Application Note

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

Compact Zynq System with SW-defined Floating-Point 8xSIMD EdkDSP Accelerator

Trenz Electronic TE0720-2IF Module and TE0706-02 Carrier Board

Jiří Kadlec, Zdeněk Pohl, Lukáš Kohout
kadlec@utia.cas.cz , xpohl@utia.cas.cz , kohoutl@utia.cas.cz
phone: +420 2 6605 2216
UTIA AV CR, v.v.i.

Revision history:

Rev.	Date	Author	Description
1	12.01.2018	Jiří Kadlec	Initial internal draft for the Productive 4.0 consortium meeting 17-18.1.2018 (Lisabon, PT).
2	30.01.2018	Jiří Kadlec	Demonstrator description prior to the Productive 4.0 project conference 6-7.3.2018 (Athens, GR)
3	14.03.2018	Jiří Kadlec	Fixed function names in table 6. Released for public download.

Acknowledgements:

This work has been partially supported by ECSEL JU project Productive4.0 No. 737459.

Table of Contents

Table of Figures

Table of Tables

1. EdkDSP IP Core - Introduction

This report describes design of compact HW system based on Zynq all programmable 28nm chip with two Arm A9 processors and programmable logic area. System is optimised for Ethernet connected computing nodes serving for industrial automation, local data processing and data communication. The documented HW architecture is one of candidates for wider use within the ECSEL Productive 4.0 project for the edge computing node in the Industry 4.0 solutions.

The demonstrated Zynq system includes the run-time reprogrammable 8xSIMD EdkDSP IP core. It combines the MicroBlaze and the floating point single Instruction multiple data (SIMD) data flow unit (DFU). The SIMD DFU is controlled by a run-time reprogrammable finite state machine implemented by Xilinx PicoBlaze6 8 bit controller with dedicated embedded (on Zynq executed) C compiler.

The application note describes the installation of the HW system, the SW API, algorithmic implementation and mapping to the 8xSIMD EdkDSP IP. Presented HW system is also compatible with the Xilinx SDSoC 2017.1 design environment. The SDSoC is supporting automated compilation of user-defined C/C++ ARM functions into HW accelerators with data movers (zero-copy, DMA, SG-DMA) and the automated integration of generated accelerators with the ARM Linux or stand-alone operating systems.

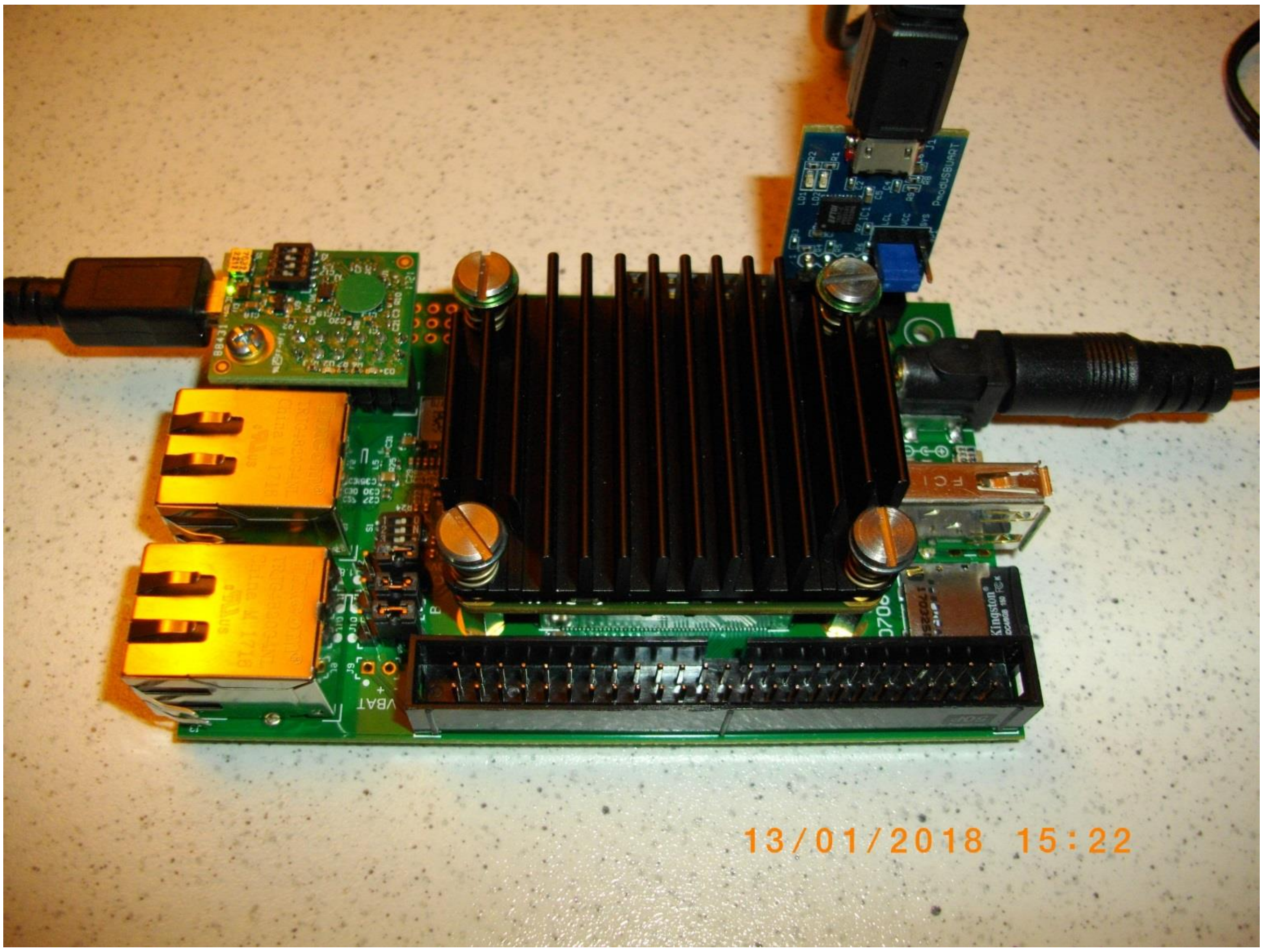


Figure 1: TE0706-02 carrier board with TE0720-2IF Zynq module

2. Implementation Details

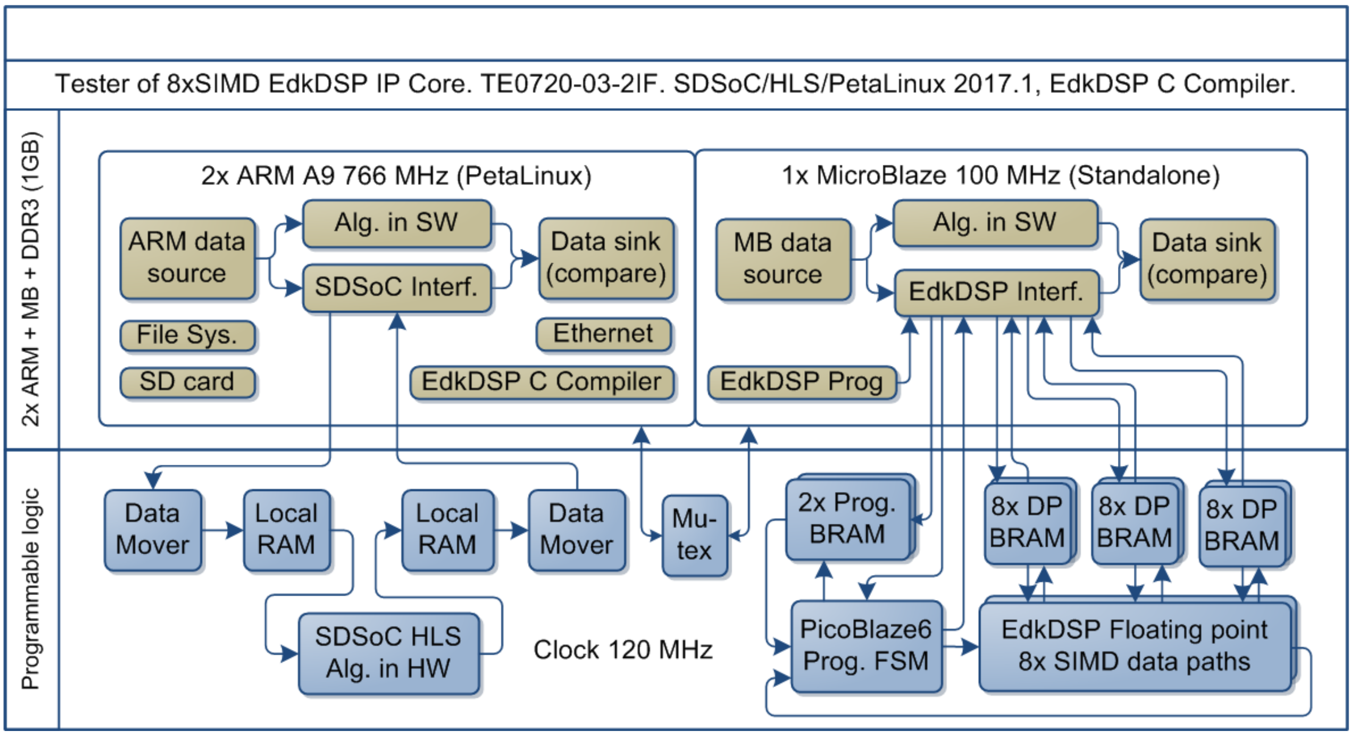


Figure 2: SDSoC compatible Zynq system with 8xSIMD EdkDSP floating point accelerator.

Evaluation system

The 8xSIMD EdkDSP IP Core is programmed, evaluated and debugged in HW Xilinx Zynq module TE0720-IF [1]. The 28nm Xilinx Zynq device xc7z020-2I has two 32 bit ARM Cortex A9 processors operating at 766 MHz and single MicroBlaze 32 bit soft core processor operating at 100 MHz. The Zynq programmable logic area is used for one 8xSIMD EdkDSP IP (operating at 120 MHz) and also for demonstration of compatibility of the EdkDSP IP with examples of Xilinx SDSoC 2017.1 HW accelerators.

The EdkDSP IP Core is 8xSIMD floating point accelerator. It is reprogrammable in runtime by change of firmware of a PicoBlaze6 8bit controller. The PicoBlaze6 controller schedules vector operations performed in the 8xSIMD floating point data paths. The PicoBlaze6 controller serves as re-programmable finite state machine (FSM). It is programmed by firmware compiled by an EdkDSP C Compiler and Assembler. The EdkDSP C Compiler and Assembler are implemented as application programs running on the embedded PetaLinux 2017.1 operating system. The 8xSIMD EdkDSP IP is controlled by the 32bit MicroBlaze processor. MicroBlaze runs programs from the DDR3 memory. The DDR3 is interfaced by an Instruction and Data cache (32k x 32bit) with HP0 AXI interface. The 8xSIMD EdkDSP IP is connected to the MicroBlaze by local dual-ported memories. MicroBlaze implements data communication from DDR3 to 8xSIMD EdkDSP dual-ported memories in software. This communication is performed in parallel with the 8xSIMD parallel floating point computation in the 8xSIMD EdkDSP IP.

Parameters of the 8xSIMD EdkDSP IP core

8x SIMD EdkDSP floating point accelerator IP core supports 8xSIMD vector floating point operations performed from/to dual-ported BRAMs A, B , Z. Each dual-ported BRAM has 8 parallel layers of 1024 32bit words. The set of supported floating point operations is different for different grades [10|20|30|40] of the 8xSIMD EdkDSP accelerator IPs. The supported floating point operations are summarised in *Table 1*.

The accelerator **bce_fp12_1x8_0_axiw_v1_10** is area **optimized** and supports only data transfers and vector floating point operations FPADD, FPSUB in 8 SIMD data paths.

The accelerator **bce_fp12_1x8_0_axiw_v1_20** performs identical operations as bce_fp12_1x8_0_axiw_v1_10 plus the vector floating point MAC operations in 8 SIMD data paths. MAC is supported for length of vectors 1 up to 10. This accelerator is optimized for applications like floating point matrix multiplication with one row and column dimensions <= 10.

The accelerator **bce_fp12_1x8_0_axiw_v1_30** supports identical operations as bce_fp12_1x8_0_axiw_v1_20 plus HW-accelerated computation of the floating point vector-by-vector dot-product operators 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 efficient in case of floating point matrix by matrix multiplications, where one of the dimensions is large (in the range from 11 to 250).

The accelerator **bce_fp12_1x8_0_axiw_v1_40** supports identical operations as bce_fp12_1x8_0_axiw_v1_30 plus an additional HW support of dot product. It is computed in 8 data paths with HW-supported wind-up into single scalar result propagated into all SIMD planes.

All **bce_fp12_1x8_0_axiw_v1_[10|20|30|40]** accelerators support single data path for pipelined, floating-point division operations with vector operands taken from the first SIMD plain and the result is propagated into all 8 SIMD plains.

All **bce_fp12_1x8_0_axiw_v1_[10|20|30|40]** accelerators are suitable for applications like adaptive normalised LMS and NLMS filters and square root free versions of adaptive RLS QR filters and adaptive RLS LATTICE filters.

Table 1: (8xSIMD) EdkDSP bce_fp12_1x8_40 accelerator vector operations.

Name in MicroBlaze C value (dec)	**8xSIMD Floating point Operation**
WAL_BCE_JK_VVER = 0	Return capabilities of the (8xSIMD) EdkDSP accelerator
WAL_BCE_JK_VZ2A = 1	8xSIMD copy $a_m[i]$ <= $z_m[j]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VB2A = 2	8xSIMD copy $a_m[i]$ <= $b_m[j]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VZ2B = 3	8xSIMD copy $b_m[i]$ <= $z_m[j]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VA2B = 4	8xSIMD copy $b_m[i]$ <= $a_m[j]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VADD = 5	8xSIMD add $z_m[i]$ <= $a_m[j]$ + $b_m[k]$]; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VADD_BZ2A = 6	8xSIMD add $a_m[i]$ <= $b_m[j]$ + $z_m[k]$]; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VADD_AZ2B = 7	8xSIMD add $b_m[i]$ <= $a_m[j]$ + $z_m[k]$]; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VSUB = 8	8xSIMD sub $z_m[i]$ <= $a_m[j]$ - $b_m[k]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VSUB_BZ2A = 9	8xSIMD sub $a_m[i]$ <= $b_m[j]$ - $z_m[k]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VSUB_AZ2B = 10	8xSIMD sub $b_m[i]$ <= $a_m[j]$ - $z_m[k]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VMULT = 11	8xSIMD mult $z_m[i]$ <= $a_m[j]$ * $b_m[k]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VMULT_BZ2A = 12	8xSIMD mult $a_m[i]$ <= $b_m[j]$ * $z_m[k]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VMULT_AZ2B = 13	8xSIMD mult $b_m[i]$ <= $a_m[j]$ * $z_m[k]$; m=1..8 IP core: 10,20,30,40
WAL_BCE_JK_VPROD = 14	8xSIMD vector products: IP core: 30,40 $z_m[i]$ <= a_m'[j..j+nn]*b_m[k..k+nn]; m=1..8; nn range 1..255
WAL_BCE_JK_VMAC = 15	8xSIMD vector MACs: IP core: 20,30,40

		z_m[i..i+nn] <= z_m[i..i+nn] + a_m[j..j+nn] * b_m[k..jk+nn]; nn range 1..13
WAL_BCE_JK_VMSUBAC	**= 16**	8xSIMD vector MSUBACs IP core: 20,30,40 z_m[i..i+nn] <= z_m[i..i+nn] - a_m[j..j+nn] * b_m[k..jk+nn]; nn range 1..13
WAL_BCE_JK_VPROD_S8	**= 17**	8xSIMD vector product (extended) IP core: 40 z_m[i] <= ((a_1'[j..j+nn]*b_1[k..k+nn]+a_2'[j..j+nn]*b_2[k..k+nn]) + (a_3'[j..j+nn]*b_3[k..k+nn]+a_4'[j..j+nn]*b_4[k..k+nn])) + ((a_5'[j..j+nn]*b_5[k..k+nn]+a_6'[j..j+nn]*b_6[k..k+nn]) + (a_7'[j..j+nn]*b_7[k..k+nn]+a_8'[j..j+nn]*b_8[k..k+nn])); m=1..8; nn range 1..255
WAL_BCE_JK_VDIV	**= 20**	vector division (extended) IP core: 10,20,30,40 z_m[i] <= a_1[j] / b_1[k]; m=1..8

Ports of the 8xSIMD EdkDSP accelerator

• bce_atoa[0:9]	Memory A address (addressing 1024 32 bit floating point values)
• bce_atob[0:9]	Memory B address (addressing 1024 32 bit floating point values)
• bce_atoz[0:9]	Memory Z address (addressing 1024 32 bit floating point values)
• bce_done[0:7]	Vector operation in progress or finished
• bce_led4b[0:3]	4 bit output, intended for led signalling. (Unconnected in the evaluation design).
• bce_mode[0:3]	Mode of the communication protocol PicoBlaze6 - MicroBlaze
• bce_op[0:7]	Vector operation to be performed.
• bce_port[0:7]	8 bit output port. (Unconnected in the evaluation design).
• bce_port_id[0:7]	8 bit output External port address. Address space [0x0 ... 0x1F] is reserved for optimized construction of the VLIW instruction to the 8xSIMD vector processing unit of the EdkDSP. Address space [0x20 ... 0xFF] can be used by the user.
• bce_port_wr	1 bit output. Write strobe for write of 8 bit data to the external port address.
• bce_r_pb	1 bit output. Reset of the PicoBlaze6.
• bce_we	1 bit output. Write strobe signals start of execution of a VLIW instruction by the 8xSIMD vector processing unit of the EdkDSP.
• bce_dip4b[0:3]	4bit input (Connected to a constant in the evaluation design).
• Bce_gpi8b[0:7]	8bit input (Connected to a constant in the evaluation design).

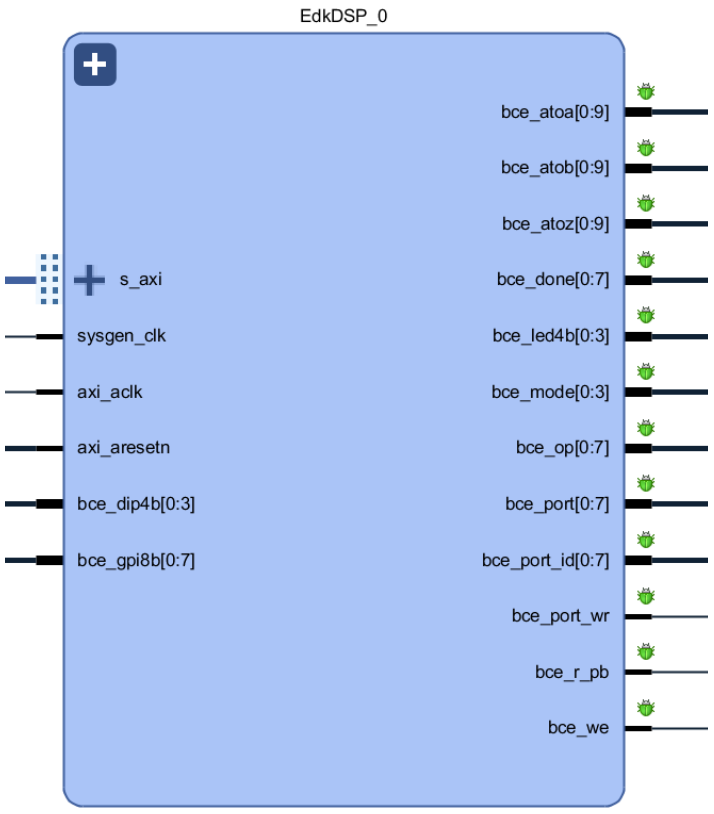


Figure 3: 8xSIMD EdkDSP floating point accelerator IP core.

Interface of the 8xSIMD EdkDSP IP to the MicroBlaze processor

The EdkDSP IP core is connected to the 100 MHz MicroBlaze processor via the 100 MHz 32bit AXI lite bus represented by port **s_axi** , 100 MHz clock input **axi_aclk** and an asynchronous reset signal **axi_aresetn**. See *Figure 3*.

The debug ports are used for the real-time visualisation, debug and analysis of the computation implemented inside of the 8xSIMD data flow unit (DFU) of the (8xSIMD) EdkDSP accelerator IP. This makes easier to debug the compiled PicoBlaze6 firmware code. The implemented in circuit logic analyser (ILA) debug probes can capture 8096 data samples and provide visibility for the auto-generated addresses and for the detailed schedule of vector operation in the 8xSIMD EdkDSP IP core. See *Figure 3*.

Figure 4 presents connection of the two parts of the 8xSIMD EdkDSP IP core.

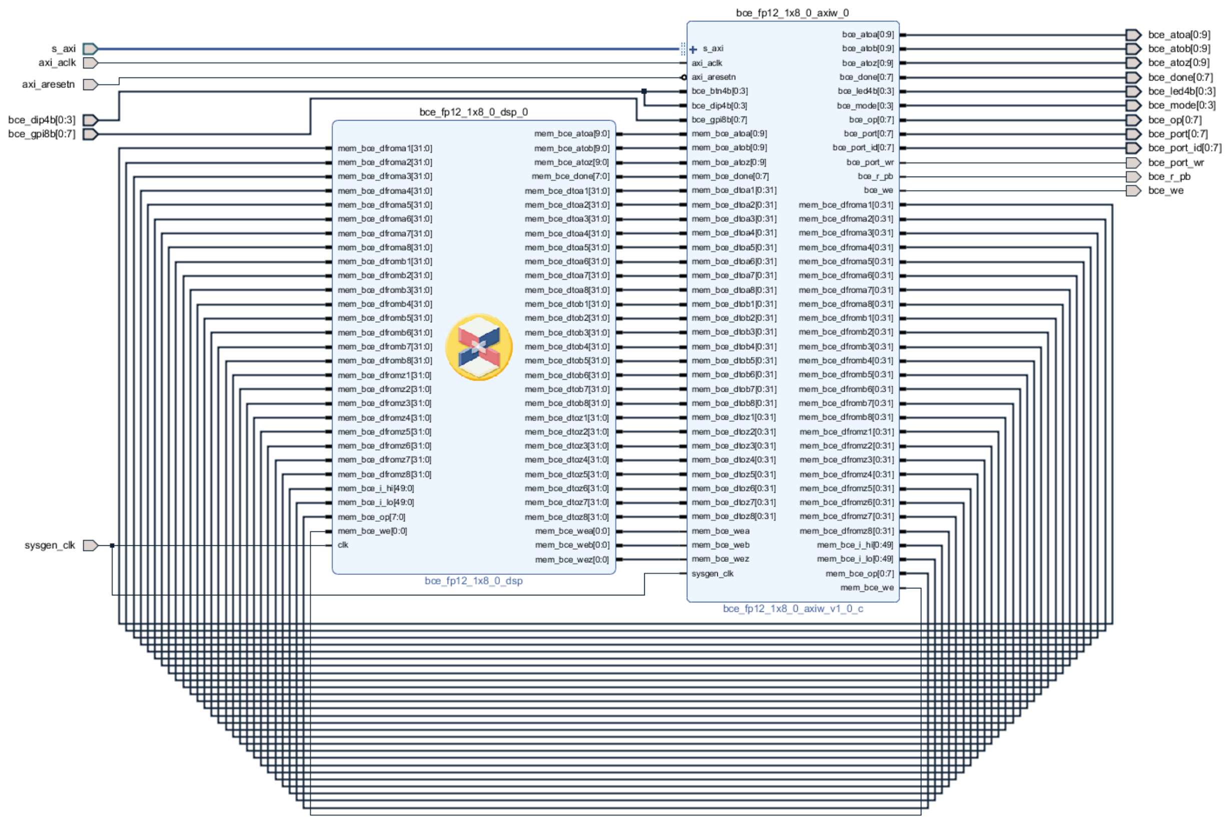


Figure 4: Internal details of (8xSIMD) EdkDSP floating point accelerator IP core.

All bce_fp12_1x8_0_axiw_v1_[10|20|30|40] accelerators versions have identical Edk IP part.

The DSP part has identical ports and connectivity (see

Figure 4) for all bce_fp12_1x8_0_axiw_v1_[10|20|30|40] accelerators versions.

The Edk part of the EdkDSP floating point accelerator IP core **bce_fp12_1x8_0_axiw_v1_0_c** includes inside the PicoBlaze6 controller, its program memories P0 and P1 and the 8xSIMD dual-ported block-ram memories 8xA, 8xB and 8xZ designed for parallel access. The **bce_fp12_1x8_0_axiw_v1_0_c** IP is designed in the Xilinx System Generator 14.5 and ported to the Vivado 2017.1 compatible IP core. The PicoBlaze6 firmware executes C code and supports C constructs like loops, while, if, else, function calls etc.

The first of the two ports of all block-rams are accessed by the MicroBlaze as memory via the Axi-lite bus.

- The second of the two ports of both program memories P0 and P1 are connected to the PicoBlaze6 controller.
- The second of the two ports of all data memories 8xA, 8xB and 8xZ are connected to the floating point data paths of the data flow unit (DFU) unit and support parallel access.

The DFU **bce_fp12_1x8_0_dsp** is designed in the Xilinx System Generator for DSP 2017.1. It contains 8 pipelined floating point ADD units, 8 pipelined floating point MULT units and one pipelined floating point DIV unit. The DFU supports all vector operations defined in *Table 1*.

- The 100bit VLIW instruction is transferred in two 50bit ports **mem_bce_i_lo** and **mem_bce_i_hi**. The VLIW instruction is set by dedicated PicoBlaze6 output ports. See *Table 2* .
- The 8xSIMD data flow unit executes 8xSIMD floating point operations defined in *Table 1*.
- The concrete 8xSIMD operation is defined by the PicoBlaze6 DFU_OP 8bit output register driving the **mem_bce_op** port of the **bce_fp12_1x8_0_axiw_v1_0_c** IP. The transfer of the complete VLIW instruction (100+8 bits) is triggered by the write strobe signal **mem_bce_we** . It is activated by PicoBlaze6 program write of the 8xSIMD operation DFU_OP. See *Table 2* .

The 8xSIMD data flow unit (DFU) indicates end of the operation in the 8bit output port **mem_bce_done**. PicoBlaze6 program can execute few instructions in parallel to the 8xSIMD operation defined in DFU_OP. End of the 8xSIMD operation is detected by the PicoBlaze6 program by reading of the input 8bit port **mem_bce_done**. PicoBlaze6 firmware defines the sequence of VLIW instructions for the 8xSIMD DFU unit by its dedicated output registers. PicoBlaze6 addresses of these dedicated output registers are listed in *Table 2* .

Table 2: PicoBlaze6 ports forming VLIW instruction for the 8xSIMD EdkDSP data flow unit.

PicoBlaze6 registers used for definition of the 100 bit wide VLIW instruction for the EdkDSP Data Flow Unit	Format [msb..lsb]	VLIW [2x 50bit] mem_bce_i_hi mem_bce_i_lo	Description of sections defined in the VLIW instruction for the EdkDSP Data Flow Unit
[00b, DFU_CNT]	[2bit,8bit]	10 bit [49..40]	Number of 8xSIMD steps (0 .. 255)
[00b, DFU_Z_INC]	[2bit,8bit]	10 bit [39..30]	Auto increment of Z address (0 .. 255)
[DFU_Z_MEM_BANK, DFU_Z_MEM_SADDR]	[2bit,8bit]	10 bit [29..20]	Set Z address after auto incr overflow
[DFU_Z_MEM_BANK, DFU_Z_MEM_ADDR]	[2bit,8bit]	10 bit [19..10]	Initial Z address
[00b, DFU_B_INC]	[2bit,8bit]	10 bit [09..00]	Auto increment of B address (0 .. 255)
[DFU_B_MEM_BANK, DFU_B_MEM_SADDR]	[2bit,8bit]	10 bit [49..40]	Set B address after auto incr overflow
[DFU_B_MEM_BANK, DFU_B_MEM_ADDR]	[2bit,8bit]	10 bit [39..20]	Initial B address
[00b, DFU_A_INC]	[2bit,8bit]	10 bit [29..20]	Auto increment of A address (0 .. 255)
[DFU_A_MEM_BANK, DFU_A_MEM_SADDR]	[2bit,8bit]	10 bit [19..10]	Set A address after auto incr overflow
[DFU_A_MEM_BANK, DFU_A_MEM_ADDR]	[2bit,8bit]	10 bit [09..00]	Initial A address
[0000b, PBP_REG01]	[4bit,4bit]	8 bit	Set actual VLIW instr. memory (0 .. 15)
[DFU_OP]	[8bit]	8 bit	Execute SIMD operation with parameters in the actual VLIW instr. memory (set by the PBP_REG01 port).

3. EdkDSP IP Core – PicoBlaze6 C Application Interface Functions

The EdkDSP compiler embedded compilation of simple C and ASM programs or the PicoBlaze6 controller. PicoBlaze6 programs can use predefined and precompiled library functions listed in *Table 3*. Functions are optimized in the PicoBlaze6 assembler code, and occupy fixed area of the firmware and serve as common simple API for C and ASM PicoBlaze6 programs.

PicoBlaze6 firmware image with precompiled support functions is present in MicroBlaze header file **fill_def_program_store.h** PicoBlaze6 application program firmware is merged with this precompiled image by the MicroBlaze SW program.

Table 3: PicoBlaze6 precompiled support functions

PicoBlaze6 predefined functions	Description
unsigned char mb2pb_read_data();	Single unsigned char from MicroBlaze to PicoBlaze6
void pb2mb_write(unsigned char data);	Single unsigned char from PicoBlaze6 to MicroBlaze
void pb2mb_eoc(unsigned char data);	EOC unsigned char from PicoBlaze6 to MicroBlaze
void pb2mb_req_reset(unsigned char data);	Request from PicoBlaze6 to MicroBlaze to initiate PB reset
void pb2mb_reset();	Information from PicoBlaze6 to MicroBlaze - PB reset
void pb2dfu_set(unsigned char mem, unsigned char data);	Set one section of the VLIW instruction for the data flow unit (DFU) to an unsigned char data. VLIW instruction sections are addressed as PicoBlaze6 8bit output ports defined in *Table 2*
void pb2dfu_wait4hw();	PicoBlaze6 function is waiting for the termination of data flow unit operation.
unsigned char led2pb();	Write from PicoBlaze6 to 4 bit led output port
unsigned char btn2pb();	Read from 4 bit input port to PicoBlaze6
unsigned char hex_h(unsigned char ch);	Translate upper 4 bit nibble of an unsigned char to ascii
unsigned char hex_l(unsigned char ch);	Translate lower 4 bit nibble of an unsigned char to ascii
void pb2lcd_ascii_char(unsigned char ch, unsigned char pos);	Write from PicoBlaze6 to LCD asci alphanumerical display

4. EdkDSP IP Core – MicroBlaze C Application Interface Functions

MicroBlaze program is responsible for data communication, programming and initialization of the PicoBlaze6 and global scheduling of the implemented algorithm. The API providing MicroBlaze - Picoblaze6 interface is called Worker Abstraction Layer (WAL).

- 8xSIMD EdkDSP memory pointers and program memory pointers (from MicroBlaze view) are defined in *Table 4*.
- WAL error codes are defined in *Table 5*.
- 8xSIMD EdkDSP is supported by API functions collected in the WAL API are listed and described in *Table 6*.

Table 4: MicroBlaze access names to 8xSIMD EdkDSP memory banks

MicroBlaze access names	Description of the 8xSIMD EdkDSP memory banks
WAL_BCE_JK_DMEM_A	index of the A data memory banks (8x [0..1023] 32bit words)
WAL_BCE_JK_DMEM_B	index of the B data memory banks (8x [0..1023] 32bit words)
WAL_BCE_JK_DMEM_Z	index of the Z data memory banks (8x [0..1023] 32bit words)
WAL_CMEM_MB2PB	index to MB2PB control memory (the control register of the worker)
WAL_CMEM_PB2MB	index to PB2MB control memory (the status register of the worker)
WAL_PBID_P0	index to P0 control memory (PicoBlaze program memory 1)
WAL_PBID_P1	index to P1 control memory (PicoBlaze program memory 2)

Table 5: MicroBlaze WAL error codes

MicroBlaze WAL codes	Value	Description
WAL_RES_OK	0	all is OK
WAL_RES_WNULL	1	argument is a NULL
WAL_RES_ERR	-1	generic error
WAL_RES_ENOINIT	-2	not initiated
WAL_RES_ENULL	-3	null pointer
WAL_RES_ERUNNING	-4	worker is running
WAL_RES_ERANGE	-5	index/value is out of range

Table 6: MicroBlaze API functions for communication with 8xSIMD EdkDSP IP core

MicroBlaze API functions for communication with 8xSIMD EdkDSP IP core
wal_init_worker() - generalised function for worker initialising
***wrk** is a pointer to the worker structure. This function is designed for calling from user application. The function checks if the *wrk structure is prepared to initiate worker (the family description structure must be set). Then the assigned family function (init_wrk()) is called. In the called function all arrays of pointers to shared memories should be initiated. Return Value: The function returns return code WAL_RES_OK if successful and WAL_RES_E... if any error occurs. **int wal_init_worker(struct wal_worker *wrk);**

wal_done_worker - generalised function for worker clean-up

***wrk** is a pointer to the worker structure

This function is designed for calling from user application. The function calls done function (done_wrk()) assigned to family description structure. In the called function all dynamically allocated worker structures, memories and resources should be clean-up and released if they have been created in the worker init function.

Return Value: The function returns WAL_RES_... codes.

int wal_done_worker(struct wal_worker *wrk);

wal_reset_worker() - generalised function for worker hard reset

***wrk** is a pointer to the worker structure

This function is designed for calling from user application. The function calls reset function (reset_wrk()) assigned to the family description structure. In the called function the worker control registers should be reset (by HARD RESET bit in the worker control register). The reset is not acknowledged by accelerator.

Return Value: The function returns WAL_RES_... codes.

int wal_reset_worker(struct wal_worker *wrk);

wal_start_operation() - generalised function for starting operation on the accelerator.

***wrk** is a pointer to the worker structure. ***pbid** is an index of used PB firmware (WAL_PBID_...)

This function is designed for calling from user application. The function checks if the accelerator is in the idle state and then it calls function for starting operation (start_op()) assigned to the family description structure. The called function should start a new accelerator operation by setting accelerator control register and checking status register. This function is blocking, i.e. it waits for acknowledgement from accelerator.

Return Value: The function returns WAL_RES_... codes.

int wal_start_operation(struct wal_worker *wrk, unsigned int pbid);

wal_end_operation() - generalised function for finishing operation on the accelerator.

***wrk** is a pointer to the worker structure.

This function is designed for calling from user application. The function checks if the accelerator is in processing state and then it calls function for ending operation (end_op()) assigned to the family description structure. The called function should stop processing operation on the accelerator. And it waits for synchronization with the accelerator, therefore the function is blocking.

Return Value: The function returns WAL_RES_... codes.

int wal_end_operation(struct wal_worker *wrk);

wal_mb2pb() - generalised function for setting worker control register.

***wrk** is a pointer to the worker structure. **data** is user data to be send to worker control register.

This function is designed for calling from user application. The function calls function for setting worker control register (mb2pb()) assigned to the family description structure. The called function should send user data through control register with controlling READ bit. It should also waits for synchronization with accelerator.

Return Value: The function returns WAL_RES_... codes.

int wal_mb2pb(struct wal_worker *wrk, const uint32_t data);

wal_pb2mb() - generalised function for reading worker status register.

***wrk** is a pointer to the worker structure. ***data** is a pointer to an output buffer where read user data is written.

This function is designed for calling from user application. The function calls function for reading worker status register (pb2mb()) assigned to the family description structure. The called function should read user data through worker status register with waiting for synchronization with accelerator.

Return Value: The function returns WAL_RES_... codes.

int wal_pb2mb(struct wal_worker *wrk, uint32_t *data);

wal_mb2cmem() - generalised function for writing a block of data to any worker control or support memory

***wrk** is a pointer to the worker structure. **memid** is an index of control/support memory where data are written to (WAL_CMEM_... or WAL_..._SMEM_...). **memoffs** is offset in selected memory (in words not in bytes). **outbuf** is a pointer to memory where data are read from. **len** is a number of words to copy from **outbuf** to accelerator control memory.

This function is designed for calling from user application. The function checks index of the required memory and then it calls function for writing data to any control/support memory (mb2cmem()) assigned to the
family description structure. The called function should get a pointer to the right memory according to the required index **memid**. For accessing support memories they have to define indices greater then indices to control memories. Then the called function should copy a block of data from CPU memory **outbuf** to an accelerator control/support memory selected by **memid** and offset in selected memory **memoffs**.

Return Value: The function returns WAL_RES_... codes.

```
int wal_mb2cmem(struct wal_worker *wrk, unsigned int memid,
                unsigned int memoffs, const uint32_t *outbuf, unsigned int len);
```

wal_cmem2mb() - generalised function for reading a block of data from any worker control or support memory

***wrk** is a pointer to the worker structure. **memid** is an index of control/support memory where data are read from
(WAL_CMEM_... or WAL_..._SMEM_...). **memoffs** is offset in selected memory (in words not in bytes). ***inbuf** is a pointer to memory where data are written to. **len** is a number of words to copy from accelerator control memory.

This function is designed for calling from user application. The function checks index of the required memory and then it calls function for reading data from any control/support memory (cmem2mb()) assigned to the family description structure. The called function should get a pointer to the right memory according to the required index **memid**. For accessing support memories they have to define indices greater then indices to control memories. Then the called function should copy a block of data from the accelerator control/support memory selected by **memid** and offset in selected memory **memoffs**.

Return Value: The function returns WAL_RES_... codes.

```
int wal_cmem2mb(struct wal_worker *wrk, unsigned int memid,
                unsigned int memoffs, uint32_t *inbuf, unsigned int len);
```

wal_mb2dmem() - generalised function for writing a block of data to any worker data memory

***wrk** is a pointer to the worker structure. **simdid** is an index of SIMD which data memories are indexed. **memid** is an index of control/support memory where data are written to (WAL_CMEM_... or WAL_..._SMEM_...). **memoffs** is offset in selected memory (in words not in bytes). ***outbuf** is a pointer to memory where data are read from. **len** is a number of words to copy from ***outbuf** to accelerator control memory.

This function is designed for calling from user application. The function checks index of the required memory and then it calls function for writing data to any data memory (mb2dmem()) assigned to the family description structure. The called function should get a pointer to the right memory according to the required SIMD **simdid** and memory index **memid**. Then the called function should copy a block of data from CPU memory ***outbuf** to the accelerator data memory with offset inside the selected memory **memoffs**.

Return Value: The function returns WAL_RES_... codes.

```
int wal_mb2dmem(struct wal_worker *wrk, unsigned int simdid, unsigned int memid,
                unsigned int memoffs, const void *outbuf, unsigned int len);
```

wal_dmem2mb() - generalised function for writing a block of data to any worker data memory

***wrk** is a pointer to the worker structure. **simdid** is an index of SIMD which data memories are indexed. **memid** is an index of control/support memory where data are read from (WAL_CMEM_... or WAL_..._SMEM_...). **memoffs** is offset in selected memory (in words not in bytes). ***inbuf** is a pointer to memory where data are written to. **len** is a number of words to copy from accelerator control memory.

This function is designed for calling from user application. The function checks index of the required memory and then it calls function for reading data from any data memory (dmem2mb()) assigned to the family description structure. The called function should get pointer to the right memory according to the required SIMD **simdid** and memory index **memid**. Then the called function should copy a block of data from the accelerator data memory with offset inside the selected memory **memoffs**.

Return Value: The function returns WAL_RES_... codes.

```
int wal_dmem2mb(struct wal_worker *wrk, unsigned int simdid, unsigned int memid,
                unsigned int memoffs, void *inbuf, unsigned int len);
```

wal_set_firmware() - generalised function for writing PicoBlaze firmware

***wrk** is a pointer to the worker structure. **pbid** is an index of used PB firmware (WAL_PBID_...). ***fwbuf** is a pointer to a firmware in CPU memory. **fwsize** is a size of the firmware in words, it can be a negative value to set full firmware (4096 words).

This function is designed for calling from user application. The function checks if all arguments are correct and then it calls function for writing PB firmware (set_fw()). The called function should copy firmware from CPU memory ***fwbuf** to PicoBlaze6 program memory in the accelerator. The PB program memory is selected by the argument **pbid**. The firmware needn't be full 4096 word long. The firmware length (in words) can be set by the argument **fwsize**. If the **fwsize** is a negative value (you can use defined value WAL_FW_WHOLE) the function assumes the FW length is 4096 words.

Return Value: The function returns WAL_RES_... codes.

int wal_set_firmware(struct wal_worker *wrk, int pbid, const unsigned int *fwbuf, int fwsize);

wal_bce_jk_get_id() - implementation of the worker get_id() function for the BCE_JK families

***wrk** is a pointer to the worker structure. **pbid** is an index of used PB firmware (WAL_PBID_...). **outval** is a pointer to an output buffer for read worker ID.

The function emulates reading worker ID from hardware because the BCE_JK families don't support this operation in the hardware.

Return Value: The function always returns WAL_RES_OK.

int wal_jk_get_id(struct wal_worker *wrk, int pbid, unsigned int *outval);

wal_bce_jk_get_cap() - implementation of the worker get_cap() function for the BCE_JK families

***wrk** is a pointer to the worker structure. **pbid** is an index of used PB firmware (WAL_PBID_...). ***outval** is a pointer to an output buffer for read capabilities.

The function sends operation WAL_BCE_JK_VVER to accelerator, reads the worker capabilities and returns the read value in the ***outval** buffer.

Return Value: The function returns WAL_RES_... codes.

int wal_bce_jk_get_cap(struct wal_worker *wrk, int pbid, unsigned int *outval);

wal_bce_jk_get_lic() - implementation of the get_lic() function for the BCE_JK families

***wrk** is a pointer to the worker structure. **pbid** is an index of used PB firmware (WAL_PBID_...). ***outval** is a pointer to an output buffer for read license.

The function reads the license from the worker. For BCE_JK families the license is a 2bit license down-counter contained in the value returned by accelerator operation WAL_BCE_JK_VVER. The 2bit license counter is returned in the ***outval** buffer.

Return Value: The function returns WAL_RES_... codes.

int wal_bce_jk_get_lic(struct wal_worker *wrk, int pbid, unsigned int *outval);

All worker abstraction layer API functions listed in *Table 6* are precompiled into the MicroBlaze library **wal.a** and declared in MicroBlaze header files wal.h and **wal_bce_jk.h** .

The worker abstraction layer API functions listed in *Table 6* support instantiation of several (more than 1) instances of the 8xSIMD EdkDSP IP core.

5. EdkDSP IP Core – Integration with dual core ARM A9 Linux

The 8xSIMD EdkDSP IP core is integrated in a tester system with architecture presented in *Figure 2* and photo of the HW presented by *Figure 1* and *Figure 5*.

The dual core ARM Cortex A9 system runs configured PetaLinux 2017.1 operating system and supports:

- Ethernet 1 Gbit
- SSH, telnet, FTP, …
- The system image is located on SD card. After the initial boot, the file system is decompressed to the RAM FS in DDR3. The SD card file system is mounted and visible in the running Petalinux.
- Symmetrical multiprocessing on two ARM A9 processors
- SDSoC 2017.1 generated HW accelerators with data movers based on:
    - Simple DMA with HW supported data movers (DMA data width 32bit or 64bit) with no ARM interrupts. Simple DMA requires allocation of continuous memory space.
    - SG DMA with data movers (DMA data width 32bit or 64bit) with ARM interrupts. SG DMA can work with continuous allocation of memory or with standard Linux allocation of memory, where the continuous allocation is not guaranteed.
    - HW data movers connected to the advanced cache coherent port resolving in HW the cache coherency of dual core ARM access and data mover access to DDR3.

The MicroBlaze processor and the 8xSIMD EdkDSP IP core require initialisation and synchronisation with Linux and the dual core ARM subsystem. This is arranged by the following configuration of reserved DDR3 memory (1 GB)

Table 7: Organisation of DDR3 memory

Memory Area (in Bytes)	Size	Description
0x0000 0000 … 0x27FF FFFF	640 M Byte	Memory managed by standard Linux memory allocation mechanism. Used by dual core Arm A9 symmetrical multiprocessing 32 bit Linux
0x2800 0000 … 0x280F FFFF	1 M Byte	Reserved for MicroBlaze – ARM communication It is continuous memory reserved in Linux configuration
0x2800 0000 … 0x2810 0FFF	*4 kByte*	*Reserved for PicoBlaze6 f0 firmware (MicoBlaze and ARM)*
0x2800 1000 … 0x2810 1FFF	*4 kByte*	*Reserved for PicoBlaze6 f1 firmware (MicoBlaze and ARM)*
0x2800 2000 … 0x2810 2FFF	*4 kByte*	*Reserved for PicoBlaze6 f2 firmware (MicoBlaze and ARM)*
0x2800 3000 … 0x2810 3FFF	*4 kByte*	*Reserved for PicoBlaze6 f3 firmware (MicoBlaze and ARM)*
0x2800 4000 … 0x281F FFFF	*Reserved*	*Reserved for 8xSIMD EdkDSP data (MicoBlaze and ARM)*
0x2810 0000 … 0x29FF FFFF	15 M Byte	MicroBlaze program & data. Microblaze processor IP is configured for execution of its code from 0x28100000. It is a part of the continuous memory reserved in Linux.
0x2A00 0000 … 0x2FFF FFFF	112 M Byte	Continuous memory reserved for video frame buffers.
0x3000 0000 … 0x3FFF FFFF	256 M Byte	Memory reserved for SDSoC data mover and DMA drivers.

Linux user application uses the four reserved 4k Byte areas for copy of four PicoBlaze6 firmware programs. These programs can be compiled on the dual core ARM A9 from the C and ASM source codes stored as asci files on the mounted SD card file system. Compiled firmware programs are read by the user application running on ARM from the SD card files and copied as data to the reserved 4kB continuous memory areas. MicroBlaze program (after HW mutex based synchronisation) reads this data and uses them for programming of PicoBlaze6 FSM of the 8xSIMD EdkDSP IP.

6. Setup of Hardware

HW setup is based on commercially accessible components [1], [2], [3], [4], [5]:

TE0720-2IF; Part: XC7Z020-2CLG484I; 1 GByte DDR; Industrial Grade (-40°C to +85°C) [1].
Heatsink for TE0720, spring-loaded embedded [2].
TE0706-02 Carrier Board from Trenz Electronic [3]
Pmod USBUART Serial converter & interface [4].
TE0790-02 XMOD FTDI JTAG Adapter - Xilinx compatible [5].

See the technical reference manuals (TRM) for the description of the TE0720-02-2IF or TE0720-03-2IF module [1] and the TE0706-02 carrier board [3].

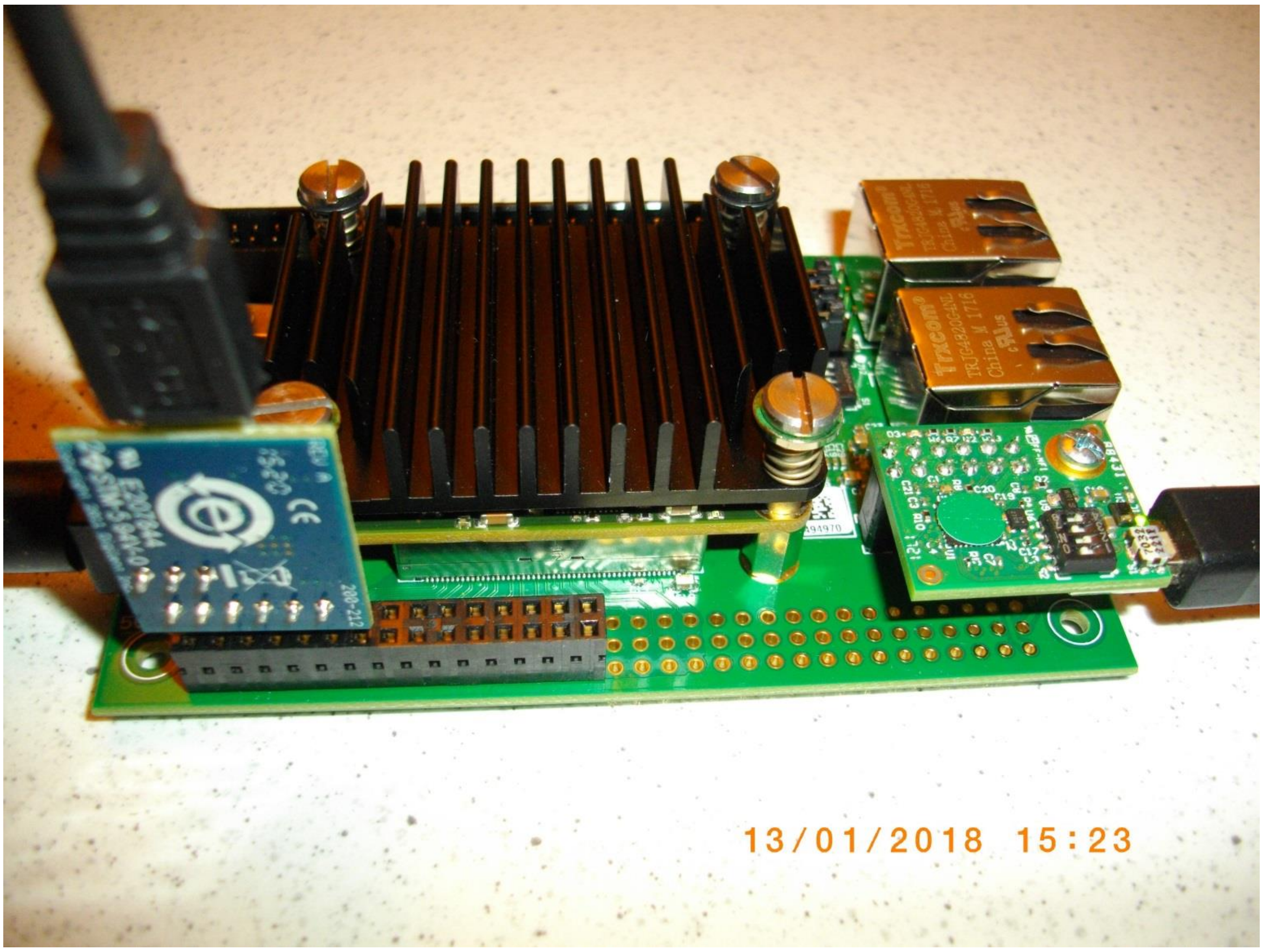


Figure 5: MicroUSB cable: Pmod USBUART. MiniUSB cable: XMOD FTDI JTAG adapter.

Set the TE0706-02 carrier board switches and jumpers for the TE0720-03-2IF module as follows:

The TE0720-2IF Zynq device works with all IO-bank supply-voltages 3.3V.

- Set jumpers to generate VCCIOA= VCCIOB= VCCIOC=3.3V
  J10: connect 2-3; J11: connect 2-3; J12: connect 2-3
- Set switch S1:
  1=ON 2=ON 3=ON 4=OFF

The TE0706-02 board ARM serial terminal/JTAG is connected to the PC by a Mini USB (type B) cable via the TE0790-02 XMOD FTDI JTAG adapter [5]. See *Figure 1* and *Figure 5*.

- Set switch in the XMOD module to:
- 1=ON 2=OFF 3=ON 4=OFF

The serial terminal for MicroBlaze is connected to the PC by a Micro USB cable via the USBUART pmod.
The J6 connector on the TE0706 carrier board has three lines of 32 pins named:

[A1 … A32]
[B1 … B32]
[C1 … C32].

The USBUART pmod is connected to pins [B1 … B6] of connector J6B (central line B). See the implemented solution on *Figure 5*.

- The jumper on the USBUART pmod is set to the default: connect lcl-vcc. With his setup, the USBUART pmod convertor chip is powered from the 5V USB from the PC and generates the local 3.3V for the pmod. See *Figure 1* and *Figure 5*.

7. Reference Application for the 8xSIMD EdkDSP IP Core

The reference application problem is the active acoustics noise cancellation for the hands free telephony.

The near end signal e(i) (voice of a speaker) is disturbed by a disturbance signal received by the near end microphone. This unknown disturbance y(i) is generated by a known (measured) far end signal (example: noise from the motor engine) u(i). The objective of the active acoustics noise cancellation is to use the measured disturbed near end microphone signal d(i) and the signal measured by the far end microphone u(i) for reconstruction of the near end speaker signal e(i) with cancelled disturbance.

The transfer function from the far end (known) source of the disturbance is modelled by a recursive FIR filter with 2000 coefficients with sampling rate 90 kHz.

Recursive FIR filter algorithm:
Objective of FIR filter is to generate sequence of modelled system outputs d(i) based on the sequence of system inputs u(i) and constant vector of N FIR filter coefficients. The generated output sequence includes also the random additive output noise defined by white noise signal e(i).

x(i) = u(i)
y(i) = [w(1), w(2), … , w(N)] * [x(i), x(i-1), … x(i-N+1)]^T
d(i) = y(i) + e(i)

Recursive adaptive LMS filter algorithm:
Objective of adaptive LMS filter is to identify recursively an unknown vector of N=2000 FIR filter coefficients from a sequence of system inputs u(i) and system outputs d(i) with sampling rate 90 kHz. The algorithm works under an assumption that the measured output sequence d(i) has been generated by a FIR filter with unknown coefficients with dimension N=2000 and includes also the unknown random white noise signal. Signal e(i) is estimated by the adaptive LMS filter.

x(i) = u(i)
y(i) = [w(1), w(2), … , w(N)] * [x(i), x(i-1), … x(i-N+1)]^T
e(i) = d[i]-y[i]
[w(1), w(2), … , w(N)] = [w(1), w(2), … , w(N)] + mu * e(i) * [x(i), x(i-1), … x(i-N+1)]

Where N is order of the FIR and LMS filter. N = 2000 in the implemented designs.

u(i) is scalar, floating point input to the system
d(i) is scalar, floating point output of a system
y(i) is scalar, floating point output of FIR filter
e(i) is scalar, floating point prediction error
[w(1), w(2), … , w(N)] is vector of N scalar , floating point FIR filter coefficients, N=2000.
mu is scalar , floating point constant used for control of the speed of convergence of the adaptive LMS filter.

The 8xSIMD EdkDSP IP Core
The 8xSIMD EdkDSP IP Core is configured for accelerated floating point computation of the recursive FIR filter with constant parameters N=2000 and for acceleration of the adaptive recursive LMS filter with N=2000 unknown coefficients with required sustained sampling frequency 90 kHz. The FIR filter models the environment and generates the sequence of u(i), d(i) data measurements. The LMS filter serves for reconstruction of the unknown e(i) sequence – the speaker voice with partially cancelled disturbance from the far distance source.

Requirements and main implementation results (for the floating point FIR & LMS filter implementation on the 8xSIMD EdkDSP IP) are listed in *Table 8.*

Table 8: Requirements and results.

Parameter	**Requirement**	**SW MicroBlaze 100 MHz**	**8xSIMD EdkDSP 120 MHz**
FIR filter sampling rate Order N=2000	90 kHz	2.5 kHz (NO)	288 kHz (YES)
FIR sustained performance (MFLOPs)	360 MFLOPs	10 MFLOPs (NO)	1152 MFLOPs (YES)
LMS filter sampling rate Order N=2000	90 kHz	1.25 kHz (NO)	92 KHz (YES)
LMS sustained performance (MFLOPs)	720 MFLOPs	10 MFLOPs (NO)	738 MFLOPs (YES)
Bit exact identical results for 8xSIMD EdkDSP IP and MB (FIR and LMS)	Required	YES	YES
Parallel EdkDSP computation and data transfers to/from DDR3 by MB	Required	YES	YES
Runtime change of 8xSIMD EdkDSP IP	Required	NA	YES
Embedded 8xSIMD EdkDSP C compiler	Required	NA	YES
Compatibility with SDSoC 2017.1	Required	YES	YES
Compatibility with PetaLinux 2017.1	Required	YES	YES
Compatibility with free SDK 2017.1 and free edition of Vivado HLS 2017.1	Required	YES	YES

Summary of main results:

- The required LMS filter sampling rate 90 KHz (with N=2000) was reached.
- The maximum is 92 kHz for the adaptive LMS filter and 288 kHz for the FIR filter.
- The sustained floating-point performance of the 8xSIMD EdkDSP is 738 MFLOPs in case of the adaptive LMS filter and 1152 MFLOPs in case of the FIR filter.
- The 8xSIMD EdkDSP operates in parallel to the Cortex A9 processor without additional computing load.
- The 8xSIMD EdkDSP operates in parallel to each of the 21 Linux examples and 19 standalone examples of HW accelerators generated from selected Cortex A9 C/C++ functions in the Xilinx SDSoC 2017.1 design environment.
- The embedded C/ASM compiler utilities for the 8xSIMD EdkDSP accelerator run as Linux applications on the dual core Arm Cortex A9 processor. These utilities can re-compile new EdkDSP firmware from the modified C/ASM source code in the runtime.

8. Installation and Use of Base Evaluation Package

This chapter describes the installation and use of a base evaluation package. Package is demonstrating:

- In-circuit Logic Analyser (ILA) JTAG based inspection/observation/debug of the 8xSIMD EdkDSP IP. ILA works with internal buffer for 8k samples and operates at 120 MHz. See *Figure 7*, *Figure 8*, *Figure 9*, *Figure 10*.
- The standalone examples support ILA and additionally can display the on-chip temperature via JTAG. See *Figure 11*
- Embedded Compilation from a C/ASM source code to firmware for the reprogrammable PicoBlaze6 finite state machine (FSM) scheduling inside of the 8xSIMD EdkDSP IP core the floating point computation sequences performed in the 8xSIMD data flow unit (DFU).
  This embedded compilation is supported for the Linux examples. See *Figure 12 Figure 13*, *Figure 14*.
- There is no need to install Xilinx SDK 2017.1 or Xilinx Vivado 2017.1 tools.
- The In-circuit Logic Analyser (ILA) JTAG based inspection/observation/debug can be performed from the free Xilinx Lab Vivado 2017.1 tool installed on Win7 (64bit) or Win 10 (64bit) PC
- The In-circuit Logic Analyser (ILA) JTAG based inspection/observation/debug can be also performed from a 32bit PC with the Xilinx Lab Vivado 2016.4 tool installed on Win7 (32bit) or Win 10 (32bit).
- The linux target examples support 1GBit Ethernet, SSH telnet and file system management tools like the Total Commander for an Ethernet based access from PC to the SD card files and editing of these files from user PC.

The base evaluation package provides 21 demos for the Linux target and the 19 precompiled demos for the standalone target. *Table 9* describes demos, PL resources and the HW/SW SDSoC 2017.1. acceleration data.

Table 9: Description of ARM SDSoC acceleration examples compatible with 8xSIMD EdkDSP IP

Linux **HW/SW Accel.**	**Stand-alone** **HW/SW Accel.**	**Description of ARM SDSoC acceleration examples. All examples are extended versions of the Xilinx GitHub SDSoC 2017.1 examples. SW extensions support the initialisation of the MicroBlaze processor and the 8xSIMD EdkDSP IP core.**
t01_l 7.86x	t01_s 10.08x	**array_partition** - This example shows how to use array partitioning to improve performance of a hardware function. Slices: 78.28% Luts: 46.27% Registers: 29.12% Block RAM: 91.79% DSPs: 79.55%
t02_l	t02_s	**burst_rw** - This is simple example of using AXI4-master interface for burst read and write. Slices: 68.96% Luts: 52.59% Registers: 24.57% Block RAM: 51.43% DSPs: 9.55%
t03_l 19.06x	t03_s 20.36x	**custom_data_type** - This is a simple example of RGB to HSV conversion to demonstrate Custom Data Type usage in hardware accelerator. Xilinx HLS compiler supports custom data type to operate within the hardware function and also it acts as a memory interface between PL to DDR3. Slices: 74.23% Luts: 49.58% Registers: 26.35% Block RAM: 51.43% DSPs: 10.91%
t04_l 0.584x	t04_s 0.59x	**data_access_random** - This is a simple example of matrix multiplication (Row x Col) to demonstrate random data access pattern. Slices: 78.04% Luts: 51.03% Registers: 28.83% Block RAM: 62.86% DSPs: 13.64%
t05_l	t05_s	**dependence_inter** - This is a simple example to demonstrate inter dependence attribute. Using inter dependence attribute user can provide additional dependency details to compiler which allow compiler to perform unrolling/pipelining to get better performance.

5.67x	6.25x	Slices: 73.64% Luts: 47.76% Registers: 26.05% Block RAM: 55.00% DSPs: 22.27%
t06_l 12.6x	t06_s 7.73x	**direct_connect** - This is a simple example of matrix multiplication with matrix addition (Out = (A x B) + C) to demonstrate direct connection which helps to achieve increasing in system parallelism and concurrency. Slices: 87.95% Luts: 57.94% Registers: 33.76% Block RAM: 95.36% DSPs: 82.27%
t07_l	t07_s	**dma_sg** - This example demonstrates how to use Scatter-Gather DMAs for data transfer to/from hardware accelerator. Slices: 84.95% Luts: 56.38% Registers: 32.39% Block RAM: 59.29% DSPs: 9.55%
t08_l	t08_s	**dma_simple** - This example demonstrates how to insert Simple DMAs for data transfer between user program and hardware accelerator. Slices: 78.20% Luts: 50.57% Registers: 28.17% Block RAM: 56.43% DSPs: 9.55%
t09_l	*Not Imple-mented as Stand-Alone*	**file_io_manr_sobel** - Linux video processing application that reads input video from a file and writes out the output video to a file. Video processing includes Motion Adaptive Noise Reduction (MANR) followed by a Sobel filter for edge detection. You can run it by supplying a 1080p YUV422 file as input with limiting number of frames to a maximum of 20 frames. Slices: 89.23% Luts: 58.91% Registers: 33.60% Block RAM: 62.50% DSPs: 10.91%
t10_l	*Not Imple-mented as Stand-Alone*	**file_io_optical** - Linux video processing application that reads input video from a file and writes out the output video to a file. Video processing performs LK Dense Optical Flow over two Full HD frames video file. You can run it by supplying a 1080p YUV422 file route85_1920x1080.yuv as input. Slices: 99.71% Luts: 81.96% Registers: 49.53% Block RAM: 85.00% DSPs: 35.45%
t11_l	t11_s	**full_array_2d** - This is a simple example of accessing full data from 2D array. Slices: 72.08% Luts: 49.26% Registers: 26.47% Block RAM: 87.50% DSPs: 12.27%
t12_l	t12_s	**hello_vadd** - This is a basic hello world kind of example which demonstrates how to achieve vector addition using hardware function. Slices: 73.42% Luts: 48.75% Registers: 26.04% Block RAM: 53.21% DSPs: 9.55%
t13_l	t13_s	**lmem_2rw** - This is a simple example of vector addition to demonstrate how to utilize both ports of Local Memory. Slices: 74.38% Luts: 49.39% Registers: 26.42% Block RAM: 55.36% DSPs: 9.55%
t14_l	t14_s	**loop_fusion** - This example will demonstrate how to fuse two loops into one to improve the performance of a C/C++ hardware function. Slices: 74.77% Luts: 49.95% Registers: 27.02% Block RAM: 53.21% DSPs: 15.00%
t15_l	t15_s	**loop_perfect** - This nearest neighbor example is to demonstrate how to achieve better performance using perfect loop. Slices: 86.75% Luts: 60.82% Registers: 32.78% Block RAM: 53.21% DSPs: 15.45%
t16_l	t16_s	**loop_pipeline** - This example demonstrates how loop pipelining can be used to improve the performance of a hardware function. Slices: 73.42% Luts: 48.75% Registers: 26.04% Block RAM: 53.21% DSPs: 9.55%
t17_l 7.31x	t17_s 9.93x	**loop_reorder** - This is a simple example of matrix multiplication (Row x Col) to demonstrate how to achieve better pipeline II factor by loop reordering. Slices: 81.13% Luts: 57.76% Registers: 30.33% Block RAM: 91.79% DSPs: 81.82%
t18_l 1.91x	t18_s 4.09x	**shift_register** - This example demonstrates how to shift values in each clock cycle. Slices: 76.20% Luts: 49.99% Registers: 27.48% Block RAM: 53.21% DSPs: 24.55%
t19_l	t19_s	**sys_port** - This is a simple example which demonstrates sys_port usage. Slices: 88.41% Luts: 58.06% Registers: 34.23% Block RAM: 61.07% DSPs: 9.55%
t20_l 0.07x	t20_s 0.17x	**systolic_array** - Matrix multiplication implemented as systolic array. Slices: 80.22% Luts: 54.76% Registers: 30.10% Block RAM: 53.21% DSPs: 61.36%
t21_l	t21_s	**wide_memory_rw** - Wide memory read write 64 bit wide. Slices: 73.82% Luts: 47.24% Registers: 26.98% Block RAM: 55.36% DSPs: 9.55%

Installation and use of the Base Evaluation Package – standalone examples

In case of standalone target:

(1) In Win 7 or Win 10 (32bit or 64bit PC), unzip the basic evaluation package
TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1.zip
to directory of your choice. We will use:
C:\TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1\

(2) Select one of the examples (t01_s … t21_s) and copy the content of sd_card directory to the SD card.
Example. Copy BOOT.bin from
C:TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1\SD_release\t01_s\sd_card\BOOT.bin
to the root of the SD card as single file.

(3) Connect USB cable from J7 connector to the PC. It will serve as ARM terminal and JTAG line.

(4) Connect another USB cable to the USBUART Pmod module present in the J5 connector to the PC. It will serve as MicroBlaze terminal.

(5) Power ON the carrier board and open putty (or similar) terminal client for both USB serial lines. Set the serial communication to: [speed 115200, data bits 8, stop bits 1, parity none and flow control None] in both cases.

(6) Insert SD card to the TE0706-02 carrier board.

(7) Reset the carrier board (S2 button).

- The standalone system will start. See
Figure 6.
- The ARM terminal will present output from the t01_s example.
- The MicroBlaze terminal will present output from the 8xSIMD EdkDSP IP. See
Figure 6.

(8) In PC, open the Vivado Lab tool. See *Figure 7*.

Open Hardware Manager
Press Auto Connect icon in Hardware window
- Open description of debug nets present in file, thus specifying the probes file

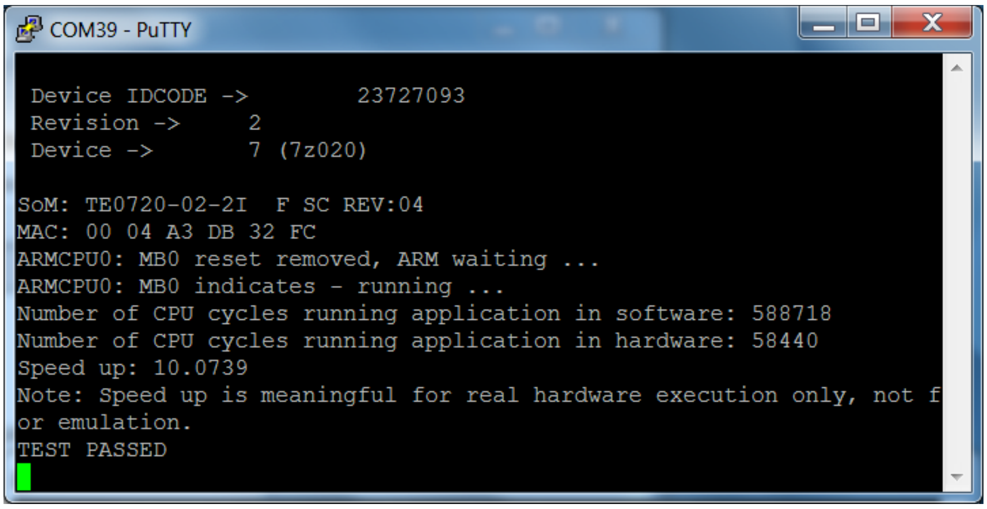
c:\TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1\SD_release\t01_s\debug_nets.ltx

- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP.
See *Figure 8*, *Figure 9*, *Figure 10*.
- Open new perspective and observe the chip temperature. See *Figure 11*.

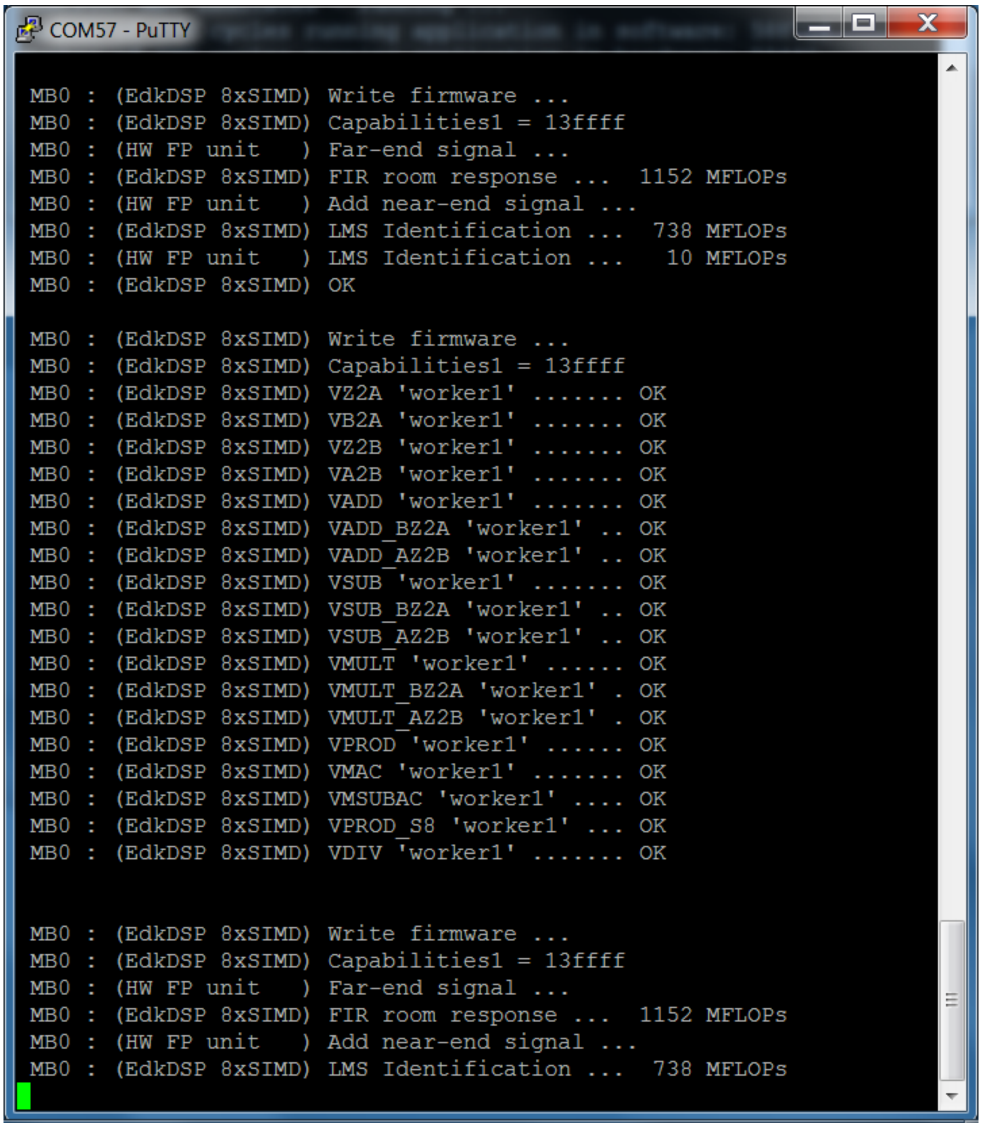
(9) Close Vivado Lab tool project.

(10)Remove SD card and reprogram it in PC to test another example.

(11)Go to step (6).

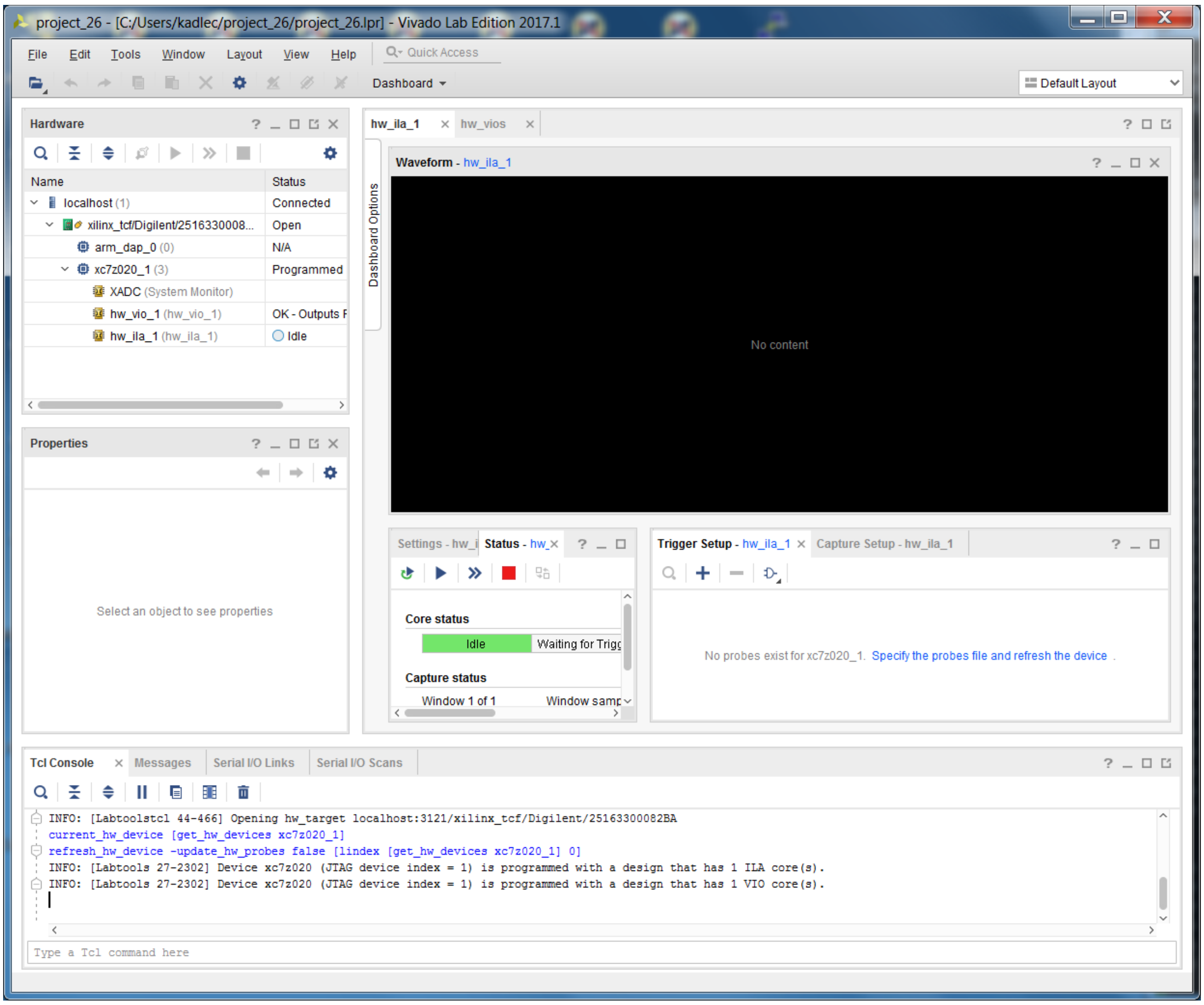


```
 Device IDCODE ->        23727093
 Revision ->      2
 Device ->        7 (7z020)

SoM: TE0720-02-2I  F SC REV:04
MAC: 00 04 A3 DB 32 FC
ARMCPU0: MB0 reset removed, ARM waiting ...
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 588718
Number of CPU cycles running application in hardware: 58440
Speed up: 10.0739
Note: Speed up is meaningful for real hardware execution only, not f
or emulation.
TEST PASSED
```



```
 MB0 : (EdkDSP 8xSIMD) Write firmware ...
 MB0 : (EdkDSP 8xSIMD) Capabilities1 = 13ffff
 MB0 : (HW FP unit    ) Far-end signal ...
 MB0 : (EdkDSP 8xSIMD) FIR room response ...  1152 MFLOPs
 MB0 : (HW FP unit    ) Add near-end signal ...
 MB0 : (EdkDSP 8xSIMD) LMS Identification ...  738 MFLOPs
 MB0 : (HW FP unit    ) LMS Identification ...   10 MFLOPs
 MB0 : (EdkDSP 8xSIMD) OK

 MB0 : (EdkDSP 8xSIMD) Write firmware ...
 MB0 : (EdkDSP 8xSIMD) Capabilities1 = 13ffff
 MB0 : (EdkDSP 8xSIMD) VZ2A 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VB2A 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VZ2B 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VA2B 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VADD 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VADD_BZ2A 'worker1' .. OK
 MB0 : (EdkDSP 8xSIMD) VADD_AZ2B 'worker1' .. OK
 MB0 : (EdkDSP 8xSIMD) VSUB 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VSUB_BZ2A 'worker1' .. OK
 MB0 : (EdkDSP 8xSIMD) VSUB_AZ2B 'worker1' .. OK
 MB0 : (EdkDSP 8xSIMD) VMULT 'worker1' ...... OK
 MB0 : (EdkDSP 8xSIMD) VMULT_BZ2A 'worker1' . OK
 MB0 : (EdkDSP 8xSIMD) VMULT_AZ2B 'worker1' . OK
 MB0 : (EdkDSP 8xSIMD) VPROD 'worker1' ...... OK
 MB0 : (EdkDSP 8xSIMD) VMAC 'worker1' ....... OK
 MB0 : (EdkDSP 8xSIMD) VMSUBAC 'worker1' .... OK
 MB0 : (EdkDSP 8xSIMD) VPROD_S8 'worker1' ... OK
 MB0 : (EdkDSP 8xSIMD) VDIV 'worker1' ....... OK


 MB0 : (EdkDSP 8xSIMD) Write firmware ...
 MB0 : (EdkDSP 8xSIMD) Capabilities1 = 13ffff
 MB0 : (HW FP unit    ) Far-end signal ...
 MB0 : (EdkDSP 8xSIMD) FIR room response ...  1152 MFLOPs
 MB0 : (HW FP unit    ) Add near-end signal ...
 MB0 : (EdkDSP 8xSIMD) LMS Identification ...  738 MFLOPs
```

Figure 6: Release demo t01_s. ARM and 8xSIMD EdkDSP terminal output.



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2516330008...	Open
arm_dap_0 (0)	N/A
xc7z020_1 (3)	Programmed
XADC (System Monitor)	
hw_vio_1 (hw_vio_1)	OK - Outputs F
hw_ila_1 (hw_ila_1)	Idle

```
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/25163300082BA
current_hw_device [get_hw_devices xc7z020_1]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7z020_1] 0]
INFO: [Labtools 27-2302] Device xc7z020 (JTAG device index = 1) is programmed with a design that has 1 ILA core(s).
INFO: [Labtools 27-2302] Device xc7z020 (JTAG device index = 1) is programmed with a design that has 1 VIO core(s).
```

Figure 7: Release demo t01_s. Vivado Lab Tool is open.

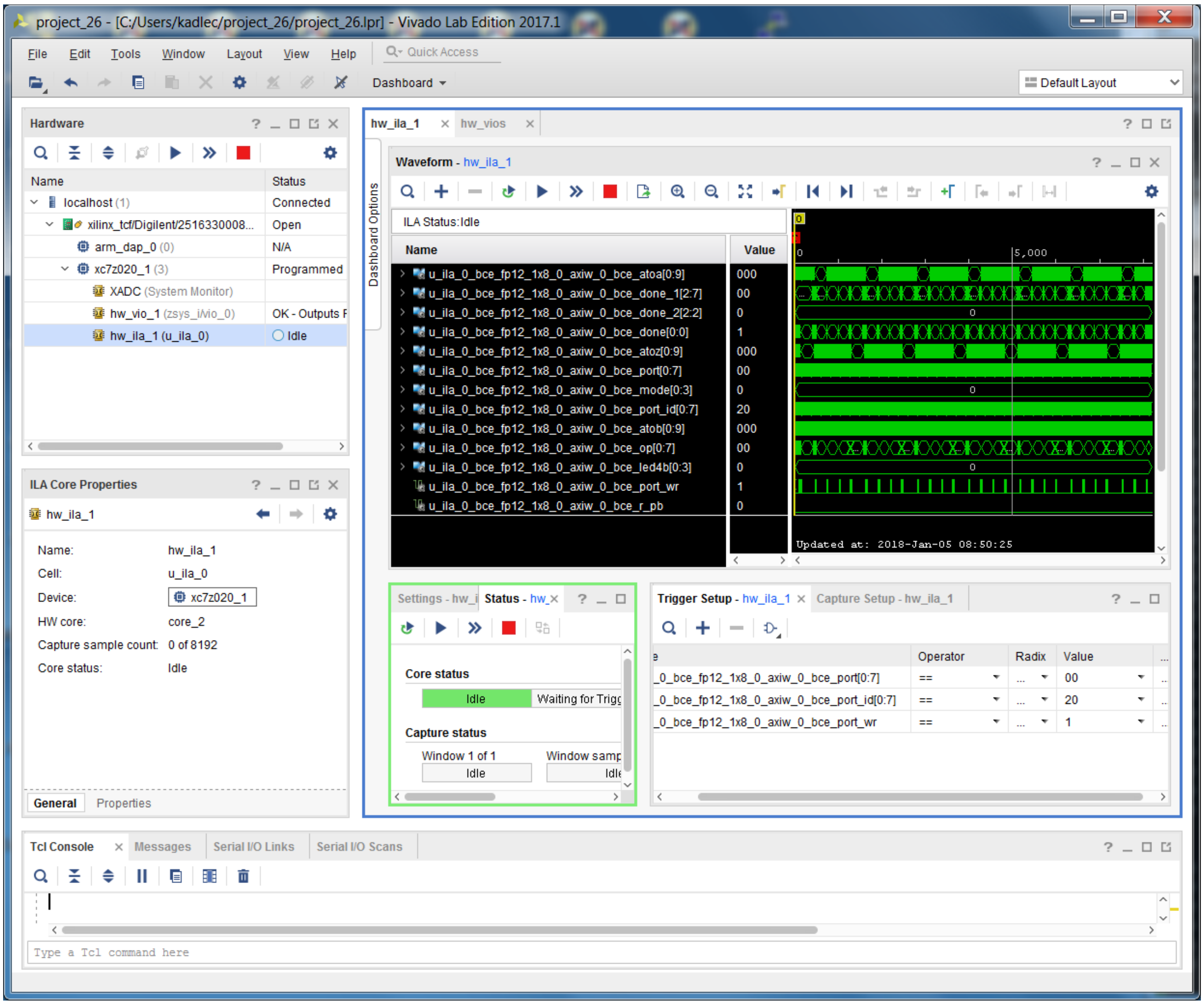
The Vivado Lab tool is connected to the chip. You have to specify the probes file (See *Figure 8*).

X:\SD_release\t01_s\debug_nets.ltx

Names and parameters of probes are added to the ILA Waveform window. See *Figure 8*.

Use **+** to select probes used for triggering, and select the condition for the trigger for each probe and their combinations (use AND as default).

Some of debug probes can be used to trigger the capturing of data. The ILA can be triggered from the EdkDSP firmware running on the PicoBlaze6 running inside of the (8xSIMD) EdkDSP unit.



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2516330008...	Open
arm_dap_0 (0)	N/A
xc7z020_1 (3)	Programmed
XADC (System Monitor)	
hw_vio_1 (zsys_iVio_0)	OK - Outputs F
hw_ila_1 (u_ila_0)	Idle

ILA Core Properties

hw_ila_1

Name:	hw_ila_1
Cell:	u_ila_0
Device:	xc7z020_1
HW core:	core_2
Capture sample count:	0 of 8192
Core status:	Idle

Waveform - hw_ila_1

ILA Status: Idle

Name	Value
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atoa[0:9]	000
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done_1[2:7]	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done_2[2:2]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done[0:0]	1
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atoz[0:9]	000
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port[0:7]	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_mode[0:3]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_id[0:7]	20
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atob[0:9]	000
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_op[0:7]	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_led4b[0:3]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_wr	1
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_r_pb	0

Updated at: 2018-Jan-05 08:50:25

Settings - hw_i | Status - hw_

Core status: Idle — Waiting for Trigg

Capture status: Window 1 of 1 — Idle; Window samp — Idl

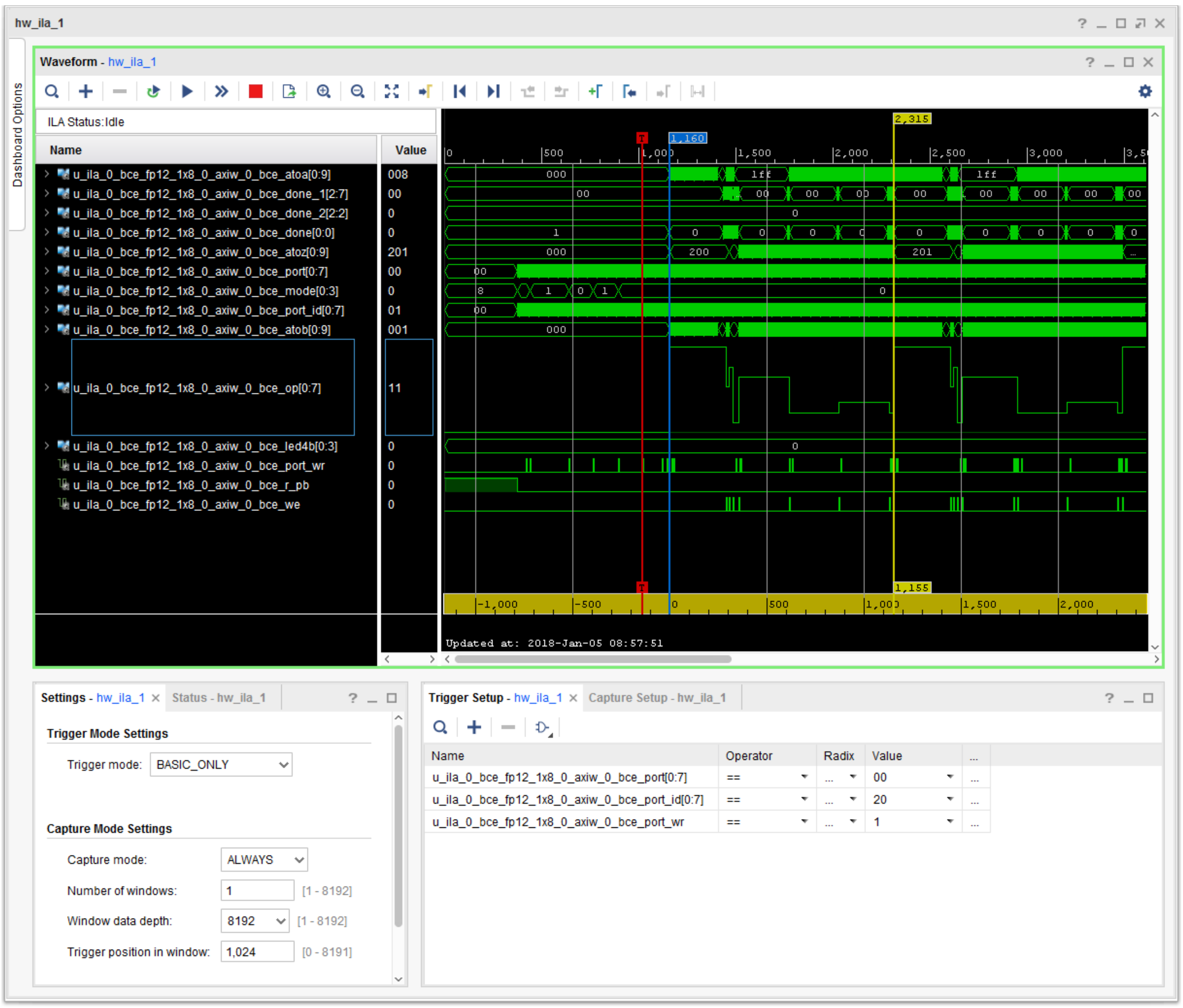
Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

e	Operator	Radix	Value
_0_bce_fp12_1x8_0_axiw_0_bce_port[0:7]	==	...	00
_0_bce_fp12_1x8_0_axiw_0_bce_port_id[0:7]	==	...	20
_0_bce_fp12_1x8_0_axiw_0_bce_port_wr	==	...	1

Figure 8: Release demo t01_s. Probes file is specified. Trigger conditions are set.

In SDK, open the **X:\zsys\edkdsp\a\f2.c** file. See section of the **LMS** C code firmware. This C code includes the additional call to the **pb2dfu_set()** function used for selective triggering of the ILA scope in specified point of computation of the EdkDSP accelerator.

```
pb2dfu_set(0x20, 0);  // To provide the trigger (0x00 on port 0x20) for the  ILA
for (i = 0; i < 4; i++) {
        for (j = 2; j <= 3; j++) {
                lms(j, n, op);
                pb2mb_eoc(led);
        }
}
…
```



ILA Status: Idle

Name	Value
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atoa[0:9]	008
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done_1[2:7]	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done_2[2:2]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done[0:0]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atoz[0:9]	201
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port[0:7]	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_mode[0:3]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_id[0:7]	01
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atob[0:9]	001
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_op[0:7]	11
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_led4b[0:3]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_wr	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_r_pb	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_we	0

Settings - hw_ila_1 Status - hw_ila_1

Trigger Mode Settings

Trigger mode: BASIC_ONLY

Capture Mode Settings

Capture mode:	ALWAYS	
Number of windows:	1	[1 - 8192]
Window data depth:	8192	[1 - 8192]
Trigger position in window:	1,024	[0 - 8191]

Trigger Setup - hw_ila_1 Capture Setup - hw_ila_1

Name	Operator	Radix	Value
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port[0:7]	==	...	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_id[0:7]	==	...	20
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_wr	==	...	1

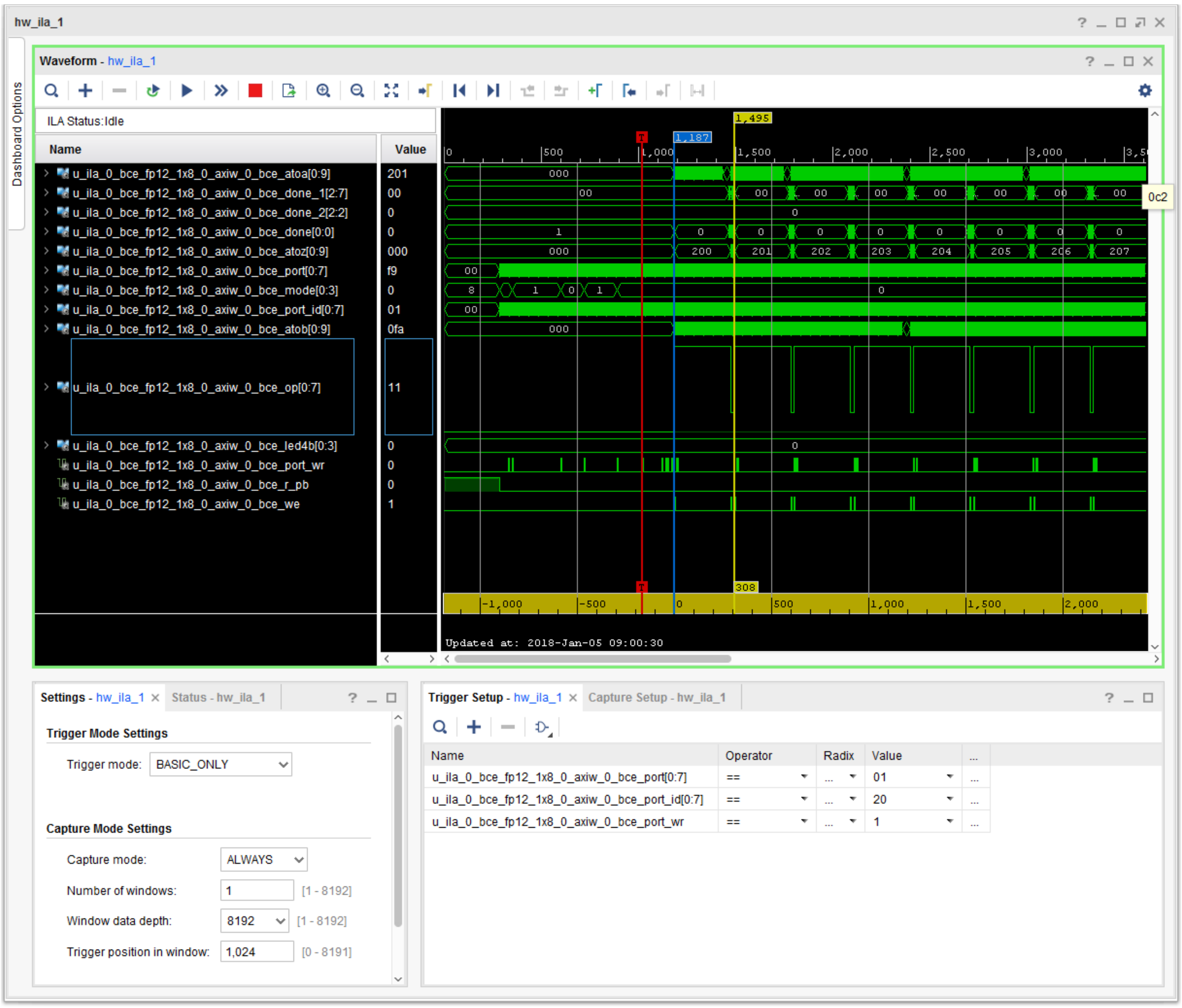
Figure 9: Release demo t01_s. Details of the 8xSIMD EdkDSP LMS filter computation.

In Vivado Lab Edition, in the ILA configuration page, change the trigger condition to:
(bce_port_wr ==1) AND (bce_port_id[0:7]==0x20) AND (bce_port[0:7]==**0x00**)

In Vivado Lab Edition 2017.1, arm the hw_ila_1 core by pressing **Run Trigger** button in **Hardware** window. Armed hw_ila_1 core will wait until the recompiled EdkDSP firmware comes to the point, where PicoBlaze6 calls function **pb2dfu_set(0x20, 0).**

ILA core starts to capture 8K samples of all debug signals with the sampling rate 120 MHz. Data are captured and sent via jtag USB connection in Vivado Lab Edition 2017.1 for visualisation and analysis in the waveform window.

This snapshot stores the detailed trace of the initial 8192 clock cycles of the FIR filter computation.
See *Figure 10*.



Name	Value
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atoa[0:9]	201
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done_1[2:7]	00
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done_2[2:2]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_done[0:0]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atoz[0:9]	000
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port[0:7]	f9
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_mode[0:3]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_id[0:7]	01
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_atob[0:9]	0fa
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_op[0:7]	11
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_led4b[0:3]	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_wr	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_r_pb	0
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_we	1

Settings - hw_ila_1 | Status - hw_ila_1

Trigger Mode Settings

Trigger mode: BASIC_ONLY

Capture Mode Settings

Capture mode: ALWAYS

Number of windows: 1 [1 - 8192]

Window data depth: 8192 [1 - 8192]

Trigger position in window: 1,024 [0 - 8191]

Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

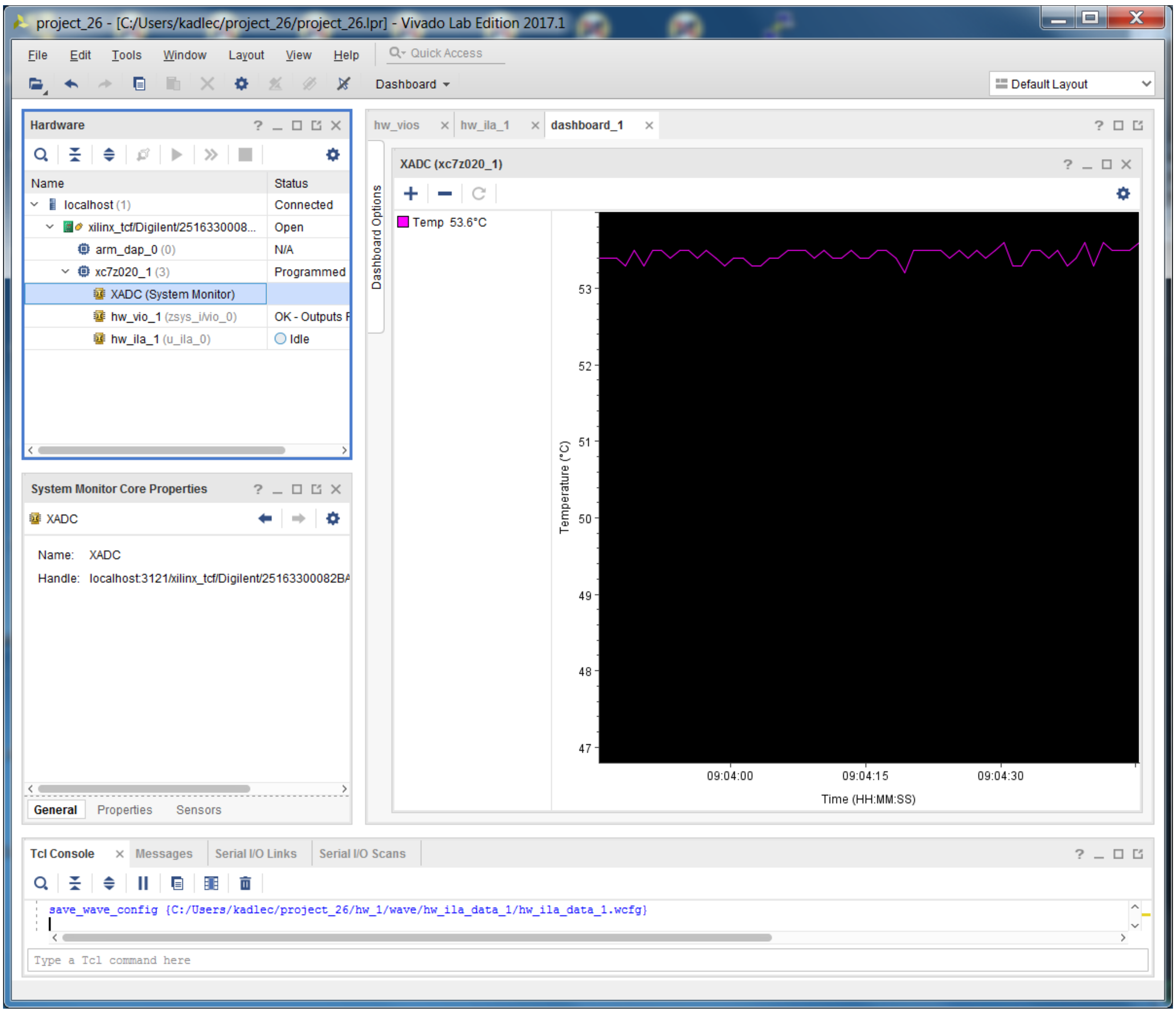
Name	Operator	Radix	Value
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port[0:7]	==	...	01
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_id[0:7]	==	...	20
u_ila_0_bce_fp12_1x8_0_axiw_0_bce_port_wr	==	...	1

Figure 10: Release demo t01_s. Details of the 8xSIMD EdkDSP FIR filter computation.

In Vivado Lab. Edition, in the ILA configuration page, change the trigger condition to bce_port[0:7]==**0x01**: to capture start of the FIR filter. See *Figure 10*. The PicoBlaze C code of the FIR example is listed in *Figure 17*.

The Vivado Lab. screens presented in *Figure 9* and *Figure 10* display also the 1024 samples before the trigger event. This mode is set in the trigger mode settings window. Screens display how the PicoBlaze6 controller reset signal **bce_r_pb** is deactivated. Picoblaze6 reads the 8 bit parameters **op** and **n** from the MicroBlaze before the trigger evet. See complete program listing in *Figure 17* with these initial lines of the PicoBlaze6 SW:

```
…
void main() {
        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 {
                n = mb2pb_read_data();
                pb2dfu_set(0x20, 0);  // To provide the trigger (0x00 on port 0x20) for the  ILA
                …
```



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2516330008...	Open
arm_dap_0 (0)	N/A
xc7z020_1 (3)	Programmed
XADC (System Monitor)	
hw_vio_1 (zsys_i/vio_0)	OK - Outputs F
hw_ila_1 (u_ila_0)	Idle

Name: XADC
Handle: localhost:3121/xilinx_tcf/Digilent/25163300082BA

```
save_wave_config {C:/Users/kadlec/project_26/hw_1/wave/hw_ila_data_1/hw_ila_data_1.wcfg}
```

Figure 11: Release demo t01_s. Stanalone demo supports measurements of the chip temperature.

The standalone demos support measurement of the chip temperature in a new dashboard connected to the XADC system monitor.

Installation and use of Base Evaluation Package – linux examples

In case of Linux target:

(1) In Win 7 or Win 10 (32bit or 64bit PC), unzip the basic evaluation package
TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1.zip
to directory of your choice. We will use:
C:\TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1\

(2) Select one of the examples (t01_l … t21_l) and copy the content of sd_card directory to the SD card.
Example. Copy content of the directory from
C:\TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1\SD_release\t01_l\sd_card\
to the root of the SD card

(3) Connect Mini USB cable from J7 connector to the PC. It will serve as ARM terminal and JTAG line.

(4) Connect Micro USB cable from to the USBUART Pmod module present in the J5 connector) to the PC. It will serve as MicroBlaze terminal.

(5) Power ON the carrier board. And open putty (or similar) terminal client for both USB serial lines.
Set the serial communication to [speed 115200, data bits 8, stop bits 1, parity none and flow control None] in both cases.

(6) Insert SD card to the TE0706-02 carrier board.

(7) Reset the carrier board.

- The linux system will start. See *Figure 12*.
type user name:

```
root
```

type password:

```
root
```

- Mount SD card to the directory (See *Figure 13*) /mnt by typing:

```
mount   /dev/mmcblk0p1   /mnt
```

- Change directory (See *Figure 13*) to /mnt

```
cd /mnt
```

-Compile firmware for the PicoBlaze6 by the EdkDSP C compiler (See *Figure 13*):

```
./edkdsp/tools/cc_fx.sh ./edkdsp/a
```

(or ./edkdsp/tools/cc_fx.sh ./edkdsp/b   or   ./edkdsp/tools/cc_fx.sh ./edkdsp/c …)

- The PicoBlaze6 C source code f0.c f1.c f2.c and f3.c from the directory ./edkdsp/a
are compiled by the EdkDSP C compiler to the firmware files (See *Figure 13*):

```
  ./f0.dec   ./f1.dec  ./f2.dec  ./f3.dec
```

- The ARM terminal will present output from the EdkDSP C compiler
- The MicroBlaze terminal is not active. EdkDSP is not programmed yet.

- Start the linux application (See *Figure 14*) by typing:

```
./t01_l.elf
```

- The ARM terminal will present output from the t01_l.elf example. See *Figure 14*.

- The MicroBlaze terminal will present output from the 8xSIMD EdkDSP IP
  working with new firmware programs as re-compiled by the EdkDSP C compiler
  from the C source code files: f0.c f1.c f2.c and f3.c
  from the directory ./edkdsp/a
  The output from the 8xSIMD EdkDSP is identical to the standalone output. See *Figure 6*.

(8) In PC, open the Vivado Lab tool. See *Figure 7*.
- Open Hardware Manager
- Press Auto Connect icon in Hardware window
- Open description of debug nets present in file, thus specifying the probes file. See *Figure 8*.

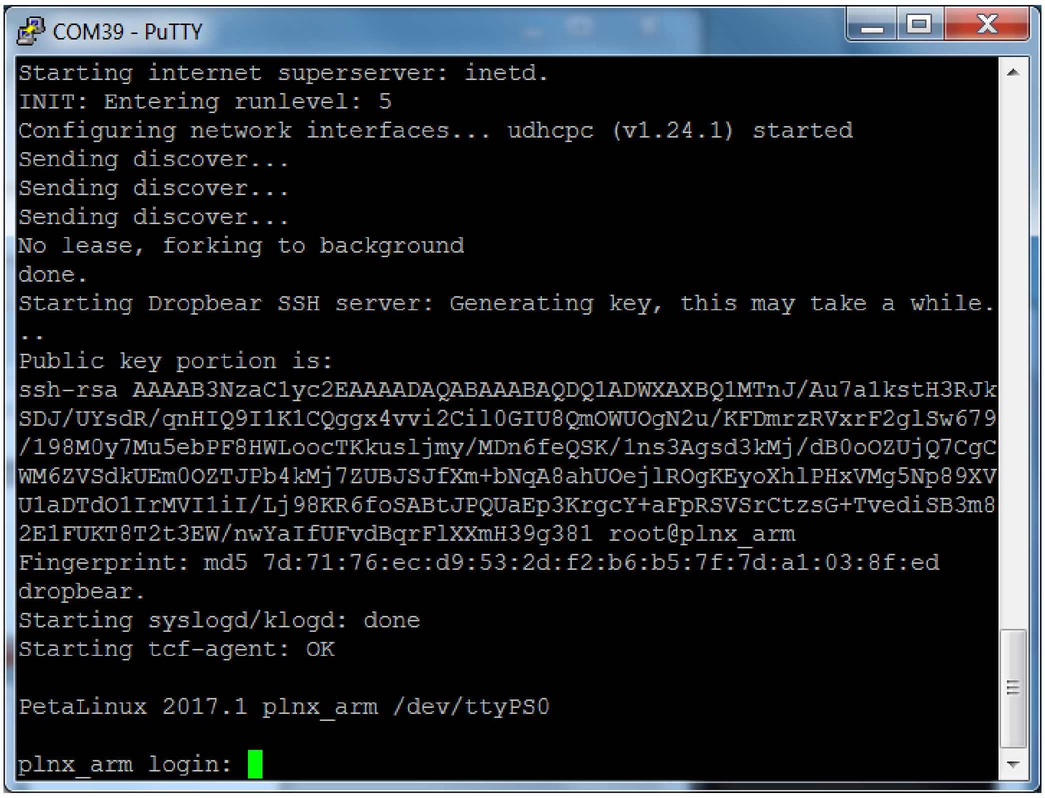
```
c:\TE0720_MB_EdkDSP_1x8_zsys_2if_ila_8k_usb_sw1\SD_release\t01_s\debug_nets.ltx
```

- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP. See *Figure 9*, *Figure 10*.

(9) Close Vivado Lab tool project.

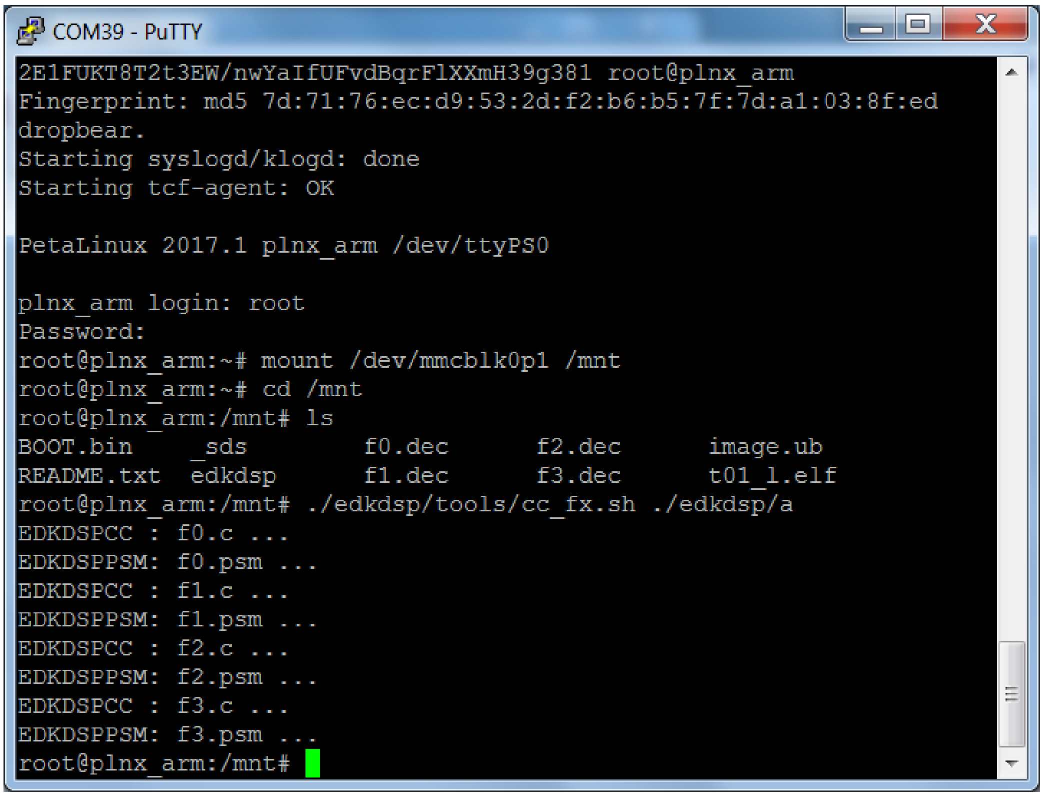
(10)Remove SD card and reprogram it in PC to test another example.

(11)Go to step (6).



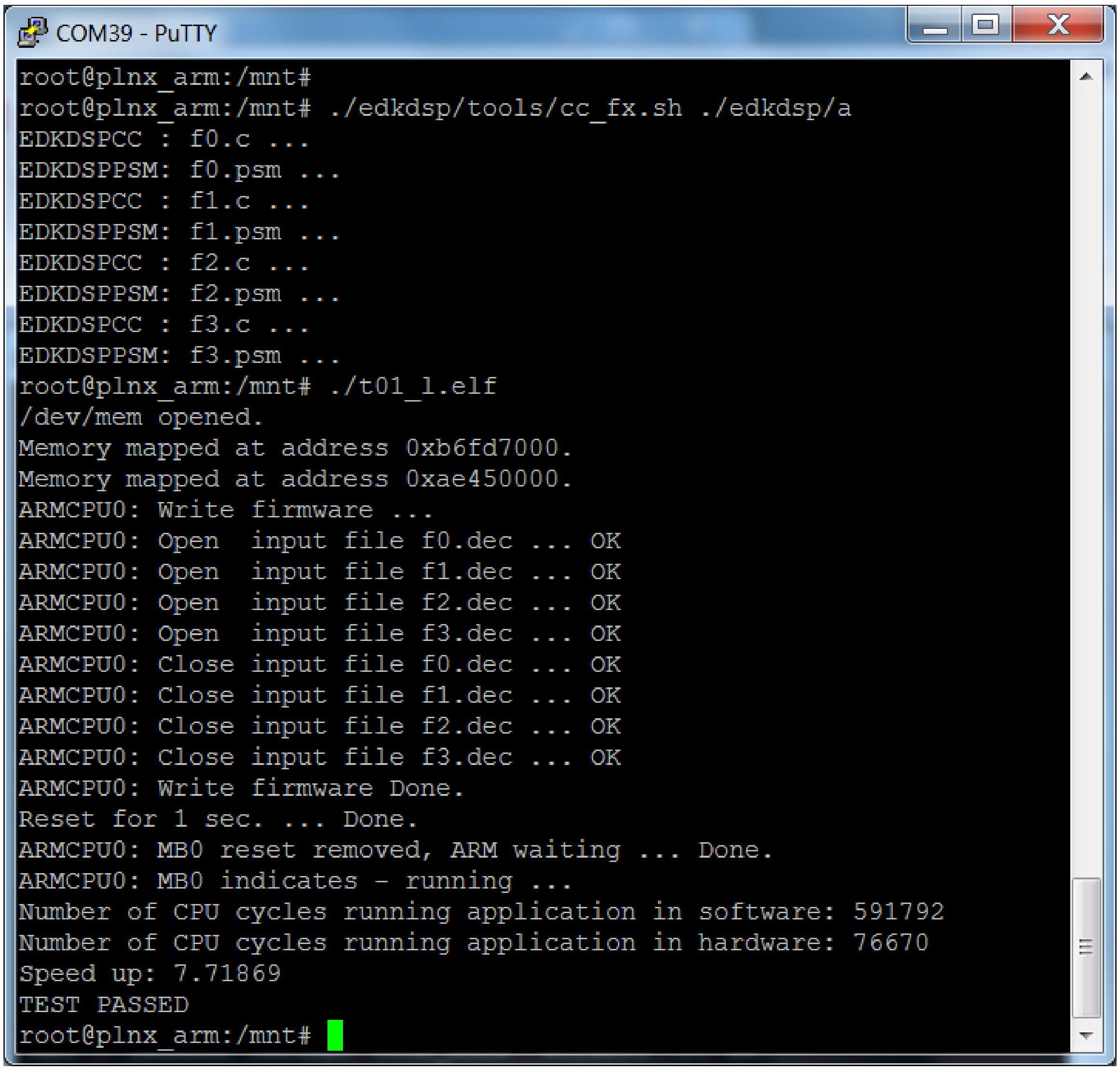
```
Starting internet superserver: inetd.
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.24.1) started
Sending discover...
Sending discover...
Sending discover...
No lease, forking to background
done.
Starting Dropbear SSH server: Generating key, this may take a while.
..
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDQ1ADWXAXBQ1MTnJ/Au7a1kstH3RJk
SDJ/UYsdR/qnHIQ9I1K1CQggx4vvi2Cil0GIU8QmOWUOgN2u/KFDmrzRVxrF2glSw679
/198M0y7Mu5ebPF8HWLoocTKkusljmy/MDn6feQSK/1ns3Agsd3kMj/dB0oOZUjQ7CgC
WM6ZVSdkUEm0OZTJPb4kMj7ZUBJSJfXm+bNqA8ahUOejlROgKEyoXhlPHxVMg5Np89XV
U1aDTdO1IrMVI1iI/Lj98KR6foSABtJPQUaEp3KrgcY+aFpRSVSrCtzsG+TvediSB3m8
2E1FUKT8T2t3EW/nwYaIfUFvdBqrFlXXmH39g381 root@plnx_arm
Fingerprint: md5 7d:71:76:ec:d9:53:2d:f2:b6:b5:7f:7d:a1:03:8f:ed
dropbear.
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2017.1 plnx_arm /dev/ttyPS0

plnx_arm login:
```

Figure 12: Release demo t01_l. Linux start.



```
2E1FUKT8T2t3EW/nwYaIfUFvdBqrFlXXmH39g381 root@plnx_arm
Fingerprint: md5 7d:71:76:ec:d9:53:2d:f2:b6:b5:7f:7d:a1:03:8f:ed
dropbear.
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2017.1 plnx_arm /dev/ttyPS0

plnx_arm login: root
Password:
root@plnx_arm:~# mount /dev/mmcblk0p1 /mnt
root@plnx_arm:~# cd /mnt
root@plnx_arm:/mnt# ls
BOOT.bin     _sds        f0.dec      f2.dec      image.ub
README.txt   edkdsp      f1.dec      f3.dec      t01_l.elf
root@plnx_arm:/mnt# ./edkdsp/tools/cc_fx.sh ./edkdsp/a
EDKDSPCC : f0.c ...
EDKDSPPSM: f0.psm ...
EDKDSPCC : f1.c ...
EDKDSPPSM: f1.psm ...
EDKDSPCC : f2.c ...
EDKDSPPSM: f2.psm ...
EDKDSPCC : f3.c ...
EDKDSPPSM: f3.psm ...
root@plnx_arm:/mnt#
```

Figure 13: Release demo t01_l. Login, Compilation of firmware in the EdkDSP C Compiler.



COM39 - PuTTY

```
root@plnx_arm:/mnt#
root@plnx_arm:/mnt# ./edkdsp/tools/cc_fx.sh ./edkdsp/a
EDKDSPCC : f0.c ...
EDKDSPPSM: f0.psm ...
EDKDSPCC : f1.c ...
EDKDSPPSM: f1.psm ...
EDKDSPCC : f2.c ...
EDKDSPPSM: f2.psm ...
EDKDSPCC : f3.c ...
EDKDSPPSM: f3.psm ...
root@plnx_arm:/mnt# ./t01_l.elf
/dev/mem opened.
Memory mapped at address 0xb6fd7000.
Memory mapped at address 0xae450000.
ARMCPU0: Write firmware ...
ARMCPU0: Open  input file f0.dec ... OK
ARMCPU0: Open  input file f1.dec ... OK
ARMCPU0: Open  input file f2.dec ... OK
ARMCPU0: Open  input file f3.dec ... OK
ARMCPU0: Close input file f0.dec ... OK
ARMCPU0: Close input file f1.dec ... OK
ARMCPU0: Close input file f2.dec ... OK
ARMCPU0: Close input file f3.dec ... OK
ARMCPU0: Write firmware Done.
Reset for 1 sec. ... Done.
ARMCPU0: MB0 reset removed, ARM waiting ... Done.
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 591792
Number of CPU cycles running application in hardware: 76670
Speed up: 7.71869
TEST PASSED
root@plnx_arm:/mnt#
```

Figure 14: Release demo t01_l. Program and start 8xSIMD EdkDSP demo.

9. Installation and Use of Extended Evaluation Package

The extended evaluation package is offered to the ECSEL PRODUCTIVE 4.0 project partners [8] on their written request to UTIA for free. See the license conditions listed in next sections of this report.

The extended evaluation package supports:

- Compilation from C source code and debug for the MicroBlaze processor for Linux and standalone targets
- Creation and Release of SD cards with new compiled MicroBlaze SW and new compiled Picoblaze6 firmware for Linux and standalone targets.
- In-circuit Logic Analyser (ILA) JTAG based inspection/observation/debug of the 8xSIMD EdkDSP IP. ILA works with internal buffer for 8k samples and operates at 120 MHz.
- Embedded Compilation from a C/ASM source code to firmware for the reprogrammable PicoBlaze6 finite state machine (FSM) scheduling inside of the 8xSIMD EdkDSP IP core the floating point computation sequences performed in the 8xSIMD data flow unit (DFU).
  This embedded compilation is supported for the Linux examples.
- The standalone examples also support ILA and additionally can display the on-chip temperature via JTAG.
- The extended evaluation package requires the Xilinx SDK 2017.1 tools (download is free). SDK serves for compilation of MicroBlaze code, download of compiled MicroBlaze code via JTAG and for the debug of this code in parallel with the ILA inspection/observation/debug of the EdkDSP IP core.
- The In-circuit Logic Analyser (ILA) JTAG based inspection/observation/debug can be performed from the free Xilinx Lab Vivado 2017.1 tool installed on Win7 (64bit) or Win 10 (64bit) PC.
- The linux target examples support 1G Bit Ethernet, SSH telnet and file system management tools like the Total Commander for an Ethernet based access from PC to the SD card files and editing of these files from user PC.

The extended evaluation package provides 21 precompiled designs for the linux target and 19 precompiled designs for the standalone target as described in *Table 9* .

Installation and use of extended evaluation package – standalone examples

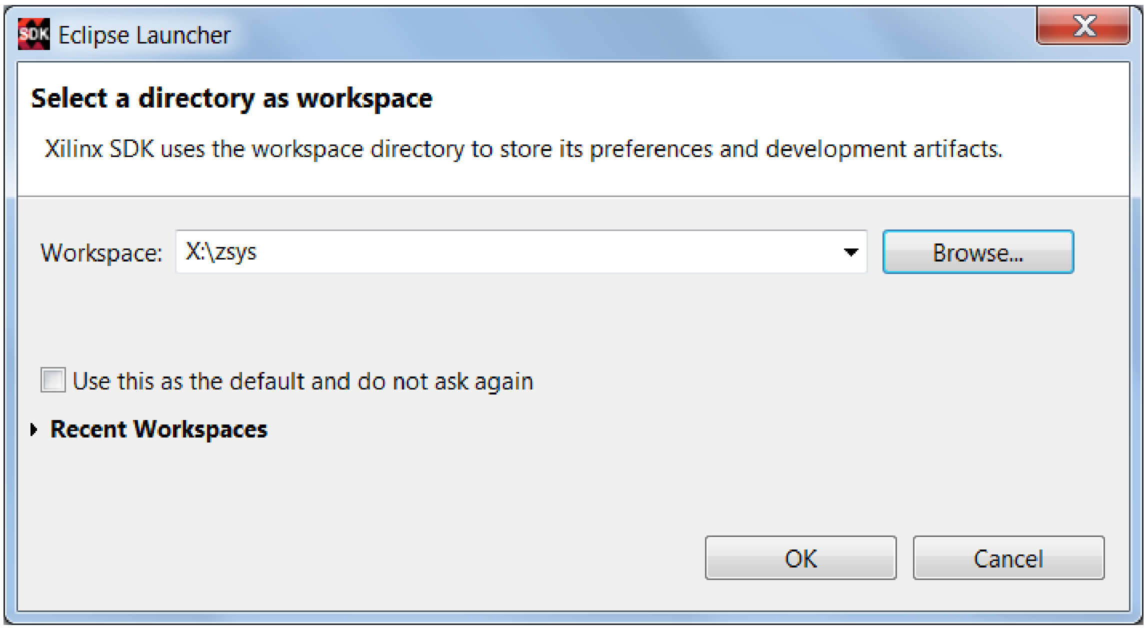
In case of standalone target:

(1) In Win 7 or Win 10 (64bit PC), unzip the basic evaluation package
TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL.zip
to directory of your choice. We will use:

C:\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL

In Xilinx SDK 2017.1 create a new workspace in the directory

X:\zsys


Eclipse Launcher

Select a directory as workspace

Xilinx SDK uses the workspace directory to store its preferences and development artifacts.

Workspace: X:\zsys

Browse...

☐ Use this as the default and do not ask again

Recent Workspaces

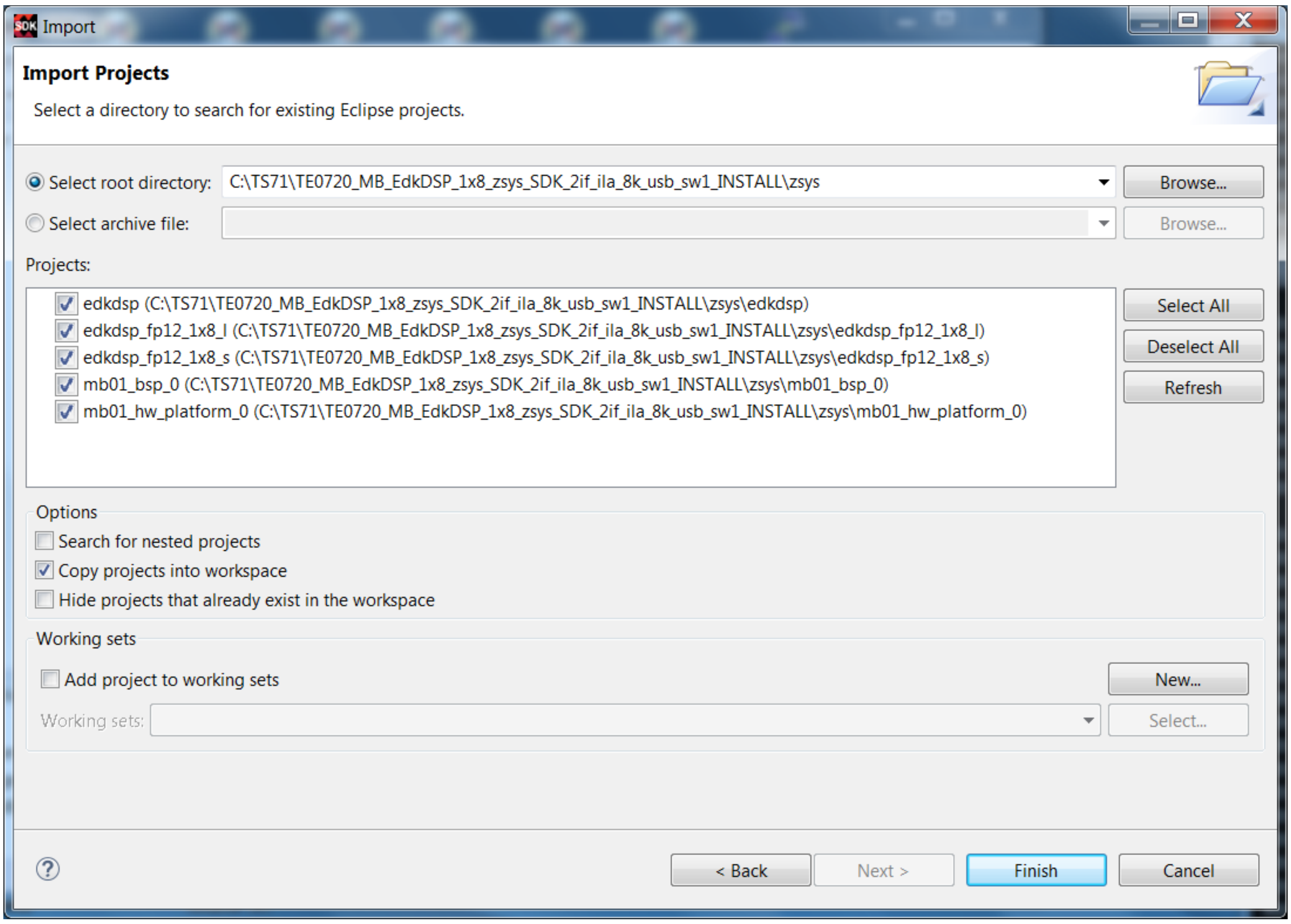
OK Cancel

Figure 15: Create new SDK 2017.1 workspace.

Import (with copy) all SDK projects from

C:\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys

to the SDK workspace X:\zsys


Import

Import Projects

Select a directory to search for existing Eclipse projects.

(● Select root directory:) C:\TS71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys

(Select archive file:)

Projects:

- [✓] edkdsp (C:\TS71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys\edkdsp)
- [✓] edkdsp_fp12_1x8_l (C:\TS71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys\edkdsp_fp12_1x8_l)
- [✓] edkdsp_fp12_1x8_s (C:\TS71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys\edkdsp_fp12_1x8_s)
- [✓] mb01_bsp_0 (C:\TS71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys\mb01_bsp_0)
- [✓] mb01_hw_platform_0 (C:\TS71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys\mb01_hw_platform_0)

Options

- [ ] Search for nested projects
- [✓] Copy projects into workspace
- [ ] Hide projects that already exist in the workspace

Working sets

- [ ] Add project to working sets

Working sets:

Figure 16: Import the extended evaluation package projects into the SDK Workspace.

(2) Select one of the examples (t01_s … t21_s) and copy the content of the sd_card directory to the SD card. Example. Copy BOOT.bin from

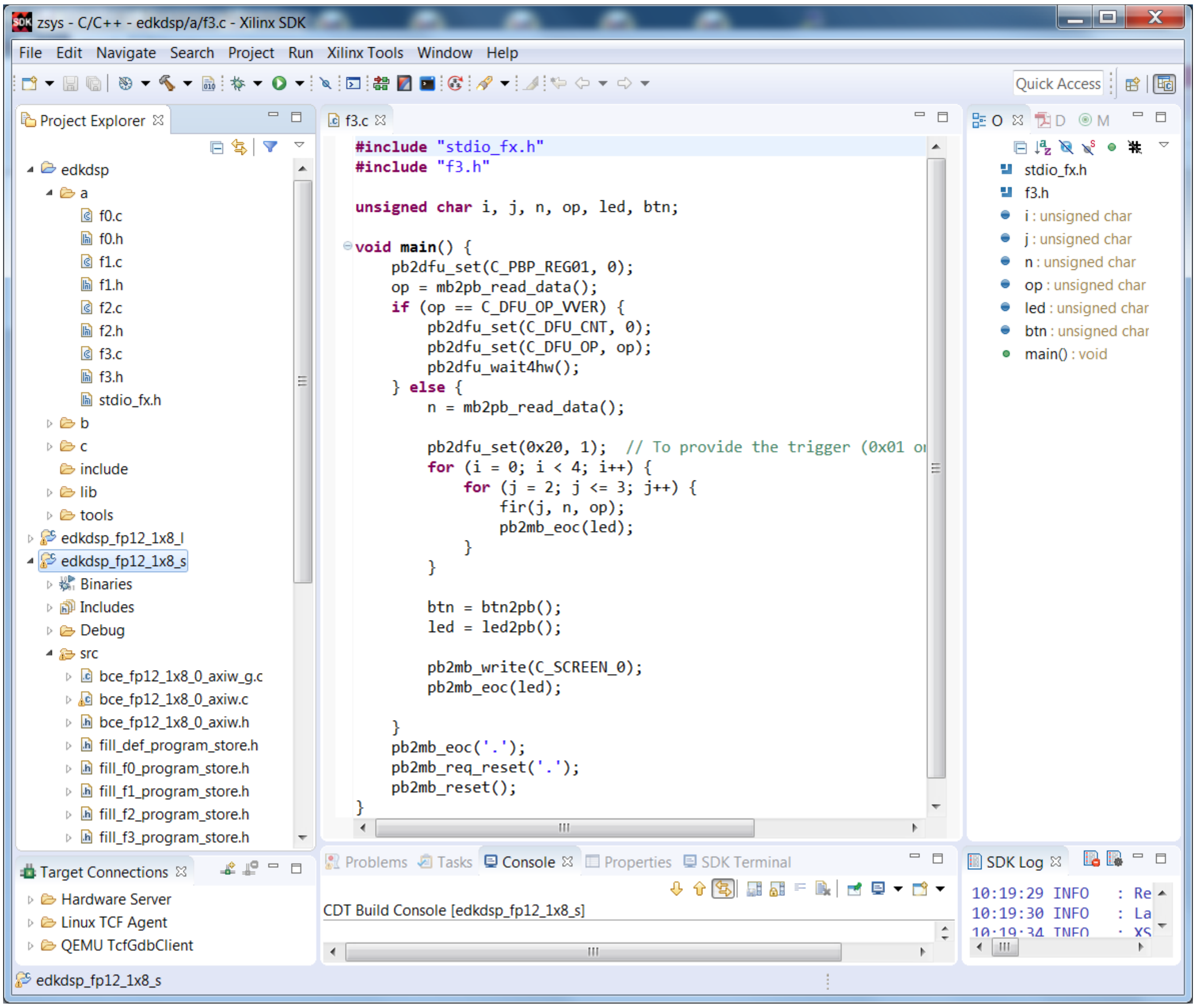
X:\SD_debug\t01_s\sd_card\BOOT.bin

to the root of the SD card as a single file.

(3) Connect Mini USB cable from J7 connector to the PC. It will serve as ARM terminal and JTAG line.

(4) Connect Micro USB cable to the USBUART Pmod module present in the J5 connector to the PC. It will serve as MicroBlaze terminal.

(5) Power ON the carrier board. And open putty (or similar) terminal client for both USB serial lines. Set the serial communication to [speed 115200, data bits 8, stop bits 1, parity none and flow control None] in both cases.



```
#include "stdio_fx.h"
#include "f3.h"

unsigned char i, j, 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 {
        n = mb2pb_read_data();

        pb2dfu_set(0x20, 1);  // To provide the trigger (0x01 o
        for (i = 0; i < 4; i++) {
            for (j = 2; j <= 3; j++) {
                fir(j, n, op);
                pb2mb_eoc(led);
            }
        }

        btn = btn2pb();
        led = led2pb();

        pb2mb_write(C_SCREEN_0);
        pb2mb_eoc(led);

    }
    pb2mb_eoc('.');
    pb2mb_req_reset('.');
    pb2mb_reset();
}
```

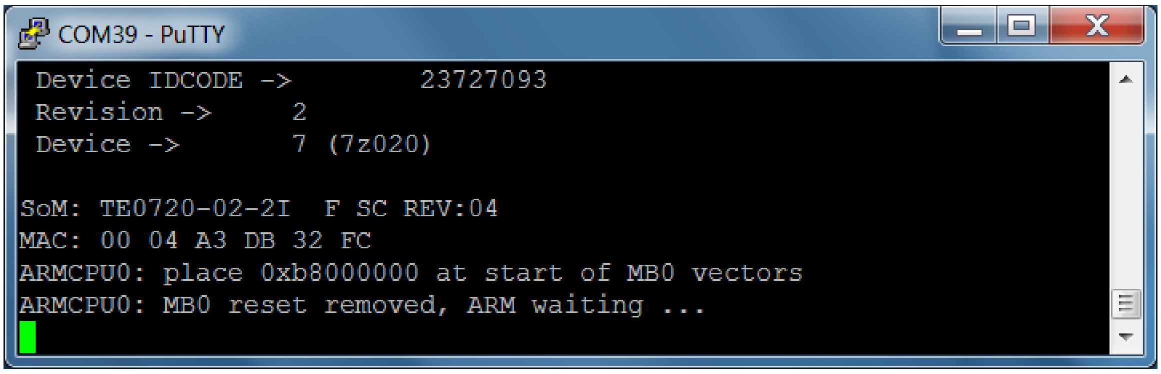
```
10:19:29 INFO  : Re
10:19:30 INFO  : La
10:19:34 INFO  : XS
```

Figure 17: SDK compiles MicroBlaze SW projects for the standalone debug target.

(6) Insert SD card to the TE0706-02 carrier board.

(7) Reset the carrier board.

- The standalone system will start.
- The ARM terminal will present output from the t01_s example. The Arm application is waiting for the MicroBlaze.



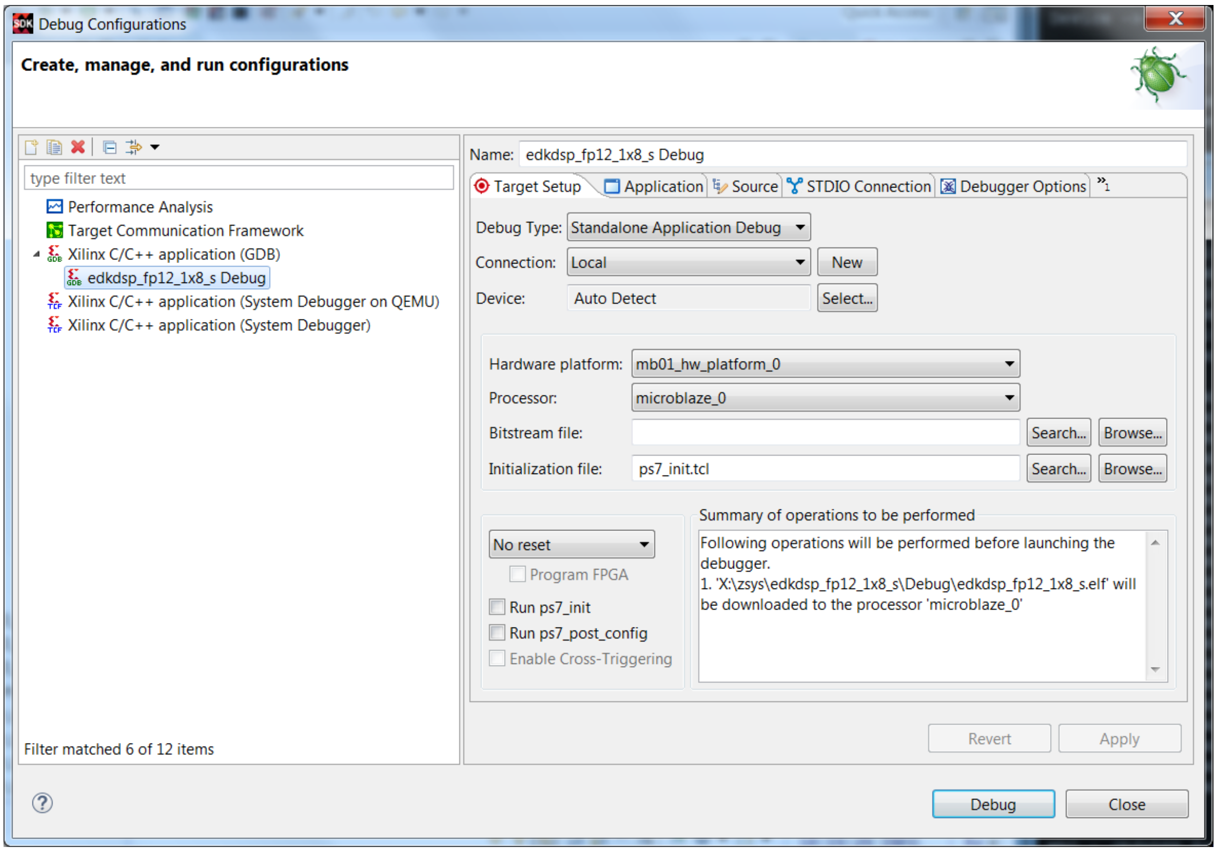
```
 Device IDCODE ->        23727093
 Revision ->       2
 Device ->         7 (7z020)

SoM: TE0720-02-2I  F SC REV:04
MAC: 00 04 A3 DB 32 FC
ARMCPU0: place 0xb8000000 at start of MB0 vectors
ARMCPU0: MB0 reset removed, ARM waiting ...
```

Figure 18: Debug demo t01_l. Execution of the ./t01_s.elf example from the SD card.

- The Xilinx SDK project edkdsp_fp12_1x8_s already includes PicoBlaze6 firmware header files
  fill_f0_program_store.h, fill_f1_program_store.h, fill_f2_program_store.h, fill_f3_program_store.h
  These files can be recreated from C source code by the EdkDSP C compiler in the
  linux target session as described in the next section.

- In the Xilinx SDK workspace, compile the edkdsp_fp12_1x8_s project with the existing (or new, recompiled) PicoBlaze6 firmware headers fill_f0_program_store.h, fill_f1_program_store.h, fill_f2_program_store.h, fill_f3_program_store.h.

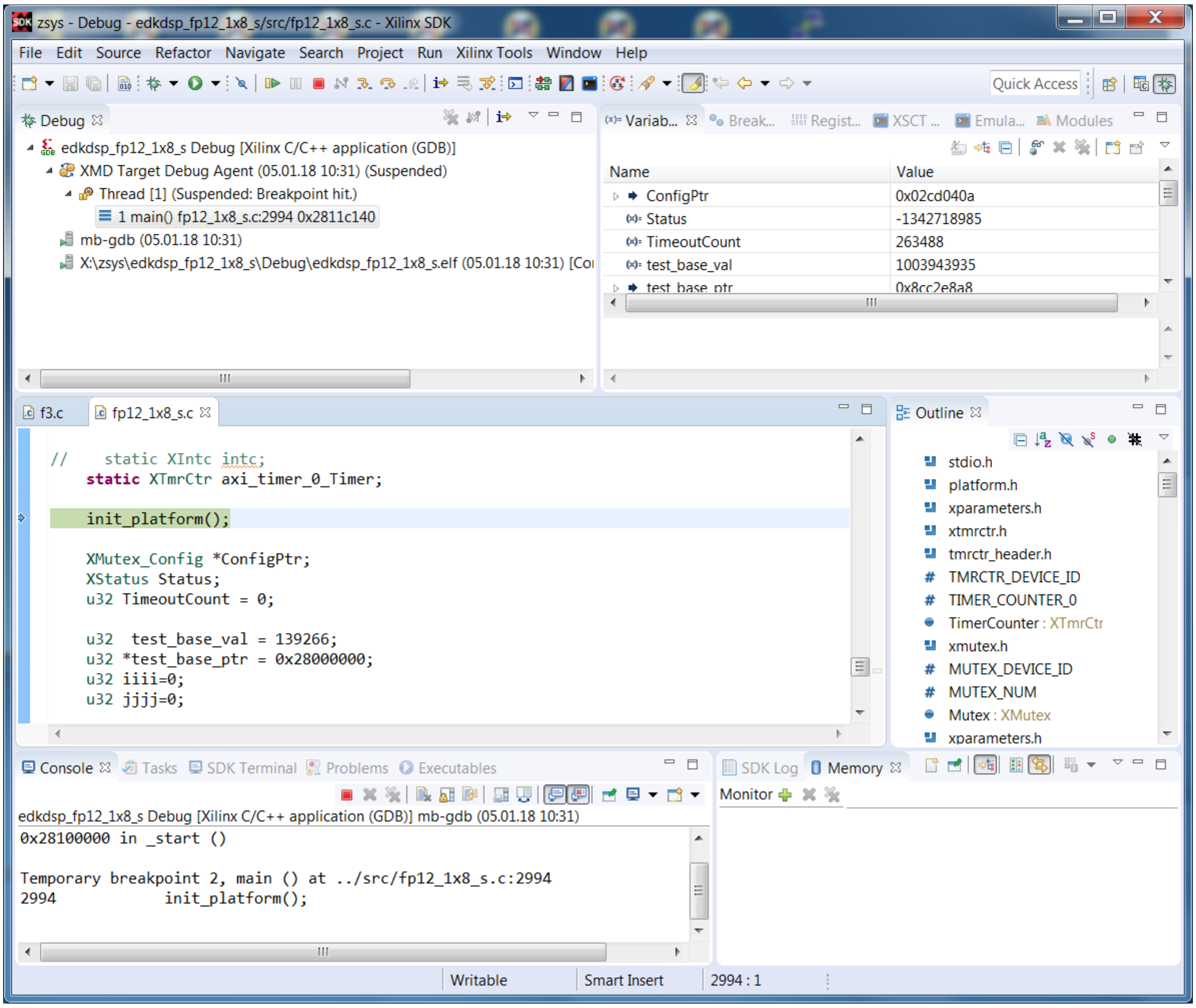
- In the Xilinx SDK workspace, select Debug of MicroBaze project **edkdsp_fp12_1x8_s**. In the Debug Configurations, select "No reset", unselect "Run ps7_init", unselect "Run ps7_post_config" click "Apply".



Name: edkdsp_fp12_1x8_s Debug
Debug Type: Standalone Application Debug
Connection: Local
Device: Auto Detect
Hardware platform: mb01_hw_platform_0
Processor: microblaze_0
Bitstream file:
Initialization file: ps7_init.tcl
No reset
[ ] Run ps7_init
[ ] Run ps7_post_config

Summary of operations to be performed
Following operations will be performed before launching the debugger.
1. 'X:\zsys\edkdsp_fp12_1x8_s\Debug\edkdsp_fp12_1x8_s.elf' will be downloaded to the processor 'microblaze_0'

Figure 19: Debug demo t01_s. Open project edkdsp_fp12_1x8_s for debug.


zsys - Debug - edkdsp_fp12_1x8_s/src/fp12_1x8_s.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Tools Window Help
Quick Access
Debug
Variab... Break... Regist... XSCT ... Emula... Modules
f3.c fp12_1x8_s.c
Outline
Console Tasks SDK Terminal Problems Executables
SDK Log Memory
Monitor
Writable Smart Insert 2994 : 1

```
edkdsp_fp12_1x8_s Debug [Xilinx C/C++ application (GDB)]
  XMD Target Debug Agent (05.01.18 10:31) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() fp12_1x8_s.c:2994 0x2811c140
  mb-gdb (05.01.18 10:31)
  X:\zsys\edkdsp_fp12_1x8_s\Debug\edkdsp_fp12_1x8_s.elf (05.01.18 10:31) [Co
```

Name	Value
ConfigPtr	0x02cd040a
Status	-1342718985
TimeoutCount	263488
test_base_val	1003943935
test_base_ptr	0x8cc2e8a8

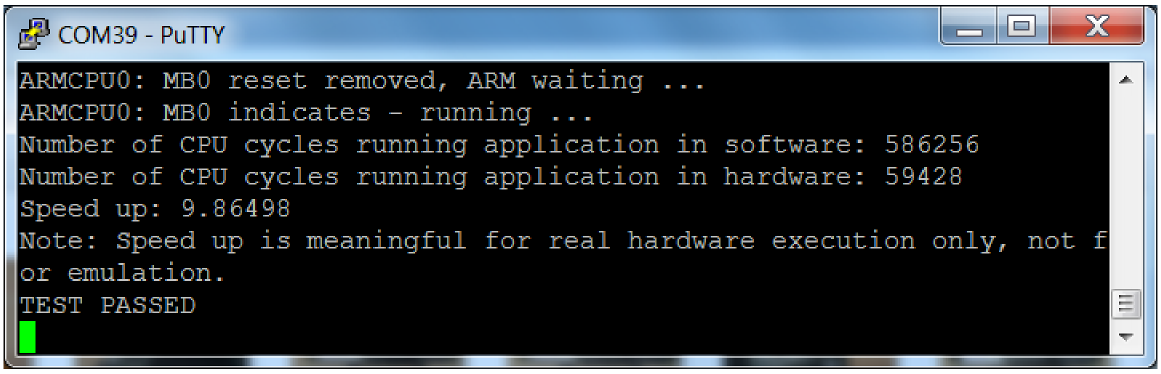
```
//    static XIntc intc;
    static XTmrCtr axi_timer_0_Timer;

    init_platform();

    XMutex_Config *ConfigPtr;
    XStatus Status;
    u32 TimeoutCount = 0;

    u32  test_base_val = 139266;
    u32 *test_base_ptr = 0x28000000;
    u32 iiii=0;
    u32 jjjj=0;
```

```
stdio.h
platform.h
xparameters.h
xtmrctr.h
tmrctr_header.h
TMRCTR_DEVICE_ID
TIMER_COUNTER_0
TimerCounter : XTmrCtr
xmutex.h
MUTEX_DEVICE_ID
MUTEX_NUM
Mutex : XMutex
xparameters.h
```

```
edkdsp_fp12_1x8_s Debug [Xilinx C/C++ application (GDB)] mb-gdb (05.01.18 10:31)
0x28100000 in _start ()

Temporary breakpoint 2, main () at ../src/fp12_1x8_s.c:2994
2994            init_platform();
```

Figure 20: Debug demo t01_s. Start the free-run from the debugger.

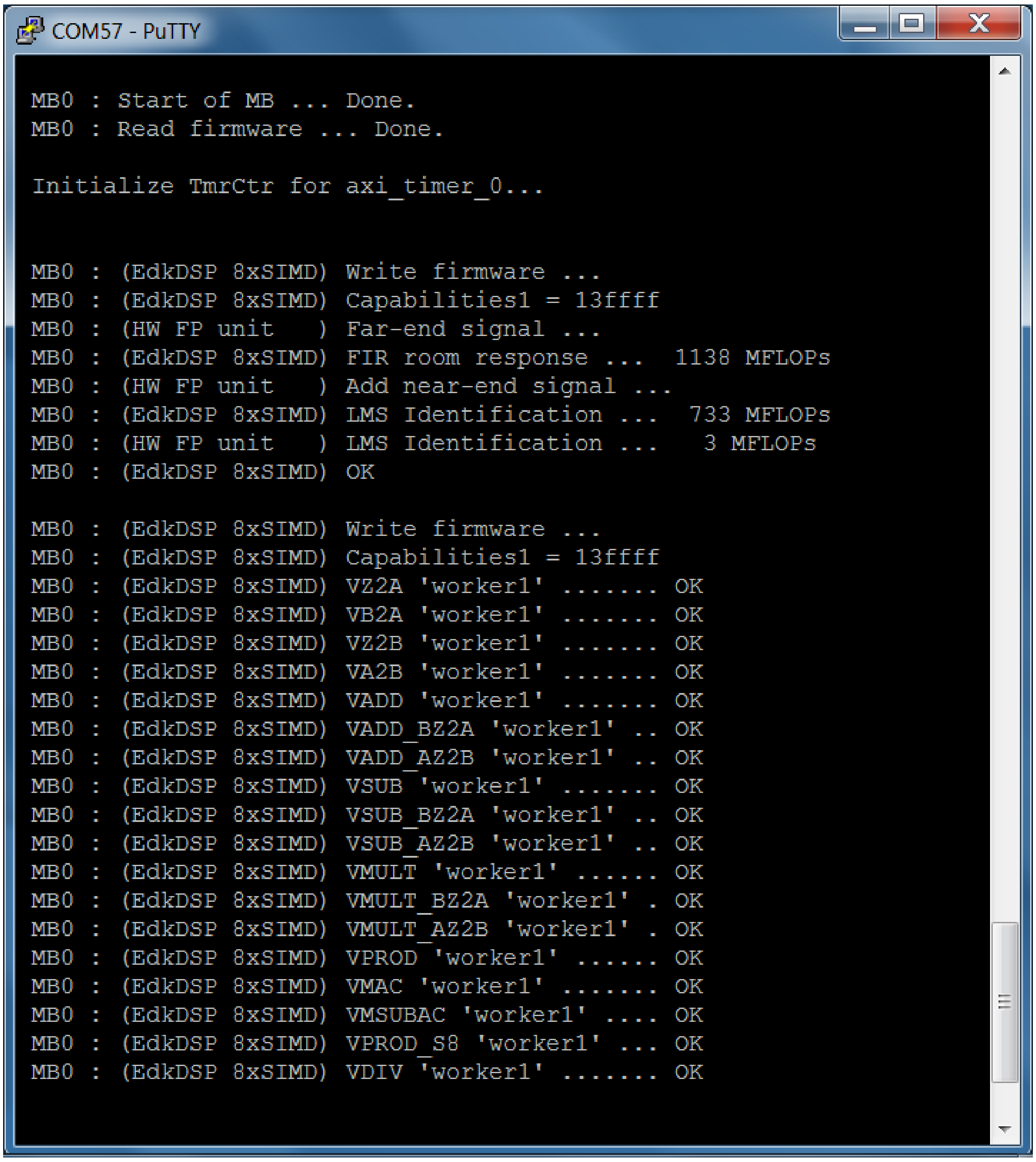
- In the SDK debugger, step through the MicroBlaze source code, inspect content of variables, set the breakpoints, step through the code and finally select the free run of the MicroBlaze code.
- At this stage, the ARM terminal will present the output from the ARM t01_s.elf example. See *Figure 21*.



```
ARMCPU0: MB0 reset removed, ARM waiting ...
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 586256
Number of CPU cycles running application in hardware: 59428
Speed up: 9.86498
Note: Speed up is meaningful for real hardware execution only, not f
or emulation.
TEST PASSED
```

Figure 21: Debug demo t01_s. Arm started EdkDSP and runs SDSoC akcelerátor demo.

The MicroBlaze terminal will present output from the debugged MicroBlaze and the 8xSIMD EdkDSP IP core. See *Figure 22*.



```
MB0 : Start of MB ... Done.
MB0 : Read firmware ... Done.

Initialize TmrCtr for axi_timer_0...


MB0 : (EdkDSP 8xSIMD) Write firmware ...
MB0 : (EdkDSP 8xSIMD) Capabilities1 = 13ffff
MB0 : (HW FP unit    ) Far-end signal ...
MB0 : (EdkDSP 8xSIMD) FIR room response ...  1138 MFLOPs
MB0 : (HW FP unit    ) Add near-end signal ...
MB0 : (EdkDSP 8xSIMD) LMS Identification ...  733 MFLOPs
MB0 : (HW FP unit    ) LMS Identification ...    3 MFLOPs
MB0 : (EdkDSP 8xSIMD) OK

MB0 : (EdkDSP 8xSIMD) Write firmware ...
MB0 : (EdkDSP 8xSIMD) Capabilities1 = 13ffff
MB0 : (EdkDSP 8xSIMD) VZ2A 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VB2A 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VZ2B 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VA2B 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VADD 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VADD_BZ2A 'worker1' .. OK
MB0 : (EdkDSP 8xSIMD) VADD_AZ2B 'worker1' .. OK
MB0 : (EdkDSP 8xSIMD) VSUB 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VSUB_BZ2A 'worker1' .. OK
MB0 : (EdkDSP 8xSIMD) VSUB_AZ2B 'worker1' .. OK
MB0 : (EdkDSP 8xSIMD) VMULT 'worker1' ...... OK
MB0 : (EdkDSP 8xSIMD) VMULT_BZ2A 'worker1' . OK
MB0 : (EdkDSP 8xSIMD) VMULT_AZ2B 'worker1' . OK
MB0 : (EdkDSP 8xSIMD) VPROD 'worker1' ...... OK
MB0 : (EdkDSP 8xSIMD) VMAC 'worker1' ....... OK
MB0 : (EdkDSP 8xSIMD) VMSUBAC 'worker1' .... OK
MB0 : (EdkDSP 8xSIMD) VPROD_S8 'worker1' ... OK
MB0 : (EdkDSP 8xSIMD) VDIV 'worker1' ....... OK
```

Figure 22: Debug demo t01_s. MicroBlaze project output (Compiled for Debug).

(8) In PC, open the Vivado Lab tool. See *Figure 7*.
- Open Hardware Manager.
- Press Auto Connect icon in Hardware window to connect to the board via JTAG line.
- Open description of debug nets present in file, thus specifying the probes file.

X:\SD_debug\t01_s\debug_nets.ltx

- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP. See *Figure 8*, *Figure 9*, *Figure 10*.

- Open new perspective and observe the chip temperature. See *Figure 11*. Close Vivado Lab tool project.

(9) In SDK debugger, stop MicroBlaze processor and close the debug session.
(10) Remove SD card and reprogram it in the PC to test another example.
(11) Go to step (6).

Installation and use of Extended Evaluation Package – linux examples

In case of Linux target:

(1) In Win 7 or Win 10 (64bit PC), unzip the basic evaluation package
TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL.zip
to directory of your choice. We will use:

C:\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\

Open new Xilinx SDK 2017.1 workspace in the directory

X:\zsys

Import (with copy) all SDK projects from

C:\TE71\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\zsys\

to the SDK.

(2) Select one of the examples (t01_l … t21_l) and copy the content of sd_card directory to the SD card.
Example. Copy content of the directory from
X:\SD_debug\t01_l\sd_card\
to the root of the SD card

(3) Connect USB cable from J7 connector to the PC. It will serve as ARM terminal and JTAG line.

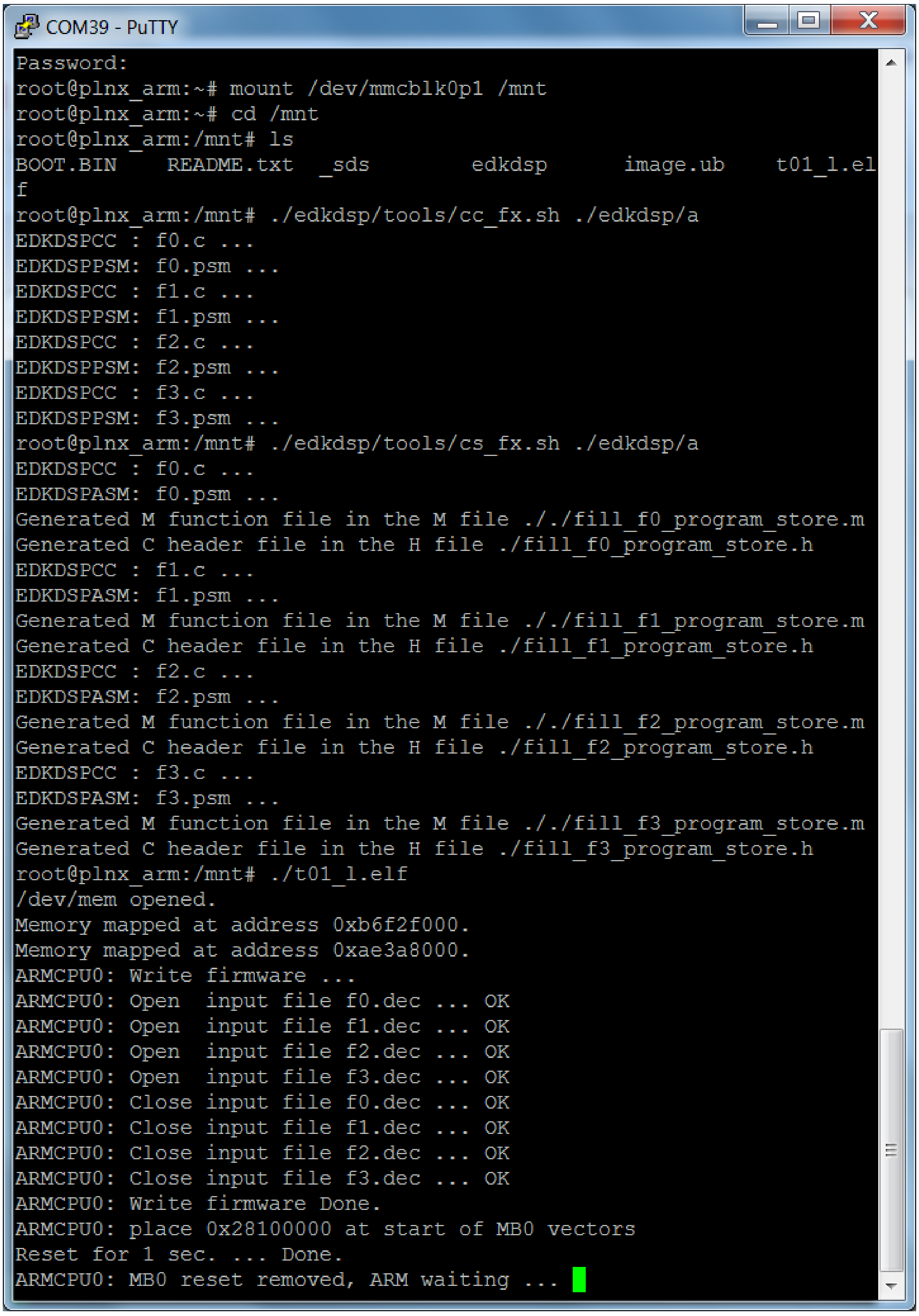
(4) Connect USB cable to the USBUART Pmod module present in the J5 connector to the PC. It will serve as MicroBlaze terminal.

(5) Power ON the carrier board. And open putty (or similar) terminal client for both USB serial lines.
Set the serial communication to [speed 115200, data bits 8, stop bits 1, parity none and flow control None] in both cases.

(6) Insert SD card to the TE0706-02 carrier board.

(7) Reset the carrier board.
- The linux system will start. See *Figure 23*.
- Type the linux user name and password:

root
root


COM39 - PuTTY

```
Password:
root@plnx_arm:~# mount /dev/mmcblk0p1 /mnt
root@plnx_arm:~# cd /mnt
root@plnx_arm:/mnt# ls
BOOT.BIN     README.txt  _sds        edkdsp      image.ub    t01_l.el
f
root@plnx_arm:/mnt# ./edkdsp/tools/cc_fx.sh ./edkdsp/a
EDKDSPCC : f0.c ...
EDKDSPPSM: f0.psm ...
EDKDSPCC : f1.c ...
EDKDSPPSM: f1.psm ...
EDKDSPCC : f2.c ...
EDKDSPPSM: f2.psm ...
EDKDSPCC : f3.c ...
EDKDSPPSM: f3.psm ...
root@plnx_arm:/mnt# ./edkdsp/tools/cs_fx.sh ./edkdsp/a
EDKDSPCC : f0.c ...
EDKDSPASM: f0.psm ...
Generated M function file in the M file ././fill_f0_program_store.m
Generated C header file in the H file ./fill_f0_program_store.h
EDKDSPCC : f1.c ...
EDKDSPASM: f1.psm ...
Generated M function file in the M file ././fill_f1_program_store.m
Generated C header file in the H file ./fill_f1_program_store.h
EDKDSPCC : f2.c ...
EDKDSPASM: f2.psm ...
Generated M function file in the M file ././fill_f2_program_store.m
Generated C header file in the H file ./fill_f2_program_store.h
EDKDSPCC : f3.c ...
EDKDSPASM: f3.psm ...
Generated M function file in the M file ././fill_f3_program_store.m
Generated C header file in the H file ./fill_f3_program_store.h
root@plnx_arm:/mnt# ./t01_l.elf
/dev/mem opened.
Memory mapped at address 0xb6f2f000.
Memory mapped at address 0xae3a8000.
ARMCPU0: Write firmware ...
ARMCPU0: Open  input file f0.dec ... OK
ARMCPU0: Open  input file f1.dec ... OK
ARMCPU0: Open  input file f2.dec ... OK
ARMCPU0: Open  input file f3.dec ... OK
ARMCPU0: Close input file f0.dec ... OK
ARMCPU0: Close input file f1.dec ... OK
ARMCPU0: Close input file f2.dec ... OK
ARMCPU0: Close input file f3.dec ... OK
ARMCPU0: Write firmware Done.
ARMCPU0: place 0x28100000 at start of MB0 vectors
Reset for 1 sec. ... Done.
ARMCPU0: MB0 reset removed, ARM waiting ...
```

Figure 23: Compiled EdkDSP firmware. Started debug demo - linux target t01_l.

- Mount SD card to the directory (See *Figure 23*) /mnt by typing:

```
  mount   /dev/mmcblk0p1   /mnt
```

- Change directory to /mnt

```
  cd /mnt
```

-Compile firmware for the PicoBlaze6 by the EdkDSP C compiler (see *Figure 23*) :
```
  ./edkdsp/tools/cc_fx.sh ./edkdsp/a
  (or ./edkdsp/tools/cc_fx.sh ./edkdsp/b  or ./edkdsp/tools/cc_fx.sh ./edkdsp/c etc.)
```

- The PicoBlaze6 C source code files from the directory ./edkdsp/a
```
   ./edkdsp/a/f0.c  ./edkdsp/a/f1.c  ./edkdsp/a/f2.c  ./edkdsp/a/f3.c
  (or from the directory ./edkdsp/b or  from the directory ./edkdsp/c   etc.)
```

   are compiled by the EdkDSP C compiler to the firmware files:

```
   ./f0.dec    ./f1.dec   ./f2.dec  ./f3.dec
```

- Optionally, you can also compile the PicoBlaze6 firmware into header files for the standalone target. Compile firmware for the PicoBlaze6 by the EdkDSP C compiler (See *Figure 23*):
```
  ./edkdsp/tools/cs_fx.sh   ./edkdsp/a
  (or ./edkdsp/tools/cs_fx.sh ./edkdsp/b  or ./edkdsp/tools/cs_fx.sh ./edkdsp/c   etc.)
```

Generated header files with PicoBlaze6 firmware for the standalone target EdkDSP IP target are created and stored in the SD card root directory:

```
 ./fill_f0_program_store.h  ./fill_f1_program_store.h
 ./fill_f2_program_store.h   ./fill_f3_program_store.h
```

These headers serve for the standalone MicroBlaze projects. Headers are compiled directly into the debugged MicroBlaze standalone application as described above.

- Execute the ARM linux application See *Figure 23*.

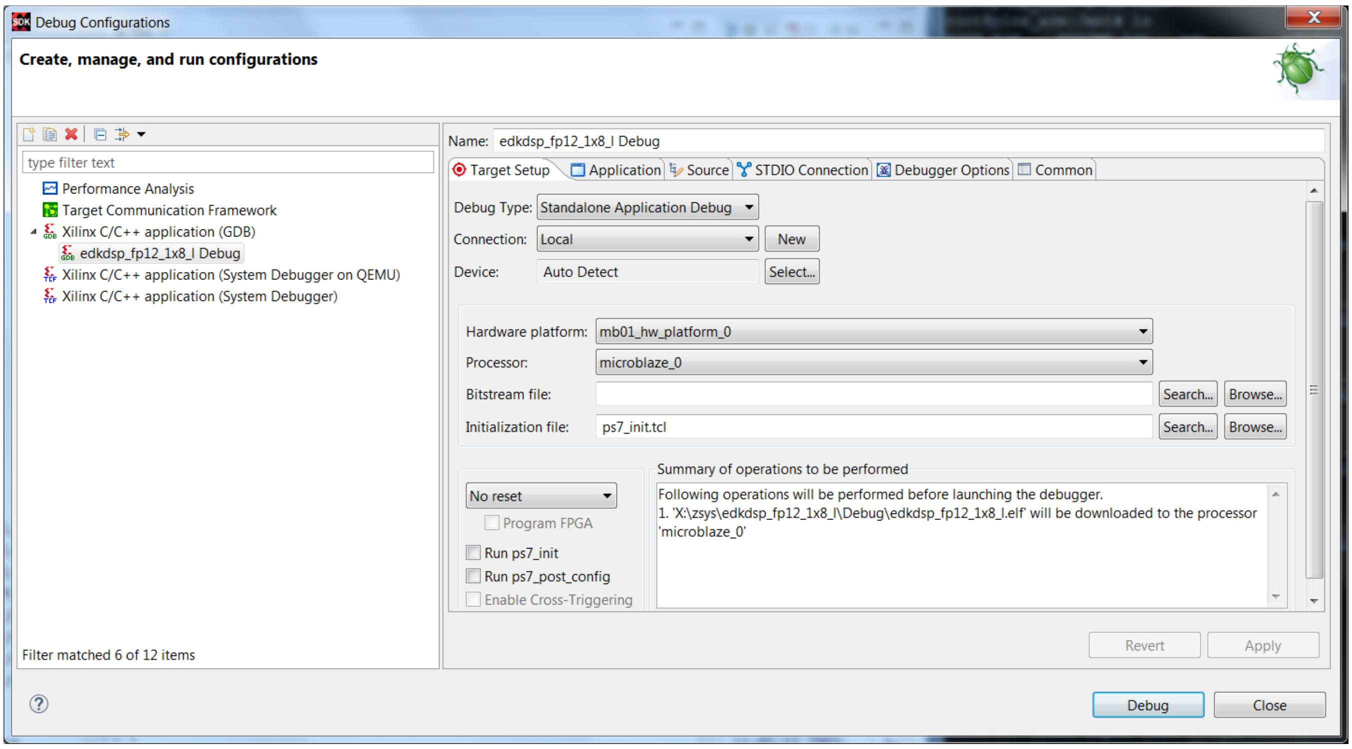
- The ARM terminal will present output from the EdkDSP C compiler
- The MicroBlaze terminal will present output from the 8xSIMD EdkDSP IP

- Start the linux application by typing

```
./t01_l.elf
```

- The ARM terminal will present output from the t01_l.elf example. The Arm application is waiting for the MicroBlaze in this stage.

- In the Xilinx SDK environment on the PC, select debug project (See *Figure 24*):
```
  X:\zsys\edkdsp_fp12_1x8_l
```


Debug Configurations

Create, manage, and run configurations

type filter text

- Performance Analysis
- Target Communication Framework
- Xilinx C/C++ application (GDB)
  - edkdsp_fp12_1x8_l Debug
- Xilinx C/C++ application (System Debugger on QEMU)
- Xilinx C/C++ application (System Debugger)

Filter matched 6 of 12 items

Name: edkdsp_fp12_1x8_l Debug

Target Setup | Application | Source | STDIO Connection | Debugger Options | Common

Debug Type: Standalone Application Debug
Connection: Local
Device: Auto Detect

Hardware platform: mb01_hw_platform_0
Processor: microblaze_0
Bitstream file:
Initialization file: ps7_init.tcl

No reset
Program FPGA
Run ps7_init
Run ps7_post_config
Enable Cross-Triggering

Summary of operations to be performed

```
Following operations will be performed before launching the debugger.
1. 'X:\zsys\edkdsp_fp12_1x8_l\Debug\edkdsp_fp12_1x8_l.elf' will be downloaded to the processor
'microblaze_0'
```

Figure 24: Select MicroBlaze project edkdsp_fp12_1x8_l for debug.

- In the SDK debugger, step through the MicroBlaze source code, inspect the content of variables, set breakpoints etc. See
Figure 25.

- In the SDK debugger, select free run of the MicroBlaze code. See
Figure 25.

- The MicroBlaze terminal will present output from the 8xSIMD EdkDSP IP working with new firmware programs as re-compiled by the EdkDSP C compiler from the C source code files: f0.c, f1.c, f2.c and f3.c from the directory ./edkdsp/a
Output is identical to *Figure 22*.

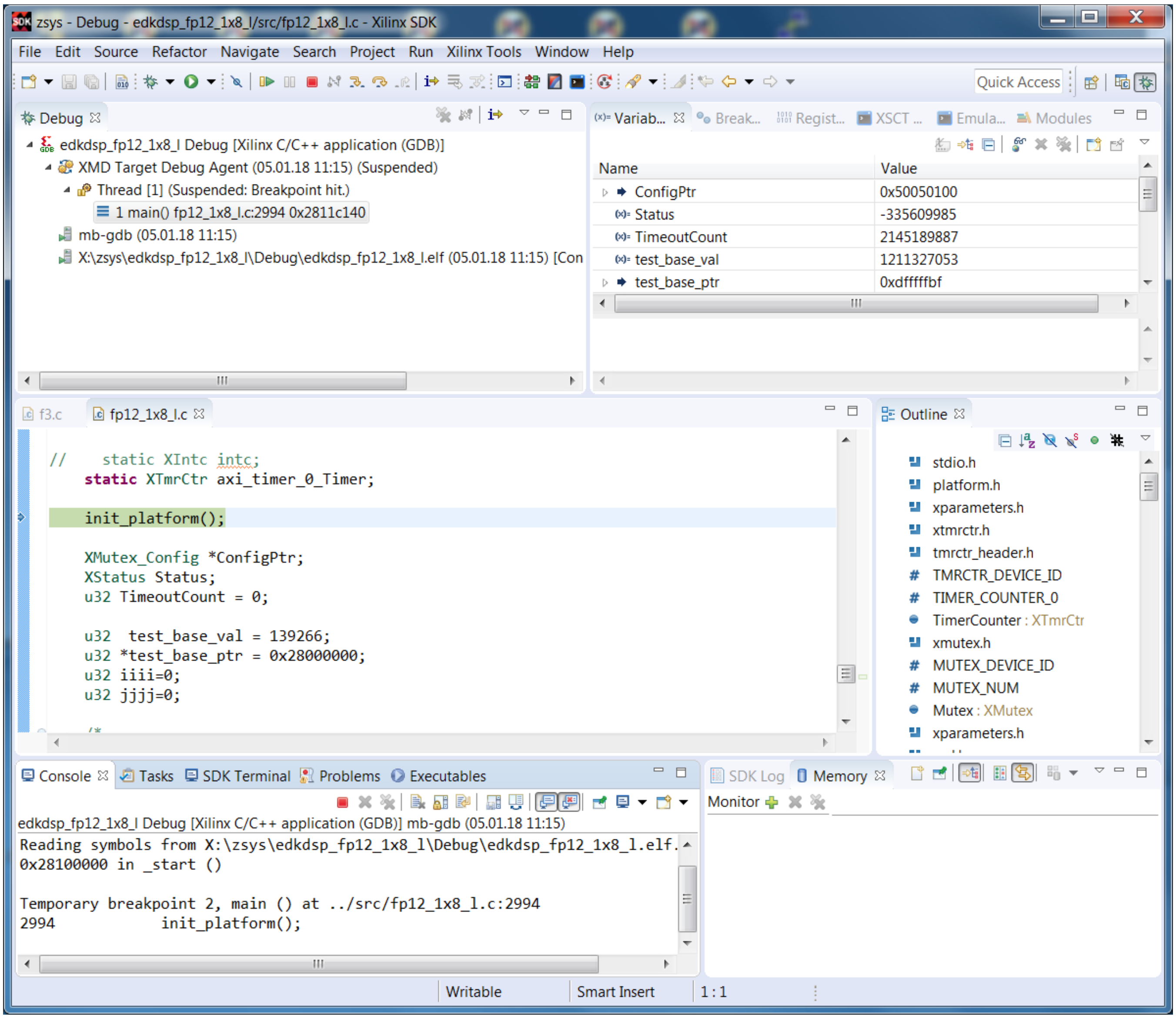
- The ARM terminal will continue to present output from the t01_l.elf example. See *Figure 26*.

- In ARM terminal, type:

```
ls - lr
```

to see listing of files compiled by the EdkDSP C compiler. See *Figure 26*.

The compiled header files fill_f0_program_store.h, fill_f1_program_store.h, fill_f2_program_store.h, and fill_f3_program_store.h. can be used as new source code for the standalone MicroBlaze project X:\zsys\edkdsp_fp12_1x8_s .


zsys - Debug - edkdsp_fp12_1x8_l/src/fp12_1x8_l.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Tools Window Help
Quick Access
Debug
Variab... Break... Regist... XSCT ... Emula... Modules
f3.c fp12_1x8_l.c
Outline
Console Tasks SDK Terminal Problems Executables
SDK Log Memory
Monitor
Writable Smart Insert 1 : 1

```
edkdsp_fp12_1x8_l Debug [Xilinx C/C++ application (GDB)]
  XMD Target Debug Agent (05.01.18 11:15) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() fp12_1x8_l.c:2994 0x2811c140
  mb-gdb (05.01.18 11:15)
  X:\zsys\edkdsp_fp12_1x8_l\Debug\edkdsp_fp12_1x8_l.elf (05.01.18 11:15) [Con
```

Name	Value
ConfigPtr	0x50050100
Status	-335609985
TimeoutCount	2145189887
test_base_val	1211327053
test_base_ptr	0xdfffffbf

```
//    static XIntc intc;
    static XTmrCtr axi_timer_0_Timer;

    init_platform();

    XMutex_Config *ConfigPtr;
    XStatus Status;
    u32 TimeoutCount = 0;

    u32  test_base_val = 139266;
    u32 *test_base_ptr = 0x28000000;
    u32 iiii=0;
    u32 jjjj=0;
```

```
stdio.h
platform.h
xparameters.h
xtmrctr.h
tmrctr_header.h
TMRCTR_DEVICE_ID
TIMER_COUNTER_0
TimerCounter : XTmrCtr
xmutex.h
MUTEX_DEVICE_ID
MUTEX_NUM
Mutex : XMutex
xparameters.h
```

```
edkdsp_fp12_1x8_l Debug [Xilinx C/C++ application (GDB)] mb-gdb (05.01.18 11:15)
Reading symbols from X:\zsys\edkdsp_fp12_1x8_l\Debug\edkdsp_fp12_1x8_l.elf.
0x28100000 in _start ()

Temporary breakpoint 2, main () at ../src/fp12_1x8_l.c:2994
2994            init_platform();
```

Figure 25: Select free run of MicroBlaze project edkdsp_fp12_1x8_l.

(8) In PC, open the Vivado Lab tool. See *Figure 7*.
- Open Hardware Manager.
- Press Auto Connect icon in Hardware window to connect to the board via JTAG line
- Open description of debug nets present in file, thus specifying the probes file

- Open description of debug nets present in file

X:\SD_debug\t01_l\debug_nets.ltx

- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP.
See *Figure 8*, *Figure 9*, *Figure 10*.

(9) Close Vivado Lab tool project.

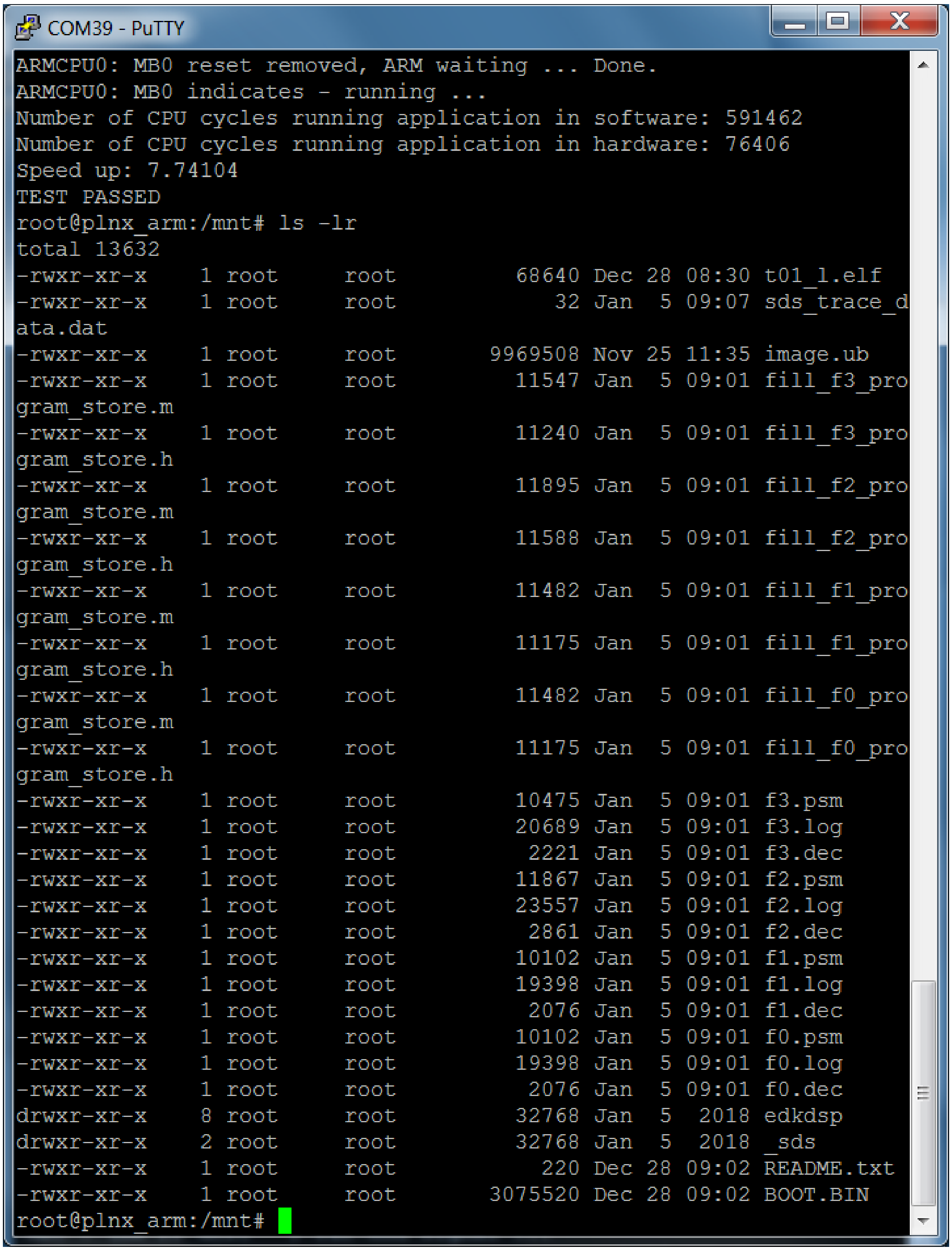
(10) In SDK debugger, stop MicroBlaze processor and close the debug session.

(11) Exit from linux by typing on the ARM terminal:

```
exit
```

(12) Remove SD card and reprogram it in the PC to test another example.

(13) Go to step (6).

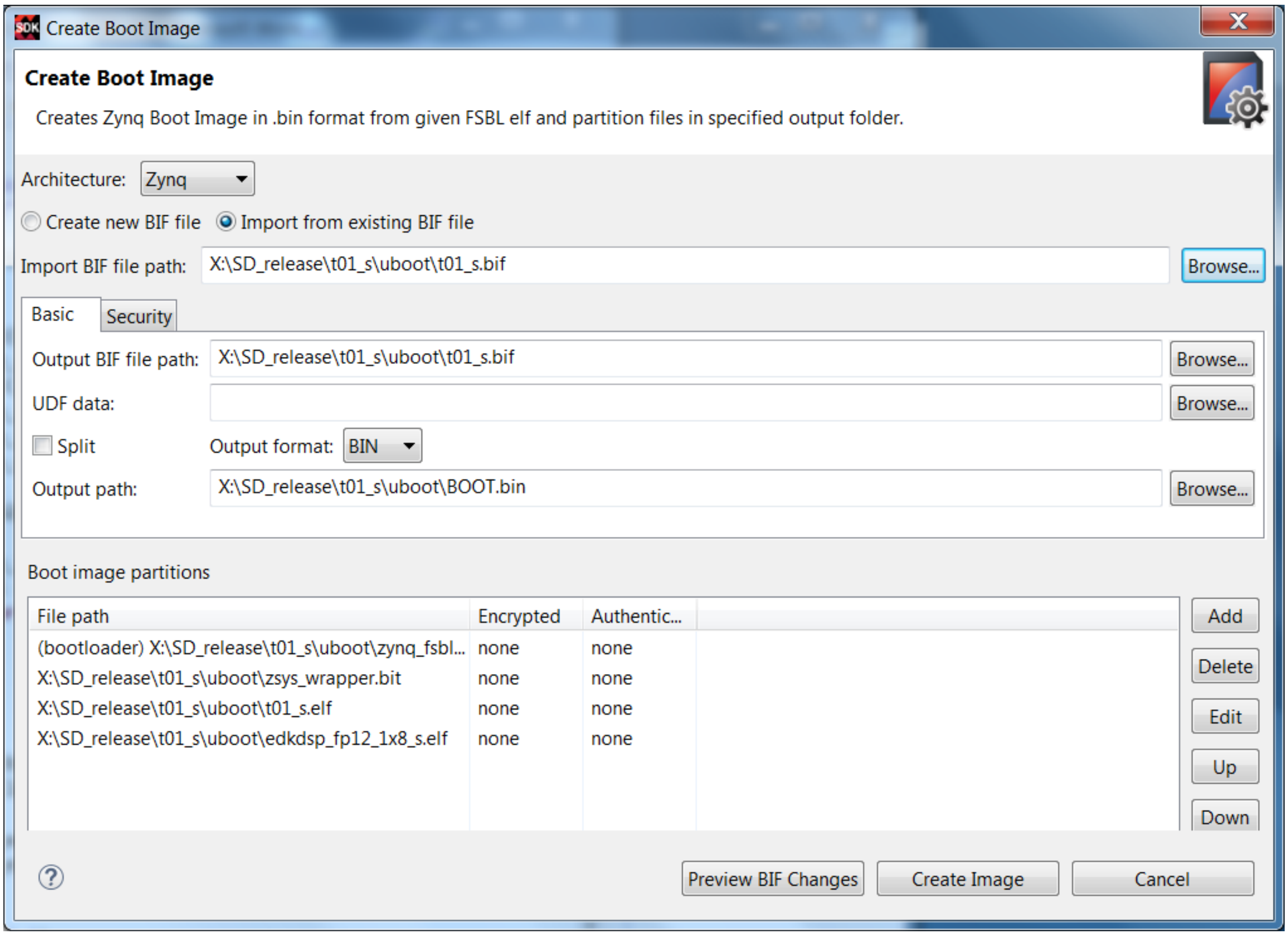


```
ARMCPU0: MB0 reset removed, ARM waiting ... Done.
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 591462
Number of CPU cycles running application in hardware: 76406
Speed up: 7.74104
TEST PASSED
root@plnx_arm:/mnt# ls -lr
total 13632
-rwxr-xr-x    1 root     root         68640 Dec 28 08:30 t01_l.elf
-rwxr-xr-x    1 root     root            32 Jan  5 09:07 sds_trace_d
ata.dat
-rwxr-xr-x    1 root     root       9969508 Nov 25 11:35 image.ub
-rwxr-xr-x    1 root     root         11547 Jan  5 09:01 fill_f3_pro
gram_store.m
-rwxr-xr-x    1 root     root         11240 Jan  5 09:01 fill_f3_pro
gram_store.h
-rwxr-xr-x    1 root     root         11895 Jan  5 09:01 fill_f2_pro
gram_store.m
-rwxr-xr-x    1 root     root         11588 Jan  5 09:01 fill_f2_pro
gram_store.h
-rwxr-xr-x    1 root     root         11482 Jan  5 09:01 fill_f1_pro
gram_store.m
-rwxr-xr-x    1 root     root         11175 Jan  5 09:01 fill_f1_pro
gram_store.h
-rwxr-xr-x    1 root     root         11482 Jan  5 09:01 fill_f0_pro
gram_store.m
-rwxr-xr-x    1 root     root         11175 Jan  5 09:01 fill_f0_pro
gram_store.h
-rwxr-xr-x    1 root     root         10475 Jan  5 09:01 f3.psm
-rwxr-xr-x    1 root     root         20689 Jan  5 09:01 f3.log
-rwxr-xr-x    1 root     root          2221 Jan  5 09:01 f3.dec
-rwxr-xr-x    1 root     root         11867 Jan  5 09:01 f2.psm
-rwxr-xr-x    1 root     root         23557 Jan  5 09:01 f2.log
-rwxr-xr-x    1 root     root          2861 Jan  5 09:01 f2.dec
-rwxr-xr-x    1 root     root         10102 Jan  5 09:01 f1.psm
-rwxr-xr-x    1 root     root         19398 Jan  5 09:01 f1.log
-rwxr-xr-x    1 root     root          2076 Jan  5 09:01 f1.dec
-rwxr-xr-x    1 root     root         10102 Jan  5 09:01 f0.psm
-rwxr-xr-x    1 root     root         19398 Jan  5 09:01 f0.log
-rwxr-xr-x    1 root     root          2076 Jan  5 09:01 f0.dec
drwxr-xr-x    8 root     root         32768 Jan  5  2018 edkdsp
drwxr-xr-x    2 root     root         32768 Jan  5  2018 _sds
-rwxr-xr-x    1 root     root           220 Dec 28 09:02 README.txt
-rwxr-xr-x    1 root     root       3075520 Dec 28 09:02 BOOT.BIN
root@plnx_arm:/mnt#
```

Figure 26: Output from ARM MicroBlaze fort t01_l. Compiled EdkDSP firmware.

Updating of the release SD card images for new standalone-release-target

Modified Picoblaze6 C source code can be compiled to firmware headers in the embedded EdkDSP C compiler (linux target). Resulting headers can be included in the SDK MicroBlaze standalone release target project. See *Figure 17*. The standalone-release-target SD card image can be updated by re-compilation of the (possibly modified) C source code for the MicroBlaze in the SDK project with included updated PicoBlaze firmware header files. See *Figure 27*.


Create Boot Image

Create Boot Image

Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq

Create new BIF file [✓ Import from existing BIF file]

Import BIF file path: X:\SD_release\t01_s\uboot\t01_s.bif

Basic | Security

Output BIF file path: X:\SD_release\t01_s\uboot\t01_s.bif

UDF data:

Split: ☐ Output format: BIN

Output path: X:\SD_release\t01_s\uboot\BOOT.bin

Boot image partitions

File path	Encrypted	Authentic...
(bootloader) X:\SD_release\t01_s\uboot\zynq_fsbl...	none	none
X:\SD_release\t01_s\uboot\zsys_wrapper.bit	none	none
X:\SD_release\t01_s\uboot\t01_s.elf	none	none
X:\SD_release\t01_s\uboot\edkdsp_fp12_1x8_s.elf	none	none

Figure 27: Create BOOT.bin for the t01_s demo.

Copy the content of directory:
C:\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\SD_release\t01_s\uboot\
to
X:\SD_release\t01_s\uboot\

The new BOOT.bin image can be created from these files:

X:\SD_release\t01_s\uboot\t01_s.bif
X:\SD_release\t01_s\uboot\zynq_fsbl.elf
X:\SD_release\t01_s\uboot\zynq_wrapper.bit.elf
X:\SD_release\t01_s\uboot\t01_s.elf
X:\SD_release\t01_s\uboot\edkdsp_fp12_1x8_s.elf

Replace the old
X:\SD_release\t01_l\uboot\edkdsp_fp12_1x8_s.elf
with the new file recompiled in the SDK (with new PicoBlaze6 firmware headers) from the SDK project directory:
X:\zsys\edkdsp_fp12_1x8_s\Release\edkdsp_fp12_1x8_s.elf

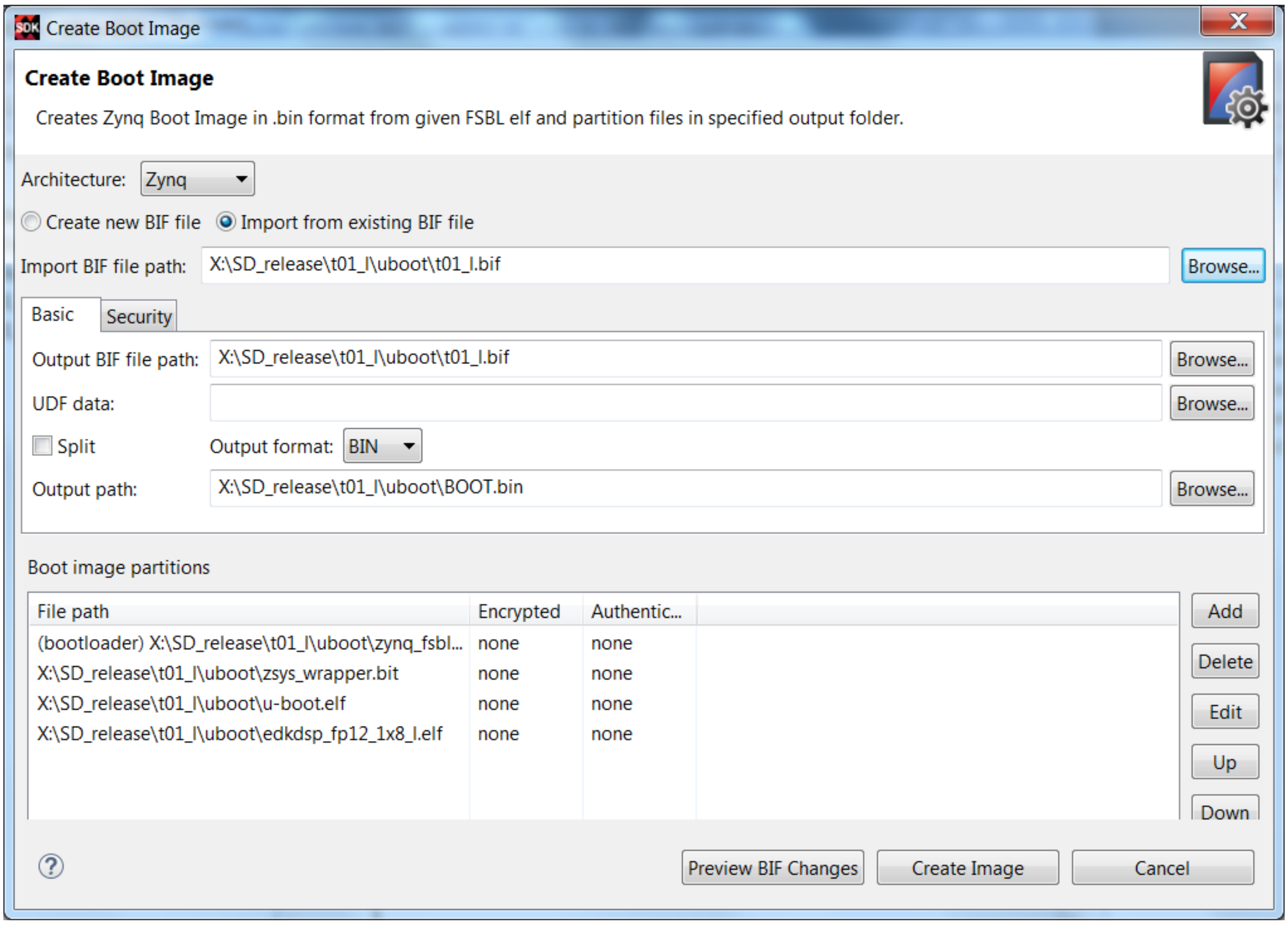
Use the BOOT.bin generation utility (In the SDK workspace: Xilinx Tools -> Create Boot Image) and create the new BOOT.bin file (See *Figure 27*):
X:\SD_release\t01_s\uboot\BOOT.bin

Copy this new BOOT.bin file it to:
X:\SD_release\t01_s\sd_card\BOOT.bin

The content of the standalone-release-target SD card is updated with new MicroBlaze and PicoBlaze6 firmware.

Updating of the release SD card images for new linux-release-target

The linux-release-target SD card image can be updated by re-compilation of the (possibly modified) C source code for the MicroBlaze in the SDK project. See *Figure 28*.


Create Boot Image

Create Boot Image

Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq

( ) Create new BIF file [✓ Import from existing BIF file]

Import BIF file path: X:\SD_release\t01_l\uboot\t01_l.bif

Basic | Security

Output BIF file path: X:\SD_release\t01_l\uboot\t01_l.bif

UDF data:

☐ Split   Output format: BIN

Output path: X:\SD_release\t01_l\uboot\BOOT.bin

Boot image partitions

File path	Encrypted	Authentic...
(bootloader) X:\SD_release\t01_l\uboot\zynq_fsbl...	none	none
X:\SD_release\t01_l\uboot\zsys_wrapper.bit	none	none
X:\SD_release\t01_l\uboot\u-boot.elf	none	none
X:\SD_release\t01_l\uboot\edkdsp_fp12_1x8_l.elf	none	none

Figure 28: Create BOOT.bin for the t01_l demo.

Copy the content of directory:
C:\TE0720_MB_EdkDSP_1x8_zsys_SDK_2if_ila_8k_usb_sw1_INSTALL\SD_release\t01_l\uboot\
to
X:\SD_release\t01_l\uboot\

The new BOOT.bin image can be created from these files:

X:\SD_release\t01_l\uboot\t01_l.bif
X:\SD_release\t01_l\uboot\zynq_fsbl.elf
X:\SD_release\t01_l\uboot\zynq_wrapper.bit.elf
X:\SD_release\t01_l\uboot\u-boot.elf
X:\SD_release\t01_l\uboot\edkdsp_fp12_1x8_l.elf

Replace

X:\SD_release\t01_l\uboot\edkdsp_fp12_1x8_l.elf

with a new file recompiled in the SDK project directory:

X:\zsys\edkdsp_fp12_1x8_l\Release\edkdsp_fp12_1x8_l.elf

Use the BOOT.bin generation utility of the SDK and create the new BOOT.bin file:

X:\SD_release\t01_l\uboot\BOOT.bin

Copy this new BOOT.bin file to:
X:\SD_release\t01_l\sd_card\BOOT.bin

Update the linux-release-target SD card with the actual PicoBlaze6 C programs and actual firmware files compiled from C source code by the embedded EdkDSP C compiler:

X:\SD_release\t01_l\sd_card\edkdsp\a\f0.c
X:\SD_release\t01_l\sd_card\edkdsp\a\f1.c
X:\SD_release\t01_l\sd_card\edkdsp\a\f2.c
X:\SD_release\t01_l\sd_card\edkdsp\a\f3.c

X:\SD_release\t01_l\sd_card\f0.dec
X:\SD_release\t01_l\sd_card\f1.dec
X:\SD_release\t01_l\sd_card\f2.dec
X:\SD_release\t01_l\sd_card\f3.dec

The content of the linux-release-target SD card is updated with new MicroBlaze and PicoBlaze6 firmware.

10. References

[1] TE0720-2IF; Part: XC7Z020-2CLG484I; 1 GByte DDR; Grade: Industrial;
http://shop.trenz-electronic.de/en/TE0720-03-2IF-Xilinx-Zynq-module-XC7Z020-2CLG484I-ind.-temp.-range-1-Gbyte

https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/TE0720/REV02/Documents/TE0720%20User%20Manual-v45-20150323_1407.pdf

https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/TE0720/REV03/Documents/TRM-TE0720-03.pdf

[2] Heatsink for TE0720, spring-loaded embedded;
https://shop.trenz-electronic.de/en/26922-Heatsink-for-TE0720-spring-loaded-embedded?c=38

[3] TE0706 - Carrierboard for Trenz Electronic Modules with 4 x 5 cm Form factor
https://shop.trenz-electronic.de/en/TE0706-02-TE0706-Carrierboard-for-Trenz-Electronic-Modules-with-4-x-5-cm-Form-factor?c=261

https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/carrier_boards/TE0706/REV02/documents/SCH-TE0706-02.PDF

[4] Pmod USBUART: Serial converter & interface.
https://shop.trenz-electronic.de/en/24242-Pmod-USBUART-USB-to-UART-Interface?c=80

[5] XMOD FTDI JTAG Adapter - Xilinx compatible
https://shop.trenz-electronic.de/en/TE0790-02-XMOD-FTDI-JTAG-Adapter-Xilinx-compatible

[6] Vivado HLx Web Install Client - 2017.1.
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2015-4.html

[7] SDSoC - 2017.1 Full Product Installations.
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/sdx-development-environments/sdsoc/2015-4.html

[8] PRODUCTIVE 4.0 Project www page in UTIA with pointers to evaluation packages for download
http://sp.utia.cz/index.php?ids=projects/productive40

11. Base Evaluation Package

The **base evaluation package** can be downloaded from UTIA www pages [8] free of charge.

Deliverables:
The base evaluation package [8] includes evaluation bitstreams with single (8xSIMD) EdkDSP IP working in parallel with selected HW-accelerated SDSoC algorithms on the Trenz Electronic TE0720-2IF module [1] located on the Trenz Electronic TE0706-02 carrier [3] with PMOD USBUART adapter [4] and XMOD FTDI JTAG Adapter [5].

The evaluation package [8] includes bitstreams compiled with the evaluation version of the (8xSIMD) EdkDSP IP core. Bitstreams contain these IPs:

bce_fp12_1x8_0_axiw_v1_10_c	Evaluation version of the AXI-lite interface
bce_fp12_1x8_40	Evaluation version of the floating point data path

The base evaluation version of the (8xSIMS) EdkDSP IP is compiled into bitstreams with a HW limit on number of vector operations. The termination of the nonexclusive, non-transferable evaluation license of this evaluation IP core is reported in advance by the demonstrator on the PMOD USBUART terminal. The evaluation designs run again after reset (TE0706-02 button S2).

The base evaluation package [8] includes these binary applications:

edkdsppp.elf	EdkDSP C pre-processor binary for ARM PetaLinux running on the evaluation board.
edkdspcc.elf	EdkDSP C compiler binary for ARM PetaLinux running on the evaluation board.
edkdsppsm.elf	EdkDSP ASM compiler binary for ARM PetaLinux running on the evaluation board.

These binary applications have no time restriction. The user of the evaluation package has nonexclusive, non-transferable license from UTIA to use these utilities for compilation of the firmware for the Xilinx PicoBlaze6 processor inside of the 8xSIMD EdkDSP IP in precompiled designs. The source code of these compilers is owned by UTIA and it is not provided in the evaluation package.

The base evaluation package [8] includes demonstration firmware in C source code for the Xilinx PicoBlaze6 processor for the family of UTIA EdkDSP accelerators for the Trenz Electronic TE0720-2IF module [1] on Trenz Electronic TE0706-02 carrier board [3].

HW boards are not part of deliverables. HW can be ordered separately from [1] – [5].

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. See also the Disclaimer section.

12. Extended Evaluation Package for PRODUCTIVE 4.0 partners

This extended evaluation package includes **MicroBlaze and PicoBlaze6 C code and precompiled bitstreams of HW projects** for the Trenz Electronic TE0720-2IF module [1] located on the Trenz Electronic TE0706-02 carrier [3] with PMOD USBUART adapter [4] and XMOD FTDI JTAG Adapter [5] **with the evaluation version of the (8xSIMD) EdkDSP IP. Partners of the ECSEL PRODUCTIVE 4.0 project [8]** can order this extended package from UTIA AV CR, v.v.i., by email request for quotation to kadlec@utia.cas.cz.

UTIA AV CR, v.v.i., will provide to the PRODUCTIVE 4.0 project partner quotation by email. After confirmation of the quotation by the customer, UTIA AV CR, v.v.i., will send to the customer this invoice:

The extended evaluation package with MicroBlaze and PicoBlaze6 C code and precompiled bitstream of HW projects for the Trenz Electronic TE0720-2IF module [1] located on the Trenz Electronic TE0706-02 carrier [3] with PMOD USBUART adapter [4] and XMOD FTDI JTAG Adapter [5] with the evaluation version of the 8xSIMD EdkDSP IP for the partners in the ECSEL PRODUCTIVE 4.0 project
(Without VAT) **0,00 Eur**

After receiving confirmation from the PRODUCTIVE 4.0 project partner about the zero-invoice received, UTIA AV CR, v.v.i. will send within 5 working days by standard mail printed version of this application note together with DVD with the Deliverables described in this section.

Deliverables:
The extended evaluation package for PRODUCTIVE 4.0 partners [8] includes MicroBlaze and PicoBlaze6 C code and precompiled bitstreams of HW projects. MicroBlaze and PicoBlaze6 SW projects can be modified and recompiled by the PRODUCTIVE 4.0 project partner.

The extended evaluation version of the UTIA 8xSIMD EdkDSP accelerator IP is provided in precompiled bitstreams of HW projects with these IPs:

bce_fp12_1x8_0_axiw_v1_10_c	Evaluation version of the AXI-lite interface
bce_fp12_1x8_40	Evaluation version of the floating point data path

The extended evaluation version of the 8xSIMS EdkDSP IP is compiled into bitstream with an HW limit on number of vector operations. The termination of the nonexclusive, non-transferable evaluation license of this evaluation IP core is reported in advance by the demonstrator on the PMOD USBUART terminal. The evaluation designs run again after the reset.

The extended evaluation package [8] includes these binary applications:

edkdsppp.elf	EdkDSP C pre-processor binary for ARM PetaLinux running on the evaluation board.
edkdspcc.elf	EdkDSP C compiler binary for ARM PetaLinux running on the evaluation board.
edkdsppsm.elf	EdkDSP ASM compiler binary for ARM PetaLinux running on the evaluation board.
edkdspasm.elf	EdkDSP ASM compiler binary for ARM PetaLinux running on the evaluation board.

These binary applications have no time restriction. The user of the evaluation package has nonexclusive, non-transferable license from UTIA to use these utilities for compilation of the firmware for the Xilinx PicoBlaze6 processor inside of the UTIA EdkDSP accelerators in precompiled designs. The source code of these compilers is owned by UTIA and it is not provided in the evaluation package.

The extended evaluation package for PRODUCTIVE 4.0 partners includes demonstration firmware in C source code for the Xilinx PicoBlaze6 processor for the family of UTIA EdkDSP accelerators for the Trenz Electronic TE0720-2IF module [1] on Trenz Electronic TE0706-02 carrier board [3].

The extended evaluation package for PRODUCTIVE 4.0 partners includes SDK SW projects with C source code for MicroBlaze. The extended evaluation package [8] includes static library for MicroBlaze processor:

libwal.a	SDK 2017.1 UTIA static library with EdkDSP API for MicroBlaze

This library has no time restriction. Source code of this library is not provided in this evaluation package.

HW boards are not part of deliverables. HW can be ordered separately from references [1] – [5].

Partners of the ECSEL PRODUCTIVE 4.0 project [8] can order the hardware [1] - [5] directly from the company Trenz Electronic or **order the complete evaluation system from UTIA AV CR, v.v.i**.

In case of an order from UTIA AV CR, v.v.i., an email request for a quotation to kadlec@utia.cas.cz is required. UTIA AV CR, v.v.i., will provide to the PRODUCTIVE 4.0 project partner quotation by email. After confirmation of the quotation by the PRODUCTIVE 4.0 project partner, UTIA AV CR, v.v.i., will buy from company Trenz Electronic boards [1]-[5] with cables and power supply. UTIA will assemble and test the complete evaluation system and send them to the PRODUCTIVE 4.0 project partner for price identical to the price offered by the company Trenz Electronic plus the transport cost and the VAT.

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. See also the Disclaimer section.

Disclaimer