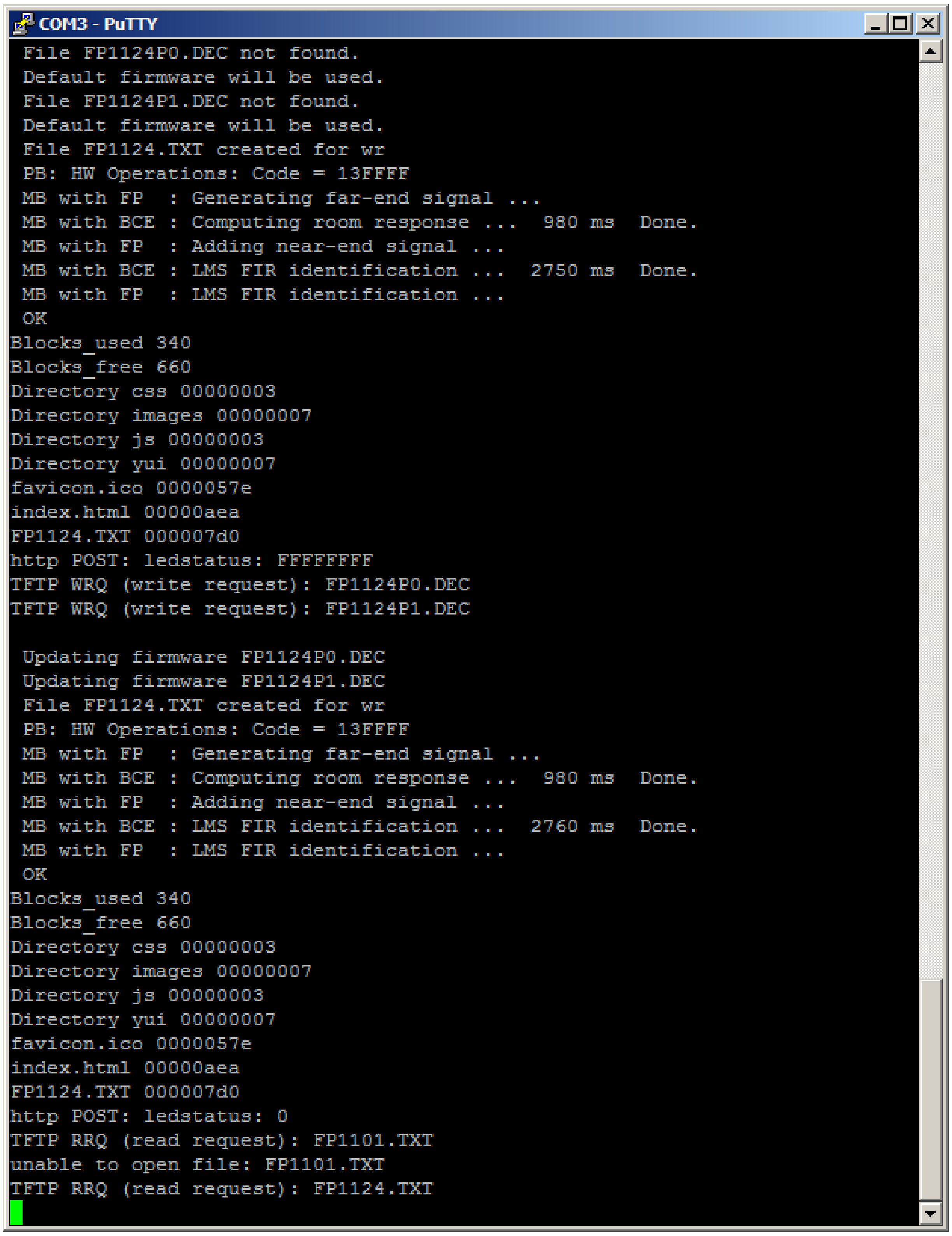
Figure 52 is presenting the console content after execution of the FIR/LMS run with the downloaded of firmware files FP1124P0.DEC and FP1124P1.DEC present in the RAM based file system of the SP605 board.



```
Initializing MFS at 0x50391DB4
Done.
Located index.html
Initializing video cores.
 Digital output display is present on the DVI_Out connector.
 dvi_out pll locked
 init dvi output - mode[1]=<iic_dvi_out_>65MHz>
Output resolution set to 1280x792p


-----lwIP Socket Mode Demo Application ------
Board IP: 192.168.8.10
Netmask : 255.255.255.0
Gateway : 192.168.8.1

              Server   Port Connect With..
-------------------- ------ --------------------
         tftp server     69 $ tftp -i 192.168.8.10 PUT <source-file>
         http server     80 Point your web browser to http://192.168.8.10

auto-negotiated link speed: 1000
http POST: switch state: 0
http POST: ledstatus: 0

 File FP1124P0.DEC not found.
 Default firmware will be used.
 File FP1124P1.DEC not found.
 Default firmware will be used.
 File FP1124.TXT created for wr
 PB: HW Operations: Code = 13FFFF
 MB with FP  : Generating far-end signal ...
 MB with BCE : Computing room response ...   980 ms   Done.
 MB with FP  : Adding near-end signal ...
 MB with BCE : LMS FIR identification ...  2750 ms   Done.
 MB with FP  : LMS FIR identification ...
 OK
Blocks_used 340
Blocks_free 660
Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1124.TXT 000007d0
http POST: ledstatus: FFFFFFFF
TFTP WRQ (write request): FP1124P0.DEC
TFTP WRQ (write request): FP1124P1.DEC
```

Figure 51: socket_xhd_fp12_1x8_fir_lms demo application terminal screen with new program files

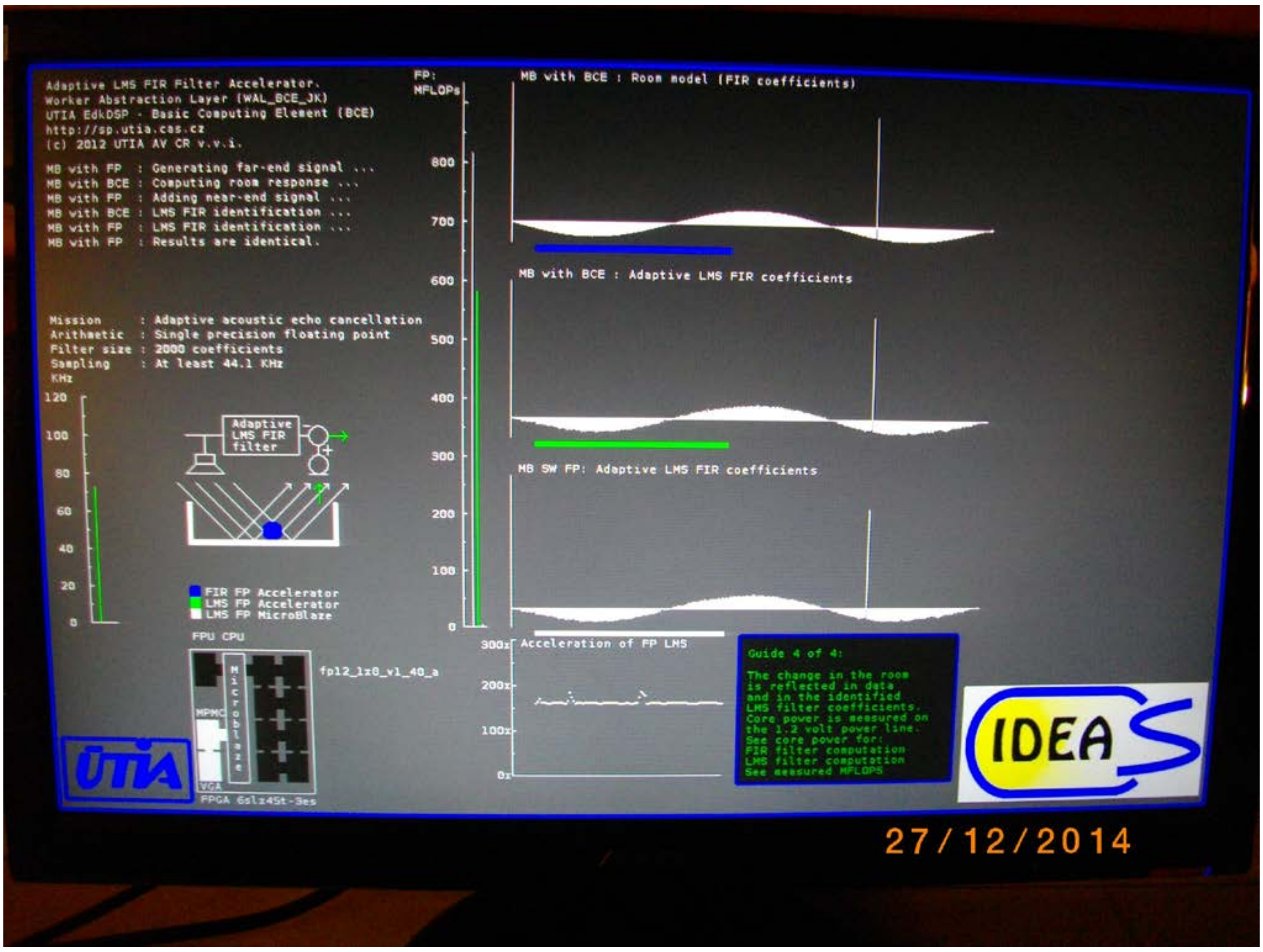
3.8 Getting of results from the EdkDSP filesystem back to PC



```
 File FP1124P0.DEC not found.
 Default firmware will be used.
 File FP1124P1.DEC not found.
 Default firmware will be used.
 File FP1124.TXT created for wr
 PB: HW Operations: Code = 13FFFF
 MB with FP  : Generating far-end signal ...
 MB with BCE : Computing room response ...  980 ms  Done.
 MB with FP  : Adding near-end signal ...
 MB with BCE : LMS FIR identification ...  2750 ms  Done.
 MB with FP  : LMS FIR identification ...
 OK
Blocks_used 340
Blocks_free 660
Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1124.TXT 000007d0
http POST: ledstatus: FFFFFFFF
TFTP WRQ (write request): FP1124P0.DEC
TFTP WRQ (write request): FP1124P1.DEC

 Updating firmware FP1124P0.DEC
 Updating firmware FP1124P1.DEC
 File FP1124.TXT created for wr
 PB: HW Operations: Code = 13FFFF
 MB with FP  : Generating far-end signal ...
 MB with BCE : Computing room response ...  980 ms  Done.
 MB with FP  : Adding near-end signal ...
 MB with BCE : LMS FIR identification ...  2760 ms  Done.
 MB with FP  : LMS FIR identification ...
 OK
Blocks_used 340
Blocks_free 660
Directory css 00000003
Directory images 00000007
Directory js 00000003
Directory yui 00000007
favicon.ico 0000057e
index.html 00000aea
FP1124.TXT 000007d0
http POST: ledstatus: 0
TFTP RRQ (read request): FP1101.TXT
unable to open file: FP1101.TXT
TFTP RRQ (read request): FP1124.TXT
```

Figure 52: socket_xhd_fp12_1x8_fir_lms demo after execution. Firmware program files are deleted.

See the measured MFLOP/s performance of the FIR filter and LMS filter computation on the DVI display screen. See Figure 53.



```
Adaptive LMS FIR Filter Accelerator.
Worker Abstraction Layer (WAL_BCE_JK)
UTIA EdkDSP - Basic Computing Element (BCE)
http://sp.utia.cas.cz
(c) 2012 UTIA AV CR v.v.i.

MB with FP  : Generating far-end signal ...
MB with BCE : Computing room response ...
MB with FP  : Adding near-end signal ...
MB with BCE : LMS FIR identification ...
MB with FP  : LMS FIR identification ...
MB with FP  : Results are identical.

Mission     : Adaptive acoustic echo cancellation
Arithmetic  : Single precision floating point
Filter size : 2000 coefficients
Sampling    : At least 44.1 KHz
```

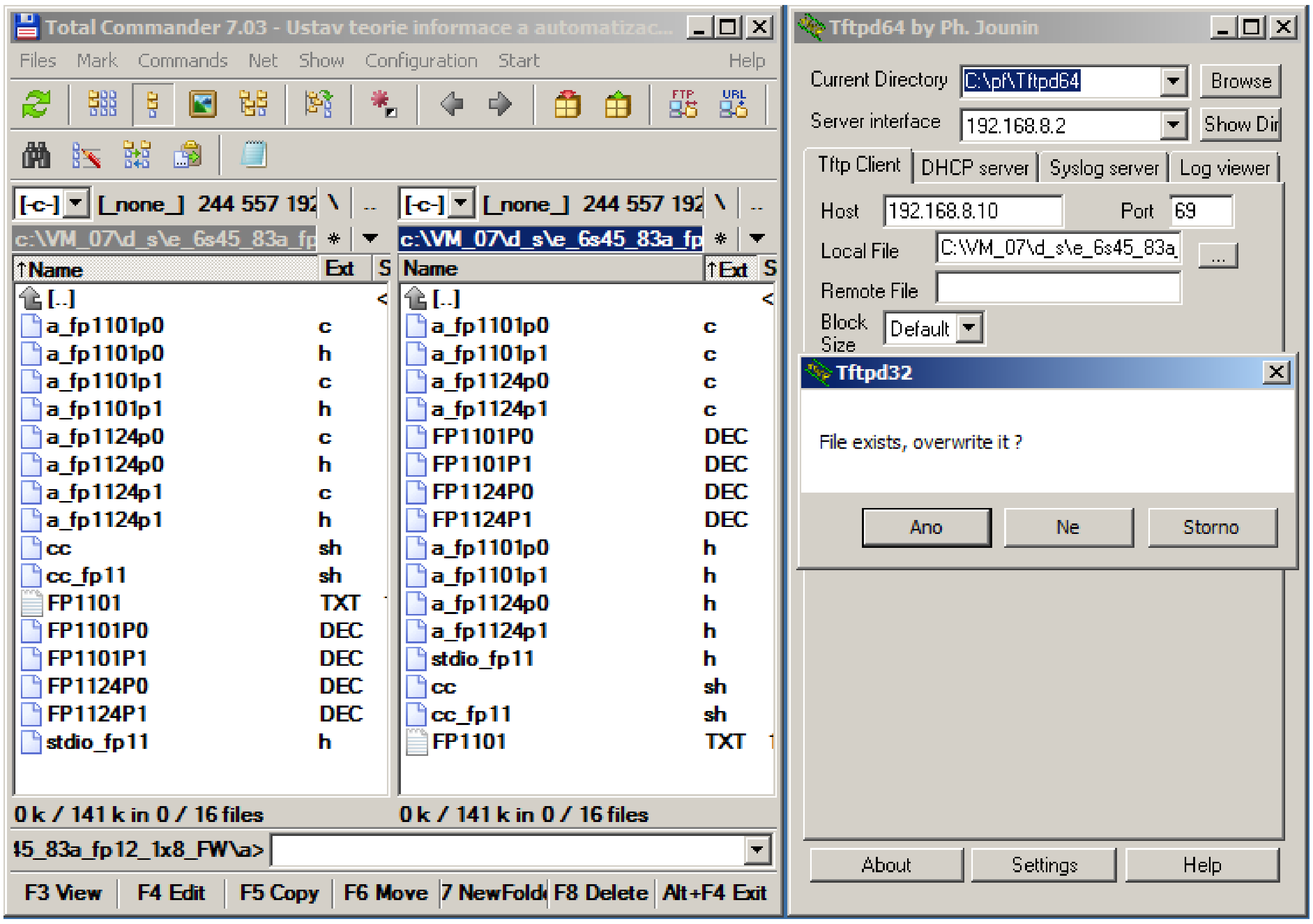
```
Guide 4 of 4:

The change in the room
is reflected in data
and in the identified
LMS filter coefficients.
Core power is measured on
the 1.2 volt power line.
See core power for:
FIR filter computation
LMS filter computation
See measured MFLOPS
```

Figure 53: Release configuration of the socket_xhd_fp12_1x8_fir_lms demo application

It is 820 MFLOP/s for the 2000 tap FIR filter computation and 580 MFLOP/s for the 2000 tap adaptive LMS filter identification. The LMS filter computation is accelerated 160x in comparison to the area optimized MicroBlaze processor with the optimized C code and HW floating point support. The Acceleraton of FP LMS graph in Figure 53 indicates sections with acceleration slightly higher than 160x. This corresponds to the additional parallel processing in the MicroBlaze as response to the parallel request from Ethernet explorer. MicroBlaze has been computing the decompression of the jpg logos into the RGB bitmap and therefore the parallel floating point computation of LMS filter identification has been slower and therefore the relative acceleration of the EdkDSP computation has been higher. See Figure 53.

The results of both computations (EdkDSP accelerator and MicroBlaze, only with its HW floating point unit) have been tested to be bit-exact identical. See Figure 53. It is important for the verification, that the computation of the EdkDSP accelerator can be reproduced and tested on the MicroBlaze with bit-exact identical floating point results.

The ascii log file FP1124.TXT created by execution of the demo can be uploaded to the PC with TFTP client served by the TFTP server on SP605 board. See Figure 54.


Total Commander 7.03 - Ustav teorie informace a automatizac...

Files Mark Commands Net Show Configuration Start Help

[-c-] [_none_] 244 557 192 \ ..

c:\VM_07\d_s\e_6s45_83a_fp

Name	Ext
[..]	
a_fp1101p0	c
a_fp1101p0	h
a_fp1101p1	c
a_fp1101p1	h
a_fp1124p0	c
a_fp1124p0	h
a_fp1124p1	c
a_fp1124p1	h
cc	sh
cc_fp11	sh
FP1101	TXT
FP1101P0	DEC
FP1101P1	DEC
FP1124P0	DEC
FP1124P1	DEC
stdio_fp11	h

0 k / 141 k in 0 / 16 files

Name	Ext
[..]	
a_fp1101p0	c
a_fp1101p1	c
a_fp1124p0	c
a_fp1124p1	c
FP1101P0	DEC
FP1101P1	DEC
FP1124P0	DEC
FP1124P1	DEC
a_fp1101p0	h
a_fp1101p1	h
a_fp1124p0	h
a_fp1124p1	h
stdio_fp11	h
cc	sh
cc_fp11	sh
FP1101	TXT

0 k / 141 k in 0 / 16 files

45_83a_fp12_1x8_FW\a>

F3 View F4 Edit F5 Copy F6 Move F7 NewFold F8 Delete Alt+F4 Exit

Tftpd64 by Ph. Jounin

Current Directory: C:\pf\Tftpd64

Server interface: 192.168.8.2

Tftp Client | DHCP server | Syslog server | Log viewer

Host: 192.168.8.10 Port: 69

Local File: C:\VM_07\d_s\e_6s45_83a_

Remote File:

Block Size: Default

Tftpd32

File exists, overwrite it ?

Ano Ne Storno

Figure 54: Upload the FP1124.TXT ascii log file from SP605 board to PC

3.9 Overview of other projects included in the package.

This application note has presented several projects working with the EdkDSP accelerator. The evaluation package includes several modifications of these projects, which are similar. These projects can be executed in similar or identical steps. These steps are not repeated in this application note.

- SDK SW projects with "_1x8_1x1_" use only the single data path of the 8xSIMD EdkDSP accelerator. These projects demonstrate the execution of the code on much smaller EdkDSP accelerator with single data path.
- SDK SW projects with "socket" in the name use the socket version of the LwIP library. It works on top of the XilKernel multitasking OS.
- SDK SW projects with "raw" in the name use the raw version of the LwIP library. It works on top of the RAW BSP without multitasking. The request coming from the Ethernet are processed with set of callback functions.
- SDK SW projects with "xhd" in the name use the digital DVI monitor with resolution 1280x792p60 and rgb888 pixels.
- SDK SW projects with "xga" in the name use the analogue RGB monitor with resolution 1280x792p60 and rgb888 pixels. The VGA D type 9pin connector can be connected to the DVI output connector of the SP605 board with the DVI-to-VGA adapter. This adapter is connecting 1:1 the analogue RGB and sync signals from the SP605 dual use DVI connector for the monitors with VGA D type 15pin connectors.

All SDK SW projects have similar user interaction as the SW projects described in this application note.

4. References

The ISE 14.5 MicroBlaze PLB-based part of the demonstration designs provided by UTIA in this evaluation package have been developed from the ISE 13.1 MicroBlaze PLB-based reference system for the Xilinx SP605 board presented in the Xilinx XAPP1026 (v3.2) October 28, 2012 [5].

Included with the application note [5] are ISE 14.3 AXI4-based reference systems for the Xilinx ML605 and SP605 FPGA Starter Kit boards. Also included are Zynq SoC reference systems for the Xilinx ZC-702 boards. Included with this application note are ISE 13.1AXI4-based and PLB-based reference systems for the Xilinx ML605, SP601 and SP605 FPGA Starter Kit boards.

[1] Spartan-6 FPGA SP605 Evaluation Kit.
http://www.xilinx.com/products/boards-and-kits/EK-S6-SP605-G.htm

[2] Hardware Setup Guide Sp605 Evaluation Kit
http://www.xilinx.com/support/documentation/boards_and_kits/xtp089.pdf

[3] SP605 Hardware User Guide
http://www.xilinx.com/support/documentation/boards_and_kits/ug526.pdf

[4] Getting Started with the Xilinx Spartan-6 FPGA SP605 Evaluation Kit
http://www.xilinx.com/support/documentation/boards_and_kits/ug525.pdf

[5] Xilinx XAPP1026 (v3.2) October 28, 2012. LightWeight IP (lwIP) Application Examples
Authors: Anirudha Sarangi and Stephen MacMahon
http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf

To access these reference systems, the user has to register at http://www.xilinx.com www portal and download these desigs from link:
https://secure.xilinx.com/webreg/clickthrough.do?cid=107743.zip

5. Deliverables for the evaluation version of EdkDSP SP605, PLB.

The enclosed **Evaluation version of the EdkDSP demonstrator for SP605, PLB package** of can be downloaded from UTIA www pages free of charge and used for evaluation of EdkDSP accelerators on Windows 7 (32 or 64 bit).

The evaluation package includes (on 1 DVD) or as www download package these deliverables:

16 precompiled designs with UTIA EdkDSP accelerators for Xilinx SP605 board, compiled in Xilinx EDK 14.5 and XPS (ISE) 14.5. The UTIA EdkDSP accelerators are compiled with HW limit on number of vector operations. The termination of the evaluation license is reported in advance by the demonstrator on the terminal as well as on the display. All designs are compiled with the Xilinx evaluation version of the XPS Soft-TEMAC 1 Gbit Ethernet controller. It includes the time restriction as defined by Xilinx XPS 14.5.

The evaluation package includes SDK 14.5 SW projects in source code for MicroBlaze. Projects support the family of UTIA EdkDSP accelerators for Xilinx SP605 board.

The evaluation package includes these libraries:

libwal.a	EdkDSP api (SDK 14.5, MicroBlaze) for EdkDSP accelerators, SP605, PLB.
librgb.a	EdkDSP api (SDK 14.5, MicroBlaze) for Xilinx PLB display controller.
libjpeg.a	EdkDSP api (SDK 14.5, MicroBlaze) for decompression of .jpg pictures.
libmfsimage.a	EdkDSP filesystem Image in form of a library. (It can be linked to the MicroBlaze .elf).

These libraries have no time restriction. The evaluation license is provided by UTIA only for the use with the family of UTIA EdkDSP accelerators designed for the Xilinx SP605 board. Source code of these libraries is owned by UTIA and it is not provided in this evaluation package.

The evaluation package includes these binary applications for Ubuntu:

edkdsppp	EdkDSP C pre-processor binary for Ubuntu (x86 PC) under the VMware Player.
edkdspcc	EdkDSP C compiler binary for Ubuntu (x86 PC) under the VMware Player.
edkdsppsm	EdkDSP ASM compiler binary for Ubuntu (x86 PC) under the VMware Player.
edkdspasm	EdkDSP ASM compiler binary for Ubuntu (x86 PC) under the VMware Player.

These binary applications have no time restriction. The user of the evaluation package has license from UTIA to use these utilities for compilation of the firmware for the Xilinx PicoBlaze6 processor inside of the UTIA EdkDSP accelerators in the 16 precompiled designs for the Xilinx SP605 board with PLB bus. The source code of these applications is owned by UTIA and it is not provided in the evaluation package.

The evaluation package includes demonstration firmware in C source code for the Xilinx PicoBlaze6 processor in the family of UTIA EdkDSP accelerators for the Xilinx SP605 board.

The evaluation package also includes compiled versions of this firmware. The compiled firmware files can be downloaded into the UTIA EdkDSP accelerators for the Xilinx SP605 board without the need to install of the Ubuntu (x86 PC) image under the VMware Player.

On email request to kadlec@utia.cas.cz , UTIA will send 2 DVD CDs (8GB) with the Ubuntu (x86 PC) image for the VMware Player free of charge.

6. Deliverables for the release version of EdkDSP SP605, PLB.

The **release version of the EdkDSP demonstrator for SP605, PLB package** can be ordered from UTIA AV CR, v.v.i., by email request for quotation to kadlec@utia.cas.cz. UTIA will provide quotation by email. After the confirmed order (based on the UTIA AV CR, v.v.i., email quotation) received by email to kadlec@utia.cas.cz. UTIA AV CR, v.v.i. will deliver (by standard mail) to the customer the printed version of this application note together with 3 DVDs with deliverables described in this section. UTIA AV CR, v.v.i., will send to the customer (by email) and by the standard mail the invoice for:

Release version of the EdkDSP demonstrator for SP605, PLB package (without VAT)	**400,00 Eur**

The release package includes this application note and the EdkDSP DVD with these deliverables:

16 precompiled designs with UTIA EdkDSP accelerators for Xilinx SP605 board, compiled in Xilinx EDK 14.5 and XPS (ISE) 14.5. The UTIA EdkDSP accelerators included in these designs are compiled with **no HW limit on number of vector operations.** All these designs are compiled with the commercial Xilinx XPS Soft-TEMAC 1 Gbit Ethernet controller with **no time restriction.** Therefore, all these precompiled designs of the release package run on SP605 without limitations of the evaluation package.

The release package includes source code of all 16 EDK design projects with UTIA EdkDSP accelerators provided as these PLB netlist pcores generated in Xilinx ISE 14.5:

bce_fp11_1x8_0_plbw_v1_10_a
bce_fp11_1x8_0_plbw_v1_20_a
bce_fp11_1x8_0_plbw_v1_30_a
bce_fp11_1x8_0_plbw_v1_40_a
bce_fp12_1x8_0_plbw_v1_10_a
bce_fp12_1x8_0_plbw_v1_20_a
bce_fp12_1x8_0_plbw_v1_30_a
bce_fp12_1x8_0_plbw_v1_40_a

These UTIA EdkDSP PLB netlist pcores have **no HW limit on number of vector operations.** The user of the release package has license from UTIA to integrate these netlists into its own ISE 14.5 designs and to compile them to unlimited number of bit-streams for the Xilinx Spartan6 FPGA designs with the MicroBlaze and the PLB bus. This license has no time restriction. The source code of the EdkDSP accelerators is IP owned by UTIA and it is not provided in the release package to the user.

The user compiling own designs using the Xilinx XPS Soft-TEMAC 1 Gbit Ethernet controller will need to get the license for the TEMAC core from Xilinx or use the Xilinx Ethernet-lite core. Resulting designs will have no limitations of the evaluation cores.

The release package includes SDK 14.5 SW projects in source code for MicroBlaze as described in this application note. Projects support the family of UTIA EdkDSP accelerators for Xilinx SP605 board.

The release package includes these libraries:

libwal.a	EdkDSP api (SDK 14.5, MicroBlaze) for EdkDSP accelerators, SP605, PLB.
librgb.a	EdkDSP api (SDK 14.5, MicroBlaze) for Xilinx PLB display controller.
libjpeg.a	EdkDSP api (SDK 14.5, MicroBlaze) for decompression of .jpg pictures.

These libraries have no time restriction. User has license from UTIA for linking of these libraries to MicroBlaze applications with UTIA EdkDSP accelerators for SP605, PLB bus. The source code of these libraries is owned by UTIA and it is not provided in the release package.

The release package includes library:

mfsimage	EdkDSP filesystem Image in form of a library. (It can be linked to the MicroBlaze .elf).

Source code for generation of the **libmfsimage.a** by the user is included in the release package.

The release package includes these binary applications:

edkdsppp	EdkDSP C pre-processor binary for Ubuntu (x86 PC) under the VMware Player.
edkdspcc	EdkDSP C compiler binary for Ubuntu (x86 PC) under the VMware Player.
edkdsppsm	EdkDSP ASM compiler binary for Ubuntu (x86 PC) under the VMware Player.
edkdspasm	EdkDSP ASM compiler binary for Ubuntu (x86 PC) under the VMware Player.

These binary applications have no time restriction, and the user of the release package has license from UTIA to use these utilities for compilation of the firmware for the Xilinx PicoBlaze6 processor inside of the UTIA EdkDSP accelerators provided by the release package for the Xilinx SP605 board with PLB bus. The source code of these applications is owned by UTIA and it is not provided in the release package.

The release package includes demonstration firmware in C source code for the Xilinx PicoBlaze6 processor in the family of UTIA EdkDSP accelerators for the Xilinx SP605 board. The release package also includes compiled versions of this firmware. The compiled firmware files can be downloaded into the UTIA EdkDSP accelerators for the Xilinx SP605 board designs with PLB bus, without the need to install of the Ubuntu (x86 PC) image under the VMware Player.

The release package deliverables also includes two DVDs with the Ubuntu (x86 PC) image for the VMware Player (free of charge). This image is provided to ease the installation of the UTIA EdkDSP C compiler on Windows 7 (32bit or 64bit) in the VMware Player.

Any and all legal disputes that may arise from or in connection with the use, intended use of or license for the software provided hereunder shall be exclusively resolved under the regional jurisdiction relevant for UTIA AV CR, v. v. i. and shall be governed by the law of the Czech Republic.

Disclaimer