CON-FMC Example of Xilinx FFT

Version 0 Revision 1

May 5, 2019 (April 10, 2019)

Future Design Systems, Inc.
www.future-ds.com / contact@future-ds.com

Abstract

This document addresses an example of FFT, in which CON-FMC with ZedBoard is used.

Table of contents

Copyright © 2019 Future Design Systems, Inc. 1
Abstract 1
Table of contents 1
1 Introduction 2
2 Getting started 2
    2.1 Prerequisites 2
    2.2 Setup 2
    2.3 Quick start 3
3 Design in detail 3
    3.1 Hardware structure 4
        3.1.1 Address map 4
        3.1.2 Clock and reset 4
    3.2 Software 5
        3.2.1 Host-program driven BFM architecture 5
        3.2.2 C API 6
4 Directory structure and simulation 7
    4.1 Directory structure 7
    4.2 RTL simulation 8
    4.3 FPGA implementation 11
    4.4 FPGA program 12
5 Running program 12
    5.1 Memory test 12
    5.2 FFT test 13
Wish list 15
References 15
Revision history 16

1 Introduction

This example addresses an example as shown in Figure 1, where the design consists of a Xilinx FFT that is accessed by the host program through AXI BFM over USB 2.0 or 3.0.

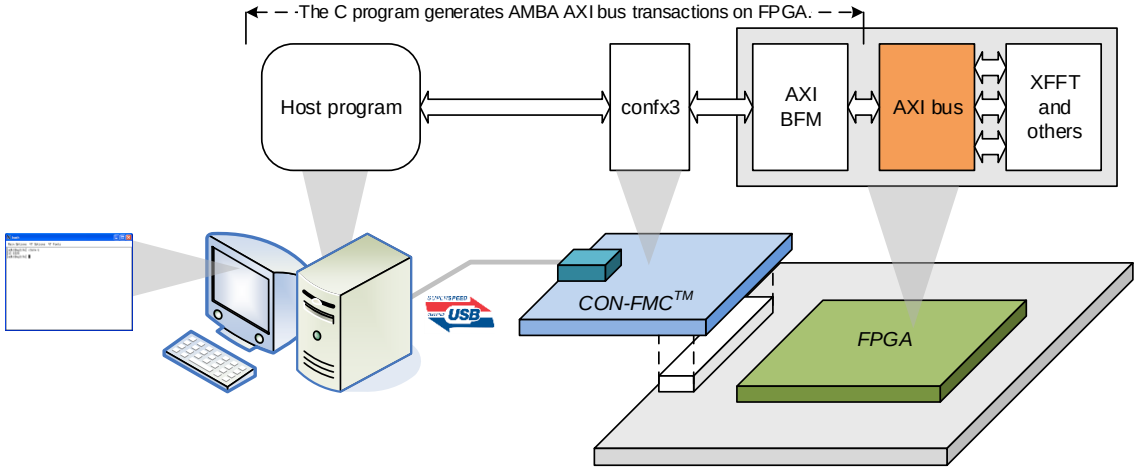


Figure 1: AMBA AXI Memory Platform

2 Getting started

2.1 Prerequisites

- Mandatory
  - ✧ CON-FMC board
  - ✧ CON-FMC SW package (Version 2019.02 or later)
  - ✧ FPGA board: Avnet ZedBoard
  - ✧ GNUPLOT and Python
- Optional
  - ✧ Simulator: industry standard Verilog simmulator, e.g., ModelSim
  - ✧ FPGA development package: Xilinx Vivado 2018.3 or later

2.2 Setup

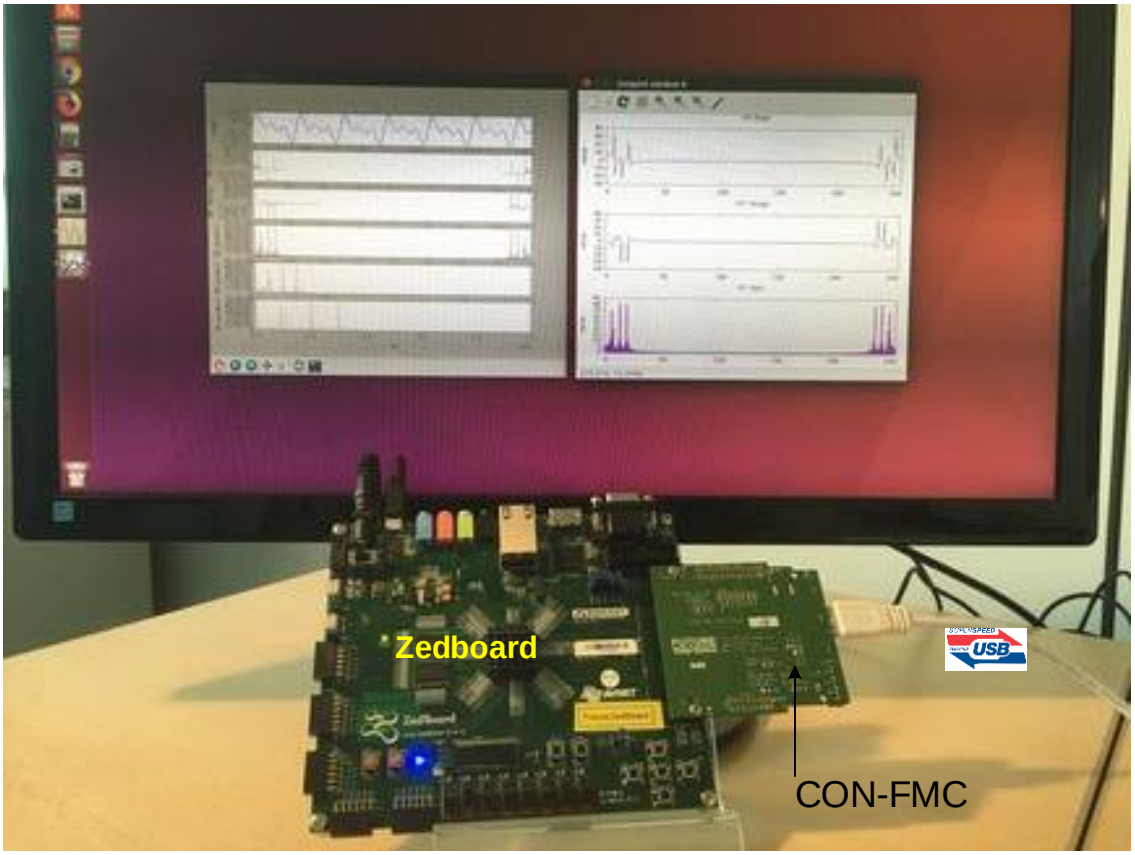


Figure 2: Setup under operation (one of two CON-FMC is used)

2.3 Quick start

1. Make sure the FPGA board is turned off
2. Connect CON-FMC to the FPGA board on FMC-LPC connector
3. Connect CON-FMC to the host computer through 3.0
4. Connect JTAG-USB to the host computer
5. Turn on FPGA board[1]
6. Down load bit-stream under the $project directory using Vivado HW manager ($project/hw/pnr/vivado.zedboard/fpga.bit) or
   - Go to '$project/hw/pnr/vivado.zedboard/download'
   - Then run 'make'
7. Go to '$project/sw.native/test_mem' directory
8. Run 'make cleanup; make; make run'
9. Then, host-program writes and read the memory in the FPGA.
10. Run 'make plot' to check result

3 Design in detail

[1] After turn on, CON-FMC should be detected by '$ lsusb' command. It should list '0x04B4:0x00F3' device.

3.1 Hardware structure

Figure 3 shows internal structure of the design; i.e., FPGA design, where all modules are connected to 'trx_AXI' through AMB AXI bus.

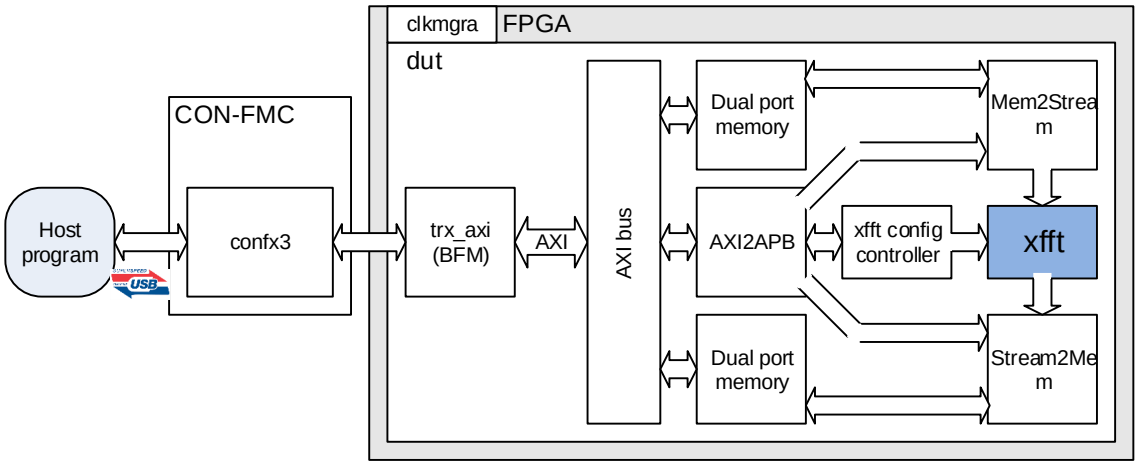


Figure 3: Hardware structure

3.1.1 Address map

AXI BFM generates AMBA AXI transactions on the AXI bus.

- S0: APB : 0x0000_0000
- S1: axi_mem: 0x1000_0000 for raw-data (16Kbyte)
- S2: axi_mem: 0x2000_0000 for fft-data (16Kbyte)

- P0: xfft_config : 0x0000_0000
- P1: m2s : 0x0001_0000
- P2: s2m : 0x0002_0000

3.1.2 Clock and reset

As shown in Figure 4, one clock is fed into the FPGA, which is 100Mhz BOARD_CLK_IN. This clock is adjusted by FPGA internal MMCM and three clocks are provided and frequencies of these clock can be changed by setting corresponding macros[2].

- SYS_CLK: CON-FMC interface clock (keep it 80Mhz)
  - ✧ it is for USB interface and can be 80Mhz or 100Mhz depending on defining macro, 'SL_PCLK_FREQ' in 'syn_define.v'.
- USR_CLK: System bus clock, i.e., ACLK (select the same as SYSCLK if there is no specific needs.)
- it is for user design and can be 80Mhz or 100Mhz depending on defining macro, 'USR_CLK_FREQ' in 'syn_define.v'.

[2] SL_PCLK_FREQ, USR_CLK_FREQ, SERIAL_CLK_FREQ in the fpga_zed.v file.

- SERIAL_CLK: clock for AXI Stream and XFFT (select reasonable such as 125Mhz)

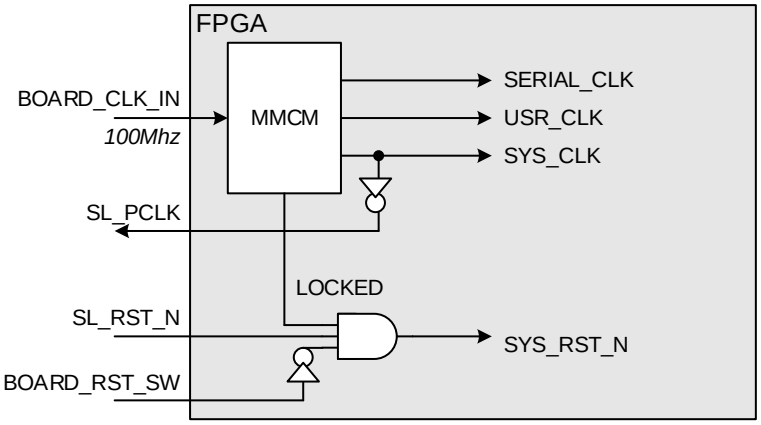


Figure 4: Clock structure

As shown in Figure 4, three reset sources are fed into the FPGA; SL_RST_N (active-low), BOARD_RST_SW (active-high), LOCKED (active-low in terms of reset). These two are active-low. In addition to this MMCM LOCED result is used to build reset.

3.2 Software

3.2.1 Host-program driven BFM architecture

Transactor (i.e., BFM: Bus Functional Model) in the FPGA generates system bus specific bus transaction under the command that comes from the host program through USB channel.

As shown in Figure 5, C program sends bus commands to the BFM through transparent communication channel over USB and then the commands direct BFM to carry out corresponding bus transactions, which can be AXI or AXI depending on BFM and system bus.

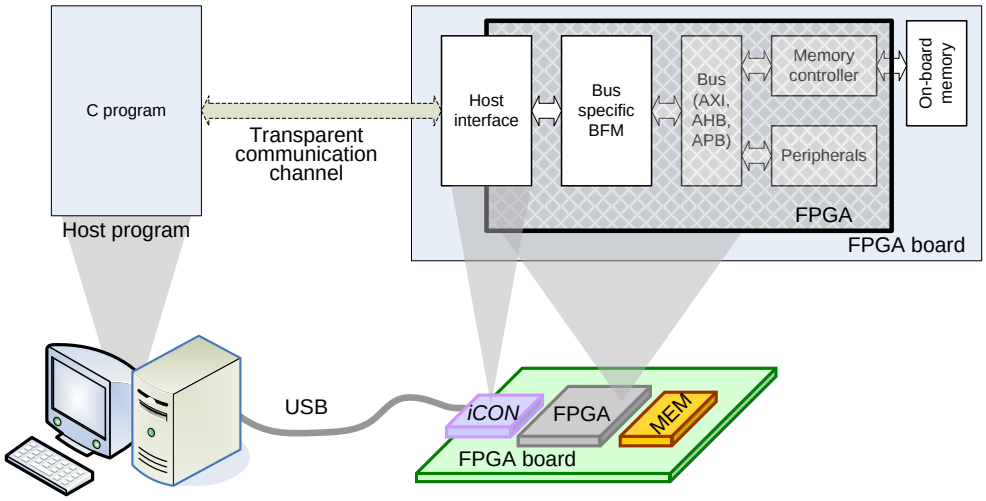


Figure 5: Host-program driven BFM architecture

3.2.2 C API

It should be noted that following C APIs are just macro and you should have proper C header and library. If you are not sure, please contact technical support of Future Design Systems.

- $CONFMC_HOME/include/conapi.h
- trx_AXI/drv/c/trx_axi_api.h

BfmWrite() generates write transaction on the system bus, where the buffer pointed by 'data' carries justified data and each item carries 'size' bytes up to 4.

```
void BfmWrite( unsigned int  addr   // address to write
             , unsigned int *data   // pointer to buffer contains data to write
             , unsigned int  size   // num of bytes for each element in the buffer
             , unsigned int  length); // num of items to write
```

BfmRead() generates read transaction on the system bus, where the buffer pointed by 'data' carries justified data and each item carries 'size' bytes up to 4.

```
void BfmRead ( unsigned int  addr // address to read
             , unsigned int *data // pointer to buffer containing data read
             , unsigned int  size // num of bytes for each item
             , unsigned int  length); // num of items to read
```

Following shows an example,

```
unsigned int addr;
unsigned int dataR, dataW;

addr = 0x80000000;
dataW = 0x12345678;
BfmWrite(addr, dataW, 4, 1); // write 4-byte of data to the address 0x80000000
BfmWrite(addr, dataR, 4, 1); // read 4-byte of data from 0x80000000
```

```
addr = 0x80000002;
dataW = 0x00001234; // two bytes of data in justified fashion
BfmWrite(addr, dataW, 2, 1); // write 2-byte of data to the address 0x80000002
BfmWrite(addr, dataR, 2, 1); // read 2-byte of data from 0x80000002

addr = 0x80000003;
dataW = 0x00000012; // one byte of data in justified fashion
BfmWrite(addr, dataW, 1, 1); // write 1-byte of data to the address 0x80000003
BfmWrite(addr, dataR, 1, 1); // read 1-byte of data from 0x80000003
```

API argument ‘addr’ should be aligned with the size and data in the ‘data’ argument is justified.

4 Directory structure and simulation

4.1 Directory structure

Table 1: Directory structure

Directory					Remarks
fex_0007_xfft					
	doc				document
	hw				Single FMC case
		beh	verilog/gpif2slv.v		Behavioral tester
		bench	verilog/top.v		Top-level for simulation
		design	verilog/fpga.v verilog/dut.v verilog/clkmgra.v		Design for FPGA
		pnr	vivado.zedboard		implementation
		sim	modelsim.vivado		HDL simulation for ModelSim
	iplib				HW IP
		amba_axi			AMBA AXI
		axi_mem2stream			AXI memory-mapped to stream
		axi_stream2mem			AXI stream to memory-map
		axi_to_apb			AXI to APB bridge
		mem_axi_dual			AMBA AXI memory
		xfft			Xilinx FFT
		xfft_config			XFFT configuration controller
	sw.native				Native-mode program to test the design
		test_mem			Memory testing
			src		
		test_xfft			XFFT testing
			src		

4.2 RTL simulation

As shown in Figure 6, the design can be simulated using tasks. It uses ‘gpif2slv’ behavioral model, which generates GPIF2 Interface transactions.

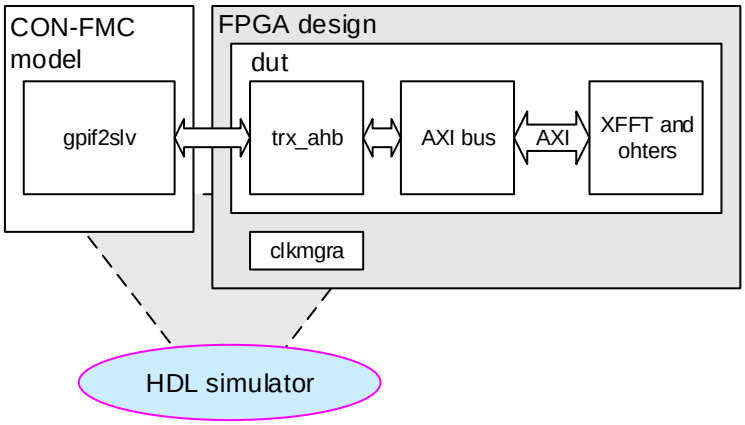


Figure 6: HW/SW co-simulation

Do as follows:

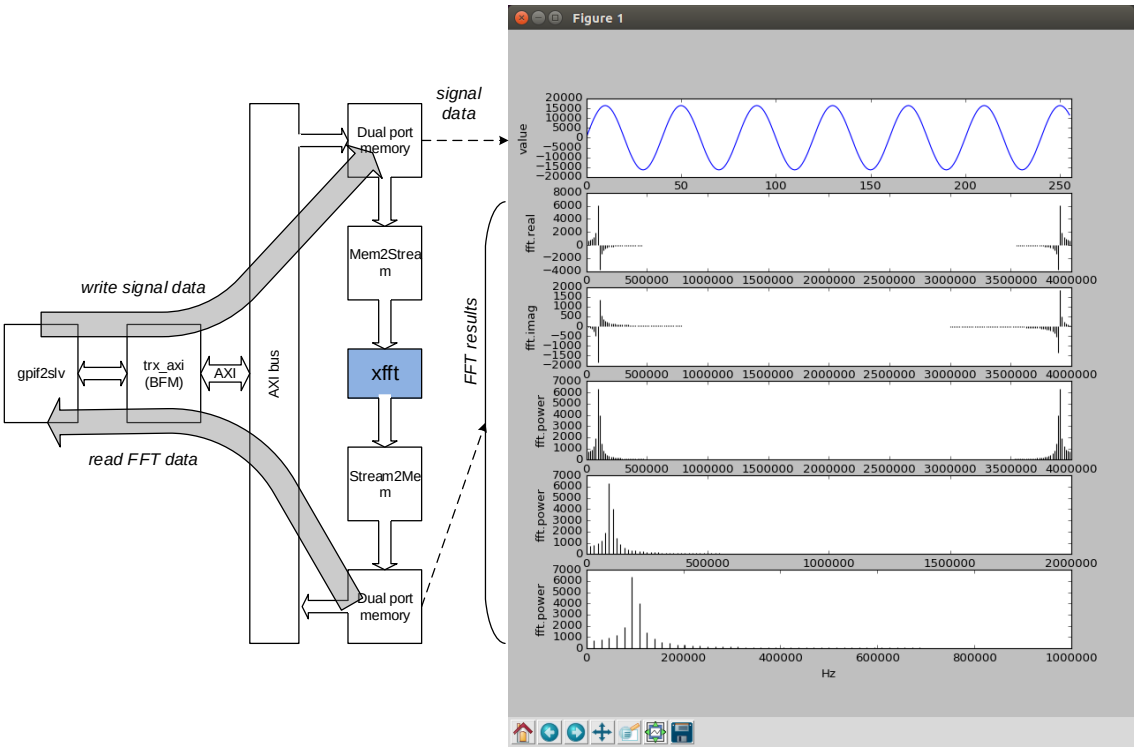
① go to $project/hw /sim/modelsim.vivado
② run ‘make’.

Note that this step does not use Xilinx XFFT instead it uses behavioral model in ‘hw/beh/xfft_16bit256samples_dummy.v’.

Simulation result can be plotted as follows.

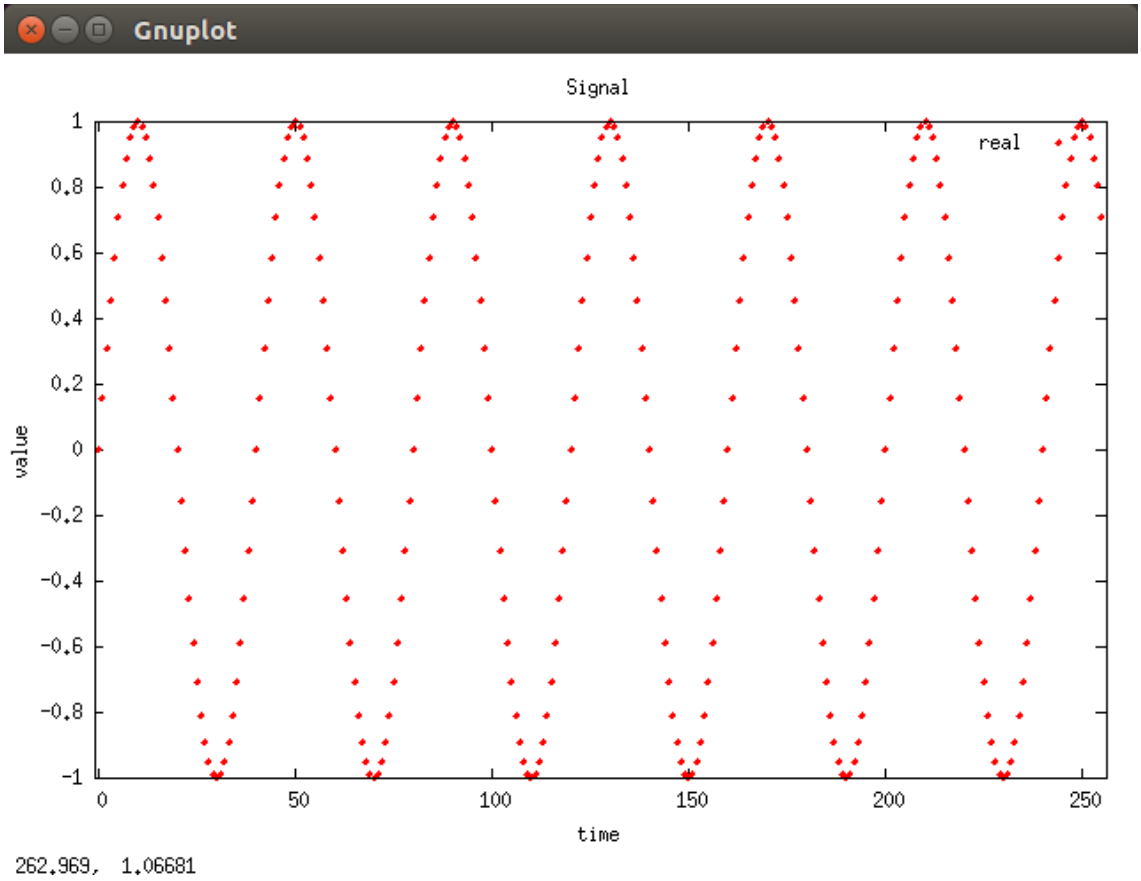
To see whole

```
$ make plot
```



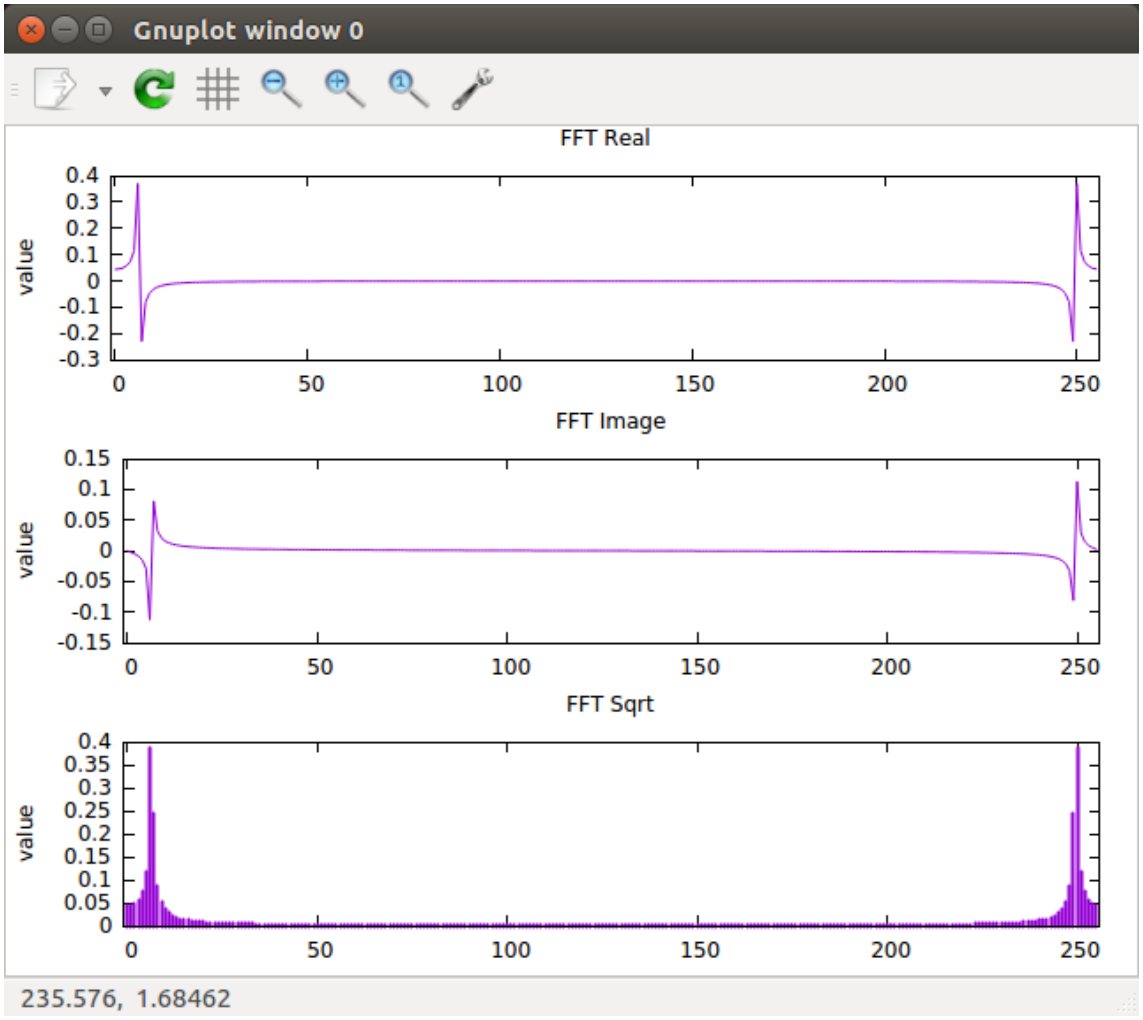
To see signal

```
$ make plot_data
```



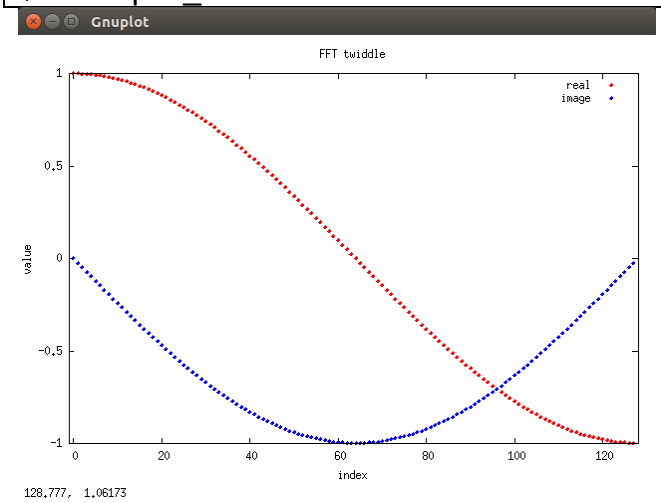
To see fft results

```
$ make plot_fft
```



To see twiddle factors

```
$ make plot_twiddle
```



4.3 FPGA implementation

Look ‘$(project)/hw/pnr/vivado.zedboard’ directory and run ‘make’ to get bit-stream for Zedboard.

4.4 FPGA program

Look ‘$(project)/hw/pnr/vivado.zedboard/download’ directory and run ‘make’ to download bit-stream to the Zedboard. Do not forget to connect JTAG connection.

5 Running program

5.1 Memory test

Look ‘$(project)/sw.native/test_mem’ directory and run ‘make cleanup; make run’ to run the program.

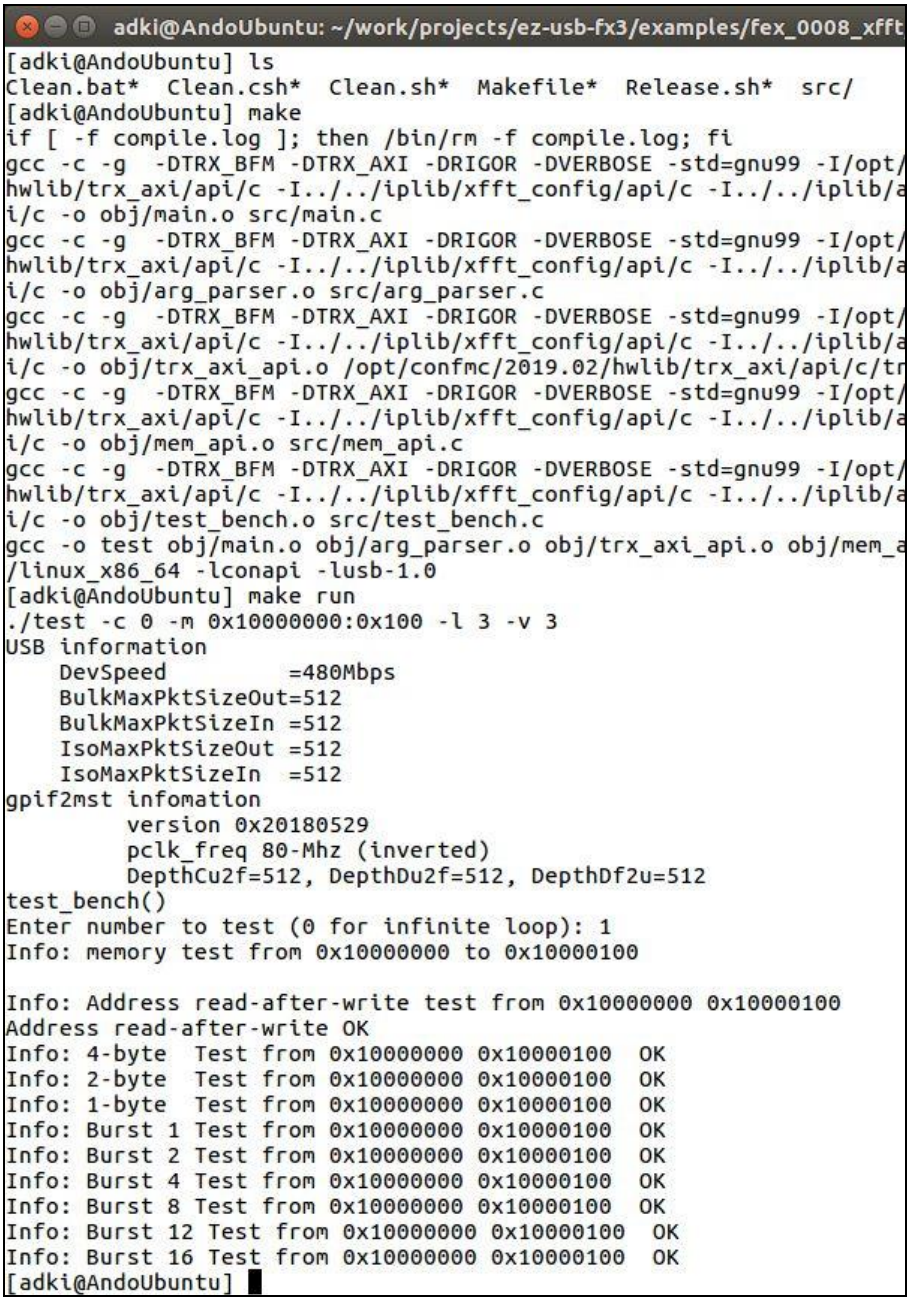


Figure 7: Memory testing case

5.2 FFT test

Look '$(project)/sw.native/test_xfft' directory and run 'make cleanup; make run' to run the program.

Figure 8: FFT testing

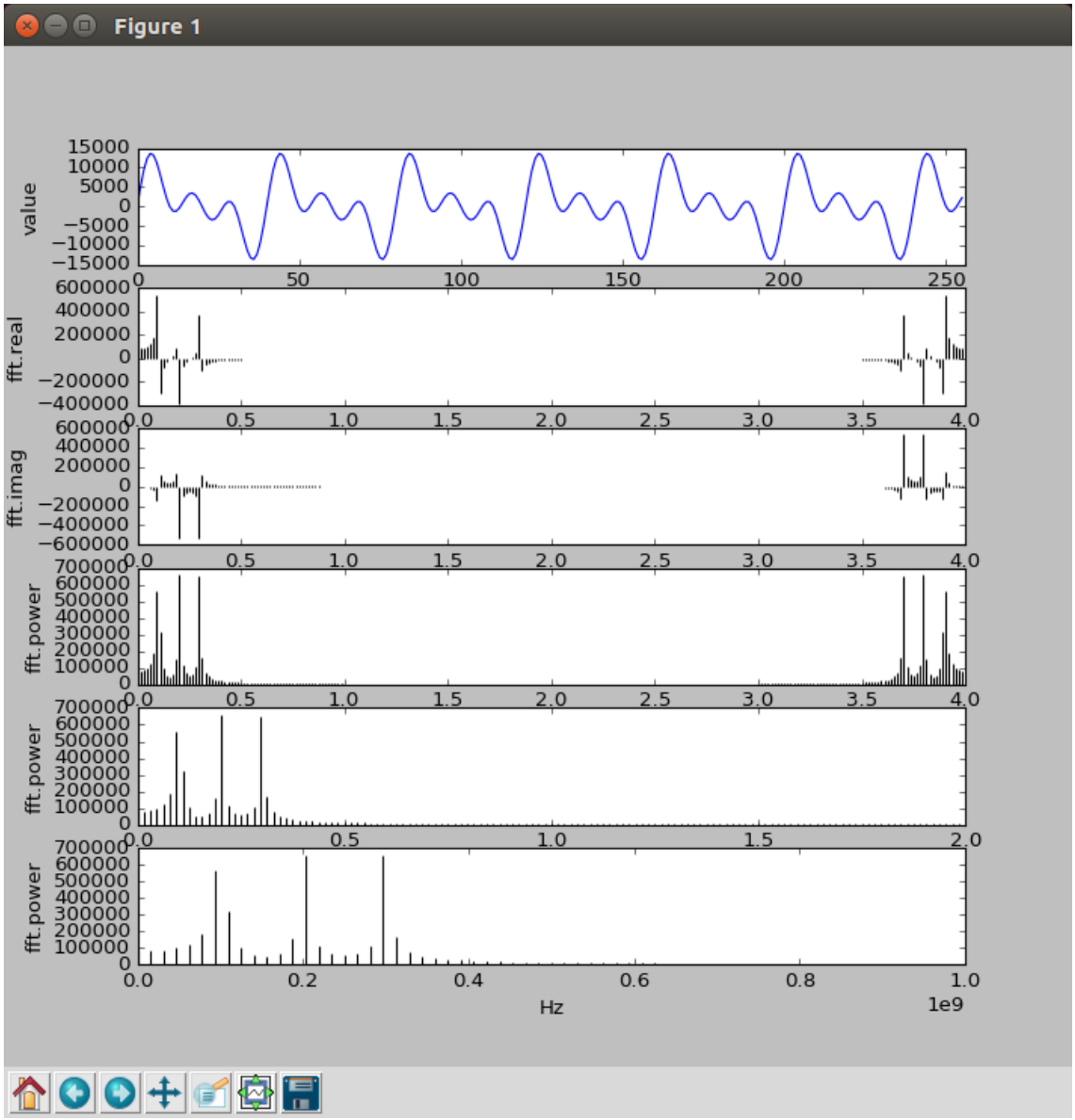
To see options, run ‘test’ with ‘-h’ command line option.

```
$ ./test -h
```

To see result, run ‘make’ with following targets

- ✧ plot: it plots signal and fft result using Python
- ✧ plot_data: it plots signal using GNUPLOT
- ✧ plot_fft: it plots fft result using GNUPLOT

```
$ make plot
```



Wish list

-

References

[1] Future Design Systems, Cypress EZ-USB FX3 GPIF-II Master Controller, FDS-TD-2018-04-002, 2018.

[2] Futue Design Systens, TRX_AXI: AMBA AXI Transactor for GPIF2MST, FDS-TD-2018-04-006, 2018.

[3] Futue Design Systens, CON-FMC API based on LIBUSB, FDS-TD-2018-04-004, 2018.

[4] AMBA Specification, Rev. 3.0, ARM.
[5] AMBA 4 AXI4-Stream Protocol Version: 1.0 Specification, IHI 0051A (ID030510), ARM Limited, 2010.
[6] Future Design Systems, AMBA AXI Memory-Mapped to Stream with Direct Memory Access, FDS-TD-2019-04-002, 2019.
[7] Future Design Systems, AMBA AXI Stream to Memory-Mapped with Direct Memory Access, FDS-TD-2019-04-003, 2019.
[8] Future Design Systems, Xilinx FFT Configuration Controller, FDS-TD-2019-04-004, 2019
[9] Xilinx, Fast Fourier Transform, LogiCORE IP product guide, PG109.

Revision history

☐ 2019.04.10.: Version 0 Revision 0 is prepared by Ando Ki.

-- End of Document --