Application Note

DTRiMC tool for TE0808-09-EG-ES1 module on TEBF0808 carrier board

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

Revision history

Rev.	Date	Author	Description
0	6.02.2021	J. Kadlec	Initial draft
1	3.06.2021	J.Kadlec	Platform updated with ILA analyzer and an SDSoC mmult HW accelerator for floating point matrix multiplication.
2	5.07.2021	J. Kadlec	DTRiMC tool for ZU09-EG-ES1

Table of Contents

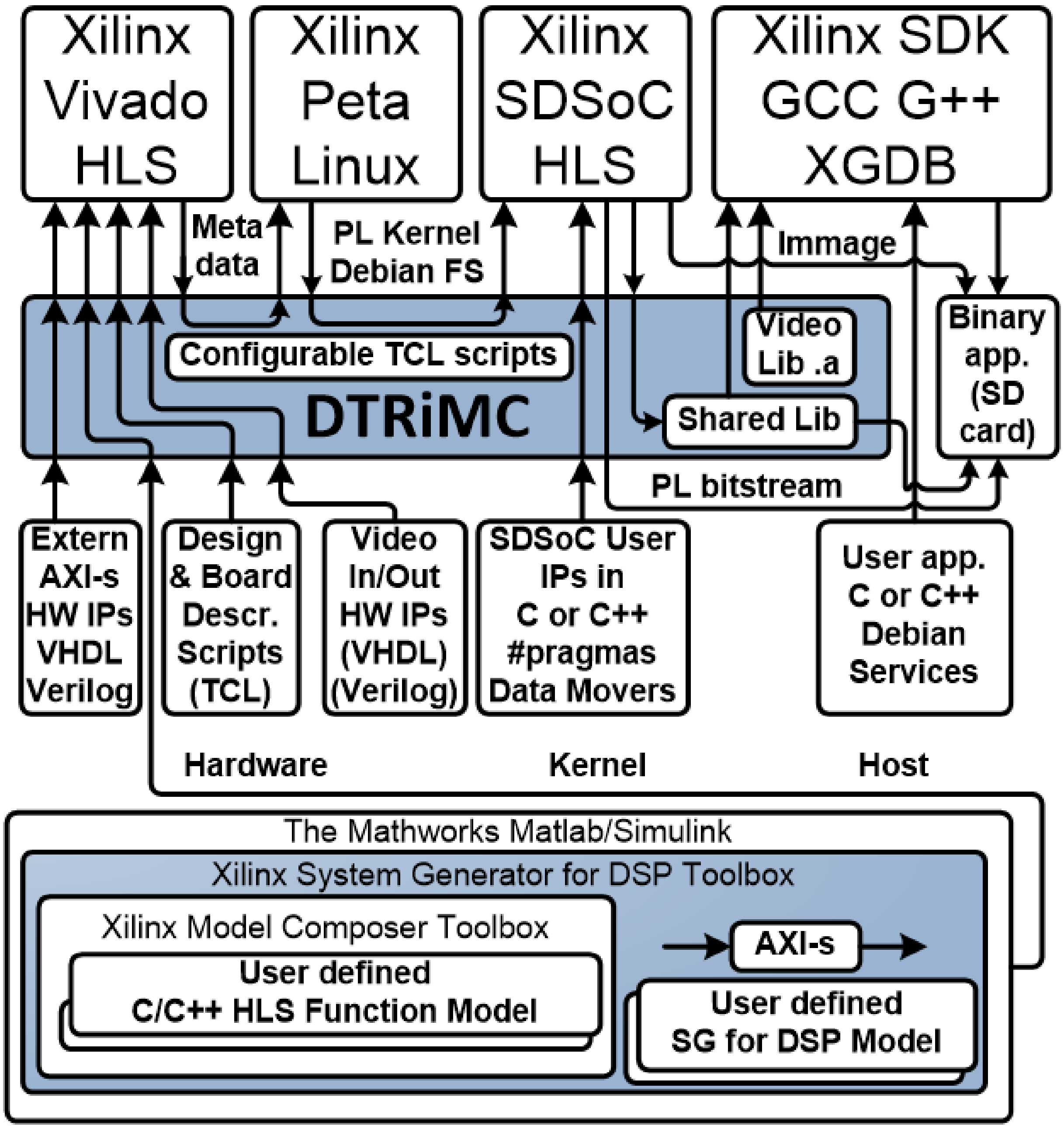
Table of Figures

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 ZU09-EG-ES1

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 eight 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerators for Zynq Ultrascale+ TE0808-09EG-ES1 module [1] on TEBF0808 carrier board [2].



Xilinx Vivado HLS
Xilinx Peta Linux
Xilinx SDSoC HLS
Xilinx SDK GCC G++ XGDB
Meta data
PL Kernel Debian FS
Immage
Video Lib .a
Binary app. (SD card)
Configurable TCL scripts
DTRiMC
Shared Lib
PL bitstream
Extern AXI-s HW IPs VHDL Verilog
Design & Board Descr. Scripts (TCL)
Video In/Out HW IPs (VHDL) (Verilog)
SDSoC User IPs in C or C++ #pragmas Data Movers
User app. C or C++ Debian Services
Hardware
Kernel
Host
The Mathworks Matlab/Simulink
Xilinx System Generator for DSP Toolbox
Xilinx Model Composer Toolbox
User defined C/C++ HLS Function Model
AXI-s
User defined SG for DSP Model

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

The TE0808-09EG-ES1 module and TEBF0808 carrier board are designed and manufactured by the company Trenz Electronic [1]. Xilinx device ZU09-EG-ES1 device requires in the design phase the Xilinx Vivado tool version 2017.4. This tool must have enabled support for the Xilinx ZU09-EG-ES1 device. The Xilinx Vivado 2017.4 is the last Xilinx toolchain still supporting the ZU09-EG-ES1 device.

The evaluation package provides SW projects and two designs containing the HW design bitstreams and API interface for SW developer in form of shared linux libraries. The SW developer can program ARM host application in C and compile by gcc compiler or in C++ and use the g++ compiler. User can use the Xilinx SDK for compilation and debug of provided SW projects on a PC (Linux or Windows 10, 64bit). The „make" utility can be also used for compilation of host applications directly on the embedded Zynq Ultrascale+ ZU09-EG-ES1 system.

All designs present in this evaluation package contain four independent twins of serial connected FP03x8 accelerators in the programmable logic part of the device.

The HW data movers supporting the data communication are represented for the SW developer as shared c/C++ library with simple SW API. The API is identical for several alternatives of HW data movers.

The evaluation package includes 8xSIMD FP32 accelerators 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 license for 8xSIMD accelerators without this restriction.

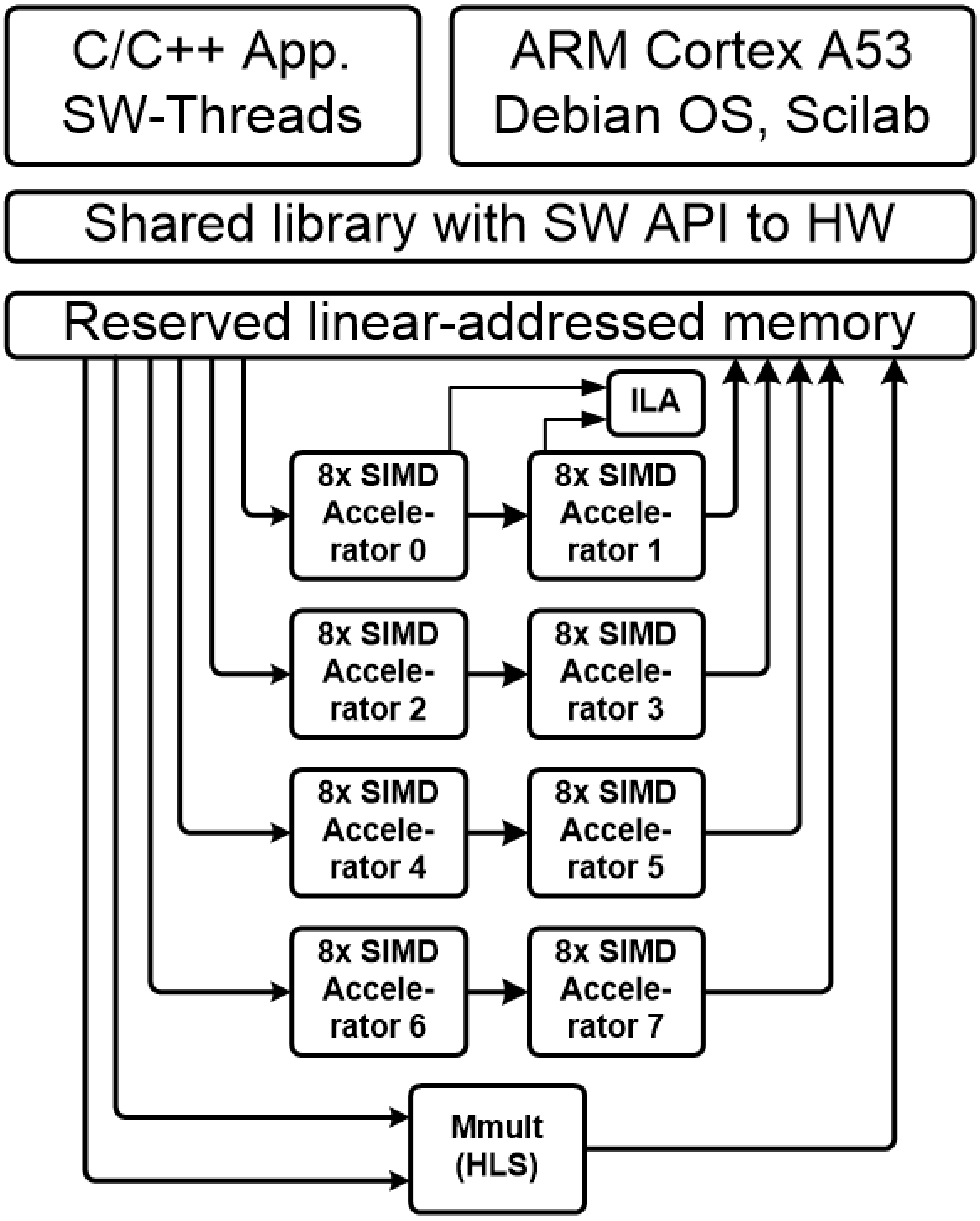


Figure 2: Eight connected FP03x8 accelerators in ZU09-EG-ES1 device

2 8xSIMD FP03x8 floating point accelerators for ZU09-EG-ES1

The FP03x8 HW accelerators serve for run-time reprogrammable 8xSIMD single precision floating point computation. The internal structure of FP03x8 accelerators is described in Figure 3.

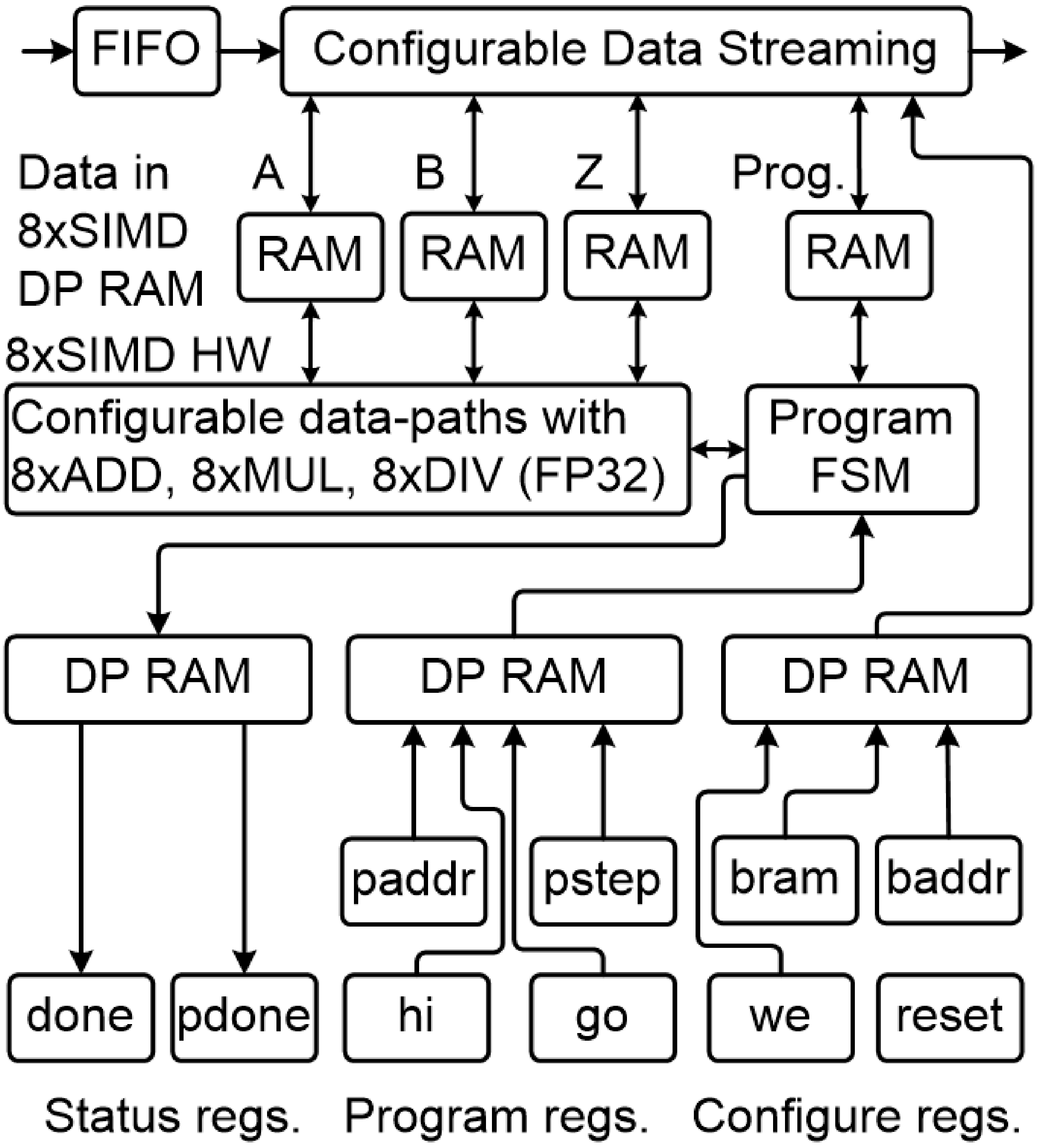


Figure 3: FP03x8 accelerator for the ZU09-EG-ES1 device

Accelerator Interfaces

Type of interface:	Device:	Clock:
• Data streaming I/O: AXI-S 32 bit	ZU09-EG-ES1	214 MHz
• Firmware program VLIW 128 bit	ZU09-EG-ES1	214 MHz
• Configuration I/O: AXI-lite 32 bit	ZU09-EG-ES1	150 MHz
• 4x ARM A53 system clock	ZU15-EG-1EE	1050 MHz

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

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 accelerators.

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:

• 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 2017.4 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 accelerators:

- **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}.

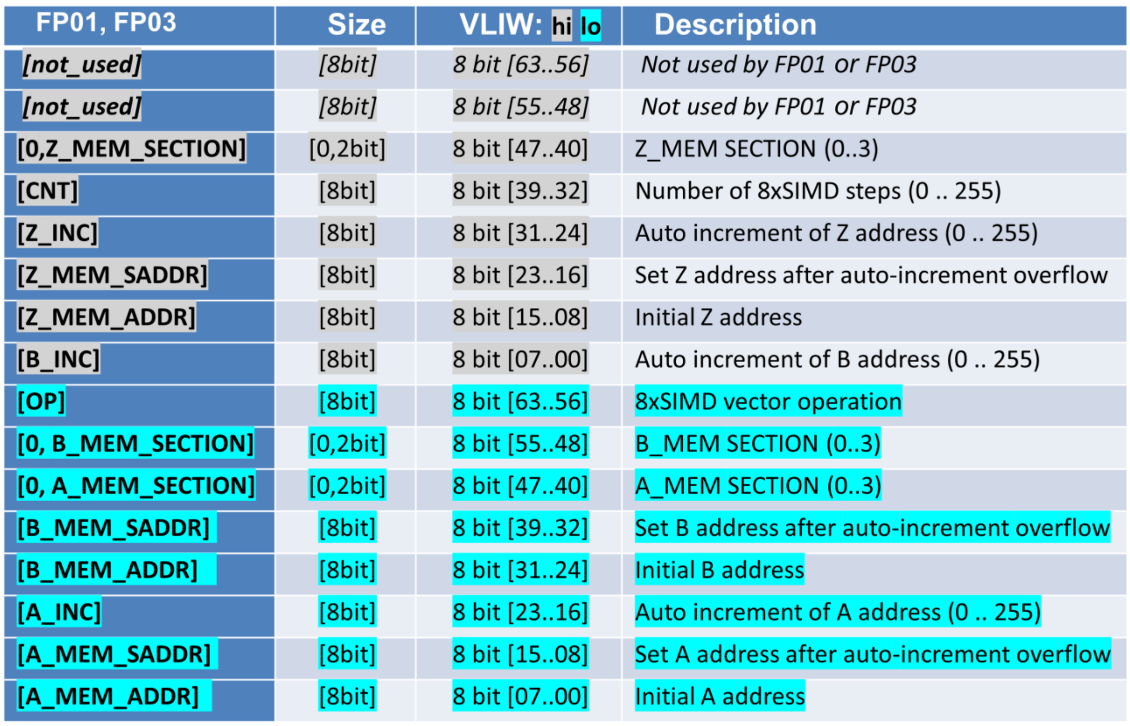
SIMD OP	code (dec)	8xSIMD Floating Point Operation Description
VPROD FP01, FP03: 30,40	14	8xSIMD vector products. $z_m[i] <= a_m'[j..j+nn]*b_m[k..k+nn]$; m=1..8; nn range 0..255
VMAC FP01, FP03: 20,30,40	15	8xSIMD vector MACs. $z_m[i..i+nn] <= z_m[i..i+nn] + a_m[j..j+nn] * b_m[k..k+nn]$; m=1..8; nn range 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 versions accelerators.

3 Programming of 8xSIMD FP03x8 floating point accelerators

Host arm application can form the VLIW program instructions in DDR4 memory as two 64bit words. In Figure 7, components of the low 64 bit word are marked by light green background. Components of the high 64bit word components are marked by light blue.

Host arm application can form a sequence of such VLIW program instructions in DDR4 memory and write them to one of two accelerator program memories.



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 128 bit wide VLIW program instruction.

Sequences of VLIW instructions present in the accelerator program memory 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 host 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 is completed.

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

The host application can also copy data/program to/from the accelerator while the sequence of VLIW instructions is being executed on current internal data and internal program 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 application, by writing 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 to a single VLIW instruction. The **paddr** and **pstep** registers are set to an identical program address in such case.

This technique can be used by the developer of host Arm program for stepping through the sequence of VLIW instructions one by one. User can modify the host application for reading partial results of each VLIW instruction. Data can be uploaded from the accelerator to host app for inspection and stepping through/debug in the Xilinx SDK 2017.4 gdb debugger GUI.

4 C and C++ evaluation projects

The PL part of the ZU09-EG-ES1 device contains:

- Eight evaluation versions of the 8xSIMD run-time-reprogrammable single-precision-floating-point HW accelerator FP03x8 organized as 4x2 accelerators.
- Mmul() floating point matrix multiplication HW accelerator created in SDSoC 2017.4.
- ILA in circuit logic analyzer

This HW can be tested by two sets of evaluation SW projects.
SW examples in C use gcc compiler.
SW examples in C++ use g++ compiler

Directory (C):	**fp03x8_v26_4x2_mulf64_all_sw_c**
SW C project:	**all**
Shared C library:	**./Debug/sd_card/libfp03x8_v26_4x2_c_mulf64_hw.so**
Shared C library:	**./Release/sd_card/libfp03x8_v26_4x2_c_mulf64_hw.so**

Directory (C++):	**fp03x8_v26_4x2 mulf64_all_sw**
SW C++ project:	**all**
Shared C++ library:	**./Debug/sd_card/libfp03x8_v26_4x2_hw.so**
Shared C++ library:	**./Release/sd_card/libfp03x8_v26_4x2_hw.so**

These shared libraries for Debian Stretch 9.8 OS provide the C and C++ SW API for the ZU09EG-ES1 device. The HW supported data transfers require data to be present in “sd_alloc” memory (continuous physical section reserved in the DDR4).

SW projects **all** demonstrate HW acceleration of eight single precision floating point matrix by matrix multiplications and test of all firmware functions of accelerators. Projects test matrix multiplications and short sequences of elementary operations of 8xSIMD HW accelerators.

Tested	**Function**
mmultf1_4xB	8x mmult use B1..B4, include parallel copy of B1..B4
mmultf1_8xB	8x mmult use B1..B8, include parallel copy of B1..B8 SDSoC HW mmult use B1..B8, include copy of A, B, Z SW mmult Scilab MEX style, 4 threads
va2bf1	8x va2b
vaddf1	8x vadd
vaddf1_az2b	8x vadd_az2b
vaddf1_bz2a	8x vadd_bz2a

vb2af1	8x vb2a
vdivf1	8x vdiv
vmulf1	8x mul
vmulf1_az2b	8x mul_az2b
vmulf1_bz2a	8x mul_bz2a
vprodf1	8x vprod
vprods8f1	8x vprods8
vsubf1	8x vsub
vsubf1_az2b	8x vsub_az2b
vsubf1_bz2a	8x vsub_bz2a
vz2af1	8x vz2a
vz2bf1	8x vz2b
vmacf1	8x vmac
vmsubacf1	8x vmsubac

The eight instances of the FP03x8 accelerators on ZU09-EG-ES1 device are controlled by 4 SW threads.

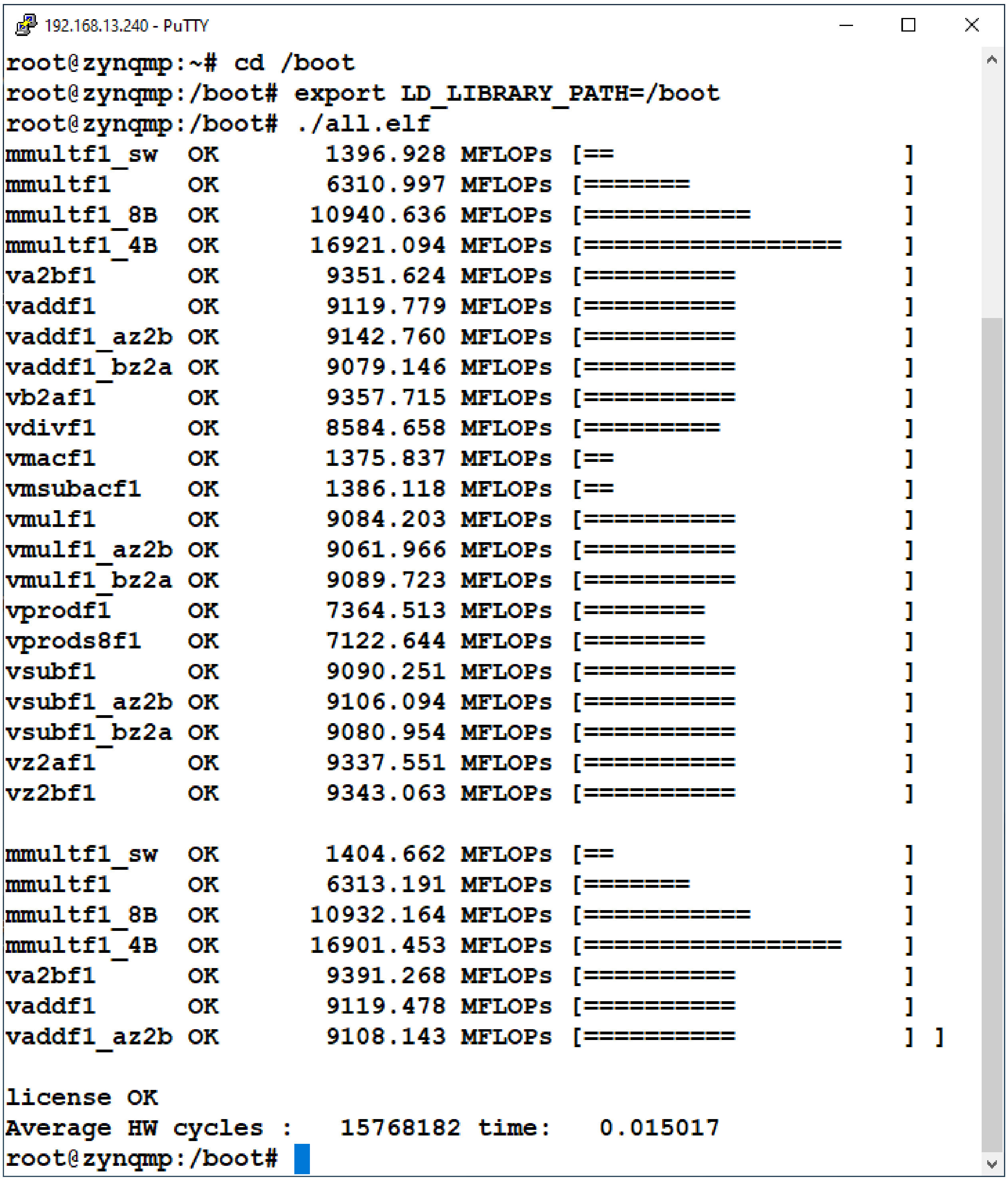
In case of C project, HW accelerators accelerate the SW optimized (-O2) code executed on four A53 cores running with 1.05 GHz clock.

In case of C++ project, HW accelerators accelerate the SW optimized (-O3) code executed on four A53 cores running with 1.05 GHz clock.

Comparison of matrix multiplication performance:

Comparison of matrix multiplication performance:

System	**Function**	**MFLOPs**
ZU09-EG-ES1	8x mmult use B1..B4, include parallel copy of B1..B4	16 921
	8x mmult use B1..B8, include parallel copy of B1..B8	10 940
	SDSoC HW mmult use B1..B8, include copy of A, B, Z	6 310
	SW mmult Scilab MEX style, 4 threads	1 396
	SW mmult C style, 1 thread	895
	SW Scilab C MEX style , 1 thread	166
I7 PC 3.0 GHz	SW Ubuntu, SciLab C mex, 1 thread:	1 933



192.168.13.240 - PuTTY

```
root@zynqmp:~# cd /boot
root@zynqmp:/boot# export LD_LIBRARY_PATH=/boot
root@zynqmp:/boot# ./all.elf
mmultf1_sw   OK       1396.928 MFLOPs [==                 ]
mmultf1      OK       6310.997 MFLOPs [=======            ]
mmultf1_8B   OK      10940.636 MFLOPs [===========        ]
mmultf1_4B   OK      16921.094 MFLOPs [=================  ]
va2bf1       OK       9351.624 MFLOPs [==========         ]
vaddf1       OK       9119.779 MFLOPs [==========         ]
vaddf1_az2b  OK       9142.760 MFLOPs [==========         ]
vaddf1_bz2a  OK       9079.146 MFLOPs [==========         ]
vb2af1       OK       9357.715 MFLOPs [==========         ]
vdivf1       OK       8584.658 MFLOPs [=========          ]
vmacf1       OK       1375.837 MFLOPs [==                 ]
vmsubacf1    OK       1386.118 MFLOPs [==                 ]
vmulf1       OK       9084.203 MFLOPs [==========         ]
vmulf1_az2b  OK       9061.966 MFLOPs [==========         ]
vmulf1_bz2a  OK       9089.723 MFLOPs [==========         ]
vprodf1      OK       7364.513 MFLOPs [========           ]
vprods8f1    OK       7122.644 MFLOPs [========           ]
vsubf1       OK       9090.251 MFLOPs [==========         ]
vsubf1_az2b  OK       9106.094 MFLOPs [==========         ]
vsubf1_bz2a  OK       9080.954 MFLOPs [==========         ]
vz2af1       OK       9337.551 MFLOPs [==========         ]
vz2bf1       OK       9343.063 MFLOPs [==========         ]

mmultf1_sw   OK       1404.662 MFLOPs [==                 ]
mmultf1      OK       6313.191 MFLOPs [=======            ]
mmultf1_8B   OK      10932.164 MFLOPs [===========        ]
mmultf1_4B   OK      16901.453 MFLOPs [=================  ]
va2bf1       OK       9391.268 MFLOPs [==========         ]
vaddf1       OK       9119.478 MFLOPs [==========         ]
vaddf1_az2b  OK       9108.143 MFLOPs [==========         ] ]

license OK
Average HW cycles :   15768182 time:   0.015017
root@zynqmp:/boot#
```

Figure 8: Performance results of all.elf application.

Performance results are listed in Figure 8. It is terminal output from the all.elf application running on Arm.

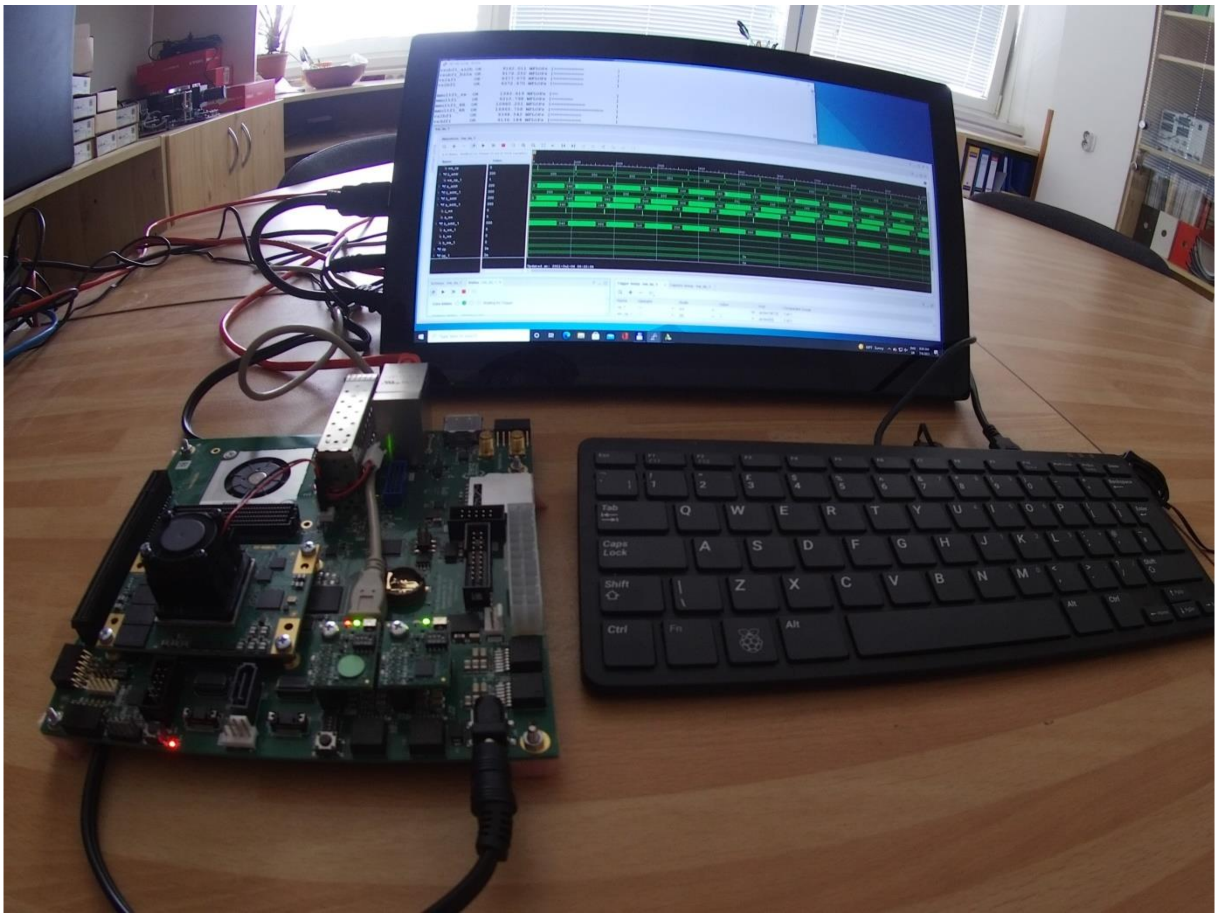


Figure 9: Running system, all.elf application, ILA display HW matrix multiplication

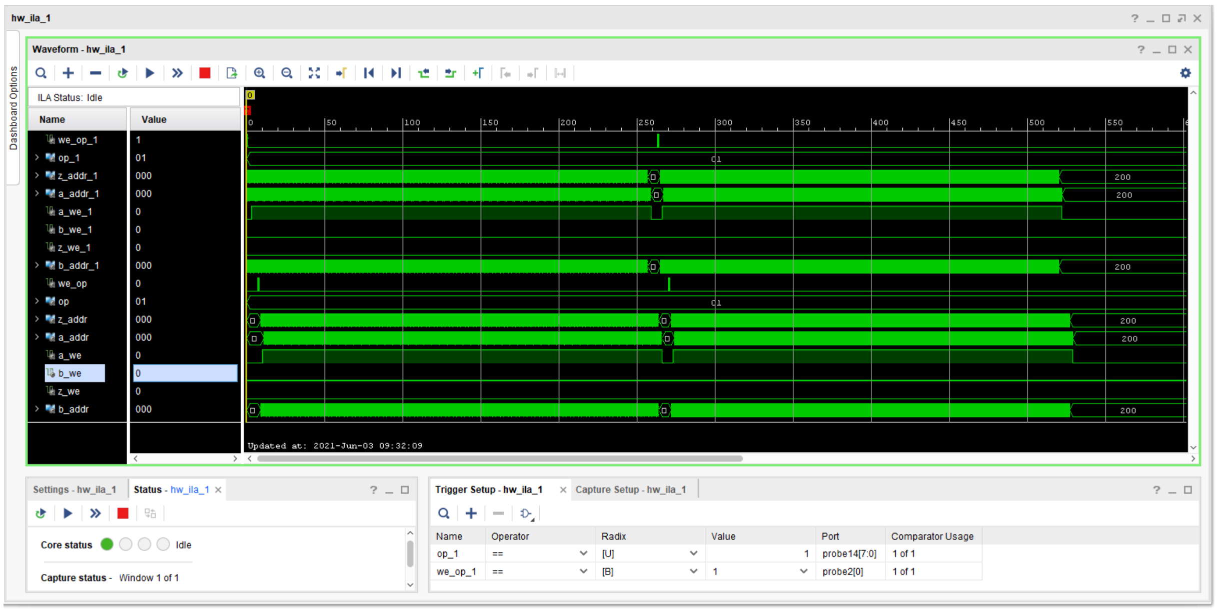
5 Power consumption

Power consumption is measured on input power line 12V. All power supply is derived from this single power source.

Power consumption	**Power [W]**
Linux system is running with all HW interfaced by the library libfp03x8_v26_4x2_c_mulf64_hw.so is present in the device. No user app is running	12,00
Linux system is running with all HW interfaced by the library libfp03x8_v26_4x2_c_mulf64_hw.so is present in the device. SW application **all.elf** is running. It performs all tests of all 8 8xSIMD HW accelerators. Tests control HW accelerators from 4 SW threads running on 4 Arm cores. See the terminal output in Figure 8 and photograph of complete system in Figure 9.	13,32

6 ILA – In-circuit Logic Analyzer

System created by the Design Time Resource integration of Model Composer DTRiMC tool includes HW IP of the Vivado-Lab tool 2017.4 ILA – In-circuit Logic Analyzer. It is connected to HW 8xSIMD accelerators 0 and 1. ILA can be triggered by specific instruction and displays addresses and we signals with 214 MHz clock. It is configured to sample 1024 data. See Figure 2.

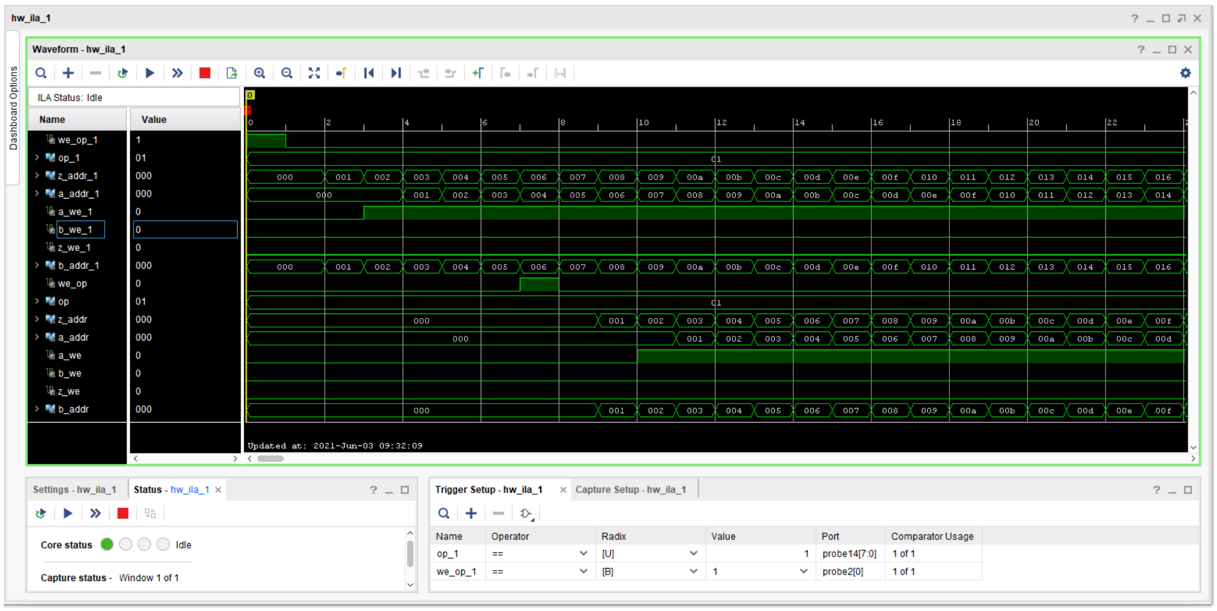


Name	Value
we_op_1	1
op_1	01
z_addr_1	000
a_addr_1	000
a_we_1	0
b_we_1	0
z_we_1	0
b_addr_1	000
we_op	0
op	01
z_addr	000
a_addr	000
a_we	0
b_we	0
z_we	0
b_addr	000

Name	Operator	Radix	Value	Port	Comparator Usage
op_1	==	[U]	1	probe14[7:0]	1 of 1
we_op_1	==	[B]	1	probe2[0]	1 of 1

Figure 10: instruction vz2a.

Figure 10 presents complete vz2a operation test. It is SW project vz2af0. It performs copy of 512 FP data from all Z memories of accelerators to all A memories. Copy is executed as program sequence of two VLIW instructions, each performing copy of 256 FP data. This is visible in Figure 10.In ILA, we can zoom to see the details. See Figure 11. The we_op_1 == 1 and op_1 == 1 is the trigger condition for ILA set by user. The we_op_1 can bee 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 8xSIMD memories in parallel.



Name	Value
we_op_1	1
op_1	01
z_addr_1	000
a_addr_1	000
a_we_1	0
b_we_1	0
z_we_1	0
b_addr_1	000
we_op	0
op	01
z_addr	000
a_addr	000
a_we	0
b_we	0
z_we	0
b_addr	000

Name	Operator	Radix	Value	Port	Comparator Usage
op_1	==	[U]	1	probe14[7:0]	1 of 1
we_op_1	==	[B]	1	probe2[0]	1 of 1

Figure 11: instruction vz2a detail.

Figure 11 also demonstrates the relation of both observed accelerators. Both accelerators compute the vz2a instructions with time shift of 7 clock cycles. This time shift is given by the shifted start due to the sequential execution of ARM instructions activating the computation in 8xSIMD HW accelerators. We see that the accelerator with *_1 variables was started by ARM program first.

The instantiated ILA helps mainly in analysis and debug of more complex 8xSIMD HW accelerator program sequences.

7 License

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

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

The license for the evaluation versions of accelerators 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 **FP03x8** accelerators is available in UTIA. UTIA offers this license on commercial base. Contract with UTIA is required. For information about details of the commercial license write to Jiri Kadlec kadlec@utia.cas.cz.

8 Conclusion

This application note describes evaluation system precompiled by the Design Time Resource integration of Model Composer DTRiMC tool developed in the frame of FitOptiVis project. See Figure 1.

DTRiMC tool serves for integration of eight 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerators for Zynq Ultrascale+ TE0808-09EG-ES1 module [1] on TEBF0808 carrier board [2].

DTRiMC tool requires the 8xSIMD, FP03x8 accelerator as input. The the 8xSIMD, FP03x8 accelerator is not included in the evaluation package. UTIA offers this license on commercial base.

To test the SW application and integration of the eight 8xSIMD, FP03x8 accelerators, UTIA provides for free download the evaluation package with precompiled evaluation version of 8xSIMD, FP03x8 accelerators in a bitstream. The evaluation package presents these system properties:

The run-time reconfigurable floating point accelerators for the ZU09-EG-ES1 device have been designed and realized with respect to the following considerations and requirements:

1. Software utilizing the accelerator can be developed also directly on the embedded system, using the C compiler (gcc) or C++ compiler (g++) present in the Debian Stretch 9.8 operating system running on the Arm A53 device.
2. The entire HW platform with 4x2 FP32x8 SIMD HW accelerators, is provided in form of a shared library. The provided shared library API is compatible with the standard gcc and g++ based compilation flows. Scripts are auto generated for the standard Debian OS “make” of the embedded system.

3. The 4x2 FP32x8 SIMD HW hardware of the floating point accelerators is fixed. Reconfiguration is performed by reprogramming the firmware code. The firmware defines what sequences of operations will do the programmable finite state machine (FSM) inside the accelerator.
4. Data communication is implemented as an AXI-stream and supports accelerator chaining. The 4x2 FP32x8 SIMD HW hardware configuration is provided.
5. In DTRiMC tool, the data communication support HW data movers are defined in design time and cannot be changed during the run time. The following variants are prepared:
   a. Zero copy (ZC) HW data movers with C interface, (minimal HW resources)
   b. Zero copy (ZC) HW data movers with C++ interface (minimal HW resources)
   c. DMA data HW data movers with C++ interface
   d. Combination of ZC HW (DDR to Accelerator) and SG DMA HW (Accelerator to DDR) with interrupts and C++ interface.

   All communication alternatives work with identical SW API. It means that the user host SW code for ARM A53 remains identical and does not need modifications for all four versions of HW data movers.

6. Software must be able to query and identify which SIMD FP operations are supported by each HW accelerator. Based on this information, the software can be reconfigured to take the advantage of supported operations.
7. The accelerator must be able to query and identify information about the actual status of the HW license defined in with each HW accelerator.
8. The HW accelerator scheduler executing the sequence of VLIW operation is very simple. It can execute only a linear sequence of VLIW vector instructions. It does not support *for-loops*, *if-else*, and similar constructs. There is also no support for checking for the overflow/underflow or NaN in performed floating point operations. All these program control constructs have to be implemented in the host code running on the ARM A53 processor.
9. Computations performed in HW accelerators can overlap with stream-based data communications. This is controlled by the user host software running on the ARM A53 processor, usually in several parallel executed threads.
10. Data are stored as 64 bit words. This arrangement enables potential use the Ultra RAM blocks (4096x64b) present in some larger Zynq UltraScale+ devices without affecting the accelerator library API or user code.

Reconfiguration of accelerator by change of firmware

The FP32x8 HW accelerator executes sequences of VLIW vector instructions (firmware) stored in accelerator program memory. This firmware can be first defined in the Arm host software and then downloaded via the streaming interface to the accelerator. The program memory will usually contain multiple different sequences of VLIW instructions.

Computation performed in the accelerator can overlap with stream-based data communication. This is controlled by the Arm host software and it can be used for run-time reconfiguration by loading a new VLIW instruction sequence to the accelerator program memory while computation is in progress.

For example, consider an application which needs to perform accelerated multiplication of 64x64 matrices (Z[64,64] = A[64,64] × B[64,64]). The application running on the host will split the matrix operation into shorter sequences of VLIW instructions and loaded instruction sequences into the accelerator program memory schedule scheduled by the application software running on the ARM host by adjusting pointers to instruction sequences to be loaded into the accelerator program memory while streaming parts of matrix B[64,64] from host DDR memory to the accelerator. Rows of the matrix are propagated as identical to all 8xSIMD memories in 8 stages.

Reconfiguration of accelerator by temporary change of firmware

Application software can temporarily reconfigure the accelerator in the following steps:

1. Save pars of data and firmware from accelerator to DDR4,
2. Change firmware and upload it to the accelerator,
3. Execute the firmware (for example the **SupOp** instruction)
4. Read the results from accelerator data memory into ARM host memory,
5. Restore saved data and firmware back from DDR to accelerator.

After performing the above steps, the accelerator data and firmware are back in its original state and can continue. The application software running on the ARM host has information about the supported SIMD operations as well as about the status of the HW license.

This temporary replacement of firmware and data can be re-used and work independently on the actual content of the accelerator.

Consider a scenario in which the host application software needs to find out if needed VLIW instructions and the corresponding SIMD operation is actually supported by the HW accelerator.

This information is required by the host software to decide, which firmware version can be used for programming of the HW accelerator:

- If the **DotProd** instruction is supported by the HW accelerator, the computation of 64x64 matrix multiplication (Z[64,64] = A[64,64] × B[64,64]) will use the instruction to improve efficiency.
- If the **DotProd** instruction is unsupported by the HW accelerator, the host software running on the ARM processor can implement the accelerated matrix multiplication using different sequences of **Mac** (multiply and accumulate) VLIW instructions.
- If the **Mac** instruction is also unsupported, the matrix multiplication can be implemented by again different sequence of **Add** and **Mult** VLIW instructions.

The performance of the matrix multiplication migt be reduced in the last case, but such HW accelerator requires less HW resources in the programmable logic of the device.

Use of such compact HW accelerators with reduced set of VLIW instructions might be necessary if the programmable logic area is limited. See **Chyba! Nenalezen zdroj odkazů.** or available HW accelerator versions.

9 References

[1] The module supported by this application note (TE0808-ES1 with Xilinx device xczu9eg-ffvc900-1-i-es1 and 2GB DDR4) is not in production at present.
This module was available as one of the first widely used Zynq Ultrascale+ industrial module, mainly in the time period from 2016 and 2017.

Similar, currently available products:

Trenz Electronic, UltraSOM+ MPSoC Module with Zynq UltraScale+ ZU9EG-1FFVC900E, 4 GByte DDR4, module with order number: TE0808-04-9BE21-A

https://shop.trenz-electronic.de/en/TE0808-04-9BE21-A-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-ZU9EG-1FFVC900E-4-GByte-DDR4

or module with order number: TE0808-05-9BE21-A

https://shop.trenz-electronic.de/en/TE0808-05-9BE21-A-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-XCZU9EG-1FFVC900E-4-GByte-DDR4

On request, UTIA can recompile HW systems for TE0808-ES1 device presented in this application note also for these currently supported modules.
Designs for these two modules would require Xilinx Vivado and SDK 2018.2 tools.

[2] Trenz Electronic, UltraITX+ Baseboard for Trenz Electronic TE080X UltraSOM+
https://shop.trenz-electronic.de/en/TEBF0808-04A-UltraITX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261

10 APPENDIX - Confidence test

This is basic confidence test of the evaluation package.

Unzip evaluation package to Win 10 directory of your choice.
`c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\`

Precompiled HW and SW projects are located in directory:
`c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\`

sd_card image is located in directory:
`c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release_sdcard\`

INSTALLATION OF TOOLS

- Install Xilinx SDK 2017.4 on Win 10 PC 64 bit.
- Install Xilinx Lab Tools 2017.4 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 disk image on PC and use Win32DiskImager to write the disk image from the PC to the SD card.

HW SETUP

- Insert the SD with disk image card to the Zynq Ultrascale+ board.
- Connect PC and Zynq Ultrascale+ to Ethernet.
- Connect USB serial terminal cable to Zynq Ultrascale+ and to PC.

TEST

- Zynq Ultrascale+ will start to boot OS. If you have Full HD HDMI monitor, kbd and mouse connected to Zynq, you will get access to the Zynq Ultrascale+ Debian desk-top.
- Optional: Open Putty terminal. Set it to:
  (115200 bps, 8 data bits, stop bit 1, parity none, flow control off)
- Optional: Use Putty terminal to login as user: **`root`** password: **`root`**
- Change directory to **`/boot`**
- Export path to the shared library. Type in Debian Linux terminal or desktop terminal:
  `export LD_LIBRARY_PATH=/boot`
- Start application code by typing:
  `./all.elf`

RESULT

- The application will compute eight single precision floating point matrix multiplications
  - In SW on ARM A53
  - HW accelerated by single SDSoC mmult() HW accelerator
  - HW accelerated by eight 8xSIMD FP03x8 accelerators.
- The application will tests all elementary operations of 8xSIMD HW accelerators
- Results of ARM and HW accelerated computations are compared to be identical and MFLOPs performance is displayed.
- The running **`sobel_all.elf`** application can be stopped by Ctrl-C key on the Arm terminal keyboard.

Compilation and debug of projects from source code

The evaluation package includes SW projects for Xilinx SDK 2017.4 tool running on Win10 or Ubuntu PC.

These projects can be modified and recompiled for ARM and executed on Zynq Ultrascale+ with or without debugging support.

In Xilinx SDK 2017.4, select working directory for C or for C++ projects:
fp03x8_v26_4x2_mulf64_all_sw_c (C projects in this directory link to the same .so library)
or
fp03x8_v26_4x2_mulf64_all_sw (C++ projects in this directory link to an another .so library)

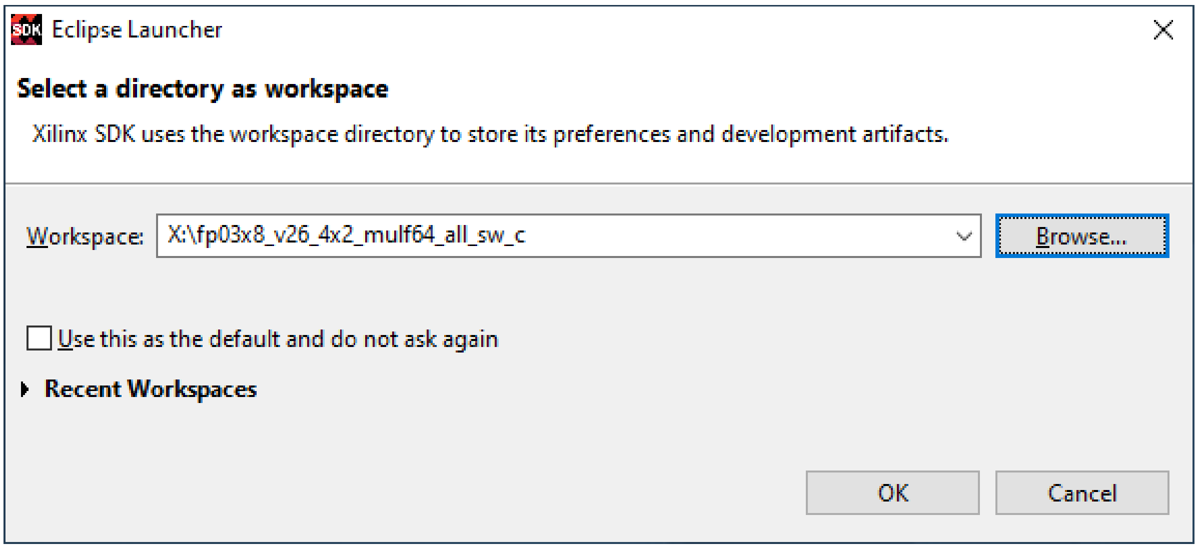
Each project has two configurations:

- **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 Xilinx SDK 2017.4 tool on your PC.

DEBUG of SW application from Xilinx SDK 2017.4

The application can be executed or debugged from the SDK 2017.4 For example:


Eclipse Launcher

Select a directory as workspace

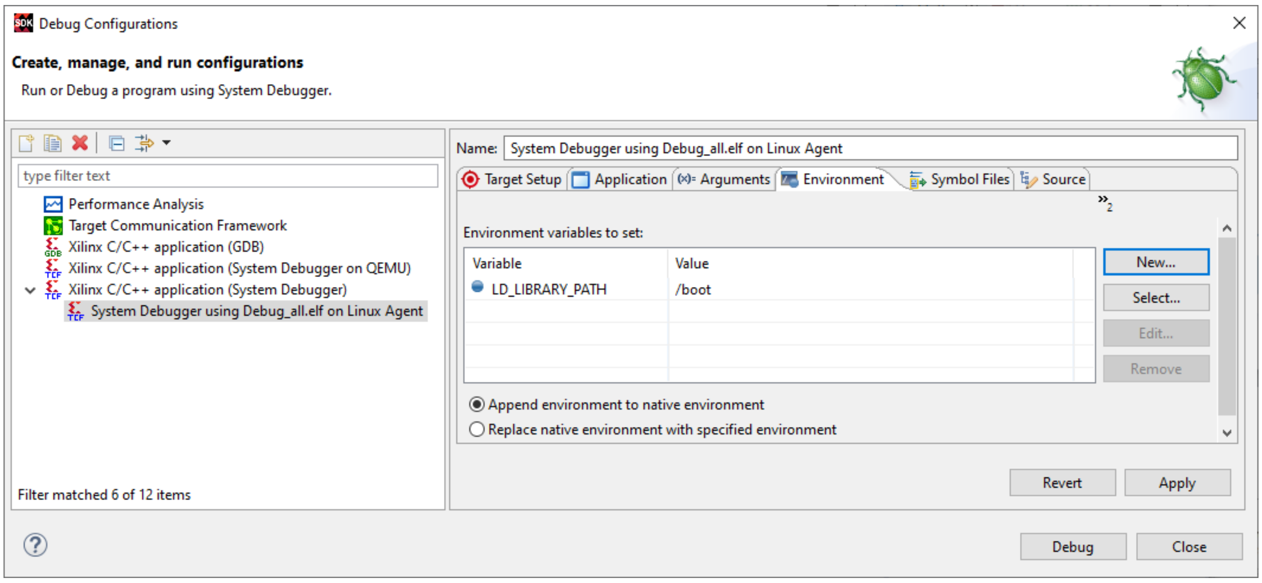
Xilinx SDK uses the workspace directory to store its preferences and development artifacts.

Workspace: X:\fp03x8_v26_4x2_mulf64_all_sw_c

☐ Use this as the default and do not ask again

Figure 12: Select SDK workspace.

SDK debugger needs environment information about the location of the actual shared library on the board. For example:


Debug Configurations

Create, manage, and run configurations

Run or Debug a program using System Debugger.

Name: System Debugger using Debug_all.elf on Linux Agent

Environment variables to set:

Variable	Value
LD_LIBRARY_PATH	/boot

(● Append environment to native environment) ○ Replace native environment with specified environment

type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (System Debugger on QEMU)
Xilinx C/C++ application (System Debugger)
System Debugger using Debug_all.elf on Linux Agent
Filter matched 6 of 12 items
Target Setup
Application
Arguments
Environment
Symbol Files
Source
New...
Select...
Edit...
Remove
Revert
Apply
Debug
Close

Figure 13: Define the environment variable.

Before start of Debug, binary copy the content of the sd_card directory associated to the project:

```
.\fp03x8_v26_4x2_mulf64_all_sw_c\Debug\sd_card\*.*
```

to Debian file system as
/boot/*.*

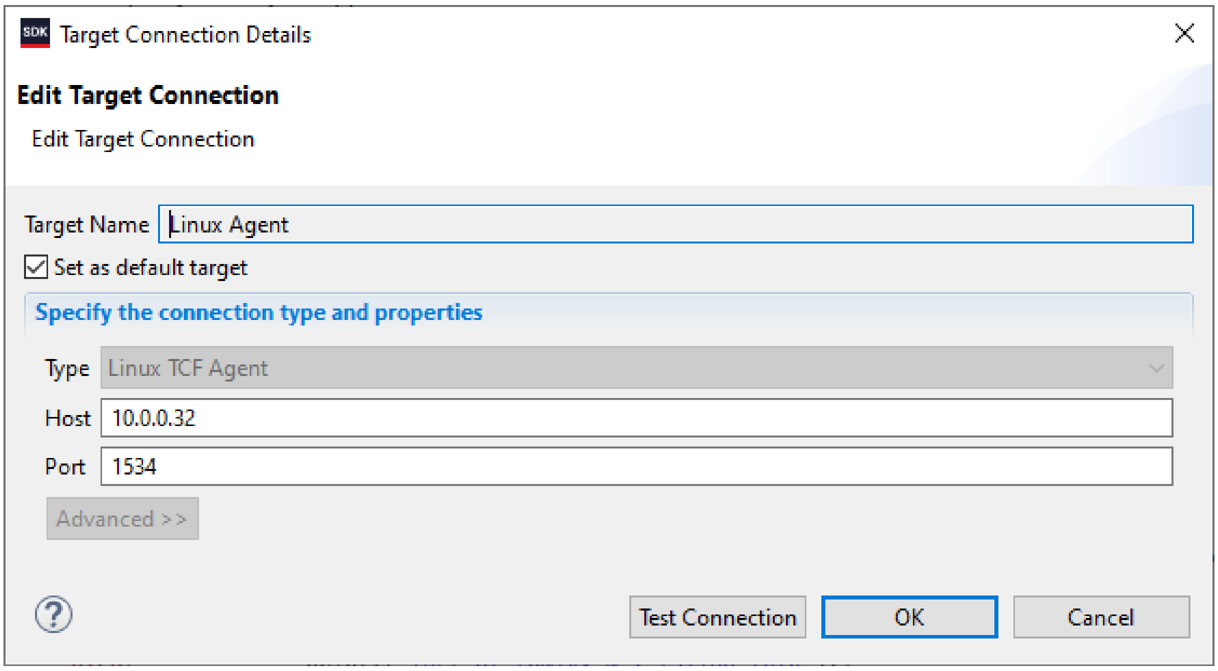
You can copy the content to the SD card on your PC.
Enter SD card to the board and power ON.

Alternatively, you can use Ethernet for perform binary copy to the SD card.
If you use Ethernet, you have to type

```
reboot
```

to reboot the board with correct bitstream loaded to the programmable logic part of the Zynq Ultrascale+.

To debug from the PC in the Xilinx SDK debugger GUI, the Zynq Ultrascale+ TCF server has to be accessible from the PC via Ethernet.


Target Connection Details

Edit Target Connection

Edit Target Connection

Target Name: Linux Agent

[✓] Set as default target

Specify the connection type and properties

Type: Linux TCF Agent

Host: 10.0.0.32

Port: 1534

Figure 14: Test connection to Linux TCF Agent.

Compile SW application directly on the Zynq Ultrascale+ board

Xilinx SDK 2017.4 tool creates files for the make utility, which can be used for compilation of SW application directly on the board with use of the gcc C compiler or the g++ C++ compiler of the Arm Debian OS.

You can copy complete SDK 2017.4 project to the Debian file system and compile on board by copy complete content of the C or C++ directory with SDK projects. Example of C++:

```
.\fp03x8_v26_4x2_mulf64_all_sw
```

to the Debian file system into directory:

```
/home/fp03x8_v26_4x2_mulf64_all_sw
```

To compile in Debian the **`all`** project (C++), change the directory to:

```
cd /home/fp03x8_v26_4x2_mulf64_all_sw/all/Debug
```

and export the relative path to the Debug version of the shared library by typing in Debian console:

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

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

```
make clean
make
```

Finally, execute the re-compiled C++ debug version of the application compiled by ARM Debian g++ compiler by typing in the Debian console:

```
./all.elf
```

You are done.

To close the Debian OS, type in the Debian terminal:

```
halt
```

This will close all open files on the SD file system an halt the ARM system.

Press the S1 button on the TEBF0808 carrier board to stop the power supply on the board. The ventilator will stop.

Now you can safely remove the SD card. PC terminal remains connected.

You can modify the SD card in PC and continue with next test.

To close all work, you can close the PC terminal and then completely power down the board.

Guide for compilation and use of C MEX functions in Scilab

In Debian terminal, change directory to

```
/home/zu9_es_deb_eval_ila_release_scilab/cc/mmultf
```

Start Scilab by typing

```
scilab
```

In Scilab execute script

```
mmultf_cc.sce
```

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

```
quit
```

Copy created shared library **`libmex_mmultf.so`**

`/home/zu9_es_deb_eval_ila_release_scilab/cc/mmultf/mmultf.so`
to
`/home/zu9_es_deb_eval_ila_release_scilab/test/test_mmultf_4xB/mmultf.so`

In Debian terminal, change directory to

```
/home/zu9_es_deb_eval_ila_release_scilab/test/test_mmultf_4xB
```

Start Scilab by typing in the terminal

`scilab`

In Scilab, execute script

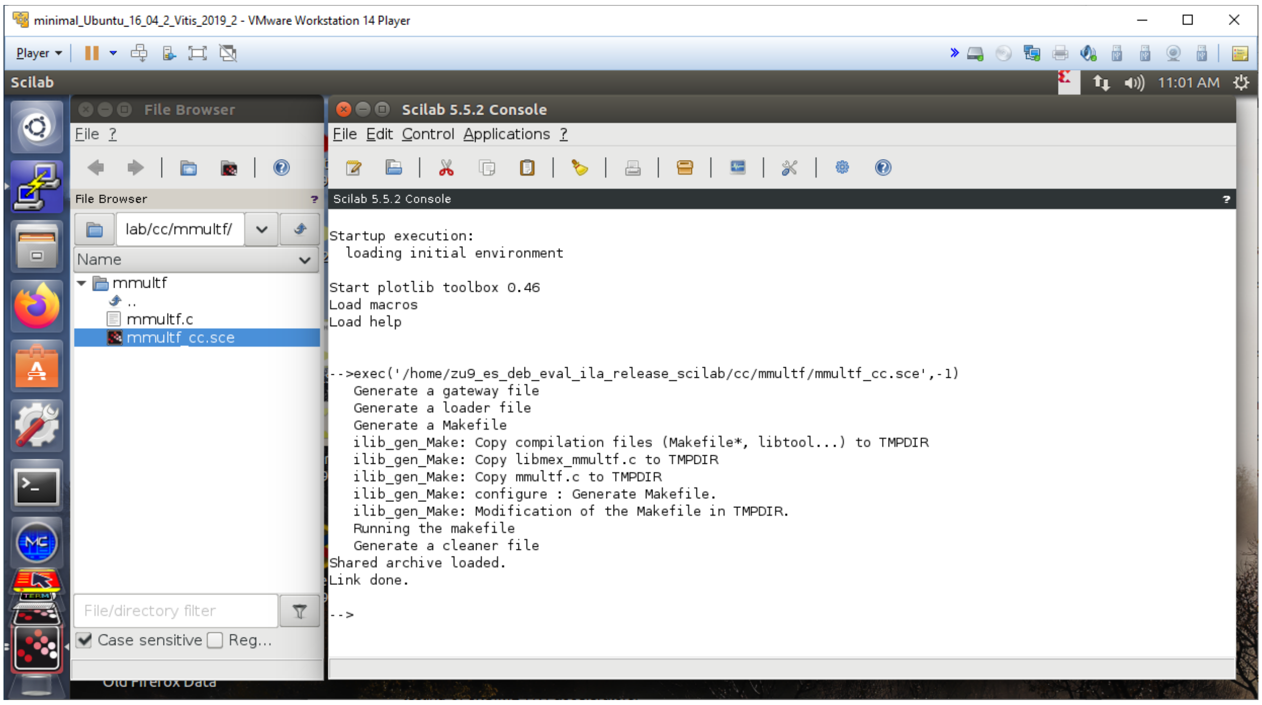
`mmultf_4xB_test.sce`

This script will dynamically load shared library **`libmex_mmultf.so`** , call **`mmultf`** C MEX function and generate reference header files with single precision floating point data used for testing of 8xSIMD HW accelerators.

Quit Scilab by typing

`quit`

Use same process to compile and use all other reference MEX C functions. See Figure 15.



```
Startup execution:
  loading initial environment

Start plotlib toolbox 0.46
Load macros
Load help


-->exec('/home/zu9_es_deb_eval_ila_release_scilab/cc/mmultf/mmultf_cc.sce',-1)
   Generate a gateway file
   Generate a loader file
   Generate a Makefile
   ilib_gen_Make: Copy compilation files (Makefile*, libtool...) to TMPDIR
   ilib_gen_Make: Copy libmex_mmultf.c to TMPDIR
   ilib_gen_Make: Copy mmultf.c to TMPDIR
   ilib_gen_Make: configure : Generate Makefile.
   ilib_gen_Make: Modification of the Makefile in TMPDIR.
   Running the makefile
   Generate a cleaner file
Shared archive loaded.
Link done.

-->
```

Figure 15: C MEX compilation in Arm Scilab. Remote X11 Desktop.

11 APPENDIX – DTRiMC tool guidelines

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

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

Name of the IP:	fp03x8_v26_v40
ID:	2
Device:	xczu9eg-ffvc900-1-i-es1
Tool chain:	Vivado/SDSoC 2017.4 ES1 enabled
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 IP is not included in the evaluation package in required HDL source code. The compiled evaluation version of the fp03x8_v26_v40 IP is present in the BOOT.bin files in sd_card directories of the evaluation package.

The evaluation package can be downloaded from UTIA for free from 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\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila\`
   Change directory to:
   ```
   c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila\zusys\
   ```
2. Add the UTIA 8xSIMD HW IP to the package as the directory \ip
3. `c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila\zusys\ip_lib\ip\`
4. On Win10, open dos terminal window, change directory to the folder
5. `c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila\zusys\`
6. 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:
   ```
   X:
   cd zusys
   ```
7. 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=2
   ```
   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
   ```

8. Start the Xilinx Vivado 2017.4 and create the design by executing of script:
   ```
   X:\zusys\vivado_create_project_guimode.cmd
   ```
9. Optional:
   You can use Vivado automation to add ILA monitor to enable ILA capturing of selected accelerator outputs of your choice.
10. In Vivado console, execute command:
   ```
   TE::hw_build_design -export_prebuilt
   ```
   After the Vivado compilation, new hardware description file *zusys.hdf* is generated in folder:
   ```
   X:\zusys\prebuilt\hardware\15eg_1eb_sk\zusys.hdf
   ```

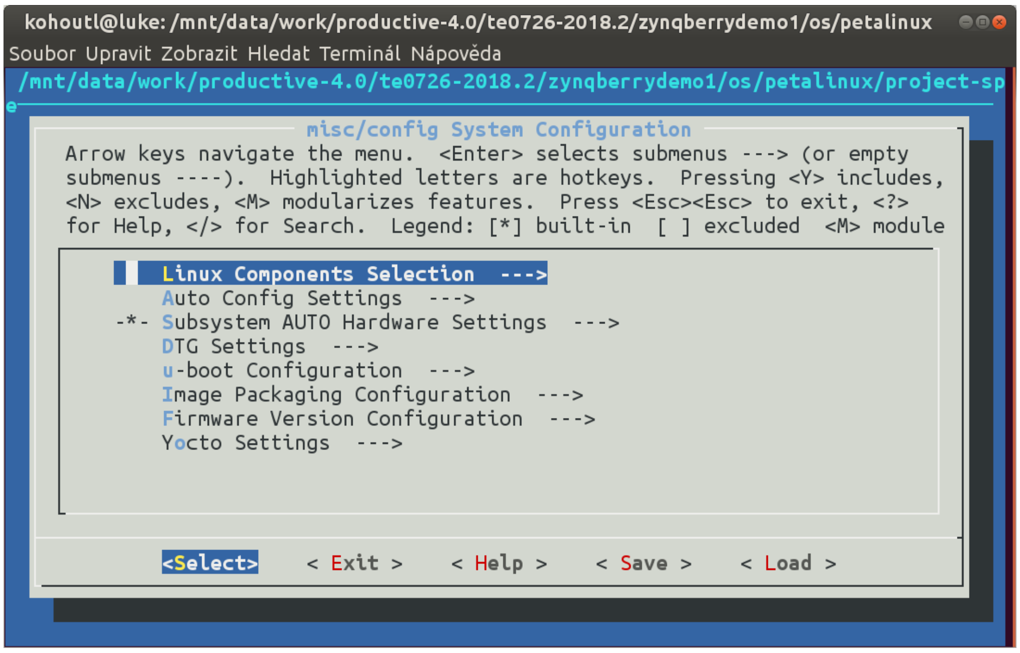
Guide for configuration and compilation of PetaLinux in the DTRiMC tool

The configuration and compilation of the *Petalinux 2017.4* kernel and *Debian 9.8 Stretch* image as the FitOptiVis run time resource for the Zynq Ultrascale+ module TE0808-ES1 with xczu9eg-ffvc900-1-i-es1 device is described now. The configuration is performed on the Ubuntu 16.04 LTS.

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-v2017.4-final
```
The standard PetaLinux 2017.4 distribution requires few modifications.

1. Copy content of these Win 10 directories:
   ```
   X:\zusys\prebuilt
   X:\zusys\os
   ```
   to Ubuntu directories:
   ```
   /home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/
   /home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/prebuilt
   ```
2. In Ubuntu, open terminal window and set path to the PetaLinux 2018.2:
   ```
   source /opt/petalinux/petalinux-v2017.4-final/settings.sh
   ```
3. Go to the directory copied from the evaluation package with pre-defined configuration for the Zynq Ultrascale+ module TE0808-03-15EG-1EE:
   ```
   cd
   /home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/petalinux
   ```
   It contains a predefined configuration according to Zynq Ultrascale+ board requirements.
4. The zusys.hdf file created in Win 10 in Vivado 2018.2 tool is present in the Ubuntu folder:
5. /home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/prebuilt/hardware/es1_sk/zusys.hdf
6. Use the zusys.hdf file as input fo the PetaLinux configuration by (on single line)

```
petalinux-config --get-hw-description=/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/prebuilt/hardware/es1_sk/
```



```
kohoutl@luke: /mnt/data/work/productive-4.0/te0726-2018.2/zynqberrydemo1/os/petalinux
Soubor Upravit Zobrazit Hledat Terminál Nápověda
 /mnt/data/work/productive-4.0/te0726-2018.2/zynqberrydemo1/os/petalinux/project-sp
e
                        misc/config System Configuration
    Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
    submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
    <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>
    for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module

            Linux Components Selection  --->
            Auto Config Settings  --->
        -*- Subsystem AUTO Hardware Settings  --->
            DTG Settings  --->
            u-boot Configuration  --->
            Image Packaging Configuration  --->
            Firmware Version Configuration  --->
            Yocto Settings  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

7. Verrify 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)  --->
```

8. 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*.

9. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

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

```
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/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.

11. Go to the folder with PetaLinux:

```
cd /home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/petalinux/
```

12. The 64bit 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.

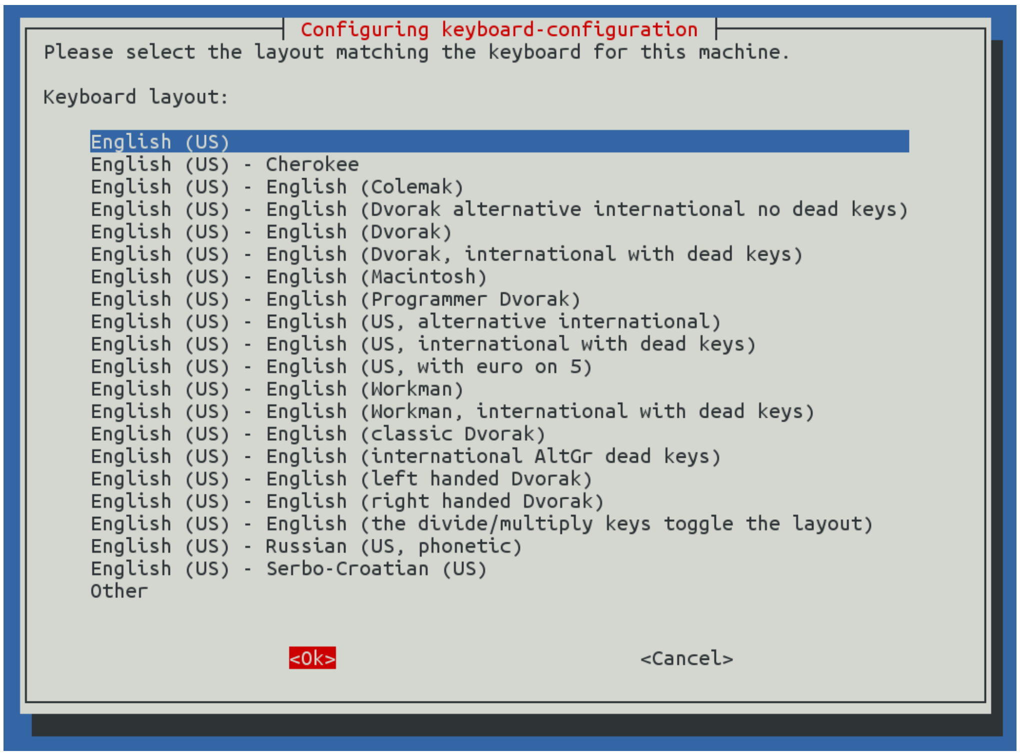
Tool	*Package*
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-arm-static	qemu-user-static

13. 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.


```
Configuring keyboard-configuration
Please select the layout matching the keyboard for this machine.

Keyboard layout:

    English (US)
    English (US) - Cherokee
    English (US) - English (Colemak)
    English (US) - English (Dvorak alternative international no dead keys)
    English (US) - English (Dvorak)
    English (US) - English (Dvorak, international with dead keys)
    English (US) - English (Macintosh)
    English (US) - English (Programmer Dvorak)
    English (US) - English (US, alternative international)
    English (US) - English (US, international with dead keys)
    English (US) - English (US, with euro on 5)
    English (US) - English (Workman)
    English (US) - English (Workman, international with dead keys)
    English (US) - English (classic Dvorak)
    English (US) - English (international AltGr dead keys)
    English (US) - English (left handed Dvorak)
    English (US) - English (right handed Dvorak)
    English (US) - English (the divide/multiply keys toggle the layout)
    English (US) - Russian (US, phonetic)
    English (US) - Serbo-Croatian (US)
    Other

                 <Ok>                          <Cancel>
```

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

```
zip te0808-debian-stretch te0808-debian-stretch.img
```

15. Copy compressed image file from Ubuntu

```
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys
/os/petalinux/te0808-debian-stretch.zip
```

to Win 10 file:

```
X:\zusys\prebuilt\os\petalinux\default\ te0808-debian-stretch.zip
```

16. Copy these files from Ubuntu

```
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys
/os/petalinux/images/linux/image.ub
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/
petalinux/images/linux/u-boot.elf
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys/os/
petalinux/images/linux/bl31.elf
```

to Win 10 files:

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

17. In Ubuntu, clean Petalinux project files

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

18. In Ubuntu, delete files

```
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys
/os/petalinux/te0808-debian-stretch.zip
/home/devel/work/TS74fp03x8_TEBF0808_DTRiMC_zu9es1/zu9_es_deb_eval_ila/zusys
/os/petalinux/te0808-debian-stretch.img
```

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

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

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

- Petalinux kernel image *image.ub*
- Compressed Debian image *TE0808-debian.zip*
- U-boot program *u-boot.elf*
- Support firmware *bl31.elf*

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

Guide for creation of SDSoC platform OS in the DTRiMC tool

1. 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:\zusys\prebuilt\boot_images\ es1_sk\u-boot\BOOT.bin
   ```
2. 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 to the directory
   `X:\SDSoC_PFM\TE0808\ES1`
   and it is also packed into the ZIP file.

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

1. On Win10, i*n the open dos terminal window, cancel the current virtual drive X: by executing from* the command line
   ```
   _use_virtual_drive.cmd
   ```
   and response (1)
2. Change directory to
   `c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila\SDSoC_PFM\TE0808\ES1\`
3. 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:\ES1`
   ```
   _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 ES1
   ```
4. Open SDSoC project in directory
   ```
   X:\ES1
   ```
5. In SDSoC import two HW kernel design projects
   `fp03x8_v26_4x2_c_mulf64_hw`
   and
   `fp03x8_v26_4x2_mulf64_hw`
   from the directory
   `c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila\SDSoC_PFM_src\TE0808\ES1\`

6. Define the custom SDSoC platform
   `X:\ES1\zusys`
7. Change both imported projects from Debug to the Release compilation target
8. Compile both projects by the SDSoC compiler
9. Result of compilation are the SD cards with BOOT.bin file shared object library definition files in directories:

   X:\ES1\fp03x8_v26_4x2_c_mulf64_hw\Release\sd_card\

   X:\ES1\fp03x8_v26_4x2_mulf64_hw\Release\sd_card\
10. Copy content of the directory

    X:\ES1\fp03x8_v26_4x2_c_mulf64_hw\Release\sd_card\

    to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw_c\Release\sd_card\
    and also to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw_c\Debug\sd_card\
11. Copy content of the directory

    X:\ES1\fp03x8_v26_4x2_mulf64_hw\Release\sd_card\

    to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw\Release\sd_card\
    and also to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw\Debug\sd_card\
12. Optional:

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

    x:\ ES1\ fp03x8_v26_4x2_c_mulf64_hw \Release_sds\p0\vivado\prj\prj.runs\impl_1\

    to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw_c\Release\sd_card\
    and also to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw_c\Debug\sd_card\

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

    x:\ES1\ fp03x8_v26_4x2_mulf64_hw \Release_sds\p0\vivado\prj\prj.runs\impl_1\

    to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw\Release\sd_card\
    and also to

    c:\home\work\TS74fp03x8_TEBF0808_DTRiMC_zu9es1\zu9_es_deb_eval_ila_release\fp03x8_v26_4x2_mulf64_all_sw\Debug\sd_card\
13. Clean both projects
14. Close SDSoC tool

Guide for retargeting of the DTRiMC tool for another device/module

The DTRiMC tool is configured for Trenz Electronic module with ID=2, TE0808-ES1, with device xczu9eg-ffvc900-1-i-es1, memory 2GB and short module name es1_sk.

Hovewer, the DTRiMC tool scripts can be modified to target different Trenz Electronic module and device. See the actual list of Trenz Electronic modules in:

https://shop.trenz-electronic.de/en/Products/Trenz-Electronic/TE08XX-Zynq-UltraScale/TE0808-Zynq-UltraScale/

Use text editor of your choice and open and modify script *design_basic_settings.sh*

Modify ID of the module/device:

```
@set PARTNUMBER=2
```

Trenz Electronic modules supported by this release of the DTRiMC tool.

ID	Module	Partname	Memory	ShortName
2	**TE0808-ES1**	**xczu9eg-ffvc900-1-i-es1**	**2GB**	**es1_sk**
4	TE0808-ES2	xczu9eg-ffvc900-1-i-es2	2GB	es2_sk
6	TE0808-2ES2	xczu9eg-ffvc900-2-i-es2	2GB	2es2_sk
8	TE0808-04-09EG-1EA	xczu9eg-ffvc900-1-e	2GB	9eg_1ea_sk
10	TE0808-04-09EG-1EB	xczu9eg-ffvc900-1-e	4GB	9eg_1eb_sk
12	TE0808-04-09EG-1ED	xczu9eg-ffvc900-1-e	4GB	9eg_1eb_sk
14	TE0808-04-09EG-2IB	xczu9eg-ffvc900-2-i	4GB	9eg_2ib_sk
16	TE0808-04-15EG-1EB	xczu15eg-ffvc900-1-e	4GB	15eg_1eb_sk
18	TE0808-04-09EG-1EE	xczu9eg-ffvc900-1-e	4GB	9eg_1eb_sk
20	TE0808-04-09EG-1EL	xczu9eg-ffvc900-1-e	4GB	9eg_1eb_sk
22	TE0808-04-09EG-2IE	xczu9eg-ffvc900-2-i	4GB	9eg_2ib_sk
24	TE0808-04-15EG-1EE	xczu15eg-ffvc900-1-e	4GB	15eg_1eb_sk
26	TE0808-04-06EG-1EE	xczu6eg-ffvc900-1-e	4GB	6eg_1ee_sk
28	TE0808-04-06EG-1E3	xczu6eg-ffvc900-1-e	4GB	6eg_1ee_sk

After the change of the target, the DTRiMC tool requires also change of the input 8xSIMD HW IP core. Contact UTIA to get license for the required HW IP version.

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

Name of the IP:	fp03x8_v26_v40
ID:	*Select ID*
Device:	*Select partname*
Tool chain:	Vivado/SDSoC 2017.4 ES enabled
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

Implement all design steps with the DTRiMC tool for the retargeted module/device.

Disclaimer