Application Note

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

DTRiMC tool for TE0808-15-EG-1EE module on TEBF0808 carrier board

Jiři Kadlec, Zdenek Pohl, Lukas Kohout

Revision history

Rev.	Date	Author	Description
0	2.03.2021	J. Kadlec	Initial draft
1	5.07.2021	J. Kadlec	DTRiMC tool for TE0808-15-EG-1EE
2	11.07.2021	J. Kadlec	Support for wider range of modules

Table of Contents

1 DTRiMC tool for TE0808-15-EG-1EE....1
2 8xSIMD FP03x8 floating point accelerators for ZU15-EG-1EE....2
3 Programming of 8xSIMD FP03x8 floating point accelerators....8
HW version of matrix multiplication and LK Dense Optical Flow....10
4 C++ evaluation project....12
5 Power consumption....15
6 ILA – In-circuit Logic Analyzer....15
7 License....17
8 Conclusion....17
Reconfiguration of accelerator by change of firmware....18
Reconfiguration of accelerator by temporary change of firmware....19
9 References....19
10 APPENDIX - Confidence test....20
Compilation and debug of projects from source code....21
DEBUG of SW application from Xilinx SDK 2018.2....21
Guide for compilation and use of C MEX functions in Scilab....24
11 APPENDIX – DTRiMC tool guidelines....26
Guide for compilation of HW in the DTRiMC tool....26
Guide for configuration and compilation of PetaLinux in the DTRiMC tool....27
Guide for configuration and compilation of Debian OS in the DTRiMC tool....28
Guide for creation of SDSoC platform OS in the DTRiMC tool....31
Guide for creation of shared library and HW kernel in the DTRiMC tool....31
Guide for retargeting of the DTRiMC tool for another device/module....33
Disclaimer....34

Table of Figures

Figure 1: Design Time Resource integration of Model Composer DTRiMC tool....1
Figure 2: Sobel edge detection and Eight FP03x8 accelerators in ZU15-EG-1EE device....3
Figure 3: LK Dense Optical Flow and eight FP03x8 accelerators in ZU15-EG-1EE device....4
Figure 4: FP03x8 accelerator for the ZU15-EG-1EE device....5
Figure 5: Internal block rams of accelerators....6
Figure 6: Floating point functions present in all accelerators {10 or 20 or 30 or 40}....8
Figure 7: Specific functions present only in some versions accelerators....8
Figure 8: Structure of the 128 bit wide VLIW program instruction....9
Figure 9: HW accelerators in parallel with Sobel Edge Detection....10
Figure 10: HW accelerators in parallel with LK Dense Optical Flow....11
Figure 11: Performance results of sobel_all.elf application....14
Figure 12: Instruction vz2a....16
Figure 13: Instruction vz2a detail....16
Figure 14: Select debug workspace....21
Figure 15: Define the environement variable....22
Figure 16: Test connection to Linux TCF Agent....23
Figure 17: C MEX compilation in Arm Scilab. Remote X11 Desktop....25

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 TE0808-15-EG-1EE

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+ TE0808-15EG-1EE module [1] on TEBF0808 carrier board [2].

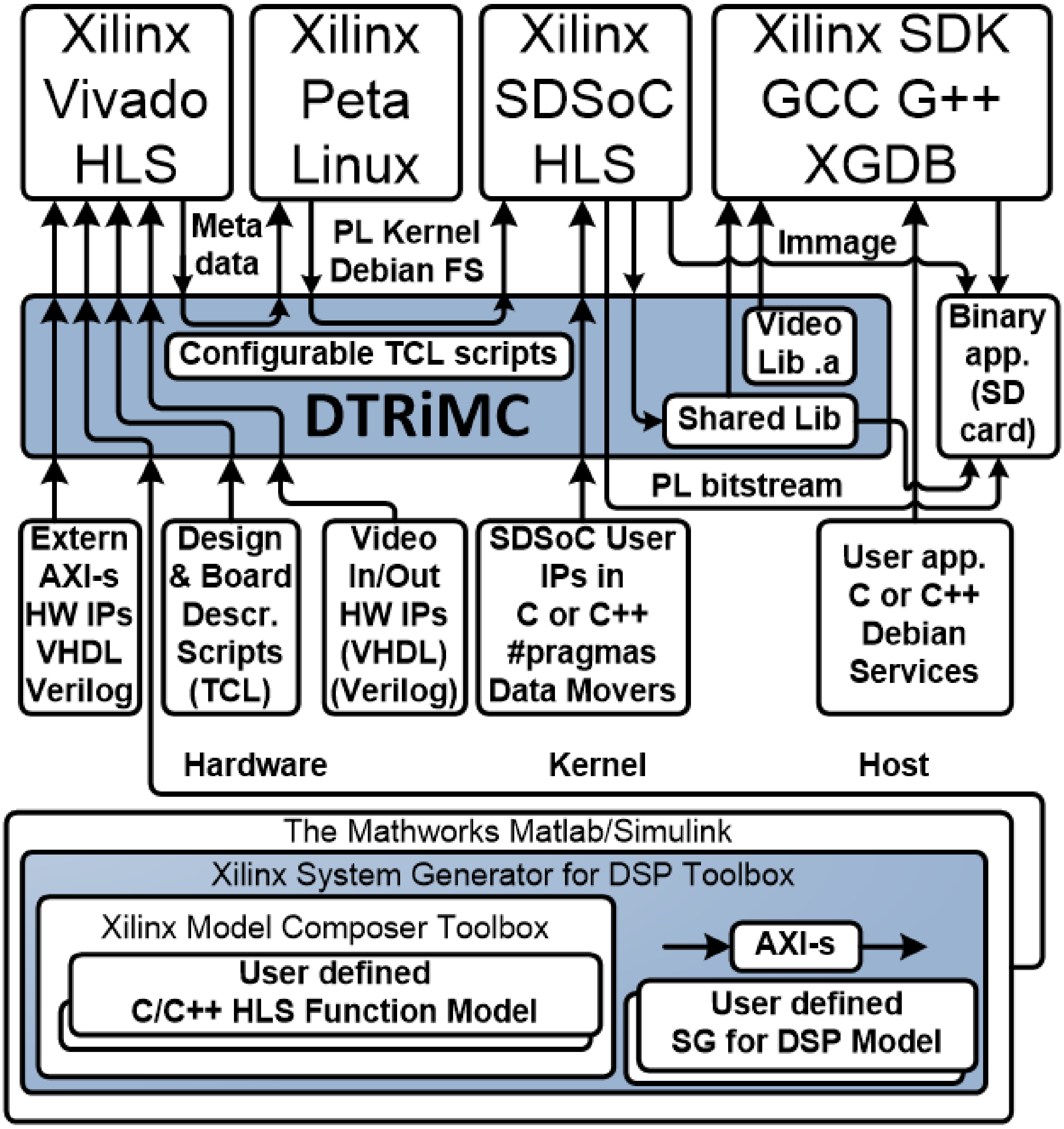


Figure 1: Design Time Resource integration of Model Composer DTRiMC tool.

This application note describes an evaluation package with eight 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerators for Zynq Ultrascale+ TE0808-15EG-1EE module [1] on TEBF0808 carrier board [2]. The TE0808-15EG-1EE module and TEBF0808 carrier board are designed and manufactured by the company Trenz Electronic [1]. Xilinx device ZU15-EG-1EE device requires in the design phase Xilinx Vivado tools version 2018.2. These tools must have enabled support for the Xilinx ZU15-EG-1EE device. The Xilinx Vivado 2018.2 is currently the last Xilinx toolchain supporting the ZU15-EG-1EE device.

The evaluation package provides several pre-compiled HW designs (see Figure 2 and Figure 3) represented in form of SD-cards containing the designs and API interface for SW developer in form of shared Debian Linux libraries. The SW developer can program ARM host application in standard gcc or g++ compiler and „make“ can be used for compilation of host applications directly on the embedded Zynq Ultrascale+ ZU15-EG-1EE based system.

2 8xSIMD FP03x8 floating point accelerators for ZU15-EG-1EE

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 4.

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.

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 DMA transaction from ARM processor.
- AXI stream data/program output from HW accelerator to ARM. The output side channel indicates the last transferred word sent from the component to Arm processor.
- AXI-lite input/output configuration registers.

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.

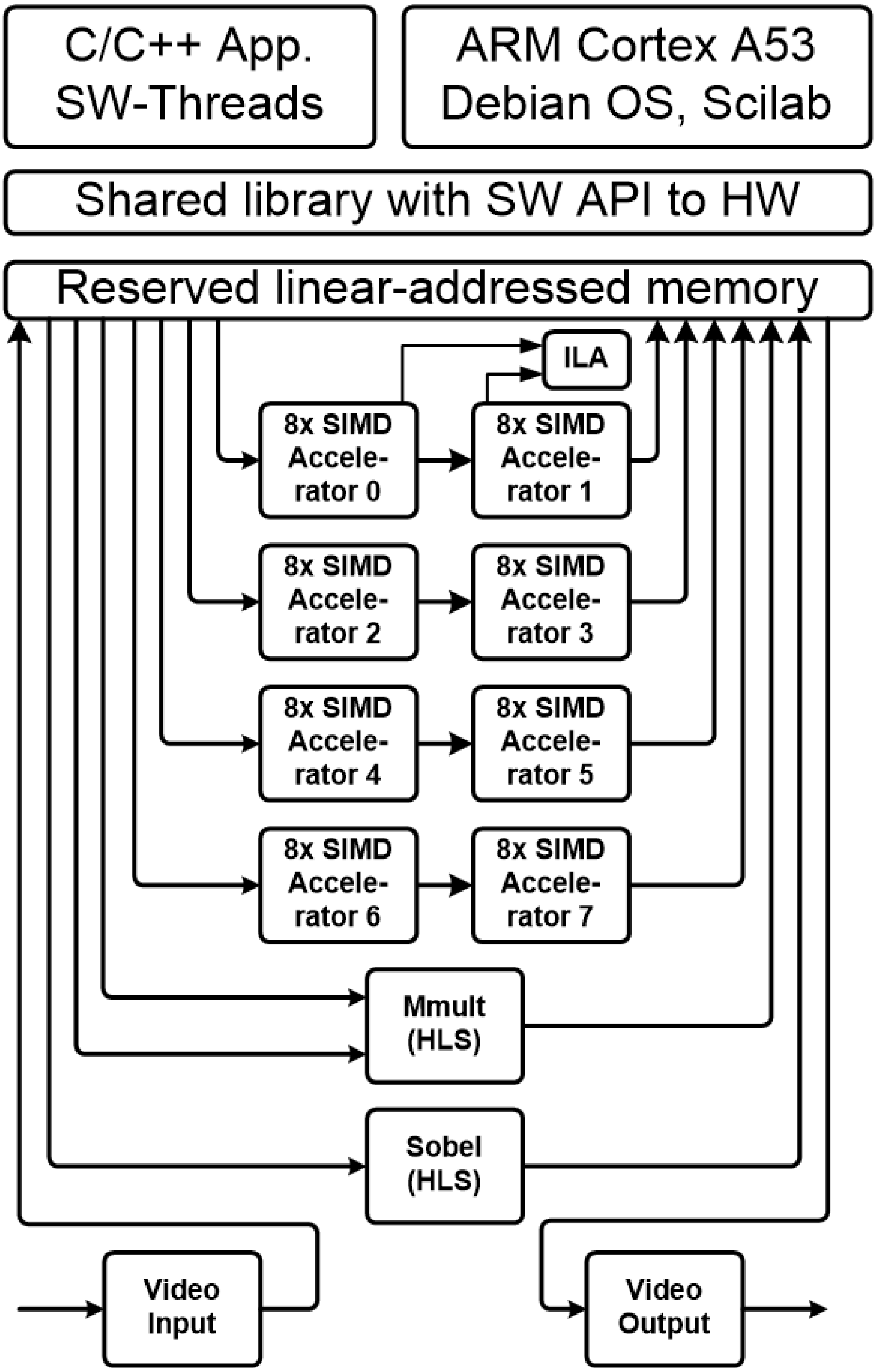


Figure 2: Sobel edge detection and Eight FP03x8 accelerators in ZU15-EG-1EE device

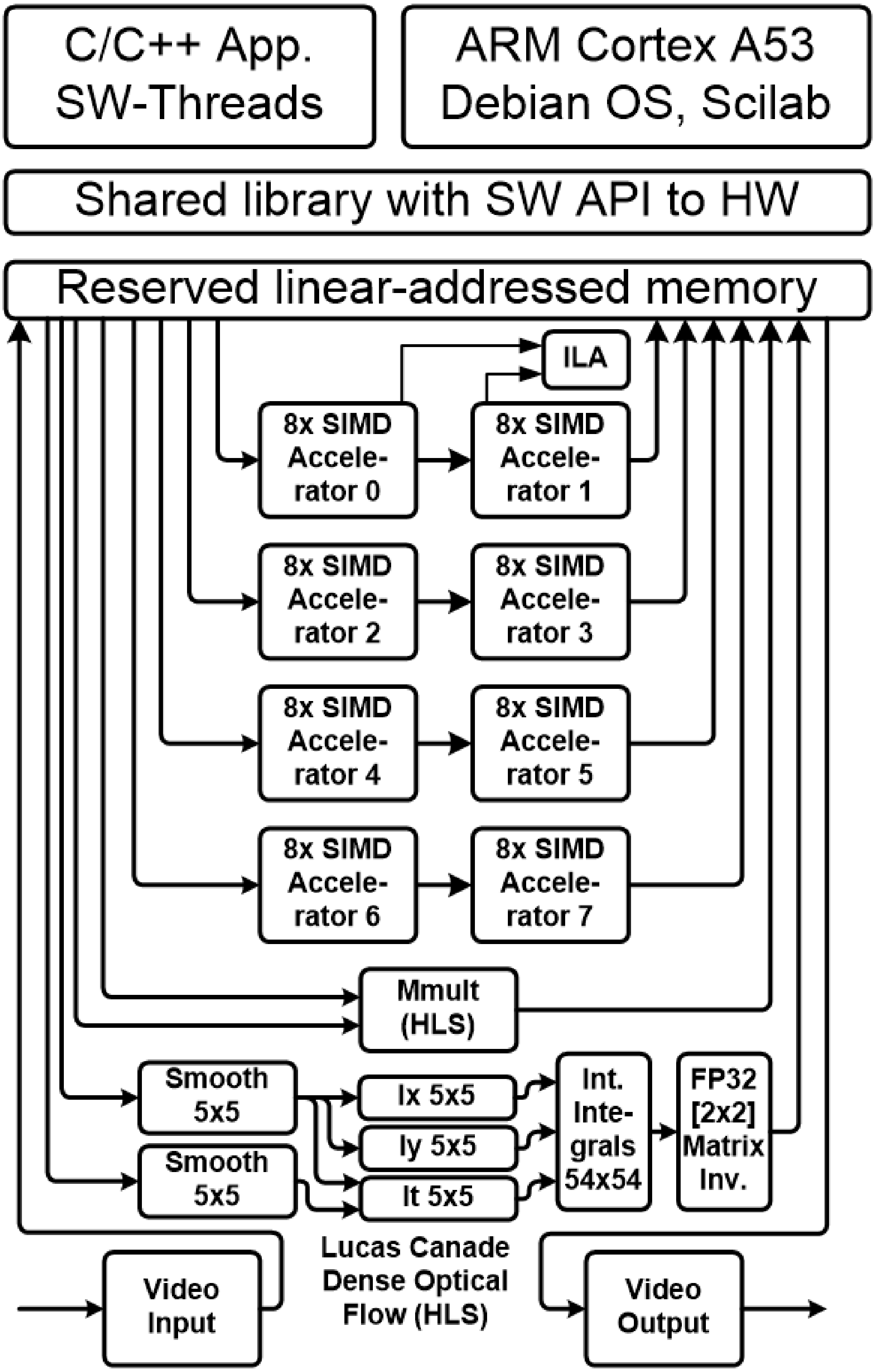


Figure 3: LK Dense Optical Flow and eight FP03x8 accelerators in ZU15-EG-1EE device

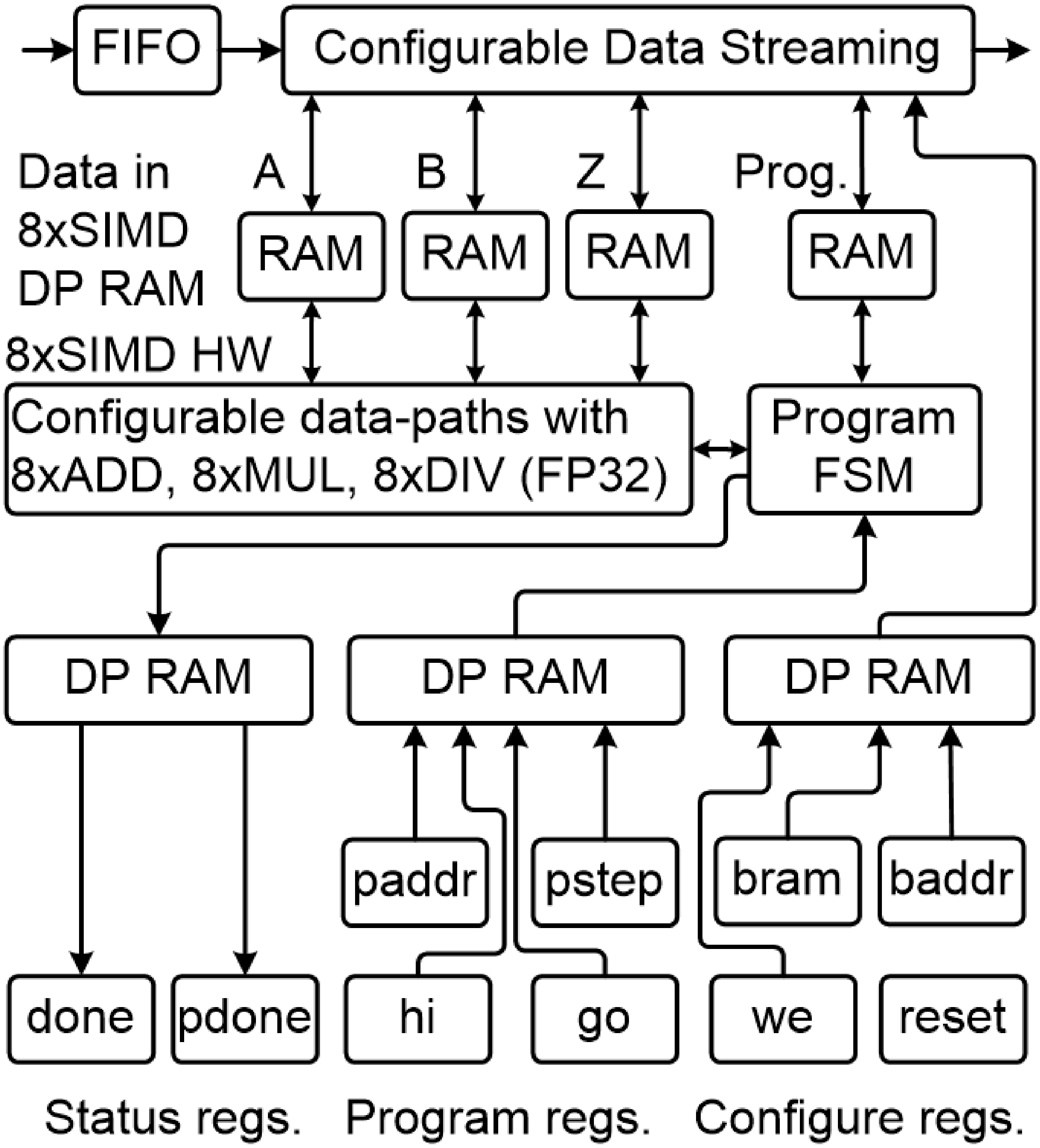


Figure 4: FP03x8 accelerator for the ZU15-EG-1EE device

Accelerator Interfaces

Type of interface:	Device:	Clock:
• Data streaming I/O: AXI-S 32 bit	ZU15-EG-1EE	250 MHz
• Firmware program VLIW 128 bit	ZU15-EG-1EE	250 MHz
• Configuration I/O: AXI-lite 32 bit	ZU15-EG-1EE	100 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 5: 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 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 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 6: 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 7: 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. Components of the low 64 bit word are marked by light green background. Components of the high 64 bit word components are marked by light blue. See Figure 8.

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 8: 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 4).

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 2018.2 gdb debugger GUI.

HW version of matrix multiplication and LK Dense Optical Flow

Four chains of accelerators {0,1}, {2,3}, {4,5} and {6,7} are independent and can be configured and executed in threads started from the user host application. Threads can run in parallel under control of user application kern on four cores of the Arm A53 processor. HW accelerated matrix multiplications run in parallel with HW accelerated LK Dense Optical Flow.

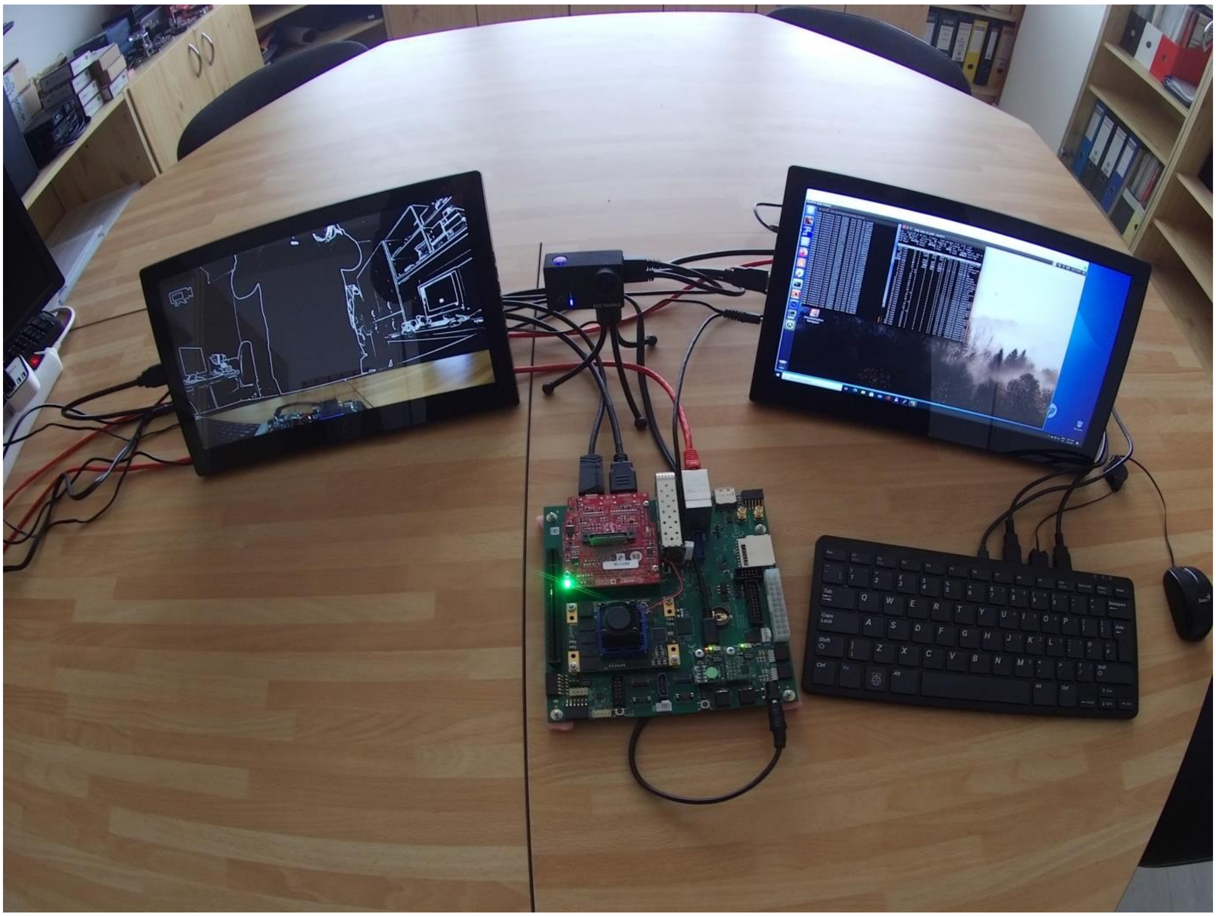


Figure 9: HW accelerators in parallel with Sobel Edge Detection

The Sobel edge detection demo with SG DMA presented in Figure 9 is started from Zynq command line by typing:

```
cd /boot
export LD_LIBRARY_PATH=/boot
./sobel_all.elf
```

Demo is linearry increasing the programmable count of micro lines from ¼ of the frame to the full frame. Part of the image which is not processed by edge detection is propagated via video frame buffers to the output screen.

Demo can be stopped by Ctrl-C.

The while (!done) serves for detection of the “Ctrl C” user-defined break from the computing loop. The optflow_processing() function computes the LK Dense Optical Flow for Full HD 60 FPS video.

The parallel computation of matrix multiplication is started after the return from the optflow_processing() function. The FULL HD 60 FPS frame corresponds to 16ms.

If optflow_processing() function is implemented with SG DMA data movers it requires cca 10 ms. The 4x2 FP03x8 8xSIMD accelerators can perform 32x8= 256 Matrix Multiplications in remaining 6 ms.

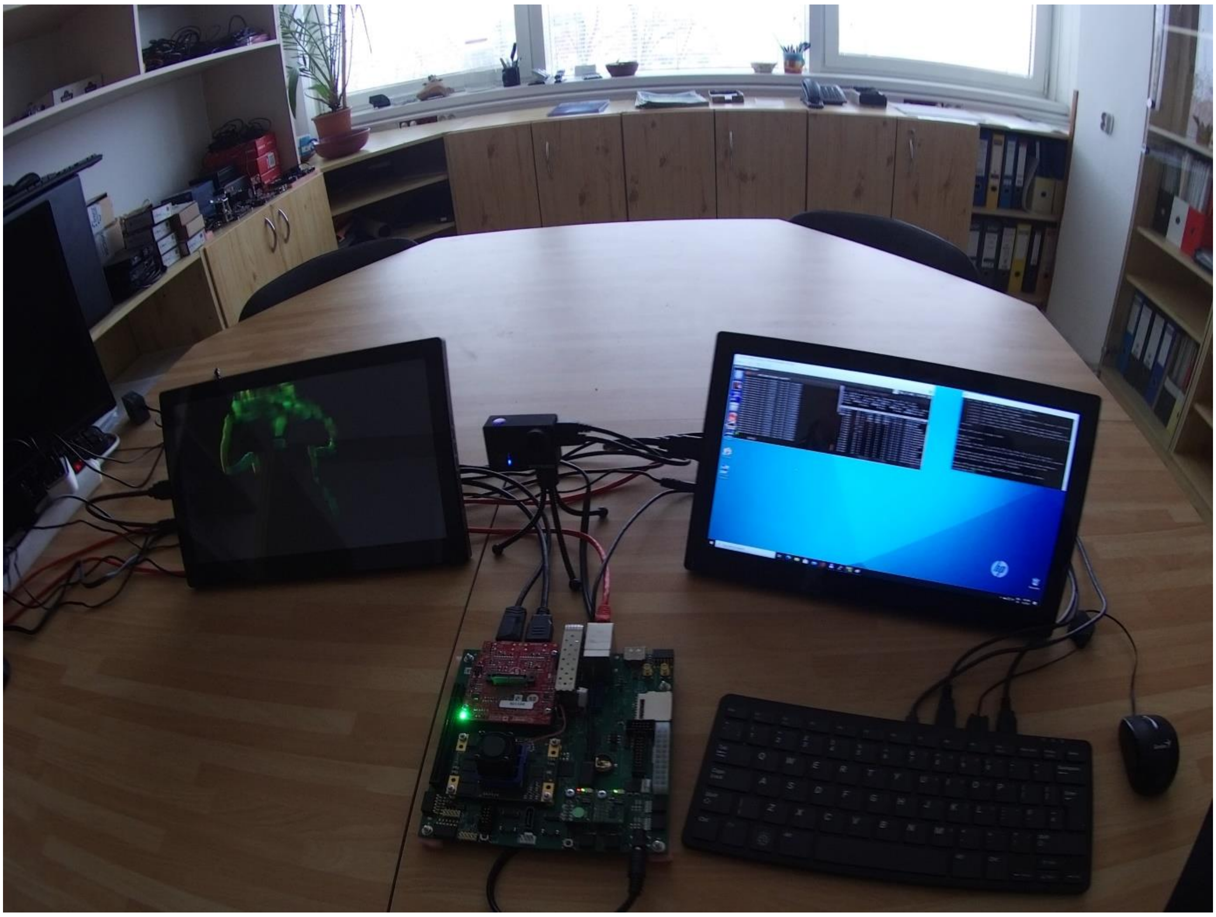


Figure 10: HW accelerators in parallel with LK Dense Optical Flow

The Lucas Kanade Dense Optical Flow demo with SG DMA (presented in Figure 10) is started from Zynq command line by typing:

```
cd /boot
export LD_LIBRARY_PATH=/boot
./dof_all.elf
```

Demo can be stopped by Ctrl-C.

4 C++ evaluation project

The PL part of the ZU15-EG-1EE device contains eight evaluation versions of the 8xSIMD run-time-reprogrammable single-precision-floating-point HW accelerator FP03x8 organized as 4x2 accelerators. Released four evaluation HW platforms exported for linking with the arm A53 host programs have these properties:

- Sobel filter Edge detection with DMA data mover with 4x 8xSIMD HW acclerators
  Directory (C++): **te02_4x2_async_mulf64_sgdma_sw**
  SW C++ project: **sobel_all**
  SW C++ project: **sobel_all_var_frames**
  SW C++ project: **sobel_mmultf1_4xB**
  Shared C++ library: **./Debug/sd_card/ libte02_4x2_async_mulf64_sgdma_hw.so**
  Shared C++ library: **./Release/sd_card/ libte02_4x2_async_mulf64_sgdma_hw.so**

  Debug and Release version of shared libraries Debian Stretch 9.8 OS for the ZU15-EG-1EE device for the SDK 2018.2 C++ SW flow (g++ compiler). PL is working with 4x2 evaluation versions of FP03x8 HW accelerators with zero copy data movers.
  SDSoC matrix multiplication HW IP is connected by SG DMA.

- LK Dense Optical Flow with DMA data mover with 4x 8xSIMD HW acclerators
  Directory (C++): **te03_4x2_async_mulf64_sgdma_sw**
  SW C++ project: **dof_all**
  SW C++ project: **dof_mmultf1_4xB**
  Shared C++ library: **./Debug/sd_card/ libte03_4x2_async_mulf64_sgdma_hw.so**
  Shared C++ library: **./Release/sd_card/ libte03_4x2_async_mulf64_sgdma_hw.so**

  Debug and Release version of shared libraries Debian Stretch 9.8 OS for the ZU15-EG-1EE device for the SDK 2018.2 C++ SW flow (g++ compiler). PL is working with 4x2 evaluation versions of FP03x8 HW accelerators with zero copy data movers.
  SDSoC matrix multiplication HW IP is connected by SG DMA.

SW projects **sobel_all, sobel_all_var_frames** and **dof_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 vector operations of 8xSIMD HW accelerators. The eight instances of the FP03x8 accelerators on ZU15-EG-1EE device are controlled by 4 SW threads. HW accelerators accelerate the SW optimized (-O3) code executed on four A53 cores running with 1.05 GHz clock.

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

Comparison of matrix multiplication performance:

System	Function	MFLOPs
ZU15-EG-1EE	8x mmult use B1..B4, include parallel copy of B1..B4	19111
	8x mmult use B1..B8, include parallel copy of B1..B8	15417
	SDSoC HW mmult use B1..B8, include copy of A, B, Z	6366
	SW mmult Scilab MEX style, 4 threads	1364
	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:	1933

```
192.168.13.240 - PuTTY
Waiting for valid input video signal (requesting 1920x1080p)
HDMII cable plugged
Detected signal:
ADV7611 Video Input Information
        Video Input      = HDMI, Progressive
        Color Depth      = 8 bits per channel
        HSYNC Timing     = hav=1920, hfp=88, hsw=44(hsp=1), hbp=148
        VSYNC Timing     = vav=1080, vfp=04, vsw=05(vsp=1), vbp=036
        Video Dimensions = 1920 x 1080
        Pixel Clock      = 148.500000 MHz
        Frame rate       = 60.000000 FPS
Got 1920x1080p resolution, initializing
Initialize VDMA ... RX Init ... OK
Starting VDMA parking mode ...
Parking started

FPS:   91.27 mmultf1_sw  OK       1364.943 MFLOPs [==                   ]
FPS:   69.01 mmultf1     OK       6366.971 MFLOPs [=======              ]
FPS:   67.87 mmultf1_8B  OK      15417.249 MFLOPs [================     ]
FPS:   68.01 mmultf1_4B  OK      19111.201 MFLOPs [==================== ]
FPS:   70.77 va2bf1      OK       9392.350 MFLOPs [==========           ]
FPS:   69.71 vaddf1      OK       9282.141 MFLOPs [==========           ]
FPS:   68.30 vaddf1_az2b OK       9280.668 MFLOPs [==========           ]
FPS:   69.10 vaddf1_bz2a OK       9256.548 MFLOPs [==========           ]
FPS:   74.57 vb2af1      OK       9381.434 MFLOPs [==========           ]
FPS:   67.27 vdivf1      OK       9094.980 MFLOPs [==========           ]
FPS:   78.46 vmacf1      OK       1062.255 MFLOPs [==                   ]
FPS:   79.19 vmsubacf1   OK       1057.155 MFLOPs [==                   ]
FPS:   69.63 vmulf1      OK       9282.773 MFLOPs [==========           ]
FPS:   67.48 vmulf1_az2b OK       9229.378 MFLOPs [==========           ]
FPS:   68.69 vmulf1_bz2a OK       9242.343 MFLOPs [==========           ]
FPS:   78.32 vprodf1     OK       6778.990 MFLOPs [=======              ]
FPS:   55.69 vprods8f1   OK       6761.253 MFLOPs [=======              ]
FPS:   67.62 vsubf1      OK       9293.461 MFLOPs [==========           ]
FPS:   66.27 vsubf1_az2b OK       9276.677 MFLOPs [==========           ]
FPS:   66.58 vsubf1_bz2a OK       9232.290 MFLOPs [==========           ]
FPS:   70.75 vz2af1      OK       9339.185 MFLOPs [==========           ]
FPS:   72.55 vz2bf1      OK       8964.343 MFLOPs [=========            ]

FPS:   95.28 mmultf1_sw  OK       1396.383 MFLOPs [==                   ]
FPS:   69.26 ^Cmmultf1     OK       6347.718 MFLOPs [=======              ]
license OK
Memory device closed
Exiting ...
root@zynqmp:/boot#
```

Figure 11: Performance results of sobel_all.elf application.

Performance results are listed in Figure 11. It is terminal output from the sobel_all.elf application running on Arm. The FPS is value of maximal achievable FPS (sobel filter is computed in parallel to call to the eight SIMD accelerators in each frame.) The video input FPS is 60.0 FPS (see Figure 11). This is defined by the information received from the video input device. The video output is also fixed, 60 FPS and this is defined by the video output HW.

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 libte02_4x2_async_mulf64_sgdma_hw.so is present in the device. No user app is running	14,64
Linux system is running with all HW interfaced by the library libte02_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. **sobel_all.elf** is running. It performs HW accelerated edge detection and scrolls through all tests of all 8 8xSIMD hw accelerators. These tests are controlled by 4 SW threads. See Figure 11.	17,04
Linux system is running with all HW interfaced by the library libte02_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. **sobel_mmultf1_4xB.elf** is running. The application performs It performs HW accelerated edge detection and in parallel it also performs repeated tests of HW accelerated floating point matrix multiplications on 8 8xSIMD HW accelerators. These tests are controlled by 4 SW threads.	18,72
Linux system is running with all HW interfaced by the library libte03_4x2_async_mulf64_sgdma_hw.so is present in the device. No user app is running	14,64
Linux system is running with all HW interfaced by the library libte03_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. **dof_all.elf** is running. In each frame, the application performs first the HW accelerated dense optical flow controlled from single SW thread and then performs sequence of repeated tests of supported firmware operaions on all 8 8xSIMD HW accelerators.	17,52
Linux system is running with all HW interfaced by the library libte03_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. **dof_mmultf1_3xB.elf** is running. This application performs the HW accelerated dense optical flow controlled from single SW thread and in parallel it also performs repeated tests of HW accelerated floating point matrix multiplications on 6 8xSIMD HW accelerators. These tests are controlled by 3 SW threads. Two 8xSIMD HW accelerators are not used.	19,32

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 2018.2 ILA – In-circuit Logic Analyser.

It is connected to HW 8xSIMD accelerators 0 and 1. ILA can be triggred by specific instruction and displays addresses and we signals with 250 MHz clock. It is configured to sample 1024 data. See Figure 2.

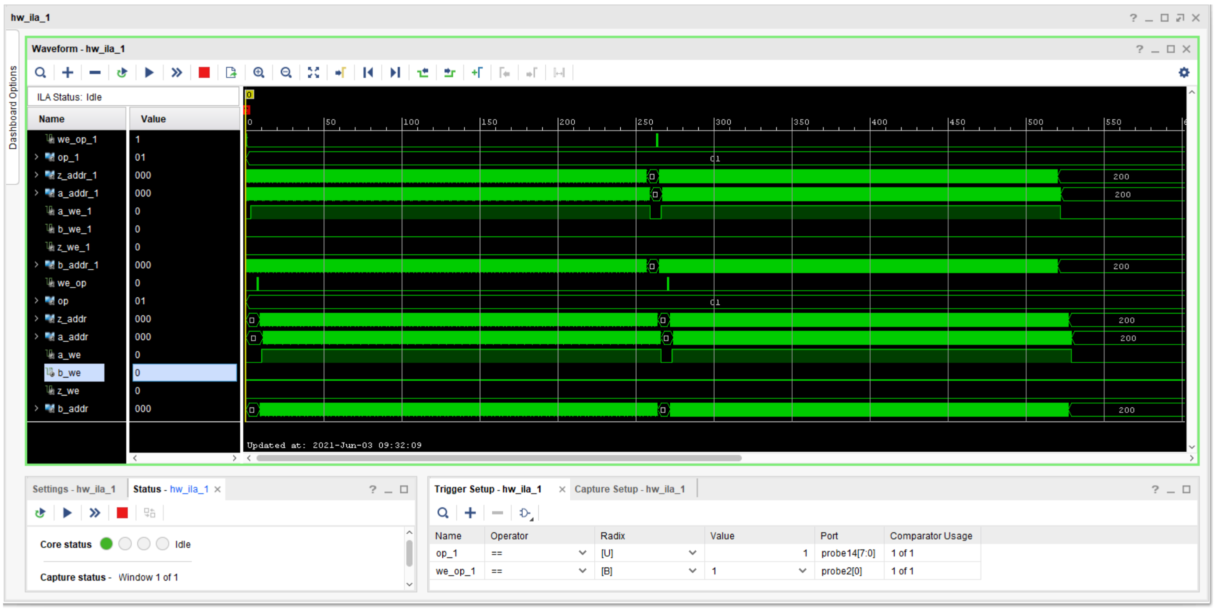


Figure 12: Instruction vz2a.

Figure 12 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 12.

In ILA, we can zoom to see the details. See Figure 13. 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.

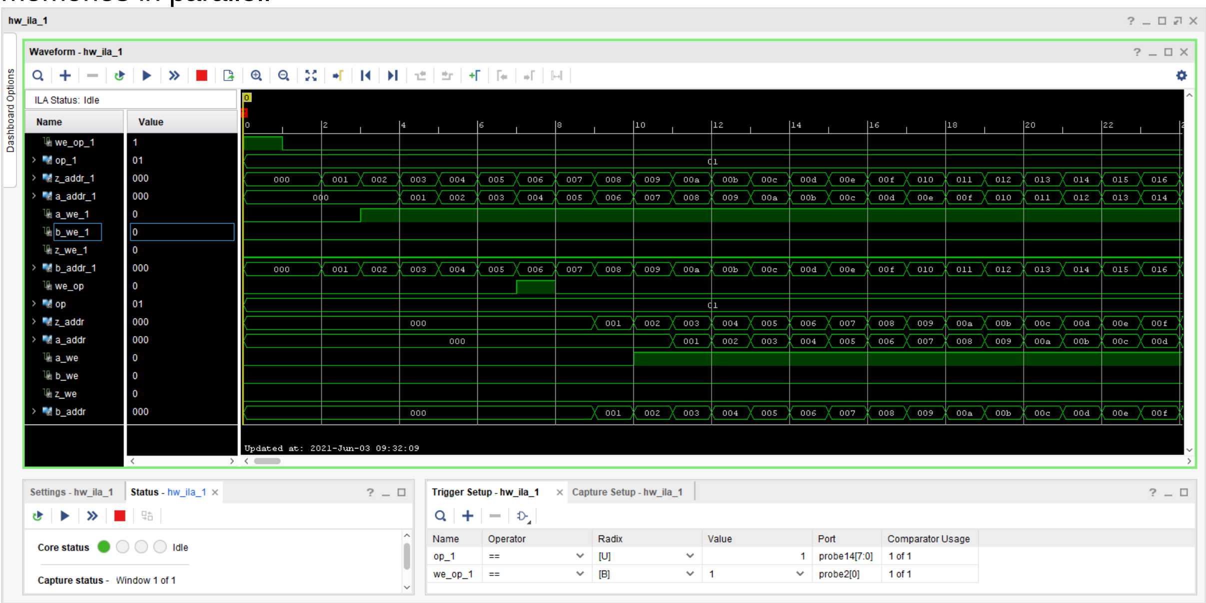


Figure 13: Instruction vz2a detail.

Figure 13 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 6 and Figure 7**.**

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 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 ZU15-EG-1EE 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 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 configurations are 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 might 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 Figure 7 for available HW accelerator versions.

9 References

[1] UltraSOM+ MPSoC Module with Zynq UltraScale+ ZU15EG-1FFVC900E, 4 GB DDR4
https://shop.trenz-electronic.de/en/TE0808-04-BBE21-A-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-ZU15EG-1FFVC900E-4-GB-DDR4

[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\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\`

Precompiled HW and SW projects are located in directory:
`c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\`

Sd_card image is located in directory:
`c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_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 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.
- Connect Full HD HDMI 60 FPS video source to Imageon board.
- Connect Full HD HDMI 60 FPS display to Imageon board.

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:
  `./sobel_all.elf`

RESULT

- The application will synchronize with the video input and start to perform HW accelerated edge detection (sobel algorithm) with 60 FPS.
- In parallel to the video processing, 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 of 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 2018.2 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 2018.2, select working directory.

te02_4x2_async_mulf64_sgdma_sw (Projects in this directory link to the same libte02_4x2_async_mulf64_sgdma_hw.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 2018.2 tool on your PC.

DEBUG of SW application from Xilinx SDK 2018.2

The application can be executed or debugged from the SDK 2018.2 For example:

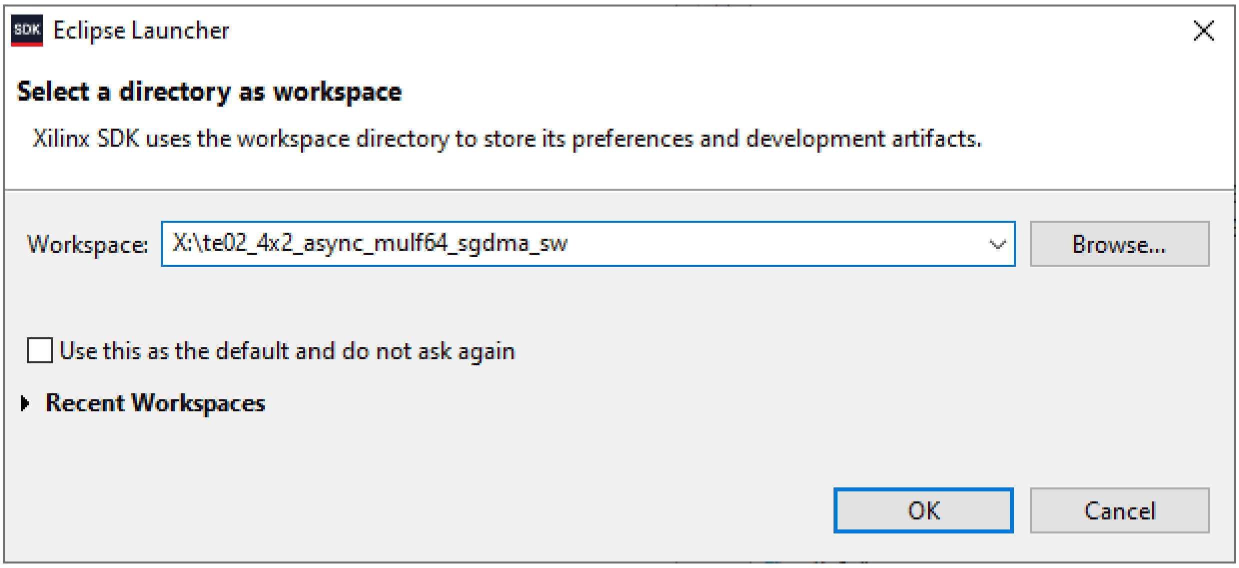


Figure 14: Select debug workspace.

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

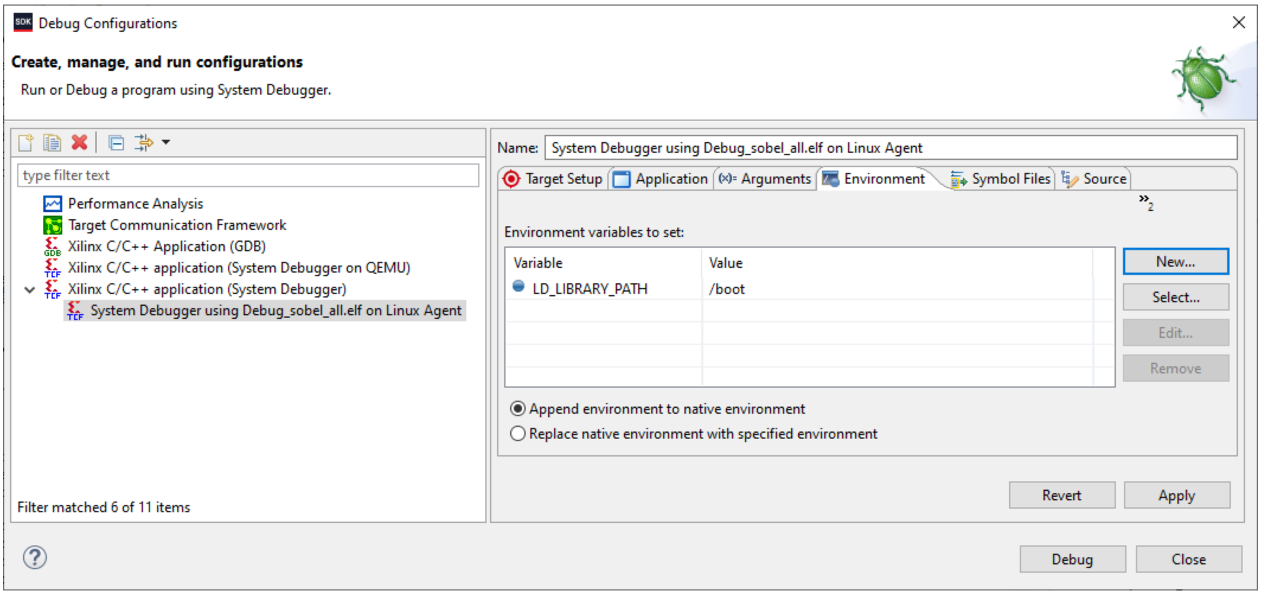


Figure 15: Define the environement variable.

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

`.\te02_4x2_async_mulf64_sgdma_sw\ sobel_all\Debug\sd_card\*.*`

as SD card files

`/boot/*.*`

Alternatively, you can also copy the content to the SD card directly 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.

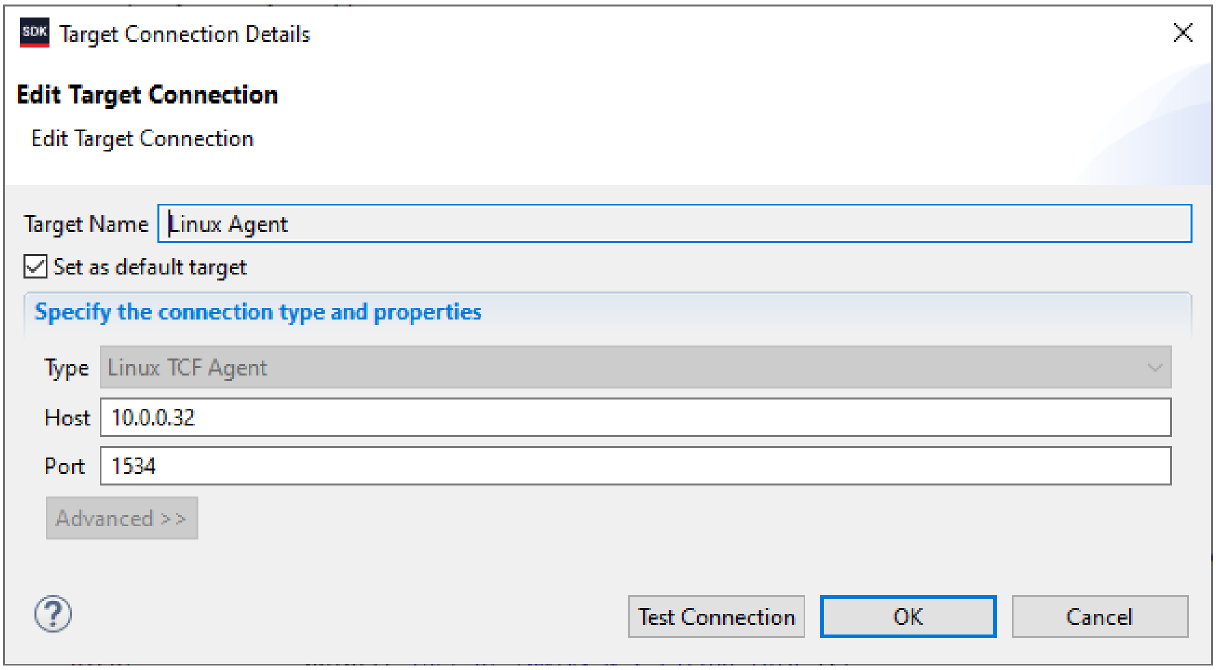


Figure 16: Test connection to Linux TCF Agent.

Compile SW application directly on the Zynq Ultrascale+ 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 or C++ directory with SDK projects. Example of C++:

`c:\home\work\TS82fp03x8_TEBF0808\zu15eg_deb_eval_ila_release\te02_4x2_async_mulf64_sgdma_sw\`

to the Debian file system into directory:

`/home/te02_4x2_async_mulf64_sgdma_sw`
To compile in Debian the **`all`** project (C++), change the directory to:

```
cd /home/te02_4x2_async_mulf64_sgdma_sw/sobel_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:

./sobel_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 on the carrier board will stop.

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

You can modify the SD card in PC and continue with other tests.

To close all work, you can close the PC terminal and then completely power down the TEBF0808 carrier board.

Guide for compilation and use of C MEX functions in Scilab

In Debian terminal, change directory to

/home/zu15eg_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/zu15eg_deb_eval_ila_release_scilab/cc/mmultf/libmex_mmultf.so

to

```
/home/zu15eg_deb_eval_ila_release_scilab/test/test_mmultf_4xB/
libmex_mmultf.so
```

In Debian terminal, change directory to

```
/home/zu15eg_deb_eval_ila_release_scilab/test/test_mmultf_4xB
```

Start Scilab by typing

```
scilab
```

In Scilab execute script

```
mmultf_4xB_test.sce
```

This script will execute mmultf C MEX function present in shared library **`libmex_mmultf.so`** and generate in the current directory 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.

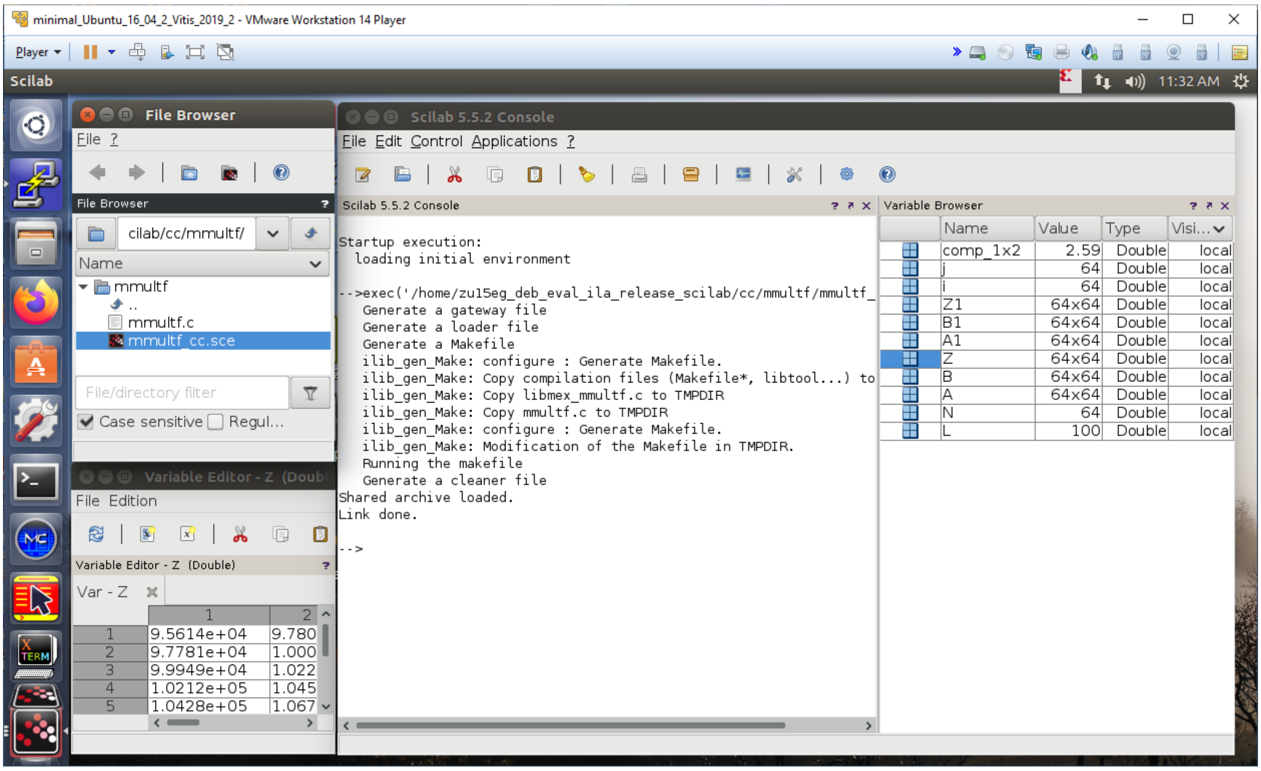


Figure 17: C MEX compilation in Arm Scilab. Remote X11 Desktop.

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:	24
Device:	xczu15eg-ffvc900-1-e
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 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\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila\`
   Change directory to:
   ```
   c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila\zusys\
   ```
2. Add the UTIA 8xSIMD HW IP to the package as the directory \ip
3. c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila\zusys\ip_lib\ip
4. On Win10, open dos terminal window, change directory to the folder
   ```
   c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila\zusys\
   ```
5. 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
   ```
6. 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=24
   ```
   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
   ```

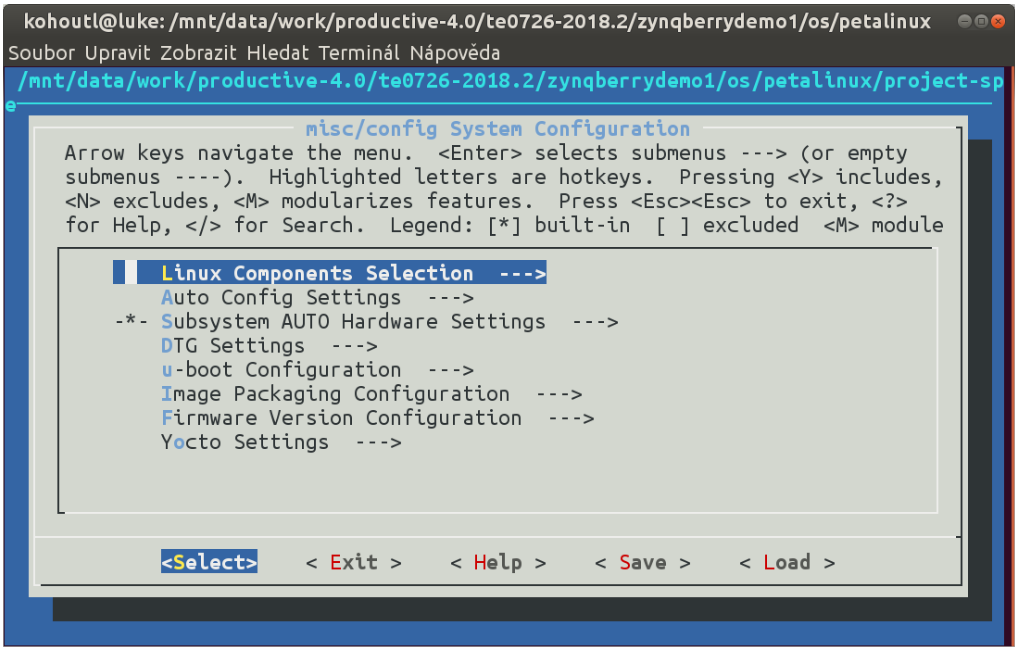
7. Start the Xilinx Vivado 2018.2 and create the design by executing of script:
```
X:\zusys\vivado_create_project_guimode.cmd
```
8. Optional:
You can use Vivado automation to add ILA monitor to enable ILA capturing of selected accelerator outputs of your choice.
9. In Vivado console, execute command:
```
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_1e_4gb\zusys.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 Ultrascale+ module TE0808-03-15EG-1EE 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-v2018.2-final
```
The standard PetaLinux 2018.2 distribution requires few modifications.

1. Copy content of these Win 10 directories:
```
X:\zusys\prebuilt
X:\zusys\os
```
to Ubuntu directories:
```
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys/os
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys/prebuilt
```
2. In Ubuntu, open terminal window and set path to the PetaLinux 2018.2:
```
source /opt/petalinux/petalinux-v2018.2-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/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_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:
```
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys/prebuilt/hardware/15eg_1e_4gb/zusys.hdf
```
5. Use the zusys.hdf file as input fo the PetaLinux configuration by (on single line)
```
petalinux-config --get-hw-
description=/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys/prebuilt/hardware/15eg_1e_4gb/
```



6. 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)  --->
```

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/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys/os
/petalinux/images/linux/image.ub
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys/os
/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_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.

10. Go to the folder with PetaLinux:

```
cd
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zu
sys/os/petalinux/
```

11. 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.

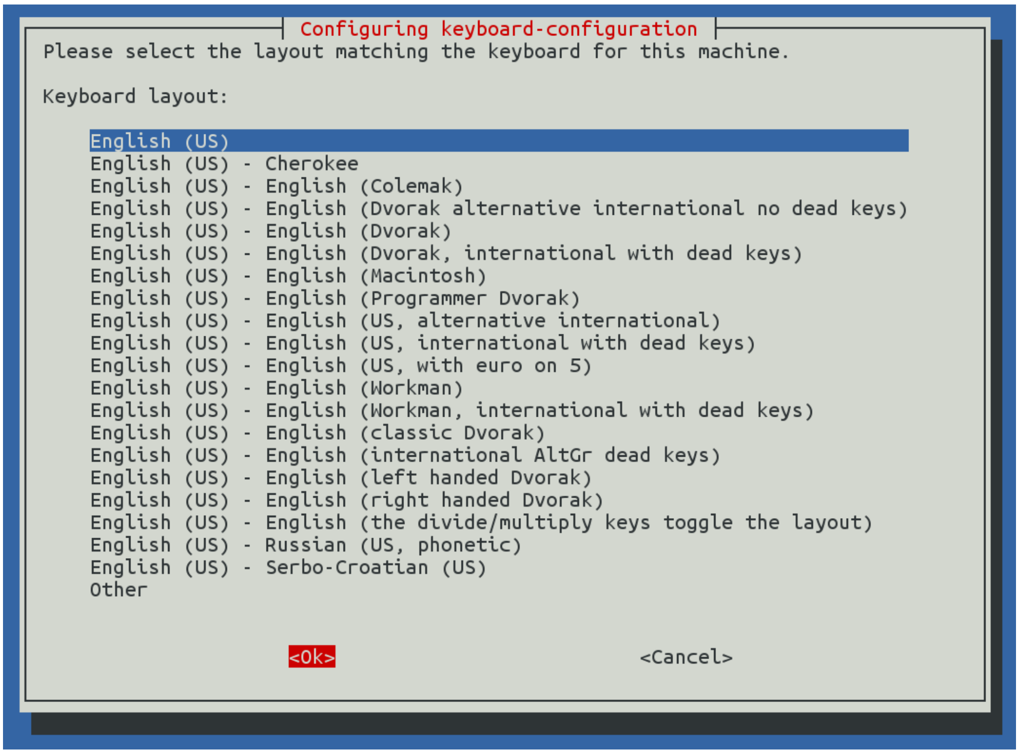
Tool	*Package*
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-arm-static	qemu-user-static

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 TE0808-debian TE0808-debian.img
```

14. Copy compressed image file from Ubuntu

```
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys
/os/petalinux/TE0808-debian.zip
```

to Win 10 file:

```
X:\zusys\prebuilt\os\petalinux\default\TE0808-debian.zip
```

15. Copy these files from Ubuntu

```
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys
/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_deb_eval_ila/zusys
/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp03x8_TEBF0808_DTRiMC_zu15eg/zu15eg_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
```

16. In Ubuntu, clean Petalinux project files

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

17. In Ubuntu, delete files

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/TE0808-debian.zip
/home/devel/work/TS82/TE0808/zusys/os/petalinux/TE0808-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 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\15eg_1e_4gb\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\TE0808-04\15EG-1EE`
   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\TE0808-04\15EG-1EE\
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:*\15EG-1EE
   ```
   _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 15EG-1EE
   ```
4. Open SDSoC project in directory
   ```
   X:\15EG-1EE
   ```
5. In SDSoC import two HW kernel design projects

   te02_4x2_async_mulf64_sgdma_hw

   and
   te03_4x2_async_mulf64_sgdma_hw

   from the directory

   c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila\SDSoC_PFM_src\TE0808-04\15EG-1EE\
6. Define the custom SDSoC platform

X:\15EG-1EE\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:\15EG-1EE\te02_4x2_async_mulf64_sgdma_hw\Release\sd_card\

   X:\15EG-1EE\te03_4x2_async_mulf64_sgdma_hw\Release\sd_card\
10. Copy content of the directory

    X:\15EG-1EE\te02_4x2_async_mulf64_sgdma_hw\Release\sd_card\

    to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te02_4x2_async_mulf64_sgdma_sw\Release\sd_card\
    and also to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te02_4x2_async_mulf64_sgdma_sw\Debug\sd_card\
11. Copy content of the directory

    X:\15EG-1EE\te03_4x2_async_mulf64_sgdma_hw\Release\sd_card\

    to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te03_4x2_async_mulf64_sgdma_sw\Release\sd_card\
    and also to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te03_4x2_async_mulf64_sgdma_sw\Debug\sd_card\
12. Optional:

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

    x:\ 15EG-1EE\ te02_4x2_async_mulf64_sgdma_hw\Release\

    _sds\p0\vivado\prj\prj.runs\impl_1\

    to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te02_4x2_async_mulf64_sgdma_sw\Release\sd_card\
    and also to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te02_4x2_async_mulf64_sgdma_sw\Debug\sd_card\

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

    x:\ 15EG-1EE\ te03_4x2_async_mulf64_sgdma_hw\Release\

    _sds\p0\vivado\prj\prj.runs\impl_1\

    to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te03_4x2_async_mulf64_sgdma_sw\Release\sd_card\
    and also to

    c:\home\work\TS82fp03x8_TEBF0808_DTRiMC_zu15eg\zu15eg_deb_eval_ila_release\te03_4x2_async_mulf64_sgdma_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=24, TE0808-04-15EG-1EE, with device xczu15eg-ffvc900-1-e, memory 4GB and short module name 15eg_1e_4gb.

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=24

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

ID	Module	Partname	Memory	ShortName
4	TE0808-ES2	xczu9eg-ffvc900-1-i-es2	2GB	es2_2gb
6	TE0808-2ES2	xczu9eg-ffvc900-2-i-es2	2GB	2es2_2gb
8	TE0808-04-09EG-1EA	xczu9eg-ffvc900-1-e	2GB	9eg_1e_2gb
10	TE0808-04-09EG-1EB	xczu9eg-ffvc900-1-e	4GB	9eg_1e_4gb
12	TE0808-04-09EG-1ED	xczu9eg-ffvc900-1-e	4GB	9eg_1e_4gb
14	TE0808-04-09EG-2IB	xczu9eg-ffvc900-2-i	4GB	9eg_2i_4gb
16	TE0808-04-15EG-1EE	xczu15eg-ffvc900-1-e	4GB	15eg_1e_4gb
18	TE0808-04-09EG-1EE	xczu9eg-ffvc900-1-e	4GB	9eg_1e_4gb
20	TE0808-04-09EG-1EL	xczu9eg-ffvc900-1-e	4GB	9eg_1e_4gb
22	TE0808-04-09EG-2IE	xczu9eg-ffvc900-2-i	4GB	9eg_2ib_sk
24	**TE0808-04-15EG-1EE**	**xczu15eg-ffvc900-1-e**	**4GB**	**15eg_1e_4gb**
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
36	TE0808-04-9GI21-A	xczu9eg-ffvc900-2-i	4GB	9eg_2i_4gb
38	TE0808-04-9BE21-A	xczu9eg-ffvc900-1-e	4GB	9eg_1e_4gb
40	TE0808-04-6BE21-L	xczu6eg-ffvc900-1-e	4GB	6eg_1e_4gb
42	TE0808-04-6BE21-A	xczu6eg-ffvc900-1-e	4GB	6eg_1e_4gb
44	TE0808-04-9BE21-L	xczu9eg-ffvc900-1-e	4GB	9eg_1e_4gb
46	TE0808-04-BBE21-A	xczu15eg-ffvc900-1-e	4GB	15eg_1e_4gb

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 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

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

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.