Documentation on FITER accelerator

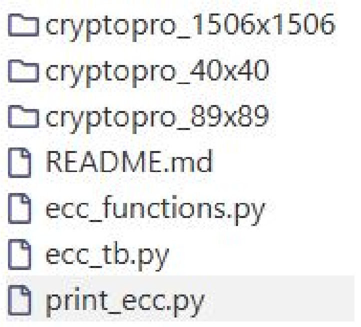
1 Introduction

We have provided the verilog code for three different prime size $m = (40, 89, 1506)$, each in a separated folder (cryptopro_mxm). Each folder is compose of a folders that has the verilog files for all the sub-modules of our accelerator for all three modular operations (addition, subtraction and reduction). On the top module, there are the files on the accelerator:

- mem_mux and mux_in are two multiplexers.
- mem_unit_m is the register bank.
- cryptoprocessor_wrapper_m is the register banks and all control signals.
- cryptoprocessor_wrapper_m is the whole accelerator.
- wrapper_m_tb is a small test-bench for whole accelerator.
- tb_dbl_m, tb_4_iso_c_m and tb_4_iso_e_m represent three test-bench for the three elliptic curves operations that are used in isogeny VDF evaluation.

The last three files print_ecc.py, ecc_tb.py and ecc_functions.py are three pythons files that are use to generate test cases and test-bench for the elliptic curves operations.

Fig. 1 shows all files/folders.



```
cryptopro_1506x1506
cryptopro_40x40
cryptopro_89x89
README.md
ecc_functions.py
ecc_tb.py
print_ecc.py
```

Figure 1: Files and folders

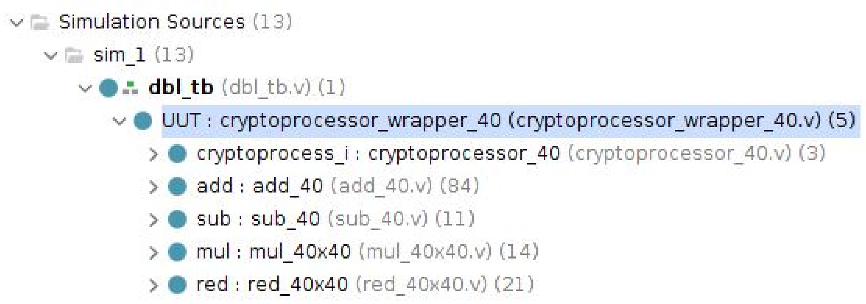
We would like to highlight that the low-bit size $m = (40, 89)$ should be used for testing as $m = 1506$ requires a very powerful system.

2 Test bench

2.1 High level function

We provide test-bench for high level function such as Point-Doubling, 4 isogeny computation and 4-isogeny evaluation in the respective files named tb_dbl.v, tb_4_iso_c.v and tb_4_iso_e. These files should be included by the whole accelerator (under the name cryptoprocessor_wrapper and all of its sub-modules), see Fig. 2 for reference.

These three test bench are generated via a python script to generate a randomised input and generates the corresponding output for validation with the output of the accelerator.



```
Simulation Sources (13)
  sim_1 (13)
    dbl_tb (dbl_tb.v) (1)
      UUT : cryptoprocessor_wrapper_40 (cryptoprocessor_wrapper_40.v) (5)
        cryptoprocess_i : cryptoprocessor_40 (cryptoprocessor_40.v) (3)
        add : add_40 (add_40.v) (84)
        sub : sub_40 (sub_40.v) (11)
        mul : mul_40x40 (mul_40x40.v) (14)
        red : red_40x40 (red_40x40.v) (21)
```

Figure 2: Files hierarchy on Vivado

2.2 Lower level functions

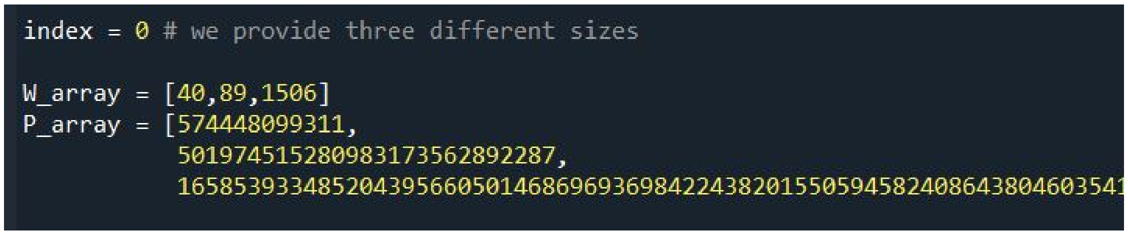
We will describe every submodules:

- AND_array_m represents an AND operation: out $\leftarrow$ ina & (inb $\times$ m). inb is one bit and ina is m-bit long.
- AND_matrix_m_m represents an AND operation between two m-bit long integer.
- add_m is the m-bit long modular addition using CSA.
- add_m_lut is an LUT table used in add_m.
- csa_m represents a carry save adder (CSA) for three m-bit inputs.
- csa_tree_x_y represents a csa tree with $x \cdot y$ long inputs appended together that are reduced to y-bits.
- csa_tree_x_y_truncated represents a special csa tree with $x \cdot y$ long inputs appended together that are reduced to y-bits.
- fa is an Full Adder.
- ha is an Half Adder.
- or_gate is an OR gate.
- red_m_lut is an LUT used in the reduction.
- red_part 1_$m \times m$ represents AND gates between a an m-bit in CS form and p_prime.
- sub_m is the m-bit long subtraction in CS form.
- sub_m_lut is an LUT used in the subtraction unit.
- trimming is one AND gate with one of the input inverted.
- xor_m_array is the XOR operation of two m-bit inputs.

The modular reduction operation is directly “instantiated” in the wrapper verilog files.

3 Example of a run

Here, we will show in detail an example.
First, open the “print_ecc.py” file.
Select which the prime size 40/89/1506-bit in the relevant section, see Fig. 3.
Run the print.py file on python3. It will produce three test bench for all three elliptic curves operations.
Run the test bench, here is an example for 40-bit for point doubling. The input point in projective



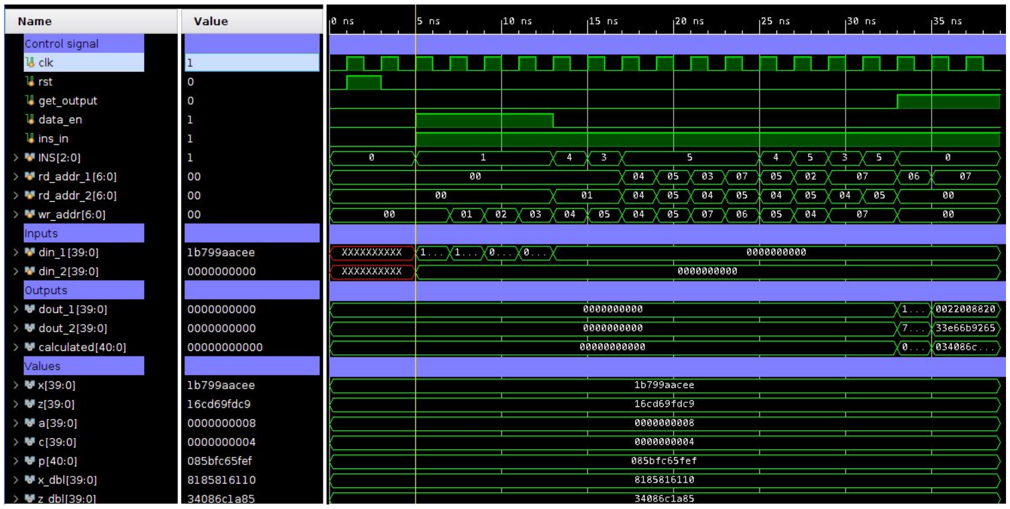
```
index = 0 # we provide three different sizes

W_array = [40,89,1506]
P_array = [574448099311,
           501974515280983173562892287,
           16585393348520439566050146869693698422438201550594582408643804603541
```

Figure 3: Code to change size

coordinate $P(X:Z) = (11086081573, 354753799392)$. The curve is the $(A, C) = (8:4)$ representing the curve with a curve parameter of 6 and a j-invariance of $j = 1728$. The correct output (line 57-58 in the test bench) should be $[2] \cdot P(X_2, Z_2) = (556290629904, 223479601797)$.

Fig. 4 represent the wave form of simulating the point doubling operation.



Name	Value
Control signal	
clk	1
rst	0
get_output	0
data_en	1
ins_in	1
INS[2:0]	1
rd_addr_1[6:0]	00
rd_addr_2[6:0]	00
wr_addr[6:0]	00
Inputs	
din_1[39:0]	1b799aacee
din_2[39:0]	0000000000
Outputs	
dout_1[39:0]	0000000000
dout_2[39:0]	0000000000
calculated[40:0]	00000000000
Values	
x[39:0]	1b799aacee
z[39:0]	16cd69fdc9
a[39:0]	0000000008
c[39:0]	0000000004
p[40:0]	085bfc65fef
x_dbl[39:0]	8185816110
z_dbl[39:0]	34086c1a85

Figure 4: Wave form for point doubling

All the different signals are classified by theirs types in Fig. 4:

- Control Signal
- Data inputs
- Data Outputs and its representation in unsigned number format
- All signals under the "value" section are all the test values and precommupted output to test our accelerator.

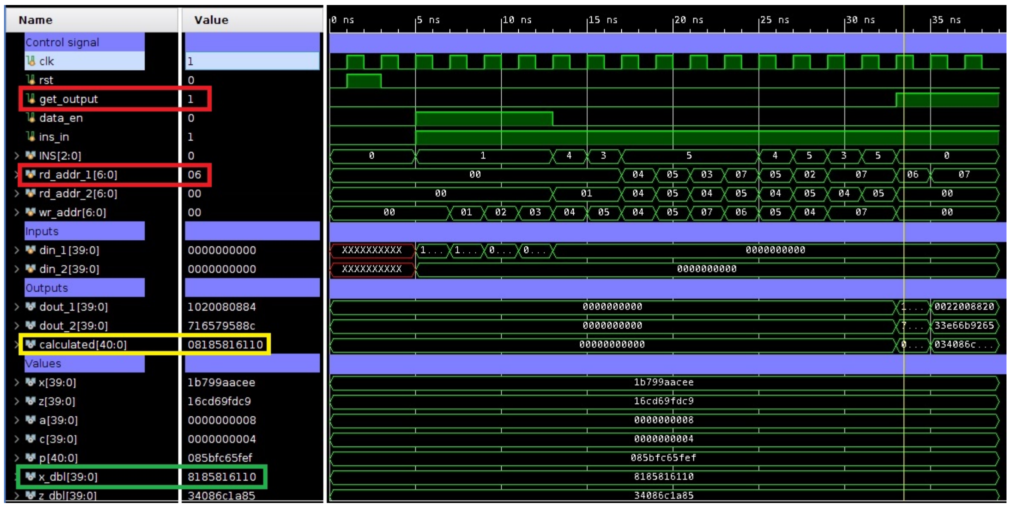
To check the results, we instructed our accelerator to output what value it has calulated. This is shown in Fig. 5, the red box present the two main control signal to check outputs. The yellow box in Fig. 5 and in Fig. 6 represent the computed output in normal representation, which is compared with the precomputed value in the green box.

If one would like to enter some specific inputs to the test, in the "print_ecc.py" file set the rand_x variable to False and manually enter the input in the in_x table, see the red boxes in Fig. 7.

4 Instructions list

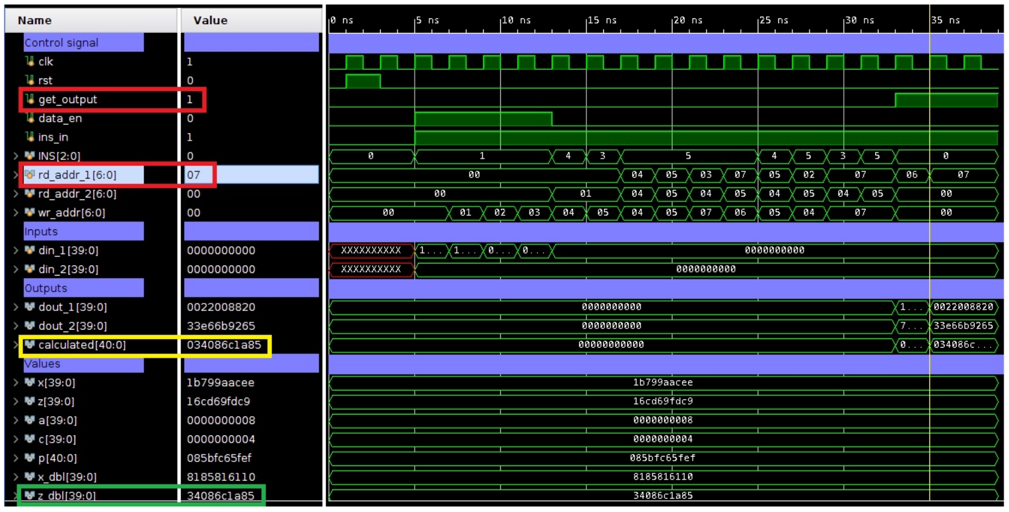
The FITHER accelerator supports eight instructions:

- Load inputs



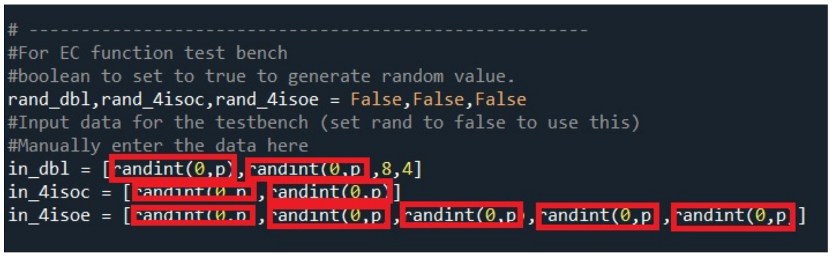
Name	Value
Control signal	
clk	1
rst	0
get_output	1
data_en	0
ins_in	1
INS[2:0]	0
rd_addr_1[6:0]	06
rd_addr_2[6:0]	00
wr_addr[6:0]	00
Inputs	
din_1[39:0]	0000000000
din_2[39:0]	0000000000
Outputs	
dout_1[39:0]	1020080884
dout_2[39:0]	716579588c
calculated[40:0]	08185816110
Values	
x[39:0]	1b799aacee
z[39:0]	16cd69fdc9
a[39:0]	0000000008
c[39:0]	0000000004
p[40:0]	085bfc65fef
x_dbl[39:0]	8185816110
z_dbl[39:0]	34086c1a85

Figure 5: Wave form for the first output



Name	Value
Control signal	
clk	1
rst	0
get_output	1
data_en	0
ins_in	1
INS[2:0]	0
rd_addr_1[6:0]	07
rd_addr_2[6:0]	00
wr_addr[6:0]	00
Inputs	
din_1[39:0]	0000000000
din_2[39:0]	0000000000
Outputs	
dout_1[39:0]	0022008820
dout_2[39:0]	33e66b9265
calculated[40:0]	034086c1a85
Values	
x[39:0]	1b799aacee
z[39:0]	16cd69fdc9
a[39:0]	0000000008
c[39:0]	0000000004
p[40:0]	085bfc65fef
x_dbl[39:0]	8185816110
z_dbl[39:0]	34086c1a85

Figure 6: Wave form for the second output



```
# -------------------------------------------------------
#For EC function test bench
#boolean to set to true to generate random value.
rand_dbl,rand_4isoc,rand_4isoe = False,False,False
#Input data for the testbench (set rand to false to use this)
#Manually enter the data here
in_dbl = [randint(0,p),randint(0,p),8,4]
in_4isoc = [randint(0,p),randint(0,p)]
in_4isoe = [randint(0,p),randint(0,p),randint(0,p),randint(0,p),randint(0,p)]
```

Figure 7: How to manually enter data

- Get ouput
- Copy
- Modular Addition
- Modular subtraction
- Modular Multiplication

- Idle accelerator
- Reset

Most of these instruction work by sending an instruction to the accelerator via the command_in signal (the three most significant bits). The list is here:

- INS = 0 => Idle
- INS = 1 => Load input
- INS = 2 => Copy point rd to wr
- INS = 3 => Mod. ADD
- INS = 4 => Mod. SUB
- INS = 5 => Mod. MUL

But to both load some data into FITHER and get some output, we must enable some control signal.

To load data into the cryptoprocessor, one must both set data_en to 1 and set INS to 1 while din_1 and din_2 to the wanted input.

The instruction to make the accelerator out some data is to set the control signal get_out to one, rd_addr_1 to which address we want. The rest should be set to 0.

To reset the accelerator, the rst signal should be set to 1 for one cc.