http://sp.utia.cz

Application Note

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

STM32H753 Terminal with TE0723-03-07S-1C Accelerator - HW Data Movers

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	6.04.2020	J. Kadlec	Initial draft
1	23.08.2021	J. Kadlec	App. note is addressing details of: HW data movers for TE0723-03-07S-1C

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

ECSEL Joint Undertaking

Table of Contents

Table of Figures

Acknowledgement

This work has been partially supported by ECSEL WAKeMeUP project, No. 783176 [19] and by the corresponding Czech NFA (MSMT) institutional support project MSMT 8A18001.

1 Data movers for TE0723-03-07S-1C board

This application note describes package for evaluation of STM32H753 terminal with ArduZynq HW accelerator module [1], [2] supporting Debian OS.

The terminal is using the STM32H753 Nucleo 144 board [12] or [14] and Adafruit TFT display [17]. The terminal and benchmarks are described in detail in application notes and evaluation packages [8], [9], [10] and [11]. Terminal with Zynq Ultrascale+ device is described in [7].

Programmable logic (PL) of the Zynq device contains HW data movers serving for acceleration of data exchange between DDR3L memory and the PL part of the device. HW data movers write data to a 1024x32bit FIFO HW IP with AXI-Stream data interface.

HW data movers are generated from C source code by the Xilinx SDSoC 2018.2 compiler [3] in the HW design phase. Several types of data movers can be created. This application note serves for description and for comparison of these data movers. Figure 1 describes the top level, SW/HW view of the Zynq system with data movers and the FIFO HW IP.

ZynqBerry TE0723-03-07S-1C board [1] works with Xilinx Zynq XC07007S-1C device with a single core ARM A9 32 bit processor, 512 MB of DDR3L memory and limited size of programmable logic on the single 28 nm chip.

The ZynqBerry TE0723-03-07S-1C board (as well as two similar boards TE0723-03M, TE0723-03-41C64-A with larger Xilinx XC07010-1C chip) are designed and manufactured by company Trenz Electronic [1], [2], [6].

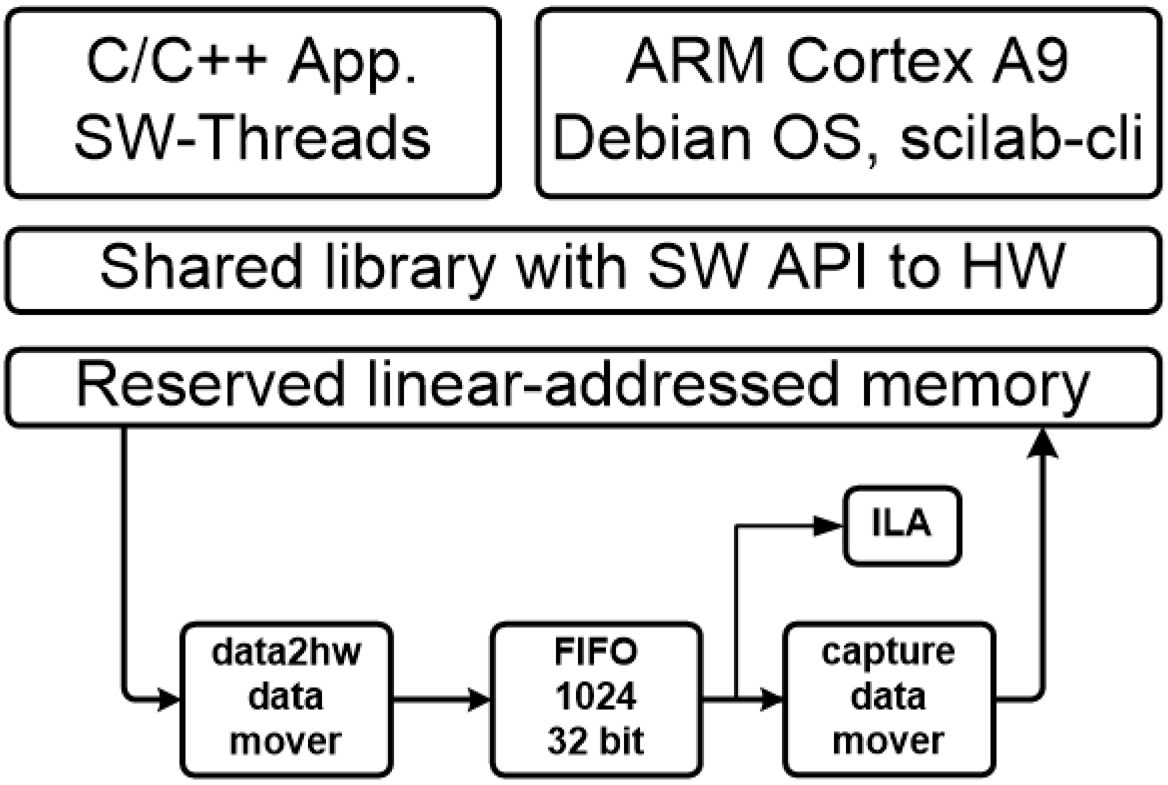


Figure 1: HW accelerated data path in Zynq on the TE0723-03-07S-1C board

Input:

- Data copied via AXI stream interface controlled by HW data mover `data2hw` from the reserved part of the ARM processor DDR3 memory.

Output:

- Data copied via AXI stream interface controlled by HW data mover `capture` to the reserved part of the ARM processor DDR3 memory.

Connectivity:

- AXI stream data input from ARM to the HW data path. The side channel indicates the last transferred word sent.
- AXI stream data output from the HW data path to ARM. The output AXI stream side channel indicates the last transferred word.

Interfaces:	Device:	Clock:
• Data stream AXI-S 32 bit	xc7z007sclg225-1	111 MHz
• ARM A9 system clock	xc7z007sclg225-1	650 MHz

User SW for the Debian OS can be cross-compiled by the g++ compiler in Xilinx SDK [4] on Win 10 PC or Ubuntu PC.

Command „make“ can be also used for compilation of the user C++ SW directly on the A9 processor. The HW data communication is represented for the SW developer as a shared C++ library with simple SW API, identical for several HW data-mover alternatives generated by the Xilinx SDSoC 2018.2 compiler [3].

2 ARM SW API for Streaming of Data

Serial streaming SW API for HW accelerated data movement from/to the non-cacheable linear address space memory is defined by sequence of two calls to these asynchronous non-blocking functions:

```
void data2hw_wrapper(unsigned *src, unsigned len);
void capture_wrapper(unsigned *storage, unsigned len);
```

Example:

```
data2hw_wrapper((unsigned*)A1_A2, len);    //1
capture_wrapper((unsigned*)B1_B2, len);    //2
…
sds_wait(1);
sds_wait(2);
```

`unsigned *src` is pointer to memory start of vector of 32bit wide words of data source
`unsigned *storage` is pointer to memory start of vector of 32bit wide words of data destination

`sds_wait()` synchronization functions are implemented as:

- Functions performing SW pooling in case of DMA and Zero Copy (ZC) data movers.
- Interrupt service routines initiated by an interrupt from the HW data mover in case of the Scatter Gather (SG) HW data movers.

Arm A9 processor can execute some additional SW instructions in parallel with HW accelerated copy of data. This SW code should be located in place marked by the … dots.

Calls to blocking functions `sds_wait(1)` and `sds_wait(2)` is obligatory. ARM A9 processor SW waits there for the complete end of the HW supported data transfer.

All HW data movers supporting the data communication are represented for the SW developer in shared C++ Debian OS libraries.

3 C++ projects for evaluation of HW accelerated copy of data

The evaluation package accompanying this application note contains the Xilinx Vivado 2018.2 base support package HW project, four Xilinx SDSoC 2018.2 HW projects and four Xilinx SDK C++ SW application projects serving for evaluation of four HW data movers.

The base support package HW project contains in PL part of the device one 32 bit wide AXI-Stream FIFO HW IP. The SDSoC 2018.2 projects generate four versions of HW data-movers from specification in format of CPP functions with pragmas. The access functions for these SDSoC generated HW data movers are exported by the SDSoC projects into four shared libraries. The shared libraries are linked with the Debian OS SW user applications in the Xilinx SDK SW projects. Test applications run on the 32 bit single core, ARM A9 processor of the xc7z007sclg225-1 device on the TE0723-03-07S-1C board.

- **Zero-Copy (ZC)** data movers. HW accelerated copy of data from/to reserved, non-cacheable, linear address space memory area.
  Directory (C++): **copy_zc_1x1_sw**
  SW C++ project: **copy_1x1_sw**
  Shared C++ library: **./Debug/sd_card/libcopy_zc_1x1_hw.so**
  Shared C++ library: **./Release/sd_card/libcopy_zc_1x1_hw.so**

- **Dma (DMA)** data movers. HW accelerated copy of data from/to reserved, non-cacheable, linear address space memory area.
  Directory (C++): **copy_dma_1x1_sw**
  SW C++ project: **copy_1x1_sw**
  Shared C++ library: **./Debug/sd_card/libcopy_dma_1x1_hw.so**
  Shared C++ library: **./Release/sd_card/libcopy_dma_1x1_hw.so**

- **Scatter-Gather (SG)** data movers. HW accelerated copy of data from/to reserved, non-cacheable, linear address space memory area.
  Directory (C++): **copy_sg_1x1_sw**
  SW C++ project: **copy_1x1_sw**
  Shared C++ library: **./Debug/sd_card/libcopy_sg_1x1_hw.so**
  Shared C++ library: **./Release/sd_card/libcopy_sg_1x1_hw.so**

- **Scatter-Gather Malloc (SG-malloc)** data movers. HW accelerated copy of data from/to standard Debian OS memory area.
  Directory (C++): **copy_sg_malloc_1x1_sw**
  SW C++ project: **copy_1x1_sw**
  Shared C++ library: **./Debug/sd_card/libcopy_sg_malloc_1x1_hw.so**
  Shared C++ library: **./Release/sd_card/libcopy_sg_malloc_1x1_hw.so**

The shared Debian Stretch 9.8 OS libraries for the SDK 2018.2 C++ SW flow (g++ compiler) provide interfaces to the HW data movers. In all four cases, the **copy_1x1_sw** project demonstrates performance of HW supported data copy for a single precision floating point matrix [64x64].

HW data mover performance is compared with the optimized (-O3) ARM host SW implementation of SW data copy of a single precision floating point matrix [64x64] from user space memory to linear addressable non-cacheable memory area and back to user space memory.

TE0723M-07S	***MByte/s***
ZC HW data movers	166.3
ZC SW copy	19.4
DMA HW data movers	159.0
DMA SW copy	19.4
SG HW data movers	75.7
SG SW copy	19.4
SG-malloc HW data movers	9.6
SG-malloc SW copy	236.4

Figure 2: Performance of data copy for different data movers and for SW.

Matrix [64x64] has 4096 32 bit FP32 (single precision floating point) words. HW supported copy is performed in four blocks of 1024 32 bit words. HW data movers can copy one 32 bit word each 111 MHz clock in all four cases. This corresponds to a peak performance 444 Mbyte/s. However, this performance is not reached due to the data mover initialization SW overhead.

The ARM SW data mover initialization overheads are relatively short in case of `ZC` data movers (see Figure 2, Figure 4) and `DMA` data movers (see Figure 5, Figure 6). Data have to be present in the linear addressable, non-cacheable memory. Se

The initialization overhead is longer in case of `SG` data movers (see Figure 7, Figure 8) even if data are present in the linear addressable, non-cacheable memory. This is due to the overhead related to creation, and release of SW threads implementing the interrupt service functions.

`SG-malloc` data mover (see Figure 9, Figure 10) performs copy of data allocated by standard C++ `malloc()` function in standard, cached Debian user space memory. `SG-malloc` data movers have relatively large SW overhead and relatively low performance for short data vectors (like in the implemented example). The advantage of `SG-malloc` data movers is the possibility to work directly with the Debian user space data allocated by the standard C/C++ `malloc`() function. Also performance of the SW copy is high in the `SG-malloc` case. SW copy is performed by ARM processing system (PS) and works with potentially cached data in the Debian user space. `SG-malloc` data movers work with advanced cache coherent AXI port (ACP) of the Zynq PS.

`SG` and `SG-malloc` data movers (see Figure 7, Figure 8 and Figure 9, Figure 10) use interrupts and interrupt service routines to indicate the end of data mover operation. Interrupt service routines are implemented as non-active Debian OS process threads activated only by the coming interrupt. This solution removes the SW pooling and results in reduced ARM processor load. The associated cost is an additional SW overhead related to creating, execution and termination of interrupt service process threads. `ZC` and `DMA` data movers use SW pooling in the `sds_wait()`synchronization functions. It requires 100% load of one ARM A9 processor core. The TE0723M-07S device works with single core ARM A9 processor.

Four versions of data movers provided in the evaluation package accompanying this application note are presented in Figure 3 - Figure 10.

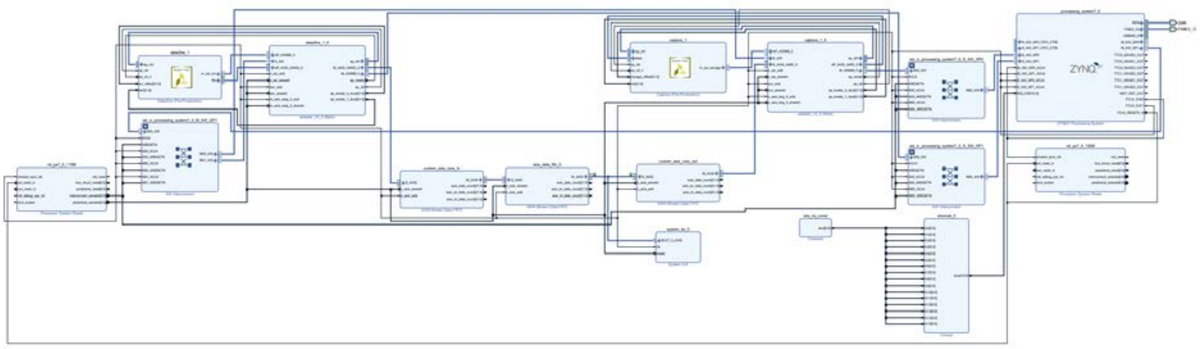


Figure 3: Design with ZC data movers.

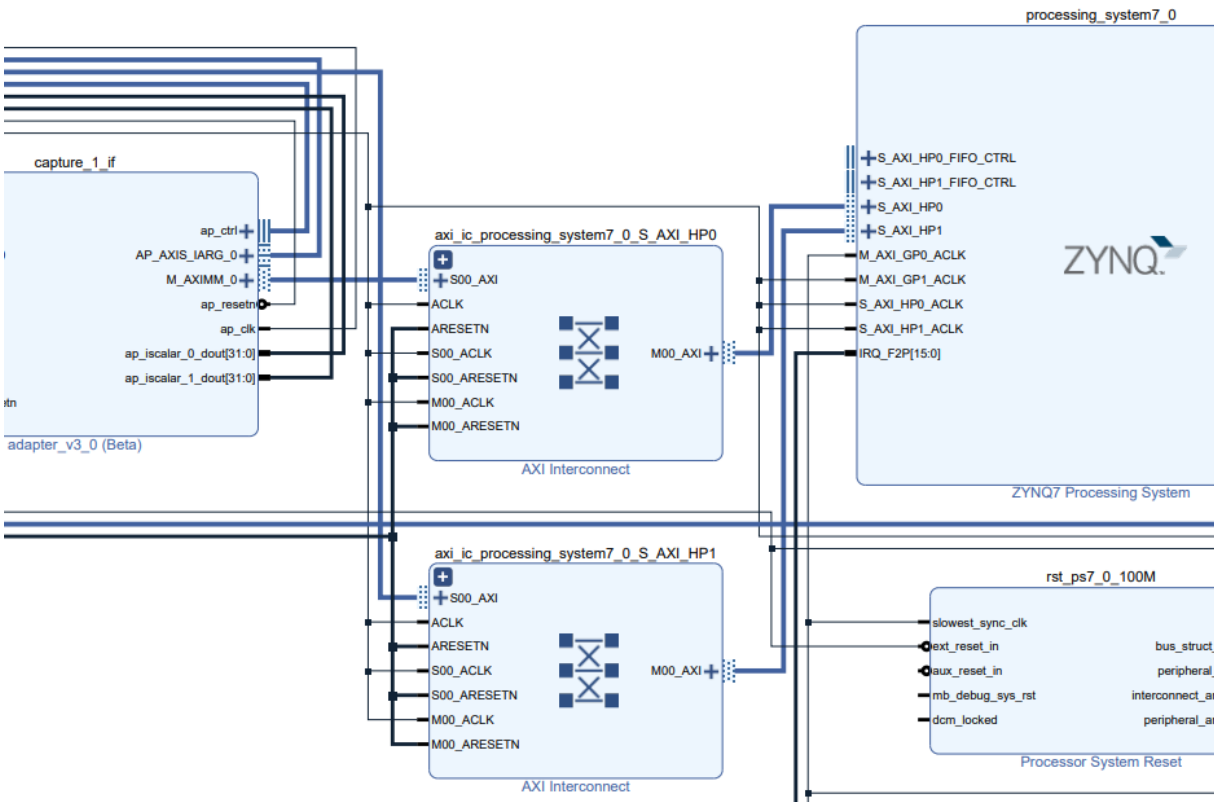


Figure 4: Detail of design with ZC data movers.

File:
`TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila_release_area\copy_zc_1x1_hw\zsys.pdf`

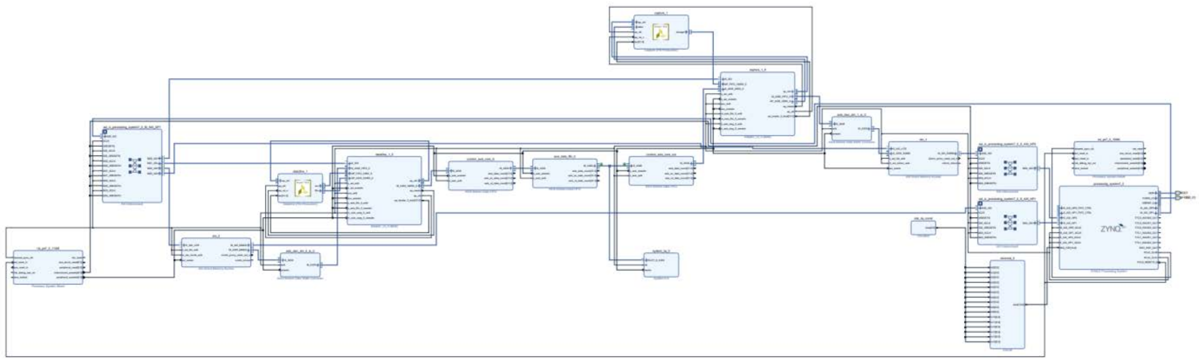


Figure 5: Design with DMA data movers.

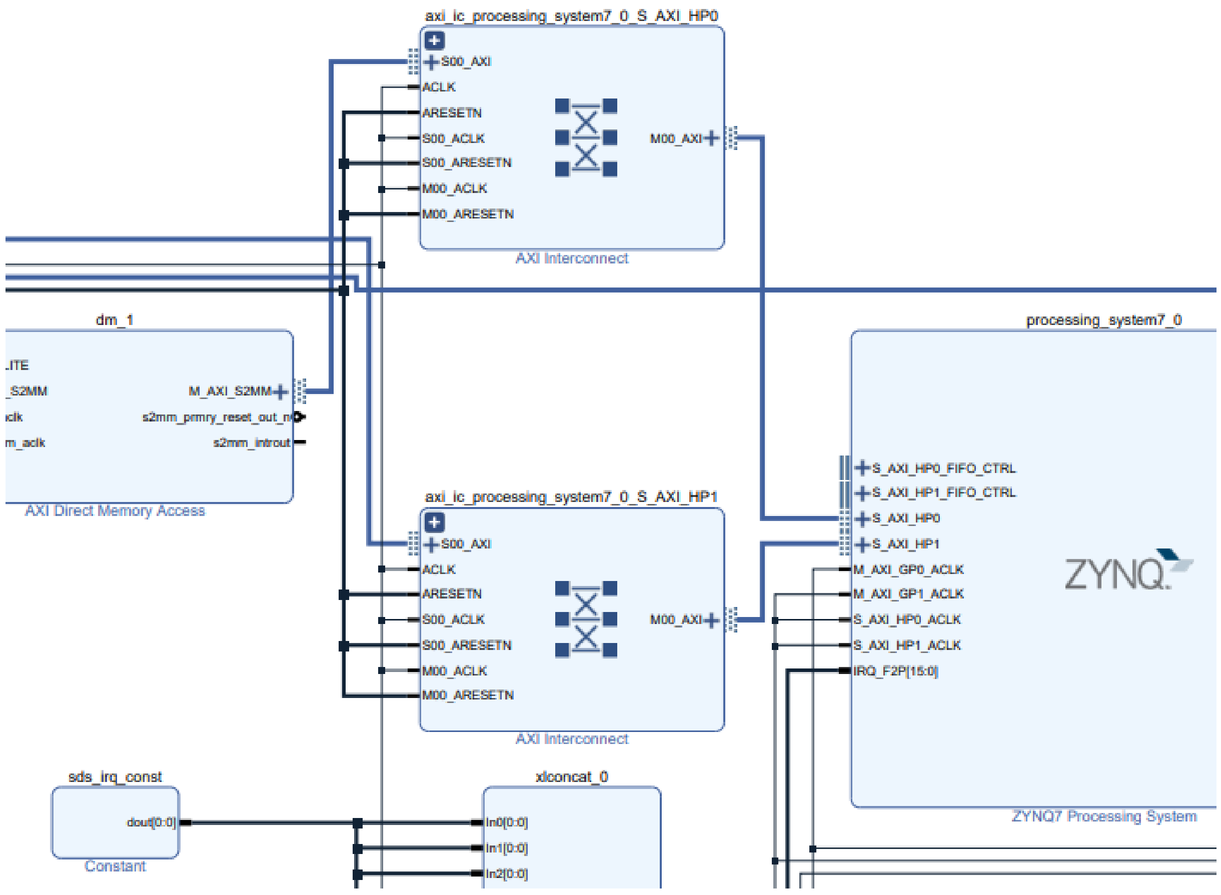


Figure 6: Detail of design with DMA data movers.

File:
`TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila_release_area\copy_dma_1x1_hw\zsys.pdf`

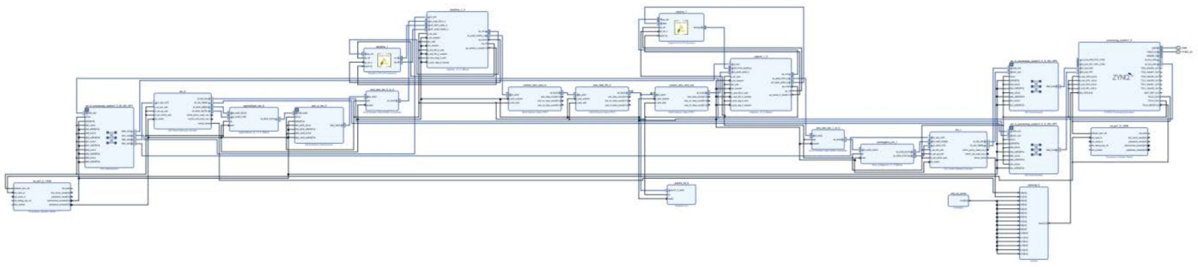


Figure 7: Design with SG data movers.

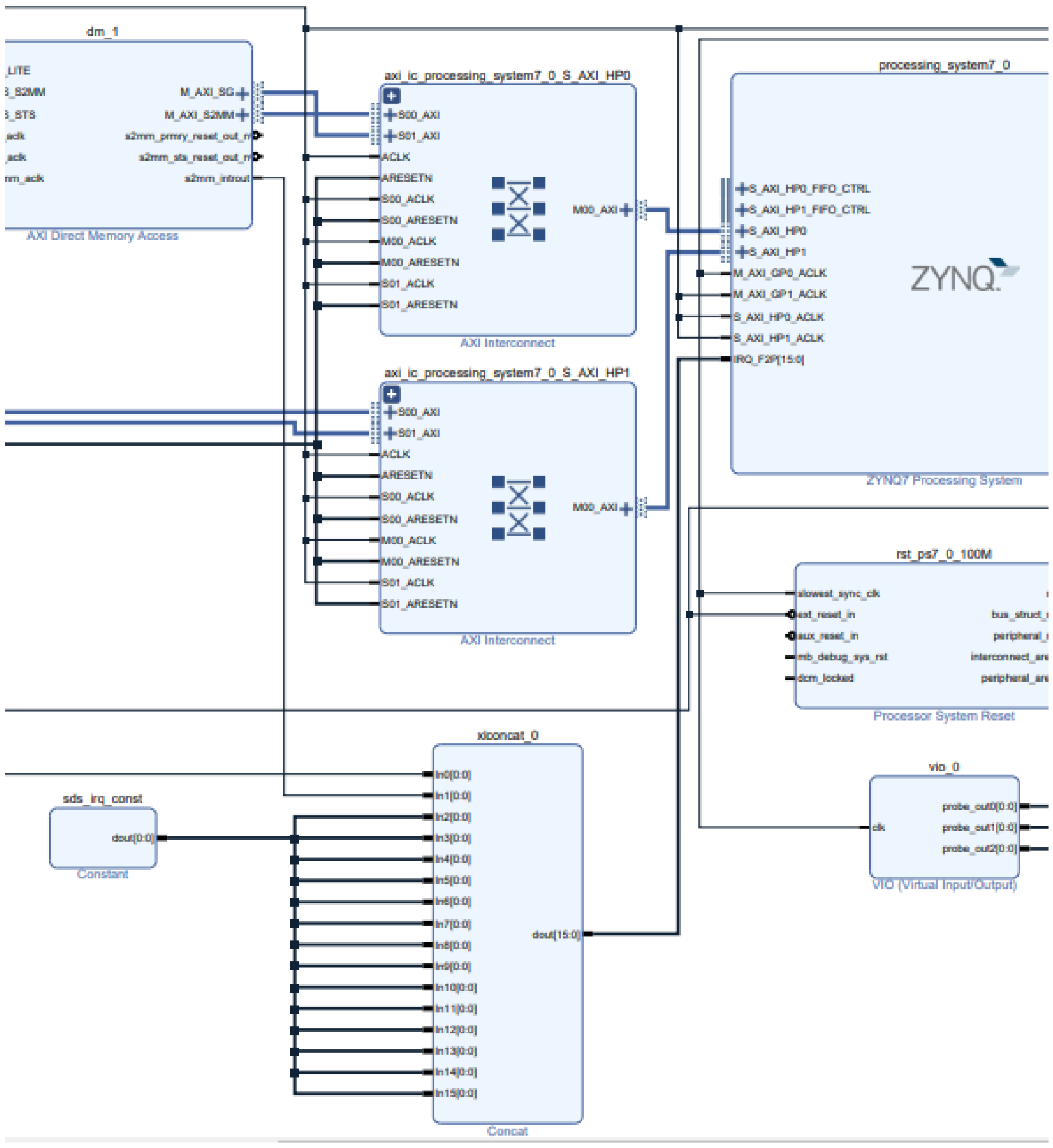


Figure 8: Detail of design with SG data movers.

File:
`TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila_release_area\copy_sg_1x1_hw\zsys.pdf`

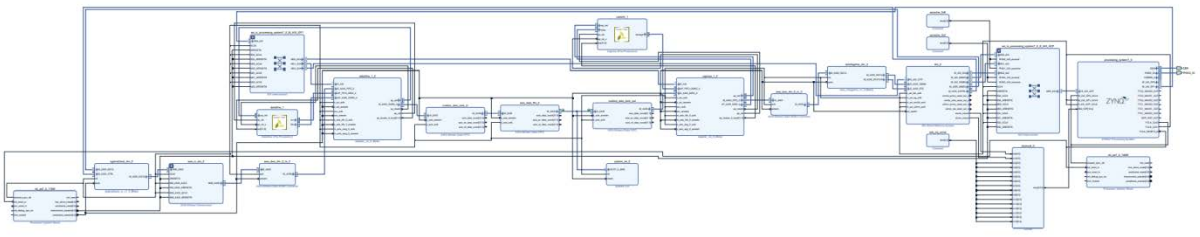


Figure 9: Design with SG-malloc data movers.

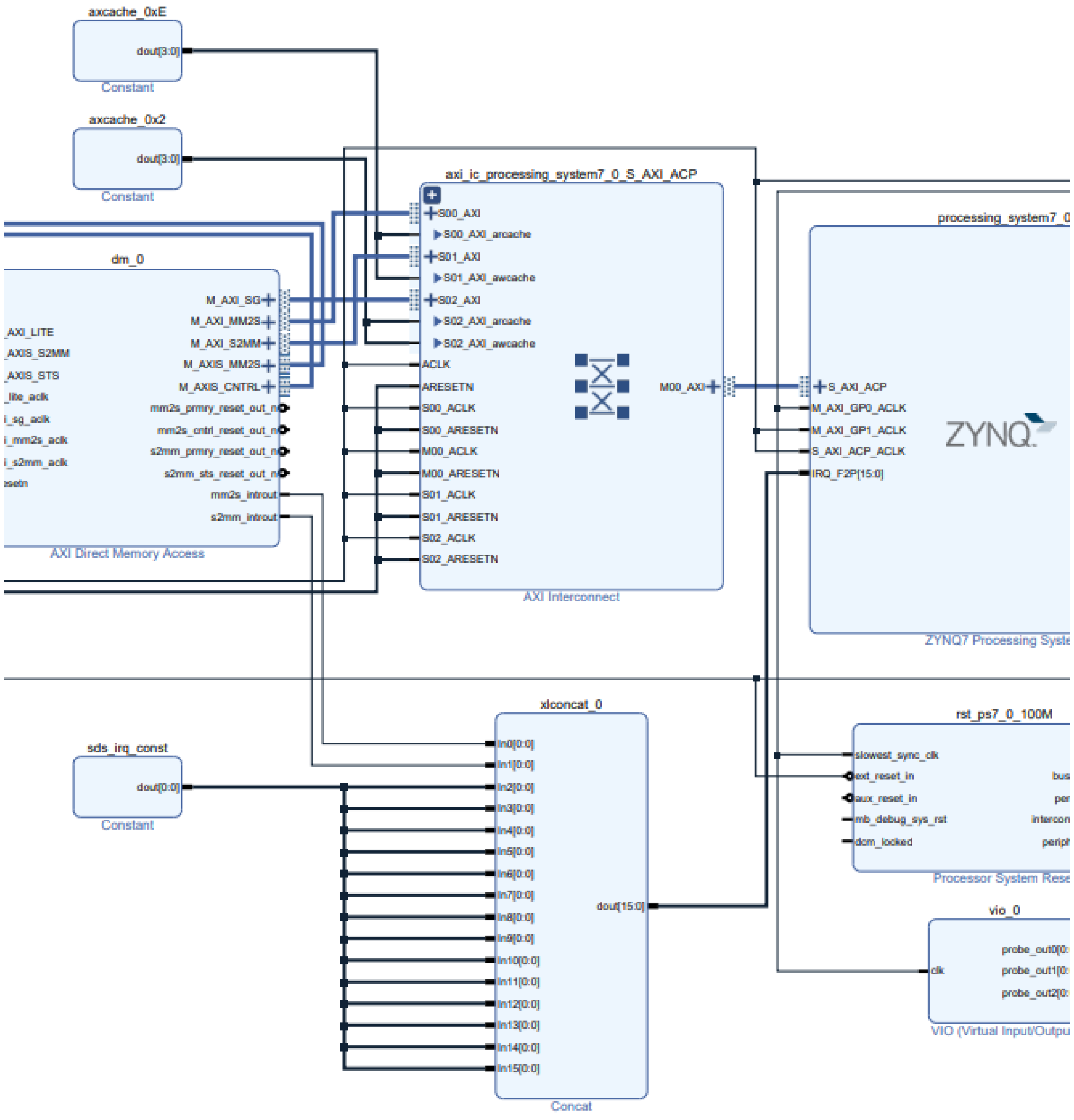


Figure 10: Detail of design with SG-malloc data movers.

File:
`TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila_release_area\copy_sg_malloc_1x1_hw\zsys.pdf`

Data mover programmable logic requirements

TE0723M-07S	***Site Type***	***Used***	***Available***	***Util%***
ZC	Slice LUTs	7704	14400	53.50
ZC	Slice Registers	11113	28800	38.59
ZC	Block RAM Tile	9	50	18.00
DMA	Slice LUTs	8538	14400	59.29
DMA	Slice Registers	12741	28800	44.24
DMA	Block RAM Tile	12.5	50	25.00
SG	Slice LUTs	12936	14400	89.83
SG	Slice Registers	20804	28800	72.24
SG	Block RAM Tile	22	50	44.00
SG-malloc	Slice LUTs	10979	14400	76.24
SG-malloc	Slice Registers	16964	28800	58.90
SG-malloc	Block RAM Tile	17.5	50	35.00

Figure 11: Programmable logic resources used in designs with different data movers.

Design with ZC data mover is using minimal PL resources (see Figure 11).

Design with DMA data movers is using two AXI Direct Memory Access HW IPs (see Figure 5 and Figure 6).

Design with SG data movers is using two AXI Direct Memory Access HW IPs configured for SG DMA data transfers. Data movers are connected to S_AXI_HP0 and S_AXI_HP1 high performance ports of the ZYNQ processor. Design is using two interrupts (see Figure 7 and Figure 8).

Design with SG-malloc data movers is using one AXI Direct Memory Access HW IPs configured for SG DMA data transfers. It is connected to the S_AXI_ACP advanced cache coherent port of the ZYNQ processor. Design is using two interrupts (see Figure 9 and Figure 10).

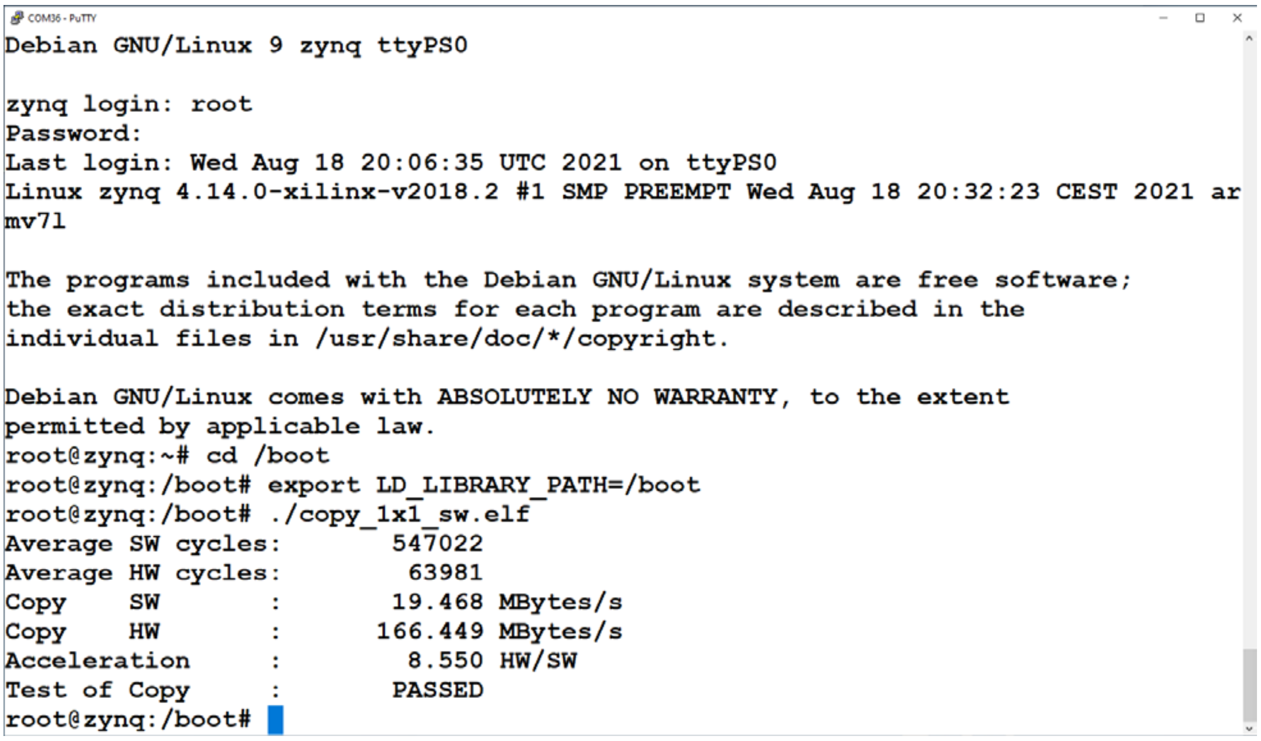
Design with SG and SG-malloc data movers require nearly all PL resources of the small TE0723M-07S device. See Figure 7, Figure 9 and Figure 11.

Details of all four designs can be analysed in the block diagrams (in pdf vector format). These diagrams are included in the evaluation package accompanying this application note.

Designs can be recompiled in SDSoC 2018.2 and Vivado 2018.2 can be used to see all details and configuration of HW IPs.

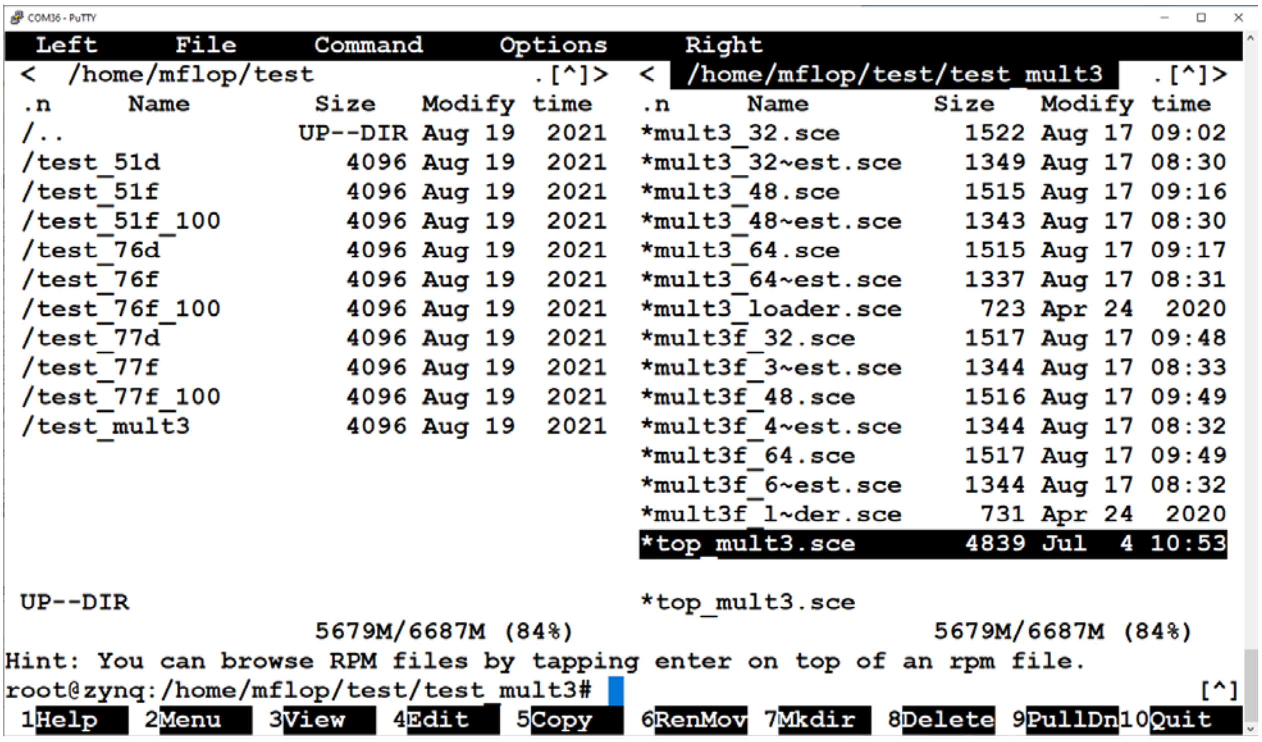
Data mover performance evaluation

Debian console listing from SW application `copy_1x1_sw.elf` on system with ZC data movers is presented in Figure 12.


```
Debian GNU/Linux 9 zynq ttyPS0

zynq login: root
Password:
Last login: Wed Aug 18 20:06:35 UTC 2021 on ttyPS0
Linux zynq 4.14.0-xilinx-v2018.2 #1 SMP PREEMPT Wed Aug 18 20:32:23 CEST 2021 ar
mv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@zynq:~# cd /boot
root@zynq:/boot# export LD_LIBRARY_PATH=/boot
root@zynq:/boot# ./copy_1x1_sw.elf
Average SW cycles:       547022
Average HW cycles:        63981
Copy     SW        :      19.468 MBytes/s
Copy     HW        :     166.449 MBytes/s
Acceleration       :       8.550 HW/SW
Test of Copy       :      PASSED
root@zynq:/boot#
```
Figure 12: Debian terminal, TE0723M-07S device, test of ZC data movers.

DSP benchmarks

Double and Single precision DSP benchmarks are defined by scilab-cli scripts. Figure 13 presents the related Debian OS directory structure in midnight commander tool.


```
  Left     File     Command     Options     Right
 <  /home/mflop/test               .[^]> <  /home/mflop/test/test_mult3    .[^]>
 .n      Name         Size  Modify time   .n      Name          Size  Modify time
 /..                UP--DIR Aug 19  2021  *mult3_32.sce         1522 Aug 17 09:02
 /test_51d             4096 Aug 19  2021  *mult3_32~est.sce     1349 Aug 17 08:30
 /test_51f             4096 Aug 19  2021  *mult3_48.sce         1515 Aug 17 09:16
 /test_51f_100         4096 Aug 19  2021  *mult3_48~est.sce     1343 Aug 17 08:30
 /test_76d             4096 Aug 19  2021  *mult3_64.sce         1515 Aug 17 09:17
 /test_76f             4096 Aug 19  2021  *mult3_64~est.sce     1337 Aug 17 08:31
 /test_76f_100         4096 Aug 19  2021  *mult3_loader.sce      723 Apr 24  2020
 /test_77d             4096 Aug 19  2021  *mult3f_32.sce        1517 Aug 17 09:48
 /test_77f             4096 Aug 19  2021  *mult3f_3~est.sce     1344 Aug 17 08:33
 /test_77f_100         4096 Aug 19  2021  *mult3f_48.sce        1516 Aug 17 09:49
 /test_mult3           4096 Aug 19  2021  *mult3f_4~est.sce     1344 Aug 17 08:32
                                          *mult3f_64.sce        1517 Aug 17 09:49
                                          *mult3f_6~est.sce     1344 Aug 17 08:32
                                          *mult3f_1~der.sce      731 Apr 24  2020
                                          *top_mult3.sce        4839 Jul  4 10:53

 UP--DIR                                  *top_mult3.sce
                     5679M/6687M (84%)                       5679M/6687M (84%)
Hint: You can browse RPM files by tapping enter on top of an rpm file.
root@zynq:/home/mflop/test/test_mult3#                                       [^]
 1Help   2Menu   3View   4Edit   5Copy   6RenMov 7Mkdir  8Delete 9PullDn10Quit
```
Figure 13: Floating point benchmark scilab-cli scripts. Debian OS tool: mc -s

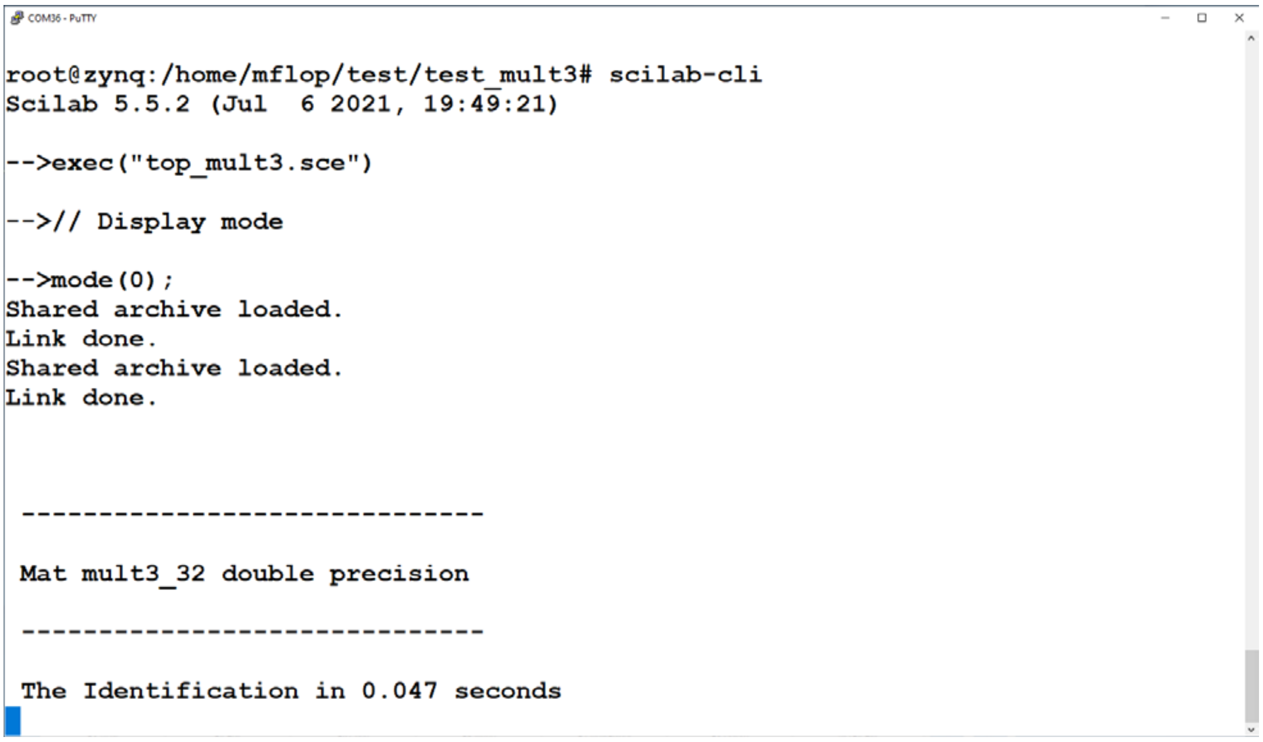

```
root@zynq:/home/mflop/test/test_mult3# scilab-cli
Scilab 5.5.2 (Jul  6 2021, 19:49:21)

-->exec("top_mult3.sce")

-->// Display mode

-->mode(0);
Shared archive loaded.
Link done.
Shared archive loaded.
Link done.



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

 Mat mult3_32 double precision

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

 The Identification in 0.047 seconds
```

Figure 14: scilab-cli script top_mult3.sce is controlled from STM32H753 terminal joystick

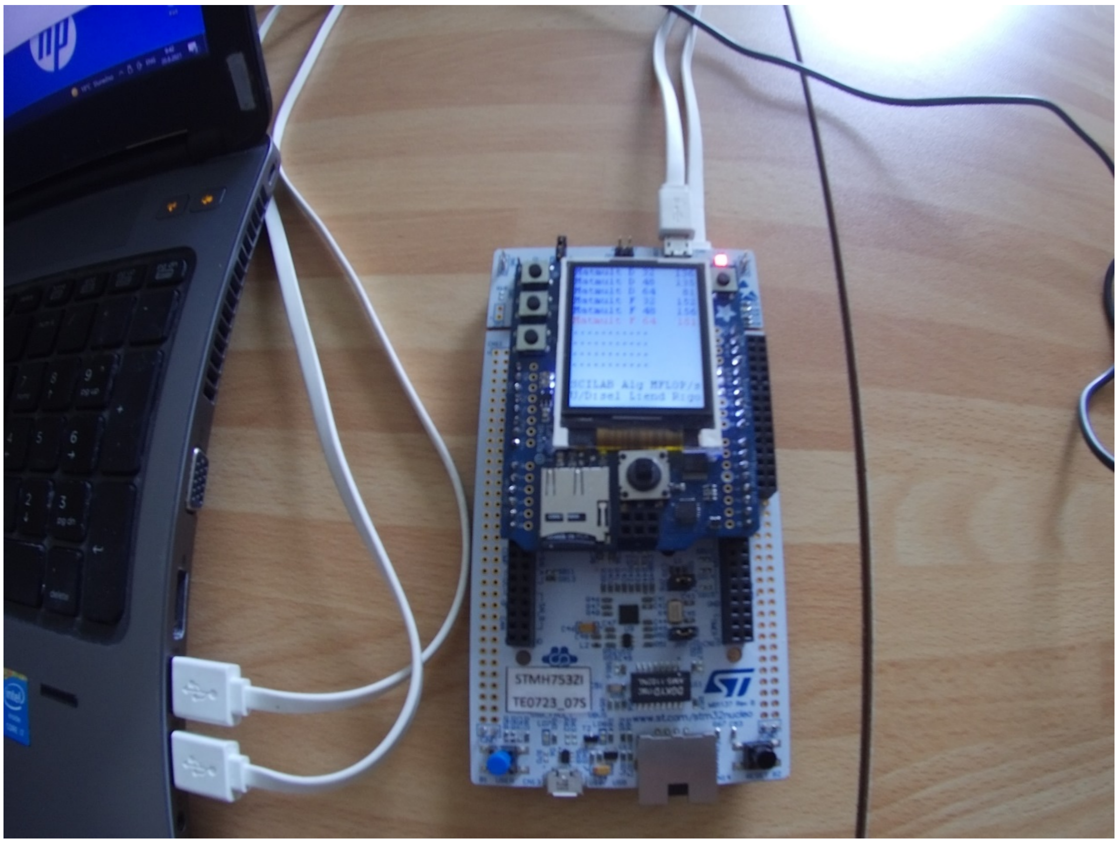


Figure 15: STM32H753 terminal screen defined by scilab-cli script top_mult3.sce

Measured matrix multiplication performance of the A9 processor 650 MHz (see Figure 15):

Double precision [32x32]	136 MFLOP/s
Double precision [48x48]	136 MFLOP/s
Double precision [64x64]	112 MFLOP/s
Single precision [32x32]	152 MFLOP/s
Single precision [48x48]	156 MFLOP/s
Single precision [64x64]	151 MFLOP/s

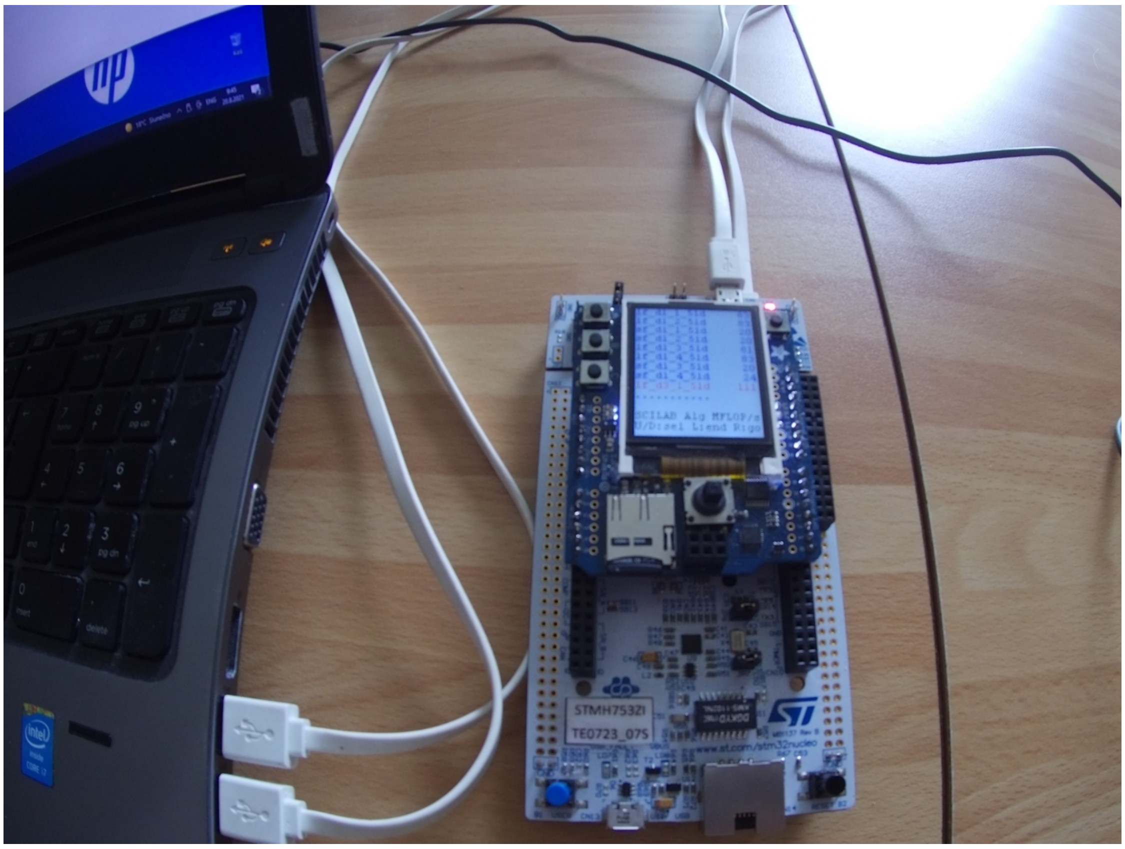


Figure 16: STM32H753 terminal screen defined by scilab-cli script top_md51d.sce

The application note and evaluation package [9] describes in detail the DSP benchmark algorithms and benchmarks generated by the terminal in scilab-cli [18] scripts. It presents benchmark results and provides SW projects for the SW4STM32 System Workbench for STM32 [15] and for the STM32CubeH7 Firmware Package V1.5.0 [16].

Benchmark results for single core A9 MPU on TE0723-03-07S-1C are listed in Figure 17 and Figure 18. Performance results for other CPUs and MCUs are taken from [9].

The STM32H753Z terminal with TE0723-03-07S-1C shield supports creation and execution of these DSP and matrix multiplication benchmarks in scilab-cli interpret. See Figure 16.

We compare adaptive RLS QRD Inverse-update identification benchmark [9] for A53 MPU, single core A9 MPU on TE0723-03-07S-1C shield and STM32F767ZI, STM32H7AZI-Q, STMH743ZI, STM32H753ZI and STM32H723ZG MCUs. See Figure 17 and Figure 18.

top_51d.sce Adapt sys. & C headers generated by scripts:	Zynq Us+ A53 1200 MHz Double precision MFLOP/s	**TE0723-03-07S 650 MHz Double precision MFLOP/s**	F767 CM7 216 MHz Double precision MFLOP/s	H7A3 CM7 280 MHz Double precision MFLOP/s	H743/53 CM7 400 MHz Double precision MFLOP/s	H723 CM7 520 MHz Double precision MFLOP/s
lf_d1_1_51d	159	**83**	19.5	25.2	36,4	46,5
lf_d1_2_51d	167	**86**	20.3	26.3	37.8	48.8
sf_d1_1_51d	48	**20**	9.6	12.0	18.0	20,6
sf_d1_2_51d	48	**20**	9.6	13.1	18.0	24,0
lf_d1_3_51d	159	**81**	19.5	36.4	36.4	46,5
lf_d1_4_51d	159	**86**	20.3	37.8	37.8	48,8
sf_d1_3_51d	48	**20**	9.6	12.0	18.0	20,6
sf_d1_4_51d	48	**20**	9.6	13.1	18.0	24,0
lf_d3_1_51d	211	**109**	22.6	28.9	39.8	55,3

Figure 17: SP MFLOP/s: A53, A9 and F767, H7A3, H743, H753, H723 MCUs.

top_51f.sce Adapt sys. & C headers generated by scripts:	Zynq Us+ A53 1200 MHz Single precision MFLOP/s	**TE0723-03-07S 650 MHz Single precision MFLOP/s**	F767 CM7 216 MHz Single precision MFLOP/s	H7A3 CM7 280 MHz Single precision MFLOP/s	H743/53 CM7 400 MHz Single precision MFLOP/s	H723 CM7 520 MHz Single precision MFLOP/s
lf_d1_1_51f	189	**91**	36.9	47.3	68.7	88,9
lf_d1_2_51f	189	**97**	38.3	49.6	70.3	91,6
sf_d1_1_51f	48	**20**	14.4	18.0	28.8	28,8
sf_d1_2_51f	48	**24**	14.4	18.0	28.8	36,0
lf_d1_3_51f	189	**94**	36.9	47.3	68.7	88,9
lf_d1_4_51f	177	**97**	38.3	49.6	70.3	91,6
sf_d1_3_51f	48	**24**	14.4	18.0	28.8	28,8
sf_d1_4_51f	48	**20**	14.4	18.0	28.8	36,0
lf_d3_1_51f	277	**135**	47.7	60.9	86.1	114,7

Figure 18: DP MFLOP/s: A53, A9 and F767, H7A3, H743, H753, H723 MCUs.

The evaluation package accompanying this application note provides scilab-cli scripts for these benchmarks (see [9] for details):

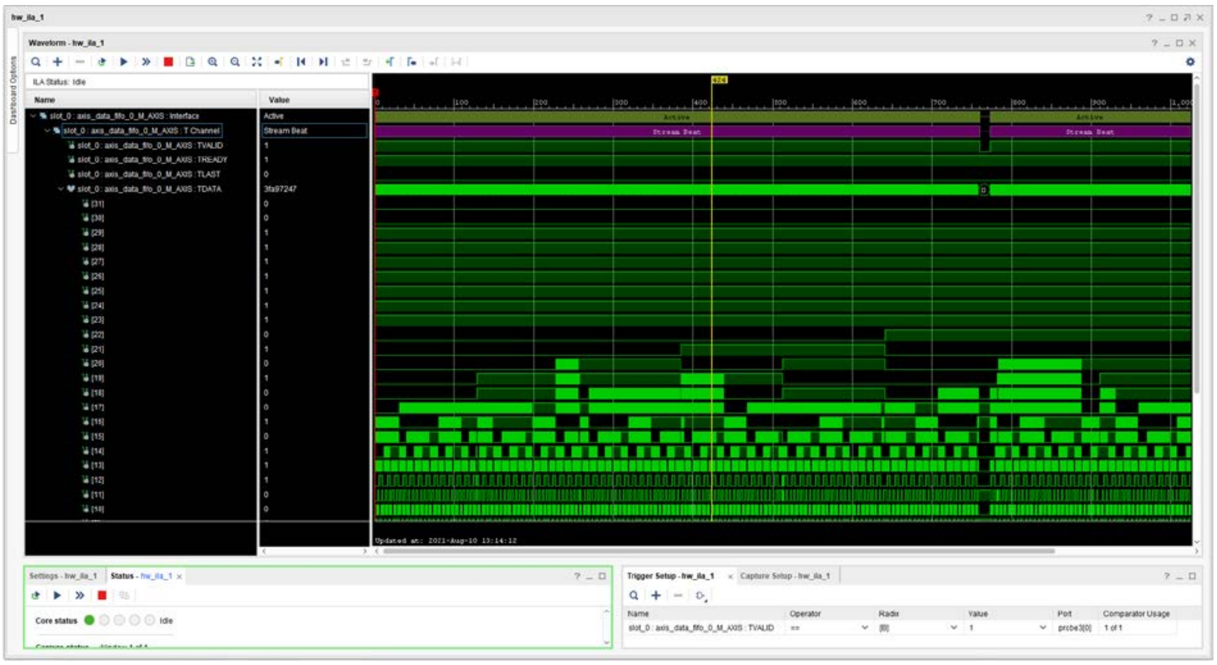
- top_51d.sce Double precision RLS QR Inverse update, no square root.
- top_51f.sce Single precision RLS QR Inverse update, no square root.
- top_51f_100.sce Single precision RLS QR Inverse update, no square root, for M4 MCU.
- top_76d.sce Double precision RLS QR Direct update, no square root.
- top_76f.sce Single precision RLS QR Direct update, no square root.
- top_76f_100.sce Single precision RLS QR Direct update, no square root, for M4 MCU.
- top_77d.sce Double precision RLS QR Direct update, square root.
- top_77f.sce Single precision RLS QR Direct update, square root.
- top_77f_100.sce Single precision RLS QR Direct update, square root, for M4 MCU.
- top_mult3.sce Double and Single precision matrix multiplications.

All these benchmarks can be created and executed by the terminal (see Figure 15 and Figure 16).

4 ILA – In-circuit Logic Analyzer

System includes HW IP of the Xilinx Vivado-Lab tool 2018.2 ILA – In-circuit Logic Analyzer. It is connected to the output of the 1024x32 bit FIFO HW IP. See Figure 1. The start of ILA capturing can be triggered by logic combination of input values defined by the user from the connected PC.

ILA monitor displays values of AXI-S stream bus with the 111 MHz clock resolution. It is configured to store 1024 data samples. See Figure 19.



ILA Status: Idle

Name	Value
slot_0 : axis_data_fifo_0_M_AXIS : Interface	Active
slot_0 : axis_data_fifo_0_M_AXIS : T Channel	Stream Beat
slot_0 : axis_data_fifo_0_M_AXIS : TVALID	1
slot_0 : axis_data_fifo_0_M_AXIS : TREADY	1
slot_0 : axis_data_fifo_0_M_AXIS : TLAST	0
slot_0 : axis_data_fifo_0_M_AXIS : TDATA	3fa97247
[31]	0
[30]	0
[29]	1
[28]	1
[27]	1
[26]	1
[25]	1
[24]	1
[23]	1
[22]	0
[21]	1
[20]	0
[19]	1
[18]	0
[17]	0
[16]	1
[15]	0
[14]	1
[13]	1
[12]	1
[11]	0
[10]	0

Updated at: 2021-Aug-10 13:14:12

Core status Idle

Name	Operator	Radix	Value	Port	Comparator Usage
slot_0 : axis_data_fifo_0_M_AXIS : TVALID	==	[B]	1	probe3[0]	1 of 1

Figure 19: AXI-S bus captured by ILA for design with ZC data movers.

Figure 19 presents example of data transfer captured by ILA. It corresponds to ARM host SW `copy_1x1_sw.elf` and HW design with ZC data movers. It performs HW accelerated copy of [64x64] floating point matrix. Figure 19 displays AXI-S control signals and part of 32 bit wide transferred floating point data.

5 References

[1] "ArduZynq" Arduino compatible Module with Xilinx Z-7007S Single-Core 512MB DDR3L
https://shop.trenz-electronic.de/en/TE0723-03-11C64-A-ArduZynq-Arduino-compatible-Module-with-Xilinx-Z-7007S-Single-Core-512MB-DDR3L?c=349

[2] Trenz Electronic Wiki – TE0723 Resources
https://wiki.trenz-electronic.de/display/PD/TE0723+-+ArduZynq

[3] SDSoC - 2018.2 Full Product Installation
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis/archive-sdsoc.html

[4] Software Development Kit Standalone Web Install Client - 2018.2
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis/archive-sdk.html

[5] Vivado Lab Solutions - 2018.2
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

[6] "ArduZynq" Arduino compatible Xilinx Zynq-7010 SoC-Module
https://shop.trenz-electronic.de/en/TE0723-03-41C64-A-ArduZynq-Arduino-compatible-Xilinx-Zynq-7010-SoC-Modul?c=349

https://shop.trenz-electronic.de/en/TE0723-03M-ArduZynq-Arduino-compatible-Xilinx-Zynq-7010-SoC-module?c=349

[7] STM32H753 Terminal with Zynq Ultrascale+ Accelerator
http://sp.utia.cz/index.php?ids=results&id=2018_2_te0820_fp03x8_2x1_ila_te0706_zu3cg

[8] Benchmarks for STM32H7 MCUs
http://sp.utia.cz/index.php?ids=results&id=STM32H7_benchmarks

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

[10] Evaluation version of 8xSIMD FP01x8 accelerator for ArduZynq shield
http://sp.utia.cz/index.php?ids=results&id=te0723_fp01x8

[11] STM32 Nucleo-H743ZI with Adafruit 1.8" TFT Shield V2
http://sp.utia.cz/index.php?ids=results&id=nucleo-LCD-V2

[12] UM1974 User manual STM32 Nucleo-144 boards (MB1137)
https://www.st.com/resource/en/user_manual/dm00244518-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf

[13] UM2408 User manual STM32H7 Nucleo-144 boards (MB1363)
https://www.st.com/resource/en/user_manual/dm00499171-stm32h7-nucleo144-boards-mb1363-stmicroelectronics.pdf

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

[15] 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

[16] 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

[17] 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

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

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

6 APPENDIX - Confidence test

This is basic confidence test of the evaluation package.

Unzip evaluation package to Win 10 directory of your choice. We will use:
`c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\`

Precompiled HW and SW projects are located in directory:
`c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila_release\copy_zc_1x1_sw`

Compressed SD card image with ARM Debian OS is located in directory:
`c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila_release_sdcard\`

INSTALLATION OF TOOLS

- Install Xilinx SDK 2018.2 on Win 10 PC 64 bit [3].
- Install Xilinx Lab Tools 2018.2 on Win 10 PC 64 bit [5].
- Install Win32DiskImager for writing of image to 16 GB SD card, (Class 10).
- Install Putty (for USB based serial console and Ethernet based serial console).
- Unzip ARM Debian OS disk image on Win 10 PC and use the Win32DiskImager to write the disk image (16 GB) from the PC to the 16 GB SD card (Class 10).

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

- Remove SD card from the TE0723-03-07S-1C board.
- Connect the TE0723-03-07S-1C board to PC by the USB serial terminal cable.
- Copy the BOOT.BIN file from

  `c:\home\work\TS82fp01x8_TE0723\xc7z07s_deb_eval_fifo_ila_release\copy_zc_1x1_sw\Release\sd_card\BOOT.BIN`
  to
  `c:\home\work\TS82fp01x8_TE0723\xc7z07s_deb_eval_fifo_ila\zsys\prebuilt\boot_images\m\NA\BOOT.BIN`

- Change directory to
  `c:\home\work\TS82fp01x8_TE0723\xc7z07s_deb_eval_fifo_ila\zsys`

  Execute this script in Win 10 terminal:

  `program_flash_binfile.cmd`

  This script will write content of the **`BOOT.BIN`** file to the ZynqBerry board flash.
- Power-off the TE0723-03-07S-1C board by removing the USB cable from the PC.

HW SETUP

- Insert the SD card with Debian OS disk image to the TE0723-03-07S-1C board.
- Connect PC and TE0723-03-07S-1C board to Ethernet.

- Connect USB serial terminal cable to TE0723-03-07S-1C to PC. This will power-on the board.

TEST
- TE0723-03-07S-1C board will start to boot Debian OS. The boot process starts by reading data from the BOOT.BIN present in the internal flash. Only the second stage of the boot process is performed from the SD card.
- In Win 10 PC, open Putty terminal. Set it to:
  (115200 bps, 8 data bits, stop bit 1, parity none, flow control off)
- Use Putty terminal to login as user: **`root`** password: **`root`**
- Change directory to **`/boot`**
- Export path to the shared library. Type in the Putty Debian OS terminal:

  ```
  export LD_LIBRARY_PATH=/boot
  ```
- Start application code by typing in the Putty Debian OS terminal:

  ```
  ./copy_zc_1x1_sw.elf
  ```

RESULT
- The application will copy single precision floating point by:
  - SW on host ARM A9 processor
  - HW accelerated on host ARM A9 processor by zero copy (ZC) data movers.
- Results of ARM SW and HW accelerated copy are compared to be identical and Mbyte/s performance is measured, computed and displayed. See Figure 12.

Compilation and debug of projects from source code

The evaluation package includes SW projects for Xilinx SDK 2018.2 tool running on Win 10.

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

```
c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fif
o_ila_release\copy_zc_1x1_sw\
```

Projects in this directory link to this shared library:
```
c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fif
o_ila_release\copy_zc_1x1_sw\Release\sd_card\libcopy_zc_1x1_hw.so
```
or
```
c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fif
o_ila_release\copy_zc_1x1_sw\Debug\sd_card\libcopy_zc_1x1_hw.so
```

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

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

Recompile ARM host SW application directly on the TE0723-03-07S-1C board

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

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

```
c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fif
o_ila_release\copy_zc_1x1_sw\
```

to the ARM host Debian OS directory:

```
/home/copy_zc_1x1_sw/
```

Change the directory in ARM Debian OS to:

```
cd /home/copy_zc_1x1_sw/Debug/
```

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

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

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

```
make clean
make
```

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

```
./copy_zc_1x1_sw.elf
```

You are done. The compiled application is running on the TE0723 board. See Figure 12.
To close correctly the Debian OS, type in the Debian OS terminal:

```
halt
```

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

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

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

The Debian OS image includes **`scilab-cli`** . It is command line version of the scilab SW interpret [18]. To use it, take these steps. In Debian OS terminal, change directory to

cd /**`home/mflop/cc/cc_mult3d`**

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

```
scilab-cli
```

In **`scilab-cli`**, execute script **`mult3_cc.sce`** by command

```
exec(”mult3_cc.sce.sce”)
```

This script will compile C MEX function **`mult3.c`** to shared library **`libmex_mult3.so`** in the same directory

Quit **`scilab-cli`** by typing

```
quit
```

Copy created shared library **`libmex_mult3.so`**

`/home/mflop/cc/cc_mult3d/libmex_mult3.so`

to

`/home/mflop/test/test_mult3/libmex_mult3.so`

In Debian terminal, change directory to

`/home/test/test_mult3`

Start **`scilab-cli`** by typing

```
scilab-cli
```

In scilab-cli execute script **`multf_32_test.sce`** by command

```
exec(”mult3_32_test.sce”)
```

scilab-cli will call and execute **`mult3()`** C MEX function present in shared library **`libmex_mult3.so`** , measures execution time and also generates data header files in **`./mult3_32/`** subdirectory. Created header files contain single precision floating point reference data used in STM32H7 projects for testing of C implementations on MCU devices.

Quit **`scilab-cli`** by typing.
`quit`

MEX C code is compiled with –O2. Use **`Make_me.sh`** script to compile with –O3.

7 APPENDIX – Design guidelines

Guide for compilation of HW for the xc7z07s device

1. Unpack the evaluation package to Win 10 directory. In this guide, we unpack to this directory:
   ```
   c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila
   \
   ```
   Change directory to:
   ```
   c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila
   \zsys\
   ```
2. On Win 10, open dos terminal window, change directory to the folder
   ```
   c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila
   \zsys\
   ```
3. To overcome limitations of Win 10 related to the need of short directory paths, use the script *_use_virtual_drive.cmd* to create a virtual short path to your directory drive *X:\zusys* Type command:
   ```
   _use_virtual_drive.cmd
   ```
   Select `X` as name of the virtual drive and select `0` to create the virtual drive.
   Go to the created virtual short-path directory by typing in the win 10 terminal:
   ```
   X:
   cd zsys
   ```
4. 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 Win 10. Line 38:
   ```
   @set XILDIR=C:/Xilinx
   ```
   Select proper Xilinx device:
   ```
   @set PARTNUMBER=4
   ```
   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
   ```
5. Start the Xilinx Vivado 2018.2 and create the design by executing of script:
   ```
   X:\zsys\vivado_create_project_guimode.cmd
   ```
6. Optional:
   You can use *Vivado* automation and to the created HW design Xilinx *In Circuit Logic Analyzer* (ILA) monitor to enable capturing of selected accelerator outputs of your choice.
7. In *Vivado* console, execute command:
   ```
   TE::hw_build_design -export_prebuilt
   ```
   After the Vivado compilation, new hardware description file `zsys.hdf` is generated in folder:
   ```
   X:\zsys\prebuilt\hardware\m\zsys.hdf
   ```

Guide for configuration and compilation of the PetaLinux

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 TE0723-03-07S-1C board is described now. The configuration has to be performed in the Ubuntu 16.04 LTS OS. The evaluation package is using the Ubuntu 16.04 LTS in the *VMware Workstation Player* in Win 10. The Petalinux 2018.2 distribution can be downloaded to the Ubuntu 16.04 LTS from

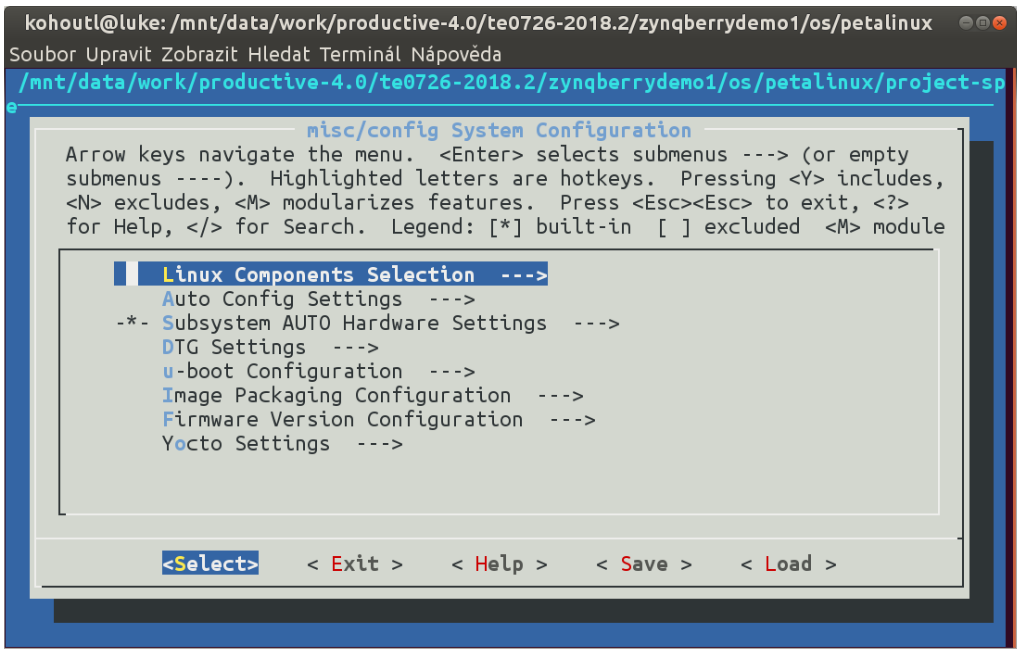
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2018-2.html

and installed to the default Ubuntu directory:

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

The standard PetaLinux 2018.2 distribution requires few modifications.

1. Copy content of these Win 10 directories:
   ```
   X:\zsys\prebuilt
   X:\zsys\os
   ```
   to Ubuntu directories:
   ```
   /home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila/zsys/prebuilt/
   /home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila/zsys/os/
   ```
2. In Ubuntu, open terminal window and set path to the PetaLinux 2018.2:
   ```
   source /opt/petalinux/petalinux-v2018.2-final/settings.sh
   ```
3. Change directory to the directory copied from the evaluation package with pre-defined configuration:
   ```
   /home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila/zsys/os/petalinux/
   ```
   It contains a predefined configuration according to Zynq TE0723-03-07S-1C board requirements.
4. The zsys.hdf file created in Win 10 in Vivado 2018.2 tool is present in the Ubuntu folder:
   ```
   /home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila/zsys/prebuilt/prebuilt/hardware/7s/
   ```
5. Use the `zusys.hdf` file as input for the PetaLinux configuration by (on single line)
   ```
   petalinux-config --get-hw-description=/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila/zsys/prebuilt/prebuilt/hardware/7s/
   ```


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

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

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

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

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

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

8. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

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

```
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila
/zsys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo_ila
/zsys/os/petalinux/images/linux/bl31.elf
```

Guide for configuration and compilation of the Debian OS

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

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

```
cd
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/
```

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

    When some of them are missing, install them by:

```
sudo apt install Package
```

Table 1: tools with a corresponding package name.

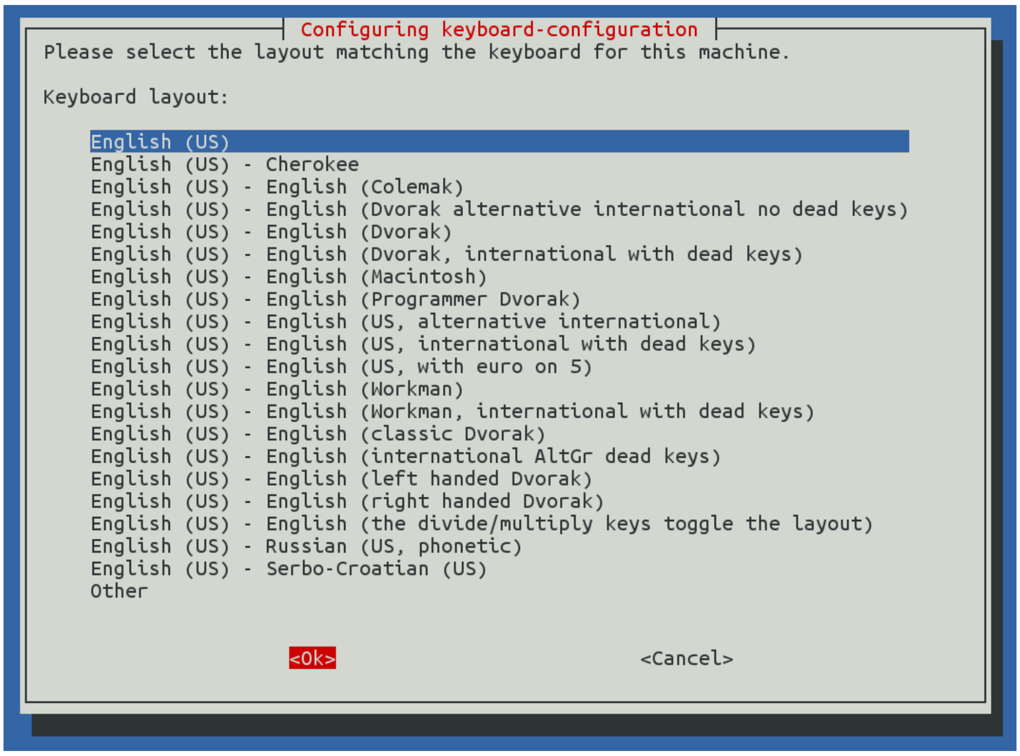
Tool	***Package***
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-ARM-static	qemu-user-static

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

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

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

During the creation procedure, you will be asked to set language. Choose *English (US)*. The resultant image file will be called TE0723-debian.img its size will be 7 GB.



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

Keyboard layout:

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

                <Ok>                          <Cancel>
```

13. Compress the created image to file TE0723-debian.zip:

```
zip TE0723-debian TE0723-debian.img
```

14. Copy compressed image file from Ubuntu

```
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/TE0723-debian.zip
```

to Win 10 file:

```
X:\zsys\prebuilt\os\petalinux\default\TE0723-debian.zip
```

15. Copy these files from Ubuntu

```
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/images/linux/bl31.elf
```

to Win 10 files:

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

16. In Ubuntu, clean Petalinux project files

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

17. In Ubuntu, delete files

```
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/TE0723-debian.zip
/home/devel/work/TS82fp01x8_TE0723_TERMINAL_xc7z07s/xc7z07s_deb_eval_fifo
_ila/zsys/os/petalinux/TE0723-debian.img
```

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

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

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

- Petalinux kernel image `image.ub`
- Compressed Debian image `TE0723-debian.zip`
- U-boot program `u-boot.elf`

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

Guide for creation of SDSoC platform for the TE0723-03-07S-1C

20. In the open Vivado 2018.2 console, create and compile the initial *BOOT.bin* file and the initial SW modules by execution of the command:
    ```
    TE::sw_run_hsi
    ```
    The resulting *BOOT.bin* file will be located in the folder
    ```
    X:\zsys\prebuilt\boot_images\m\u-boot\BOOT.bin
    ```
21. These files are created:
    ```
    X:\zsys\prebuilt\software\m\hello_TE0723.elf
    X:\zsys\prebuilt\software\m\zynq_fsbl.elf
    X:\zsys\prebuilt\software\m\zynq_fsbl_flash.elf
    ```
    File `zynq_fsbl.elf` is correct first stage board loader (FSBL) file, while the `zynq_fsbl_flash.elf` is special FSBL file used only for programming of the on board flash.
22. Move `zynq_fsbl_flash.elf` file to some different temporary location before next step.
23. In Vivado 2018.2 console, create the SDSoC platform by execution of the command:
    ```
    TE::ADV::beta_util_sdsoc_project
    ```
    The SDSoC 2018.2 platform is generated in the directory
    `X:\SDSoC_PFM\TE0723\03m\zsys`
    and it is also packed into the ZIP file in directory
    X:\SDSoC_PFM\TE0723\
24. Return `zynq_fsbl_flash.elf` file back from the temporary location to
    X:\zsys\prebuilt\software\m\zynq_fsbl_flash.elf
    It will be used later on by the TE0723-03-07S-1C board flash programming script
    `program_flash_binfile.cmd`

25. On Win 10, 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)
26. Change directory to

    ```
    c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_fifo_ila
    \SDSoC_PFM\TE0723-03\07s-1c\
    ```
27. In Win 10, 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:\07s-1c`

    ```
    _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 07s-1c
    ```
28. Open SDSoC 2018.2 project in directory

    ```
    X:\07s-1c
    ```
29. In SDSoC import four HW data mover design projects:

    ```
    copy_dma_1x1_hw
    copy_sg_1x1_hw
    copy_sg_malloc_1x1_hw
    copy_zc_1x1_hw
    ```

    from the directory with source-code of SDSoC projects

    ```
    c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval_ila\SDSo
    C_PFM_src\TE0723-03\07s-1c\
    ```

    In SDSoC select the custom SDSoC platform

    ```
    X:\07s-1c\zsys
    ```
30. Change imported project from Debug to the Release compilation target
31. Compile project by the Xilinx SDSoC 2018.2 compiler
32. Result of compilation are the SD cards with the `BOOT.BIN` file and the shared object library file `libcopy_zc_1x1_hw.so` in:

    ```
    X:\07s-1c\copy_dma_1x1_hw\Release\sd_card\
    X:\07s-1c\copy_sg_1x1_hw\Release\sd_card\
    X:\07s-1c\copy_sg_malloc_1x1_hw\Release\sd_card\
    X:\07s-1c\copy_zc_1x1_hw\Release\sd_card\
    ```
33. For all four projects, copy content of these directories like:

    ```
    X:\07s-1c\copy_zc_1x1_hw\Release\sd_card\
    ```

    to

    ```
    c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval__ila_rel
    ease\copy_zc_1x1_sw\Release\sd_card\
    ```

    and also like:

    ```
    c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval__ila_rel
    ease\copy_zc_1x1_sw\Debug\sd_card\
    ```
34. Optional:

    For all four data mover projects, copy ILA nets definition files `debug_nets.ltx` and `zsys_wrapper.ltx` from the directory like:

`X:\07s-1c\copy_zc_1x1_hw\Release\_sds\p0\vivado\prj\prj.runs\impl_1\`

to

`c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval__ila_release\copy_zc_1x1_sw\Release\sd_card\`

and also to:

`c:\home\work\TS82fp01x8_TE0723_TERMINAL_xc7z07s\xc7z07s_deb_eval__ila_release\copy_zc_1x1_sw\Debug\sd_card\`

35. In SDSoC, clean all four SDSoC projects to save disk space.
36. Close Xilinx SDSoC 2018.2 tool.
37. The created `BOOT.BIN` files will be used for programming of the TE0723-03-07S-1C board flash. The shared object library files like the `libcopy_zc_1x1_hw.so` will be linked to applications compiled for ARM in SDK and also used in the runtime on ARM.

    This is described in the first section of the chapter 10 APPENDIX - Confidence test.

Disclaimer