Arrowhead Compatible Zynq with SDSoC 2017.4 and Floating-Point 8xSIMD EdkDSP Accelerators

Supported Trenz Electronic Modules:
TE0720-03-2IF, TE0720-03-1QF, TE0720-03-14S-1C, TE0720-03-1CFA
Supported Trenz Electronic Carrier Boards: TE0703-05, TE0706-02

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). For Vivado and SDK ver. 2017.1
2	30.01.2018	Jiří Kadlec	Demonstrator description prior to the Productive 4.0 project conference in Athens 6-7.3.2018 For Vivado and SDK ver. 2017.1.
3	15.05.2018	Jiří Kadlec	Revision for Vivado and SDK ver. 2017.4.1 Supported Trenz Electronic Zynq modules: TE0720-03-2IF, TE0720-03-1QF, TE0720-03-14S-1C Supported Trenz Electronic Carrier Boards: TE0703-05, TE0706-02
4	18.05.2018	Jiří Kadlec	Added board line description and licensing conditions for the development package.
5	24.05.2018	Jiří Kadlec	Added support for TE0720-03-1CFA-S starter kit
6	05.04.2019	Jiri Kadlec	Support for Debian and Arrowhead framework G4.0 on RaspberryPi 3B (lite installation) C++ Clients: Producer and Consumer running on Zynq TE0720 devices.
7	09.05.2019	Jiri Kadlec	V2: Updated Fig. 36. Evaluation package includes Image of SD card with Debian for TE0720-03-2IF with SDSoC te06_l application, Arrowhead Producer and Consumer.

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 one or 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. 2 carrier boards and 3 Zynq modules from Trenz Electronic are supported.

The demonstrated Zynq systems include 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 2017.4.1 design environment. The SDSoC is supporting automated compilation of user-defined C/C++ ARM functions into HW accelerators with several types of data movers (zero-copy, DMA, SG-DMA) and the automated integration of generated accelerators as an ARM Linux operating system or standalone application.

Debian image is provided for the Zynq board in format of image for the SD card. Chapter 10 describes simple installtion of additional SW packages and templates to get compatibility with Arrowhead framework G4.0 Java services. These services run together with the Arrowhead database on a separate RaspberryPi 3B board and form example of an Arrowhead local cloud. See *Figure 32*.

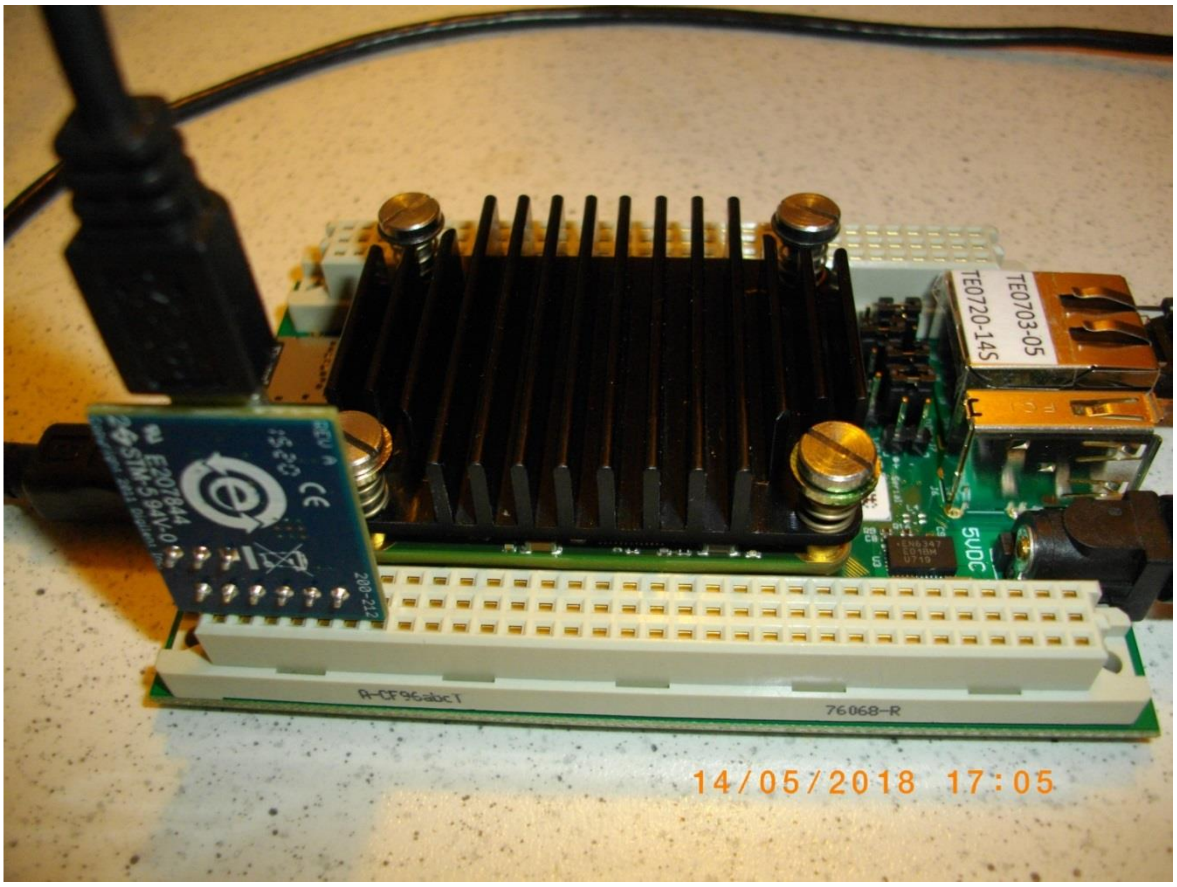


Figure 1: TE0703-05 carrier board with TE0720-03-14S-1C Zynq module

2. Implementation Details

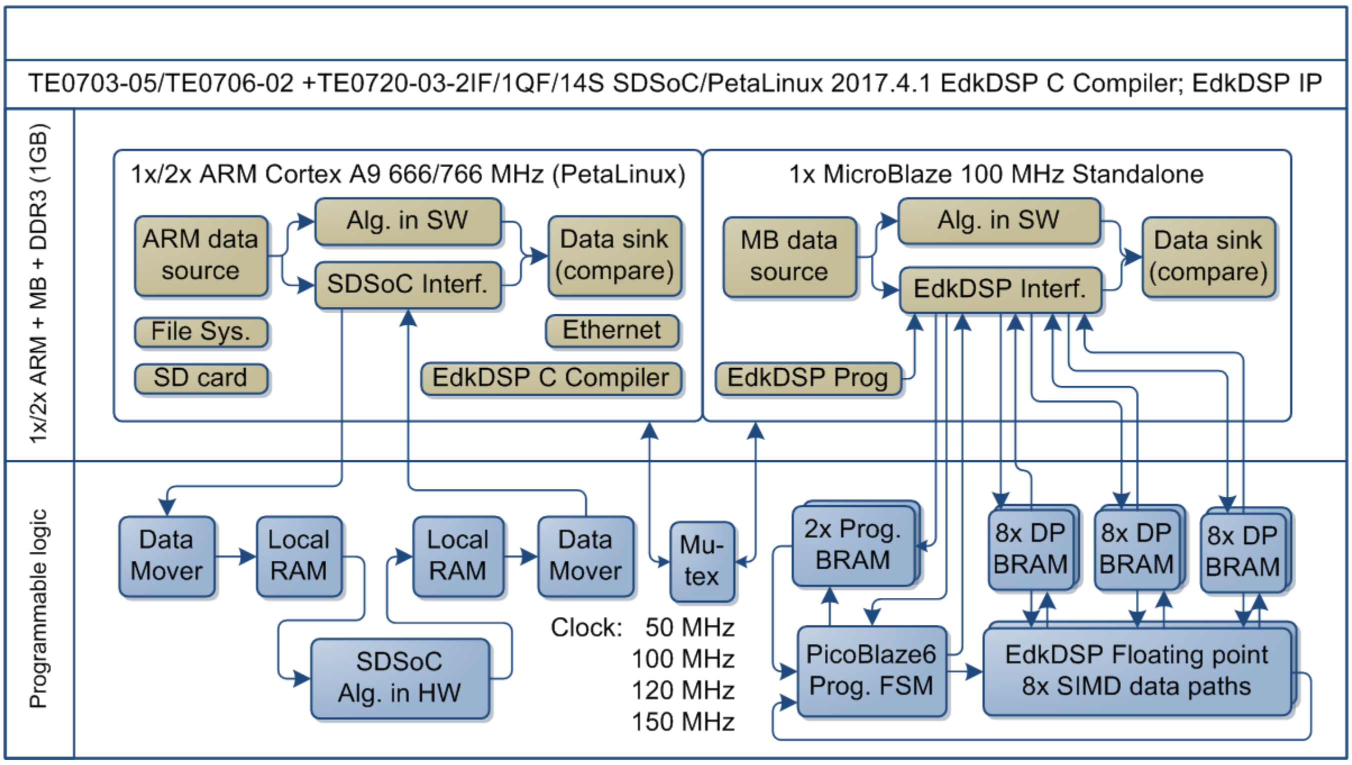


Figure 2: SDSoC compatible Zynq system with 8xSIMD EdkDSP floating point accelerator.

Evaluation system parameters

The evaluation system supports two Trenz Electronic carrier boards (**TE0703-05** and **TE0706-02**) [3] and three types of Trenz Electronic Zynq modules [1]:

- **TE0720-03-2IF** is an industrial grade (Tj = -40°C to +100°C) module, speed 2 with dual core Arm A9. The dual core Arm A9 and the PL part are **faster** in comparison to the other two modules.
- **TE0720-03-1QF** is an automotive grade (Tj = -40°C to +125°C) module, speed 1 with dual core Arm A9. This module can be used in applications requiring **wide temperature range**. Module is more expensive.
- **TE0720-03-14S-1C** is a commercial grade (Tj = 0°C to +85°C) module, speed 1 with **single core** Arm Cortex A9 and reduced programmable logic (PL) size. This is **low cost module** suitable for cost sensitive applications.
- **TE0720-03-1CFA-S** is a commercial grade (Tj = 0°C to +85°C) module, speed 1 with **dual core** Arm Cortex A9. This is assembled **starter kit** with the **TE0720-03-1CFA** module, heat sink, TE0703-05 carrier board, USB cable, SD card and the 5V/4A power supply.

Main parameters of these modules are summarised in Table 1.

Table 1: Parameters of supported Zynq modules.

Module	Xilinx Zynq device	ARM A9	A9 clock	Slices	LUTs	REGs	BRAMs	DSPs
TE0720-03-2IF	XC7Z020-2CLG484I	2x	766MHz	13300	53200	106400	140	220
TE0720-03-1QF	XA7Z020-1CLG484Q	2x	666MHz	13300	53200	106400	140	220
TE0720-03-14S-1C	XC7Z014S-1CLG484C	1x	666MHz	13300	40600	81200	107	170
TE0720-03-1CFA	XC7Z020-1CLG484C	2x	666MHz	13300	53200	106400	140	220

The PL part of the 28nm Zynq device contains:

- The run-time reprogrammable 8xSIMD EdkDSP floating point IP Core. It is using 120 MHz clock in case of the faster TE0720-03-2IF module and 100 MHz clock in case of the other two modules.
- MicroBlaze 32 bit soft core processor operating at 100 MHz.
- One of HW accelerators generated in Xilinx SDSoC 2017.4.1 from C/C++ reference SW ARM A9 function and operating with 150 MHz, 120 MHz, 100 MHz or 50 MHz clock.

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 2017.4.1 operating system. The 8xSIMD EdkDSP IP is controlled by the 32bit MicroBlaze processor.

The 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 32 bit 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 2.

- 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 2: (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).

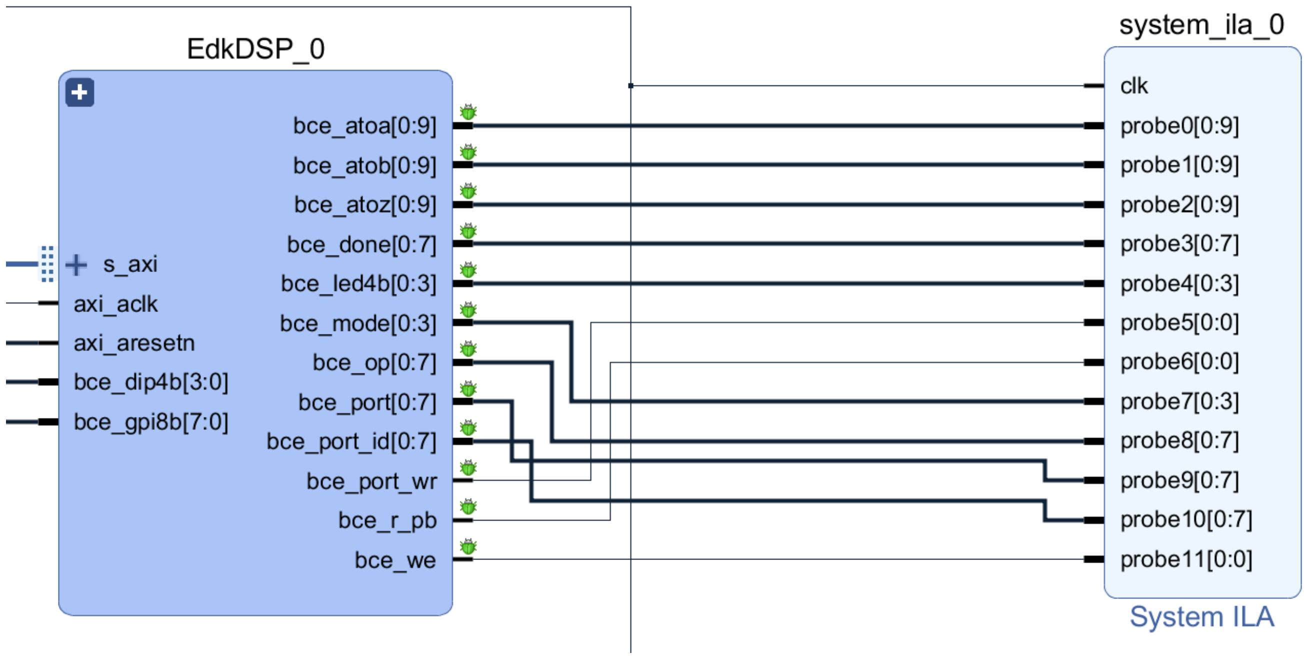


Figure 3: 8xSIMD EdkDSP floating point accelerator IP core with System ILA.

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 (System ILA) debug probes can capture 8192 data samples in case of TE0720-03-2IF and TE0720-03-1QF module and 2048 data samples in case of TE0720-03-14S-1C module. System ILA provides 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.

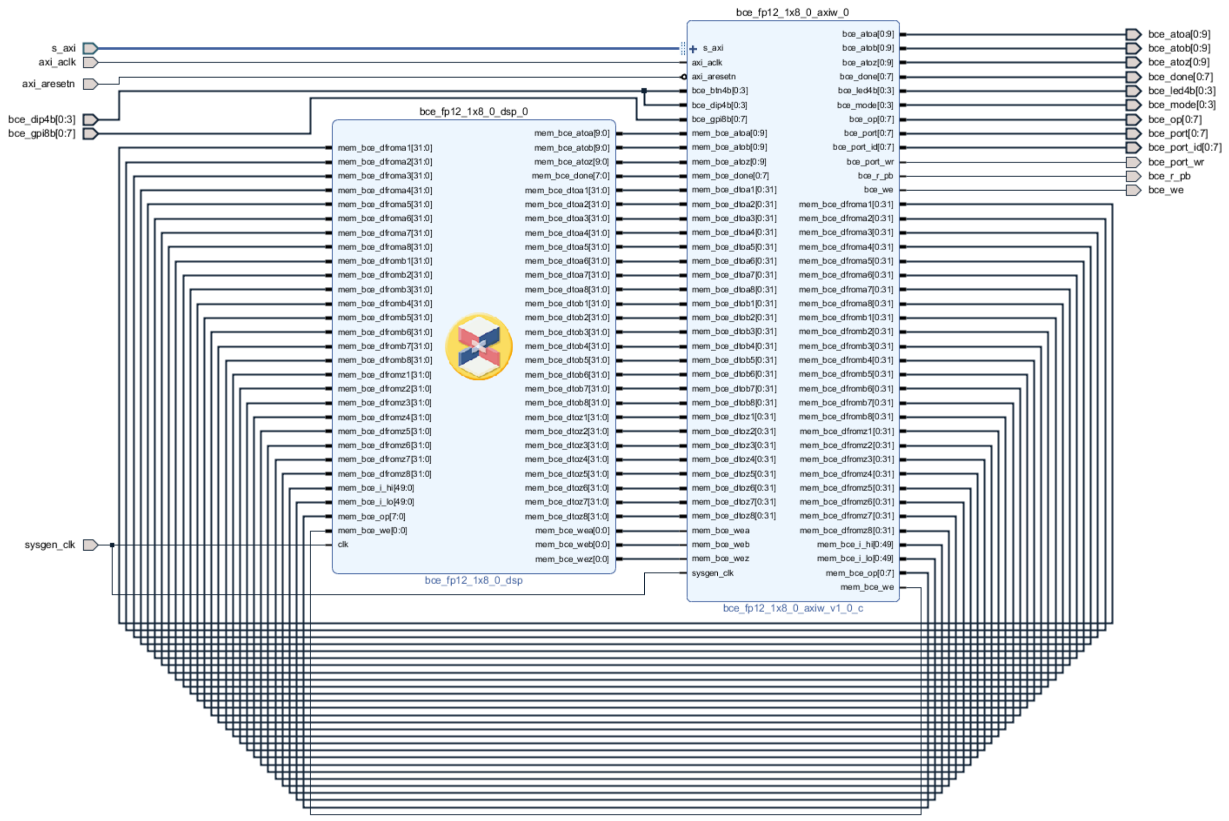


Figure 4: Internal details of (8xSIMD) EdkDSP floating point accelerator IP core.

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 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 2017.4.1 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 2017.4.1. 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 2.

- 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 3.
- The 8xSIMD data flow unit executes 8xSIMD floating point operations defined in Table 2.
- 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 3.

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 3.

Table 3: 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 4*. 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 4: 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 3*
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 5*.
- WAL error codes are defined in *Table 6*.
- 8xSIMD EdkDSP is supported by API functions collected in the WAL API are listed and described in *Table 7*.

Table 5: 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 6: 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 7: 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_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_get_capabilities(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_get_license(struct wal_worker *wrk, int pbid, unsigned int *outval);

All worker abstraction layer API functions listed in *Table 7* 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 7* 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 in *Figure 1* and *Figure 5*.

The dual core ARM Cortex A9 system runs configured PetaLinux 2017.4.1 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 2017.4.1 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 8: 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 components [1], [2], [3], [4], [5] designed and manufactured by company Trenz Electronic:

TE0720-03-2IF; Part: XC7Z020-2CLG484I; 1 GByte DDR; Industrial Grade (Tj = -40°C to +100°C) [1].
TE0720-03-1QF; Part: XA7Z020-1CLG484Q; 1 GByte DDR; Automotive Grade (Tj = -40°C to +125°C) [1].
TE0720-03-214S-1C; Part: XC7Z014S-1CLG484C; 1 GByte DDR; Industrial Grade (Tj = 0°C to +85°C) [1].
Heatsink for TE0720, spring-loaded embedded [2]. The heatsink serves for the passive cooling of Zynq module.
TE0706-02 Carrier Board from Trenz Electronic [3]. Board targets extension with second Ethernet in the Zynq PL.
TE0703-05 Carrier Board from Trenz Electronic [3]. Board targets wide I/O with pre-processing in a Lattice FPGA.
Pmod USBUART Serial converter & interface [4]. Serves for output from MicroBlaze to PC console via PC USB.
TE0790-02 XMOD FTDI JTAG Adapter - Xilinx compatible [5]. Supports console and Jtag in case of TE0706-02.

The technical reference manuals (TRM) of the TE0720-03-2IF, TE0720-03-1QF and TE0720-03-214S-1C modules can be downloaded from [1] and TRM for carrier board TE0706-02 or TE0703-05 can be downloaded from [3].

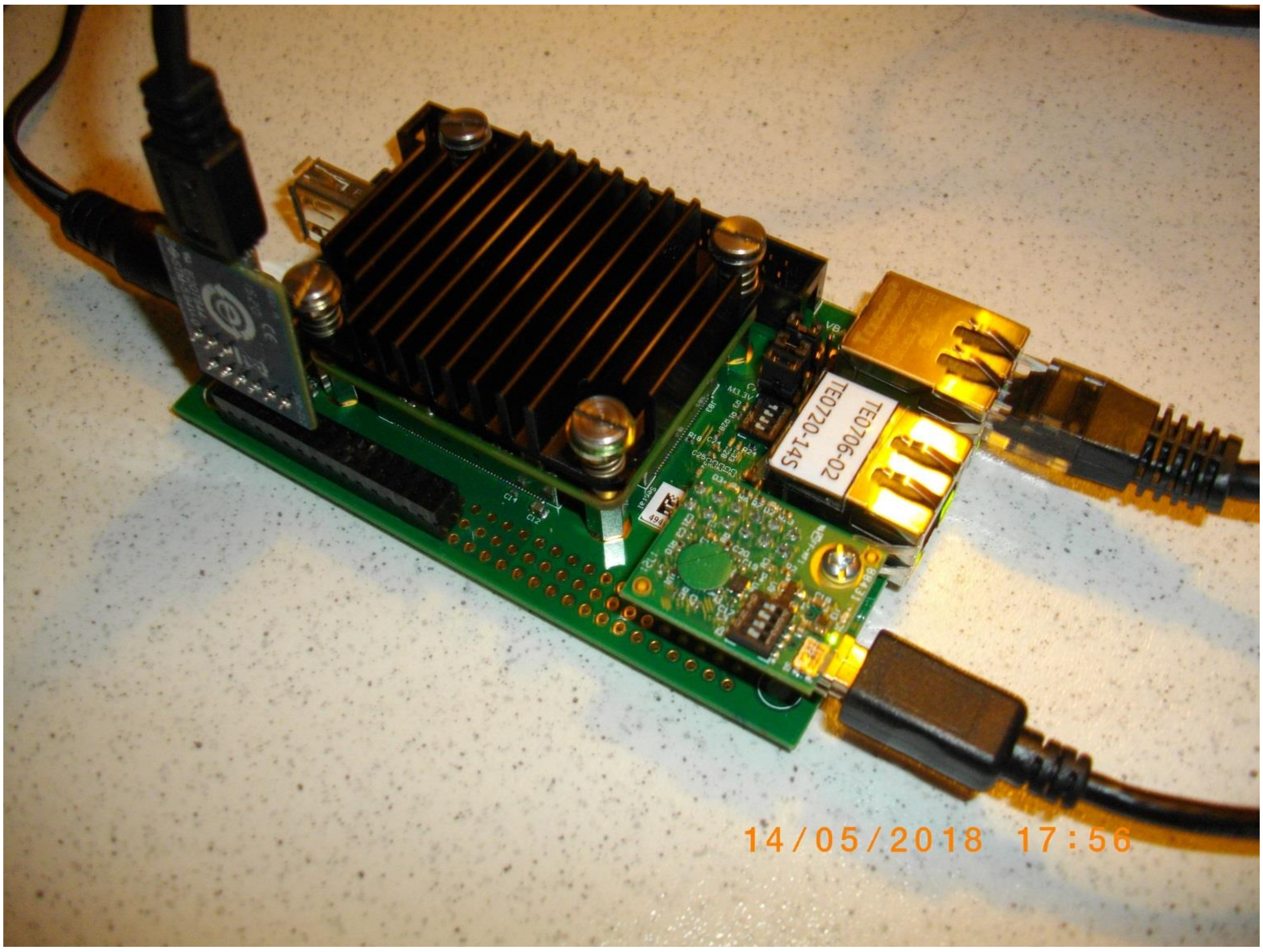


Figure 5: TE0706-02; TE0720-03-14S-1C; USBUART and XMOD FTDI JTAG adapter

Configuration of switches and jumpers on carrier boards TE0703-05 and TE0706-02

Configuration of the TE0703-05 board (and the TE0720-03-1CFA-S starter kit with the TE0703-05 carrier)

- Set jumpers of the **TE0703-05** board to **VCCIOA=3.3V; VCCIAOB=1.8V; VCCIOC=3.3V; VCCIOD=3.3V** by: **J5: connect 2-3; J8: connect 1-2; J9: connect 2-3; J10: connect 2-3**
- Set switch **S1** of the **TE0706-02** board to: **1=OFF; 2=ON; 3=ON; 4=ON**

Configuration of TE0706-02 board

- Set jumpers of the **TE0706-02** board to generate **VCCIOA=3.3V; VCCIOC=3.3V; VCCIOD=3.3V** by **J10: connect 2-3; J11: connect 2-3; J12: connect 2-3**
  In case of the TE0706-02 board the **VCCIAOB=1.8V** is set directly on the PCB (no dedicated jumper).
- Set switch S1 of the **TE0706-02** board to: **1=ON; 2=ON; 3=ON; 4=OFF**

Configuration of TE0790-02 xmod adapter

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 jumper on the USBUART pmod is set to the default: connect **`lcl-vcc`**. With this setup, the USBUART pmod convertor chip is powered from the PC 5V USB source. The TE0790-02 xmod adapter generates its local 3.3V power supply by an on-module DC2DC power converter. See *Figure 1* and *Figure 5*.

Configuration of USBUART pmod adapter

The serial terminal for MicroBlaze is connected to the PC by a Micro USB cable via the USBUART pmod adapter. The J6 connector on the TE0706-02 and J2 connector on the TE0703-05 have three lines of 32 pins named:

```
[A1  A2  A3  A4  A5  A6  …  A32]
[B1  B2  B3  B4  B5  B6  …  B32]
[C1  C2  C3  C4  C5  C6  …  C32]
```

In case of the TE0706-02 board, the USBUART pmod is connected to pins [B1 … B6] of connector J6B (central line B). See Table 9, *Figure 6* and the concrete implemented solution on *Figure 5*:

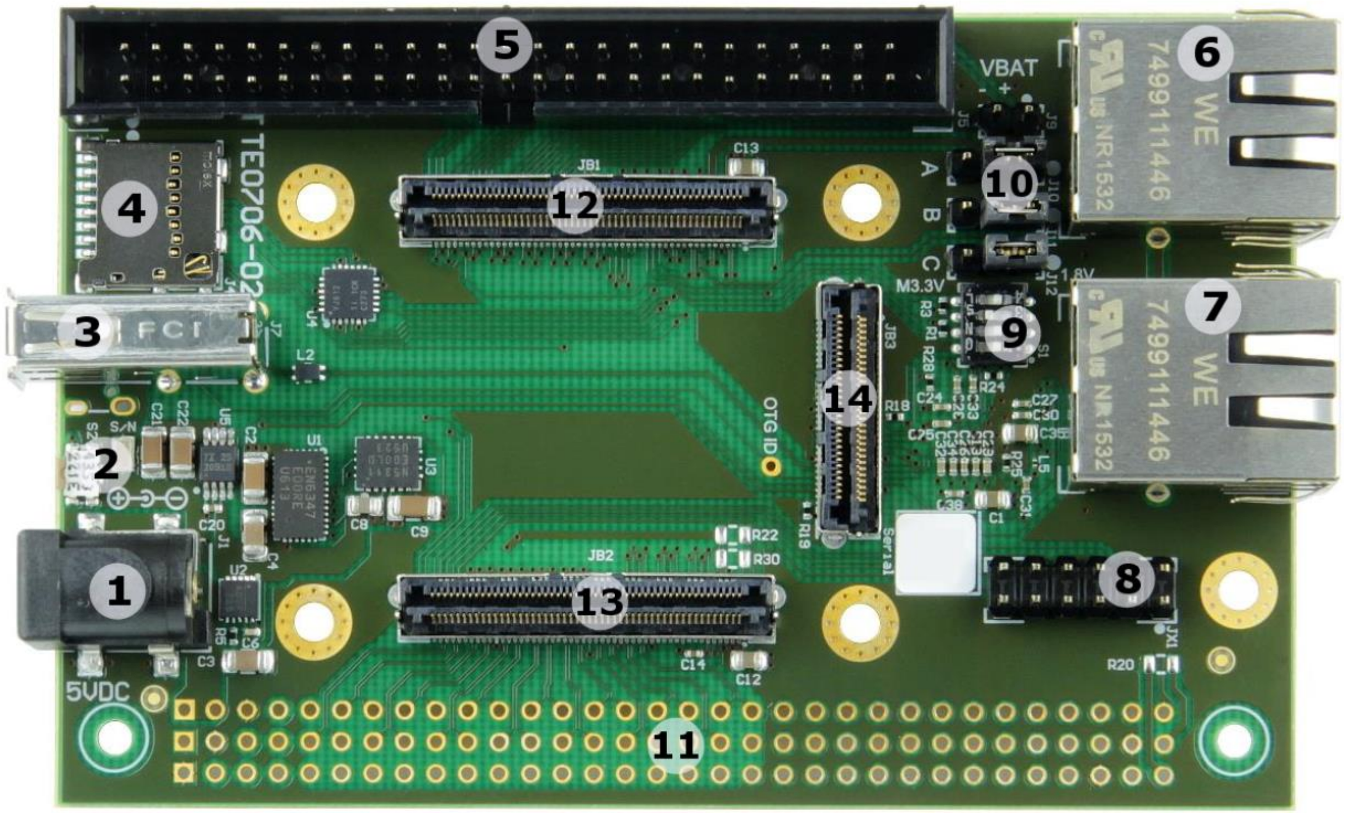
Table 9: Connection of USBUART to TE0706-02

TE0706-02	USBUART	Name	Function
J6 pin B1	J2 pin 6	3.3V	Disconnected by USBUART jumper. Power for USBUART from PC USB 5V
J6 pin B2	J2 pin 5	GND	Ground
J6 pin B4	J2 pin 3	TXD	FPGA Pin: AB2; FPGA design net: uart_pmod_tx; Direction: from PC to FPGA
J6 pin B5	J2 pin 2	RXD	FPGA pin: U5; FPGA design net: uart_pmod_rx; Direction: from FPGA to PC

In case of the TE0703-05, the USBUART pmod can be also to pins [B1 … B6] of the connector J2 if the communication from PC to MicroBlaze is not needed. If needed, use a custom cable. See Table 10 and *Figure 7*.

Table 10: Connection of USBUART to TE0703-05

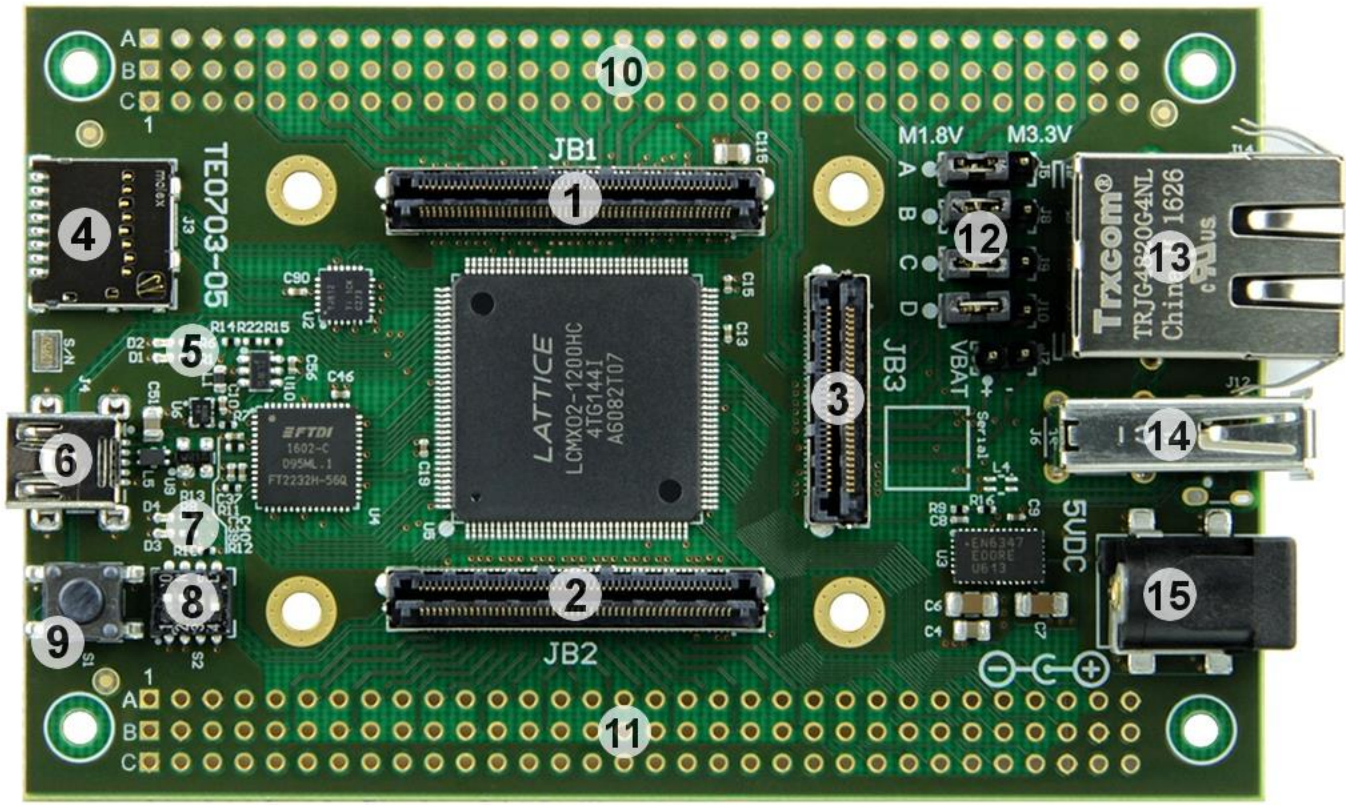
TE0703-05	USBUART	Name	Function
J2 pin B1	J2 pin 6	3.3V	Disconnected by USBUART jumper. Power for USBUART from PC USB 5V
J2 pin B2	J2 pin 5	GND	Ground
J2 pin **C3**	J2 pin 3	TXD	FPGA Pin: AB2; FPGA design net: uart_pmod_tx; Direction: from PC to FPGA
J2 pin B5	J2 pin 2	RXD	FPGA pin: U5; FPGA design net: uart_pmod_rx; Direction: from FPGA to PC



1. 5V power connector jack, J1
2. Reset switch, S2
3. USB2.0 type A receptacle, J7
4. Micro SD card socket with Card Detect, J4
5. 50 pin IDC male connector, J5
6. 1000Base-T Gigabit RJ45 Ethernet MagJack, J3
7. 1000Base-T Gigabit RJ45 Ethernet MagJack, J2
8. XMOD JTAG- / UART-header, JX1
9. User DIP-switch, S1
10. VCCIO selection jumper block, J10 - J12
11. External connector (VG96) placeholder, J6
12. Samtec Razor Beam™ LSHM-150 B2B connector, JB1
13. Samtec Razor Beam™ LSHM-150 B2B connector, JB2
14. Samtec Razor Beam™ LSHM-130 B2B connector, JB3

Figure 6: TE0706-02 Carrier Board.

Figure 6 presents main components and connector locations of the TE0706-02 Carrier Board [3]. The evaluation package released together with this application note supports single 1000Base-T Gigabit RJ45 Ethernet MagJack, J3 as Arm A9 PetaLinux eth0. See *Figure 6*. Output path from MicroBlaze to PC and input path from the PC keyboard to MicroBlaze is supported by USBUART connected directly to the connector **J6: B1…B6** pins. See https://wiki.trenz-electronic.de/display/PD/TE0706+TRM for source of the photo and for detailed description of the **TE0706-02** carrier board.



1. Samtec Razor Beam™ LSHM-150 B2B connector, JB1
2. Samtec Razor Beam™ LSHM-150 B2B connector, JB2
3. Samtec Razor Beam™ LSHM-130 B2B connector, JB3
4. Micro SD card socket with detect switch, J3
5. LED indicators D1 and D2
6. Mini-USB type B connector, J4
7. LED indicators D3 and D4
8. Configuration DIP switches, S2
9. User push button (Reset), S1
10. External connector (VG96) placeholder, J1
11. External connector (VG96) placeholder, J2
12. VCCIO voltage selection jumper block, J5, J8, J9 and J10
13. Trxcom 1000Base-T Gigabit RJ45 Magjack, J14
14. USB type A receptacle, J6 (optional micro USB 2.0 type B receptacle available, J12)
15. 5V power connector jack, J13

Figure 7: TE0703-05 Carrier Board.

Figure 7 presents main components and connector locations of the TE0703-05 Carrier Board [3]. The precompiled designs can be used without modification on the TE0703-05. Output path from MicroBlaze to PC is supported if the USBUART is connected to the **J2: B1…B6** pins directly. Output path from MicroBlaze to PC and input path from the PC keyboard to MicroBlaze is supported only if the USBUART is connected to the **J2: B1 B2 C3 B5** pins indirectly (via a custom made cable). See https://wiki.trenz-electronic.de/display/PD/TE0703+TRM for source of the photo and for description of the **TE0703-05** carrier board.

7. Reference Application for the 8xSIMD EdkDSP IP Core

The reference application 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 75 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 75 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 75 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 11.*

Table 11: Requirements and results.

Parameter (Module TE0720-03-2IF)	**Requirement**	**SW MicroBlaze 100 MHz** Requirements met (YES/NO)	**8xSIMD EdkDSP 120 MHz** Requirements met (YES/NO)
FIR filter sampling rate Order N=2000	75 kHz	2.25 kHz (NO)	279.70 kHz (YES)
FIR sustained performance (MFLOPs)	300 MFLOPs	9 MFLOPs (NO)	1119 MFLOPs (YES)
LMS filter sampling rate Order N=2000	75 kHz	1.125 kHz (NO)	90.75 KHz (YES)
LMS sustained performance (MFLOPs)	600 MFLOPs	9 MFLOPs (NO)	728 MFLOPs (YES)
Parameter (Modules TE0720-03-1QF, TE0720-03-14S-1C)	**Requirement**	**SW MicroBlaze 100 MHz** Requirements met (YES/NO)	**8xSIMD EdkDSP 100 MHz** Requirements met (YES/NO)
FIR filter sampling rate Order N=2000	75 kHz	2.25 kHz (NO)	244.4 kHz (YES)
FIR sustained performance (MFLOPs)	300 MFLOPs	9 MFLOPs (NO)	978 MFLOPs (YES)
LMS filter sampling rate Order N=2000	75 kHz	1.125 kHz (NO)	77.03 KHz (YES)
LMS sustained performance (MFLOPs)	600 MFLOPs	9 MFLOPs (NO)	618 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 MicroBlaze	Required	YES	YES
Runtime change of 8xSIMD EdkDSP IP	Required	NA	YES
Embedded 8xSIMD EdkDSP C compiler	Required	NA	YES
Compatibility with SDSoC 2017.4.1	Required	YES	YES
Compatibility with PetaLinux 2017.4.1	Required	YES	YES
Compatibility with free SDK 2017.4.1 and free edition of Vivado HLS 2017.4.1	Required	YES	YES

Summary of main results related to the performance of the 8xSIMD EdkDSP IP:

- The required LMS filter sampling rate 75 KHz (with N=2000) is reached for the TE0720-03-2IF module.
- The maximum sampling rate is 90.75 kHz for the adaptive LMS filter and 279.7 kHz for the FIR filter on the TE0720-03-2IF module with the 120 MHz 8xSIMD EdkDSP.
- The sustained floating-point performance of the 120 MHz 8xSIMD EdkDSP on TE0720-03-2IF module is 728 MFLOPs in case of the adaptive LMS filter and 1119 MFLOPs in case of the FIR filter.
- The maximum sampling rate is 77.03 kHz for the adaptive LMS filter and 244.4 kHz for the FIR filter on the on TE0720-03-1QF or TE0720-03-14S-1C module with the 100 MHz 8xSIMD EdkDSP.
- The sustained floating-point performance of the 100 MHz 8xSIMD EdkDSP on TE0720-03-1QF or TE0720-03-14S-1C module is 618 MFLOPs in case of the adaptive LMS filter and 978 MFLOPs in case of the FIR filter.
- The 8xSIMD EdkDSP is controlled from the 100 MHz MicroBlaze processor and operates in parallel to the Cortex A9 processor(s).
- 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 2017.4.1 design environment.
- The embedded C/ASM compiler utilities for the 8xSIMD EdkDSP accelerator run as Linux applications on the 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 100 MHz (1qf and 14s device) and 120 MHz (2if device). See *Figure 9*, *Figure 10*, *Figure 11*, *Figure 12*.
- The standalone examples support ILA and additionally can display the on-chip temperature via JTAG. See *Figure 13*
- 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 14 Figure 15*, *Figure 16*.
- There is no need to install Xilinx SDK 2017.4.1, Xilinx Vivado 2017.4.1 tools or Xilinx SDSoC 2017.4.1.
- The In-circuit Logic Analyser (ILA) JTAG based inspection/observation/debug can be performed from the free Xilinx Lab Vivado 2017.4.1 tool installed on Win7 (64bit) or Win 10 (64bit) PC
- The Linux target examples support 1GBit Ethernet, SSH telnet and file system management tools like the Total Commander for an ftp based access from PC to the SD card files.

The base evaluation package provides 21 demos for the Linux target and the 19 precompiled demos for the standalone target. *Table 12* describes demos, PL resources and the HW/SW SDSoC 2017.4.1. acceleration data.

Table 12: Description of ARM SDSoC acceleration examples compatible with 8xSIMD EdkDSP IP

Linux HW/SW Acceleration	**Standalone HW/SW Acceleration**	**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.**
te01_l	te01_s	**array_partition** - This example shows how to use array partitioning to improve performance of a hardware function. It performs int32 matrix multiplication C[32,32] = A[32,32] * B[32,32]
2if: 3.39x	2if: 6.62x	150 MHz Slices: 63.20% Luts: 44.36% Registers: 23.69% BRAMs: 76.79% DSPs: 54.55%
1qf: 4.40x	1qf: 7.29x	150 MHz Slices: 65.14% Luts: 43.27% Registers: 26.00% BRAMs: 76.79% DSPs: 54.55%
14s: 4.46x	14s: 7.17x	150 MHz Slices: 63.71% Luts: 56.60% Registers: 34.05% BRAMs: 89.25% DSPs: 70.59%
1cfa:	1cfa:	150 MHz Slices: 65.06% Luts: 44.34% Registers: 24.66% BRAMs: 76.79% DSPs: 54.55%
te02_l	te02_s	**burst_rw** - This is simple example of using AXI4-master interface for burst read and write.
2if:	2if:	150 MHz Slices: 56.80% Luts: 38.86% Registers: 21.14% BRAMs: 51.43% DSPs: 9.55%
1qf:	1qf:	150 MHz Slices: 55.72% Luts: 38.89% Registers: 21.14% BRAMs: 51.43% DSPs: 9.55%
14s:	14s:	150 MHz Slices: 54.65% Luts: 50.87% Registers: 27.67% BRAMs: 56.07% DSPs: 12.35%
1cfa:	1cfa:	150 MHz Slices: 56.06% Luts: 38.86% Registers: 21.14% BRAMs: 51.43% DSPs: 9.55%
te03_l	te03_s	**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.
2if: 22.48x	2if: 25.16x	150 MHz Slices: 60.69% Luts: 42.18% Registers: 22.93% BRAMs: 51.43% DSPs: 10.91%
1qf: 25.43x	1qf: 28.94x	150 MHz Slices: 59.81% Luts: 42.21% Registers: 23.07% BRAMs: 51.43% DSPs: 10.91%
14s: 25.88x	14s: 28.88x	150 MHz Slices: 59.32% Luts: 55.23% Registers: 30.07% BRAMs: 56.07% DSPs: 14.12%
1cfa:	1cfa:	150 MHz Slices: 60.29% Luts: 42.22% Registers: 22.97% BRAMs: 51.43% DSPs: 10.91%
te04_l	te04_s	**data_access_random** - This is a simple example of int32 matrix multiplication (Row x Col) C[32,32]= A[32,32]*B[32,32] to demonstrate random data access pattern.
2if: 0.57x	2if: 0.57x	120 MHz Slices: 65.63% Luts: 43.55% Registers: 25.34% BRAMs: 56.43% DSPs: 13.64%
1qf: 0.63x	1qf: 0.63x	120 MHz Slices: 64.33% Luts: 43.58% Registers: 25.35% BRAMs: 56.43% DSPs: 13.64%
14s: 0.63x	14s: 0.63x	120 MHz Slices: 64.60% Luts: 57.02% Registers: 33.19% BRAMs: 62.62% DSPs: 17.65%
1cfa:	1cfa:	120 MHz Slices: 65.34% Luts: 43.57% Registers: 25.35% BRAMs: 56.43% DSPs: 13.64%
te05_l	te05_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.
2if: 5.84x	2if: 6.51x	150 MHz Slices: 58.66% Luts: 40.36% Registers: 22.57% BRAMs: 55.00% DSPs: 22.27%
1qf: 6.42x	1qf: 7.16x	150 MHz Slices: 59.05% Luts: 40.30% Registers: 22.80% BRAMs: 55.00% DSPs: 22.27%
14s: 6.60x	14s: 7.22x	150 MHz Slices: 58.81% Luts: 52.72% Registers: 29.85% BRAMs: 60.75% DSPs: 28.82%
1cfa:	1cfa:	150 MHz Slices: 60.74% Luts: 40.30% Registers: 22.80% BRAMs: 55.00% DSPs: 22.27%
te06_l	te06_s	**direct_connect** - This is a simple example of int32 matrix multiplication with matrix addition (Out[32,32] = (A[32,32] * B[32,32]) + C[32,32]) to demonstrate direct connection which helps to achieve increasing in system parallelism and concurrency.
2if: 8.61x	2if: 9.14x	150 MHz Slices: 75.00% Luts: 49.24% Registers: 29.73% BRAMs: 82.50% DSPs: 57.73%
1qf: 8.36x	1qf: 8.92x	120 MHz Slices: 72.65% Luts: 49.21% Registers: 29.73% BRAMs: 82.50% DSPs: 57.73%
14s: 9.55x	14s: 9.92x	150 MHz Slices: 74.31% Luts: 62.99% Registers: 40.41% BRAMs: 96.73% DSPs: 74.71%
1cfa:	1cfa:	120 MHz Slices: 73.46% Luts: 49.20% Registers: 29.73% BRAMs: 82.50% DSPs: 57.73%
te07_l	te07_s	**dma_sg** - This example demonstrates how to use Scatter-Gather DMAs for data transfer to/from hardware accelerator.
2if:	2if:	150 MHz Slices: 73.83% Luts: 48.92% Registers: 29.41% BRAMs: 60.00% DSPs: 9.55%
1qf:	1qf:	120 MHz Slices: 72.84% Luts: 48.94% Registers: 29.41% BRAMs: 60.00% DSPs: 9.55%

14s:	14s:	150 MHz Slices: 74.14% Luts: 64.08% Registers: 38.59% BRAMs: 67.29% DSPs: 12.35%
1cfa:	1cfa:	120 MHz Slices: 74.60% Luts: 48.95% Registers: 29.41% BRAMs: 60.00% DSPs: 9.55%
te08_l	te08_s	**dma_simple** - This example demonstrates how to insert Simple DMAs for data transfer between user program and hardware accelerator.
2if:	2if:	150 MHz Slices: 63.16% Luts: 43.06% Registers: 24.72% BRAMs: 56.43% DSPs: 9.55%
1qf:	1qf:	150 MHz Slices: 64.74% Luts: 43.07% Registers: 24.78% BRAMs: 56.43% DSPs: 9.55%
14s:	14s:	150 MHz Slices: 62.58% Luts: 56.34% Registers: 32.44% BRAMs: 62.62% DSPs: 12.35%
1cfa:	1cfa:	120 MHz Slices: 64.49% Luts: 43.07% Registers: 24.72% BRAMs: 56.43% DSPs: 9.55%
te09_l (With Linux SD file R/W functions)	Not imple-mented as standalone	**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.
2if:	NA	120 MHz Slices: 75.92% Luts: 51.33% Registers: 30.58% BRAMs: 63.21% DSPs: 10.91%
1qf:	NA	120 MHz Slices: 75.98% Luts: 51.32% Registers: 30.66% BRAMs: 63.21% DSPs: 10.91%
14s:	NA	120 MHz Slices: 76.30% Luts: 67.15% Registers: 40.15% BRAMs: 71.50% DSPs: 12.62%
1cfa:	NA	120 MHz Slices: 76.29% Luts: 51.31% Registers: 30.66% BRAMs: 63.21% DSPs: 10.91%
te10_l (With Linux SD file R/W functions)	Not imple-mented as standalone	**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.
2if:	NA	120 MHz Slices: 99.50% Luts: 79.55% Registers: 51.34% BRAMs: 88.93% DSPs: 40.91%
1qf:	NA	50 MHz Slices: 99.35% Luts: 79.58% Registers: 46.96% BRAMs: 88.93% DSPs: 40.91%
14s:	NA	SW impl. Slices: 43.88% Luts: 40.25% Registers: 19.66% BRAMs: 50.00% DSPs: 12.35%
1cfa:	NA	100 MHz Slices: 98.62% Luts: 79.60% Registers: 50.90% BRAMs: 88.93% DSPs: 40.91%
te11_l	te11_s	**full_array_2d** - This is a simple example of accessing full data from 2D array.
2if:	2if:	150 MHz Slices: 60.20% Luts: 41.98% Registers: 23.00% BRAMs: 55.36% DSPs: 12.27%
1qf:	1qf:	150 MHz Slices: 59.52% Luts: 41.91% Registers: 23.09% BRAMs: 55.36% DSPs: 12.27%
14s:	14s:	150 MHz Slices: 59.86% Luts: 54.84% Registers: 30.23% BRAMs: 61.21% DSPs: 15.88%
1cfa:	1cfa:	150 MHz Slices: 59.91% Luts: 41.92% Registers: 23.09% BRAMs: 55.36% DSPs: 12.27%
te12_l	te12_s	**hello_vadd** - This is a basic hello world kind of example which demonstrates how to achieve vector addition using hardware function.
2if:	2if:	150 MHz Slices: 60.06% Luts: 41.46% Registers: 22.59% BRAMs: 53.21% DSPs: 9.55%
1qf:	1qf:	150 MHz Slices: 58.95% Luts: 41.48% Registers: 22.59% BRAMs: 53.21% DSPs: 9.55%
14s:	14s:	150 MHz Slices: 57.40% Luts: 54.29% Registers: 29.57% BRAMs: 58.41% DSPs: 12.35%
1cfa:	1cfa:	150 MHz Slices: 59.52% Luts: 41.50% Registers: 22.59% BRAMs: 53.21% DSPs: 9.55%
te13_l	te13_s	**lmem_2rw** - This is a simple example of vector addition to demonstrate how to utilize both ports of Local Memory.
2if:	2if:	150 MHz Slices: 61.30% Luts: 42.13% Registers: 23.02% BRAMs: 55.36% DSPs: 9.55%
1qf:	1qf:	150 MHz Slices: 61.26% Luts: 42.14% Registers: 23.02% BRAMs: 55.36% DSPs: 9.55%
14s:	14s:	150 MHz Slices: 59.48% Luts: 55.12% Registers: 30.13% BRAMs: 61.21% DSPs: 12.35%
1cfa:	1cfa:	150 MHz Slices: 62.19% Luts: 42.21% Registers: 23.02% BRAMs: 55.36% DSPs: 9.55%
te14_l	te14_s	**loop_fusion** - This example will demonstrate how to fuse two loops into one to improve the performance of a C/C++ hardware function.
2if:	2if:	150 MHz Slices: 61.41% Luts: 42.73% Registers: 23.57% BRAMs: 53.21% DSPs: 15.00%
1qf:	1qf:	150 MHz Slices: 60.62% Luts: 42.72% Registers: 23.79% BRAMs: 53.21% DSPs: 15.00%
14s:	14s:	150 MHz Slices: 60.64% Luts: 55.86% Registers: 31.14% BRAMs: 58.41% DSPs: 19.41%
1cfa:	1cfa:	150 MHz Slices: 62.74% Luts: 42.71% Registers: 23.79% BRAMs: 53.21% DSPs: 15.00%
te15_l	te15_s	**loop_perfect** - This nearest neighbor example is to demonstrate how to achieve better performance using perfect loop.

2if:	2if:	150 MHz Slices: 75.26% Luts: 53.49% Registers: 29.28% BRAMs: 53.21% DSPs: 15.45%
1qf:	1qf:	150 MHz Slices: 73.72% Luts: 53.43% Registers: 29.51% BRAMs: 53.21% DSPs: 15.45%
14s:	14s:	150 MHz Slices: 74.11% Luts: 69.93% Registers: 38.63% BRAMs: 58.41% DSPs: 20.00%
1cfa:	1cfa:	150 MHz Slices: 74.62% Luts: 53.42% Registers: 29.51% BRAMs: 53.21% DSPs: 15.45%
te16_l	te16_s	**loop_pipeline** - This example demonstrates how loop pipelining can be used to improve the performance of a hardware function.
2if:	2if:	150 MHz Slices: 60.06% Luts: 41.46% Registers: 22.59% BRAMs: 53.21% DSPs: 9.55%
1qf:	1qf:	150 MHz Slices: 58.95% Luts: 41.48% Registers: 22.59% BRAMs: 53.21% DSPs: 9.55%
14s:	14s:	150 MHz Slices: 57.40% Luts: 54.29% Registers: 29.57% BRAMs: 58.41% DSPs: 12.35%
1cfa:	1cfa:	150 MHz Slices: 59.52% Luts: 41.50% Registers: 22.59% BRAMs: 53.21% DSPs: 9.55%
te17_l	te17_s	**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. It performs int32 matrix multiplication C[32,32] = A[32,32] * B[32,32]
2if: 4.27x	2if: 7.12x	150 MHz Slices: 68.44% Luts: 45.64% Registers: 26.45% BRAMs: 76.79% DSPs: 56.36%
1qf: 4.66x	1qf: 7.72x	150 MHz Slices: 67.90% Luts: 44.64% Registers: 27.53% BRAMs: 76.79% DSPs: 56.36%
14s: 4.92x	14s: 7.85x	150 MHz Slices: 66.91% Luts: 58.41% Registers: 36.04% BRAMs: 89.25% DSPs: 72.94%
1cfa:	1cfa:	150 MHz Slices: 67.88% Luts: 44.65% Registers: 27.53% BRAMs: 76.79% DSPs: 56.36%
te18_l	te18_s	**shift_register** - This example demonstrates how to shift values in each clock cycle.
2if: 1.96x	2if: 4.19x	150 MHz Slices: 63.03% Luts: 42.68% Registers: 24.19% BRAMs: 53.21% DSPs: 24.55%
1qf: 2.02x	1qf: 4.54x	150 MHz Slices: 62.23% Luts: 42.40% Registers: 24.52% BRAMs: 53.21% DSPs: 24.55%
14s: 2.10x	14s: 4.52x	150 MHz Slices: 61.28% Luts: 55.41% Registers: 32.10% BRAMs: 58.41% DSPs: 31.76%
1cfa:	1cfa:	150 MHz Slices: 62.87% Luts: 42.41% Registers: 24.52% BRAMs: 53.21% DSPs: 24.55%
te19_l	te19_s	**sys_port** - This is a simple example which demonstrates sys_port usage.
2if:	2if:	120 MHz Slices: 83.92% Luts: 54.55% Registers: 34.77% BRAMs: 65.00% DSPs: 9.55%
1qf:	1qf:	120 MHz Slices: 80.92% Luts: 54.53% Registers: 34.77% BRAMs: 65.00% DSPs: 9.55%
14s:	14s:	120 MHz Slices: 81.75% Luts: 71.42% Registers: 45.53% BRAMs: 73.86% DSPs: 12.35%
1cfa:	1cfa:	120 MHz Slices: 84.68% Luts: 54.56% Registers: 34.77% BRAMs: 65.00% DSPs: 9.55%
te20_l	te20_s	**systolic_array** - Matrix multiplication implemented as systolic array.
2if: 0.066x	2if: 0.162x	150 MHz Slices: 68.55% Luts: 47.36% Registers: 26.67% BRAMs: 53.21% DSPs: 61.36%
1qf: 0.077x	1qf: 0.177x	150 MHz Slices: 66.75% Luts: 46.26% Registers: 27.82% BRAMs: 53.21% DSPs: 61.36%
14s: 0.068x	14s: 0.198x	150 MHz Slices: 67.15% Luts: 60.51% Registers: 36.42% BRAMs: 58.41% DSPs: 79.41%
1cfa:	1cfa:	150 MHz Slices: 70.08% Luts: 46.29% Registers: 27.82% BRAMs: 53.21% DSPs: 61.36%
te21_l	te21_s	**wide_memory_rw** - Wide memory read write 64 bit wide.
2if:	2if:	150 MHz Slices: 60.07% Luts: 39.74% Registers: 23.34% BRAMs: 55.36% DSPs: 9.55%
1qf:	1qf:	150 MHz Slices: 59.34% Luts: 39.77% Registers: 23.34% BRAMs: 55.36% DSPs: 9.55%
14s:	14s:	150 MHz Slices: 58.41% Luts: 52.05% Registers: 30.55% BRAMs: 61.21% DSPs: 12.35%
1cfa:	1cfa:	150 MHz Slices: 59.59% Luts: 39.78% Registers: 23.34% BRAMs: 55.36% DSPs: 9.55%

Installation and use of the Release 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_EdkDSP_2if_te706_ila8k_Release_INSTALL.zip`
to directory of your choice. We will use:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\`

(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:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\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 or TE0703-05 carrier board.

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

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

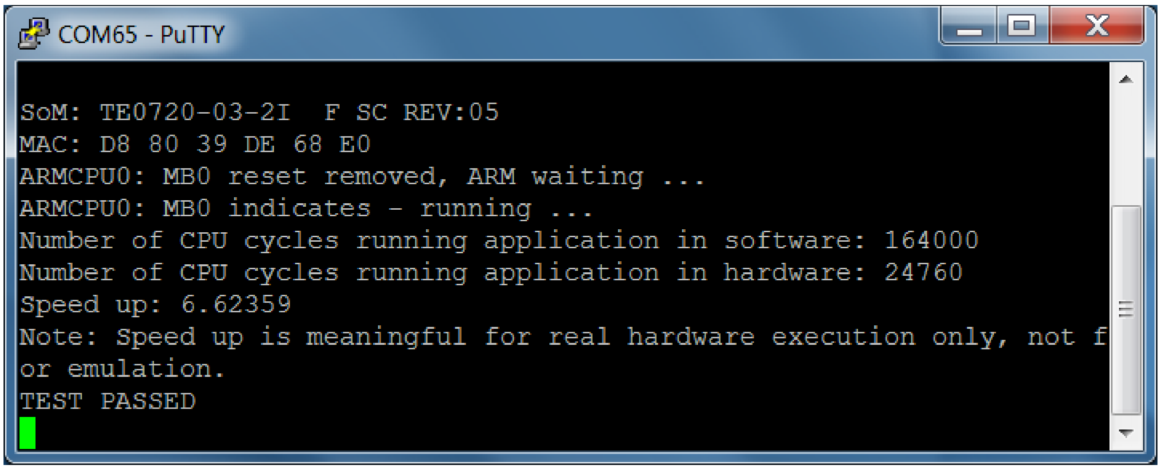
(8) In PC, open the Vivado Lab tool 2017.4.1 See *Figure 9*.

Open Hardware Manager
Press Auto Connect icon in Hardware window
- Open description of debug nets present in file, thus specifying the probes file as:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\debug_nets.ltx`
- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP.
See *Figure 10*, *Figure 11*, *Figure 12*.
- Open new perspective and observe the chip temperature. See *Figure 13*.

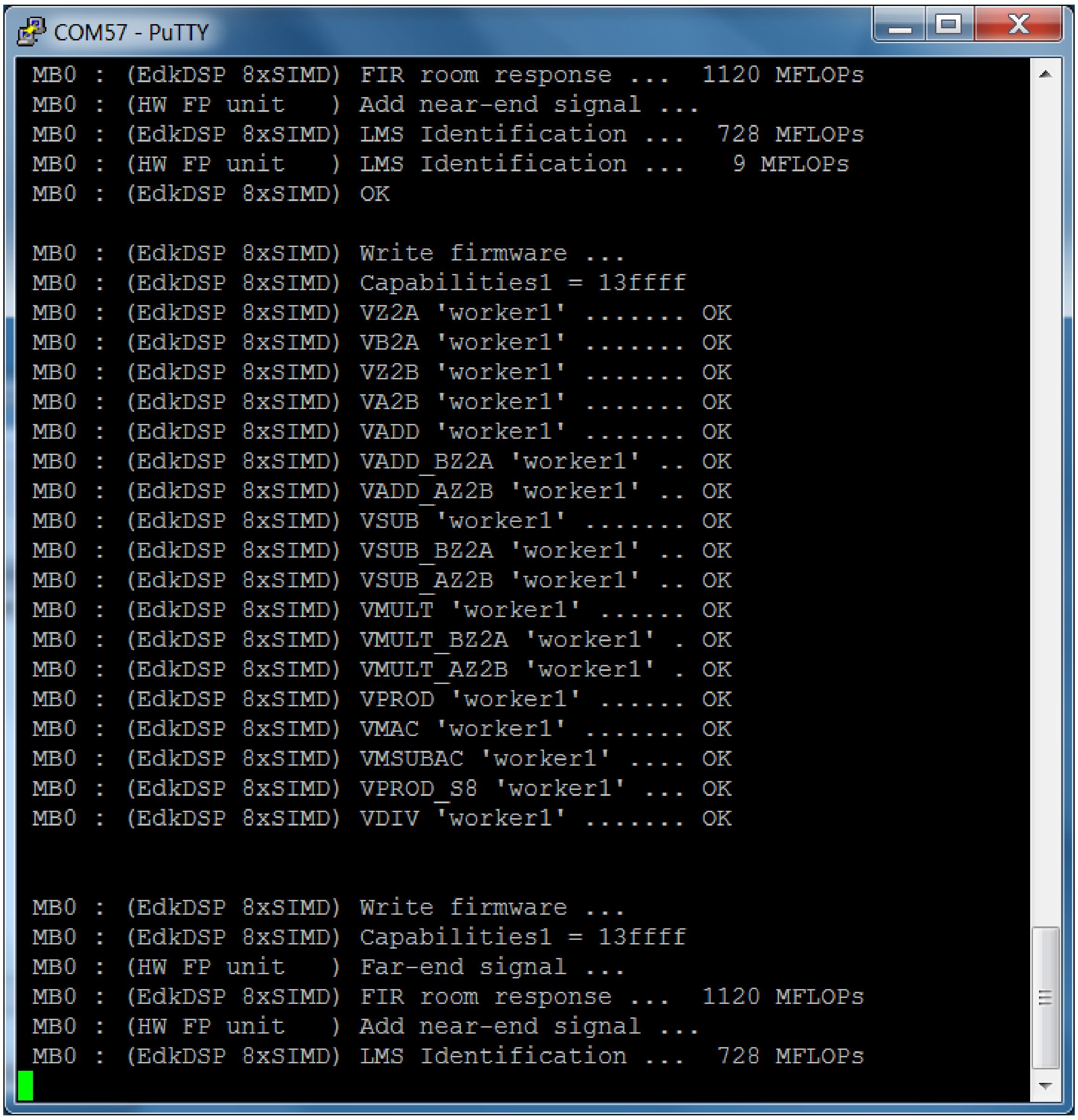
(9) Close Vivado Lab 2017.4.1 tool project.

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

(11) Go to step (6).

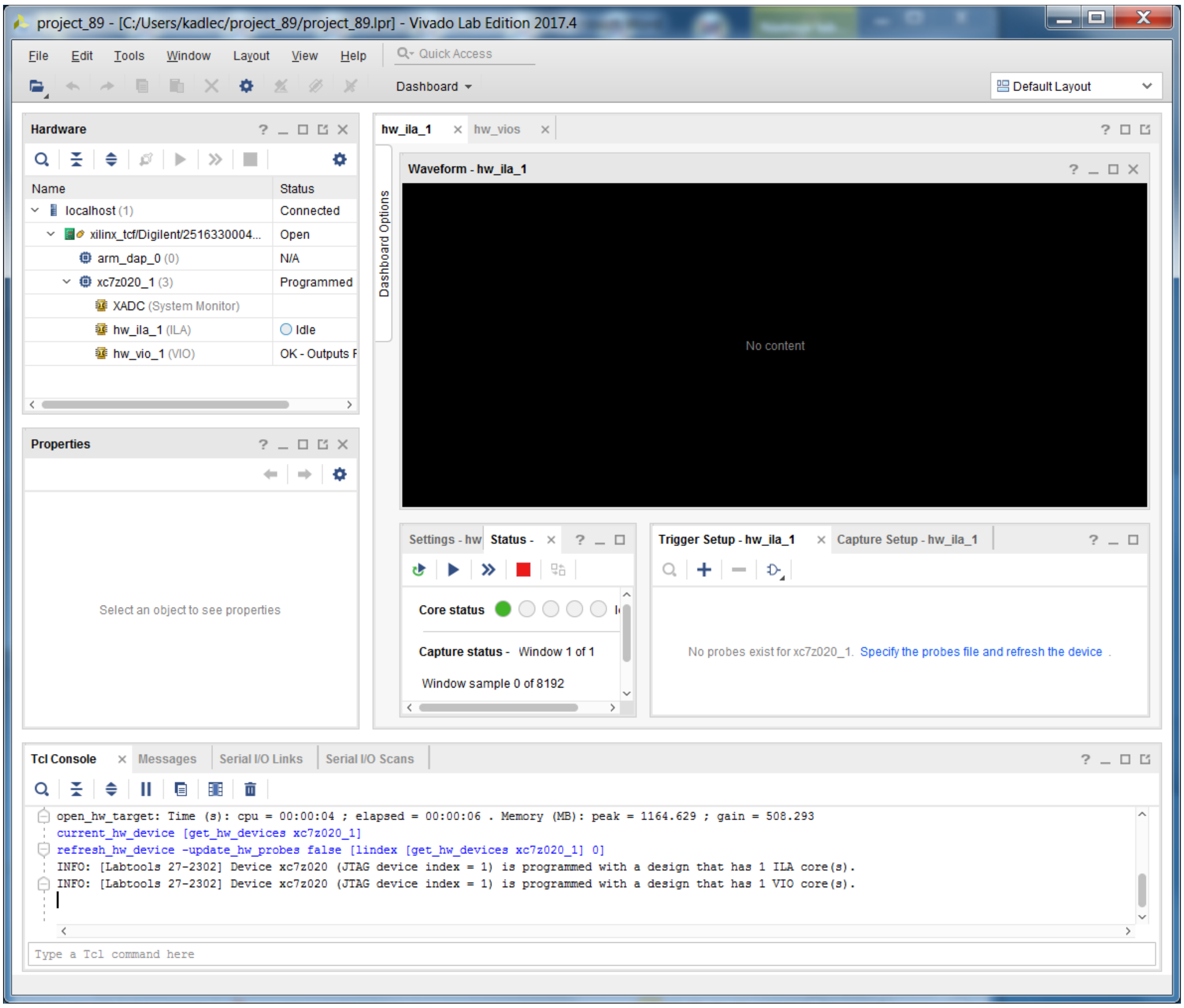


```
SoM: TE0720-03-2I   F SC REV:05
MAC: D8 80 39 DE 68 E0
ARMCPU0: MB0 reset removed, ARM waiting ...
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 164000
Number of CPU cycles running application in hardware: 24760
Speed up: 6.62359
Note: Speed up is meaningful for real hardware execution only, not f
or emulation.
TEST PASSED
```



```
 MB0 : (EdkDSP 8xSIMD) FIR room response ...  1120 MFLOPs
 MB0 : (HW FP unit   ) Add near-end signal ...
 MB0 : (EdkDSP 8xSIMD) LMS Identification ...  728 MFLOPs
 MB0 : (HW FP unit   ) LMS Identification ...    9 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 ...  1120 MFLOPs
 MB0 : (HW FP unit   ) Add near-end signal ...
 MB0 : (EdkDSP 8xSIMD) LMS Identification ...  728 MFLOPs
```

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



```
open_hw_target: Time (s): cpu = 00:00:04 ; elapsed = 00:00:06 . Memory (MB): peak = 1164.629 ; gain = 508.293
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 9: Release demo t01_s. Vivado Lab Tool is open.

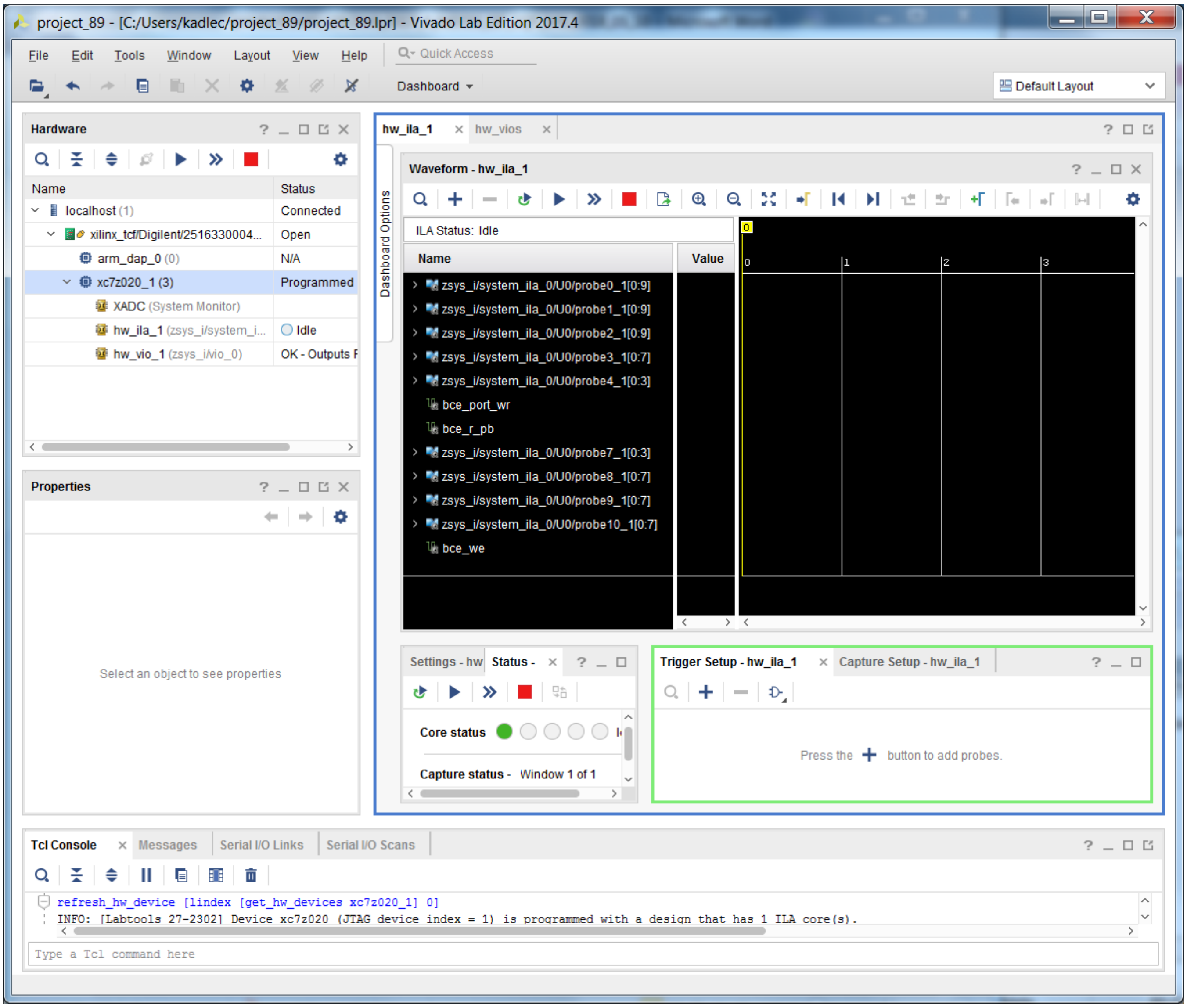
The Vivado Lab tool is connected to the chip. You have to specify the probes file (See *Figure 10*).

```
c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01
_s\Release\debug_nets.ltx
```

Names and parameters of probes are added to the ILA Waveform window. See *Figure 10*.

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/2516330004...	Open
arm_dap_0 (0)	N/A
xc7z020_1 (3)	Programmed
XADC (System Monitor)	
hw_ila_1 (zsys_i/system_i...	Idle
hw_vio_1 (zsys_i/vio_0)	OK - Outputs F

ILA Status: Idle

Name	Value
zsys_i/system_ila_0/U0/probe0_1[0:9]	
zsys_i/system_ila_0/U0/probe1_1[0:9]	
zsys_i/system_ila_0/U0/probe2_1[0:9]	
zsys_i/system_ila_0/U0/probe3_1[0:7]	
zsys_i/system_ila_0/U0/probe4_1[0:3]	
bce_port_wr	
bce_r_pb	
zsys_i/system_ila_0/U0/probe7_1[0:3]	
zsys_i/system_ila_0/U0/probe8_1[0:7]	
zsys_i/system_ila_0/U0/probe9_1[0:7]	
zsys_i/system_ila_0/U0/probe10_1[0:7]	
bce_we	

Core status ●
Capture status - Window 1 of 1

```
refresh_hw_device [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).
```

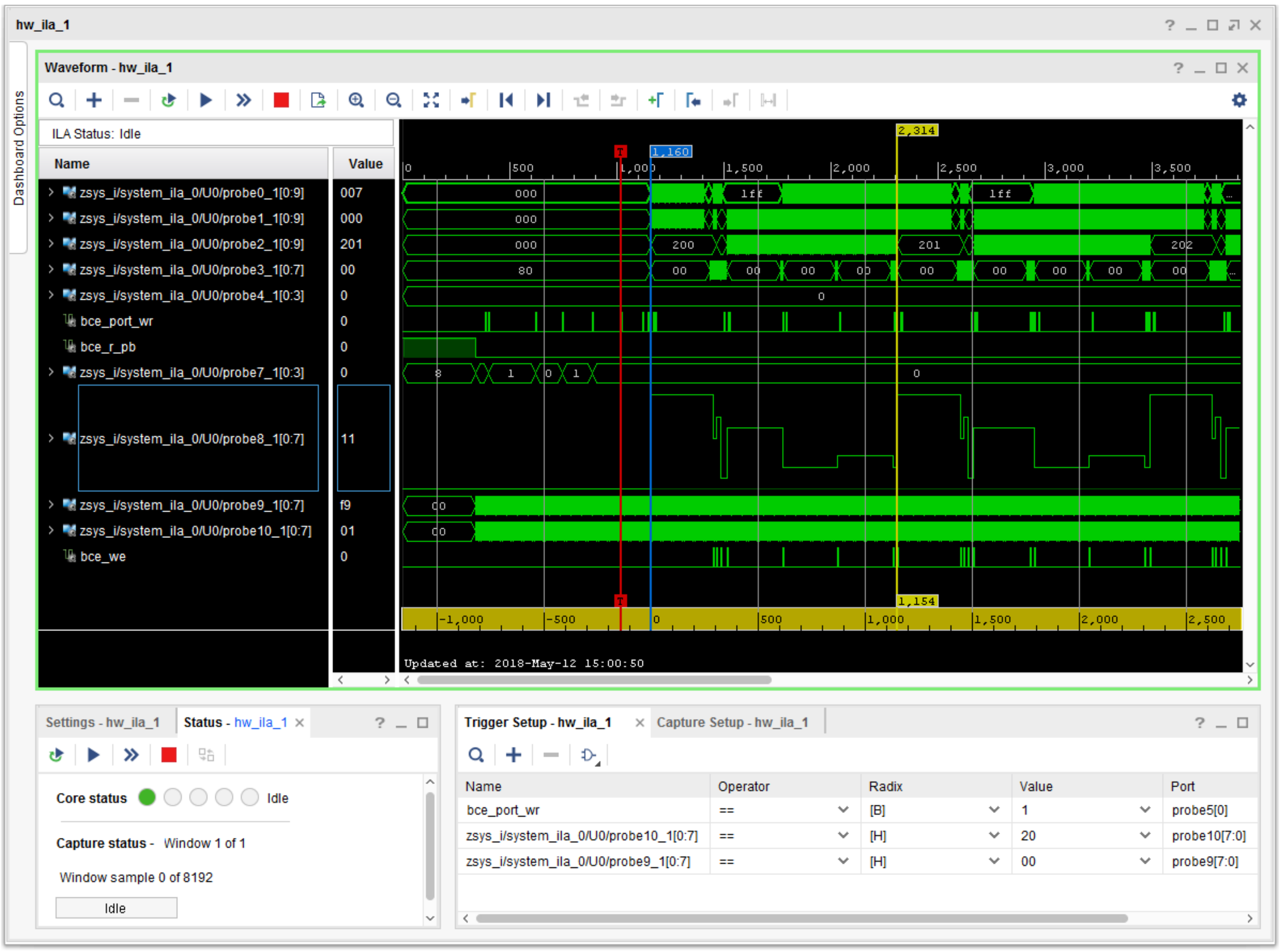
Type a Tcl command here

Figure 10: Release demo t01_s. Probes file is specified. Trigger conditions are set.

In Xilinx SDK 2017.4.1, open the EdkDSP C soure file:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SDK_Workspace\edkdsp\a\f2.c`

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);  // 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
zsys_i/system_ila_0/U0/probe0_1[0:9]	007
zsys_i/system_ila_0/U0/probe1_1[0:9]	000
zsys_i/system_ila_0/U0/probe2_1[0:9]	201
zsys_i/system_ila_0/U0/probe3_1[0:7]	00
zsys_i/system_ila_0/U0/probe4_1[0:3]	0
bce_port_wr	0
bce_r_pb	0
zsys_i/system_ila_0/U0/probe7_1[0:3]	0
zsys_i/system_ila_0/U0/probe8_1[0:7]	11
zsys_i/system_ila_0/U0/probe9_1[0:7]	f9
zsys_i/system_ila_0/U0/probe10_1[0:7]	01
bce_we	0

Core status ● Idle

Capture status - Window 1 of 1

Window sample 0 of 8192

Idle

Name	Operator	Radix	Value	Port
bce_port_wr	==	[B]	1	probe5[0]
zsys_i/system_ila_0/U0/probe10_1[0:7]	==	[H]	20	probe10[7:0]
zsys_i/system_ila_0/U0/probe9_1[0:7]	==	[H]	00	probe9[7:0]

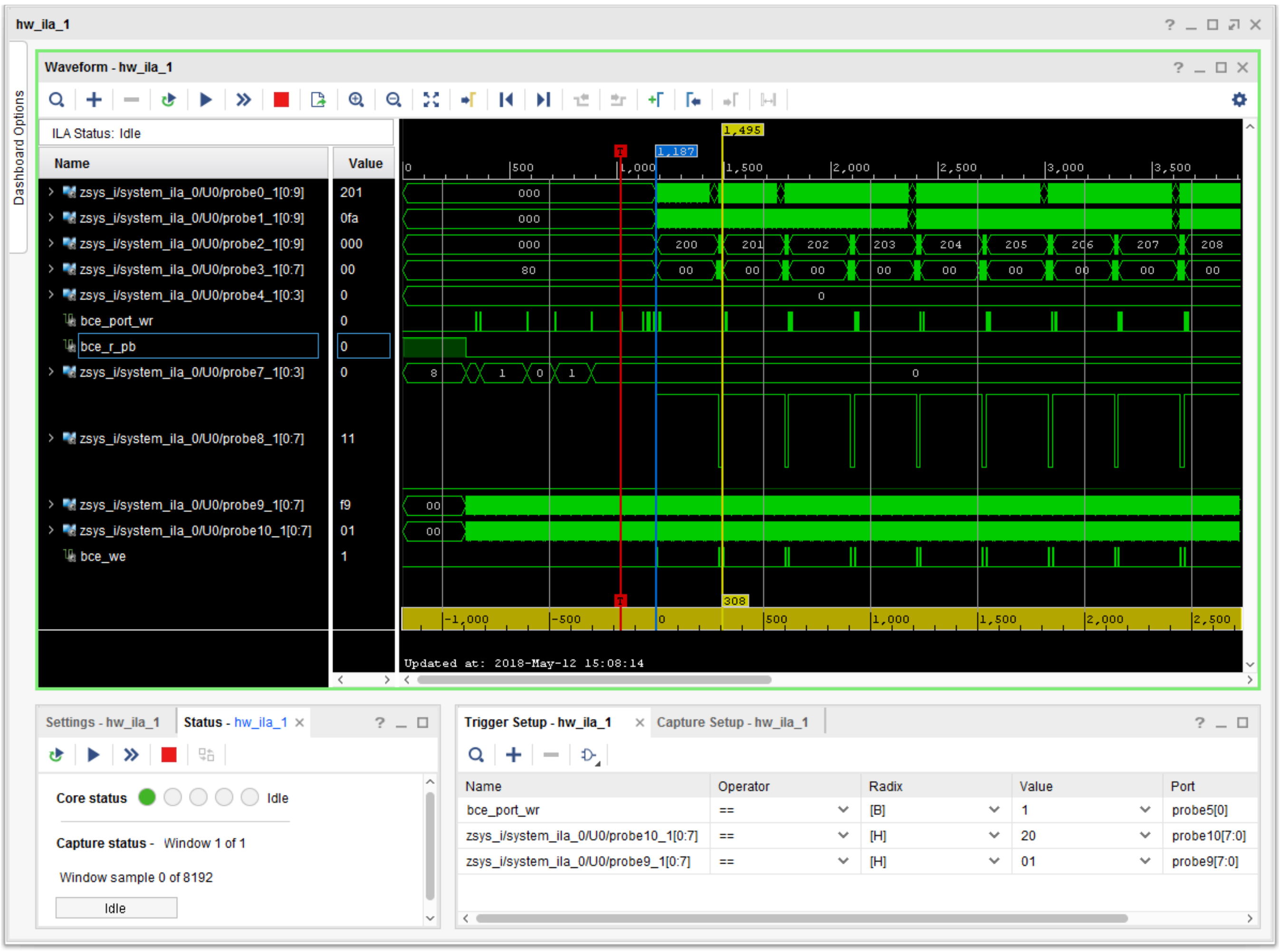
Figure 11: Release demo t01_s. Details of the 8xSIMD EdkDSP LMS filter computation.

In Vivado Lab Tool 2017.4.1, in the ILA configuration page, change the trigger condition to:
`(bce_port_wr ==1) AND (probe10[0:7]    ==0x20) AND (probe9[0:7]  ==0x00).`
`(bce_port_wr ==1) AND (bce_port_id[0:7]==0x20) AND (bce_port[0:7]==0x00).`
Selecion on the first line corresponds to the System ILA input to the EdkDSP probes on the second line. See connecrions of EdkDSP and Systen ILA on *Figure 3*.

In Vivado Lab Edition 2017.4.1, arm the System ILA core by pressing **Run Trigger** button in **Hardware** window. Armed System ILA core will wait until the recompiled EdkDSP firmware comes to the point, where PicoBlaze6 calls function `pb2dfu_set(0x20, 0).`

In case of TE0720-03-2IF, ILA captures 8K samples of all debug probes at 120 MHz.
In case of TE0720-03-1QF, ILA captures 8K samples of all debug probes at 100 MHz.
In case of TE0720-03-14S-1C, ILA captures 2K samples of all debug probes at 100 MHz.

Data are captured and sent via jtag USB connection in Vivado Lab Edition 2017.4.1 for visualisation and analysis in the waveform window. This snapshot stores the detailed trace of the FIR filter computation. See *Figure 12*.



ILA Status: Idle

Name	Value
zsys_i/system_ila_0/U0/probe0_1[0:9]	201
zsys_i/system_ila_0/U0/probe1_1[0:9]	0fa
zsys_i/system_ila_0/U0/probe2_1[0:9]	000
zsys_i/system_ila_0/U0/probe3_1[0:7]	00
zsys_i/system_ila_0/U0/probe4_1[0:3]	0
bce_port_wr	0
bce_r_pb	0
zsys_i/system_ila_0/U0/probe7_1[0:3]	0
zsys_i/system_ila_0/U0/probe8_1[0:7]	11
zsys_i/system_ila_0/U0/probe9_1[0:7]	f9
zsys_i/system_ila_0/U0/probe10_1[0:7]	01
bce_we	1

Updated at: 2018-May-12 15:08:14

Core status ● Idle

Capture status - Window 1 of 1

Window sample 0 of 8192

Idle

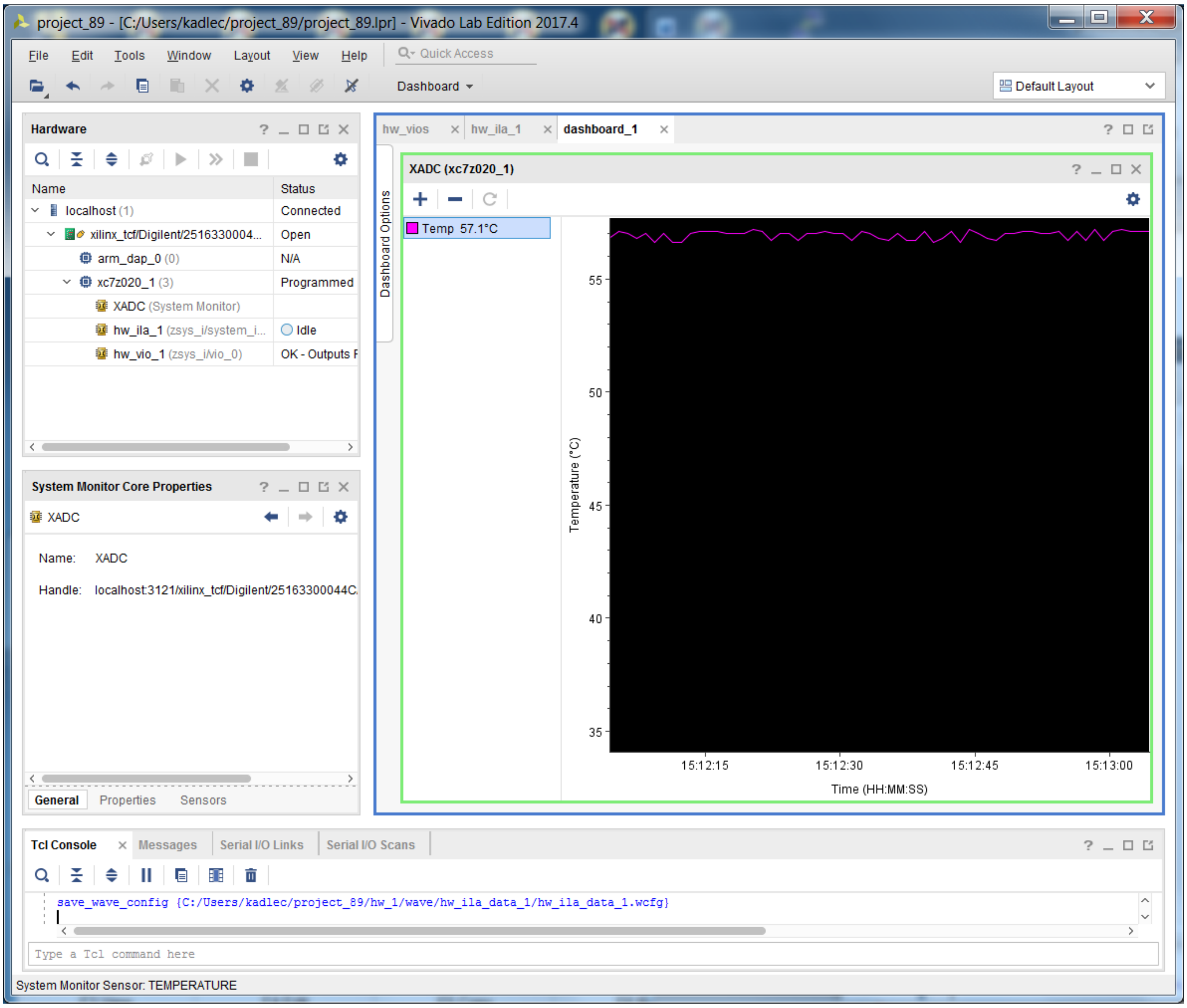
Name	Operator	Radix	Value	Port
bce_port_wr	==	[B]	1	probe5[0]
zsys_i/system_ila_0/U0/probe10_1[0:7]	==	[H]	20	probe10[7:0]
zsys_i/system_ila_0/U0/probe9_1[0:7]	==	[H]	01	probe9[7:0]

Figure 12: Release demo t01_s. Details of the 8xSIMD EdkDSP FIR filter computation.

In Vivado Lab. Tool, in the ILA configuration page, change the trigger condition to `(probe9[0:7]==0x01)`. This corresponds to the condition `bce_port[0:7]==0x01`. See connections in *Figure 3*. ILA will capture start of the FIR filter. See *Figure 12*. The PicoBlaze C code of the FIR example is listed in *Figure 19*.

The Vivado Lab. screens presented in *Figure 11* and *Figure 12* 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 19* with these initial lines of the PicoBlaze6 SW:

```
…
pb2dfu_set(0x20, 1);  // trigger (0x01 on port 0x20) for the ILA
for (i = 0; i < 4; i++) {
      for (j = 2; j <= 3; j++) {
            fir(j, n, op);
            pb2mb_eoc(led);
      }
}
…
```



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2516330004...	Open
arm_dap_0 (0)	N/A
xc7z020_1 (3)	Programmed
XADC (System Monitor)	
hw_ila_1 (zsys_i/system_i...	Idle
hw_vio_1 (zsys_i/vio_0)	OK - Outputs F

Name: XADC

Handle: localhost:3121/xilinx_tcf/Digilent/25163300044C.

```
save_wave_config {C:/Users/kadlec/project_89/hw_1/wave/hw_ila_data_1/hw_ila_data_1.wcfg}
```

Figure 13: Release demo t01_s. Standalone 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 Release 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_EdkDSP_2if_te706_ila8k_Release_INSTALL.zip`
to directory of your choice. We will use:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\`

(2) Select one of the examples (t01_l … t21_l) and copy the content of sd_card directory to the SD card.
Example. Copy the content (and the subdirectory with its content) from the directory:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release`
`\te01_l\Release\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 or TE0703-05 carrier board.

(7) Reset the carrier board.

- The Linux system will start. See *Figure 14*.
  type user name:
  `root`
  type password:
  `root`
- Mount SD card to the directory (See *Figure 15*) `/mnt` by typing:
  ```
  mount    /dev/mmcblk0p1    /mnt
  ```
- Change directory (See *Figure 15*) to `/mnt`
  ```
  cd /mnt
  ```
-Compile firmware for the PicoBlaze6 by the EdkDSP C compiler (See *Figure 15*):
  `./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 15*):
  `./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 user space application by typing:
```
./t01_l.elf
```
- The ARM terminal will present output from the `t01_l.elf` example. See *Figure 16*.
- 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`

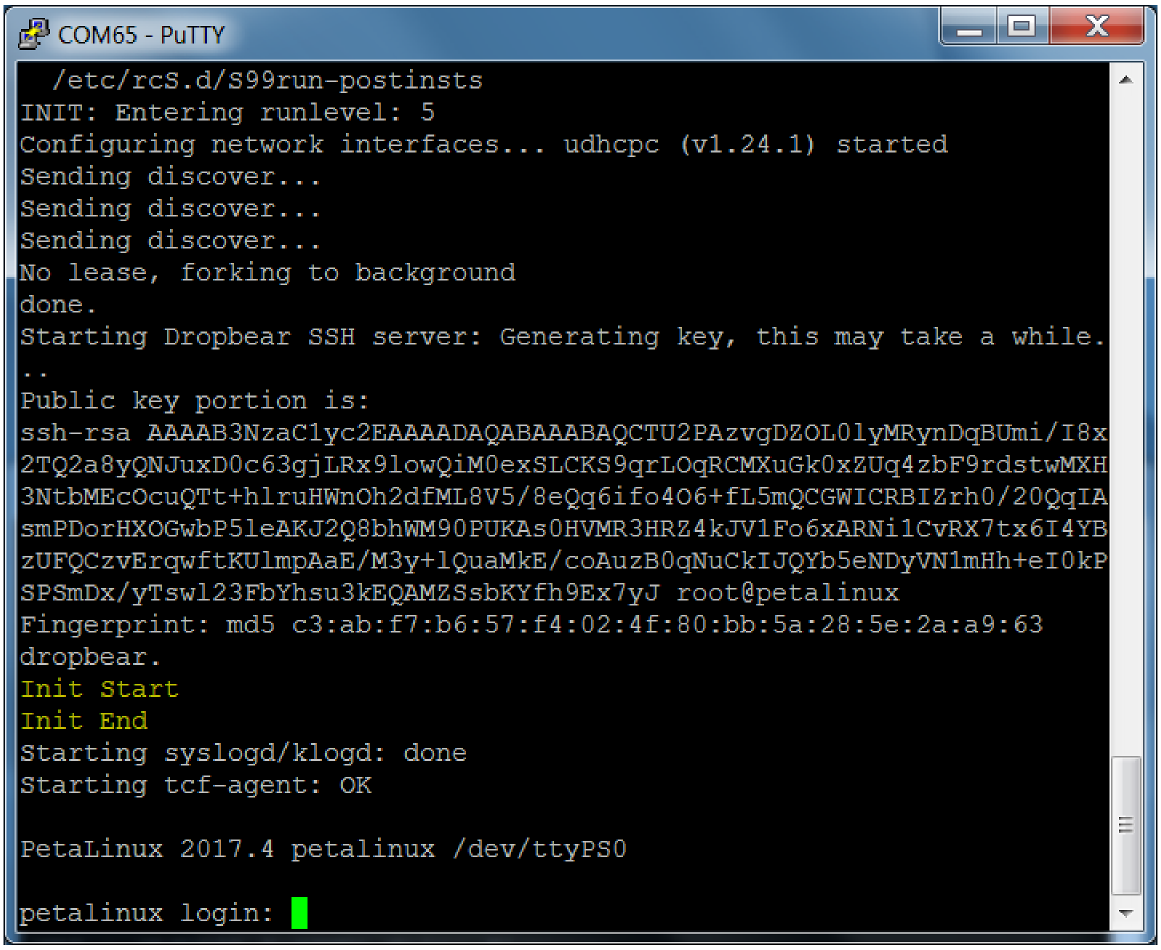
The output from the 8xSIMD EdkDSP is identical to the standalone output. See *Figure 8*.

(8) In PC, open the Vivado Lab tool. See *Figure 9*.
Open Vivado Lab tools 2017.4.1 hardware manager.
- Press Auto Connect icon in Hardware window
- Open description of debug nets present in file, thus specifying the probes file. See *Figure 10*.
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\debug_nets.ltx`
- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP.
 See *Figure 11*, *Figure 12*.

(9) Close Vivado Lab tool project.

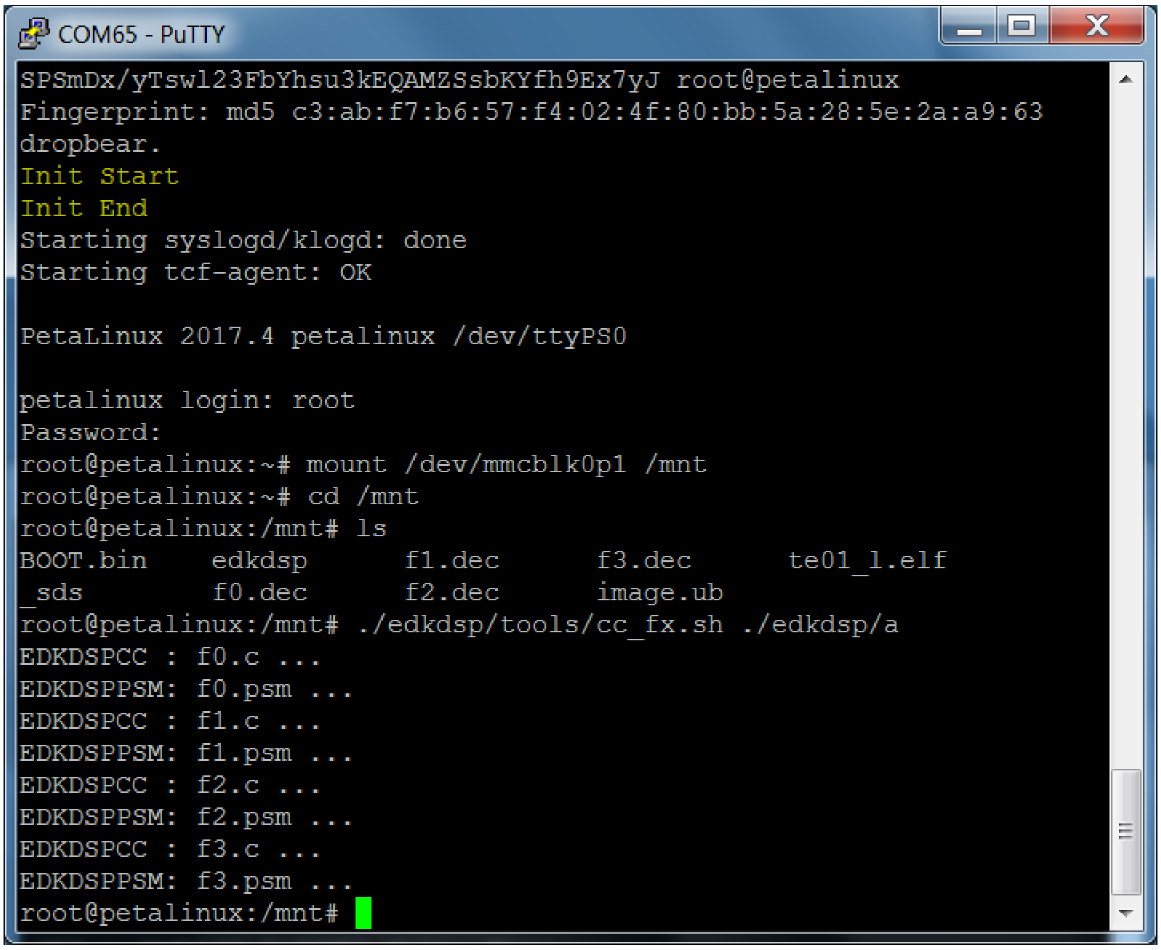
(10) Remove SD card and reprogram it in PC to test another example.

(11) Go to step (6).



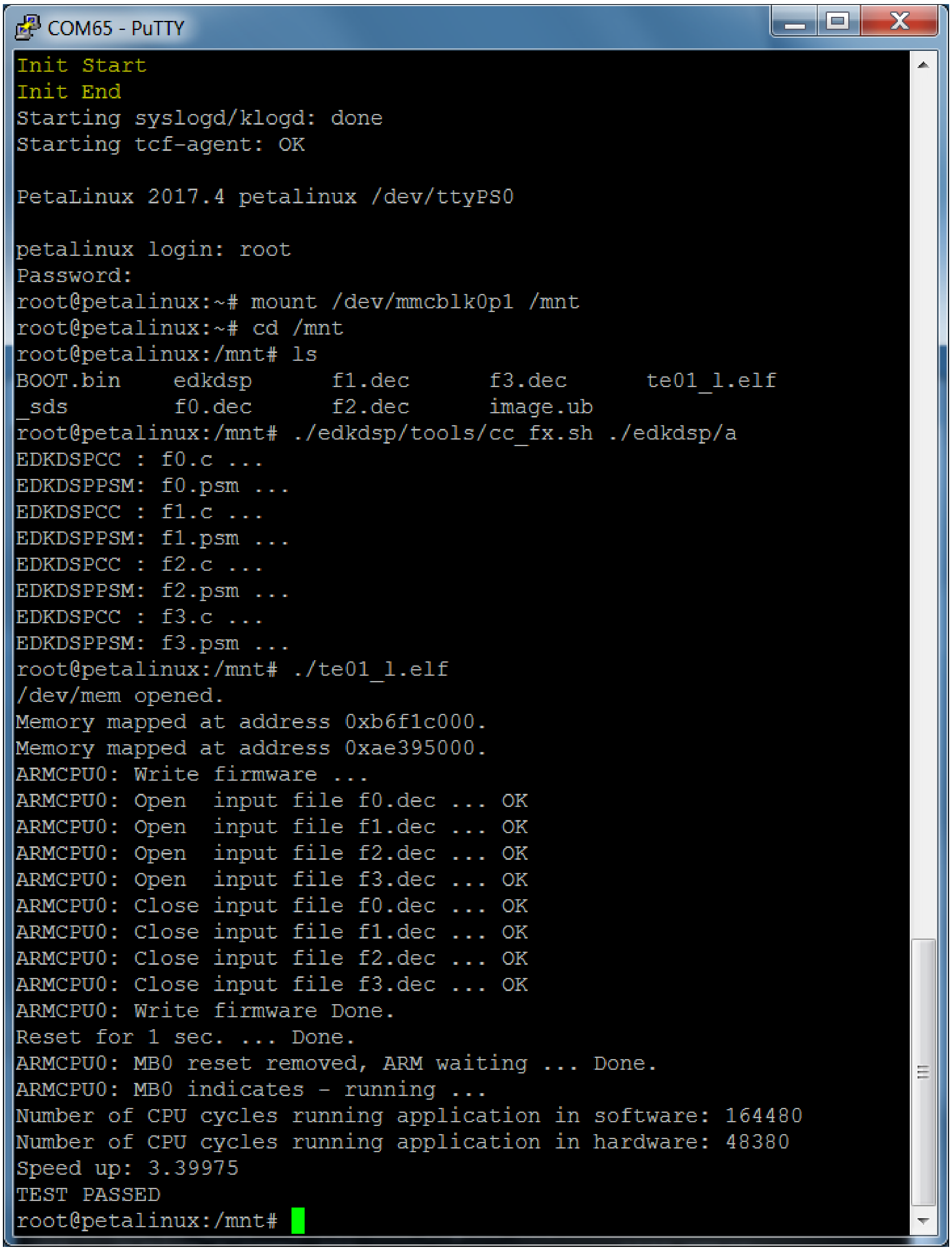
```
  /etc/rcS.d/S99run-postinsts
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.24.1) started
Sending discover...
Sending discover...
Sending discover...
No lease, forking to background
done.
Starting Dropbear SSH server: Generating key, this may take a while.
..
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCTU2PAzvgDZOL0lyMRynDqBUmi/I8x
2TQ2a8yQNJuxD0c63gjLRx9lowQiM0exSLCKS9qrLOqRCMXuGk0xZUq4zbF9rdstwMXH
3NtbMEcOcuQTt+hlruHWnOh2dfML8V5/8eQq6ifo4O6+fL5mQCGWICRBIZrh0/20QqIA
smPDorHXOGwbP5leAKJ2Q8bhWM90PUKAs0HVMR3HRZ4kJV1Fo6xARNi1CvRX7tx6I4YB
zUFQCzvErqwftKUlmpAaE/M3y+lQuaMkE/coAuzB0qNuCkIJQYb5eNDyVN1mHh+eI0kP
SPSmDx/yTswl23FbYhsu3kEQAMZSsbKYfh9Ex7yJ root@petalinux
Fingerprint: md5 c3:ab:f7:b6:57:f4:02:4f:80:bb:5a:28:5e:2a:a9:63
dropbear.
Init Start
Init End
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2017.4 petalinux /dev/ttyPS0

petalinux login:
```

Figure 14: Release demo t01_l. Linux start.



```
SPSmDx/yTswl23FbYhsu3kEQAMZSsbKYfh9Ex7yJ root@petalinux
Fingerprint: md5 c3:ab:f7:b6:57:f4:02:4f:80:bb:5a:28:5e:2a:a9:63
dropbear.
Init Start
Init End
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2017.4 petalinux /dev/ttyPS0

petalinux login: root
Password:
root@petalinux:~# mount /dev/mmcblk0p1 /mnt
root@petalinux:~# cd /mnt
root@petalinux:/mnt# ls
BOOT.bin     edkdsp      f1.dec      f3.dec      te01_l.elf
_sds         f0.dec      f2.dec      image.ub
root@petalinux:/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@petalinux:/mnt#
```

Figure 15: Release demo t01_l; Login, Compilation of firmware in the EdkDSP C Compiler.



```
Init Start
Init End
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2017.4 petalinux /dev/ttyPS0

petalinux login: root
Password:
root@petalinux:~# mount /dev/mmcblk0p1 /mnt
root@petalinux:~# cd /mnt
root@petalinux:/mnt# ls
BOOT.bin     edkdsp       f1.dec       f3.dec       te01_l.elf
_sds         f0.dec       f2.dec       image.ub
root@petalinux:/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@petalinux:/mnt# ./te01_l.elf
/dev/mem opened.
Memory mapped at address 0xb6f1c000.
Memory mapped at address 0xae395000.
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: 164480
Number of CPU cycles running application in hardware: 48380
Speed up: 3.39975
TEST PASSED
root@petalinux:/mnt#
```

Figure 16: Release demo t01_l; Program and start 8xSIMD EdkDSP demo.

9. Installation and Use of Debug Evaluation Package

The debug 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 debug 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.
    - In case of TE0720-03-2IF, ILA captures 8K samples of debug probes at 120 MHz.
    - In case of TE0720-03-1QF, ILA captures 8K samples of debug probes at 100 MHz.
    - In case of TE0720-03-14S-1C, ILA captures 2K samples of debug probes at 100 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 2017.4.1 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 2017.4.1 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.

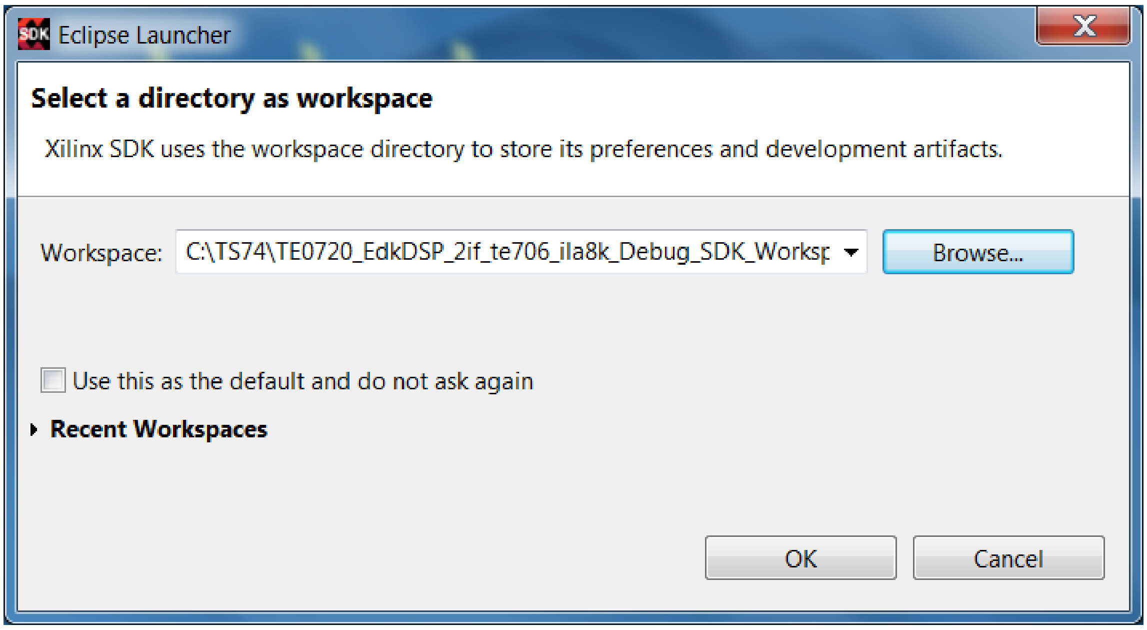
The extended evaluation package provides 21 precompiled designs for the Linux target and 19 precompiled designs for the standalone target as described in *Table 12* .

Installation and use of debug evaluation package – standalone examples

In case of standalone target:

(1) In Win 7 or Win 10 (64 bit PC), unzip the debug evaluation package:
`TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL.zip`
to directory of your choice. We will use:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\`

In Xilinx SDK 2017.4.1 create a new workspace in the directory of your choice. We will use:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\`



Select a directory as workspace

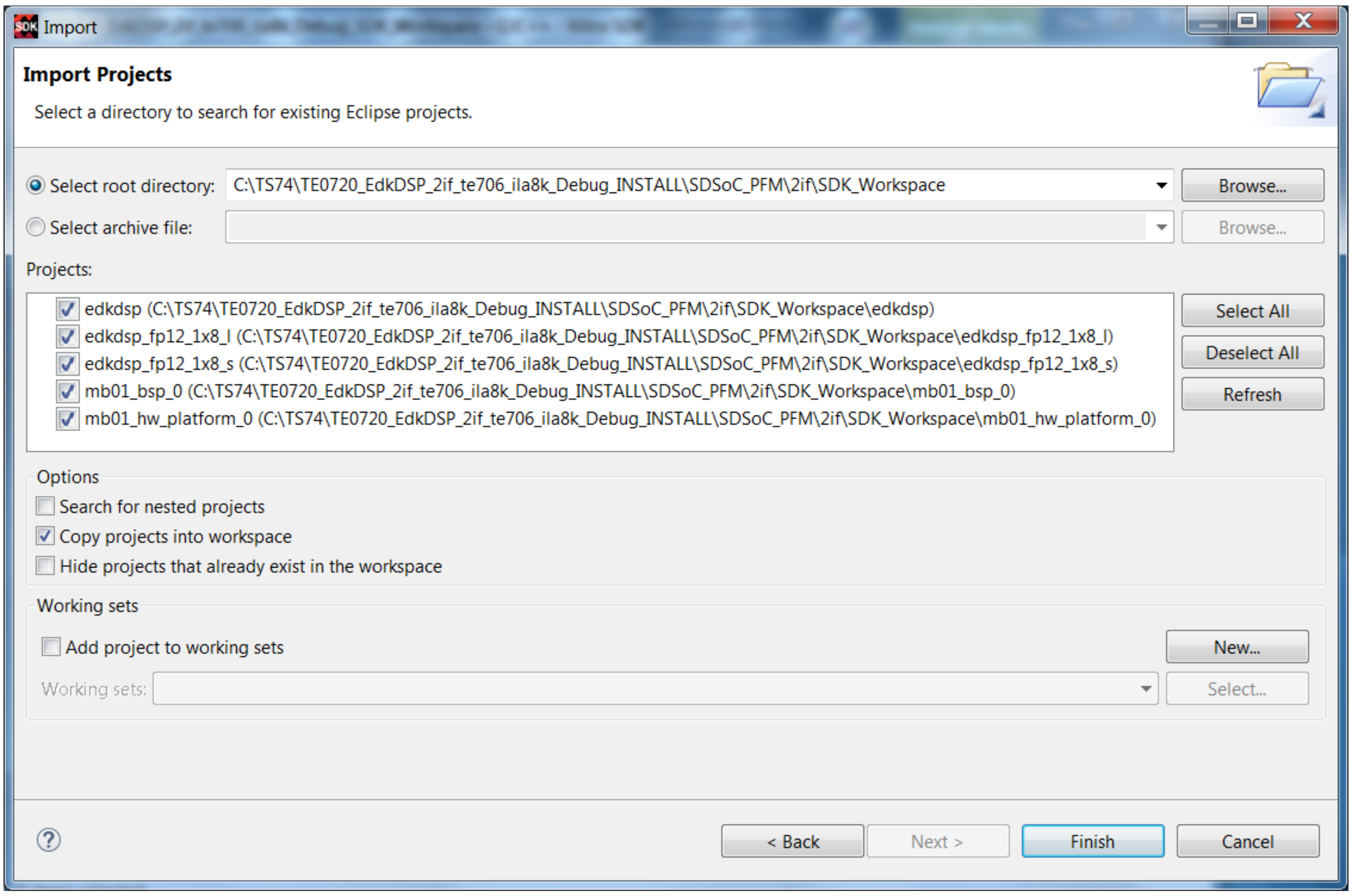
Xilinx SDK uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Worksp

☐ Use this as the default and do not ask again

Recent Workspaces

Figure 17: Create new SDK 2017.4.1 workspace.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

[✓ Select root directory:] C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace

Select archive file:

Projects:

- ☑ edkdsp (C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\edkdsp)
- ☑ edkdsp_fp12_1x8_l (C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\edkdsp_fp12_1x8_l)
- ☑ edkdsp_fp12_1x8_s (C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\edkdsp_fp12_1x8_s)
- ☑ mb01_bsp_0 (C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\mb01_bsp_0)
- ☑ mb01_hw_platform_0 (C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\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 18: Import the extended debug evaluation package projects into the SDK Workspace.

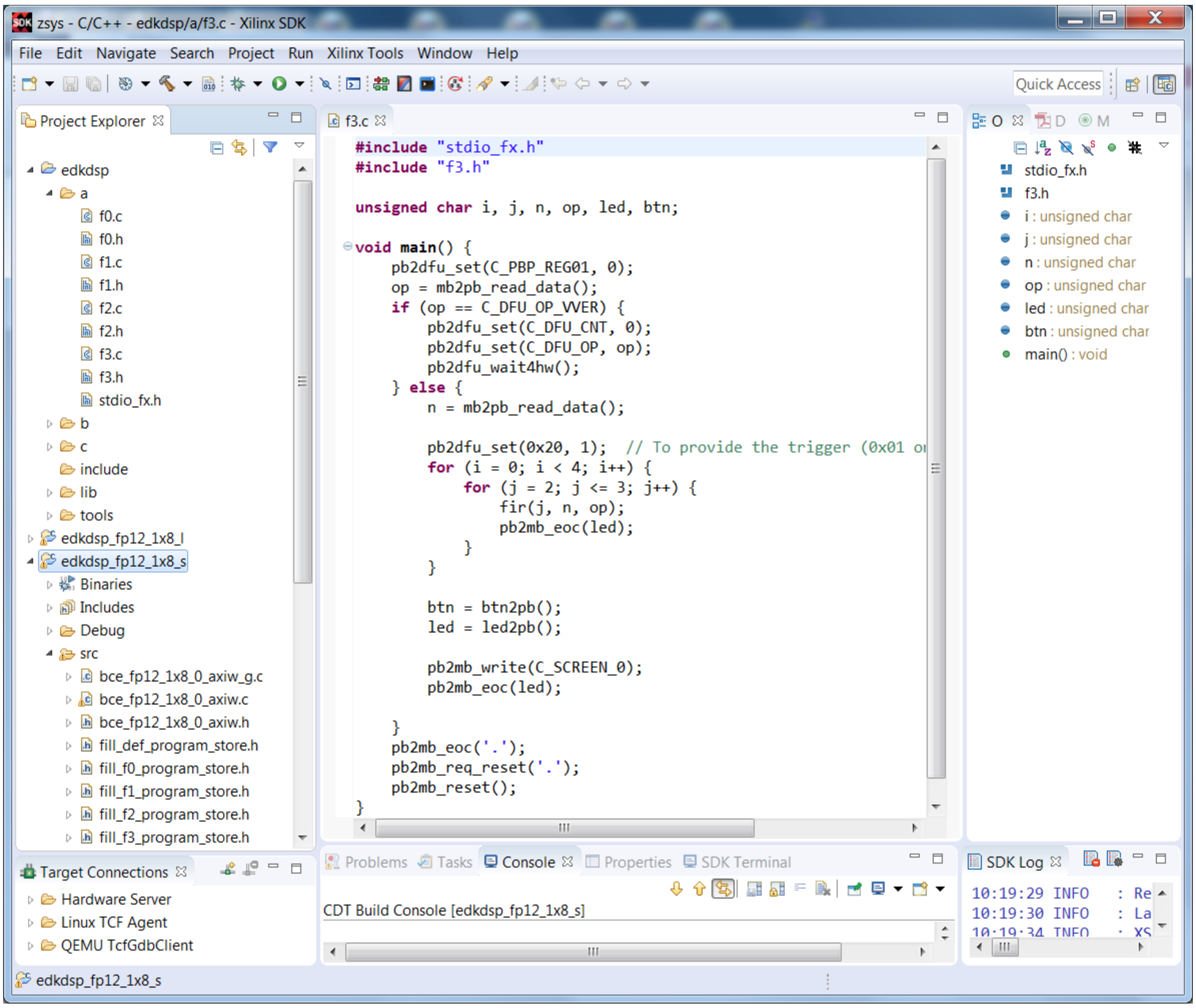
Import (with copy) all SDK projects from:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\`
to the new SDK workspace.
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\`
Both Microblaze projects will be compiled automatically by the SDK for the debug configuration.

(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`** to the root of the SD card from:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SD_debug\te01_s\Release\sd_card\BOOT.bin`

(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.



```c
#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();
}
```

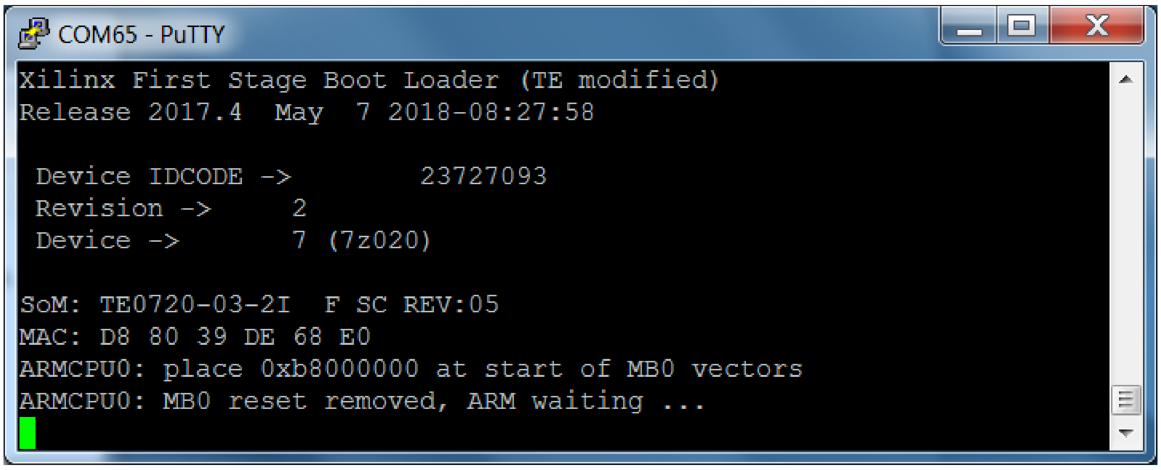
```
10:19:29 INFO  : Re
10:19:30 INFO  : La
10:19:34 INFO  : XS
```

Figure 19: SDK compiles MicroBlaze SW projects for the standalone debug target.

(6) Insert SD card to the TE0706-02 or TE0703-05 carrier board.

(7) Reset the carrier board.

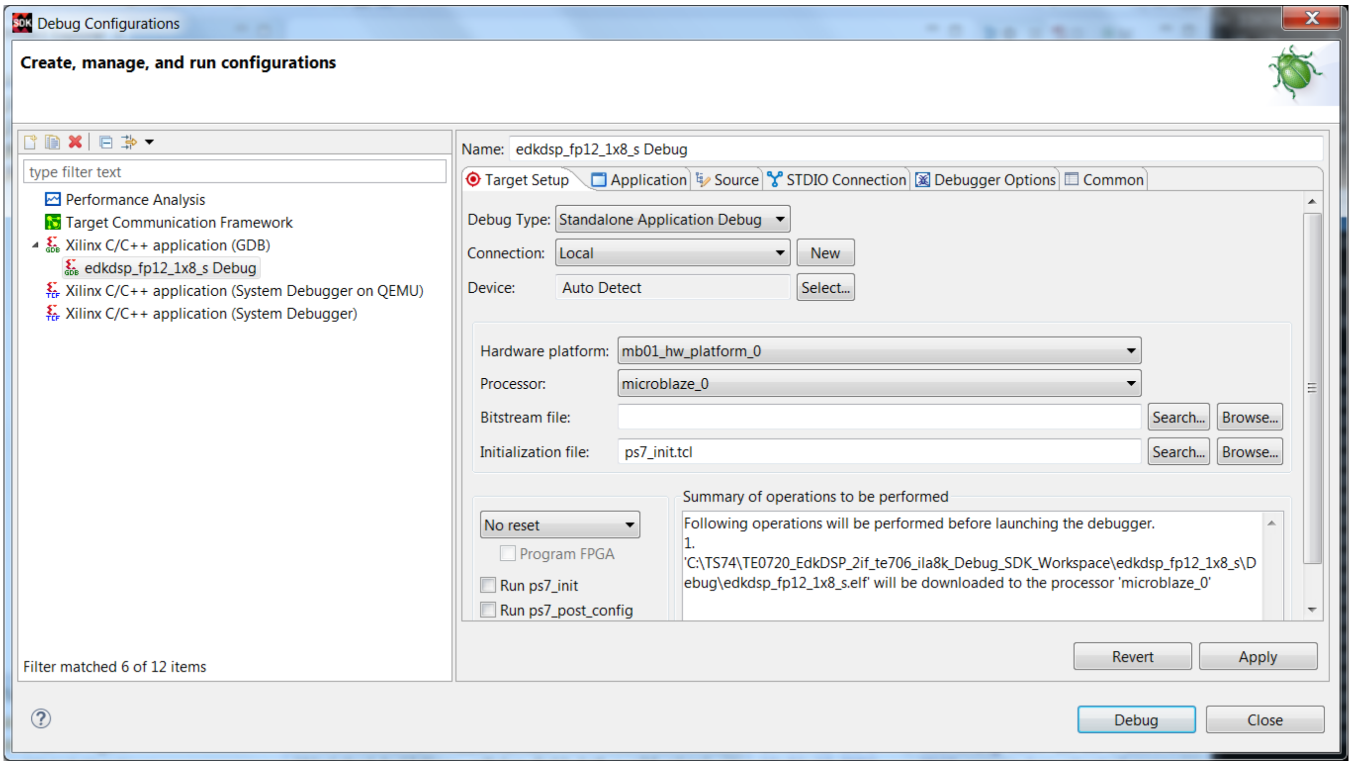
- The standalone system will start.
- The ARM terminal will present output from the **`t01_s`** example.
- The Arm application is waiting for the MicroBlaze.



```
Xilinx First Stage Boot Loader (TE modified)
Release 2017.4  May  7 2018-08:27:58

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

SoM: TE0720-03-2I  F SC REV:05
MAC: D8 80 39 DE 68 E0
ARMCPU0: place 0xb8000000 at start of MB0 vectors
ARMCPU0: MB0 reset removed, ARM waiting ...
```

Figure 20: Debug demo t01_l; Execution of the ./t01_s.elf example from the SD card.

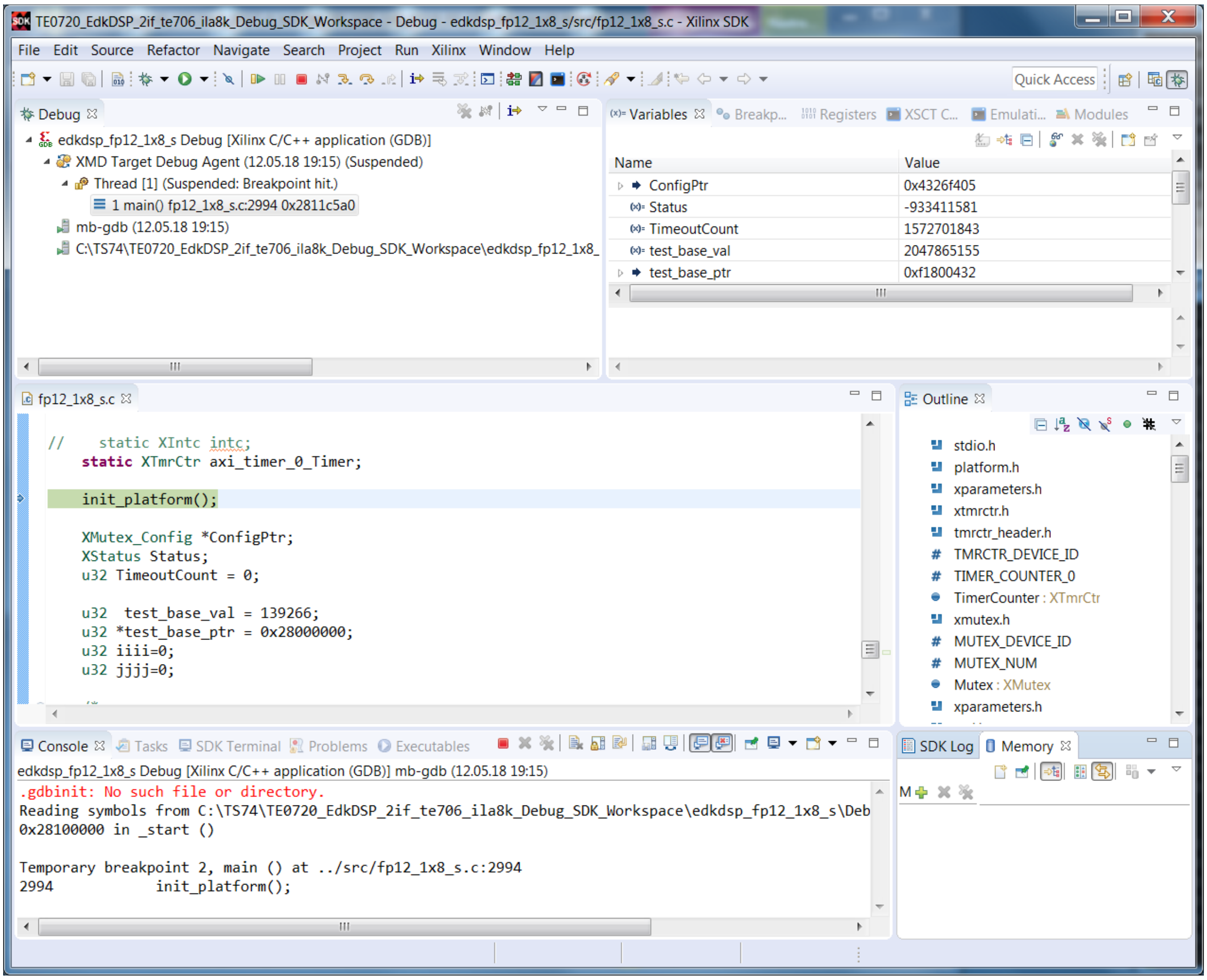
- The Xilinx SDK project **`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_s`** includes PicoBlaze6 firmware header files **`fill_f0_program_store.h, fill_f1_program_store.h, fill_f2_program_store.h`** and **`fill_f3_program_store.h`** Note: These files can be recompiled from the 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`** and **`fill_f3_program_store.h`**.
- In the Xilinx SDK workspace, select Debug of MicroBlaze project **`edkdsp_fp12_1x8_s`**. In the Debug Configurations, select “No reset”, unselect “Run ps7_init”, unselect “Run ps7_post_config” click “Apply”.


Debug Configurations
Create, manage, and run configurations
type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
edkdsp_fp12_1x8_s Debug
Xilinx C/C++ application (System Debugger on QEMU)
Xilinx C/C++ application (System Debugger)
Filter matched 6 of 12 items
Target Setup
Application
Source
STDIO Connection
Debugger Options
Common
New
Select...
Search...
Browse...
Program FPGA
Revert
Apply
Debug
Close

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.
'C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_s\Debug\edkdsp_fp12_1x8_s.elf' will be downloaded to the processor 'microblaze_0'

Figure 21: Debug demo t01_s; Open project edkdsp_fp12_1x8_s for debug.


TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace - Debug - edkdsp_fp12_1x8_s/src/fp12_1x8_s.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
Quick Access
Debug
Variables
Breakp...
Registers
XSCT C...
Emulati...
Modules
Outline
Console
Tasks
SDK Terminal
Problems
Executables
SDK Log
Memory

```
edkdsp_fp12_1x8_s Debug [Xilinx C/C++ application (GDB)]
  XMD Target Debug Agent (12.05.18 19:15) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() fp12_1x8_s.c:2994 0x2811c5a0
  mb-gdb (12.05.18 19:15)
  C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_
```

Name	Value
ConfigPtr	0x4326f405
Status	-933411581
TimeoutCount	1572701843
test_base_val	2047865155
test_base_ptr	0xf1800432

fp12_1x8_s.c

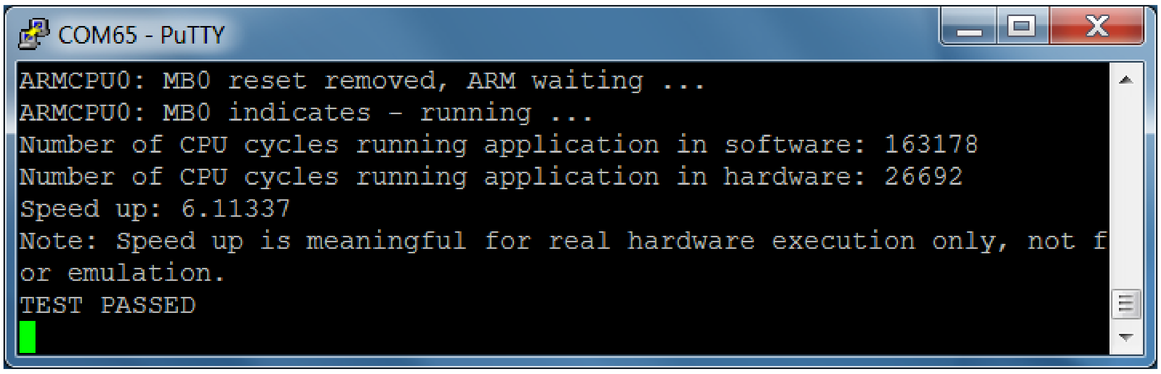
```
//    static XIntc intc;
    static XTmrCtr axi_timer_0_Timer;

    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 (12.05.18 19:15)
.gdbinit: No such file or directory.
Reading symbols from C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_s\Deb
0x28100000 in _start ()

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

Figure 22: Debug demo t01_s; Start the free-run from the debugger.

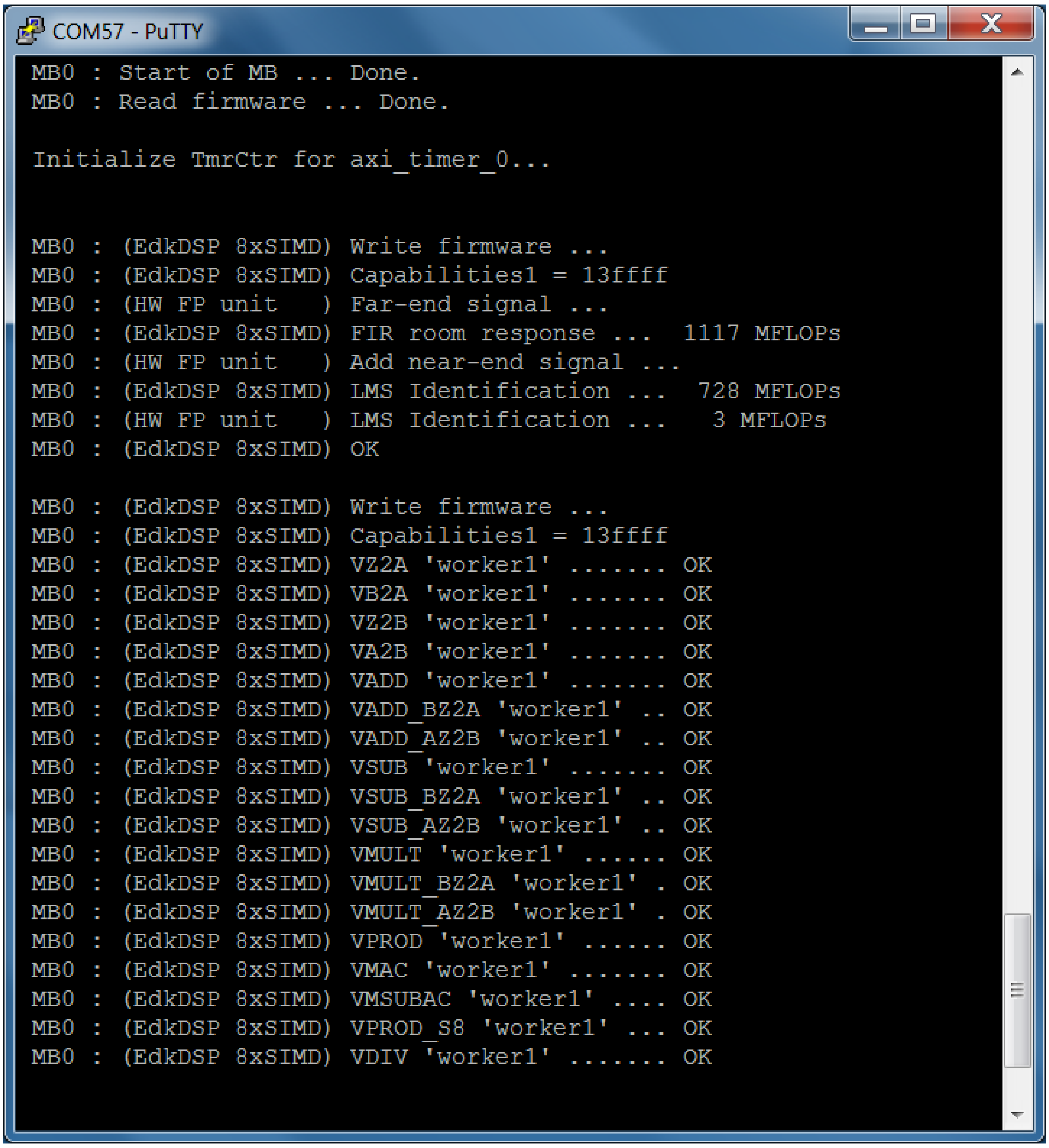
- In the SDK debugger, step through the MicroBlaze source code, inspect content of variables, set the breakpoints, step through the code and finally select the free run of the MicroBlaze code.
- At this stage, the ARM terminal will present the output from the ARM **t01_s.elf** example. See *Figure 23*.



```
ARMCPU0: MB0 reset removed, ARM waiting ...
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 163178
Number of CPU cycles running application in hardware: 26692
Speed up: 6.11337
Note: Speed up is meaningful for real hardware execution only, not f
or emulation.
TEST PASSED
```

Figure 23: 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 24*.



```
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 ...  1117 MFLOPs
MB0 : (HW FP unit    ) Add near-end signal ...
MB0 : (EdkDSP 8xSIMD) LMS Identification ...  728 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 24: Debug demo t01_s; MicroBlaze project output (Compiled for Debug).

(8) In PC, open the Vivado Lab tool. See *Figure 9*.
- 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 as:
c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SD_debug\te01_s\Release\debug_nets.ltx
- Set the ILA trigger conditions and observe process of computation in the 8xSIMD EdkDSP IP. See *Figure 10*, *Figure 11*, *Figure 12*.
- Open new perspective and observe the chip temperature. See *Figure 13*. 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 Debug Evaluation Package – Linux examples

(1) In Win 7 or Win 10 (64bit PC), unzip the basic evaluation package
`TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL.zip`
to directory of your choice. We will use:
`c:\TS74\TE0720_EdkDSP_14s_te706_ila2k_Debug_INSTALL\`
Open new Xilinx SDK 2017.4.1 workspace in the directory
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug\SDK_Workspace\`
Import (with copy) all SDK projects from
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SDK_Workspace\`
to the new 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
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SD_debug\`
`te01_l\Release\sd_card\`
to the root of the SD card/

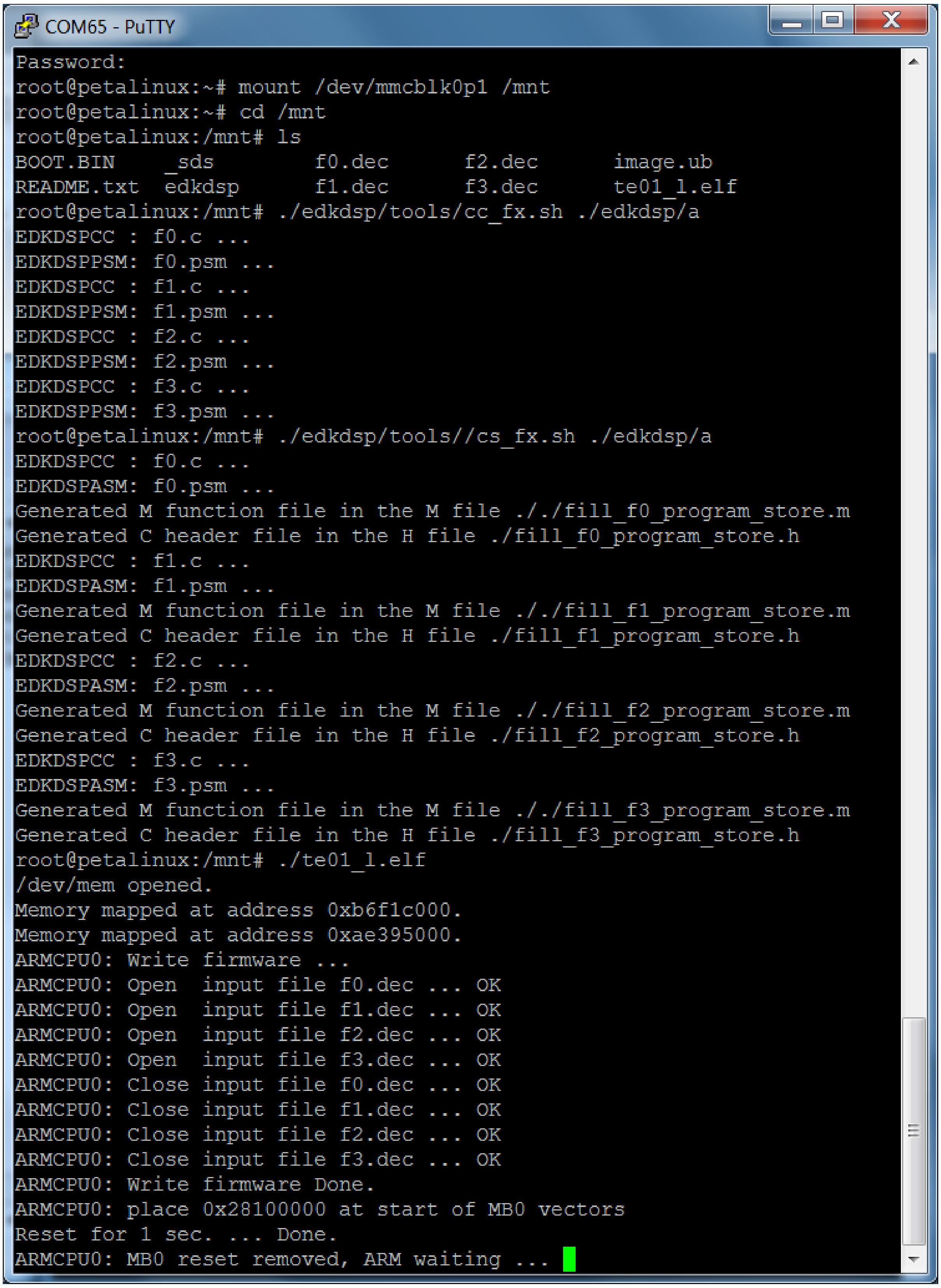
(3) Connect USB cable from J7 connector to the PC. It will serve as ARM terminal and JTAG line.

(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 or TE0703-05 carrier board.

(7) Reset the carrier board.
- The Linux system will start. See *Figure 25*.
- Type the Linux user name and password:
`root`
`root`



```
Password:
root@petalinux:~# mount /dev/mmcblk0p1 /mnt
root@petalinux:~# cd /mnt
root@petalinux:/mnt# ls
BOOT.BIN     _sds         f0.dec       f2.dec       image.ub
README.txt  edkdsp       f1.dec       f3.dec       te01_l.elf
root@petalinux:/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@petalinux:/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@petalinux:/mnt# ./te01_l.elf
/dev/mem opened.
Memory mapped at address 0xb6f1c000.
Memory mapped at address 0xae395000.
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 25: Compiled EdkDSP firmware. Started debug demo - Linux target t01_l.

- Mount SD card to the directory (See *Figure 25*) **`/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 25*) :

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

- 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
```

  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 25*):

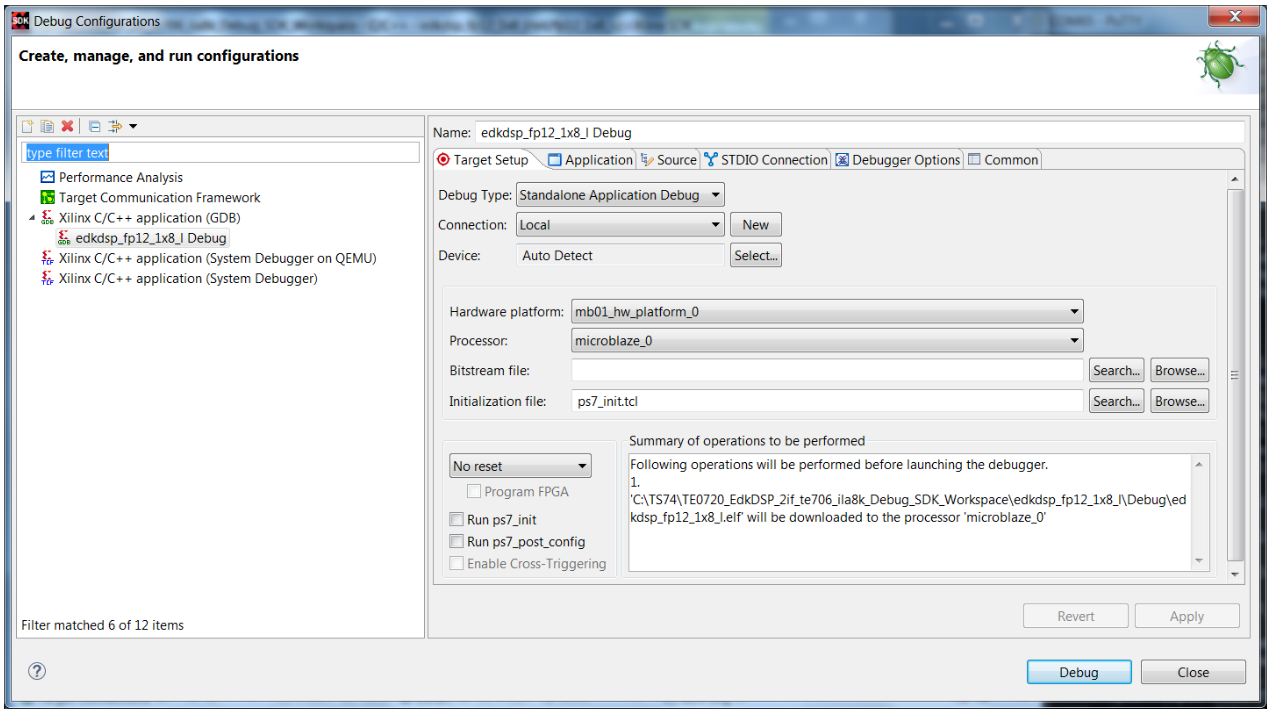
```
./edkdsp/tools/cs_fx.sh ./edkdsp/a
```

  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 25*.
- 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 26*):

```
c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\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 [New]

Device: Auto Detect [Select...]

Hardware platform: mb01_hw_platform_0

Processor: microblaze_0

Bitstream file: [Search...] [Browse...]

Initialization file: ps7_init.tcl [Search...] [Browse...]

No reset

- [ ] Program FPGA
- [ ] Run ps7_init
- [ ] Run ps7_post_config
- [ ] Enable Cross-Triggering

Summary of operations to be performed

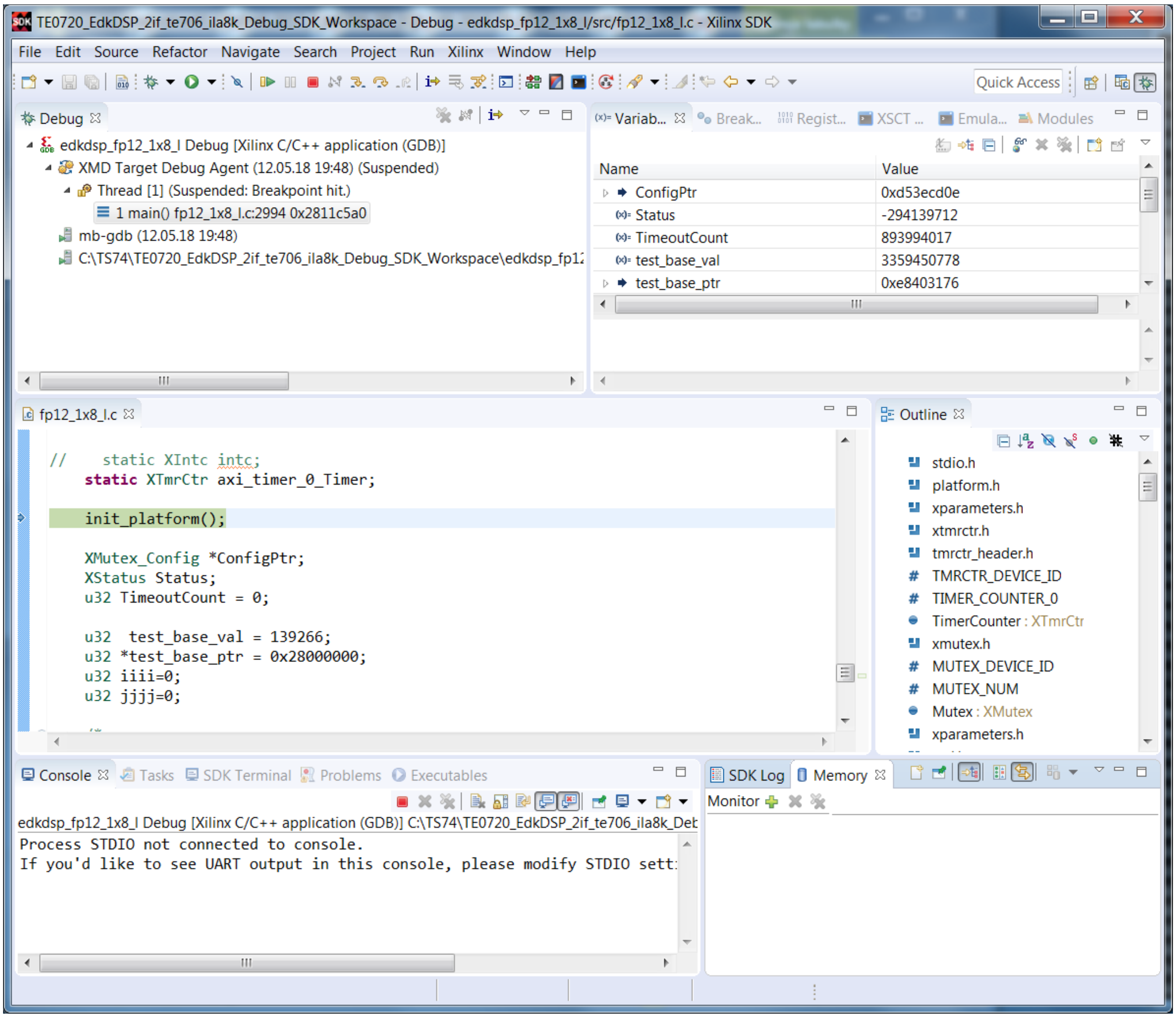
```
Following operations will be performed before launching the debugger.
1.
'C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_l\Debug\ed
kdsp_fp12_1x8_l.elf' will be downloaded to the processor 'microblaze_0'
```

[Revert] [Apply]

[Debug] [Close]

Figure 26: 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 27*.
- In the SDK debugger, select free run of the MicroBlaze code. See *Figure 27*.
- 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: **`./edkdsp/a/f0.c, ./edkdsp/a/f1.c, ./edkdsp/a/f2.c`** and **`./edkdsp/a/f3.c`**
  The terminal Output is identical to *Figure 24*.
- The ARM terminal will continue to present output from the **`t01_l.elf`** example. See *Figure 28*.
- In ARM terminal, type:
  `ls -lr`
  to see listing of files compiled by the EdkDSP C compiler. See *Figure 28*.
  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
  `c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_s`


TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace - Debug - edkdsp_fp12_1x8_l/src/fp12_1x8_l.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Window Help
Quick Access
Debug
Variab... Break... Regist... XSCT ... Emula... Modules
fp12_1x8_l.c
Outline
Console Tasks SDK Terminal Problems Executables
SDK Log Memory
Monitor

```
edkdsp_fp12_1x8_l Debug [Xilinx C/C++ application (GDB)]
  XMD Target Debug Agent (12.05.18 19:48) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() fp12_1x8_l.c:2994 0x2811c5a0
  mb-gdb (12.05.18 19:48)
  C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12
```

Name	Value
ConfigPtr	0xd53ecd0e
Status	-294139712
TimeoutCount	893994017
test_base_val	3359450778
test_base_ptr	0xe8403176

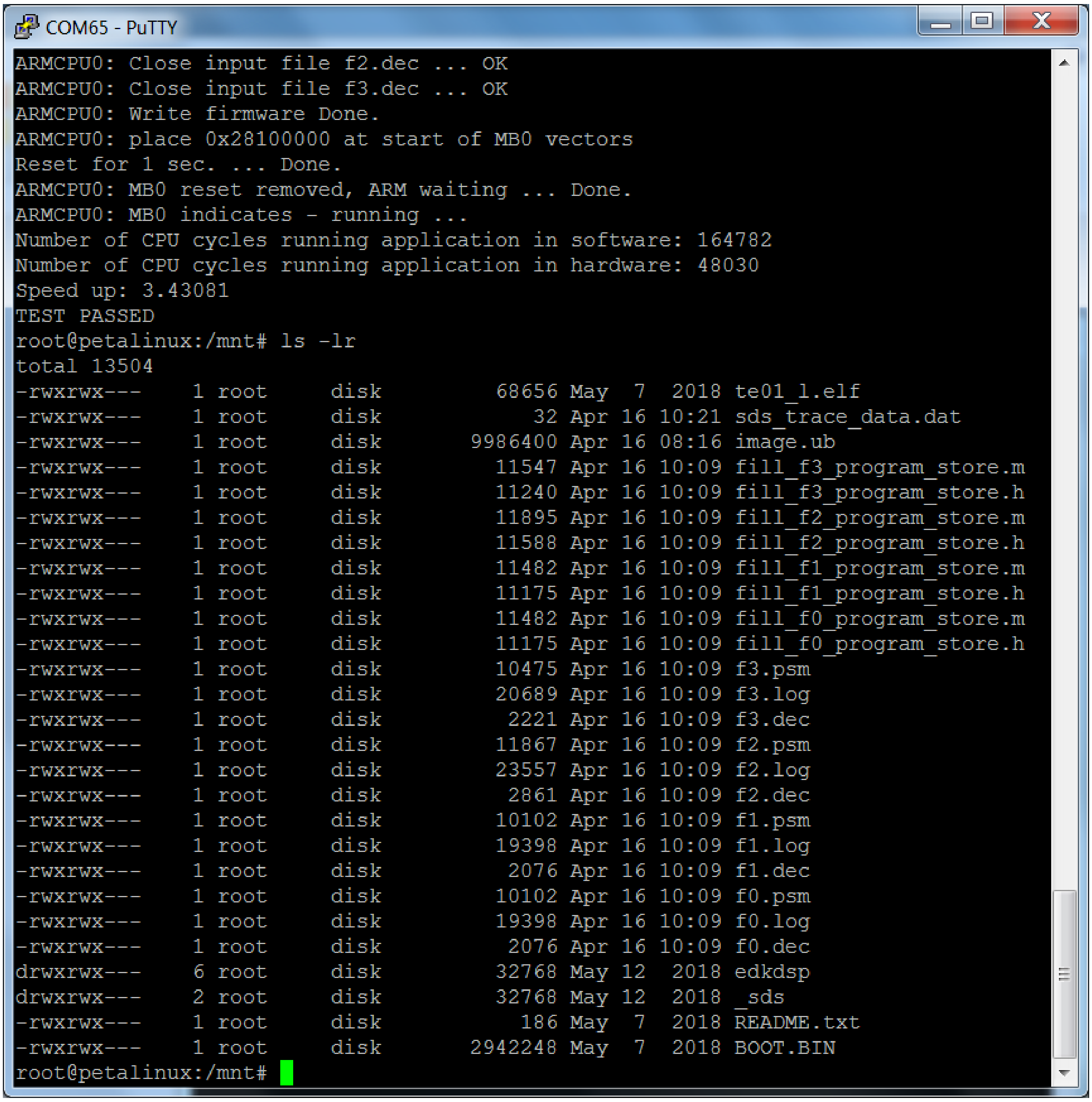
```
//    static XIntc intc;
    static XTmrCtr axi_timer_0_Timer;

    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)] C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Det
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDIO sett:
```

Figure 27: Select free run of MicroBlaze project edkdsp_fp12_1x8_l.

(8) In PC, open the Vivado Lab tool hardware manager. See *Figure 9*.
- 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
 `c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_INSTALL\SDSoC_PFM\2if\SD_debug\te01_s\Release\debug_nets.ltx`
- Set the ILA trigger conditions. See *Figure 10*, *Figure 11*, *Figure 12*. Close Vivado Lab tool project.
(9) In SDK debugger, stop MicroBlaze processor and close the debug session
(10) Exit from Linux by typing on the ARM terminal: **`exit`**
(11) Remove SD card and reprogram it in the PC to test another example.
(12) Go to step (6).

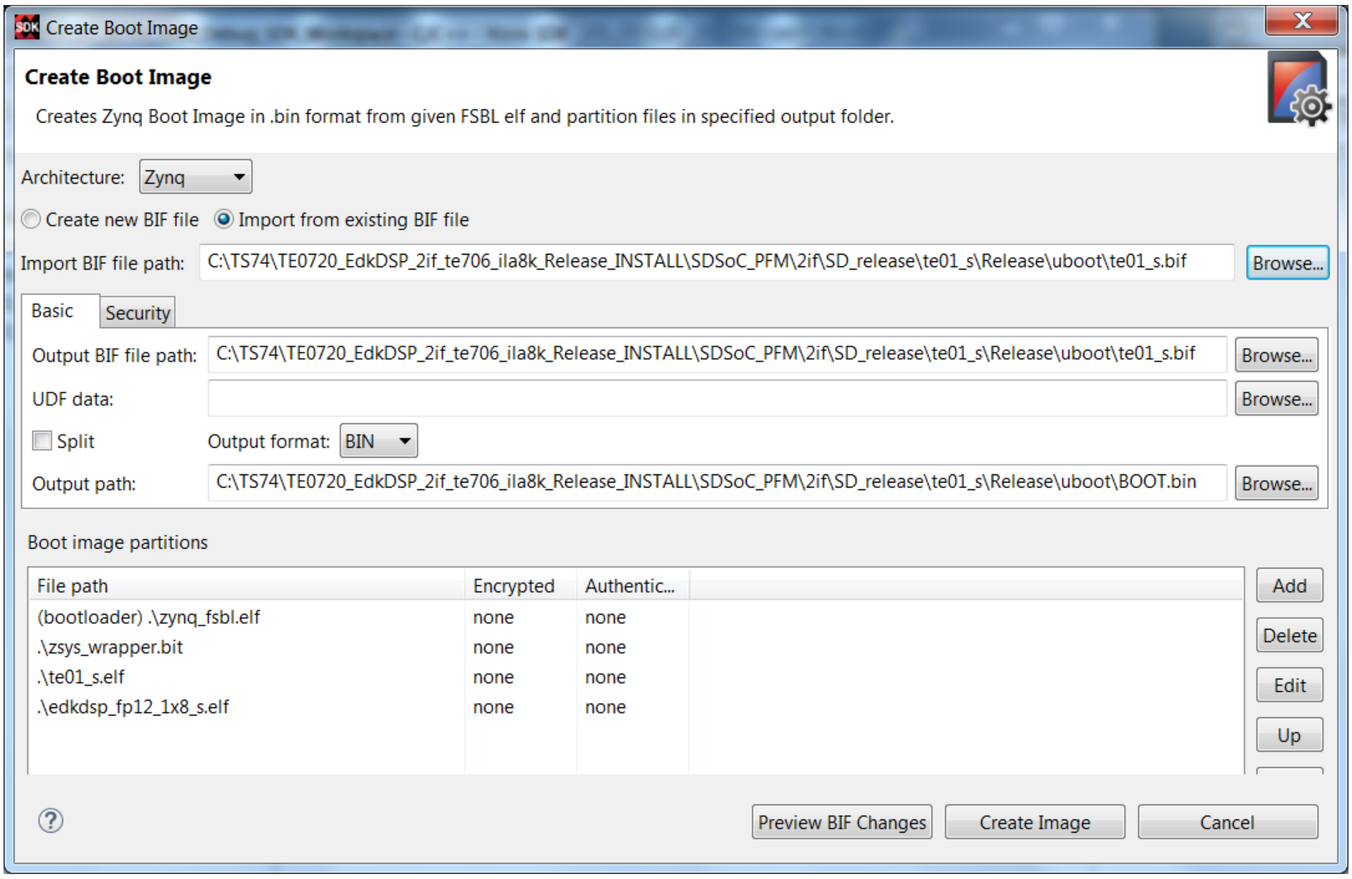


```
ARMCPU0: Close input file f2.dec ... OK
ARMCPU0: Close input file f3.dec ... OK
ARMCPU0: Write firmware Done.
ARMCPU0: place 0x28100000 at start of MB0 vectors
Reset for 1 sec. ... Done.
ARMCPU0: MB0 reset removed, ARM waiting ... Done.
ARMCPU0: MB0 indicates - running ...
Number of CPU cycles running application in software: 164782
Number of CPU cycles running application in hardware: 48030
Speed up: 3.43081
TEST PASSED
root@petalinux:/mnt# ls -lr
total 13504
-rwxrwx---    1 root     disk         68656 May  7  2018 te01_l.elf
-rwxrwx---    1 root     disk            32 Apr 16 10:21 sds_trace_data.dat
-rwxrwx---    1 root     disk       9986400 Apr 16 08:16 image.ub
-rwxrwx---    1 root     disk         11547 Apr 16 10:09 fill_f3_program_store.m
-rwxrwx---    1 root     disk         11240 Apr 16 10:09 fill_f3_program_store.h
-rwxrwx---    1 root     disk         11895 Apr 16 10:09 fill_f2_program_store.m
-rwxrwx---    1 root     disk         11588 Apr 16 10:09 fill_f2_program_store.h
-rwxrwx---    1 root     disk         11482 Apr 16 10:09 fill_f1_program_store.m
-rwxrwx---    1 root     disk         11175 Apr 16 10:09 fill_f1_program_store.h
-rwxrwx---    1 root     disk         11482 Apr 16 10:09 fill_f0_program_store.m
-rwxrwx---    1 root     disk         11175 Apr 16 10:09 fill_f0_program_store.h
-rwxrwx---    1 root     disk         10475 Apr 16 10:09 f3.psm
-rwxrwx---    1 root     disk         20689 Apr 16 10:09 f3.log
-rwxrwx---    1 root     disk          2221 Apr 16 10:09 f3.dec
-rwxrwx---    1 root     disk         11867 Apr 16 10:09 f2.psm
-rwxrwx---    1 root     disk         23557 Apr 16 10:09 f2.log
-rwxrwx---    1 root     disk          2861 Apr 16 10:09 f2.dec
-rwxrwx---    1 root     disk         10102 Apr 16 10:09 f1.psm
-rwxrwx---    1 root     disk         19398 Apr 16 10:09 f1.log
-rwxrwx---    1 root     disk          2076 Apr 16 10:09 f1.dec
-rwxrwx---    1 root     disk         10102 Apr 16 10:09 f0.psm
-rwxrwx---    1 root     disk         19398 Apr 16 10:09 f0.log
-rwxrwx---    1 root     disk          2076 Apr 16 10:09 f0.dec
drwxrwx---    6 root     disk         32768 May 12  2018 edkdsp
drwxrwx---    2 root     disk         32768 May 12  2018 _sds
-rwxrwx---    1 root     disk           186 May  7  2018 README.txt
-rwxrwx---    1 root     disk       2942248 May  7  2018 BOOT.BIN
root@petalinux:/mnt#
```

Figure 28: 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 19*. 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 29*.



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: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\uboot\te01_s.bif

Output BIF file path: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\uboot\te01_s.bif

UDF data:

☐ Split Output format: BIN

Output path: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\uboot\BOOT.bin

Boot image partitions

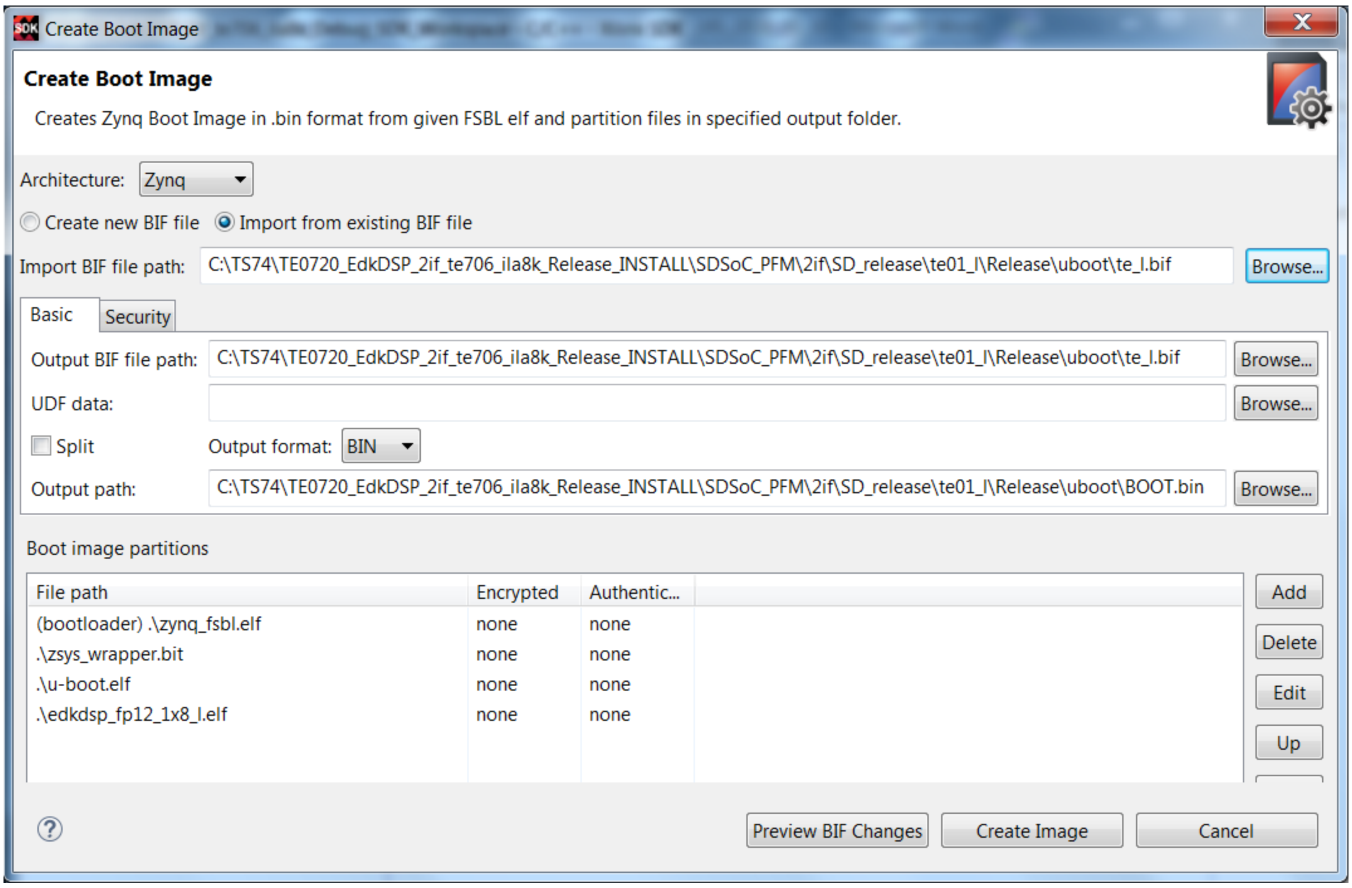
File path	Encrypted	Authentic...
(bootloader) .\zynq_fsbl.elf	none	none
.\zsys_wrapper.bit	none	none
.\te01_s.elf	none	none
.\edkdsp_fp12_1x8_s.elf	none	none

Figure 29: Create BOOT.bin for the t01_s demo.

See the content of directory:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\uboot\`
The new **`BOOT.bin`** image can be created from these five files:
`t01_s.bif, zynq_fsbl.elf, zynq_wrapper.bit.elf, t01_s.elf, edkdsp_fp12_1x8_s.elf`
Replace an old **`edkdsp_fp12_1x8_s.elf`** file with the new file recompiled in the SDK (with new PicoBlaze6 firmware headers) from the SDK project:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_s`
Use the **`BOOT.bin`** generation utility (In the SDK workspace: Xilinx Tools -> Create Boot Image) and create the new **`BOOT.bin`** file (See *Figure 29*):
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\uboot\BOOT.bin`
Copy this new **`BOOT.bin`** file it to:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_s\Release\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 30*.


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: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\uboot\te_l.bif

Basic | Security

Output BIF file path: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\uboot\te_l.bif

UDF data:

Split   Output format: BIN

Output path: C:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\uboot\BOOT.bin

Boot image partitions

File path	Encrypted	Authentic...
(bootloader) .\zynq_fsbl.elf	none	none
.\zsys_wrapper.bit	none	none
.\u-boot.elf	none	none
.\edkdsp_fp12_1x8_l.elf	none	none

Figure 30: Create BOOT.bin for the t01_l demo.

Use the content of directory:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\uboot\` The new **`BOOT.bin`** image can be created from these files:
`te_l.bif, zynq_fsbl.elf, zynq_wrapper.bit.elf, u-boot.elf, edkdsp_fp12_1x8_l.elf`

Replace:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\uboot\edkdsp_fp12_1x8_l.elf`
with a new file recompiled in the SDK project:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Debug_SDK_Workspace\edkdsp_fp12_1x8_l`
Use the **`BOOT.bin`** generation utility of the SDK and create the new **`BOOT.bin`** file:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\uboot\BOOT.bin`

Copy this new **`BOOT.bin`** file to:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\sd_card\BOOT.bin`

Copy modified **`f0.c, f1.c f2.c`** and **`f3.c`** to the directory:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\sd_card\edkdsp\a\`
Copy compiled **`f0.dec, f1.dec f2.dec`** and **`f3.dec`** to the directory:
`c:\TS74\TE0720_EdkDSP_2if_te706_ila8k_Release_INSTALL\SDSoC_PFM\2if\SD_release\te01_l\Release\sd_card\`

The content of the Linux-release-target SD card is updated with new MicroBlaze and PicoBlaze6 firmware and stored in the PC.

10. Installation of Arrowhead Framework Support

This chapter describes an installation procedure of Arrowhead client on Zynq 7000 device with support for the Xilinx SDSoC 2017.4 HW accelerators. The Zynq device runs Xilinx PetaLinux 2017.4. kernel with Debian 9.8 Stretch distribution (03.25.2019). The client SW acts as a *Producer* of a service or as a *Consumer* requesting the service from an Arrowhead framework. The base hardware platform for the Zynq device is compiled with Xilinx Vivado 2017.4 tool. The entire installation procedure has been tested on Win 7 Pro and Win 10 PC. To run and test Arrowhead clients, it is required to have running Arrowhead-framework G4.0 light-weight installation running on a RaspberryPi 3B board (RPi3).

HW configuration with simple arrowhead client example

The targeted HW works with one RPi3 board (bottom) and two Zynq boards (above). The RPi3 implements the Arrowhead framework. See [2] for the documentation. The Producer Zynq on the top board hosts C++ provider capable to measure the actual temperature of the Xilinx XC77010-1C device. The Consumer Zynq in the middle hosts C++ consumer capable to ask the Arrowhead framework about the temperature provided as service by the Producer Zynq board. Zynq boards host HW accelerators of Matrix MultiplyAdd (20x20 int32 matrices), delivering approximately 4x shorter execution time in comparison to the optimized SW running on the 650 MHz Arm Cortex A9 processor.

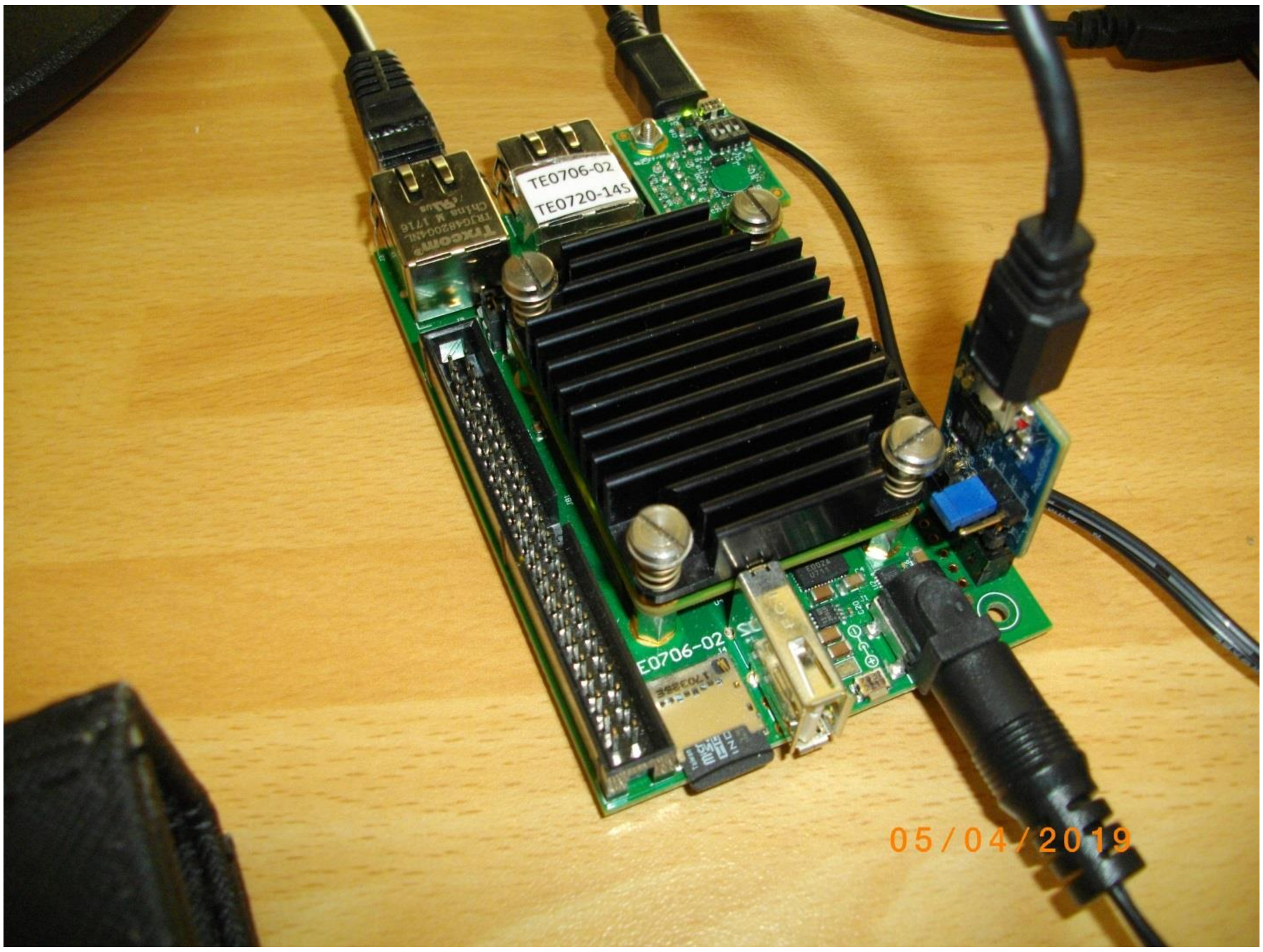


Figure 31: Zynq module (TE0720-14S on TE0706-02 carrier) with Debian an AH 4.0 Client

Installation of arrowhead framework services on RPi3

Testing and running of the Arrowhead C++ clients on Zynq board requires Ethernet access to the Arrowhead framework services. It is recommended to use the precompiled image for the RPi3 board. It includes already installed and configured Arrowhead framework G4.0 lightweight implementation. The image is available as one of results of the work package WP1 of the running ECSEL JU project Productive4.0 https://productive40.eu/.

It is accessible for all consortium project partners from the project ownCloud repository https://productive4-cloud.automotive.oth-aw.de/index.php/login . Files are present in section WP1, task 1.4. Please contact coordinator of the consortium for further information about the access to the Arrowhead-framework G4.0 light-weight installation running on the RPi3 board. After receiving the access to the download, unzip the three downloaded files *Arrowhead-40-raspi.z01*, *Arrowhead-40-raspi.z02* and *Arrowhead-40-raspi.zip* into the final image file *image_180626.img* (size 3.711.959.040 Bytes).

Copy the RPi3 image *image_180626.img* to (at least) 4GB SD card (speed grade 10). You can use the *Win32DiskImager* utility from: https://sourceforge.net/projects/win32diskimager/ .

Connect the RPi3 to USB keyboard, HDMI monitor with inserted SD card. Connect it to Ethernet with the DHCP server. Power ON the board by connecting the 5V power supply via micro USB cable. Power can be provided from the PC via the USB port or, preferably, from the dedicated 5V power supply.

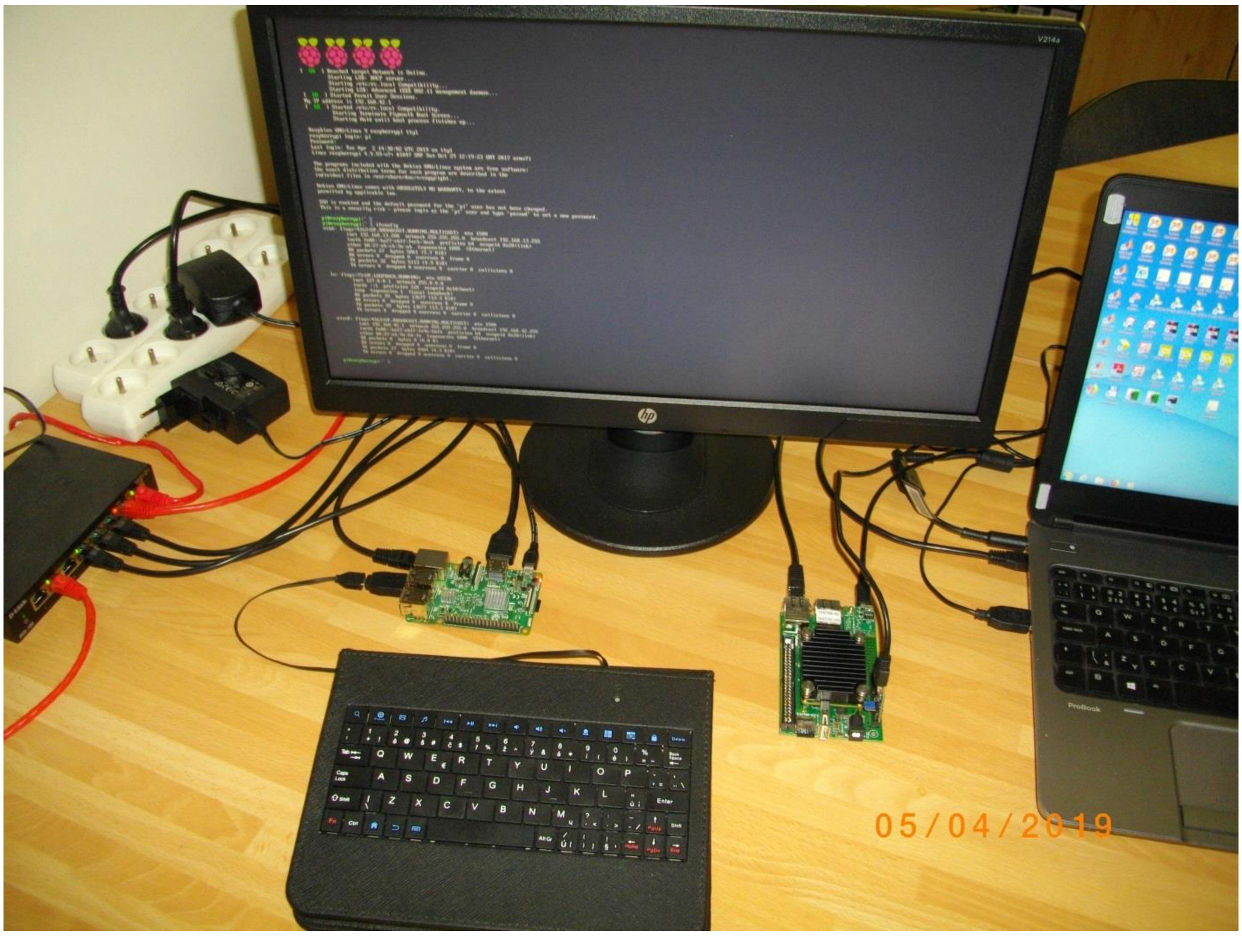


Figure 32: *The RaspberryPi 3 will boot from the SD card image with text output to the HDMI monitor.*

Login as user:

```
pi
```

Password:

```
raspberry
```

Find and write down the assigned Ethernet IP address for IP V4 and IP V6 by typing on the RPi3 keyboard:

```
ifconfig
```

To shutdown properly the RPi3 type on the RPi3 keyboard:

```
sudo halt
```

The OS will shutdown and all possibly open R/W operations to the SD card are closed. Remove temporarily the SD card and disconnect the 5V power to switch OFF the board. Return the SD card to RPi3 slot.

Install Debian immage on SD card for the Zynq board

1. Unzip the preconfigured and precompiled Debian image for the Zynq board from from this evaluation package file: *te0720-debian.zip* to the file *te0720-debian.img* (8GB).
2. Use again the *Win32DiskImager* tool for creation of the image *te0720-debian.img* on the SD card. Use 8GB SD with speed grade 10.
3. Copy to the patrtition visible from Win7 or Win10 (fat32 partition of the immage) card the selected set of files with one of precompiled HW accelerated demos for the SDSoC accelerator and precompiled firmware files and compiler tools for the 8x SIMD EdkDSP accelerator demo as described in the first part of this application note.
4. Insert created SD card to the SD slot of the carrier of the Zynq module.
5. Connect the Zynq board with your Win7 or Win 10 PC via two micro USB cable.
6. Use *putty* or similar terminal client with *speed (baud) 115200bps, data bits 8, stop bits 1, parity none and flow control none*.
7. The actual COM port number associated with your connection can be found in the windows *Device manager*.

Install Arrowhead-f support on zynq

At this stage, the Debian OS present on both Zynq board can be upgraded to become compatible with the Arrowhead framework G4.0 client and provider C++ demo applications.

1. Start Ethernet connected RPi3 board, Zynq board and the Win7 or Win 10 PC.
2. Identify and write down the Ethernet addresses set by the HDCP server. The network has to support access to the external Ethernet to get access to the needed SW repositories.

   In Win7 or Win 10 PC use WinSCP or similar tool to copy the arrowhead installation script *install-arrohead-cli-dep.sh* from this evaluation package to the */root* folder of the Zynq board:

   ```
   /root/install-arrohead-cli-dep.sh
   ```
3. To control the Zynq board, use SSH (preferred) or serial terminal of your Win7 or Win 10 PC. Log in as: user *root* pswd *root*
4. To upgrade the Debian installations on the Zynq SD card image and to install the dependencies required by the Arrowhead framework compatible C++ clients, execute on the Zynq board these commands:

   ```
   cd /root
   chmod ugo+x install-arrohead-cli-dep.sh
   ./install-arrohead-cli-dep.sh
   ```

Install arrowhead-f C++ provider on Zynq

To control the Zynq device, use SSH (preferred) or serial terminal.

1. Get the Arrowhead client source code. The sources include C++ version of the Arrowhead *Provider* and *Client* skeletons.
   ```
   cd /root
   git clone https://github.com/arrowhead-f/client-cpp
   ```
2. Compile Arrowhead *ProviderExample*.
   ```
   cd client-cpp/ProviderExample
   make
   ```
3. Modify the *ProviderExample* configuration file *ApplicationServiceInterface.ini*
   ```
   mcedit ApplicationServiceInterface.ini
   ```
   The configuration file consists of the following items.
   - sr_base_uri – an address of the Arrowhead registration service running in insecure mode, in our case it is the RPi3 IP address with port 8440.
   - sr_base_uri_https – an address of the Arrowhead registration service running in secure mode, in our case it is the RPi3 IP address with port 8441.
   - port – a port number where the *Provider* will be available on, set 8000.
   - address – *Provider* IP address, Zynq IP.
   - Address6 - *Provider* IP address in IPV6

   The *ProviderExample* configuration file example:
   ```
   [Server]
   sr_base_uri="http://10.42.0.141:8440/serviceregistry/"
   sr_base_uri_https="https://10.42.0.141:8441/serviceregistry/"
   port="8000"
   address="10.42.0.103"
   address6="[fe80::483b:e5ff:fe7f:610d]"
   ```
   Safe the file (F2) and exit the editor (F10).
4. Start the *ProviderExample*
   ```
   ./ProviderExample
   ```
   The *ProvidedExample* registers itself in the Arrowhead framework database. On *Consumer* request, it returns an artificial temperature, fixed to value 26 degrees Celsius.

Install arrowhead-f C++ consumer on Zynq

The Arrowhead *ConsumerExample* can be compiled and run on the same Zynq board.

1. Compile Arrowhead *ConsumerExample*.
   ```
   cd /root/client-cpp/ConsumerExample
   make
   ```
2. Configure the *ConsumerExample*. There are **two** configuration files: *OrchestratorInterface.ini* and *consumedServices.json*.
   a. *OrchestratorInterface.ini*
      ```
      mcedit OrchestratorInterface.ini
      ```
      The configuration file consists of the following items.
      - or_base_uri – an address of the Arrowhead orchestrator service running in insecure mode, in our case it is the RPi3 IP address with port 8440.
      - sr_base_uri_https – an address of the Arrowhead orchestrator service running in secure mode, in our case it is the RPi3 IP address with port 8441.
      - port – a port number where the *Consumer* will be available on, set 8002.
      - address – *Consumer* IP address, Zynq IP.
      - address6 - *Consumer* IP address in IPV6

The configuration file example:

```
[Server]
or_base_uri="http://10.42.0.141:8440/orchestrator/orchestration"
or_base_uri_https="https://10.42.0.141:8441/orchestrator/orchestration"
port="8002"
address="10.42.0.103"
address6="[fe80::483b:e5ff:fe7f:610d]"
```

Safe the file (F2) and exit the editor (F10).

b. *consumedServices.json*

```
mcedit consumedServices.json
```

Modify the following items in the file:

- requestForm/requesterSystem/port – Number of the *Consumer* port.
- Modify line

```
"security" : ""
```

- preferredProviders/providerSystem/address – Preferred *Provider* IP address.
- preferredProviders/providerSystem/port – Port number, where the preferred *Provider* listen on.

This configuration file should look like this:

```
{
  "consumerID": "TestconsumerID",
  "requestForm": {
    "requesterSystem": {
      "systemName": "client1",
      "address": "dontcare",
      "port": 8002,
      "authenticationInfo": "null"
    },
    "requestedService": {
      "serviceDefinition": "IndoorTemperature_ProviderExample",
      "interfaces": ["REST-JSON-SENML"],
      "serviceMetadata":{
        "security" : ""
      }
    },
    "orchestrationFlags": {
      "overrideStore" : true,
      "matchmaking" : true,
      "metadataSearch" : false,
      "pingProviders" : false,
      "onlyPreferred" : true,
      "externalServiceRequest" : false
    },
    "preferredProviders": [{
      "providerSystem":{
        "systemName": "SecureTemperatureSensor",
        "address": "10.42.0.103",
        "port":"8000"
      }
    }]
  }
}
```

Save the file (F2) and exit the mcedit editor (F10).

The Debian midnight commander tool can be started from the command line by typing:

```
mc -s
```

Run the *ConsumerExample*

```
./ConsumerExample
```

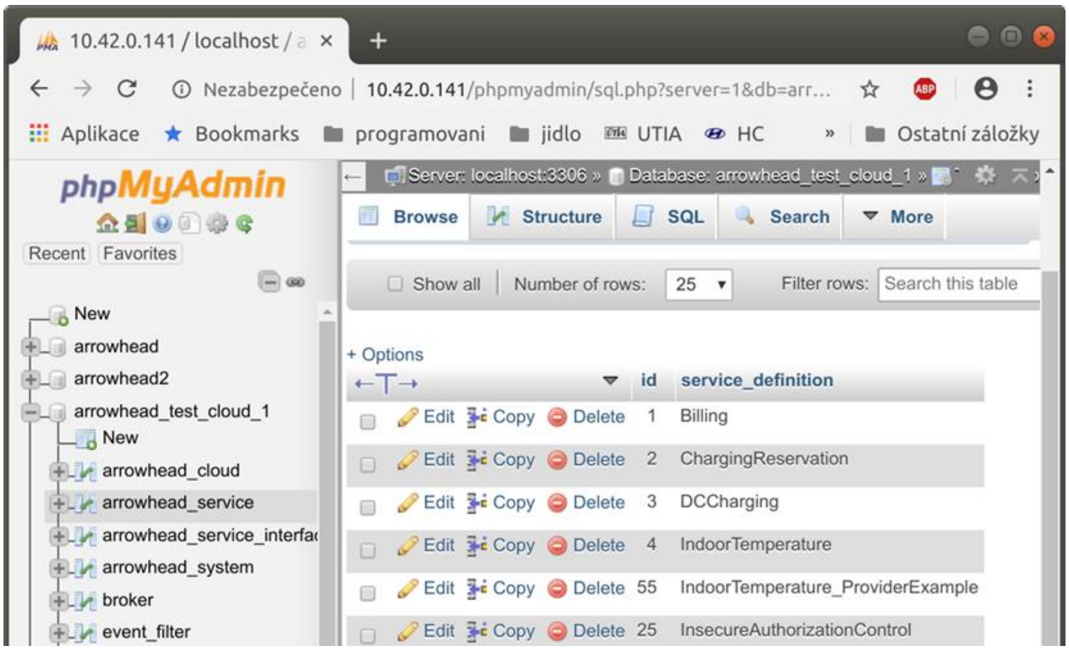
The program should show the following response from the *ProviderExample*:

```
Provider Response:
{"e":[{"n": "this_is_the_sensor_id","v":26.0,"t": "1553675692"}],"bn":
"this_is_the_sensor_id","bu": "Celsius"}
```

The *ConsumerExample* will fail in the first instance. The database of the Arrowhead-f running on the RPi3 has to be configured. The *ProviderExample* and the *ConsumerExample* have to be connected by the operator of the databaze. This is described next.

Modification of arrowhead database

The Arrowhead framework running on RPi3 provides *phpMyAdmin* interface to control its database. To allow the *ConsumerExample* to get the *ProducerExample* service response, follow these steps:



Server: localhost:3306 » Database: arrowhead_test_cloud_1 »

- New
- arrowhead
- arrowhead2
- arrowhead_test_cloud_1
  - New
  - arrowhead_cloud
  - arrowhead_service
  - arrowhead_service_interfac
  - arrowhead_system
  - broker
  - event_filter

	id	service_definition
Edit Copy Delete	1	Billing
Edit Copy Delete	2	ChargingReservation
Edit Copy Delete	3	DCCharging
Edit Copy Delete	4	IndoorTemperature
Edit Copy Delete	55	IndoorTemperature_ProviderExample
Edit Copy Delete	25	InsecureAuthorizationControl

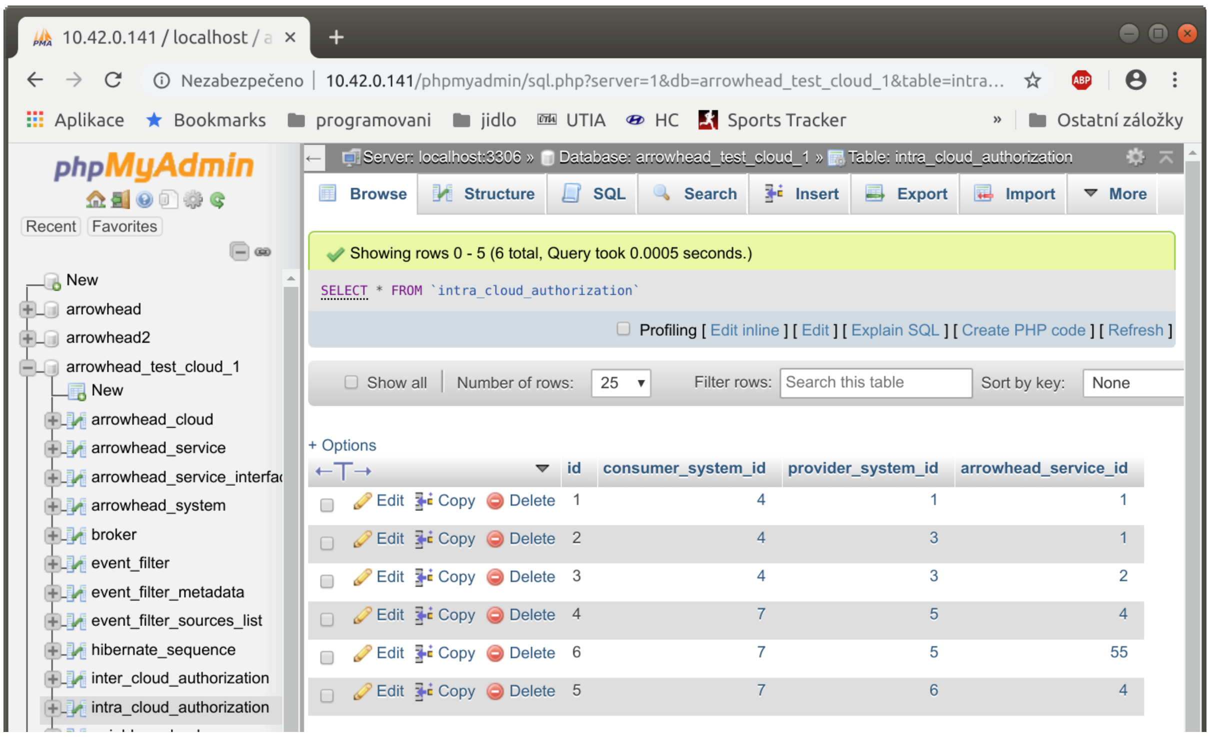
Figure 33: phpMyAdmin interface of the Arrowhead Database

1. On your Win7 or Win 10 PC, start web browser and go to the RPi3 *phpMyAdmin* web page, *http://10.42.0.141/phpmyadmin* (use the IP address of your RPi3).
   User name: *root* password: *root*
2. Get an ID of the *ProducerExample*.
   Select table *arrowhead_test_cloud_1→arrowhead_system*

and locate the line containing the IP address of the Zynq with system_name *SecureTemperatureSensor*.
In our case the ID is 5.

3. Get an ID of the *ConsumerExample*.
   Select table *arrowhead_test_cloud_1→ arrowhead_system*
   Locate the line containing system_name:
   client1.
   In our case it is 7.
4. Get an ID of the *ProducerExample* service.
   Select table *arrowhead_test_cloud_1→ arrowhead_service*
   Locate the line containing service_definition called:
   IndoorTemperature_ProviderExample.
   In our case the ID is 55.
5. In table *service_registry, check* if the *ProviderExample* is linked with its service.
   Link the *ProviderExample*, its service and the *ConsumerExample* together. In table *intra_cloud_authorization,* add a new line containing: *consumer_system_id* 7, *provider_system_id* 5 and *arrowhead_service_id 55*.

The *ConsumerExample* should get the proper response from the *ProviderExample,* now.


10.42.0.141 / localhost / a
Nezabezpečeno | 10.42.0.141/phpmyadmin/sql.php?server=1&db=arrowhead_test_cloud_1&table=intra...
Aplikace Bookmarks programovani jidlo UTIA HC Sports Tracker Ostatní záložky
phpMyAdmin
Recent Favorites
New
arrowhead
arrowhead2
arrowhead_test_cloud_1
New
arrowhead_cloud
arrowhead_service
arrowhead_service_interfac
arrowhead_system
broker
event_filter
event_filter_metadata
event_filter_sources_list
hibernate_sequence
inter_cloud_authorization
intra_cloud_authorization
Server: localhost:3306 » Database: arrowhead_test_cloud_1 » Table: intra_cloud_authorization
Browse Structure SQL Search Insert Export Import More
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all Number of rows: 25 Filter rows: Search this table Sort by key: None
+ Options

Showing rows 0 - 5 (6 total, Query took 0.0005 seconds.)

```
SELECT * FROM `intra_cloud_authorization`
```

	id	consumer_system_id	provider_system_id	arrowhead_service_id
Edit Copy Delete	1	4	1	1
Edit Copy Delete	2	4	3	1
Edit Copy Delete	3	4	3	2
Edit Copy Delete	4	7	5	4
Edit Copy Delete	6	7	5	55
Edit Copy Delete	5	7	6	4

Figure 34: The intra_cloud_authorization table of the Arrowhead Database

Test the Zynq consumer and producer

The *ProducerExample* server is running on the “Producer” Zynq board, now.
Execute the *ConsumerExample* client example on the “Consumer” Zynq board.

```
./ConsumerExample
```

The *ConsumerExample* client example program should show the modelled constant temperature response (26.0) from the *ProviderExample* and exit.

```
Provider Response:
{"e":[{"n": "this_is_the_sensor_id","v":26.0,"t": "1553675692"}],"bn":
"this_is_the_sensor_id","bu": "Celsius"}
```

This concludes the complete demo of Producer and Consumer on two Zynq boards omplemented as C++ SW code compatible with the Arrowhead framework G4.0 lite-installation on the RPi3 board.

Producer service and Consumer client can run on a single Zynqbeery board or two different Zynq boards. The configuration files and the configuration of the Arrowhead framework database described in Chapter 6 - Chapter 10 provides setup for single Zynq board.

Change of the setup for two Zynq boards involves only modification of the corresponding Ethernet addresses assigned by the DHCP server.

The HW accelerated matrix multiplication demo can be executed on both Zynq boards by executing:

```
/boot/te06_1.elf
```

See the HW acceleration measured by the number of Arm A9 clock cycles.

Producer with real temperature measurement on zynq

Real temperature of the Xilinx chip of the Zynq board can be measured by modified *ProviderExample.cpp* code. This code measures the real temperature of the chip:

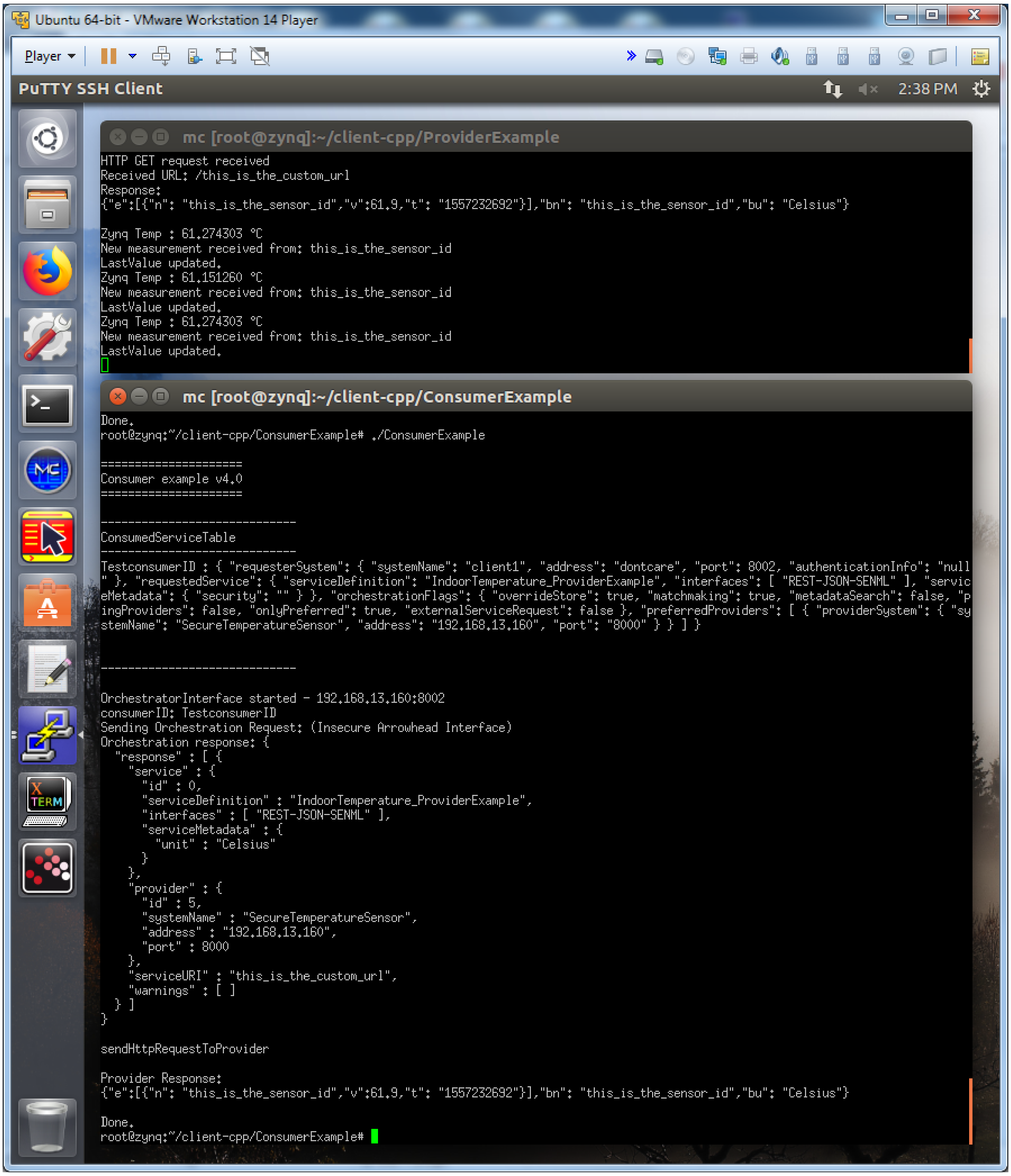
```
#pragma warning(disable:4996)

#include "SensorHandler.h"
#include <sstream>
#include <string>
#include <stdio.h>
#include <thread>
#include <list>
#include <time.h>
#include <iomanip>

#ifdef __linux__
    #include <unistd.h>
#elif _WIN32
    #include <windows.h>
#endif

#define TEMP_RAW_FILE     "/sys/bus/iio/devices/iio:device0/in_temp0_ps_temp_raw"
#define TEMP_OFFSET_FILE  "/sys/bus/iio/devices/iio:device0/in_temp0_ps_temp_offset"
#define TEMP_SCALE_FILE   "/sys/bus/iio/devices/iio:device0/in_temp0_ps_temp_scale"

const std::string version = "4.1";

bool bSecureProviderInterface = false; //Enables HTTPS interface on the application service (with token enabled)
bool bSecureArrowheadInterface = false; //Enables HTTPS interface towards ServiceRegistry AH module

inline void parseArguments(int argc, char* argv[]){
    for(int i=1; i<argc; ++i){
        if(strstr("--secureArrowheadInterface", argv[i]))
            bSecureArrowheadInterface = true;
        else if(strstr("--secureProviderInterface", argv[i]))
            bSecureProviderInterface = true;
    }
}

int main(int argc, char* argv[]){

              printf("\n=============================\nProvider Example - v%s\n=============================\n",
version.c_str());

    parseArguments(argc, argv);

              SensorHandler oSensorHandler;

//SenML format
//todo:
//generate own measured value into "measuredValue"
//"value" should be periodically updated
//"sLinuxEpoch" should be periodically updated

    std::string measuredValue; //JSON - SENML format
    time_t linuxEpochTime = std::time(0);
    std::string sLinuxEpoch = std::to_string((uint64_t)linuxEpochTime);

   FILE *f_t_raw, *f_t_off, *f_t_scale;

   if ( (f_t_raw = fopen(TEMP_RAW_FILE, "r")) == NULL ) {
      printf("Cannot open file %s \n", TEMP_RAW_FILE);
      return -1;
   }

   if ( (f_t_off = fopen(TEMP_OFFSET_FILE, "r")) == NULL ) {
      printf("Cannot open file %s \n", TEMP_OFFSET_FILE);
      return -1;
   }

   if ( (f_t_scale = fopen(TEMP_SCALE_FILE, "r")) == NULL ) {
      printf("Cannot open file %s \n", TEMP_SCALE_FILE);
      return -1;
   }
   printf("OK\n");

   int   t_raw;
   int   t_off;
   float t_scale;

   fscanf(f_t_raw, "%d", &t_raw);
   fscanf(f_t_off, "%d", &t_off);
   fscanf(f_t_scale, "%f", &t_scale);
```

```
    if ( fclose(f_t_raw) == EOF ) {
        printf("Cannot close file %s \n", TEMP_RAW_FILE);
        return -1;
    }
    printf("OK\n");

    if ( fclose(f_t_off) == EOF ) {
        printf("Cannot close file %s \n", TEMP_OFFSET_FILE);
        return -1;
    }

    if ( fclose(f_t_scale) == EOF ) {
        printf("Cannot close file %s \n", TEMP_SCALE_FILE);
        return -1;
    }

//      double value = 26.0;

    // (raw + offset) * scale ... in miligegree Celsius
    float value = ((float)(t_raw + t_off) * t_scale) / 1000.00f;

//convert double to string
     std::ostringstream streamObj;
     streamObj << std::fixed;
     streamObj << std::setprecision(1);
     streamObj << value;
     std::string sValue = streamObj.str();

     measuredValue =
         "{"
             "\"e\":[{"
                 "\"n\": \"this_is_the_sensor_id\","
                 "\"v\":" + sValue +","
                 "\"t\": \"" + sLinuxEpoch + "\""
                 "}],"
             "\"bn\": \"this_is_the_sensor_id\","
             "\"bu\": \"Celsius\""
         "}";

//do not modify below this

    oSensorHandler.processProvider(measuredValue, bSecureProviderInterface, bSecureArrowheadInterface);

    while (true) {

        linuxEpochTime = std::time(0);
        sLinuxEpoch = std::to_string((uint64_t)linuxEpochTime);

//         if (value < 30.0) value += 0.1;
//         else              value = 26.0;
```

```
    if ( (f_t_raw = fopen(TEMP_RAW_FILE, "r")) == NULL ) {
       printf("Cannot open file %s \n", TEMP_RAW_FILE);
       return -1;
    }
    fscanf(f_t_raw, "%d", &t_raw);
    if ( fclose(f_t_raw) == EOF ) {
       printf("Cannot close file %s \n", TEMP_RAW_FILE);
       return -1;
    }
    value = ((float)(t_raw + t_off) * t_scale) / 1000.00f;
    printf("Zynq Temp : %f °C\n", value);
    streamObj.clear();
    streamObj.str("");
    streamObj << std::fixed;
    streamObj << std::setprecision(1);
    streamObj << value;
    sValue = streamObj.str();
    measuredValue =
         "{"
             "\"e\":[{"
                 "\"n\": \"this_is_the_sensor_id\","
                 "\"v\":" + sValue +","
                 "\"t\": \"" + sLinuxEpoch + "\""
                 "}],"
             "\"bn\": \"this_is_the_sensor_id\","
             "\"bu\": \"Celsius\""
         "}";
    oSensorHandler.processProvider(measuredValue, bSecureProviderInterface, bSecureArrowheadInterface);
    #ifdef __linux__
       sleep(1);
    #elif _WIN32
       Sleep(1000);
    #endif
           }
  printf("Close file %s ... ", TEMP_RAW_FILE);
  if ( fclose(f_t_raw) == EOF ) {
    printf("FAILED\n");
    return -1;
  }
  printf("OK\n");
           return 0;
}
```

Figure 35: Modifications of ProviderExample.cpp C to measure temperature of the Zynq chip

All other files of the *ProviderExample* project remain identical.


Ubuntu 64-bit - VMware Workstation 14 Player
Player
PuTTY SSH Client
2:38 PM
mc [root@zynq]:~/client-cpp/ProviderExample

```
HTTP GET request received
Received URL: /this_is_the_custom_url
Response:
{"e":[{"n": "this_is_the_sensor_id","v":61.9,"t": "1557232692"}],"bn": "this_is_the_sensor_id","bu": "Celsius"}

Zynq Temp : 61.274303 °C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 61.151260 °C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 61.274303 °C
New measurement received from: this_is_the_sensor_id
LastValue updated.
```

```
Done.
root@zynq:~/client-cpp/ConsumerExample# ./ConsumerExample

====================
Consumer example v4.0
====================

------------------------------
ConsumedServiceTable
------------------------------
TestconsumerID : { "requesterSystem": { "systemName": "client1", "address": "dontcare", "port": 8002, "authenticationInfo": "null
" }, "requestedService": { "serviceDefinition": "IndoorTemperature_ProviderExample", "interfaces": [ "REST-JSON-SENML" ], "servic
eMetadata": { "security": "" } }, "orchestrationFlags": { "overrideStore": true, "matchmaking": true, "metadataSearch": false, "p
ingProviders": false, "onlyPreferred": true, "externalServiceRequest": false }, "preferredProviders": [ { "providerSystem": { "sy
stemName": "SecureTemperatureSensor", "address": "192.168.13.160", "port": "8000" } } ] }


------------------------------

OrchestratorInterface started - 192.168.13.160:8002
consumerID: TestconsumerID
Sending Orchestration Request: (Insecure Arrowhead Interface)
Orchestration response: {
  "response" : [ {
    "service" : {
      "id" : 0,
      "serviceDefinition" : "IndoorTemperature_ProviderExample",
      "interfaces" : [ "REST-JSON-SENML" ],
      "serviceMetadata" : {
        "unit" : "Celsius"
      }
    },
    "provider" : {
      "id" : 5,
      "systemName" : "SecureTemperatureSensor",
      "address" : "192.168.13.160",
      "port" : 8000
    },
    "serviceURI" : "this_is_the_custom_url",
    "warnings" : [ ]
  } ]
}

sendHttpRequestToProvider

Provider Response:
{"e":[{"n": "this_is_the_sensor_id","v":61.9,"t": "1557232692"}],"bn": "this_is_the_sensor_id","bu": "Celsius"}

Done.
root@zynq:~/client-cpp/ConsumerExample#
```

Figure 36: ProviderExample and ConsumerExample clients on Zynq.

Recompile the *ProviderExample* project by *make.* Test it on the Zynq board.

Modified ProviderExample is registered to the Arrowhead database. For debug purposes it also prints the actual temperature of the chip to its console. See
Figure 36. Modified ConsumerExample connects via the Arrowhead framework. It receives and displayes the actual chip temperature.

In
Figure 36, two instances of the Ubuntu PuTTY SSH Client are used. Both clients are connected to the Zynq module to test the Arrowhead framework.

Conclusions

Support for the Arrowhead framework (installed as G4.0 lite on the RPi3 board) has been demonstrated. See
Figure 36.

The Zynq device remains compatible with the SDSoC 2017.4 system level compiler of HW accelerators.

It also remains compatible with the 8xSIMD EdkDSP single precision floating point HW accelerator running in the programmable part of the Zynq device. The firmware for the run-time reconfigurable 8xSIMD EdkDSP IP can be compiled from source code C by compiler application running on the same Zynq device. Compiled firmware code can be downloaded by Debian user-space C-coded applications during the run-time without the need to reset or reboot the Zynq board. This application note also describes the clock-cycle-accurate debug support for the EdkDSP accelerator based on the Xilinx ChipScope logic analyser instantiated in the programmable logic of the Zynq device.

The documented Arrowhead compatible Zynq Clients bring to the framework the additional support for the acceleration of part of local computation in the programmable logic HW based on the SDSoC compiler.

11. Data Lines on TE0703-05 and TE0706-02 Carrier Boards

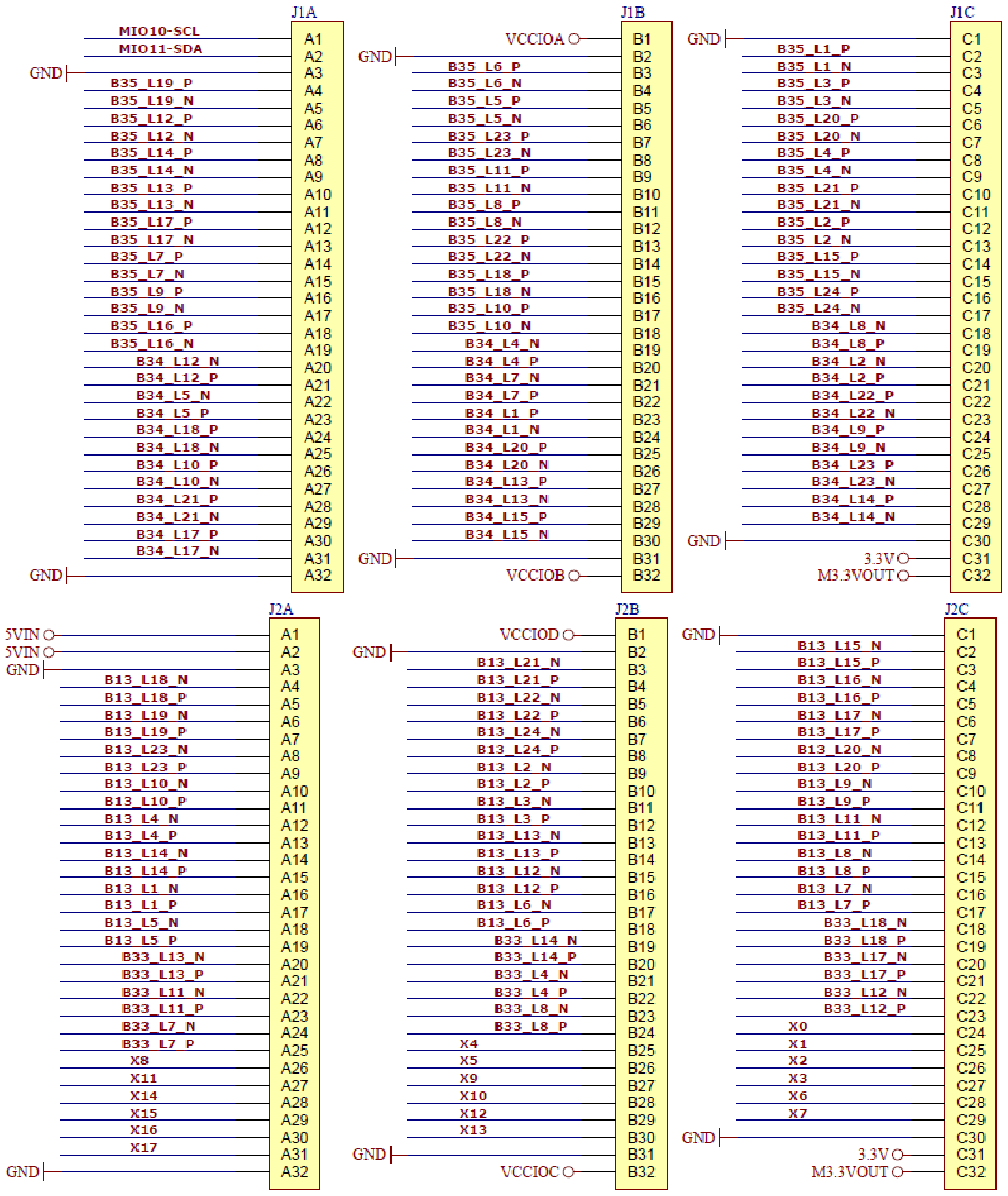


Figure 37: Connection of PCBs data lines to connectors on TE0703-05 carrier board

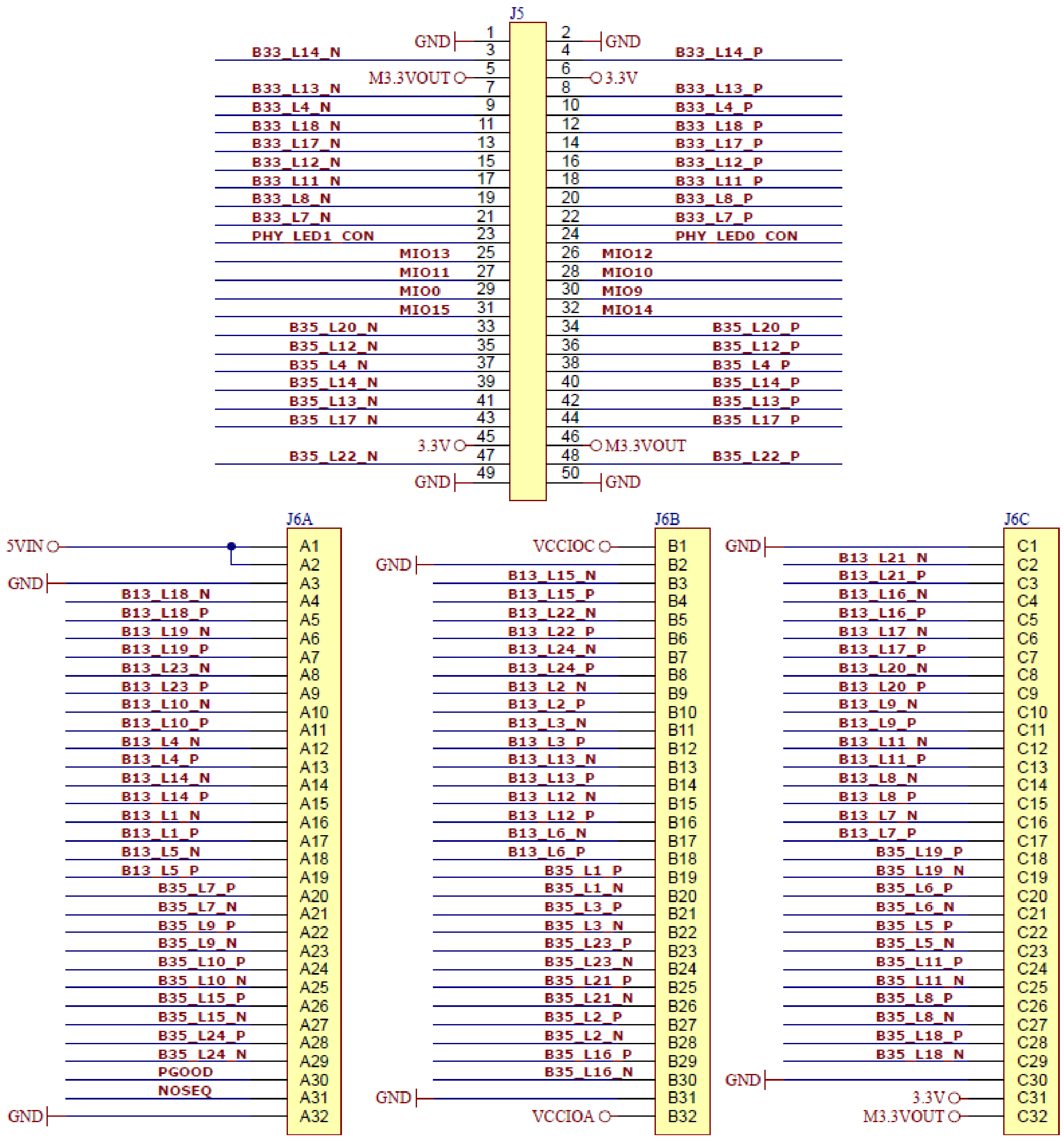


Figure 38: Connection of PCBs data lines to connectors on TE0706-02 carrier board

Figure 37 describes connection of PCBs data lines to the connectors on the TE0703-05 carrier board and *Figure 38* for the TE0703-05 carrier board. Table 13 describes the common connections of Zynq pins to TE0703-05 and TE0706-02 PCB data lines. Users of the development package can use these data for creation of own user constrains and extend the Vivado 2017.4.1 HW projects generated by the SDSoC 2017.4.1 design environment.

Table 13: Common Connections of Zynq pins to TE0703-05 and TE0706-02 PCB Data Lines

Zynq	Board 3.3V	Zynq	Board 3.3V	Zynq	Board 3.3V	Zynq	Board 1.8V
C22	B35_L16_N	AA22	B33_L7_P	AA12	B13_L7_P	J18	B34_L7_P
D22	B35_L16_P	AB22	B33_L7_N	AB12	B13_L7_N	K18	B34_L7_N
G22	B35_L24_N	AA21	B33_L8_P	AA11	B13_L8_P	J16	B34_L2_P
H22	B35_L24_P	AB21	B33_L8_N	AB11	B13_L8_N	J17	B34_L2_N
B22	B35_L18_N	Y19	B33_L11_P	AA9	B13_L11_P	L17	B34_L4_P
B21	B35_L18_P	AA19	B33_L11_N	AA8	B13_L11_N	M17	B34_L4_N
A22	B35_L15_N	Y18	B33_L12_P	AB10	B13_L9_P	N17	B34_L5_P
A21	B35_L15_P	AA18	B33_L12_N	AB9	B13_L9_N	N18	B34_L5_N
G21	B35_L22_N	V15	B33_VREF	T4	B13_L20_P	L18	B34_L12_P
G20	B35_L22_P	AA17	B33_L17_P	U4	B13_L20_N	L19	B34_L12_N
D21	B35_L17_N	AB17	B33_L17_N	AB7	B13_L17_P	J21	B34_L8_P
E21	B35_L17_P	AA16	B33_L18_P	AB6	B13_L17_N	J22	B34_L8_N
B20	B35_L13_N	AB16	B33_L18_N	AB5	B13_L16_P	J20	B34_L9_P
B19	B35_L13_P	W20	B33_L4_P	AB4	B13_L16_N	K21	B34_L9_N
C20	B35_L14_N	W21	B33_L4_N	Y4	B13_L18_P	R19	B34_L22_P
D20	B35_L14_P	W17	B33_L13_P	AA4	B13_L18_N	T19	B34_L22_N
G16	B35_L4_N	W18	B33_L13_N	AB2	B13_L15_P	J15	B34_L1_P
G15	B35_L4_P	W16	B33_L14_P	AB1	B13_L15_N	K15	B34_L1_N
C19	B35_L12_N	Y16	B33_L14_N	V5	B13_L21_P	P20	B34_L18_P
D18	B35_L12_P			V4	B13_L21_N	P21	B34_L18_N
F19	B35_L20_N			U7	B13_IO25	P17	B34_L20_P
G19	B35_L20_P			U12	B13_L5_P	P18	B34_L20_N
A19	B35_L10_N			U11	B13_L5_N	L21	B34_L10_P
A18	B35_L10_P			U10	B13_L6_P	L22	B34_L10_N
A17	B35_L9_N			U9	B13_L6_N	M19	B34_L13_P
A16	B35_L9_P			V10	B13_L1_P	M20	B34_L13_N
B15	B35_L7_N			V9	B13_L1_N	T16	B34_L21_P
C15	B35_L7_P			Y9	B13_L12_P	T17	B34_L21_N
D17	B35_L2_N			Y8	B13_L12_N	M21	B34_L15_P
D16	B35_L2_P			AA7	B13_L14_P	M22	B34_L15_N
B17	B35_L8_N			AA6	B13_L14_N	R20	B34_L17_P
B16	B35_L8_P			Y6	B13_L13_P	R21	B34_L17_N
E20	B35_L21_N			Y5	B13_L13_N	R18	B34_L23_P
E19	B35_L21_P			V12	B13_L4_P	T18	B34_L23_N
C18	B35_L11_N			W12	B13_L4_N	M16	B34_VREF
C17	B35_L11_P			W11	B13_L3_P	N19	B34_L14_P
F22	B35_L23_N			W10	B13_L3_N	N20	B34_L14_N
F21	B35_L23_P			Y11	B13_L10_P		
E18	B35_L5_N			Y10	B13_L10_N		
F18	B35_L5_P			V8	B13_L2_P		
D15	B35_L3_N			W8	B13_L2_N		
E15	B35_L3_P			V7	B13_L23_P		
F17	B35_L6_N			W7	B13_L23_N		
G17	B35_L6_P			W6	B13_L24_P		
E16	B35_L1_N			W5	B13_L24_N		
F16	B35_L1_P			R6	B13_L19_P		
H20	B35_L19_N			T6	B13_L19_N		
H19	B35_L19_P			U6	B13_L22_P		
				U5	B13_L22_N		
				R7	B13_IO0		

12. References

[1] **TE0720-03-2IF**; Part: XC7Z020-2CLG484I; 1 GByte DDR; Industrial Grade (Tj = -40°C to +100°C)
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/REV03/Documents/TRM-TE0720-03.pdf
https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/4x5/TE0720/REV03/Documents/SCH-TE0720-03-2IF.PDF

TE0720-03-1QF; Part: XA7Z020-1CLG484Q; 1 GByte DDR; Automotive Grade (Tj = -40°C to +125°C)
https://shop.trenz-electronic.de/en/TE0720-03-1QF-Xilinx-Zynq-module-ind.-temp.-range-with-Automotive-XA7Z020-1CLG484Q
https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/4x5/TE0720/REV03/Documents/SCH-TE0720-03-1QF.PDF

TE0720-03-214S-1C; Part: XC7Z014S-1CLG484C; 1 GByte DDR; Industrial Grade (Tj = 0°C to +85°C)
https://shop.trenz-electronic.de/en/TE0720-03-14S-1C-SoC-Module-with-Xilinx-Zynq-Z-7014S-Single-core-1-GByte-DDR3
https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/4x5/TE0720/REV03/Documents/SCH-TE0720-03-14S-1C.PDF

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

[3] **TE0706-02** 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
https://wiki.trenz-electronic.de/display/PD/TE0706+TRM

TE0703-05 Carrier board for Trenz Electronic Modules with 4 x 5 cm Form factor
https://shop.trenz-electronic.de/en/TE0703-05-TE0703-Carrier-board-for-Trenz-Electronic-modules-with-4-x-5-cm-form-factor?c=261
https://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/4x5/4x5_Carriers/TE0703/REV05/Documents/SCH-TE0703-05.PDF
https://wiki.trenz-electronic.de/display/PD/TE0703+TRM

[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 - 2017.4.1.
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2015-4.html

[7] SDSoC - 2017.4.1 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

13. Base Release Evaluation Package

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

Deliverables:
The base, release evaluation package [8] includes evaluation bitstreams with single (8xSIMD) EdkDSP IP working in parallel with selected HW-accelerated SDSoC algorithms on the Trenz Electronic TE0720-03-2IF, TE0720-03-1QF and TE0720-03-14S-1C module [1] located on the Trenz Electronic TE0706-02 or TE0703-05 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, release 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 will run again after the reset (TE0706-02: Reset push button S2; TE0703-05: Reset push button S1).

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 the Debian image:
te0720-debian.zip Zip archive with te0720-debian.img image for installation on the Zynq SD card.

The base, release 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 TE0720-03-2IF, TE0720-03-1QF and TE0720-03-14S-1C module [1] on Trenz Electronic TE0706-02 or TE0703-05 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.

14. Extended Debug Evaluation Package for PRODUCTIVE 4.0 partners

The extended, debug evaluation package includes **MicroBlaze and PicoBlaze6 C code and precompiled bitstreams of HW projects** for the Trenz Electronic TE0720-03-2IF, TE0720-03-1QF and TE0720-03-14S-1C module [1] located on the Trenz Electronic TE0706-02 or TE0703-05 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, debug evaluation package with MicroBlaze and PicoBlaze6 C code and precompiled bitstream of HW projects for the Trenz Electronic TE0720-03-2IF, TE0720-03-1QF and TE0720-03-14S-1C module [1] located on the Trenz Electronic TE0706-02 or TE0703-05 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, debug 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, debug 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, debug 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 will run again after the reset (TE0706-02: Reset push button S2; TE0703-05: Reset push button S1).

The extended, debug 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, debug evaluation package for PRODUCTIVE 4.0 partners includes the Debian image:
te0720-debian.zip Zip archive with te0720-debian.img image for installation on the Zynq SD card.

The extended, debug 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 TE0720-03-2IF, TE0720-03-1QF and TE0720-03-14S-1C module [1] on Trenz Electronic TE0706-02 or TE0703-05 carrier board [3].

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

libwal.a	SDK 2017.4.1 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

This disclaimer is not a license and does not grant any rights to the materials distributed herewith. Except as otherwise provided in a valid license issued to you by UTIA AV CR v.v.i., and to the maximum extent permitted by applicable law:

(1) THIS APPLICATION NOTE AND RELATED MATERIALS LISTED IN THIS PACKAGE CONTENT ARE MADE AVAILABLE "AS IS" AND WITH ALL FAULTS, AND UTIA AV CR V.V.I. HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
(2) UTIA AV CR v.v.i. shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under or in connection with these materials, including for any direct, or any indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or UTIA AV CR v.v.i. had been advised of the possibility of the same.

Critical Applications:
UTIA AV CR v.v.i. products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, Class III medical devices, nuclear facilities, applications related to the deployment of airbags, or any other applications that could lead to death, personal injury, or severe property or environmental damage (individually and collectively, "Critical Applications"). Customer assumes the sole risk and liability of any use of UTIA AV CR v.v.i. products in Critical Applications, subject only to applicable laws and regulations governing limitations on product liability.