Application Note

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

DTRiMC tool for TE0726-03M board

Jiři Kadlec, Raissa Likhonina
kadlec@utia.cas.cz

Revision history

Rev.	Date	Author	Description
0	6.04.2020	J. Kadlec	Initial draft
1	8.07.2021	J. Kadlec	DTRiMC tool for TE0726-03M
2			

Table of Contents

Table of Figures

Acknowledgement

This work has been partially supported from project FitOptiVis, project number ECSEL 783162 and the corresponding Czech NFA (MSMT) institutional support project 8A18013.

1 DTRiMC tool for TE0726-03M board

This application note describes evaluation package for the Design Time Resource integration of Model Composer DTRiMC tool. See Figure 1. It serves for integration of 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerator for Zynq device on TE0726-03M board [1].

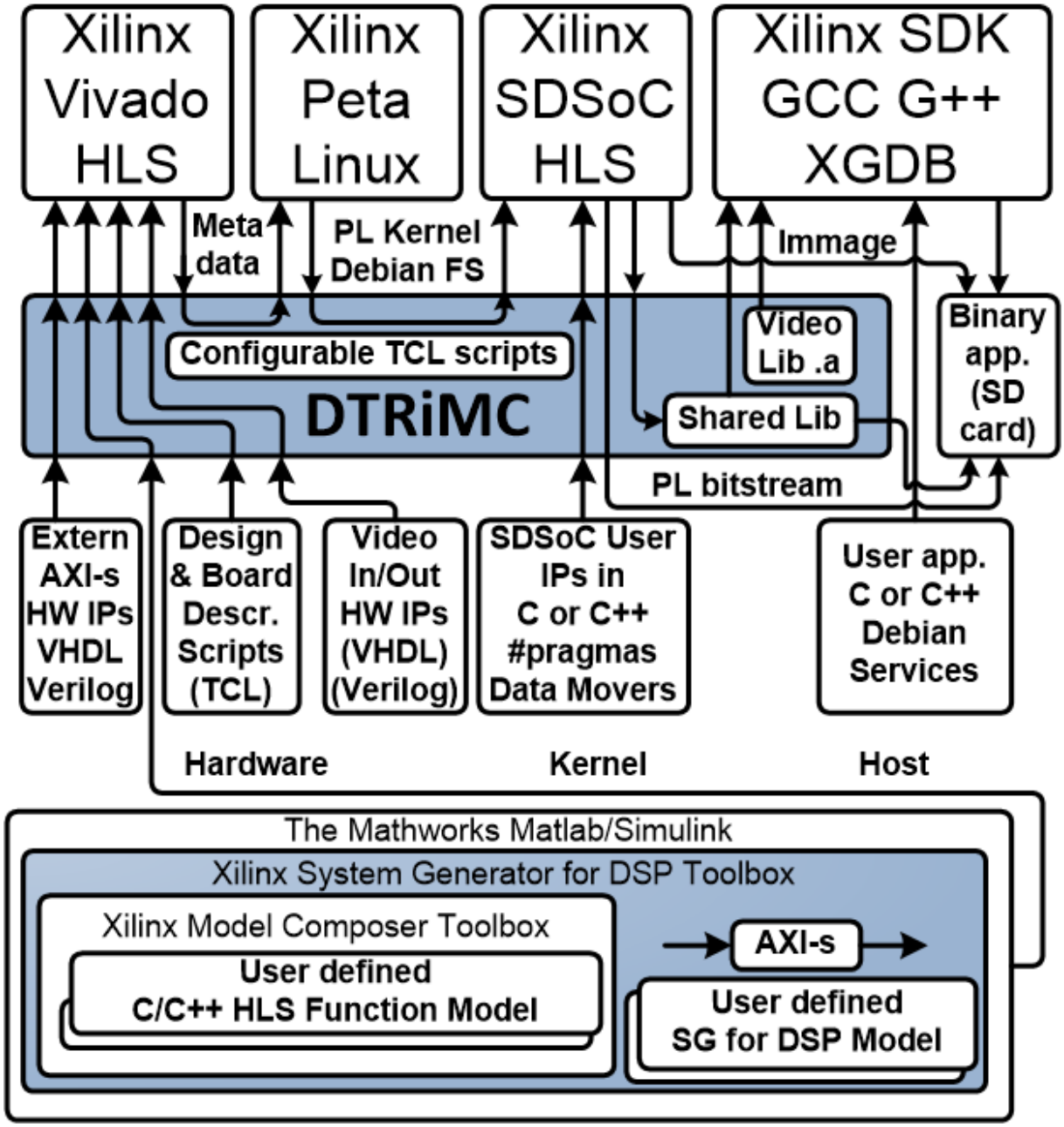


Figure 1: Design Time Resource integration of Model Composer DTRiMC tool.

This application note describes released UTIA support for the evaluation version of SIMD FP03x8 floating point run time reconfigurable accelerator for the ZynqBerry board.

ZynqBerry TE0726-03M [1] works with Xilinx XC07010-1C device with the dual core ARM A9 32 bit, 512 MB of DDR3 memory and some programmable logic area on single 28nm chip.

The ZynqBerry PCB has RaspberryPi 2 form factor. The ZynqBerry board is designed and manufactured by company Trenz Electronic [1].

SW developer can program „main“applications without SDSoC 2018.2 compiler license with the g++ compiler in Xilinx SDK on Win 10 PC. „make“ can be used directly on A9 processor.

The accelerator and HW data communication is represented for the SW developer as shared C++ library with simple SW API, identical for several alternative HW data movers.

2 8xSIMD FP03x8 floating point accelerator for TE0726M board

The FP03x8 HW accelerator serves for run-time reprogrammable 8xSIMD single precision floating point computations. The internal structure of FP03x8 accelerator is described in Figure 3.

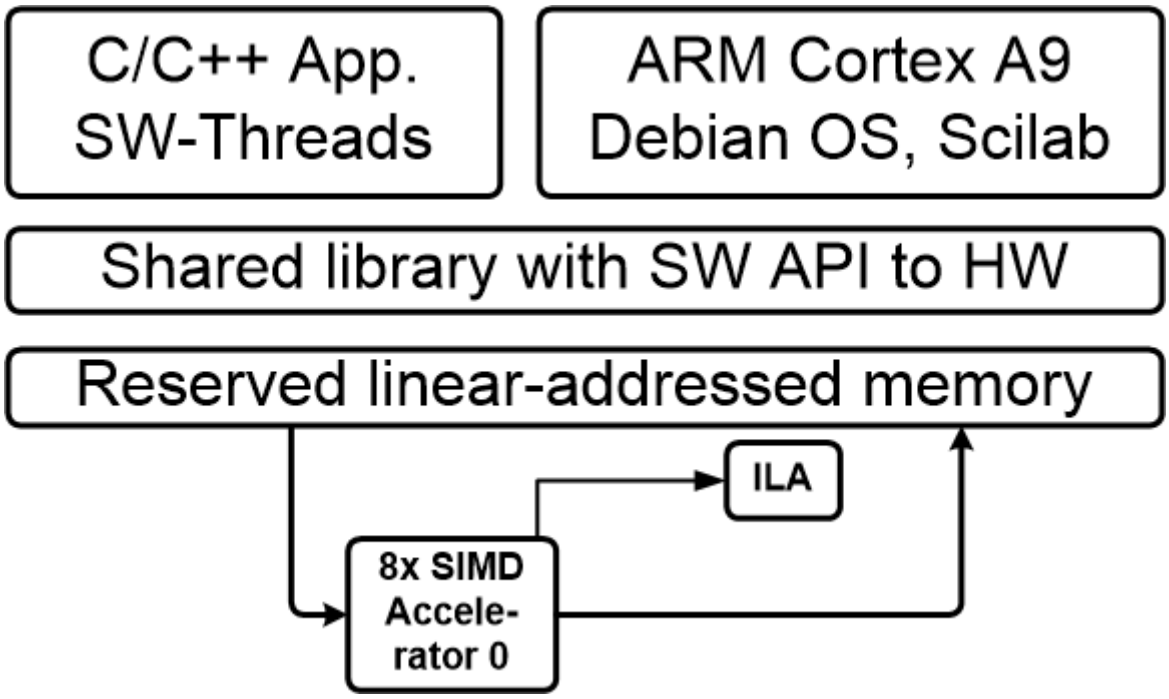


Figure 2: FP03x8 accelerator in TE0726-03M board

Input:

- Program firmware data received via AXI stream interface from ARM processor.
- Configuration Write registers for scalar control received via AXI-lite interface from ARM processor.
- Floating point single precision data received via AXI stream interface from ARM processor.

Output:

- Registers indicating end of program accessible to ARM processor via AXI-lite.
- Floating point single precision result data accessible via AXI stream interface for the ARM processor.

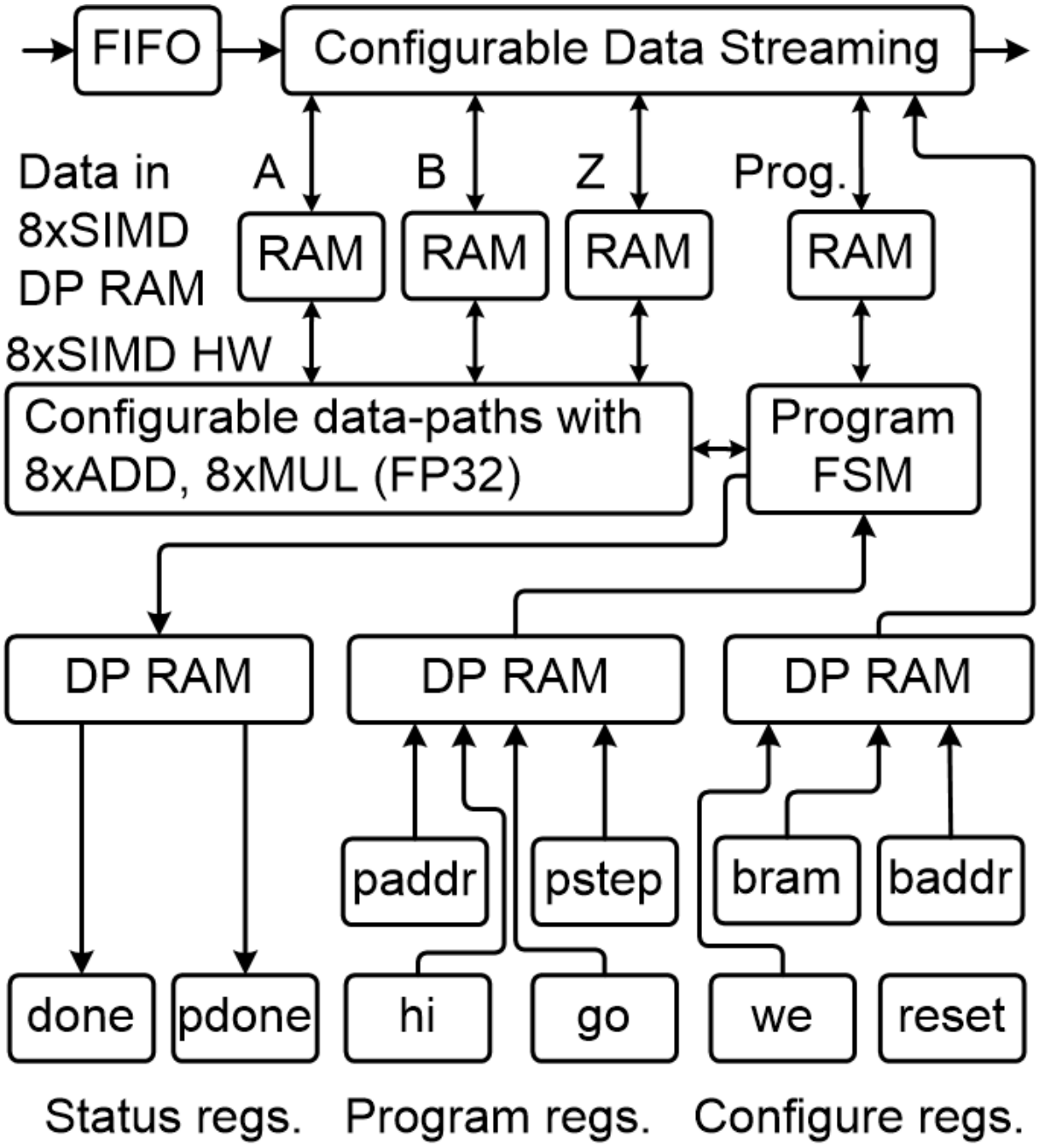


Figure 3: FP03x8 accelerator for the TE0726M board (FP32 division is not supported)

Connectivity:

- AXI stream data/program input from ARM to HW accelerator with input FIFO 1024x32. The side channel indicates the last transferred word sent to the component via the HW data mover from ARM processor.
- AXI stream data/program output from HW accelerator to ARM. The output AXI stream side channel indicates the last transferred word sent from the accelerator to the ARM processor.
- AXI-lite input/output configuration registers.

Design presented in this evaluation package contains one FP03x8 accelerator in the programmable logic part of the device. See Figure 2 and Figure 3.

The HW data movers supporting the data communication are represented for the SW developer in a shared C++ Debian OS library with simple SW API. The SW API provides identical interface for several alternatives of HW data movers. The evaluation package includes one pre-compiled 8xSIMD FP32 accelerator with HW license enabling only restricted number of operations. If these licensed operations are all used, user has to reset complete system. This will enable to use the licensed count of operations again.

Please contact UTIA (kadlec@utia.cas.cz) if you are interested in licensing of 8xSIMD accelerator HW IP without this restriction.

Accelerator Interfaces

Type of interface:	Device:	Clock:
• Data streaming I/O: AXI-S 32 bit	xc7z010clg225-1	115 MHz
• Firmware program VLIW 128 bit	xc7z010clg225-1	115 MHz
• Configuration I/O: AXI-lite 32 bit	xc7z010clg225-1	100 MHz
• ARM A9 system clock	xc7z010clg225-1	666 MHz

SIMD A 32 bit	Block 64 bit	SIMD B 32 bit	Block 64 bit	SIMD Z 32 bit	Block 64 bit	VLIW Prog	Block 64 bit
A1	0	B1	4	Z1	8	P1	12
A2		B2		Z2		P2	
A3	1	B3	5	Z3	9	P3	13
A4		B4		Z4		P4	
A5	2	B5	6	Z5	10		
A6		B6		Z6			
A7	3	B7	7	Z7	11		
A8		B8		Z8			

Figure 4: Internal block rams of accelerator.

Memory of the Accelerator in the programmable logic part of the device

- 12 dual-ported 1024x64 bit BRAMs Blocks (0 .. 11) are used as:
  - 24 Data RAMs organised as 1024x32 bit blocks: A1..A8, B1..B8 and Z1..Z8.
- 2 dual-ported 512x64 bit BRAMs Blocks (12, 13) are used as:
  - 4 Program RAMs organised as 512x32 bit blocks: P1..P3

AXI-lite Registers

Name:	**Data:**	**Description:**
reset	1 bit:	"1" Reset AXI lite Registers; "0" NOP
we	16 bit:	Write from stream to block(s) (bit 0 .. 13)
baddr	10 bit:	Stream will rd/wr from addr=baddr
bram	5 bit:	Read from Block 0 .. 13 to Stream; 16 for: Move-data-through
paddr	9 bit:	Program start address
pstep	9 bit:	Program stop address
go	1 bit:	"1" go from paddr to pstep; "0" NOP
hi	12 bit:	SubBank prog. mod: 00zz00bb00aa (bits)
done	8 bit:	Read only. "0" => Instruction runs
pdone	1 bit:	Read only. "0" => Program runs

Parameters of stream data interfaces from/to ARM DDR memory

- Maximal supported stream data size is 2048 x 32 bit
- Data streaming can have variable size:
  - Min: 2 x 32 bit
  - Max 2048 x 32 bit
- Mode of operation (same for Data and for Program):
  - **Write to a block**: It is defined by **we** (from address defined in **baddr)**
  - **Broadcast Write**: It is defined by setting more bits in **we** (from address defined in **baddr**)
  - **Read from block**: It is defined by setting **bram** (from address defined in **baddr)**
  - **Write or Broadcast Write and Read in parallel**: It is defined by setting more bits in **we** and by setting **bram** (from address defined in **baddr**)
  - **Send data through the Accelerator**: It is defined by setting: **we** = 0 and by setting **bram** =16;

Design-time support

These data streaming HW data movers are supported (can be generated by the SDSoC):

- Zero Copy HW data mover without DMA
- DMA HW data mover with DMA
- SG DMA HW data mover with SG DMA and interrupts

The design time support is based on the Xilinx SDSoC 2018.2 system level compiler.

Run-time support

- Data can be written to and/or read from the accelerator by user ARM app.
- Firmware can be written to and/or read from the accelerator user ARM app.
- Computation & data streaming can be performed in parallel.

Versions of accelerator: FP03x8_capabilities capabilities = 10, 20, 30 or 40

SIMD OP	code (dec)	8xSIMD Floating Point Operation Description
VVER	0	Return capabilities of the accelerator and status of license
VZ2A	1	8xSIMD vector copy $a_m[i] <= z_m[j]$; m=1..8
VB2A	2	8xSIMD vector copy $a_m[i] <= b_m[j]$; m=1..8
VZ2B	3	8xSIMD vector copy $b_m[i] <= z_m[j]$; m=1..8
VA2B	4	8xSIMD vector copy $b_m[i] <= a_m[j]$; m=1..8
Auto-increments:		*Example: for (n=0;n<=CNT;n++){i=i+B_INC; j=j+A_INC;}*
VADD	5	8xSIMD vector add $z_m[i] <= a_m[j] + b_m[k]$; m=1..8
VADD_BZ2A	6	8xSIMD vector add $a_m[i] <= b_m[j] + z_m[k]$; m=1..8
VADD_AZ2B	7	8xSIMD vector add $b_m[i] <= a_m[j] + z_m[k]$; m=1..8
Auto-increments:		*Example: for (n=0;n<=CNT;n++){i=i+B_INC; j=j+A_INC; k=k+Z_INC;}*
VSUB	8	8xSIMD vector sub $z_m[i] <= a_m[j] - b_m[k]$; m=1..8
VSUB_BZ2A	9	8xSIMD vector sub $a_m[i] <= b_m[j] - z_m[k]$; m=1..8
VSUB_AZ2B	10	8xSIMD vector sub $b_m[i] <= a_m[j] - z_m[k]$; m=1..8
Auto-increments:		*Example: for (n=0;n<=CNT;n++){i=i+B_INC; j=j+A_INC; k=k+Z_INC;}*
VMULT	11	8xSIMD vector mult $z_m[i] <= a_m[j] * b_m[k]$; m=1..8
VMULT_BZ2A	12	8xSIMD vector mult $a_m[i] <= b_m[j] * z_m[k]$; m=1..8
VMULT_AZ2B	13	8xSIMD vector mult $b_m[i] <= a_m[j] * z_m[k]$; m=1..8
Auto-increments:		*Example: for (n=0;n<=CNT;n++){i=i+B_INC; j=j+A_INC; k=k+Z_INC;}*

Figure 5: Floating point functions present in all accelerators {10 or 20 or 30 or 40}.

SIMD OP	code (dec)	8xSIMD Floating Point Operation Description
VPROD FP01, FP03: 30,40	14	8xSIMD vector products. $z_m[i] <= a_m'[j..j+nn]*b_m[k..k+nn]$; m=1..8; nn range 0..255
VMAC FP01, FP03: 20,30,40	15	8xSIMD vector MACs. $z_m[i..i+nn] <= z_m[i..i+nn] + a_m[j..j+nn] * b_m[k..k+nn]$; m=1..8; nn range 0..10
VMSUBAC FP01, FP03: 20,30,40	16	8xSIMD vector MSUBACs. $z_m[i..i+nn] <= z_m[i..i+nn] - a_m[j..j+nn] * b_m[k..k+nn]$; m=1..8; nn range 0..10
LONG_VPROD FP01, FP03: 40	17	Single long vector product . $z_m[i] <= ((a_1'[j..j+nn]*b_1[k..k+nn]+a_2'[j..j+nn]*b_2[k..k+nn]) + (a_3'[j..j+nn]*b_3[k..k+nn]+a_4'[j..j+nn]*b_4[k..k+nn])) + ((a_5'[j..j+nn]*b_5[k..k+nn]+a_6'[j..j+nn]*b_6[k..k+nn]) + (a_7'[j..j+nn]*b_7[k..k+nn]+a_8'[j..j+nn]*b_8[k..k+nn]));$ m=1..8; nn range 0..255
VDIV FP03: 10,20,30,40 FP01: not supported	20	8xSIMD vector Division. $z_m[i] <= a_m[j] / b_m[k]$; m=1..8
Auto-increments:		*Example: for(n=0;n<=CNT;n++){i=i+Z_INC; j=j+A_INC; k=k+B_INC;}*

Figure 6: Specific functions present only in some accelerator versions.

The 8xSIMD accelerator for TE0726 board does not support the pipelined vector division operation VDIV. This is due to the limited size of PL logic of the xc7z010clg225-1 device. Application programs can detect it by execution of VVER instruction. The VVER instruction returns status word which indicates to SW that the VDIV operation is not supported in HW accelerator on the TE0726 board.

3 Programming of 8xSIMD FP03x8 floating point accelerators

Host ARM application SW can form the VLIW program instructions in DDR4 memory as two 64bit words. Components of the low 64 bit word are marked by light green background. Components of the high 64 bit word are marked by light blue. See Figure 7. Host ARM application can form a sequence of such VLIW program instructions in DDR4 memory in run-time. It can write them to accelerator program memories in PL logic.

FP01, FP03	Size	VLIW: hi lo	Description
[not_used]	*[8bit]*	*8 bit [63..56]*	*Not used by FP01 or FP03*
[not_used]	*[8bit]*	*8 bit [55..48]*	*Not used by FP01 or FP03*
[0,Z_MEM_SECTION]	[0,2bit]	8 bit [47..40]	Z_MEM SECTION (0..3)
[CNT]	[8bit]	8 bit [39..32]	Number of 8xSIMD steps (0 .. 255)
[Z_INC]	[8bit]	8 bit [31..24]	Auto increment of Z address (0 .. 255)
[Z_MEM_SADDR]	[8bit]	8 bit [23..16]	Set Z address after auto-increment overflow
[Z_MEM_ADDR]	[8bit]	8 bit [15..08]	Initial Z address
[B_INC]	[8bit]	8 bit [07..00]	Auto increment of B address (0 .. 255)
[OP]	[8bit]	8 bit [63..56]	8xSIMD vector operation
[0, B_MEM_SECTION]	[0,2bit]	8 bit [55..48]	B_MEM SECTION (0..3)
[0, A_MEM_SECTION]	[0,2bit]	8 bit [47..40]	A_MEM SECTION (0..3)
[B_MEM_SADDR]	[8bit]	8 bit [39..32]	Set B address after auto-increment overflow
[B_MEM_ADDR]	[8bit]	8 bit [31..24]	Initial B address
[A_INC]	[8bit]	8 bit [23..16]	Auto increment of A address (0 .. 255)
[A_MEM_SADDR]	[8bit]	8 bit [15..08]	Set A address after auto-increment overflow
[A_MEM_ADDR]	[8bit]	8 bit [07..00]	Initial A address

Figure 7: Structure of the 128bit wide VLIW program instruction.

Multiple sequences of VLIW instructions (with length of the sequence from 1 (min length) up to 512 (max length)) can be autonomously executed by the accelerator (see Figure 3).

User defines start address in **paddr** AXI-lite register and end address in **pstep** AXI-lite register.

User requests execution of the sequence of VLIW operations by setting the single bit AXI-lite register **go** = 1. The accelerator executes the VLIV sequence from **paddr** to **pstep**.

State of the execution can be tested by the ARM host SW application by reading of the AXI Read only register **pdone.** If **pdone**==0, the sequence of VLIW instructions is being executed.
If **pdone**==1, the sequence of VLIW instructions has completed.

Finally the ARM host SW application must set the single bit AXI-lite register back to **go** = 0.

The ARM host SW application can also copy data/program to/from the accelerator while the sequence of VLIW instructions is being autonomously executed on current internal data with the current sequence of VLIW instrutructions present in program memory of the accelerator.

This parallel copy data/program to/from the accelerator (while accelerators executes its sequence of VLIW instructions) requires to avoid race-condition caused by parallel writing to the same memory address both by accelerator and by parallel copy of data defined by the user in the same time instance.

This has to be avoided by the user SW application. Host SW can safely write only to accelerator data which are not used for writing by the currently executed sequence of VLIW instructions.

The sequence of VLIW instructions can be also reduced up to a single VLIW instruction. The **paddr** and **pstep** registers are set to an identical program address in such case.

The internal structure of the Zynq 7000 SoC can be seen in Figure 2 and Figure 3.

4 C++ evaluation project

The PL part of the device contains one evaluation version of the 8xSIMD run-time-reprogrammable single-precision-floating-point HW accelerator FP03x8. Released evaluation HW platforms are exported for linking with the ARM A9 host programs. Platforms have these properties:

- Floating point matrix multiplication with 1x 8xSIMD HW accelerator
  Directory (C++): **fp01x8_v26x1_sw**
  SW C++ project: **fp01x8_v26x1_sw**
  Shared C++ library: **./Debug/sd_card/libfp01x8_v26x1_hw.so**
  Shared C++ library: **./Release/sd_card/libfp01x8_v26x1_hw.so**

The precompiled shared Debian Stretch 9.8 OS library for the SDK 2018.2 C++ SW flow (g++ compiler) provides interface to FP03x8 HW accelerators with zero copy data movers.

ARM host SW project **fp01x8_v26x1_sw** demonstrates HW acceleration of single precision floating point matrix by matrix multiplications.

HW accelerator performance is compared with optimized (-O3) ARM host SW implementation of floating point single precision matrix multiplication.

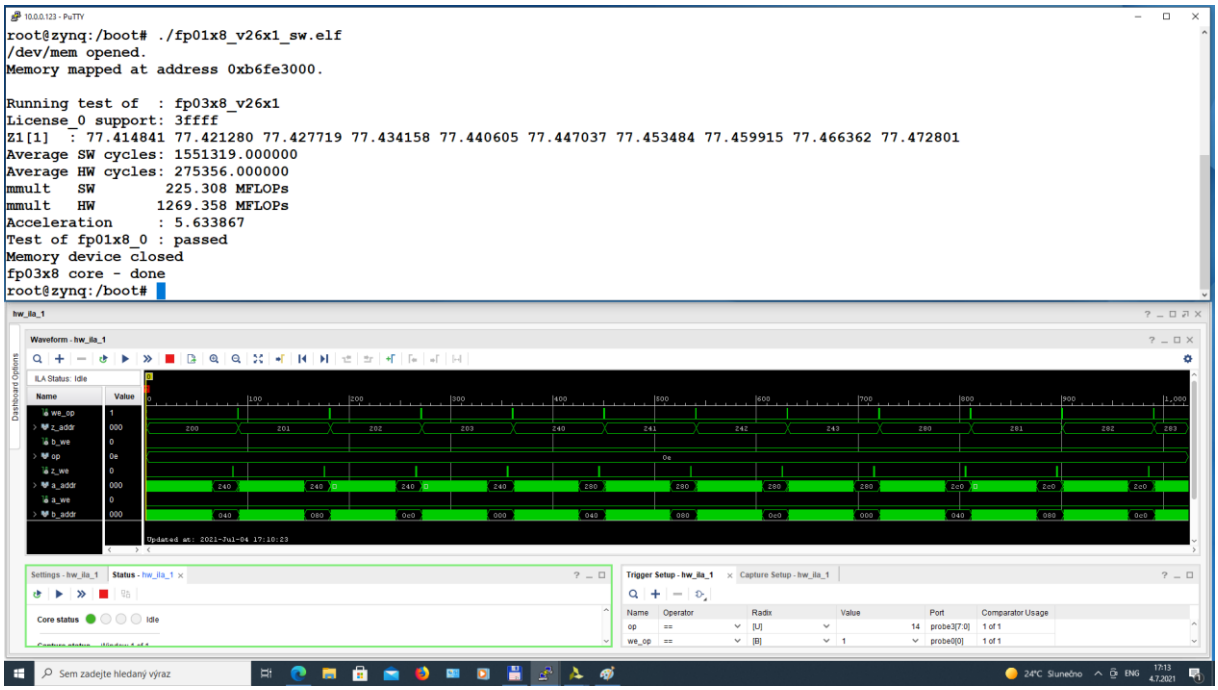


Figure 8: Performance results of fp01x8_v26x1_sw application.

Tested	**Function**
mamtul()	FP32 matrix multiplication Z[64,64] = A[64,64] * B[64,64]

Comparison of matrix multiplication performance:

System	**Function**	**MFLOPs**
TE0726M	HW accelerated matmul() on 1x 8xSIMD, 1 thread	1269
	SW matmul(); Scilab MEX style, 1 thread.	225
I7 PC 3.0 GHz	SW Ubuntu, SciLab C MEX style, 1 thread.	1933

Performance result is listed in Figure 8. It is terminal output from the fp01x8_v26x1_sw.elf application running on A9 processor with one 8xSIMD HW accelerator and ILA monitor of HW operations. Complete running system is presented by Figure 9.

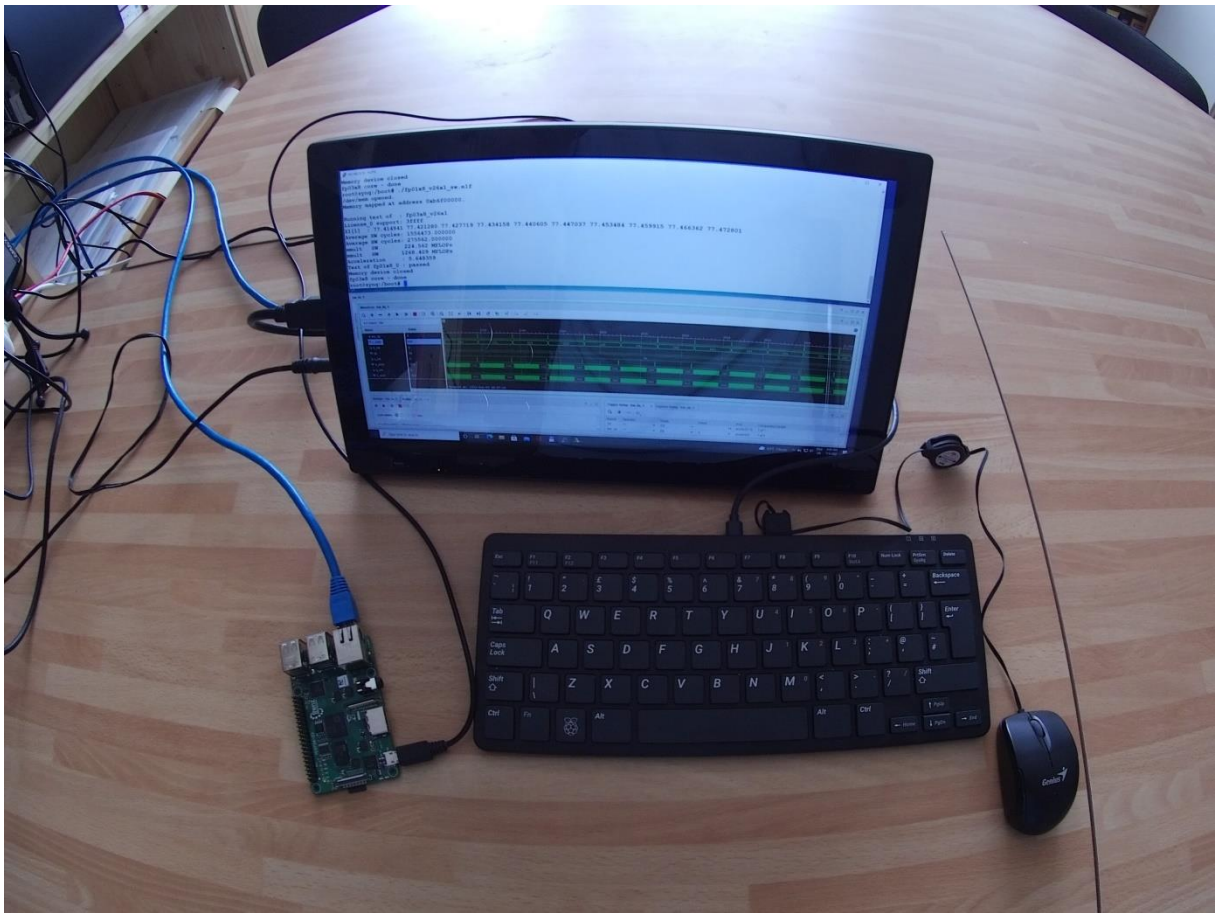


Figure 9: System is running fp01x8_v26x1_sw application.

5 Power consumption

Power consumption is measured on input power line 5V. All power supply is derived from this single power source.

Power consumption	**Power [W]**
Debian OS system is running with all HW interfaced by the library libfp01x8_v26x1_hw.so is present in the device. Remote desktop with user terminal and mousepad editor open. No user application is running.	2,8
Linux system is running with all HW interfaced by the library libfp01x8_v26x1_hw.so is present in the device. SW application **fp01x8_v26x1_sw.elf** is running. It performs HW accelerated matrix multiplication on one 8xSIMD HW accelerator.	3,3

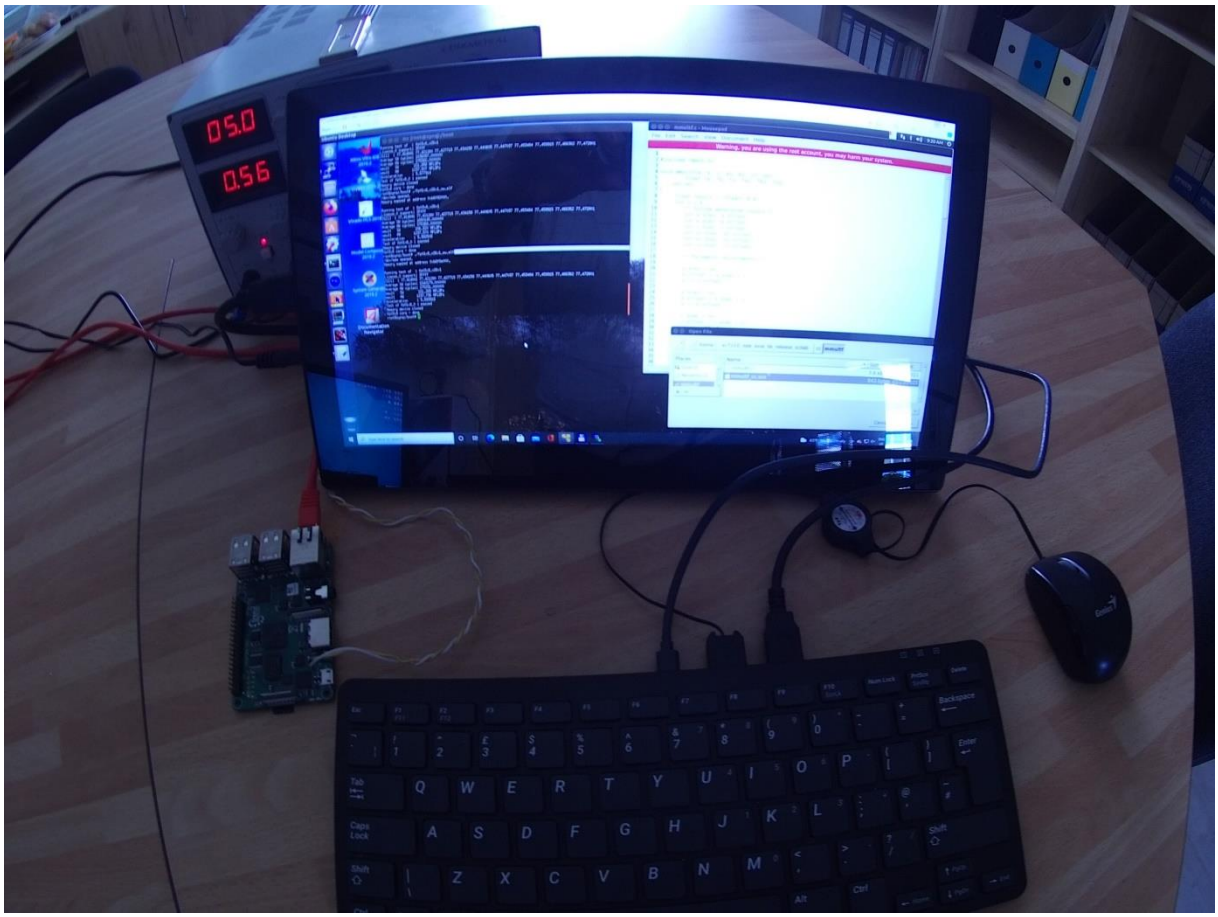


Figure 10: System is running with remote X11 desktop, Mousepad,fp01x8_v26x1_sw app.

6 ILA – In-circuit Logic Analyzer

System created by the Design Time Resource integration of Model Composer DTRiMC tool can include HW IP of the Vivado-Lab tool 2018.2 ILA – In-circuit Logic Analyzer.

It is connected to HW 8xSIMD accelerator 0. See Figure 2. The start of ILA capturing can be triggered by specific logic combination of input values defined by user. ILA monitor displays values of instructions, addresses and we signals with the 115 MHz clock used by the accelerator. It is configured to sample 1024 data. See Figure 11.

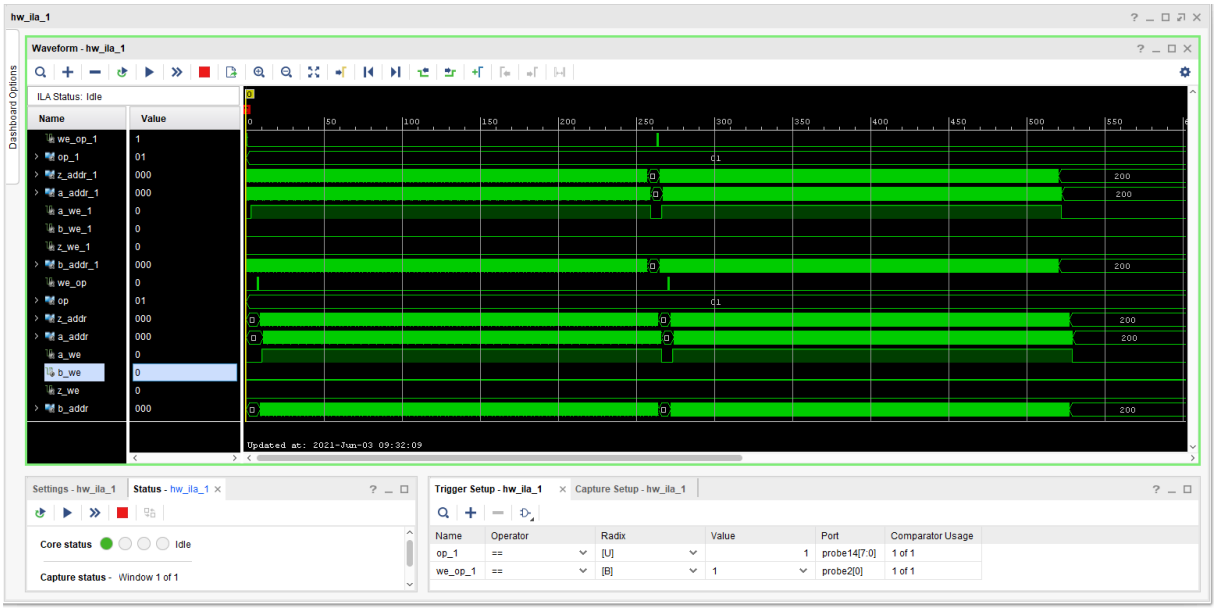


Figure 11: Captured accelerator VLIW instruction vz2a.

Figure 11 presents example of captured test of vz2a VLIW operation. It corresponds to ARM host SW project vz2af0. It performs copy of 512 FP32 data from eight Z memories of accelerators to eight A memories of the accelerator. Copy is executed as a sequence of two VLIW instructions, each performing copy of 256 FP data. This is visible in Figure 11.

In ILA, we can zoom to see the details of captured data. See Figure 12. The we_op_1 == 1 and op_1 == 1 is the trigger condition for the ILA set by the user. The we_op_1 can be seen in the first line of ILA. The address bus related to Z [z_addr_1] starts to increment, followed by address buss related to A [a_addr_1]. The signal [z_we_1] is set to 1 to write the data from A to Z in all eight 8xSIMD memories in parallel.

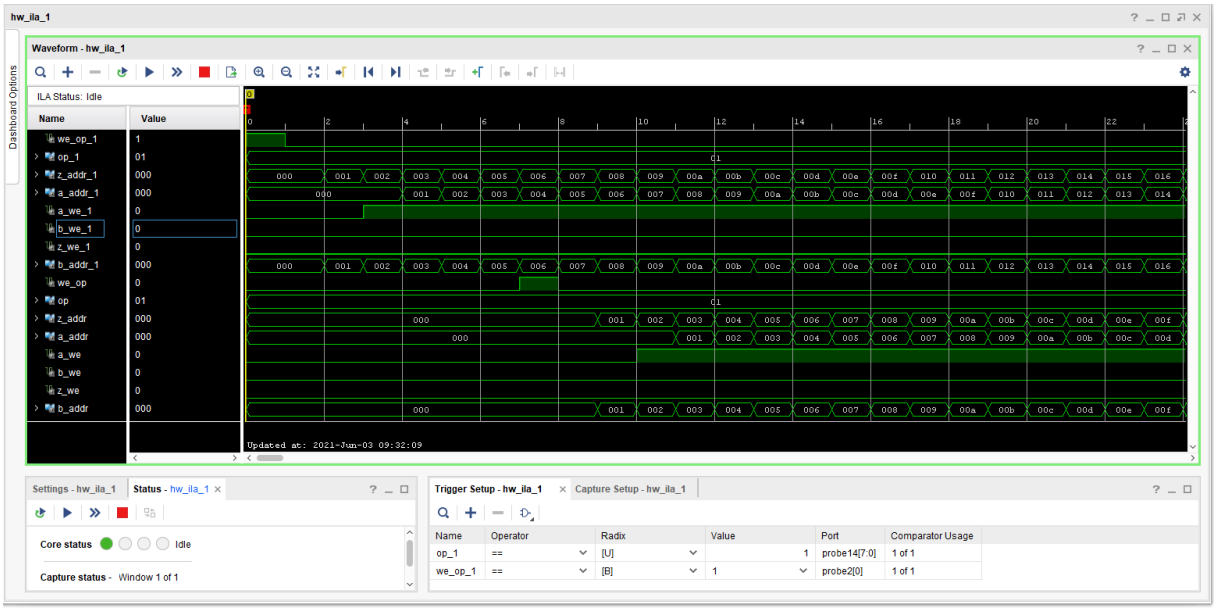


Figure 12: Instruction vz2a detail.

The instantiated ILA helps mainly in analysis and debug of more complex 8xSIMD HW accelerator sequences (like the matrix multiplication example). See Figure 8.

7 License

This evaluation package of the Design Time Resource integration of Model Composer DTRiMC tool includes precompiled system with evaluation version of the accelerator:

- **FP03x8** with **capabilities = 40.** Capabilities are described in Figure 5 and Figure 6**.**

The license for the evaluation version of the accelerator enables execution of certain large number of floating point operations before it expires. If this happens, the board has to be switched off and switched on again to restart the evaluation license again.

The commercial version of accelerators is available in UTIA. UTIA offers license on commercial base. Contract with UTIA is required. For information about details of the commercial license contact Jiri Kadlec kadlec@utia.cas.cz.

8 ARM SW API for Streaming of Data

Serial Streaming SW API
API for single accelerator and for single serial chain of multiple chained accelerators
void data2hw_wrapper(unsigned *src, unsigned len); //1
void capture_wrapper(unsigned *storage, unsigned len); //2

Example:

```
data2hw_wrapper((unsigned*)src_P1_P2, len);      //1
capture_wrapper((unsigned*)dest_P1_P2, len);     //2
…
sds_wait(1);
sds_wait(2);
```

9 Performance

Acceleration of single precision floating point Matrix by Matrix multiplication has been prepared as an application example to evaluate the performance of the released evaluation versions of accelerators. It performs: C[64,64] = A[64,64] * B[64,64].

A single instance of FP03x8 accelerator on ZynqBerry board accelerates single precision floating point Matrix by Matrix multiplication by **5.6x** in comparison to ARM A9 SW implementation.

10 References

[1] "ZynqBerry" Module with Xilinx Zynq-7010 in Raspberry Pi Form Factor https://shop.trenz-electronic.de/en/TE0726-03M-ZynqBerry-Module-with-Xilinx-Zynq-7010-in-Raspberry-Pi-Form-Factor?c=350

11 APPENDIX - Confidence test

This is basic confidence test of the evaluation package.

Unzip evaluation package to Win 10 directory of your choice. We will use:
c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10

Precompiled HW and SW projects are located in directory:
c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw

Compressed SD card image with ARM Debian OS is located in directory:
c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release_sdcard

INSTALLATION OF TOOLS

- Install Xilinx SDK 2018.2 on Win 10 PC 64 bit.
- Install Xilinx Lab Tools 2018.2 on Win 10 PC 64 bit.
- Install Win32DiskImager for writing of image to 16 GB SD card, Class (10).
- Install Putty (for USB based serial console and Ethernet based serial console).
- Unzip ARM Debian OS disk image on Win 10 PC and use the Win32DiskImager to write the disk image (16 GB) from the PC to the 16 GB SD card (Class 10).

Before test on the ZynqBerry board, you have to write to the on-board FLASH the correct BOOT.BIN file with the bitstream. It is done by performing these steps:

- Remove SD card from the TE0726M board.
- Connect the TE0726M board to PC by the USB serial terminal cable.
- Copy the BOOT.BIN file from

  c:\home\work\TS82fp01x8_TE0726\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Release\sd_card\BOOT.BIN
  to
  c:\home\work\TS82fp01x8_TE0726\xc7z10_deb_eval_ila\zsys\prebuilt\boot_images\m\NA\BOOT.BIN

- Change directory to
  c:\home\work\TS82fp01x8_TE0726\xc7z10_deb_eval_ila\zsys

  Execute this script in Win 10 terminal:

  program_flash_binfile.cmd

  This script will write content of the **BOOT.BIN** file to the ZynqBerry board flash.
- Power-off the TE0726M board by removing the USB cable from PC.

HW SETUP

- Insert the SD card with Debian OS disk image to the TE0726M board.
- Connect PC and TE0726M board to Ethernet.
- Connect USB serial terminal cable to TE0726M to PC. This will power-on the board.

TEST

- TE0726M board will start to boot Debian OS. The boot process starts by reading data from the BOOT.BIN present in the internal flash. Only the second stage of the boot process is performed from the SD card.
- In Win 10 PC, open Putty terminal. Set it to:
  (115200 bps, 8 data bits, stop bit 1, parity none, flow control off)
- Use Putty terminal to login as user: **`root`** password: **`root`**
- Change directory to **`/boot`**
- Export path to the shared library. Type in the Putty Debian OS terminal:

  ```
  export LD_LIBRARY_PATH=/boot
  ```

- Start application code by typing in the Putty Debian OS terminal:

  ```
  ./fp01x8_v26x1_sw.elf
  ```

RESULT

- The application will compute single precision floating point matrix multiplication as:
  - SW on host ARM A9 processor
  - SW on host ARM A9 processor HW accelerated by the 8xSIMD accelerator.
- Results of ARM and HW accelerated computations are compared to be identical and MFLOPs performance is displayed. See Figure 8 and Figure 9.

Compilation and debug of projects from source code

The evaluation package includes SW projects for Xilinx SDK 2018.2 tool running on Win10.

These projects can be recompiled for ARM and executed on Zynq with or without debugging support. Open SDK 2018.2 tool, in this working directory:

`c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\`

Projects in this directory link to this shared library:
`c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Release\sd_card\libfp01x8_v26x1_hw.so`
or
`c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Debug\sd_card\libfp01x8_v26x1_hw.so`

Projects have two configurations:

- **Debug** for debugging with –O0 flag with debug information symbols included.
- **Release** for maximal performance with -O3 flag and without debug symbols.

You can modify and re-compile the SW code in the Xilinx SDK 2018.2 tool on Win 10 PC..

DEBUG of SW application from Xilinx SDK 2018.2

The application can be executed or debugged from the SDK 2018.2 tool. SDK debugger needs environment information about the location of the actual shared library on the board.

Before start of Debug, copy complete content of this Debug directory:

```
c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_rel
ease\fp01x8_v26x1_sw\Debug\sd_card\*.*
```

to the top directory of the SD card visible in the Win 10 PC:

```
*.*
```

Insert the SD card to the board and power on the board by connecting USB cable to the PC.

Alternatively, you can also use the Ethernet connection to perform binary copy to the SD card. If you use Ethernet, you have to type in the Debian OS console

```
reboot
```

to reboot the Debian OS.

To debug from the PC in the Xilinx SDK debugger GUI, the Zynq TCF server has to be accessible from the PC via Ethernet. This can be tested in the SDK. See Figure 13.

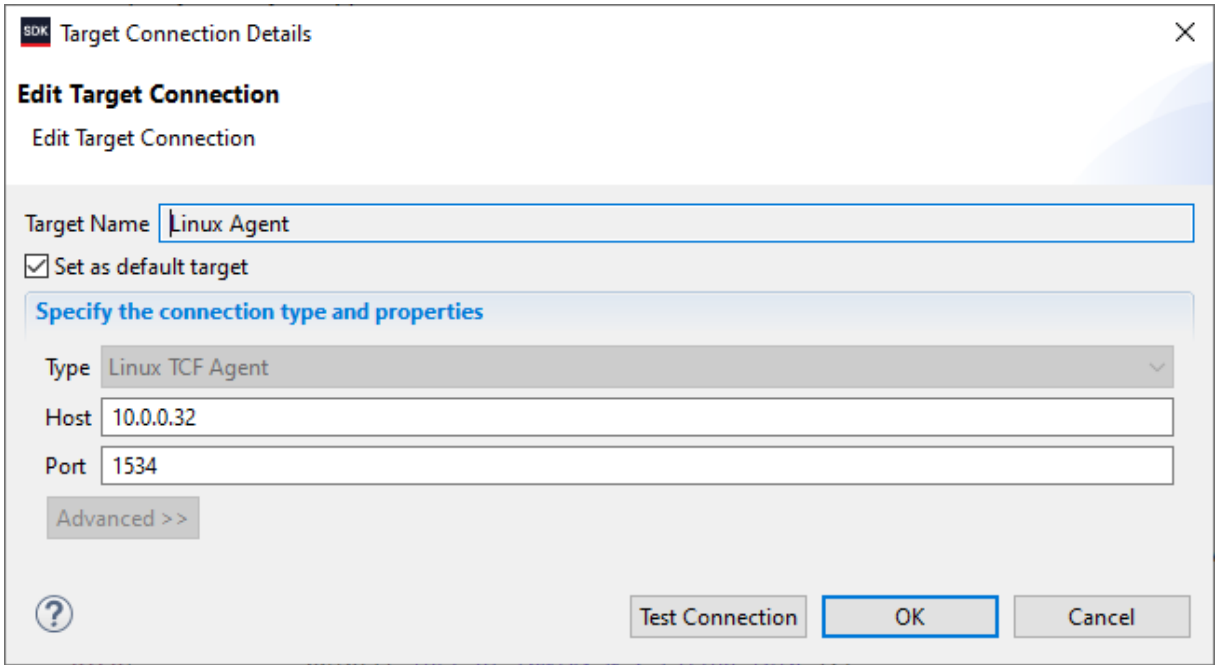


Figure 13: Test connection to Linux TCF Agent.

Recompile the ARM host SW application directly on the TE0726 board

Xilinx SDK 2018.2 tool creates files for the **`make`** utility, which can be used for compilation of SW application directly on the board with use of the **`g++`** (C++) compiler of the ARM Debian OS.

You can copy complete SDK 2018.2 project to the Debian file system and compile on board by copy complete content of the C++ SDK project directory:

c:\home\work\TS82fp01x8_TE0723_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw

to the ARM host Debian OS directory:

/home/fp01x8_v26x1_sw/

Change the directory in ARM Debian OS to:

cd /home/fp01x8_v26x1_sw/Debug/

Export the relative path to the Debug version of the shared library:

export LD_DATA_PATH=../../Debug/sd_card

In the Debian OS terminal, clean and then recompile the project by typing:

```
make clean
make
```

Finally, execute the re-compiled C++ Debug version of the SW application compiled by the ARM host Debian OS g++ compiler. Type in the Debian console:

./fp01x8_v26x1_sw.elf

You are done. The compiled application is running on the TE0726 board. See Figure 9.
To close correctly the Debian OS, type in the Debian OS terminal:

halt

This will close all open files on the SD file system and halt the ARM Debian OS.

Now you can safely remove the SD card. The USB serial terminal can remain connected.
You can modify the SD card in the Win 10 PC.
You can insert modified SD card.
You have to press the reset on the board to initiate a new Debian OS boot process (without the power-off power-on step).

Guide for compilation and use of C MEX functions in Scilab-cli

The Debian OS image includes scilab-cli SW interpreter. To use it take these steps.
In Debian OS terminal, change directory to

cd /**home/xc7z10_deb_eval_ila_release_scilab/cc/mmultf**

In ARM Debian OS terminal, Start scilab-cli interpret by typing

scilab-cli

In scilab-cli, execute script **mmultf_cc.sce** by command

exec("mmultf_cc.sce")

This script will compile C MEX function **mmultf.c** to shared library **libmex_mmultf.so** in the same directory
Quit scilab-cli by typing

```
quit
```

Copy created shared library **libmex_mmultf.so**

/home/xc7z10_deb_eval_ila_release_scilab/cc/mmultf/libmex_mmultf.so

to

/home/xc7z10_deb_eval_ila_release_scilab/test/test_mmultf_4xB/libmex_mmultf.so

In Debian terminal, change directory to

/home/xc7z10_deb_eval_ila_release_scilab /test/test_mmultf_4xB

Start scilab-cli by typing

```
scilab-cli
```

In scilab-cli execute script **mmultf_4xB_test.sce** by command

```
exec("mmultf_4xB_test.sce")
```

scilab-cli will execute **mmultf()** C MEX function present in shared library **libmex_mmultf.so** and generate reference header files in the current directory . Files contain single precision floating point reference data used for testing of 8xSIMD HW accelerators. Quit scilab-cli by typing.

```
quit
```

Use same process to compile and use all other reference MEX C functions.

12 APPENDIX – DTRiMC tool guidelines

DTRiMC tool requires the 8xSIMD HW IP core as an input. Contact UTIA to get license for use of this HW IP.

Contact UTIA to buy the required 8xSIMD HW accelerator IP:

Name of the IP:	fp03x8_v26_v40
ID:	3
Device:	xc7z010clg225-1
Tool chain:	Vivado/SDSoC 2018.2
Contact:	UTIA AV CR v.v.i.; Pod Vodarensnou vezi 4, 18208 Prague 8, Czech Republic; Jiri Kadlec; email: kadlec@utia.cas.cz tel: +420 2 6605 2216

The fp03x8_v26_v40 HW IP is not included in the evaluation package in form of the required HDL source code. The compiled evaluation version of the fp03x8_v26_v40 IP is present in the `BOOT.BIN` files in the `sd_card` directories of the evaluation package.

The evaluation package can be downloaded from UTIA for free from the www server http://sp.utia.cz/index.php?ids=projects/fitoptivis

Guide for compilation of HW in the DTRiMC tool

1. Unpack the DTRiMC evaluation package to Win10 directory. DTRiMC tool is in: `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila\`
   Change directory to:
   ```
   c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila\zsys\
   ```
2. Add the UTIA 8xSIMD HW IP to the package as the directory `\ip`
   `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila\zsys\ip_lib\ip`
3. On Win10, open dos terminal window, change directory to the folder
   `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila\zsys\`
4. To overcome limitations of Win10 related to the need of short directory paths, use the script *_use_virtual_drive.cmd* to create a virtual short path to your directory drive *X:\zusys* Type command:
   ```
   _use_virtual_drive.cmd
   ```
   Select `X` as name of the virtual drive and select `0` to create the virtual drive.
   Go to the created virtual short-path directory by typing in the win 10 terminal:
   ```
   X:
   cd zsys
   ```
5. Use text editor of your choice and open and modify script `design_basic_settings.sh`
   Select correct path to SDSoC 2018.2 tool installed on your Win7 or Win10. Line 38:
   ```
   @set XILDIR=C:/Xilinx
   ```
   Select proper Xilinx device:
   ```
   @set PARTNUMBER=3
   ```
   The selected number corresponds to the number defined in file
   `X:\zusys\board_files/TE0808_board_files.csv`
   Verify, if line 78 of script *design_basic_settings.sh* sets the SDSoC flow support by: *ENABLE_SDSOC=1*
   ```
   @set ENABLE_SDSOC=1
   ```

6. Start the Xilinx Vivado 2018.2 and create the design by executing of script:
   ```
   X:\zsys\vivado_create_project_guimode.cmd
   ```
7. Optional:
   You can use Vivado automation and to the created HW design the In Circuit Logic Analysator (ILA) monitor to enable capturing of selected accelerator outputs of your choice.
8. In Vivado console, execute command:
   ```
   TE::hw_build_design -export_prebuilt
   ```
   After the Vivado compilation, new hardware description file `zsys.hdf` is generated in folder:
   ```
   X:\zsys\prebuilt\hardware\m\zsys.hdf
   ```

Guide for configuration and compilation of PetaLinux in the DTRiMC tool

The configuration and compilation of the *Petalinux 2018.2* kernel and *Debian 9.8 Stretch* image as the FitOptiVis run time resource for the Zynq TE0726M board is described now. The configuration has to be performed in the Ubuntu 16.04 LTS OS.

The DTRiMC tool is configured for use of Ubuntu 16.04 LTS in the *VMware Workstation Player* in Win10. The Petalinux 2018.2 distribution can be downloaded to the Ubuntu 16.04 LTS from

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2018-2.html

and installed to the default Ubuntu directory:

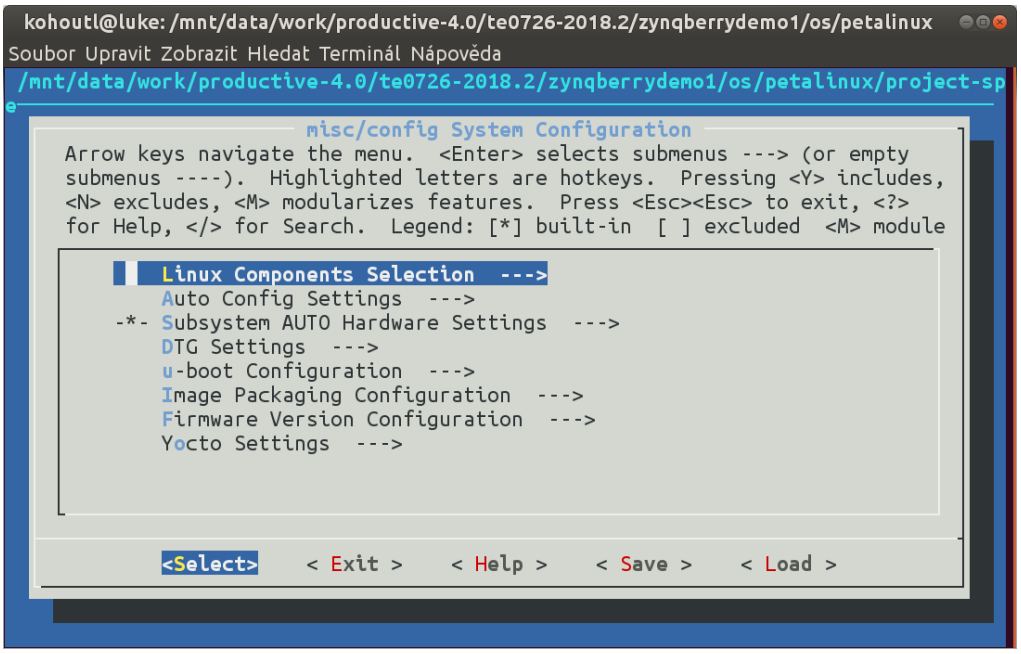
```
/opt/petalinux/petalinux-v2018.2-final
```

The standard PetaLinux 2018.2 distribution requires few modifications.

1. Copy content of these Win 10 directories:
   ```
   X:\zsys\prebuilt
   X:\zsys\os
   ```
   to Ubuntu directories:
   ```
   /home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys/prebuilt/
   /home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys/os/
   ```
2. In Ubuntu, open terminal window and set path to the PetaLinux 2018.2:
   ```
   source /opt/petalinux/petalinux-v2018.2-final/settings.sh
   ```
3. Change directory to the directory copied from the evaluation package with pre-defined configuration:
   ```
   cd /home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys/os/petalinux/
   ```
   It contains a predefined configuration according to Zynq TE0726M board requirements.
4. The zsys.hdf file created in Win 10 in Vivado 2018.2 tool is present in the Ubuntu folder:
   ```
   /home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys/prebuilt/prebuilt/hardware/m/
   ```

5. Use the `zusys.hdf` file as input for the PetaLinux configuration by (on single line)

```
petalinux-config --get-hw-
description=/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_e
val_ila/zsys/prebuilt/prebuilt/hardware/m/
```



6. Verify if the PetaLinux filesystem location is changed from the ramdisk to the extra partition on the SD card, select:

```
Image Packaging Configuration  --->
      Root filesystem type (SD card)  --->
```

7. Verrify if option to generate boot args automatically is disabled and if user defined arguments are set to:

```
earlycon clk_ignore_unused root=/dev/mmcblk0p2 rootfstype=ext4 rw
rootwait quiet
```

Leave the configuration, 3x *Exit* and *Yes*.

8. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

9. Files *image.ub*, *u-boot.elf* and *bl31.elf* are created in:

```
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys/os/
petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys/os/
petalinux/images/linux/bl31.elf
```

Guide for configuration and compilation of Debian OS in the DTRiMC tool

The file system is based on the latest stable version of Debian 9.8 Stretch distribution (03. 25. 2019). Follow the steps below.

10. In Debian, cd to the folder with PetaLinux:

```
cd
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/
```

11. The 32bit Debian image will be created by execution of the *mkdebian.sh* script. The script checks all the tools that are needed to create the image, most of them are a standard part of the Ubuntu 16.04 LTS distribution.

    When some of them are missing, install them by:

```
sudo apt install Package
```

    Table 1: tools with a corresponding package name.

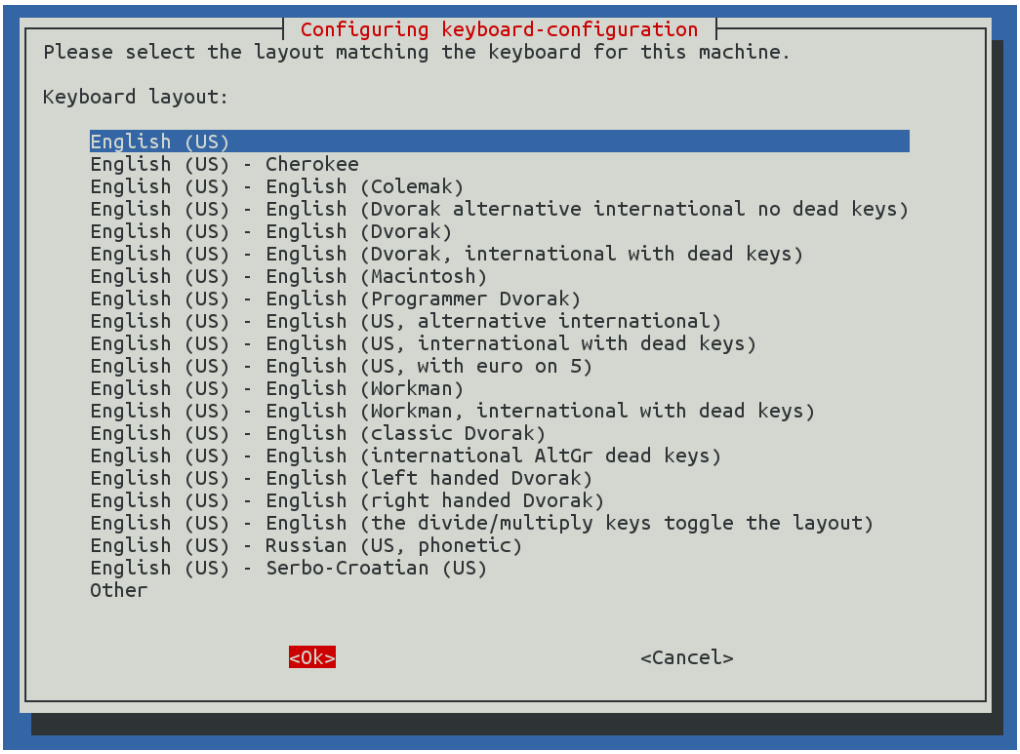
Tool	***Package***
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-ARM-static	qemu-user-static

12. Create the Debian image. It will consist of two partitions.

    The file system of the first one will be FAT32. This partition is dedicated for image of the PetaLinux kernel. The second partition will contain the Debian using EXT4 file system. Create the Debian image from the external Ethernet repositories by this command:

```
chmod ugo+x mkdebian.sh
sudo ./mkdebian.sh
```

    During the creation procedure, you will be asked to set language. Choose *English (US)*. The resultant image file will be called *TE0808-debian.img*, its size will be 7 GB.



13. Compress the created image to file TE0808-debian.zip:

```
zip te0726-debian te0726-debian.img
```

14. Copy compressed image file from Ubuntu

```
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/te0726-debian.zip
```

to Win 10 file:

```
X:\zsys\prebuilt\os\petalinux\default\te0726-debian.zip
```

15. Copy these files from Ubuntu

```
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/images/linux/bl31.elf
```

to Win 10 files:

```
X:\zsys\prebuilt\os\petalinux\default\image.ub
X:\zsys\prebuilt\os\petalinux\default\u-boot.elf
X:\zsys\prebuilt\os\petalinux\default\bl31.elf
```

16. In Ubuntu, clean Petalinux project files

```
petalinux-build -x mrproper
```

17. In Ubuntu, delete files

```
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/TE0726-debian.zip
/home/devel/work/TS82fp01x8_TE0726_DTRiMC_xc7z10/xc7z10_deb_eval_ila/zsys
/os/petalinux/TE0726-debian.img
```

18. In Ubuntu, close all applications and shut down Linux.

19. In Win 10, close the VMware Workstation Player.

You can continue with preparation of the Zynq board with created files:

- Petalinux kernel image `image.ub`
- Compressed Debian image `TE0726-debian.zip`
- U-boot program `u-boot.elf`

This ends the DTRiMC tool configuration and compilation steps for the Petalinux and Debian.

Guide for creation of SDSoC platform for TE0726M in the DTRiMC tool

20. In the open Vivado 2018.2 console, create and compile the initial *BOOT.bin* file and the initial SW modules by execution of the command:

    ```
    TE::sw_run_hsi
    ```

    The resulting *BOOT.bin* file will be located in the folder

    ```
    X:\zsys\prebuilt\boot_images\m\u-boot\BOOT.bin
    ```

21. Thes files are also created:

    ```
    X:\zsys\prebuilt\software\m\hello_te0726.elf
    X:\zsys\prebuilt\software\m\zynq_fsbl.elf
    X:\zsys\prebuilt\software\m\zynq_fsbl_flash.elf
    ```

    File `zynq_fsbl.elf` is correct first stage board loader (FSBL) file, while the `zynq_fsbl_flash.elf` is special FSBL file used only for programming of the on board flash.

22. Move file `zynq_fsbl_flash.elf` to some different temporary location before next step.

23. In Vivado 2018.2 console, create the SDSoC platform by execution of the command:

    ```
    TE::ADV::beta_util_sdsoc_project
    ```

    The SDSoC 2018.2 platform is generated in the directory

    ```
    X:\SDSoC_PFM\te0726\03m\zsys
    ```

    and it is also packed into the ZIP file in directory

    X:\SDSoC_PFM\te0726\

24. Return file `zynq_fsbl_flash.elf` back from the temporary location to

    X:\zsys\prebuilt\software\m\zynq_fsbl_flash.elf

    It will be used later on by the TE0726M board flash programming script

    ```
    program_flash_binfile.cmd
    ```

Guide for creation of shared library and HW kernel in the DTRiMC tool

25. On Win10, *in the open dos terminal window, cancel the current virtual drive X: by executing from* the command line

    ```
    _use_virtual_drive.cmd
    ```

    and response (1)

26. Change directory to

    ```
    c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila\SDSoC_PFM_src\te0726\03m\
    ```

27. In Win10, open dos terminal window and use the copy of the script *_use_virtual_drive.cmd* to create a new virtual short path to get short SDSoC directory `X:\03m`

    ```
    _use_virtual_drive.cmd
    ```

Select X as name of the virtual drive and select (0) to create the virtual drive.
Go to the created virtual short-path directory by:

```
X:
cd 03m
```

28. Open SDSoC project in directory

    ```
    X:\03m
    ```

29. In SDSoC import HW kernel design project

    `fp01x8_v26x1_hw`

    from the directory

    `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila\SDSoC_PFM_src\te0726\03m\fp01x8_v26x1_hw\`

    Define the custom SDSoC platform

    `X:\03m\zsys`

30. Change imported project from Debug to the Release compilation target
31. Compile project by the SDSoC 2018.2 compiler
32. Result of compilation are the SD cards with the `BOOT.BIN` file and the shared object library file `libfp01x8_v26x1_hw.so` in the directory:

    `X:\03m\fp01x8_v26x1_hw\Release\sd_card\`

33. Copy content of the directory

    `X:\03m\fp01x8_v26x1_hw\Release\sd_card\`

    to

    `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Release\sd_card\`

    and also to

    `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Debug\sd_card\`

34. Optional:

    Copy ILA nets definition files `debug_nets.ltx` and `zsys_wrapper.ltx` from the directory

    `X:\03m\fp01x8_v26x1_hw\Release\_sds\p0\vivado\prj\prj.runs\impl_1\`

    to

    `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Release\sd_card\`

    and also to

    `c:\home\work\TS82fp01x8_TE0726_DTRiMC_xc7z10\xc7z10_deb_eval_ila_release\fp01x8_v26x1_sw\Debug\sd_card\`

35. Clean SDSoC project to save disk space.
36. Close SDSoC 2018.2 tool.
37. The created `BOOT.BIN` file will be used for programming of TE0726M board flash.
38. The shared object library file `libfp01x8_v26x1_hw.so` is to be linked to applications compiled for ARM in SDK and also used in the runtime on ARM. This is described in the first section of the chapter 10 APPENDIX - Confidence test.

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.