Application Note

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

STM32H753 Terminal with Zynq Ultrascale+ Accelerator

Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout, Raissa Likhonina
kadlec@utia.cas.cz xpohl@utia.cas.cz kohoutl@utia.cas.cz likhonina@utia.cas.cz

Revision history

Rev.	Date	Author	Description
0	27.07.2021	J. Kadlec	Initial draft
1			
2			

WAKeMeUP has been accepted for funding within the Electronic Components and Systems For European Leadership Joint Undertaking in collaboration with the European Union's H2020 Framework Programme (H2020/2014-2020) and National/Local Authorities, under grant agreement n° 783176

Table of Contents

Table of Figures

Acknowledgement

This work has been partially supported by ECSEL WAeMeUP project, No. 783176 [1] and by the corresponding Czech NFA (MSMT) institutional support project MSMT 8A18001.

1 STM32H753 Terminal with Zynq Ultrascale+ Accelerator

This application note was developed in the frame of ECSEL JU project WAKeMeUP [1].

The application note and the corresponding evaluation package describes STM32H753 Nucleo 144 board based terminal with, an Adafruit TFT display and Zynq Ultrascale+ accelerator on TE0820 module and TE0706 carrier board. See Figure 1.

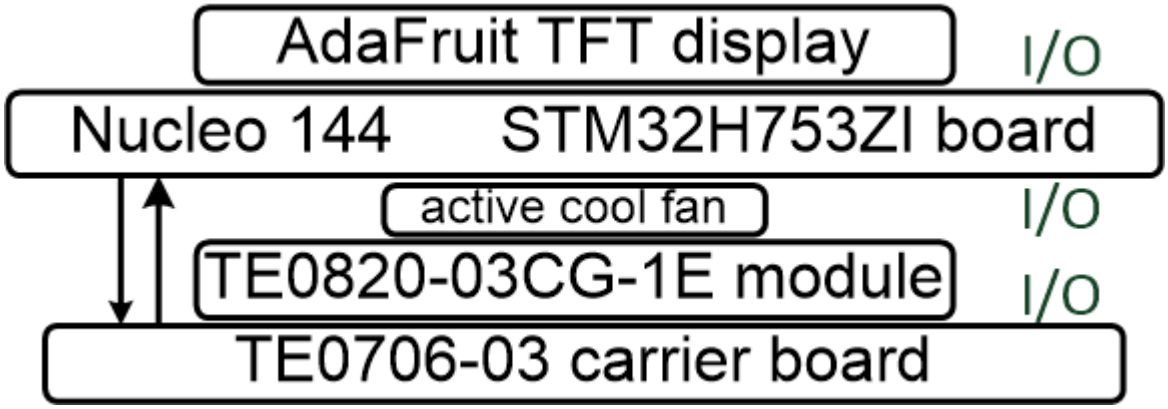


Figure 1: Stacked boards of the STM32H753 terminal with Zynq Ultrascale+ accelerator

Complete terminal is presented in Figure 2.

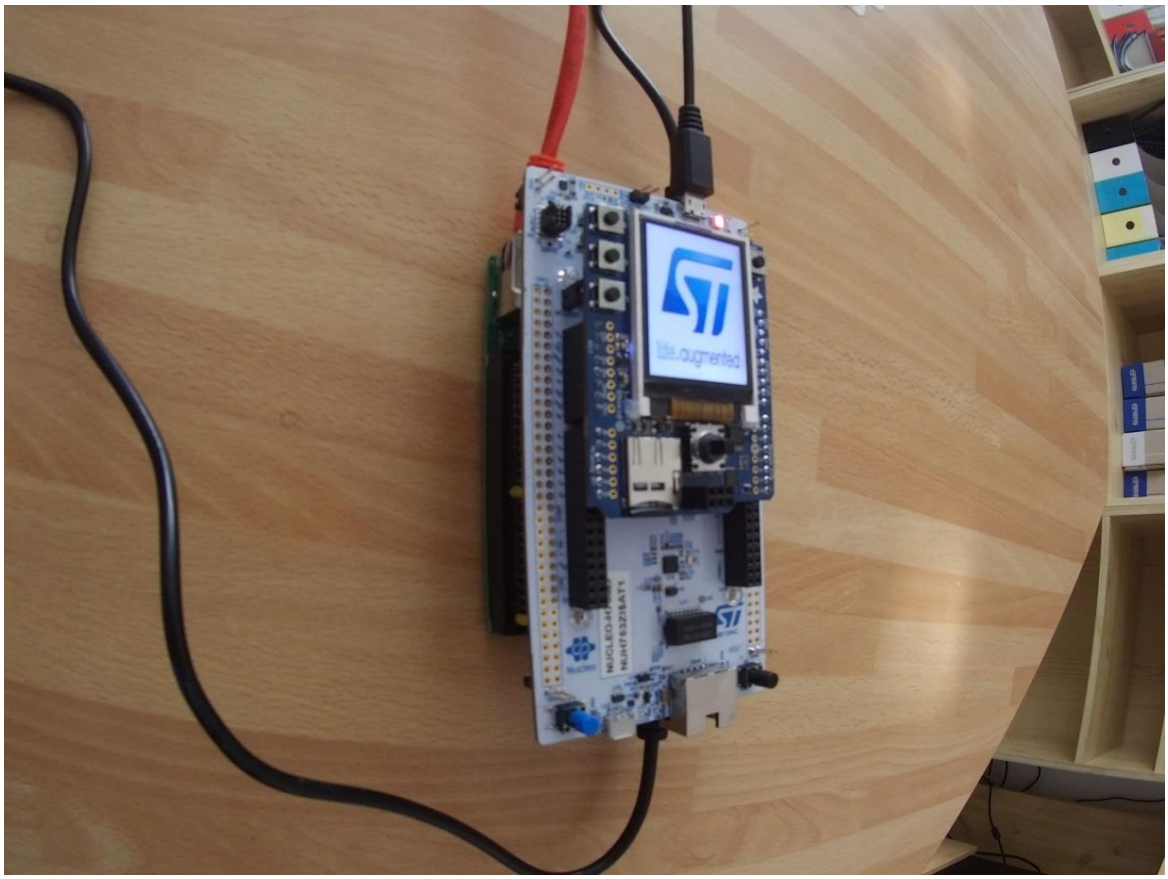


Figure 2: Nucleo 144 STM32H753 terminal with Zynq Ultrascale+ accelerator

- The Nucleo STM32H753ZI board [2] is product of ST Microelectronics.
- The Adafruit TFT display [3] is product of Adafruit.
- The TE0820 modules [4], [5], [6] and the TE0706 carrier board [7] are products of Trenz Electronic.

Source code of SW project for the “System Workbench for STM32” (AC6) tool [11] for the Nucleo-144 STM32H753ZI board is included in the released evaluation package accompanying this application note.

SW project is using the STM32CubeH7 Firmware Package V1.5.0 available for download with the STM32CubeMX – The STM32Cube initialization code generator [12].

These SW development packages are provided free of charge by STMicroelectronics.

The Zynq Ultrascale+ Debian OS supports Scilab interpret [8]. Scilab scripts prepare and visualize double precision matrix data and also call compiled C/C++ functions with MEX format identical to Matlab. In the terminal application, Scilab executes user defined script and acts as server for the STM32H7 MCU terminal.

Scilab scripts define content of user menus and execute calls to precompiled MEX functions executed on Arm A53 1.2 GHz microprocessor of the Zynq Ultrascale+ device.

Scilab can also use the ssh PuTTY terminal for 1Gbit fixed line Ethernet connection to a remote X11 Desktop for visualization of simulation results.

See Figure 3 and also
Figure 11 and
Figure 12.

Scilab script also save computed data in ascii format as .h C/C++ header files on the SD_card. These header files with “golden” data are used for port of data to benchmark STM32H7 SW projects.

Scilab measures also the Arm A53 execution time and converts it to single and double floating point performance data. These data are used for comparison with the single and double floating point performance of these STM32H7 (MCUs):

- NUCLEO-H743ZI2 Mounted Device : STM32H743ZITx (M7 MCU) [2]
- NUCLEO-H753ZI Mounted Device : STM32H753ZITx (M7 MCU) [2]
- NUCLEO-H755ZI-Q Mounted Device : STM32H755ZITx (M7+M4 MCUs) [10]
- NUCLEO-H7A3ZI-Q Mounted Device : STM32H7A3ZITxQ (M7 MCU) [10]
- NUCLEO-H723ZG Mounted Device : STM32H723ZGTx (M7 MCU) [2]

Results of this comparison and the complete list of created and released SW projects for the STM32Cube_FW_H7_1.5.0 framework with AC6 tool and STM32Cube_FW_H7_1.9.0 with STM32CubeIDE 1.7.0 tool are described in the application note [9]. Projects can be downloaded in source code in the evaluation package accompanying the application note [9].

The application note [9] describes STM32H753 terminal with Zynq accelerator on the ArduZynq Arduino compatible shield. It supports Debian OS and the command line scilab-cli interpret client. Both terminals generate identical test data for the SW benchmark projects for

evaluation of performance of several STM32H7 devices. The terminal with Zynq Ultrascale+ scilab supports the graphic output to the X11 remote desktop. See Figure 3.

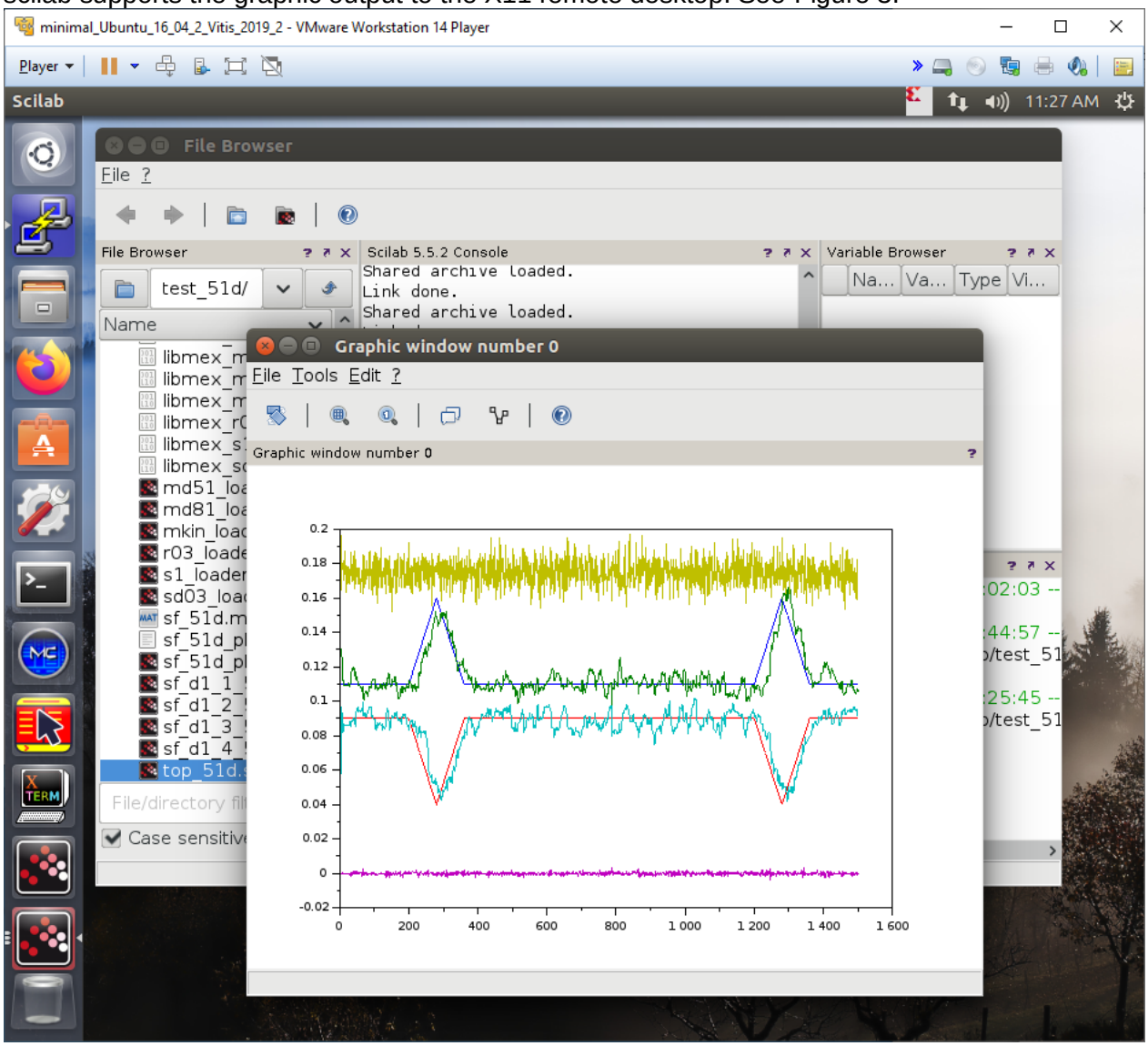


Figure 3: Terminal graphical output in scilab on remote Debian X11 desktop

The ARM A53 processor on TE0706 carrier board is connected to the STM753ZI CM7 MCU device on NUCLEO-H753ZI board by two wire USART serial line with baud rate 460800 bits/s. This USART connection is using the LVCMOS33 logic (0-3.3V).

Enabled in NUCLEO-H753ZI CM7 MCU		Enabled in Zynq Ultrascale+ MPU	
Tx PC6 D16	pin 1 in CN7 connector	Rx	UART1 rxd MIO29 bank LVCMOS33
Rx PC7 D21	pin 11 in CN7 connector	Tx	UART1 txd MIO28 bank LVCMOS33

The serial asynchronous communication is configured in the main application program of the STM753ZI CM7 MCU for the USART6 device. See source code in the evaluation package. In case of ARM A53 Debian OS, the corresponding UART1 device is accessible as /dev/ttyPS1. The baud rate 460800 bits/s is set in scilab by executing shell command:

stat=host("stty 460800 < /dev/ttyPS1");
See source code of scilab scripts in the evaluation package accompanying this app. note.

2 Zynq Ultrascale+ 8xSIMD FP03x8 HW accelerators

Programmable logic part of the ZynqUltrascale+ device contains evaluation versions of two 8xSIMD FP03x8 single precision FP32 floating-point, run-time-reconfigurable accelerators. See Figure 4. These accelerators serve as reprogrammable, general purpose HW accelerators of sequences of floating point vector operations involving vector ADD, SUB MUL, DIV and vector Dot-Product operations. See Figure 5.

The programmable logic part of the Zynq Ultrascale+ device also contains fixed hardware for single precision FP32 floating point matrix multiplication and addition. Matrices must have fixed size [16x16]. It is example of the HW accelerator designed with the Xilinx SDSoC 2018.2 system level compiler. This fixed HW accelerator serves for performance comparison with the two 8xSIMD FP03x8 run-time reprogrammable HW accelerators. See Figure 4.

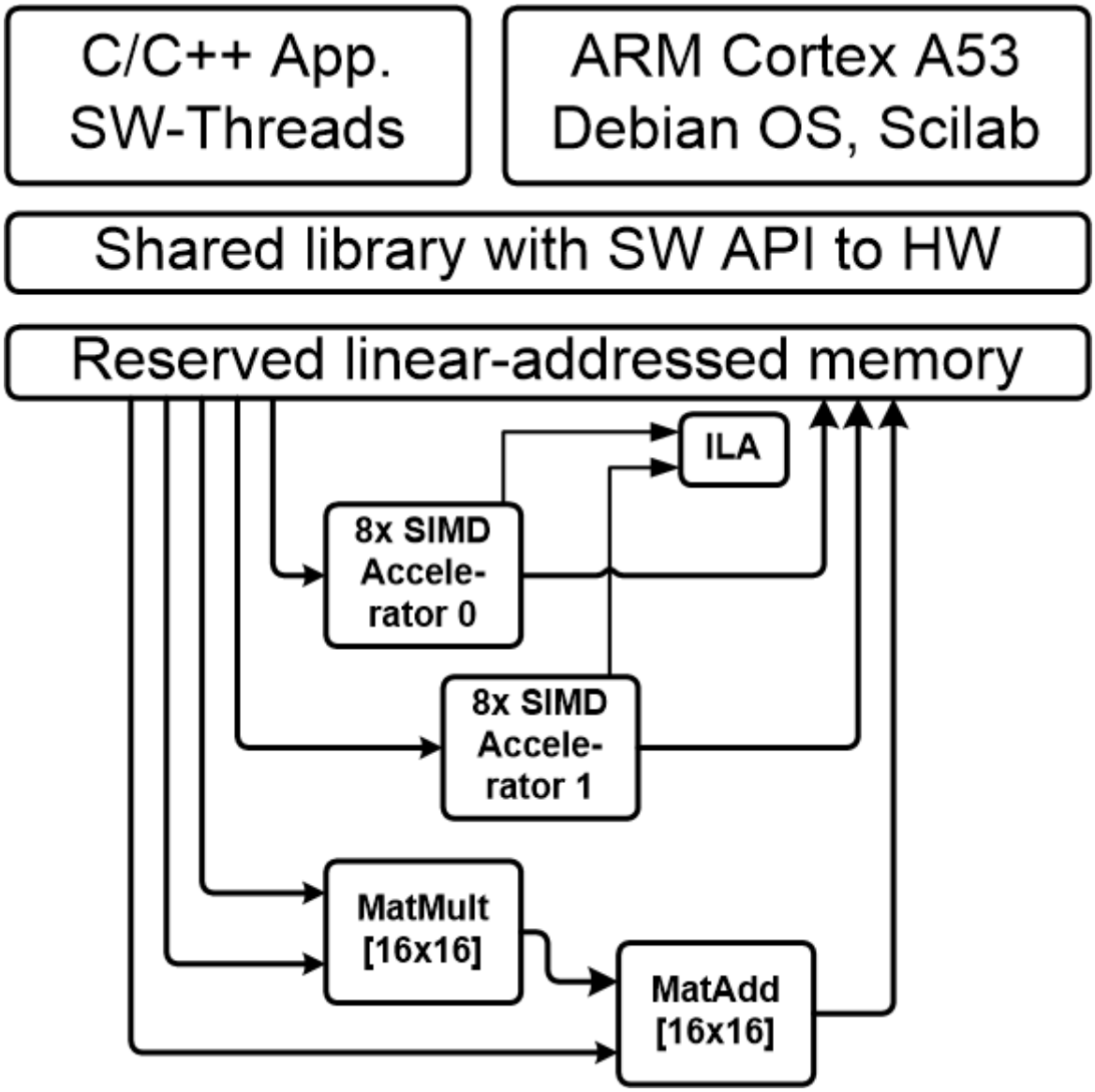


Figure 4: Zynq Ultrascale+ SoC device with HW accelerators

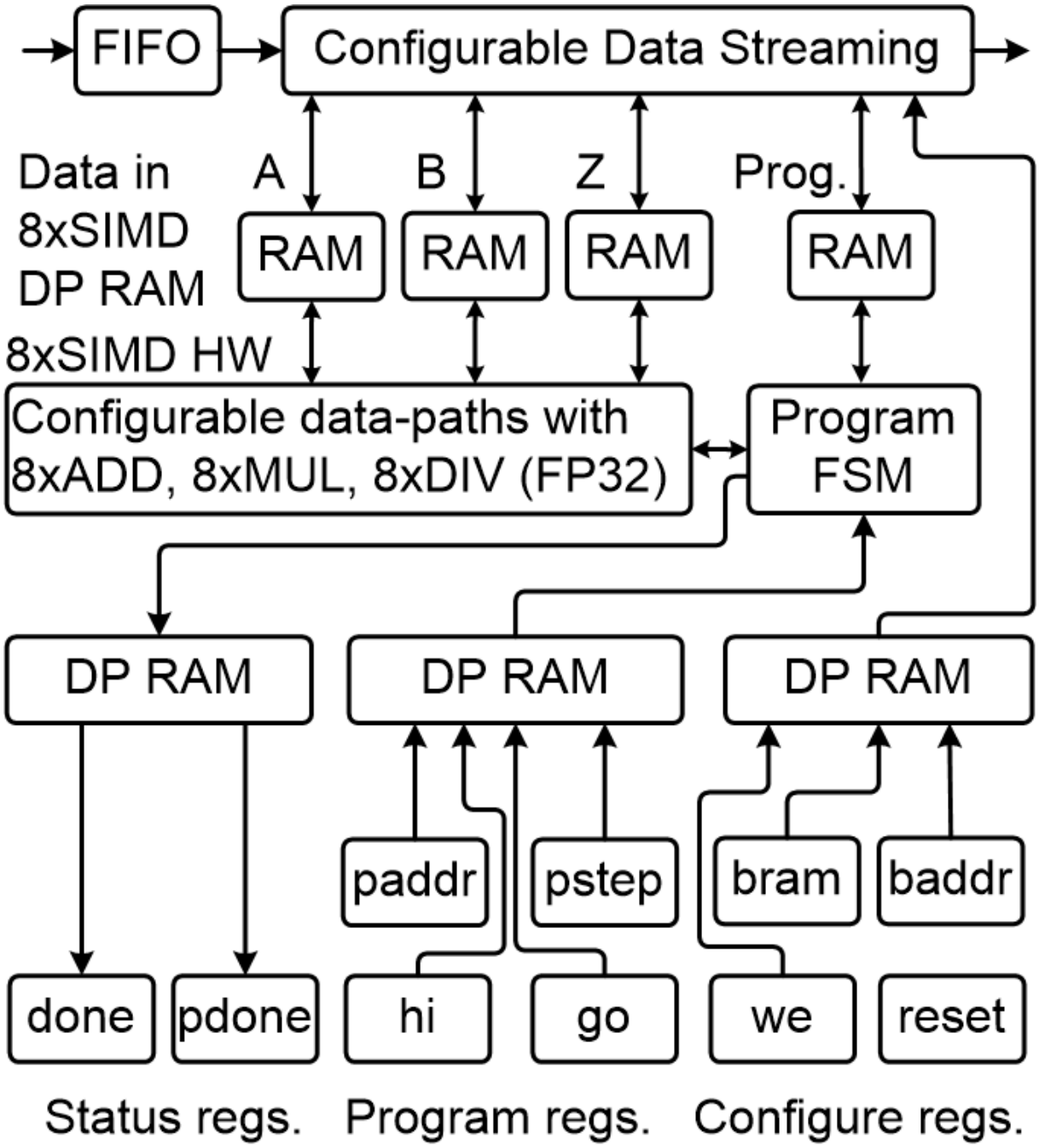


Figure 5: Internal structure of the 8xSIMD FP03x8 HW accelerator

Integration of 8xSIMD FP03x8 accelerators

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.

Supported Zynq Ultrascale+ devices and modules

This application note and evaluation package supports these Trenz Electronic modules:

ID	Module	Partname	Memory	ShortName	Ref.
2	TE0820-02-02EG-1E	xczu2eg-sfvc784-1-e	1GB	2eg_1e_1gb	
3	TE0820-02-02EG-1E3	xczu2eg-sfvc784-1-e	1GB	2eg_1e_1gb	
4	TE0820-02-02CG-1E	xczu2cg-sfvc784-1-e	1GB	2cg_1e_1gb	
5	**TE0820-02-03EG-1E**	**xczu3eg-sfvc784-1-e**	**1GB**	**3eg_1e_1gb**	
6	TE0820-02-03EG-1E3	xczu3eg-sfvc784-1-e	1GB	3eg_1e_1gb	
7	**TE0820-02-03CG-1E**	**xczu3cg-sfvc784-1-e**	**1GB**	**3cg_1e_1gb**	
8	TE0820-02-02EG-1EA	xczu2eg-sfvc784-1-e	1GB	2eg_1e_1gb	
9	TE0820-02-02EG-1EL	xczu2eg-sfvc784-1-e	1GB	2eg_1e_1gb	
10	TE0820-02-02CG-1EA	xczu2cg-sfvc784-1-e	1GB	2cg_1e_1gb	
11	TE0820-02-03EG-1EA	xczu3eg-sfvc784-1-e	1GB	3eg_1e_1gb	
12	TE0820-02-03EG-1EL	xczu3eg-sfvc784-1-e	1GB	3eg_1e_1gb	
13	TE0820-02-03CG-1EA	xczu3cg-sfvc784-1-e	1GB	3cg_1e_1gb	
14	TE0820-02-04CG-1EA	xczu4cg-sfvc784-1-e	1GB	4cg_1e_1gb	
15	**TE0820-03-04EV-1EA**	**xczu4ev-sfvc784-1-e**	**2GB**	**4ev_1e_2gb**	
16	TE0820-03-02CG-1EA	xczu2cg-sfvc784-1-e	2GB	2cg_1e_2gb	
17	TE0820-03-02EG-1EA	xczu2eg-sfvc784-1-e	2GB	2eg_1e_2gb	
18	TE0820-03-02EG-1EL	xczu2eg-sfvc784-1-e	2GB	2eg_1e_2gb	
19	TE0820-03-03CG-1EA	xczu3cg-sfvc784-1-e	2GB	3cg_1e_2gb	
20	TE0820-03-04CG-1EA	xczu4cg-sfvc784-1-e	2GB	4cg_1e_2gb	
21	TE0820-03-03EG-1EA	xczu3eg-sfvc784-1-e	2GB	3eg_1e_2gb	
22	TE0820-03-03EG-1EL	xczu3eg-sfvc784-1-e	2GB	3eg_1e_2gb	
23	TE0820-03-2AI21FA	xczu2cg-sfvc784-1-i	2GB	2cg_1i_2gb	
24	TE0820-03-2BE21FL	xczu2eg-sfvc784-1-e	2GB	2eg_1e_2gb	
25	TE0820-03-3AI210A	xczu3cg-sfvc784-1-i	2GB	3cg_1i_2gb	
26	TE0820-03-3BE21FA	xczu3eg-sfvc784-1-e	2GB	3eg_1e_2gb	[5]
27	TE0820-03-3BE21FL	xczu3eg-sfvc784-1-e	2GB	3eg_1e_2gb	
28	TE0820-03-02CG-1ED	xczu2cg-sfvc784-1-e	2GB	2cg_1e_2gb	
29	TE0820-03-2AE21FA	xczu2cg-sfvc784-1-e	2GB	2cg_1e_2gb	
30	TE0820-03-2BE21FA	xczu2eg-sfvc784-1-e	2GB	2eg_1e_2gb	
31	TE0820-03-3AE21FA	xczu3cg-sfvc784-1-e	2GB	3cg_1e_2gb	[4]
32	TE0820-03-3AI21FA	xczu3cg-sfvc784-1-i	2GB	3cg_1e_2gb	
33	TE0820-03-4AE21FA	xczu4cg-sfvc784-1-e	2GB	4cg_1e_2gb	
34	TE0820-03-4DE21FA	xczu3cg-sfvc784-1-e	2GB	4ev_1e_2gb	[6]
35	TE0820-03-4DI21FA	xczu3cg-sfvc784-1-i	2GB	4ev_1i_2gb	

UTIA has used modules highlighted in bold in the list above and assembled three terminal instances of the STM32H753 terminal withZynq Ultrascale+ accelerator. The evaluation package accompanying this application note has been used for integraton of two 8xSIMD HW accelerators and for configuration and compilation of Linux Debian OS.

- **STM32H753** with **TE0820-02-03CG-1E** module is using xczu3cg-sfvc784-1-e Zynq UltraScale+ device and 1 GB DDR4 memory. It has dual core Arm A53 processor, and no GPU graphic accelerator.
- **STM32H753** with **TE0820-02-03EG-1E** module is using xczu3eg-sfvc784-1-e Zynq Ultrascale+ device and 1 GB DDR4 memory. It has four core Arm A53 processor, and GPU graphic accelerator.
- **STM32H753** with **TE0820-03-04EV-1EA** module is using xczu4ev-sfvc784-1-e Zynq UltraScale+ device and 2 GB DDR4 memory. It has four core Arm A53 processor, GPU graphic accelerator, integrated video codec HW IP core and ultra RAM memory blocks in the programmable logic area.

General use of 8xSIMD FP03x8 accelerators

SW developer can program and use the two 8xSIMD FP03x8 HW accelerators without SDSoC 2018.2 compiler license. The standard gcc C compiler or the g++ C++ compiler and „make“ commands can be also used for cross-compilation on Win 10 PC or in the Debian OS on the dual core Arm A53 1.2 GHz microprocessor of the Zynq Ultrascale+ device .

The two 8xSIMD FP03x8 accelerators and the HW data movers supporting data communication are represented for the SW developer as shared C or C++ library with simple SW API. The API is identical for several alternatives of HW data movers.

The evaluation package provides several pre-compiled HW designs represented in form of SD-cards containing these designs and API interface for SW developer in form of shared Debian libraries for Arm host processor.

The FP03x8 HW accelerators serve for run-time reprogrammable 8xSIMD single precision floating point computations. The internal structure of FP03x8 accelerators is described in Figure 5.

All designs present in this evaluation package contain two independent 8xSIMD FP03x8 accelerators in the programmable logic part of the device. See Figure 4.

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

The evaluation package includes 8xSIMD FP03x8 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 licensing of 8xSIMD accelerators HW IPs without this restriction.

P03x8 HW accelerator

Parameters of 8xSIMD FP03x8 accelerators (same for 03CG,03EG,04EV)

Interfaces

Type of interface:	Clock:
• Data streaming I/O: AXI-S 32 bit	240 MHz
• Firmware program VLIW 128 bit	240 MHz
• Configuration I/O: AXI-lite 32 bit	100 MHz
• ARM A53 system clock	1200 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 6: 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 7: 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 8: Specific functions present only in some versions accelerators.

3 Programming of 8xSIMD FP03x8 floating point accelerators

Host Arm A53 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 9.

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 9: 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 5).

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.

The HW Sobel filter for edge detection is precompiled into the programmable logic and present in the shared libraries together with the two serial connected accelerators.

The internal structure of the Zynq Ultrascale+ SoC with two serial connected accelerators and HW accelerator for Sobel filter FP03x8 accelerator can be seen in Figure 5.

4 Evaluation of performance of Zynq Ultrascale+ accelerator

Released evaluation package includes SW projects for C (gcc) compiler and C++ (g++) compiler.

C projects:

Directory: fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_c_sw\

	Comparison of A53 SW, 8xSIMD and SDSoC
SW project: ./matmultf0_c	Matrix multiplications [64,64] (without copy of matrices to DDR4)
SW project: ./matmultf0_c_thread	Matrix multiplication [64,64] use 2 SW threads (without copy of matrices to DDR4)
SW project: ./matmultf1_c	Matrix multiplication [64,64] (with copy of matrices to DDR4)
SW project: ./matmultf1_c_thread	Matrix multiplication [64,64] use 2 threads (with copy of matrices to DDR4)
	Test of vector 8xSIMD operations
SW project: ./va2bf0_c/	Vector copy from A[4096] to B[4096]
SW project: ./vadd0_c/	Vector ADD: Z[4096] = A[4096] .+ B[4096]
SW project: ./vadd0_c_az2b/	Vector ADD: B[4096] = A[4096] .+ Z[4096]
SW project: ./vadd0_c_bz2a/	Vector ADD: A[4096] = B[2096] .+ Z[4096]
SW project: ./vb2af0_c/	Vector copy from B[4096] to A[4096]
SW project: ./vdivf0_c/	Vector DIV: Z[4096] = A[4096]./B[4096]
SW project: ./vmacf0_c/	Vector MAC
SW project: ./vmsubacf0_c/	Vector MSUBAC
SW project: ./vmul0_c/	Vector MUL: Z[4096] = A[4096] .* B[4096]
SW project: ./vmul0_c_az2b/	Vector MUL: B[4096] = A[4096] .* Z[4096]
SW project: ./vmul0_c_bz2a/	Vector MUL: A[4096] = B[2096] .* Z[4096]
SW project: ./vprodf0_c/	Vector VPROD
SW project: ./vprods8f0_c/	Vector VPRODS8
SW project: ./vsub0_c/	Vector SUB: Z[4096] = A[4096] .- B[4096]

SW project: ./vsub0_c_az2b/	Vector SUB: B[4096] = A[4096] .- Z[4096]
SW project: ./vsub0_c_bz2a/	Vector SUB: A[4096] = B[2096] .- Z[4096]
SW project: ./vz2af0_c/	Vector copy from Z[4096] to A[4096]
SW project: ./vz2bf0_c/	Vector copy from Z[4096] to B[4096]

Shared library

Shared library: ./Debug/sd_card/ libfp03x8_v26x_2x1_zc_muladdf_c_hw.so
Shared library: ./Release/sd_card/ libfp03x8_v26x_2x1_zc_muladdf_c_hw.so

C++ projects:

Directory: fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\

	Comparison of A53 SW, 8xSIMD and SDSoC
SW project: ./matmultf0	Matrix multiplications [64,64] (without copy of matrices to DDR4)
SW project: ./matmultf0_thread	Matrix multiplication [64,64] use 2 SW threads (without copy of matrices to DDR4)
SW project: ./matmultf1	Matrix multiplication [64,64] (with copy of matrices to DDR4)
SW project: ./matmultf1_thread	Matrix multiplication [64,64] use 2 threads (with copy of matrices to DDR4)
	Test of vector 8xSIMD operations
SW project: ./va2bf0/	Vector copy from A[4096] to B[4096]
SW project: ./vadd0/	Vector ADD: Z[4096] = A[4096] .+ B[4096]
SW project: ./vadd0_az2b/	Vector ADD: B[4096] = A[4096] .+ Z[4096]
SW project: ./vadd0_bz2a/	Vector ADD: A[4096] = B[2096] .+ Z[4096]
SW project: ./vb2af0/	Vector copy from B[4096] to A[4096]
SW project: ./vdivf0/	Vector DIV: Z[4096] = A[4096]./B[4096]
SW project: ./vmacf0/	Vector MAC
SW project: ./vmsubacf0/	Vector MSUBAC
SW project: ./vmul0/	Vector MUL: Z[4096] = A[4096] .* B[4096]
SW project: ./vmul0_az2b/	Vector MUL: B[4096] = A[4096] .* Z[4096]
SW project: ./vmul0_bz2a/	Vector MUL: A[4096] = B[2096] .* Z[4096]
SW project: ./vprodf0/	Vector VPROD
SW project: ./vprods8f0/	Vector VPRODS8
SW project: ./vsub0/	Vector SUB: Z[4096] = A[4096] .- B[4096]
SW project: ./vsub0_az2b/	Vector SUB: B[4096] = A[4096] .- Z[4096]
SW project: ./vsub0_bz2a/	Vector SUB: A[4096] = B[2096] .- Z[4096]
SW project: ./vz2af0/	Vector copy from Z[4096] to A[4096]
SW project: ./vz2bf0/	Vector copy from Z[4096] to B[4096]

Shared library: ./Debug/sd_card/ libfp03x8_v26x_2x1_zc_muladdf_hw.so
Shared library: ./Release/sd_card/ libfp03x8_v26x_2x1_zc_muladdf_hw.so

Evaluation package contains SW projects and Debug and Release versions of shared libraries for Debian Stretch 9.8 OS for the Zynq Ultrascale+ device for the SDK 2018.2 C SW flow with gcc compiler and for C++ SW flow with g++ compiler. Libraries provide interfaces to the programmable logic part of the device with 2 evaluation versions of 8xSIMD FP03x8 HW accelerators with zero copy data movers and fixed SDSoC HV accelerator for floating point Multiply and Add [16x16].

10.0.0.189 - PuTTY

```
root@zynqmp:/boot# cd /boot
root@zynqmp:/boot# export LD_LIBRARY_PATH=/boot
root@zynqmp:/boot# ./matmultf0.elf
Testing mmult and madd with SDSoC HW ...
Testing mmult and madd in SW ...
Acceleration SDSoC: 0.790289
Test of SDSoC HW : PASSED
/dev/mem opened.
Memory mapped at address 0x7f9cca1000.
Memory mapped at address 0x7f9cca0000.
Initialisation of 2xl 8xSIMD HW ...
Operations:     13ffff     13ffff
License   :   c0000000   c0000000
Testing of 2xl 8xSIMD HW ...

Partial listing of results. 8xSIMD accelerator 0 (C indexing):
Z1_Z2[0..2].d[0]    : 90.115639 90.690399 91.265137
Z1_Z2[0..2].d[1]    : 90.128479 90.703308 91.278152
Z3_Z4[0..2].d[0]    : 90.141327 90.716240 91.291153
Z3_Z4[0..2].d[1]    : 90.154160 90.729156 91.304153
Z5_Z6[0..2].d[0]    : 90.167007 90.742081 91.317154
Z5_Z6[0..2].d[1]    : 90.179840 90.755005 91.330162
Z7_Z8[0..2].d[0]    : 90.192680 90.767929 91.343170
Z7_Z8[0..2].d[1]    : 90.205521 90.780853 91.356171
Z1_Z2[64..66].d[0]  : 90.218369 90.793777 91.369179
Z1_Z2[64..66].d[1]  : 90.231209 90.806686 91.382187
Z3_Z4[64..66].d[0]  : 90.244049 90.819611 91.395172
Z3_Z4[64..66].d[1]  : 90.256882 90.832535 91.408188
Z5_Z6[64..66].d[0]  : 90.269722 90.845459 91.421188
Z5_Z6[64..66].d[1]  : 90.282562 90.858383 91.434189
Z7_Z8[64..66].d[0]  : 90.295403 90.871300 91.447205
Z7_Z8[64..66].d[1]  : 90.308243 90.884224 91.460205
Z1_Z2[128..130].d[0]: 90.321091 90.897156 91.473213
Z1_Z2[128..130].d[1]: 90.333923 90.910080 91.486214
...
Partial listing of results. 2xl 8xSIMD accelerators 0 and 1 (C indexing):
Z1_Z2  [0..7].d[0]: 90.115639 90.690399 91.265137 91.839882 92.414642 92.989395 93.564148 94.13
8893
Z1_Z2_1[0..7].d[0]: 91.442970 92.084274 92.725586 93.366898 94.008209 94.649521 95.290833 95.93
2137
...

Partial listing of result matrix: C[1..64,1..64]    C1[1..64,1..64] (Scilab indexing):
C[ 1, 1..3] = [90.115639 90.690399 91.265137]   C1[ 1, 1..3] = [91.442970 92.084274 92.725586]
C[ 2, 1..3] = [90.128479 90.703308 91.278152]   C1[ 2, 1..3] = [91.456459 92.097862 92.739273]
C[ 3, 1..3] = [90.141327 90.716240 91.291153]   C1[ 3, 1..3] = [91.469948 92.111450 92.752945]
C[ 4, 1..3] = [90.154160 90.729156 91.304153]   C1[ 4, 1..3] = [91.483429 92.125015 92.766617]
C[ 5, 1..3] = [90.167007 90.742081 91.317154]   C1[ 5, 1..3] = [91.496918 92.138603 92.780289]
C[ 6, 1..3] = [90.179840 90.755005 91.330162]   C1[ 6, 1..3] = [91.510399 92.152191 92.793976]
C[ 7, 1..3] = [90.192680 90.767929 91.343170]   C1[ 7, 1..3] = [91.523895 92.165771 92.807648]
C[ 8, 1..3] = [90.205521 90.780853 91.356171]   C1[ 8, 1..3] = [91.537384 92.179352 92.821320]
C[ 9, 1..3] = [90.218369 90.793777 91.369179]   C1[ 9, 1..3] = [91.550873 92.192932 92.834999]
C[10, 1..3] = [90.231209 90.806686 91.382187]   C1[10, 1..3] = [91.564354 92.206512 92.848671]
C[11, 1..3] = [90.244049 90.819611 91.395172]   C1[11, 1..3] = [91.577835 92.220093 92.862350]
C[12, 1..3] = [90.256882 90.832535 91.408188]   C1[12, 1..3] = [91.591324 92.233673 92.876022]
C[13, 1..3] = [90.269722 90.845459 91.421188]   C1[13, 1..3] = [91.604813 92.247246 92.889702]
C[14, 1..3] = [90.282562 90.858383 91.434189]   C1[14, 1..3] = [91.618301 92.260841 92.903374]
C[15, 1..3] = [90.295403 90.871300 91.447205]   C1[15, 1..3] = [91.631783 92.274422 92.917053]
C[16, 1..3] = [90.308243 90.884224 91.460205]   C1[16, 1..3] = [91.645271 92.288002 92.930717]
C[17, 1..3] = [90.321091 90.897156 91.473213]   C1[17, 1..3] = [91.658760 92.301575 92.944405]
C[18, 1..3] = [90.333923 90.910080 91.486214]   C1[18, 1..3] = [91.672241 92.315155 92.958084]
...
```

blocks

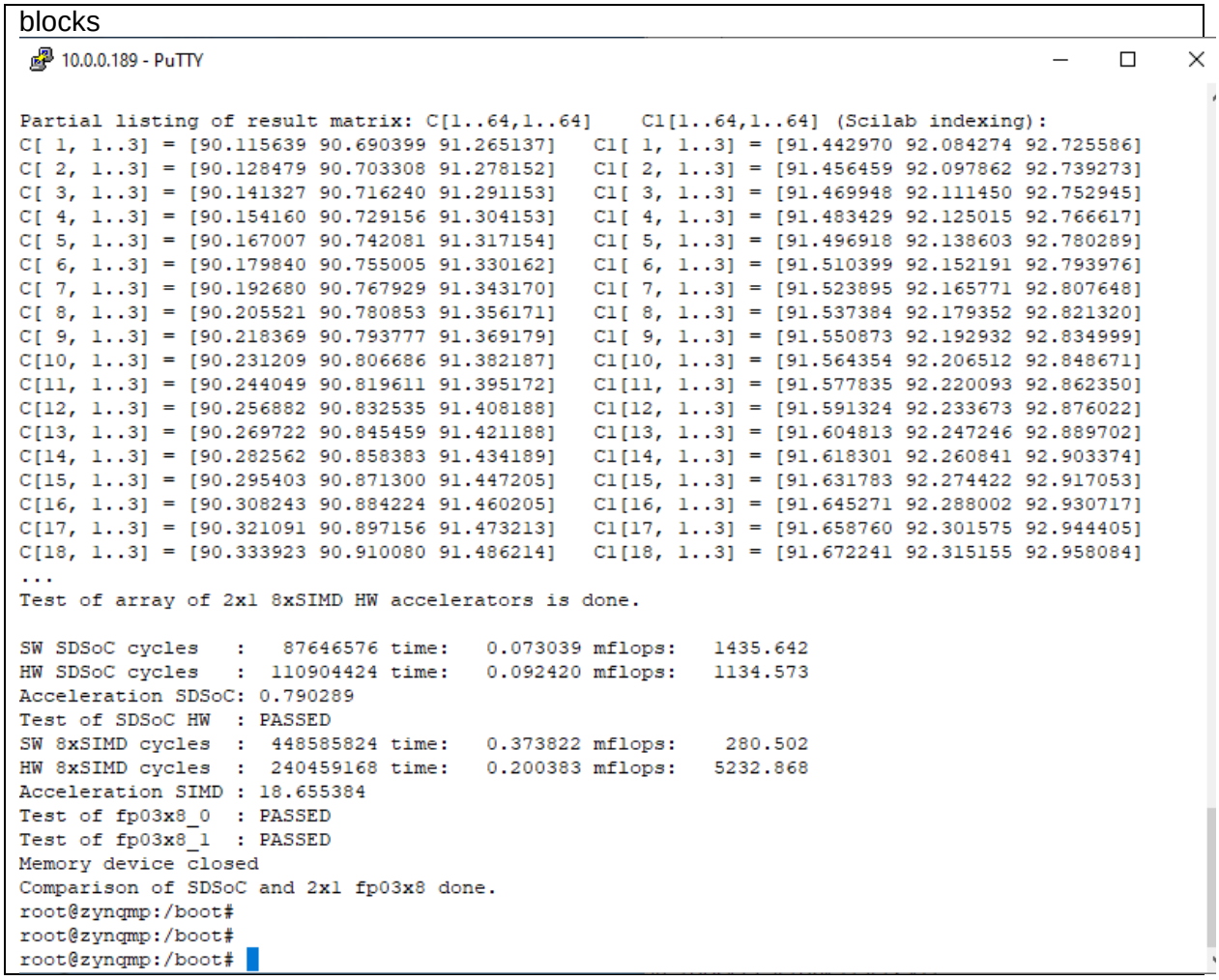


Figure 10: Console output from matrix multiplication: matmultf0.elf.

Listening (see Figure 10) indicates that the application is started from the directory /boot . Path to the shared library has to be exported before tests. The test app is started by executing ./matmultf0.elf SW application.

The test app. performs FP32 matrix multiplication of matrices with size [64x64] as:

- SW with standard scalar Arm A53 FPU unit.
- SW with Arm A53 SIMD NEON unit with SW composing the final [64x64] matrix multiplication from fixed size [16x16] blocks.
- SDSoC HW accelerator of matrix multiply and matrix add [16x16] with SW support composing the final [64x64] matrix multiplication from the fixed size [16x16] HW accelerated blocks.
- Two 8xSIMD HW accelerators

FP32 performance of 03CG, 03EG and 04EV accelerators

Function		MFLOP/s
Matrix Multiplication [64,64] 1 thread scalar FPU	A53 SW:	280
Matrix Multiplication [64,64] 1 thread NEON	A53 SW:	1435
MM (without copy of matrices to DDR4) 1 thread	2x 8xSIMD:	5232
MM (based on HW MultiplyAdd[16,16]) 1 thread	SDSoC HW:	1134
Matrix Multiplication [64,64] 2 threads scalar FPU	A53 SW:	845
Matrix Multiplication [64,64] 2 threads NEON	A53 SW:	3736
MM (without copy of matrices to DDR4) 2 threads	2x 8xSIMD:	5077
MM (based on HW MultiplyAdd[16,16]) 1 thread	SDSoC HW:	1127
Matrix Multiplication [64,64] 1 thread scalar FPU	A53 SW:	284
Matrix Multiplication [64,64] 1 thread NEON	A53 SW:	1435
MM (with copy of matrices to DDR4) 1 thread	2x 8xSIMD:	4046
MM (based on HW MultiplyAdd[16,16]) 1 thread	SDSoC HW:	1135
Matrix multiplication [64,64] 2 thread scalar FPU	A53 SW:	851
Matrix multiplication [64,64] 2 threads NEON	A53 SW:	3690
MM (with copy of matrices to DDR4) 2 threads	2x 8xSIMD:	4122
MM (based on HW MultiplyAdd[16,16]) 1 thread	SDSoC HW:	1127
Vector copy from A[4096] to B[4096]	2x 8xSIMD:	2944
	A53 SW:	448
Vector ADD: Z[4096] = A[4096] .+ B[4096]	2x 8xSIMD:	2942
	A53 SW:	207
Vector ADD: B[4096] = A[4096] .+ Z[4096]	2x 8xSIMD:	2942
	A53 SW:	200
Vector ADD: A[4096] = B[2096] .+ Z[4096]	2x 8xSIMD:	2940
	A53 SW:	199
Vector copy from B[4096] to A[4096]	2x 8xSIMD:	2943
	A53 SW:	407
Vector DIV: Z[4096] = A[4096]./B[4096]	2x 8xSIMD:	2589
	A53 SW:	156
Vector MAC	2x 8xSIMD:	293
	A53 SW:	390
Vector MSUBAC	2x 8xSIMD:	293
	A53 SW:	378
Vector MUL: Z[4096] = A[4096] .* B[4096]	2x 8xSIMD:	2940
	A53 SW:	192
Vector MUL: B[4096] = A[4096] .* Z[4096]	2x 8xSIMD:	2941
	A53 SW:	200
Vector MUL: A[4096] = B[2096] .* Z[4096]	2x 8xSIMD:	2940
	A53 SW:	193
Vector VPROD	2x 8xSIMD:	3260
	A53 SW:	553
Vector VPRODS8	2x 8xSIMD:	2507
	A53 SW:	628
Vector SUB: Z[4096] = A[4096] .- B[4096]	2x 8xSIMD:	2942

	A53 SW:	190
Vector SUB: B[4096] = A[4096] .- Z[4096]	2x 8xSIMD:	2940
	A53 SW:	211
Vector SUB: A[4096] = B[2096] .- Z[4096]	2x 8xSIMD:	2862
	A53 SW:	197
Vector copy from Z[4096] to A[4096]	2x 8xSIMD:	2940
	A53 SW:	379
Vector copy from Z[4096] to B[4096]	2x 8xSIMD:	2941
	A53 SW:	424

5 Power consumption

Power consumption is measured on input power line 5V.

Power consumption TE0820-03CG-1E 1GB module TE0706 carrier bd.	Power [W]
Linux system is running with all HW. No user app.	7,20
Linux system is running with all HW 8xSIMD accelerated matmultf0.app	7,85

Power consumption TE0820-03EG-1E 1GB module TE0706 carrier bd.	Power [W]
Linux system is running with all HW. No user app.	7.75
Linux system is running with all HW 8xSIMD accelerated matmultf0.app	9,20

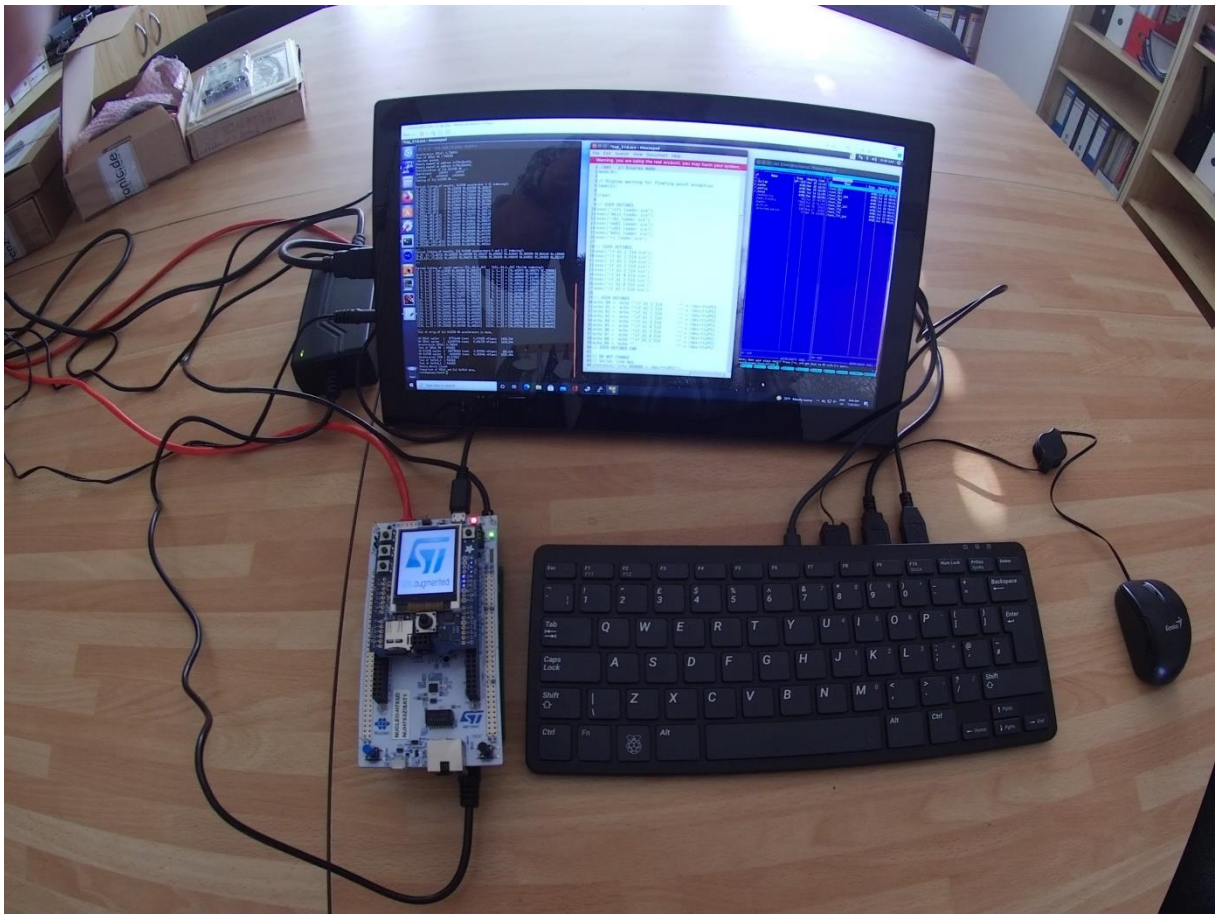


Figure 11: Terminal is running A53 matmultf0 application. Control from remote desktop.

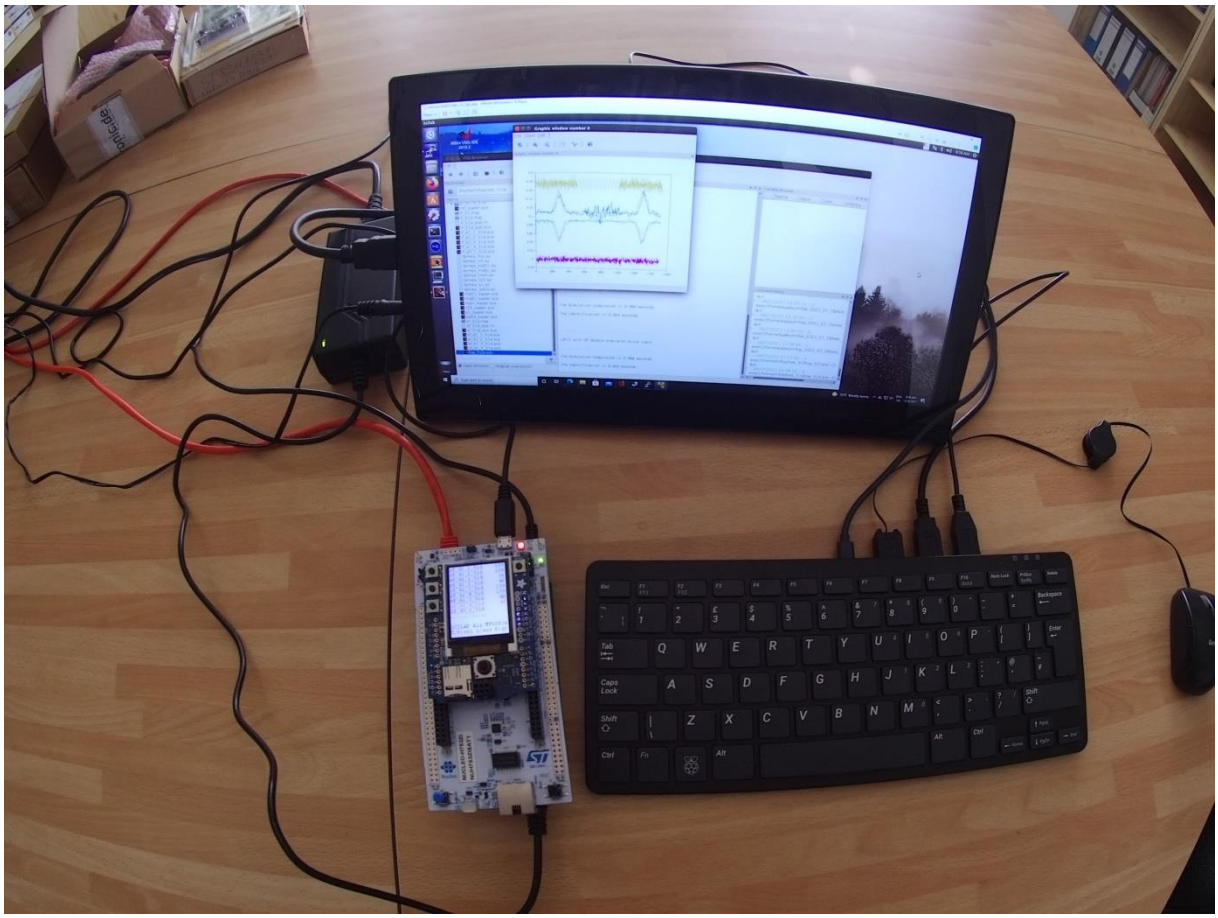


Figure 12: Control of execution of A53 scilab by STM32H753 terminal joystick menu.

6 ILA – In-circuit Logic Analyzer

System includes HW IP of the Vivado-Lab tool 2018.2 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 250 MHz clock. It is configured to sample 1024 data. See Figure 13.
Figure 13 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 13.

In ILA, we can zoom to see the details. See Figure 14. The we_op_1 == 1 and op_1 == 1 is the trigger condition for ILA set by user. The we_op_1 can be seen in the first line of ILA.
The address bus related to Z z_addr_1 starts to increment, followed by address buss related to A a_addr_1. The signal z_we_1 is set to 1 to write the data from A to Z in all 8xSIMD memories in parallel.

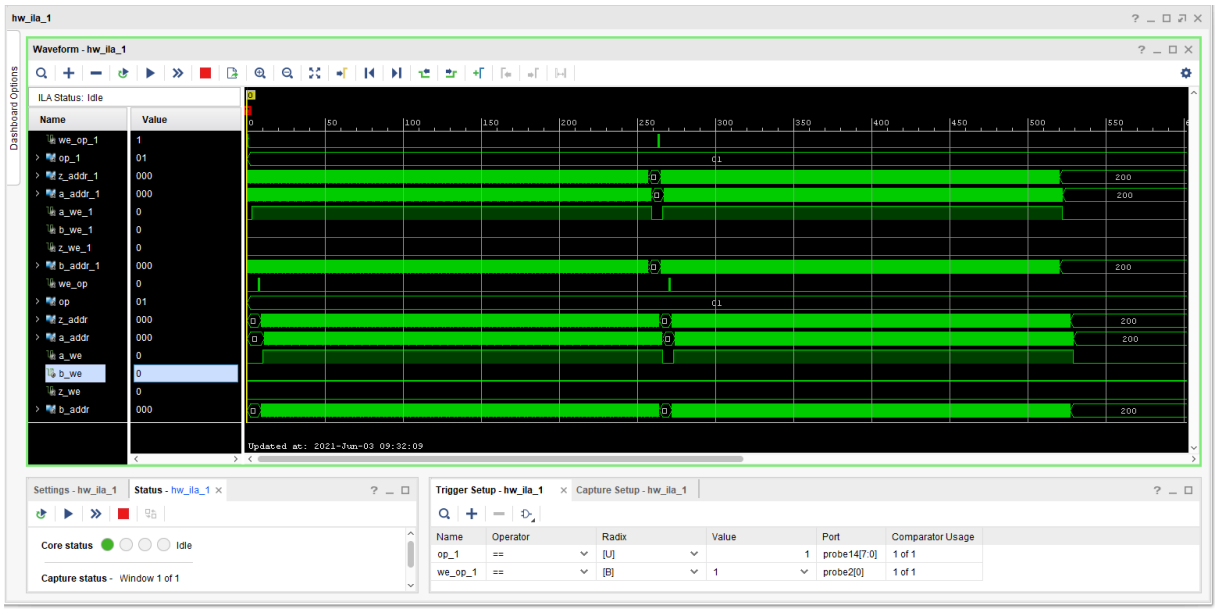


Figure 13: Instruction vz2a.

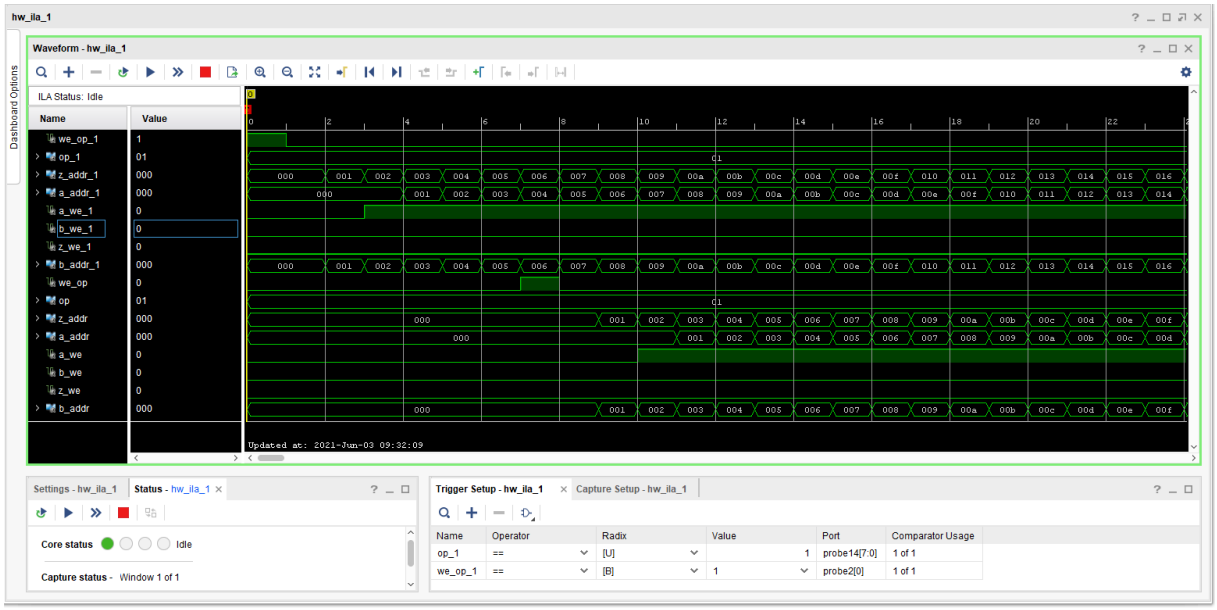


Figure 14: Instruction vz2a detail.

Figure 14 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.

7 License

This evaluation package includes precompiled system with two **evaluation versions** of the accelerator:

- **FP03x8** with **capabilities = 40** described in Figure 7 and Figure 8**.**

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

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 two 8xSIMD, FP03x8 accelerators in a bitstream. The evaluation package presents these system properties:

The run-time reconfigurable floating point accelerators for the Zynq Ultrascale+ devices 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 tho FP32x8 SIMD HW accelerators is provided in form of a shared libraries. The provided shared libraries API are compatible with the standard C (gcc compiler) and C++ (g++ compiler) SW design flows. Scripts are auto generated for the ARM A53 Debian OS “make” utility.
3. The two FP32x8 SIMD HW hardware of floating point accelerators have fixed design. Their run-time reconfiguration is performed by reprogramming the firmware code. The firmware defines what sequences of operations will do the programmable finite state machines (FSMs) inside of the accelerators.
4. Data communication is implemented by AXI-stream.
5. HW data movers are defined in design time and cannot be changed during the run time. The following variants are possible:
    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.

    The released evaluation package includes precompiled variants a) and b).

    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 alternatives of HW data movers.
6. SW API can 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. SW API can query and identify information about the actual status of the HW license defined in each 8xSIMD 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 SW 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 future use the Ultra RAM blocks (4096x64b) present in Zynq UltraScale+ device ZU04EV without affecting the accelerator library API or user code. The released evaluation package is not using Ultra RAM blocks.

Reconfiguration of accelerator by change of firmware

The 8xSIMD FP32x8 HW accelerators execute 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 of each 8xSIMD HW accelerator can contain multiple pre-loaded sequences of VLIW instructions.

Computation performed in the 8xSIMD HW accelerators 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.

SW flexibility and performance of SDSoC accelerators

The precompiled fixed FP32 matrix multiplication and matrix add HW accelerator can work only with the [16x16] block. In case of larger size matrix multiplication with size in multiples of 16 the fixed block can be reused by the SW application.

SW flexibility and performance of Arm A53 NEON accelerator

Presented FP32 matrix multiplication on Arm A53 NEON accelerator was also defined and tested only for fixed [16x16] block. The C and C++ compilers with –o3 optimization can map the FP32 matrix multiplication computation to the NEON accelerator in this case. In case of larger size matrix multiplication with size in multiples of 16 the fixed block can be reused by the SW application.

Released evaluation package presents also use of two A53 NEON accelerators in case of FP32 matrix multiplication on two SW threads.

Comparison of effectivity of used HW resources

The multiplication part of the fixed SDSoC HW accelerator is using 16 FP32 ADD HW IP cores and 16 FP32 MULT HW IP cores. The two 8xSIMD HW accelerators are together also using 16 FP32 ADD HW IP cores and 16 FP32 MULT HW IP cores. Demonstrated performance of the two 8xSIMD HW accelerators significantly outperforms the fixed SDSoC HW accelerator.

9 References

[1] WAKeMeUp, Wafers for Automotive and other Key applications using Memories, embedded in Ulsi Processors. Project number ECSEL No. 783176
http://www.wakemeup-ecsel.eu/

[2] UM2407 User manual STM32H7 Nucleo-144 boards (MB1364)
file:///C:/Users/kadlec/STM32Cube/Repository/UM2407.pdf

[3] Adafruit, „1.8" TFT Display Breakout and Shield," 10 02 2019. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/1-8-tft-display.pdf?timestamp=1558009255

[4] MPSoC Module with Xilinx Zynq UltraScale+ ZU3CG-1E
https://shop.trenz-electronic.de/en/TE0820-04-3AE21FA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU3CG-1E-2-GByte-DDR4-SDRAM-4-x-5-cm

[5] MPSoC Module with Xilinx Zynq UltraScale+ ZU3EG-1E
https://shop.trenz-electronic.de/en/TE0820-04-3BE21FA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU3EG-1E-2-GByte-DDR4-SDRAM-4-x-5-cm

[6] MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E
https://shop.trenz-electronic.de/en/TE0820-04-4DE21FA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-GByte-DDR4-SDRAM-4-x-5-cm

[7] Trenz Electronic, "Carrier Board for Trenz Electronic 7 Series".
https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261

[8] scilab (5.5.2-4+deb9u1). Scientific software package for numerical computations.
https://packages.debian.org/stretch/scilab

[9] Jiři Kadlec, Lukáš Kohout: INDUSTRIAL 40 NM DEMONSTRATOR NUCLEO STM32H755ZI-Q
http://sp.utia.cz/index.php?ids=results&id=H755ZI-Q

[10] UM2408 User manual STM32H7 Nucleo-144 boards (MB1363)
file:///C:/Users/kadlec/STM32Cube/Repository/UM2408.pdf

[11] SW4STM32 System Workbench for STM32: free IDE on Windows, Linux and OS X
SW4STM32 - System Workbench for STM32: free IDE on Windows, Linux and OS X - STMicroelectronics

[12] STM32CubeH7 Firmware Package V1.5.0 / 28-June-2019
Avalable for download by STM32CubeMX - STM32Cube initialization code generator.
https://www.st.com/en/development-tools/stm32cubemx.html

APPENDIX - Confidence test

This is basic confidence test of the evaluation package for Zynq Ultrascale+ on ZU03CG module.

Unzip evaluation package to Win 10 directory of your choice.
c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg

Precompiled HW and SW projects are located in directory:
c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw

Sd_card image is located in directory:
c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_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 system to the 5V/6A power supply.

TEST

- Zynq Ultrascale+ will start to boot the Debian OS.
- Open Putty terminal. Set it to:
  (115200 bps, 8 data bits, stop bit 1, parity none, flow control off)
- Use Putty terminal to login as user: **root** password: **root**
- Change directory to **/boot**
- To export path to the shared library, type in Debian terminal:
  export LD_LIBRARY_PATH=/boot
- Start application code by typing:
  ./matmultf0.elf

RESULT

- The application will compute single precision floating point matrix multiplications
  - In SW on ARM A53 with sequential FPU
  - In SW on ARM A53 on SIMD NEON FPU
  - In SDSoC HW accelerator supporting the [16x16] matrix mult and matrix add tiles.
  - In HW on two 8xSIMD FP03x8 accelerators.
- Results of ARM and HW accelerated computations are compared to be identical and MFLOP/s performance is displayed. See Figure 10.

Compilation and debug of projects from source code

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

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

c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw

Projects in this directory link to the same shared library:

c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\Release\sd_card

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 the Xilinx SDK 2018.2 tool on Win 10 PC.

DEBUG of SW application from Xilinx SDK 2018.2

The application can be executed or debugged from the SDK 2018.2 tool.

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

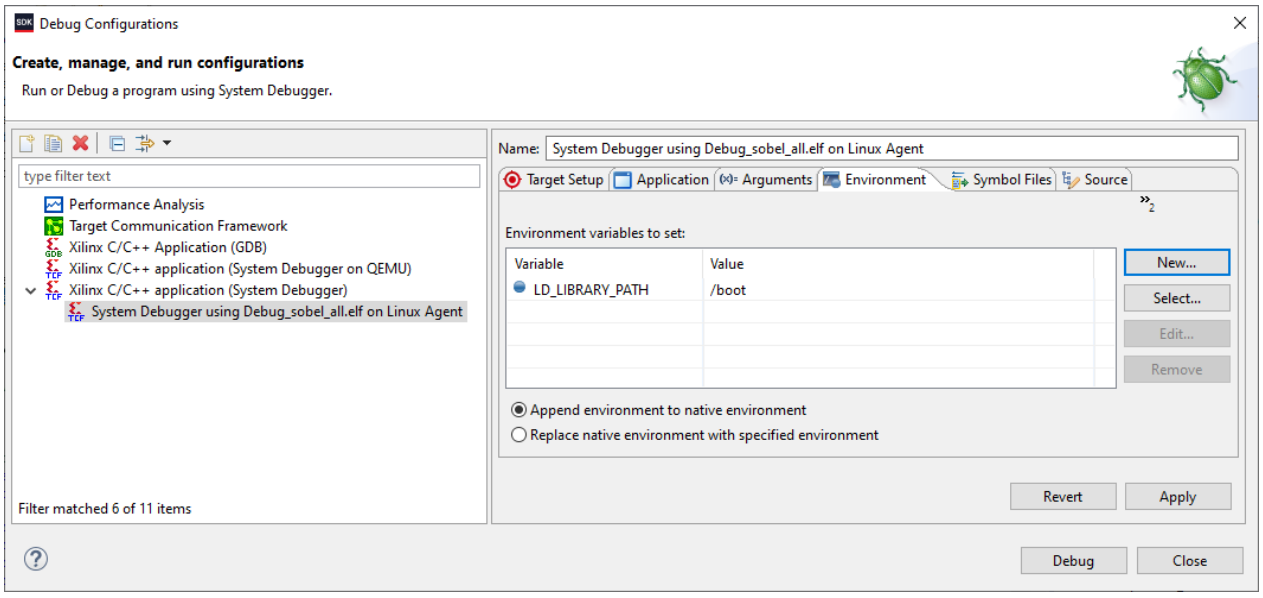


Figure 15: Define the environment variable.

Before start of Debug, copy content of this directory to the sd_card directory

c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\Debug\sd_card*.*

to SD card top directory (visible in the Win 10 file explorer).
`*.*`

Enter the SD card to the board and power ON the board.

Alternatively, you can use Ethernet to 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. This can be tested. See Figure 16.

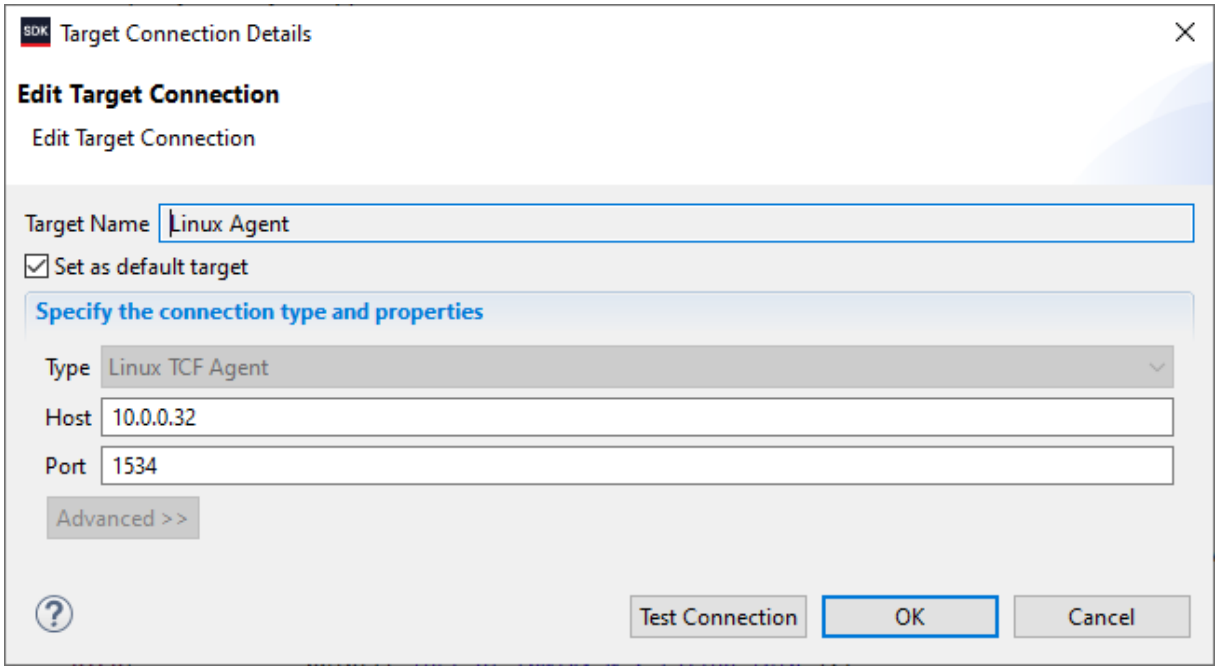


Figure 16: Test connection to Linux TCF Agent.

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 **`gcc`** C compiler or 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_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\ matmultf0\`

to the Debian file system into directory:

```
/home/fp03x8_v26_2x1_uart1_ila/fp03x8_v26x_2x1_zc_muladdf_sw/
matmultf0/
```

To compile in Arm Debian the **matmultf0** project, change the directory to:

```
cd      /home/fp03x8_v26_2x1_uart1_ila/fp03x8_v26x_2x1_zc_muladdf_sw/
matmultf0/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:

```
./matmultf0.elf
```

You are done. See the application running on the board. See Figure 10.

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

```
halt
```

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

- Remove the SD card.

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

To power down, take these steps.

- Close PuTTY terminal and disconnect USB connection to PC.
- Remove the power supply of the board.

Use of C MEX functions in Scilab

In Debian terminal, change directory to

/home/mflop/test_51d

Start Scilab by typing

scilab

In Scilab, execute script

top_51d.sce

This script is controlled from STM32H753 terminal GUI. It will execute mex_md51**()** C MEX function present in shared library **libmex_md51.so** and generate several reference header files for STM32H7 benchmarks in the current directory and measures the double precision MFLOP performance of mex_md51**()** C MEX function on ARM A53.

Quit Scilab by typing.

quit

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

See execution of the **top_51d.sce** script in Scilab on remote X11 Desktop in Figure 12.

10 APPENDIX – System design guidelines

Design flow 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:	7
Device:	xczu3cg-sfvc784-1-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/wakemeup

Guide for compilation of HW

1. Unpack evaluation package to Win10 directory. TERMINAL tool is in: `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\`
   Change directory to:
   ```
   c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\fp03x8_v26_2x1_uart1_ila\zusys\
   ```
2. Add the UTIA 8xSIMD HW IP to the package as the directory `\ip`
   ```
   c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\fp03x8_v26_2x1_uart1_ila\zusys\ip_lib\ip
   ```
3. On Win10, open dos terminal window, change directory to the folder
   ```
   c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\fp03x8_v26_2x1_uart1_ila\zusys\
   ```
4. To overcome limitations of Win10 related to the need of short directory paths, use the script *_use_virtual_drive.cmd* to create a virtual short path to your directory drive *X:\zusys* Type command:
   ```
   _use_virtual_drive.cmd
   ```
   Select `X` as name of the virtual drive and select `0` to create the virtual drive.
   Go to the created virtual short-path directory by typing in the win 10 terminal:
   ```
   X:
   cd zusys
   ```
5. Use text editor of your choice and open and modify script `design_basic_settings.sh`
   Select correct path to SDSoC 2018.2 tool installed on your Win7 or Win10. Line 38:
   ```
   @set XILDIR=C:/Xilinx
   ```
   Select proper Xilinx device:
   ```
   @set PARTNUMBER=7
   ```
   The selected number corresponds to the number defined in file `X:\zusys\board_files\TE0820_board_files.csv`
   Verify, if line 78 of script *design_basic_settings.sh* sets the SDSoC flow support by: *ENABLE_SDSOC=1*
   ```
   @set ENABLE_SDSOC=1
   ```
6. Start the Xilinx Vivado 2018.2 and create the design by executing of script:
   ```
   X:\zusys\vivado_create_project_guimode.cmd
   ```
7. Optional:
   You can use Vivado automation and to the created HW design the In circuit Logic Analysator (ILA) monitor to enable capturing of selected accelerator outputs of your choice.
8. In Vivado console, execute command:
   ```
   TE::hw_build_design -export_prebuilt
   ```
   After the Vivado compilation, new hardware description file `zusys.hdf` is generated in folder:
   ```
   X:\zusys\prebuilt\hardware\3cg_1e_1gb\
   ```

Guide for configuration and compilation of PetaLinux

The configuration and compilation of the *Petalinux 2018.2* kernel and *Debian 9.8 Stretch* image for the Zynq Ultrascale+ module `TE0820-03-03CG-1E` with Zynq Ultrascale+ device `xczu3cg-sfvc784-1-e` device is described now. The configuration has to be performed in the Ubuntu 16.04 LTS OS.

We describe use of the 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_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/
   /home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_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 TE0820-03-03CG-1E:

   ```
   cd
   /home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/petalinux/
   ```

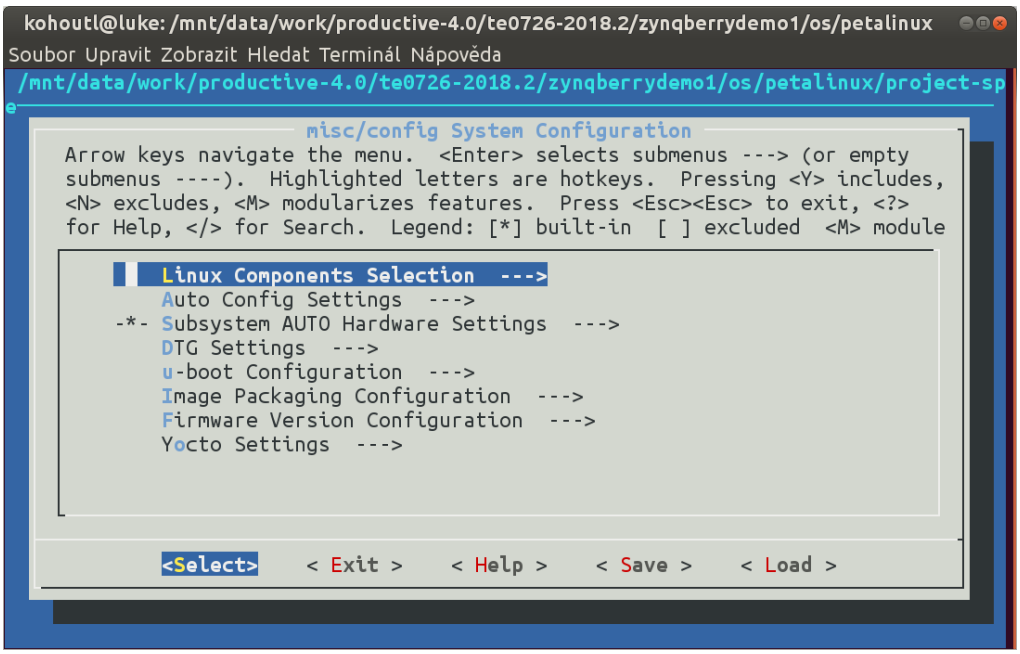
   It contains a predefined configuration according to Zynq Ultrascale+ board requirements.

4. The zusys.hdf file created in Win 10 in Vivado 2018.2 tool is present in the Ubuntu folder:

   ```
   /home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/prebuilt/hardware/3cg_1e_1gb/
   ```

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

   ```
   petalinux-config --get-hw-description=
   /home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/prebuilt/hardware/3cg_1e_1gb/
   ```



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

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

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

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

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

8. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

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

```
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_il
a/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_il
a/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_il
a/zusys/os/petalinux/images/linux/bl31.elf
```

Guide for configuration and compilation of Debian OS

The file system is based on the Debian 9.8 Stretch distribution. Follow the steps below.

10. Go to the folder with PetaLinux:

```
cd
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_il
a/zusys/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.

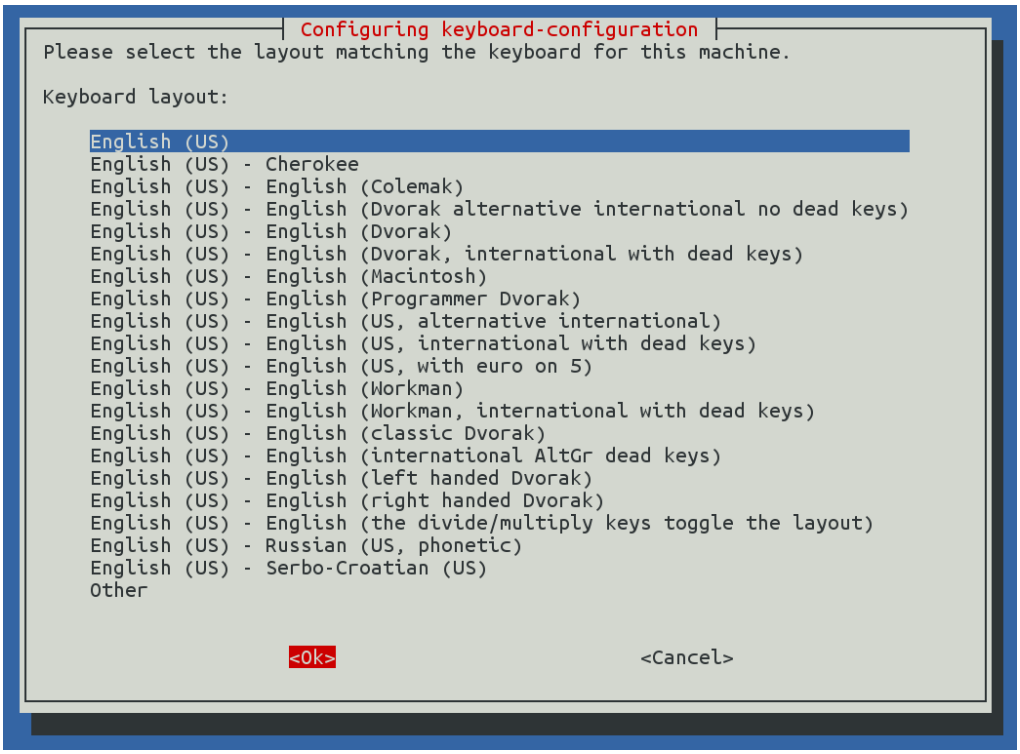
Tool	*Package*
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-arm-static	qemu-user-static

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 *TE0820-debian.img*, its size will be 7 GB.



13. Compress the created image to file TE0820-debian.zip:

```
zip TE0820-debian TE0820-debian.img
```

14. Copy compressed image file from Ubuntu

```
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/petalinux/te0820-debian.zip
```

to Win 10 file:

```
X:\zusys\prebuilt\os\petalinux\default\te0820-debian.zip
```

15. Copy these files from Ubuntu

```
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_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/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/petalinux/TE0820-debian.zip
/home/devel/work/TS82fp03x8_TE0706_TERMINAL_zu3cg/fp03x8_v26_2x1_uart1_ila/zusys/os/petalinux/TE0820-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 *TE0820-debian.zip*
- U-boot program *u-boot.elf*
- Support firmware *bl31.elf*

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

Guide for creation of SDSoC platform OS

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

    ```
    TE::sw_run_hsi
    ```

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

    ```
    X:\zusys\prebuilt\boot_images\3cg_1e_1gb\u-boot\BOOT.bin
    ```

21. 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\TE0820-02\03CG-1E\
    ```

    and it is also packed into the ZIP file.

Guide for creation of shared library and HW kernel

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

    ```
    _use_virtual_drive.cmd
    ```

    and response (1)

23. Change directory to

    ```
    c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila\SDSoC_PFM\TE0820-02\03CG-1E\
    ```

24. 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:*\ 03CG-1E

    ```
    _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 03CG-1E
    ```

25. Open SDSoC project in directory

    ```
    X:\03CG-1E
    ```

26. In SDSoC import one C and one C++ HW kernel design project

    ```
    fp03x8_v26x_2x1_zc_muladdf_c_hw
    fp03x8_v26x_2x1_zc_muladdf_hw
    ```

    from the directory

    ```
    c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila\SDSoC_PFM_src\TE0820-02\03CG-1E\
    ```

27. Define the custom SDSoC platform

`X:\03CG-1E\zusys`

28. Change both imported projects from `Debug` to the `Release` compilation target
29. Compile both projects by the Xilinx SDSoC 2018.2 compiler. Compilation time is cca 3 hours (in case of Win10 I7 laptop with 16 GB memory).
30. Result of compilation is the SD card content with BOOT.bin file shared object library definition files in directories:

    `X:\03CG-1E\fp03x8_v26x_2x1_zc_muladdf_c_hw\Release\sd_card\`

    `X:\03CG-1E\fp03x8_v26x_2x1_zc_muladdf_hw\Release\sd_card\`
31. Copy content of the C project directory

    `X:\03CG-1E\fp03x8_v26x_2x1_zc_muladdf_c_hw\Release\sd_card\`

    to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_c_sw\Debug\`

    and also to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_c_sw\Release\`

    Optional:

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

    `X:\03CG-1E\fp03x8_v26x_2x1_zc_muladdf_c_hw\Release\_sds\p0\vivado\prj\prj.runs\impl_1\`

    to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_c_sw\Debug\`

    and also to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_c_sw\Release\`
32. Copy content of the C++ project directory

    `X:\03CG-1E\fp03x8_v26x_2x1_zc_muladdf_hw\Release\sd_card\`

    to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\Debug\`

    and also to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\Release\`

    Optional:

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

    `X:\03CG-1E\fp03x8_v26x_2x1_zc_muladdf_hw\Release\_sds\p0\vivado\prj\prj.runs\impl_1\`

    to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\Debug\`

    and also to

    `c:\home\work\TS82fp03x8_TE0706_TERMINAL_zu3cg\zu3cg_deb_2x1_eval_uart1_ila_release\fp03x8_v26_2x1_uart1_ila\fp03x8_v26x_2x1_zc_muladdf_sw\Release\`
33. Clean both projects.

34. Close SDSoC tool.

Guide for retargeting for Zynq Ultrascale+ device/module

The above described design flow has been configured for Trenz Electronic module with ID=7, module TE0820-02-03CG-1E, part xczu3cg-sfvc784-1-e, memory 1GB with short module name 3cg_1e_1gb.

The evaluation package includes also pre-configured scripts and files for Trenz Electronic module with ID=5, module TE0820-02-03EG-1E, part xczu3eg-sfvc784-1-e, memory 1GB with short module name 3eg_1e_1gb.

The evaluation package includes also pre-configured scripts and files for Trenz Electronic module with ID=15, module TE0820-03-04EV-1EA, part xczu4ev-sfvc784-1-e, memory 2GB with short module name 4ev_1e_2gb.

These three sets of preconfigured design scripts can be modified to target another Trenz Electronic module. See list of Trenz Electronic modules in Chapter 2 of this application note.

After the change of the ID of the module, the design flow requires also the corresponding version of the 8xSIMD HW IP core.

Contact UTIA to get license for the required HW IP version for the selected module ID.

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 Vodarenskou vezi 4 18200 Prague 8 Czech Republic Jiri Kadlec email: kadlec@utia.cas.cz tel: +420 2 6605 2216

Perform all design steps as described in this application mote for the TE0820-02-03CG-1E module for the targeted module.

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.