Implementation and Validation of a Floating Point Unit Design Created Using VHDL

Adam Rushby
Wolfson School of Mechanical,
Electrical and Manufacturing Engineering
Loughborough University
Loughbrough, Leicstershire, UK
Email: a.rushby-14@student.lboro.ac.uk

***Abstract*—Abstract This paper presents the RTL design, Verification, and SoCFPGA implementation of an IEEE 754-compliant Floating Point Unit as well as an overview of the design work flow. The design incorporates the use of pre-compiled floating point operation blocks, and a custom control unit to handle data flow between the floating point operations and the processor. The design was written in VHDL using the IEEE 1164-standard library, validated at an Register Transfer Level using Modelsim, and then implemented onto a Xilinx ZYNQ SoC-FPGA using Vivado.**

I. Introduction

An SoC-FPGA (System on Chip Field Programmable Gate Array) is a semiconductor device that contains configurable logic blocks (CLBs) with programmable inter-connectors. This design allows for the configuration of the device to meet the desired functionality requirements set out by the device user. The particular SoC-FPGA used was a Xilinx ZYNQ-7000 SoC [1].

To access the tools necessary for designing the logic circuits, connection to a Linux server was needed. This was done through software named Xming and PuTTY. a .bashrc script is then used in the terminals to access the tools stored on the server. Xilinx CORE Generator was used to generate the VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Descriptive language) code for each individual floating point operation component needed for the FPU. The remaining parts of the FPU were then designed using VHDL within a basic test editor. A test bench for the design was created with the help of a c# script, and simulations were run using Modelsim. A Vivado .tcl is created to allow importing of the VHDL designs into Vivado. This piece of software contains all the tools necessary for generating the Bitstream files required for programming the SoC. For the project, the project type was set to RTL (Register transfer Level) with the target system being a Zedboard ZYNQ evaluation and development kit and the target language was set to VHDL.

Once the project is setup, a block diagram can be created. This will contain the system design for the target. Silicon IP can be added to the block diagram through a search function in the block diagram design area. For custom IP, this needs to be imported into the IP catalogue through the softwares menu drop-down. Once the block diagram is created, the design process is completed by validating it. Upon successful validation, a top level wrapper file can be created and a Bitstream file can be generated. Once the necessary files are created, the hardware design can be exported to the SDK (software development kit). The SoC-FPGA can then be programmed with the generated Bitstream file. Once programmed, multiple tests are then performed to determine whether the system performs as intended. Design performance is measured by comparing values recorded in reports produced by the Vivado design suite once the Bitstream has finished being generated.

II. Microarchitecture

The FPU (Floating Point Unit) Design is comprised of nine floating point operation blocks, a Control block and seven logic processes to control the flow of data throughout the FPU. The CMD_PROC is responsible for driving enable signals for five multiplexer-like processes that switch the flow of data depending on the current state of the enable signals. The VHDL code for these seven processes can be found in Appendix B and the overall design layout can be seen in Appendix F. The following operations are implemented into the FPU: addition and subtraction, multiply, divide, reciprocal, logarithm, absolute value, square root, floating point to integer, and integer to floating point. Each floating point operation block is designed to handle 32bit numbers with single precision. The VHDL designs for each block were generated using the Xilinx CORE Generator. The FPU is to be Interfaced with a set of AXI4LITE peripheral inter connectors, allowing it to receive instructions from the ARM processor on-board the SoC-FPGA.

A. CTRL

The Control block, labelled I_CTRL in the layout, controls the flow of data into and out of the FPU along with the internal operation state. The actions carried out by the control block and subsequently the rest of the FPU are given by a set of instructions in the form of an address, Data input, a read and a write signal. The architecture of CTRL is comprised of two internal signals, named: state_a and state_b, and a process named addr_proc. The VHDL code for the Control

block can be found in Appendix A.

The address port (Addr) is responsible for controlling the state of the FPU. Each address value from 0x00 to 0x2c is mapped to a specific action stored within the memory of the Control Block. The first 2 addresses control the write operation to the Control block and sets the operation command signal (cmd) of the FPU to its default value 0000 (no command). When the address is 0x00 and the write signal (wr) is HIGH, the state of the data in port (Din) is saved within the internal signal state_a. When the address is 0x04 and the write signal is high, the state of the data in port is saved within the internal signal state_b. For each mapped address that follows, the command signal is set to a value starting at 0001 for address 0x0, increasing all the way to 1010 for address 0x2c This value will determine which floating point operation is enabled within the FPU. Each subsequent address will cause the command signal to be set to its default value of 0000. Numbers 0001 through 1010 of the command signal each corresponds to a floating point operation within the FPU.

When the read signal (rd) is HIGH, the data stored in both state_a is piped to the output port operand a (opra) and the data stored in state_b is piped to output port operand b (oprb). The value of the output signal go also becomes HIGH. The input signal Dp_blocked determines if the FPU is ready to receive another set of instructions from outside the FPU. If Dp_blocked is HIGH, that means the FPU is currently busy processing a set of data. When a valid result is received (signal res_tvalid is HIGH) from one of the floating point operation blocks, the value of Dp_blocked is reset, allowing another set of instructions to be processed. For the result of a floating point operation to pipe out the FPU and HIGH validation signal must also be received at port int_res_valid. When this occurs, data at port int_res can pipe to the output port data out (Dout).

B. CMD_PROC

The Command Process (CMD_PROC) is responsible for receiving the Command signal (cmd), enabling its corresponding operation block, and, in the event the operation block can execute more the one command (see FADDSUB32), setting the correct operation for the operation block. The Command signal is used to drive a collection of nine enable signals labelled from en1_i to en9_i (en(1..9)_i). Each enable signal is responsible for controlling the flow of data to its corresponding operation block. Only one enable signal can be driven HIGH at any one time. The signal opr_addsub_tdata_i is a signal specific to the faddsub32 operation and is driven to value 0000000 when the value of the Command signal is 0001, and 00000001 when the value of the Command signal is 0010. It should be noted that Command signal values 0001 and 0010 both drive en1_i. This is the only time two different values drive the same enable signal.

C. BUSY_PROC

The Busy Process (BUSY_PROC) is responsible for driving the state of Dp_blocked to HIGH when it receives a HIGH state from the go signal, preventing any new data from entering the FPU pipeline from the Control block. It is also responsible for telling the active operation block that the FPU is ready to receive a result. This is done by driving the res_tready_i signal, that corresponds to the received enable signal, HIGH.

D. OPR_TDATA_PROC

The Operation Data Process (OPR_TDATA_PROC) is responsible for piping the signal opr_addsub_tdata_i through to the signal faddsub32_opr_tdata_i when it receives a HIGH signal from en1_i.

E. TDATA_PROC and TVALID_PROC

The Data Process (TDATA_PROC) is responsible for piping data from both operand signals (opra and oprb) to the data signals of the active operation block. The Valid Process (TVALID_PROC) is responsible for driving the data valid signals for the active operation block. Both actions are controlled by the nine enable signals.

F. FRES_TDATA_PROC and RES_TVALID_PROC

The Result Data Process (RES_TDATA_PROC) is responsible for controlling which result data output from the operation blocks will pipe back to the control block. This operation is controlled by the result valid output. Only the active operation block should output a valid data result. The Result Valid Process (RES_TVALID_PROC) controls the flow of the result valid signal back to the Control block and Busy Process.

G. FLOATING POINT OPERATION BLOCKS

Each Operation Block requires a set of data inputs. These data inputs contain the floating point operands, the ready signal and the valid signal. The ready signal is required for the operation block to output a data result, the valid signals tell the operation block that the input data operands are valid and that the result of that operation will be valid.

H. FADDSUB32

The FADDSUB32 is the addition/subtraction floating point operation block and is the only floating point operation block in the whole design to incorporate two operations in one block. This design choice means that the operation block requires an extra set of signals to control its internal operation state. This operation state is controlled by an 8bit signal and takes the values 00000000 and 00000001. 00000000 sets the state of the operation block to perform addition operations and 00000001 set the state of the operation block to perform subtraction operations.

I. FABS32

It is interesting to note that the FABS32 is the only floating point in the design to not require an aclk signal. This is due to the nature of the absolute operation which only changes the sign of the floating point number. This action is fast enough that it can be completed on the same clock cycle. This can be seen in Fig. 1

III. RTL Validation

To validate the designs at a register transfer level, a self-validating VHDL test bench is created that will apply 1000 vectors to each floating point operation in the design. Contained within the test-bench is a collection of 12 arrays; two date input arrays, and ten data output arrays (A test-bench file containing arrays of 10 data terms is shown in appendix C). Each array contains 1000 data terms that will be used to either drive the test bench or be used for result comparison. These arrays were generated using a C# script (Appendix D) that runs a synthetic test for the FPU, creating a set of known values. The script will create a formatted set of data for any size array and store it in a .txt file. The data from the .txt file does not require any editing and can then be copied into the test-bench. During the first loop within the test-bench, the first set of data terms is retrieved from the arrays within the test bench. The data is then written to the FPU and the active operation block will complete its operation and output a result. The result will then be compared against a known value for that operation given its data inputs and if the hardware result matches the software result then the FPU passes the test. Due to the behaviour of C# code, there will be occasions when the FPU will not pass the test due to numerical rounding within the data created by the C# script. This means that the Hardware result can be out by a single data bit. But this is the result of the C# script and not the hardware. To get around this, the test result is compared against the array data within the range of the result to the result plus 1. The test bench was then simulated within Modelsim. The Transcript for the completed simulation can be seen in Appendix G. The completed test-bench simulation can be seen in Appendix H.

IV. SoC FPGA Implementation

Now that the VHDL Design has been verified at a register transfer level, the design can be imported into Vivado for testing on an SoC-FPGA chipset. A Vivado .tcl script is require to package all the necessary VHDL and Netlist files required to implement the design. The script can be found in Appendix E.

Once the design is packaged, it can be imported into Vivado as a new piece of silicon IP. From there a block diagram can be created using a ZYNQ Processing System block and an AXI4LITE peripheral connection to allow the FPU to communicate with the Processing System. The Block Diagram can be seen in fig. 2

TABLE I
Timing and Area Results for Low Effort Compilation

Lanes	T Req (ns)	Freq (MHz)	RC Low Effort			
			T RC (ps)	Area RC	T Enc (ns)	Area Enc
2	2.5	400	0	884403	0.014	1202062.003
	5.0	200	-	881503	0.034	1192056.823
	10.0	100	-	855507	0.024	1164884.717
4	2.5	400	3	1758301	0.026	2337004.008
	5.0	200	-	1758760	0.076	2325966.670
	10.0	100	-	1702128	0.033	2265965.394
8	2.5	400	0	3505140	0.126	4565965.394
	5.0	200	-	3485128	0.182	4532412.024
	10.0	100	-	3396544	0.204	4446837.268
16	2.5	400	-	7051710	-	8923842.096
	5.0	200	-	6983309	0.125	8982284.0.36
	10.0	100	-	6780078	0.036	8774380.068

To allow the ZYNQ to properly connect to the AXI4LITE peripheral and the FPU, the ZYNQ block needs to be customised. Within the PS-PL configuration window the AXI Master Port interface GP0 is enabled. Moving into the clock configuration window, the PL fabric clock, FCLK_CLK0, is set to a frequency of 50MHz.

Now that the frequency is set to a known value, the circuit timings, power and hardware utilisation can be measured and compared against other frequencies. To compare and analyse timing of the design, the WNS (Worst Negative Slack), WHS (Worst Hold Slack), WPWS (Worst Pulse Width Slack) and the TNS (Total Negative Slack) is recorded [2][3]. For comparison and analysis of power for the design, the Total on-chip power is recorded. For the Hardware utilisation, the Number of Slice LUTs (Lookup-Table), Slice Registers usage, 36Kb BRAM and 18Kb BRAM usage, F7 and F8 Muxes, and DSP (Digital Signal Processor) usage is recorded [4]. The design is also tested at 100MHz for comparison (Table I. and Table II.). The implemented design option shows a visual representation of the device utilisation (Fig. 3.).

V. dasdasd

VI. Conclusion

Improvements to the FPU design can be made in two key areas, the data and valid processes and the result data and result valid processes. By combining the result and data processes into one processes, the VHDL code footprint can be reduced and the complexity of the layout diagram can be reduced. There is also a design flaw within the FPU that was overlooked during the design process where in the event no valid result signal is generated as a result of a data operation, which should not happen, the FPU will become locked, preventing further operations from be able to execute. This can be worked around by resetting the whole device.

Given a larger time frame, C code drivers could be created for running the design live on the Zedboard. A test-bench could then be ran using 1000000 vectors to provide a large workload to test the design with, providing another layer of validation.

Improvements could also be made to the C# script used to create the array data for the RTL test-bench. Removing the rounding error as best as possible will provide a more reliable validation test.

As has been shown through the Bitstream generations at multiple frequencies, as the frequency of the master port increases, the amount of time taken for each part of the timing report decreases. Increased total power usage is also seen as the frequency of the master port increases.

REFERENCES

APPENDIX A
CTRL VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library XilinxCoreLib;

entity ctrl is
port
(
clk : in std_logic;
reset : in std_logic;
Addr : in std_logic_vector(7 downto 0);
Din : in std_logic_vector(31 downto 0);
rd : in std_logic;
wr : in std_logic;
int_res : in std_logic_vector(31 downto 0);
int_res_valid : in std_logic;

dout : out std_logic_vector(31 downto 0);
opra : out std_logic_vector(31 downto 0);
oprb : out std_logic_vector(31 downto 0);
cmd : out std_logic_vector(3 downto 0);
go : out std_logic;
Dp_blocked : in std_logic --busy
);
end ctrl;

architecture rtl of ctrl is
signal state_a : std_logic_vector(31 downto 0)
    ;
signal state_b : std_logic_vector(31 downto 0)
    ;
begin
addr_proc : process(addr,reset,clk)
begin
if reset = '1' then
opra <= (others => '0');
oprb <= (others => '0');
cmd <= (others => '0');
dout <= (others => '0');
state_a <= (others => '0');
state_b <= (others => '0');
go <= '0';
elsif rising_edge(clk) then
if Dp_blocked = '0' then
--cmd <= "0000";
case conv_integer(addr) is
when 16#00# =>
cmd <= "0000"; -- no op
if wr = '1' then state_a <= Din; end if;
when 16#04# =>
cmd <= "0000"; -- no op
if wr = '1' then state_b <= Din; end if;
when 16#08# => cmd <= "0001"; --add
when 16#0c# => cmd <= "0010"; --sub
when 16#10# => cmd <= "0011"; --mul
when 16#14# => cmd <= "0100"; --div
when 16#18# => cmd <= "0101"; --absolute
when 16#1c# => cmd <= "0110"; --sqaure root
when 16#20# => cmd <= "0111"; --log
when 16#24# => cmd <= "1000"; --reciprocal
when 16#28# => cmd <= "1001"; --float to int
when 16#2c# => cmd <= "1010"; --int to float
when others => cmd <= "0000"; --else
end case;
if rd = '1' then
opra <= state_a;
oprb <= state_b;
go <= rd;
elsif rd = '0' then
go <= rd;
end if;
elsif Dp_blocked = '1' then
go <= '0';
end if;
if int_res_valid = '1' then
dout <= int_res;
end if;
end if;
end process;
end rtl;
```

APPENDIX B
FPU_WRAP

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
LIBRARY XilinxCoreLib;
-- synthesis translate_on
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity faddsub_wrap is

port
(
clk : in std_logic;
reset : in std_logic;
Addr : in std_logic_vector(7 downto 0);
Din : in std_logic_vector(31 downto 0);
rd : in std_logic;
wr : in std_logic;
dout : out std_logic_vector(31 downto 0)
);
end faddsub_wrap;

architecture rtl of faddsub_wrap is
```

```
signal opra_i, oprb_i, faddsub32_a_tdata_i,
    faddsub32_b_tdata_i, faddsub32_res_tdata_i
    , fmul32_a_tdata_i, fmul32_b_tdata_i,
    fmul32_res_tdata_i, fdiv32_a_tdata_i,
    fdiv32_b_tdata_i, fdiv32_res_tdata_i,
    fabs32_a_tdata_i, fabs32_res_tdata_i,
    fsqr32_a_tdata_i, fsqr32_res_tdata_i,
    flog32_a_tdata_i, flog32_res_tdata_i,
    frec32_a_tdata_i, frec32_res_tdata_i,
    f2i32_a_tdata_i, f2i32_res_tdata_i,
    i2f32_a_tdata_i, i2f32_res_tdata_i,
    res_tdata_i : std_logic_vector(31 downto
    0);

signal faddsub32_operation_tdata_i,
    opr_addsub_tdata_i : std_logic_vector(7
    downto 0);

signal cmd_i : std_logic_vector(3 downto 0);

signal go_i, dp_blocked_i,
    faddsub32_a_tvalid_i, faddsub32_b_tvalid_i
    , faddsub32_res_tvalid_i,
    faddsub32_res_tready_i,
    faddsub32_a_tready_i, faddsub32_b_tready_i
    , faddsub32_operation_tready_i,
    faddsub32_operation_tvalid_i,
    fmul32_a_tvalid_i, fmul32_b_tvalid_i,
    fmul32_res_tvalid_i, fmul32_res_tready_i,
    fmul32_a_tready_i, fmul32_b_tready_i,
    fdiv32_a_tvalid_i, fdiv32_b_tvalid_i,
    fdiv32_res_tvalid_i, fdiv32_res_tready_i,
    fdiv32_a_tready_i, fdiv32_b_tready_i,
    fabs32_a_tvalid_i, fabs32_res_tvalid_i,
    fabs32_res_tready_i, fabs32_a_tready_i,
    fsqr32_a_tvalid_i, fsqr32_res_tvalid_i,
    fsqr32_res_tready_i, fsqr32_a_tready_i,
    flog32_a_tvalid_i, flog32_res_tvalid_i,
    flog32_res_tready_i, flog32_a_tready_i,
    frec32_a_tvalid_i, frec32_res_tvalid_i,
    frec32_res_tready_i, frec32_a_tready_i,
    f2i32_a_tvalid_i, f2i32_res_tvalid_i,
    f2i32_res_tready_i, f2i32_a_tready_i,
    i2f32_a_tvalid_i, i2f32_res_tvalid_i,
    i2f32_res_tready_i, i2f32_a_tready_i,
    res_tvalid_i, en1_i, en2_i, en3_i, en4_i,
    en5_i, en6_i, en7_i, en8_i, en9_i :
    std_logic;

component faddsub32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
     0);
s_axis_b_tvalid : IN STD_LOGIC;
s_axis_b_tready : OUT STD_LOGIC;
s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
     0);
s_axis_operation_tvalid : IN STD_LOGIC;
s_axis_operation_tready : OUT STD_LOGIC;
s_axis_operation_tdata : IN STD_LOGIC_VECTOR(7
     DOWNTO 0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component ctrl
port
(
clk : in std_logic;
reset : in std_logic;
Addr : in std_logic_vector(7 downto 0);
Din : in std_logic_vector(31 downto 0);
rd : in std_logic;
wr : in std_logic;
int_res : in std_logic_vector(31 downto 0);
int_res_valid : in std_logic;
dout : out std_logic_vector(31 downto 0);
opra : out std_logic_vector(31 downto 0);
oprb : out std_logic_vector(31 downto 0);
cmd : out std_logic_vector(3 downto 0);
go : out std_logic;
Dp_blocked : in std_logic --busy
);
end component;

component fmul32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
     0);
s_axis_b_tvalid : IN STD_LOGIC;
s_axis_b_tready : OUT STD_LOGIC;
s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
     0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component fdiv32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
     0);
s_axis_b_tvalid : IN STD_LOGIC;
s_axis_b_tready : OUT STD_LOGIC;
s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
     0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component fabs32
port
(
s_axis_a_tvalid : IN STD_LOGIC;
```

```
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
    0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component fsqrt32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
    0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component flog32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
    0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component frec32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
    0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component f322i32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
    0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

component i322f32
port
(
aclk : IN STD_LOGIC;
s_axis_a_tvalid : IN STD_LOGIC;
s_axis_a_tready : OUT STD_LOGIC;
s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO
    0);
m_axis_result_tvalid : OUT STD_LOGIC;
m_axis_result_tready : IN STD_LOGIC;
m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31
    DOWNTO 0)
);
end component;

begin
--data flow case
CTRL_i : ctrl
port map (
clk => clk,
reset => reset,
Addr => Addr,
Din => Din,
rd => rd,
wr => wr,
int_res => res_tdata_i,
int_res_valid => res_tvalid_i,
dout => dout,
opra => opra_i,
oprb => oprb_i,
cmd => cmd_i,
go => go_i,
Dp_blocked => Dp_blocked_i
);

FAS : faddsub32
port map (
aclk => clk,
s_axis_a_tvalid => faddsub32_a_tvalid_i,
s_axis_a_tready => faddsub32_a_tready_i,
s_axis_a_tdata => faddsub32_a_tdata_i,
s_axis_b_tvalid => faddsub32_b_tvalid_i,
s_axis_b_tready => faddsub32_b_tready_i,
s_axis_b_tdata => faddsub32_b_tdata_i,
s_axis_operation_tvalid =>
    faddsub32_operation_tvalid_i,
s_axis_operation_tready =>
    faddsub32_operation_tready_i,
s_axis_operation_tdata =>
    faddsub32_operation_tdata_i,
m_axis_result_tvalid => faddsub32_res_tvalid_i
    ,
m_axis_result_tready => faddsub32_res_tready_i
    ,
m_axis_result_tdata => faddsub32_res_tdata_i
);

FMUL : fmul32
port map (
aclk => clk,
s_axis_a_tvalid => fmul32_a_tvalid_i,
s_axis_a_tready => fmul32_a_tready_i,
s_axis_a_tdata => fmul32_a_tdata_i,
s_axis_b_tvalid => fmul32_b_tvalid_i,
```

```
s_axis_b_tready => fmul32_b_tready_i,
s_axis_b_tdata => fmul32_b_tdata_i,
m_axis_result_tvalid => fmul32_res_tvalid_i,
m_axis_result_tready => fmul32_res_tready_i,
m_axis_result_tdata => fmul32_res_tdata_i
);

FDIV : fdiv32
port map (
aclk => clk,
s_axis_a_tvalid => fdiv32_a_tvalid_i,
s_axis_a_tready => fdiv32_a_tready_i,
s_axis_a_tdata => fdiv32_a_tdata_i,
s_axis_b_tvalid => fdiv32_b_tvalid_i,
s_axis_b_tready => fdiv32_b_tready_i,
s_axis_b_tdata => fdiv32_b_tdata_i,
m_axis_result_tvalid => fdiv32_res_tvalid_i,
m_axis_result_tready => fdiv32_res_tready_i,
m_axis_result_tdata => fdiv32_res_tdata_i
);

FABS : fabs32
port map (
s_axis_a_tvalid => fabs32_a_tvalid_i,
s_axis_a_tready => fabs32_a_tready_i,
s_axis_a_tdata => fabs32_a_tdata_i,
m_axis_result_tvalid => fabs32_res_tvalid_i,
m_axis_result_tready => fabs32_res_tready_i,
m_axis_result_tdata => fabs32_res_tdata_i
);

FSQR : fsqrt32
port map (
aclk => clk,
s_axis_a_tvalid => fsqr32_a_tvalid_i,
s_axis_a_tready => fsqr32_a_tready_i,
s_axis_a_tdata => fsqr32_a_tdata_i,
m_axis_result_tvalid => fsqr32_res_tvalid_i,
m_axis_result_tready => fsqr32_res_tready_i,
m_axis_result_tdata => fsqr32_res_tdata_i
);

FLOG :flog32
port map (
aclk => clk,
s_axis_a_tvalid => flog32_a_tvalid_i,
s_axis_a_tready => flog32_a_tready_i,
s_axis_a_tdata => flog32_a_tdata_i,
m_axis_result_tvalid => flog32_res_tvalid_i,
m_axis_result_tready => flog32_res_tready_i,
m_axis_result_tdata => flog32_res_tdata_i
);

FREC : frec32
port map (
aclk => clk,
s_axis_a_tvalid => frec32_a_tvalid_i,
s_axis_a_tready => frec32_a_tready_i,
s_axis_a_tdata => frec32_a_tdata_i,
m_axis_result_tvalid => frec32_res_tvalid_i,
m_axis_result_tready => frec32_res_tready_i,
m_axis_result_tdata => frec32_res_tdata_i
);

F2I : f322i32
port map (
aclk => clk,
s_axis_a_tvalid => f2i32_a_tvalid_i,
s_axis_a_tready => f2i32_a_tready_i,
s_axis_a_tdata => f2i32_a_tdata_i,
m_axis_result_tvalid => f2i32_res_tvalid_i,
m_axis_result_tready => f2i32_res_tready_i,
m_axis_result_tdata => f2i32_res_tdata_i
);

I2F : i322f32
port map (
aclk => clk,
s_axis_a_tvalid => i2f32_a_tvalid_i,
s_axis_a_tready => i2f32_a_tready_i,
s_axis_a_tdata => i2f32_a_tdata_i,
m_axis_result_tvalid => i2f32_res_tvalid_i,
m_axis_result_tready => i2f32_res_tready_i,
m_axis_result_tdata => i2f32_res_tdata_i
);

I_cmd_PROC : process(cmd_i, go_i)
begin
opr_addsub_tdata_i <= "00000000";
en1_i <= '0';--addsub
en2_i <= '0';--mul
en3_i <= '0';--div
en4_i <= '0';--ads
en5_i <= '0';--sqr
en6_i <= '0';--log
en7_i <= '0';--rec
en8_i <= '0';--f2i
en9_i <= '0';--i2f
case (cmd_i) is
when "0001" => --add
if go_i = '1' then
en1_i <= '1';
opr_addsub_tdata_i <= "00000000";
end if;
when "0010" => --sub
if go_i = '1' then
en1_i <= '1';
opr_addsub_tdata_i <= "00000001";
end if;
when "0011" => --mul
if go_i = '1' then
en2_i <= '1';
end if;
when "0100" => --div
if go_i = '1' then
en3_i <= '1';
end if;
when "0101" => --abs
if go_i = '1' then
en4_i <= '1';
end if;
when "0110" => --abs
if go_i = '1' then
en5_i <= '1';
end if;
when "0111" => --log
if go_i = '1' then
en6_i <= '1';
end if;
when "1000" => --log
if go_i = '1' then
en7_i <= '1';
end if;
when "1001" => --f2i
```

```
if go_i = '1' then
en8_i <= '1';
end if;
when "1010" => --i2f
if go_i = '1' then
en9_i <= '1';
end if;
when others => -- else
opr_addsub_tdata_i <= "00000000";
end case;
end process;

-- 1 bit register with enable
I_Busy_PROC : process(clk , reset, go_i,
    res_tvalid_i)
begin
if reset = '1' then
Dp_blocked_i <= '0';
faddsub32_res_tready_i <= '0';
fmul32_res_tready_i <= '0';
fdiv32_res_tready_i <= '0';
fabs32_res_tready_i <= '0';
fsqr32_res_tready_i <= '0';
flog32_res_tready_i <= '0';
frec32_res_tready_i <= '0';
f2i32_res_tready_i <= '0';
i2f32_res_tready_i <= '0';
elsif rising_edge(clk) then
if go_i = '1' then
Dp_blocked_i <= '1';
if en1_i = '1' then
faddsub32_res_tready_i <= '1';
elsif en2_i = '1' then
fmul32_res_tready_i <= '1';
elsif en3_i = '1' then
fdiv32_res_tready_i <= '1';
elsif en4_i = '1' then
fabs32_res_tready_i <= '1';
elsif en5_i = '1' then
fsqr32_res_tready_i <= '1';
elsif en6_i = '1' then
flog32_res_tready_i <= '1';
elsif en7_i = '1' then
frec32_res_tready_i <= '1';
elsif en8_i = '1' then
f2i32_res_tready_i <= '1';
elsif en9_i = '1' then
i2f32_res_tready_i <= '1';
end if;
elsif res_tvalid_i = '1' then
DP_blocked_i <= '0';
faddsub32_res_tready_i <= '0';
fmul32_res_tready_i <= '0';
fdiv32_res_tready_i <= '0';
fabs32_res_tready_i <= '0';
fsqr32_res_tready_i <= '0';
flog32_res_tready_i <= '0';
frec32_res_tready_i <= '0';
f2i32_res_tready_i <= '0';
i2f32_res_tready_i <= '0';
end if;
end if;
end process;
-- 32 bit register with enable -- addsub
I_tdata_Proc : process(clk, reset, en1_i,
    res_tvalid_i)
begin
if reset = '1' then
faddsub32_a_tdata_i <= (others => '0');--
    addsub
faddsub32_b_tdata_i <= (others => '0');
fmul32_a_tdata_i <= (others => '0');-- mul
fmul32_b_tdata_i <= (others => '0');
fdiv32_a_tdata_i <= (others => '0');-- div
fdiv32_b_tdata_i <= (others => '0');
fabs32_a_tdata_i <= (others => '0');-- abs
fsqr32_a_tdata_i <= (others => '0');-- sqr
flog32_a_tdata_i <= (others => '0');-- log
frec32_a_tdata_i <= (others => '0');-- rec
f2i32_a_tdata_i <= (others => '0');-- f2i
i2f32_a_tdata_i <= (others => '0');-- i2f
elsif rising_edge(clk) then
if en1_i = '1' then
faddsub32_a_tdata_i <= opra_i; -- addsub
faddsub32_b_tdata_i <= oprb_i;
elsif en2_i = '1' then
fmul32_a_tdata_i <= opra_i;-- mul
fmul32_b_tdata_i <= oprb_i;
elsif en3_i = '1' then
fdiv32_a_tdata_i <= opra_i;-- div
fdiv32_b_tdata_i <= oprb_i;
elsif en4_i = '1' then
fabs32_a_tdata_i <= opra_i;-- abs
elsif en5_i = '1' then
fsqr32_a_tdata_i <= opra_i;-- sqr
elsif en6_i = '1' then
flog32_a_tdata_i <= opra_i;-- log
elsif en7_i = '1' then
frec32_a_tdata_i <= opra_i;-- rec
elsif en8_i = '1' then
f2i32_a_tdata_i <= opra_i;-- f2i
elsif en9_i = '1' then
i2f32_a_tdata_i <= opra_i;-- i2f
elsif res_tvalid_i = '1' then
faddsub32_a_tdata_i <= (others => '0'); --
    addsub
faddsub32_b_tdata_i <= (others => '0');
fmul32_a_tdata_i <= (others => '0');-- mul
fmul32_b_tdata_i <= (others => '0');
fdiv32_a_tdata_i <= (others => '0');-- div
fdiv32_b_tdata_i <= (others => '0');
fabs32_a_tdata_i <= (others => '0');-- abs
fsqr32_a_tdata_i <= (others => '0');-- sqr
flog32_a_tdata_i <= (others => '0');-- log
frec32_a_tdata_i <= (others => '0');-- rec
f2i32_a_tdata_i <= (others => '0');-- f2i
i2f32_a_tdata_i <= (others => '0');-- i2f
end if;
end if;
end process;
-- 8 bit register with enable -- addsub only
I_opr_tdata_Proc : process(clk, reset, en1_i,
    res_tvalid_i)
begin
if reset = '1' then
faddsub32_operation_tdata_i <= (others => '0')
    ;
elsif rising_edge(clk) then
if en1_i = '1' then
faddsub32_operation_tdata_i <=
    opr_addsub_tdata_i;
elsif res_tvalid_i = '1' then
faddsub32_operation_tdata_i <= (others => '0')
    ;
```

```
end if;
end if;
end process;
-- 1 bit register with enable
I_valid_PROC : process(clk , reset, en1_i,
    res_tvalid_i)
begin
if reset = '1' then
faddsub32_operation_tvalid_i <= '0';
faddsub32_a_tvalid_i <= '0';
faddsub32_b_tvalid_i <= '0';
fmul32_a_tvalid_i <= '0';
fmul32_b_tvalid_i <= '0';
fdiv32_a_tvalid_i <= '0';
fdiv32_b_tvalid_i <= '0';
fabs32_a_tvalid_i <= '0';
fsqr32_a_tvalid_i <= '0';
flog32_a_tvalid_i <= '0';
frec32_a_tvalid_i <= '0';
f2i32_a_tvalid_i <= '0';
i2f32_a_tvalid_i <= '0';
elsif rising_edge(clk) then
if en1_i = '1' then
faddsub32_operation_tvalid_i <= '1';
faddsub32_a_tvalid_i <= '1';
faddsub32_b_tvalid_i <= '1';
elsif en2_i = '1' then
fmul32_a_tvalid_i <= '1';
fmul32_b_tvalid_i <= '1';
elsif en3_i = '1' then
fdiv32_a_tvalid_i <= '1';
fdiv32_b_tvalid_i <= '1';
elsif en4_i = '1' then
fabs32_a_tvalid_i <= '1';
elsif en5_i = '1' then
fsqr32_a_tvalid_i <= '1';
elsif en6_i = '1' then
flog32_a_tvalid_i <= '1';
elsif en7_i = '1' then
frec32_a_tvalid_i <= '1';
elsif en8_i = '1' then
f2i32_a_tvalid_i <= '1';
elsif en9_i = '1' then
i2f32_a_tvalid_i <= '1';
elsif res_tvalid_i <= '1' then
faddsub32_operation_tvalid_i <= '0';
faddsub32_a_tvalid_i <= '0';
faddsub32_b_tvalid_i <= '0';
fmul32_a_tvalid_i <= '0';
fmul32_b_tvalid_i <= '0';
fdiv32_a_tvalid_i <= '0';
fdiv32_b_tvalid_i <= '0';
fabs32_a_tvalid_i <= '0';
fsqr32_a_tvalid_i <= '0';
flog32_a_tvalid_i <= '0';
frec32_a_tvalid_i <= '0';
f2i32_a_tvalid_i <= '0';
i2f32_a_tvalid_i <= '0';
end if;
end if;
end process;

-- 1 bit register -- result
I_res_tvalid_PROC : process(clk, reset)
begin
if reset = '1' then
res_tvalid_i <= '0';
elsif rising_edge(clk) then
if faddsub32_res_tready_i = '1' then
res_tvalid_i <= faddsub32_res_tvalid_i;
elsif fmul32_res_tready_i = '1' then
res_tvalid_i <= fmul32_res_tvalid_i;
elsif fdiv32_res_tready_i = '1' then
res_tvalid_i <= fdiv32_res_tvalid_i;
elsif fabs32_res_tready_i = '1' then
res_tvalid_i <= fabs32_res_tvalid_i;
elsif fsqr32_res_tready_i = '1' then
res_tvalid_i <= fsqr32_res_tvalid_i;
elsif flog32_res_tready_i = '1' then
res_tvalid_i <= flog32_res_tvalid_i;
elsif frec32_res_tready_i = '1' then
res_tvalid_i <= frec32_res_tvalid_i;
elsif f2i32_res_tready_i = '1' then
res_tvalid_i <= f2i32_res_tvalid_i;
elsif i2f32_res_tready_i = '1' then
res_tvalid_i <= i2f32_res_tvalid_i;
end if;
end if;
end process;
-- 32 bit register -- result
I_res_tdata_PROC : process(clk, reset)
begin
if reset = '1' then
res_tdata_i <= (others => '0');
elsif rising_edge(clk) then
if faddsub32_res_tready_i = '1' then
res_tdata_i <= faddsub32_res_tdata_i;
elsif fmul32_res_tready_i = '1' then
res_tdata_i <= fmul32_res_tdata_i;
elsif fdiv32_res_tready_i = '1' then
res_tdata_i <= fdiv32_res_tdata_i;
elsif fabs32_res_tready_i = '1' then
res_tdata_i <= fabs32_res_tdata_i;
elsif fsqr32_res_tready_i = '1' then
res_tdata_i <= fsqr32_res_tdata_i;
elsif flog32_res_tready_i = '1' then
res_tdata_i <= flog32_res_tdata_i;
elsif frec32_res_tready_i = '1' then
res_tdata_i <= frec32_res_tdata_i;
elsif f2i32_res_tready_i = '1' then
res_tdata_i <= f2i32_res_tdata_i;
elsif i2f32_res_tready_i = '1' then
res_tdata_i <= i2f32_res_tdata_i;
end if;
end if;
end process;
end rtl;
```

APPENDIX C
TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
LIBRARY XilinxCoreLib;
-- synthesis translate_on
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--use work.fpu_package.all

entity tb_faddsub_wrap is

end tb_faddsub_wrap;
```

```
architecture exercise of tb_faddsub_wrap is

component faddsub_wrap
port
(
clk : in std_logic;
reset : in std_logic;
Addr : in std_logic_vector(7 downto 0);
Din : in std_logic_vector(31 downto 0);
rd : in std_logic;
wr : in std_logic;
dout : out std_logic_vector(31 downto 0)
);
end component;

signal clk_i : std_logic;
signal reset_i : std_logic;
signal Addr_i : std_logic_vector(7 downto 0);
signal Din_i : std_logic_vector(31 downto 0);
signal rd_i : std_logic;
signal wr_i : std_logic;
signal dout_i : std_logic_vector(31 downto 0);

constant clk_period : time := 10 ns;
constant c2q : time := clk_period/4;

type FPUConstant is array(999 downto 0) of
    std_logic_vector(31 downto 0);
signal A_Constant : FPUConstant;
signal B_Constant : FPUConstant;
signal ADDRESConstant : FPUConstant;
signal SUBRESConstant : FPUConstant;
signal MULRESConstant : FPUConstant;
signal DIVRESConstant : FPUConstant;
signal SQRRESConstant : FPUConstant;
signal ABSRESConstant : FPUConstant;
signal LOGRESConstant : FPUConstant;
signal RECRESConstant : FPUConstant;
signal F2IRESConstant : FPUConstant;
signal I2FRESConstant : FPUConstant;
begin

A_Constant(0) <= conv_std_logic_vector
    (16#40000000#,32); --2
A_Constant(1) <= conv_std_logic_vector
    (16#40400000#,32); --3
A_Constant(2) <= conv_std_logic_vector
    (16#40800000#,32); --4
A_Constant(3) <= conv_std_logic_vector(16#40
    A00000#,32); --5
A_Constant(4) <= conv_std_logic_vector(16#40
    C00000#,32); --6
A_Constant(5) <= conv_std_logic_vector(16#40
    E00000#,32); --7
A_Constant(6) <= conv_std_logic_vector
    (16#41000000#,32); --8
A_Constant(7) <= conv_std_logic_vector
    (16#41100000#,32); --9
A_Constant(8) <= conv_std_logic_vector
    (16#41200000#,32); --10
A_Constant(9) <= conv_std_logic_vector
    (16#41300000#,32); --11


B_Constant(0) <= conv_std_logic_vector(16#3
    F800000#,32); --1
B_Constant(1) <= conv_std_logic_vector
    (16#40000000#,32); --2
B_Constant(2) <= conv_std_logic_vector
    (16#40400000#,32); --3
B_Constant(3) <= conv_std_logic_vector
    (16#40800000#,32); --4
B_Constant(4) <= conv_std_logic_vector(16#40
    A00000#,32); --5
B_Constant(5) <= conv_std_logic_vector(16#40
    C00000#,32); --6
B_Constant(6) <= conv_std_logic_vector(16#40
    E00000#,32); --7
B_Constant(7) <= conv_std_logic_vector
    (16#41000000#,32); --8
B_Constant(8) <= conv_std_logic_vector
    (16#41100000#,32); --9
B_Constant(9) <= conv_std_logic_vector
    (16#41200000#,32); --10


ADDRESConstant(0) <= conv_std_logic_vector
    (16#40400000#,32); --3
ADDRESConstant(1) <= conv_std_logic_vector
    (16#40A00000#,32); --5
ADDRESConstant(2) <= conv_std_logic_vector
    (16#40E00000#,32); --7
ADDRESConstant(3) <= conv_std_logic_vector
    (16#41100000#,32); --9
ADDRESConstant(4) <= conv_std_logic_vector
    (16#41300000#,32); --11
ADDRESConstant(5) <= conv_std_logic_vector
    (16#41500000#,32); --13
ADDRESConstant(6) <= conv_std_logic_vector
    (16#41700000#,32); --15
ADDRESConstant(7) <= conv_std_logic_vector
    (16#41880000#,32); --17
ADDRESConstant(8) <= conv_std_logic_vector
    (16#41980000#,32); --19
ADDRESConstant(9) <= conv_std_logic_vector
    (16#41A80000#,32); --21


SUBRESConstant(0) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(1) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(2) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(3) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(4) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(5) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(6) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(7) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(8) <= conv_std_logic_vector
    (16#3F800000#,32); --1
SUBRESConstant(9) <= conv_std_logic_vector
    (16#3F800000#,32); --1


DIVRESConstant(0) <= conv_std_logic_vector
    (16#40000000#,32); --2
DIVRESConstant(1) <= conv_std_logic_vector
    (16#3FC00000#,32); --1.5
```

```
DIVRESConstant(2) <= conv_std_logic_vector
    (16#3FAAAAAB#,32); --1.333333
DIVRESConstant(3) <= conv_std_logic_vector
    (16#3FA00000#,32); --1.25
DIVRESConstant(4) <= conv_std_logic_vector
    (16#3F99999A#,32); --1.2
DIVRESConstant(5) <= conv_std_logic_vector
    (16#3F955555#,32); --1.166667
DIVRESConstant(6) <= conv_std_logic_vector
    (16#3F924925#,32); --1.142857
DIVRESConstant(7) <= conv_std_logic_vector
    (16#3F900000#,32); --1.125
DIVRESConstant(8) <= conv_std_logic_vector
    (16#3F8E38E4#,32); --1.111111
DIVRESConstant(9) <= conv_std_logic_vector
    (16#3F8CCCCD#,32); --1.1


MULRESConstant(0) <= conv_std_logic_vector
    (16#40000000#,32); --2
MULRESConstant(1) <= conv_std_logic_vector
    (16#40C00000#,32); --6
MULRESConstant(2) <= conv_std_logic_vector
    (16#41400000#,32); --12
MULRESConstant(3) <= conv_std_logic_vector
    (16#41A00000#,32); --20
MULRESConstant(4) <= conv_std_logic_vector
    (16#41F00000#,32); --30
MULRESConstant(5) <= conv_std_logic_vector
    (16#42280000#,32); --42
MULRESConstant(6) <= conv_std_logic_vector
    (16#42600000#,32); --56
MULRESConstant(7) <= conv_std_logic_vector
    (16#42900000#,32); --72
MULRESConstant(8) <= conv_std_logic_vector
    (16#42B40000#,32); --90
MULRESConstant(9) <= conv_std_logic_vector
    (16#42DC0000#,32); --110


SQRRESConstant(0) <= conv_std_logic_vector
    (16#3FB504F3#,32); --1.414214
SQRRESConstant(1) <= conv_std_logic_vector
    (16#3FDDB3D7#,32); --1.732051
SQRRESConstant(2) <= conv_std_logic_vector
    (16#40000000#,32); --2
SQRRESConstant(3) <= conv_std_logic_vector
    (16#400F1BBD#,32); --2.236068
SQRRESConstant(4) <= conv_std_logic_vector
    (16#401CC471#,32); --2.44949
SQRRESConstant(5) <= conv_std_logic_vector
    (16#402953FD#,32); --2.645751
SQRRESConstant(6) <= conv_std_logic_vector
    (16#403504F3#,32); --2.828427
SQRRESConstant(7) <= conv_std_logic_vector
    (16#40400000#,32); --3
SQRRESConstant(8) <= conv_std_logic_vector
    (16#404A62C2#,32); --3.162278
SQRRESConstant(9) <= conv_std_logic_vector
    (16#40544395#,32); --3.316625


ABSRESConstant(0) <= conv_std_logic_vector
    (16#40000000#,32); --2
ABSRESConstant(1) <= conv_std_logic_vector
    (16#40400000#,32); --3
ABSRESConstant(2) <= conv_std_logic_vector
    (16#40800000#,32); --4
ABSRESConstant(3) <= conv_std_logic_vector
    (16#40A00000#,32); --5
ABSRESConstant(4) <= conv_std_logic_vector
    (16#40C00000#,32); --6
ABSRESConstant(5) <= conv_std_logic_vector
    (16#40E00000#,32); --7
ABSRESConstant(6) <= conv_std_logic_vector
    (16#41000000#,32); --8
ABSRESConstant(7) <= conv_std_logic_vector
    (16#41100000#,32); --9
ABSRESConstant(8) <= conv_std_logic_vector
    (16#41200000#,32); --10
ABSRESConstant(9) <= conv_std_logic_vector
    (16#41300000#,32); --11


LOGRESConstant(0) <= conv_std_logic_vector
    (16#3F317218#,32); --0.6931472
LOGRESConstant(1) <= conv_std_logic_vector
    (16#3F8C9F54#,32); --1.098612
LOGRESConstant(2) <= conv_std_logic_vector
    (16#3FB17218#,32); --1.386294
LOGRESConstant(3) <= conv_std_logic_vector
    (16#3FCE0210#,32); --1.609438
LOGRESConstant(4) <= conv_std_logic_vector
    (16#3FE55860#,32); --1.791759
LOGRESConstant(5) <= conv_std_logic_vector
    (16#3FF91395#,32); --1.94591
LOGRESConstant(6) <= conv_std_logic_vector
    (16#40051592#,32); --2.079442
LOGRESConstant(7) <= conv_std_logic_vector
    (16#400C9F54#,32); --2.197225
LOGRESConstant(8) <= conv_std_logic_vector
    (16#40135D8E#,32); --2.302585
LOGRESConstant(9) <= conv_std_logic_vector
    (16#4019771E#,32); --2.397895


RECRESConstant(0) <= conv_std_logic_vector
    (16#3F000000#,32); --0.5
RECRESConstant(1) <= conv_std_logic_vector
    (16#3EAAAAAB#,32); --0.3333333
RECRESConstant(2) <= conv_std_logic_vector
    (16#3E800000#,32); --0.25
RECRESConstant(3) <= conv_std_logic_vector
    (16#3E4CCCCD#,32); --0.2
RECRESConstant(4) <= conv_std_logic_vector
    (16#3E2AAAAB#,32); --0.1666667
RECRESConstant(5) <= conv_std_logic_vector
    (16#3E124925#,32); --0.1428571
RECRESConstant(6) <= conv_std_logic_vector
    (16#3E000000#,32); --0.125
RECRESConstant(7) <= conv_std_logic_vector
    (16#3DE38E39#,32); --0.1111111
RECRESConstant(8) <= conv_std_logic_vector
    (16#3DCCCCCD#,32); --0.1
RECRESConstant(9) <= conv_std_logic_vector
    (16#3DBA2E8C#,32); --0.09090909


F2IRESConstant(0) <= conv_std_logic_vector
    (2,32); --40000000
F2IRESConstant(1) <= conv_std_logic_vector
    (3,32); --40400000
F2IRESConstant(2) <= conv_std_logic_vector
    (4,32); --40800000
```

```
F2IRESConstant(3) <= conv_std_logic_vector
    (5,32); --40A00000
F2IRESConstant(4) <= conv_std_logic_vector
    (6,32); --40C00000
F2IRESConstant(5) <= conv_std_logic_vector
    (7,32); --40E00000
F2IRESConstant(6) <= conv_std_logic_vector
    (8,32); --41000000
F2IRESConstant(7) <= conv_std_logic_vector
    (9,32); --41100000
F2IRESConstant(8) <= conv_std_logic_vector
    (10,32); --41200000
F2IRESConstant(9) <= conv_std_logic_vector
    (11,32); --41300000


I2FRESConstant(0) <= conv_std_logic_vector
    (16#40000000#,32); --2
I2FRESConstant(1) <= conv_std_logic_vector
    (16#40400000#,32); --3
I2FRESConstant(2) <= conv_std_logic_vector
    (16#40800000#,32); --4
I2FRESConstant(3) <= conv_std_logic_vector
    (16#40A00000#,32); --5
I2FRESConstant(4) <= conv_std_logic_vector
    (16#40C00000#,32); --6
I2FRESConstant(5) <= conv_std_logic_vector
    (16#40E00000#,32); --7
I2FRESConstant(6) <= conv_std_logic_vector
    (16#41000000#,32); --8
I2FRESConstant(7) <= conv_std_logic_vector
    (16#41100000#,32); --9
I2FRESConstant(8) <= conv_std_logic_vector
    (16#41200000#,32); --10
I2FRESConstant(9) <= conv_std_logic_vector
    (16#41300000#,32); --11



reset_me : process
begin
reset_i <= '1';
wait for clk_period;
reset_i <= '0';
wait;
end process;

clk_me : process
begin
clk_i <= '1';
wait for clk_period/2;
clk_i <= '0';
wait for clk_period/2;
end process;

DUT : faddsub_wrap
port map
(
clk => clk_i,
reset => reset_i,
Addr => Addr_i,
Din => Din_i,
rd => rd_i,
wr => wr_i,
dout => dout_i
);
```

```
stimPROC : process
begin
wait on reset_i until reset_i = '0';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= conv_std_logic_vector(0,32);
rd_i <= '0';
wr_i <= '0';
wait for clk_period;
--
for i in 0 to 999 loop --add
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
addr_i <= conv_std_logic_vector(4,8);
Din_i <= B_constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(8,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*15;
assert (dout_i = ADDRESConstant(i))
report "add error"
severity error;
end loop;
assert (false)
report "add complete"
severity error;
--
for i in 0 to 999 loop --sub
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
addr_i <= conv_std_logic_vector(4,8);
Din_i <= B_constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(12,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*15;
assert (dout_i = SUBRESConstant(i))
report "sub error"
severity error;
end loop;
assert (false)
report "sub complete"
severity error;
--
for i in 0 to 999 loop --mul
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
addr_i <= conv_std_logic_vector(4,8);
Din_i <= B_Constant(i);
```

```
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(16,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*12;
assert (dout_i = MULRESConstant(i))
report "mul error"
severity error;
end loop;
assert (false)
report "mul complete"
severity error;
--
for i in 0 to 999 loop --div
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
addr_i <= conv_std_logic_vector(4,8);
Din_i <= B_Constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(20,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*32;
assert (dout_i = DIVRESConstant(i))
report "div error"
severity error;
end loop;
assert (false)
report "div complete"
severity error;
--
for i in 0 to 999 loop --abs
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(24,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*3;
assert (dout_i = ABSRESConstant(i))
report "abs error"
severity error;
end loop;
assert (false)
report "abs complete"
severity error;
--
for i in 0 to 999 loop --sqrt
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(28,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*32;
assert (dout_i = SQRRESConstant(i))
report "sqrt error"
severity error;
end loop;
assert (false)
report "sqrt complete"
severity error;
--
for i in 0 to 999 loop --log
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(32,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*26;
assert ((dout_i <= LOGRESConstant(i)) and (
    dout_i >= LOGRESConstant(i)-1))
report "log error"
severity error;
end loop;
assert (false)
report "log complete"
severity error;
--
for i in 0 to 999 loop --rec
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(36,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*33;
assert ((dout_i <= RECRESConstant(i)+1) and (
    dout_i >= RECRESConstant(i)-1))
report "rec error"
severity error;
end loop;
assert (false)
report "rec complete"
severity error;
--
for i in 0 to 999 loop --f2i
--write to internal opr states
```

```
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= A_Constant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(40,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*10;
assert (dout_i = F2IRESConstant(i))
report "F2I error"
severity error;
end loop;
assert (false)
report "F2I complete"
severity error;
--
for i in 0 to 999 loop --i2f
--write to internal opr states
wr_i <='1';
addr_i <= conv_std_logic_vector(0,8);
Din_i <= F2IRESConstant(i);
wait for clk_period;
-- set opertaion state
addr_i <= conv_std_logic_vector(44,8);
-- read from internal opr states
wr_i <= '0';
rd_i <= '1';
wait for clk_period;
rd_i <= '0';
wait for clk_period*10;
assert (dout_i = I2FRESConstant(i))
report "I2F error"
severity error;
end loop;
assert (false)
report "I2F complete"
severity error;
--
wait for clk_period*10;
assert (false)
report "SIMULATION END"
severity failure;

end process;
end exercise;
```

APPENDIX D
C# SCRIPT

```
using System;
using System.Collections.Generic;
using System.IO;
namespace ConsoleApp1
{
class Program
{
static void Main(string[] args)
{
double b = 2.0;
Console.WriteLine("enter the number of loops
    to generate for");
string line = Console.ReadLine();
int a = Convert.ToInt32(line);
int k = a - 1;
try
{
StreamWriter sw = new StreamWriter("
    fpu_package.txt", false);
sw.WriteLine("");
sw.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
Console.WriteLine("--");
string d = "";
string e = "";
double g = 0;
for (int i =1;i <= a; i++)
{
float c = (float)b;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("A_Constant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c); wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
}
Space();
Console.WriteLine("--");
double f = 1.0;
for (int i = 1; i <= a; i++)
{
float c = (float)f;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("B_Constant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c); wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b + f;
float c = (float)g;
```

```
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("ADDRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c); wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b - f;
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("SUBRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c); wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b/f;
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("DIVRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b * f;
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("MULRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b;
g = Math.Sqrt(b);
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("SQRRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b;
float c = (float)g;
```

```
float h = Math.Abs(c);
d = BitConverter.ToString(BitConverter.
    GetBytes(h));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("ABSRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, h);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = Math.Log(b);
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("LOGRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = 1/b;
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("RECRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, c);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b;
int h = (int)g;
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("F2IRESConstant({1}) <=
    conv_std_logic_vector({2},32); --{0}", e,
    i-1, h);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
b = 2.0;
f = 1.0;
for (int i = 1; i <= a; i++)
{
g = b;
int h = (int)g;
float c = (float)g;
d = BitConverter.ToString(BitConverter.
    GetBytes(c));
e = Reverse(d);
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("I2FRESConstant({1}) <=
    conv_std_logic_vector(16#{0}#,32); --{2}",
     e, i-1, h);
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
b++;
f++;
}
Space();
Console.WriteLine("--");
try
{
```

```
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("");
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
Console.WriteLine("Press any key to exit");
Console.ReadLine();
}
private static void Space()
{
try
{
StreamWriter wr = new StreamWriter("
    fpu_package.txt", true);
wr.WriteLine("\n");
wr.Close();
}
catch (Exception y)
{
Console.WriteLine("Exception: " + y.Message);
}
}
private static string Reverse(string rev)
{
string[] splitstr = rev.Split('-');
var resources = new List<string>();
for(int i = 0; i <4; i++)
{
string part = splitstr[i];
char[] chararray = part.ToCharArray();
Array.Reverse(chararray);
resources.Add(new string(chararray));
}
string[] build = resources.ToArray();
string builder = string.Join("", build);
char[] chararray1 = builder.ToCharArray();
Array.Reverse(chararray1);
return new string(chararray1);
}
}
}
```

APPENDIX E
TCL SCRIPT

```
#-----------------------------
# Vivado v2013.4 (64-bit)
# SW Build 353583 on Mon Dec 9 17:26:26 MST
    2013
# IP Build 208076 on Mon Dec 2 12:38:17 MST
    2013
# Start of session at: Thu Jan 23 12:09:11
    2014
# Process ID: 8901
# Log file: /home/elvc/work/projects/elements/
    vivado.log
# Journal file: /home/elvc/work/projects/
    elements/vivado.jou
#-----------------------------
create_project -force fpu_wrap fpu_wrap -part
    xc7z020clg484-1
set_property board xilinx.com:zynq:zc706:1.1 [
    current_project]
set_property target_language VHDL [
    current_project]

add_files -norecurse coregen/f322i32.ngc
add_files -norecurse coregen/fabs32.ngc
add_files -norecurse coregen/faddsub32.ngc
add_files -norecurse coregen/fdiv32.ngc
add_files -norecurse coregen/fmul32.ngc
add_files -norecurse coregen/flog32.ngc
add_files -norecurse coregen/frec32.ngc
add_files -norecurse coregen/fsqrt32.ngc
add_files -norecurse coregen/i322f32.ngc

add_files -norecurse fpu_wrap/src/faddsub_wrap
    .vhd
add_files -norecurse fpu_wrap/src/faddsub32.
    vhd
add_files -norecurse fpu_wrap/src/fmul32.vhd
add_files -norecurse fpu_wrap/src/fdiv32.vhd
add_files -norecurse fpu_wrap/src/ctrl.vhd
add_files -norecurse fpu_wrap/src/fabs32.vhd
add_files -norecurse fpu_wrap/src/fsqrt32.vhd
add_files -norecurse fpu_wrap/src/flog32.vhd
add_files -norecurse fpu_wrap/src/frec32.vhd
add_files -norecurse fpu_wrap/src/f322i32.vhd
add_files -norecurse fpu_wrap/src/i322f32.vhd

set_property library {work} [get_files {
    fpu_wrap/src/faddsub_wrap.vhd fpu_wrap/src
    /faddsub32.vhd fpu_wrap/src/fmul32.vhd
    fpu_wrap/src/fdiv32.vhd fpu_wrap/src/ctrl.
    vhd fpu_wrap/src/fabs32.vhd fpu_wrap/src/
    fsqrt32.vhd fpu_wrap/src/flog32.vhd
    fpu_wrap/src/frec32.vhd fpu_wrap/src/
    f322i32.vhd fpu_wrap/src/i322f32.vhd
    coregen/f322i32.ngc coregen/fabs32.ngc
    coregen/faddsub32.ngc coregen/fdiv32.ngc
    coregen/fmul32.ngc coregen/flog32.ngc
    coregen/frec32.ngc coregen/fsqrt32.ngc
    coregen/i322f32.ngc } ]

update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
ipx::package_project -import_files -root_dir {
    fpu_wrap}
set_property vendor {vac} [ipx::current_core]
set_property library {elements} [ipx::
    current_core]
set_property vendor_display_name {vac} [ipx::
    current_core]
ipx::create_xgui_files [ipx::current_core]
ipx::save_core [ipx::current_core]
set_property ip_repo_paths fpu_wrap [
    current_fileset]
update_ip_catalog
 exit
```