Nota 11: Digital emulation of quantum circuits, TR2

Technical report 2, June 2024

In a preceding technical report [1], a specific digital circuit, able to emulate the execution of quantum algorithms, has been specified. This report is a continuation of [1]. It defines a behavioral VHDL model of the circuit (`nota11_1.vhd`).

1. Representation of complex numbers

The same representation as in [2] is used. Complex numbers $x = a+bi$ are represented by pairs of fixed-point 2's complement integers:

$$a = a_{t-1}\, a_{t-2}\, .\, a_{t-3}\, a_{t-4} \ldots a_0,\ b = b_{t-1}\, b_{t-2}\, .\, b_{t-3}\, b_{t-4} \ldots b_0,$$

being a_{t-1} and b_{t-1} the sign bits. Taking into account that the module $|x|$ of all the processed numbers x is smaller than or equal to 1, the coefficients a and b belong to the interval

$$-1 \leq a, b \leq 1,$$

and must be represented with two integer bits.

Define $A = a \cdot 2^{t-2}$ and $B = b \cdot 2^{t-2}$. They belong to the interval

$$-2^{t-2} \leq A, B \leq 2^{t-2},$$

and their product $P = A \cdot B$ belongs to the interval

$$-2^{2\cdot(t-2)} \leq P \leq 2^{2\cdot(t-2)},$$

so that, in 2's complement,

$$P = P_{2\cdot(t-2)+1}\, P_{2\cdot(t-2)} \cdots P_0.$$

For example,

$$1100\cdots0 = -2^{2\cdot(t-2)+1} + 2^{2\cdot(t-2)} = -2^{2\cdot(t-2)},\ 0100\cdots0 = 2^{2\cdot(t-2)}.$$

Thus, the product $a \cdot b$ is equal to

$$a \cdot b = A \cdot B \cdot 2^{-2\cdot(t-2)} = P = P_{2\cdot(t-2)+1}\, P_{2\cdot(t-2)}\, .\, P_{2\cdot(t-2)-1} \cdots P_0$$

$$\cong P_{2\cdot(t-2)+1}\, P_{2\cdot(t-2)}\, .\, P_{2\cdot(t-2)-1} \cdots P_{t-2}\,,$$

that is, a *t*-bit fixed-point number.

The following VHDL entity (`nota11_1.vhd`) computes $a \cdot b$:

```
entity fp_mul is
   port(
      a, b: in std_logic_vector(t-1 downto 0);
      c: out std_logic_vector(t-1 downto 0)
      );
end fp_mul;
architecture behavior of fp_mul is
   signal long_a, long_b: std_logic_vector(2*t-1 downto 0);
   signal long_c: std_logic_vector(4*t-1 downto 0);
begin
   long_a(2*t-1 downto t)<= (others => a(t-1));
   long_a(t-1 downto 0)<= a;
   long_b(2*t-1 downto t)<= (others => b(t-1));
   long_b(t-1 downto 0)<= b;
   long_c <=  long_a*long_b;
   c <= long_c(2*t-3 downto t-2);
end behavior;
```

2. Quantum register

Consider a quantum register made up of n qubits $q_0, q_1, \ldots, q_{n-1}$. Its state

$$|q_0\rangle \times |q_1\rangle \times \ldots \times |q_{n-2}\rangle \times |q_{n-1}\rangle$$

is defined by an expression such as

$$a_0|00\ldots00\rangle + a_1|00\ldots01\rangle + a_2|00\ldots10\rangle + a_{N-1}|11\ldots11\rangle,$$

where $N = 2^n$ and the coefficients a_k are complex numbers. The physical meaning of this expression is the following: when measuring the register state, the result will be 00…00 with a probability equal to $|a_0|^2$, 00…01 with a probability equal to $|a_1|^2$, and so on. This is one of the enormous differences between digital and quantum circuits: the state of an n-bit register is a dimension-n vector over the binary field, while the state of an n-qubit register is a dimension-2^n vector over the complex field C.

The quantum register is implemented by a dual-port memory that stores 2^n complex numbers. Using the representation of Sec.1, the memory stores 2^n $2t$-bit words. The use of two ports permits to execute in one clock period a unary operation on a single qubit (Sec.3).

The following VHDL entity (`nota11_1.vhd`) defines a dual-port memory. The write operation is synchronized by a clock signal, while the read operation is asynchronous. The model includes two register banks, one for the real part and the other for the imaginary part of the addressed coefficient. There are two address decoders, one for each port (A and B).

```
entity dual_port_ram is
port (
  data_in_A_re,data_in_A_im,data_in_B_re,data_in_B_im:
    in std_logic_vector(t-1 downto 0);
  clk, writeA, writeB: in std_logic := '0';
  address_A,address_B : in std_logic_vector(n-1 downto 0);
  data_out_A_re,data_out_A_im,data_out_B_re,data_out_B_im:
    out std_logic_vector(t-1 downto 0)
);
end dual_port_ram;

architecture behavior of dual_port_ram is
  type memory is array (0 to d-1) of std_logic_vector(t-1 downto 0);
  signal X_re, X_im: memory;
  signal EN_A, EN_B: std_logic_vector(0 to d-1);
begin
  decoder_A: process(address_A, writeA)
  begin
    for i in 0 to d-1 loop
      if i = conv_integer(address_A) then EN_A(i) <= writeA;
      else EN_A(i) <= '0';
      end if;
    end loop;
  end process;
  decoder_B: process(address_B, writeB)
  begin
    for i in 0 to d-1 loop
      if i = conv_integer(address_B) then EN_B(i) <= writeB;
      else EN_B(i) <= '0';
      end if;
    end loop;
  end process;
 register_bank: process(clk)
 begin
     if clk'event and clk = '1' then
       for i in 0 to d-1 loop
         if EN_A(i) = '1' then
           X_re(i) <= data_in_A_re; X_im(i) <= data_in_A_im; end if;
         if EN_B(i) = '1' then
           X_re(i) <= data_in_B_re; X_im(i) <= data_in_B_im; end if;
       end loop;
     end if;
  end process;
  multiplexer: process(address_A, address_B, X_re, X_im)
  begin
```

```
      for i in 0 to d-1 loop
         if i = conv_integer(address_A) then
            data_out_A_re <= X_re(i);data_out_A_im <= X_im(i); end if;
         if i = conv_integer(address_B) then
            data_out_B_re <= X_re(i);data_out_B_im <= X_im(i); end if;
      end loop;
    end process;
end behavior;
```

3. Quantum operations

According to Sec.2 of [1], the execution of a unary operation U, defined by four complex coefficients u_{00}, u_{01}, u_{10}, u_{11}, actuating on a single qubit, possibly under the control of another qubit, consists of a set of pairs of arithmetic operations:

a_i <= $u_{00} \cdot a_i + u_{01} \cdot a_{m+i}$,

a_{m+i} <= $u_{10} \cdot a_{\mathrm{i}} + u_{11} \cdot a_{m+i}$,

where a_i and a_{m+i} are also complex numbers and $m \neq 0$. The use of a dual-port memory permits to execute this pair of operations in a single clock period.

The following entity (`nota11_1.vhd`) defines an arithmetic unit that executes those two operations, using fixed-point multipliers (Sec.1):

```
entity ar_unit is
port (
  inA_re,inA_im,inB_re,inB_im: in std_logic_vector(t-1 downto 0);
  output01: in std_logic;
  outA_re,outA_im,outB_re,outB_im:     out std_logic_vector(t-1 downto 0);
  a_re,a_im,b_re,b_im, c_re,c_im,d_re,d_im:
    in std_logic_vector(t-1 downto 0)
);
end ar_unit;

architecture circuit of ar_unit is
  component fp_mul is
  port (
  a, b: in std_logic_vector(t-1 downto 0);
  c: out std_logic_vector(t-1 downto 0)
  );
  end component;
  signal
    areAre,aimAim,breBre,bimBim,aimAre,areAim,bimBre,breBim,
    creAre,cimAim,dreBre,dimBim,cimAre,creAim,dimBre,dreBim:
    std_logic_vector(t-1 downto 0);: std_logic_vector(t-1 downto 0);
begin
  mul1: fp_mul port map(a_re, inA_re, areAre);
  mul2: fp_mul port map(a_im, inA_im, aimAim);
  mul3: fp_mul port map(b_re, inB_re, breBre);
  mul4: fp_mul port map(b_im, inB_im, bimBim);
  mul5: fp_mul port map(a_im, inA_re, aimAre);
  mul6: fp_mul port map(a_re, inA_im, areAim);
  mul7: fp_mul port map(b_im, inB_re, bimBre);
  mul8: fp_mul port map(b_re, inB_im, breBim);
  mul9: fp_mul port map(c_re, inA_re, creAre);
  mul10: fp_mul port map(c_im, inA_im, cimAim);
  mul11: fp_mul port map(d_re, inB_re, dreBre);
  mul12: fp_mul port map(d_im, inB_im, dimBim);
  mul13: fp_mul port map(c_im, inA_re, cimAre);
  mul14: fp_mul port map(c_re, inA_im, creAim);
  mul15: fp_mul port map(d_im, inB_re, dimBre);
  mul16: fp_mul port map(d_re, inB_im, dreBim);

process(output01,areAre,aimAim,breBre,bimBim,aimAre,areAim,bimBre,breBim,
  creAre,cimAim,dreBre,dimBim,cimAre,creAim,dimBre,dreBim)
  begin
    if output01 = '1' then
    outA_re <= zero; outA_im <= zero; outB_re <= one; outB_im <= zero;
    else
    outA_re <= areAre - aimAim + breBre - bimBim;
    outA_im <= aimAre + areAim + bimBre + breBim;
    outB_re <= creAre - cimAim + dreBre - dimBim;
    outB_im <= cimAre + creAim + dimBre + dreBim;
    end if;
  end process;
end circuit;
```

An additional binary control signal *output*01 generates the constant values 0+0i and 1+0i at outputs A and B, respectively. Those values are used to set the initial quantum state to

$$|00...00\rangle = (1+0i)|00...00\rangle + (0+0i)|00...01\rangle + ... + (0+0i)|11...11\rangle.$$

4. Circuit structure

The proposed emulator structure is shown in Fig.2 of [1]. The main circuit parameters are the following:

- t is the number of bits of a fixed-point number;
- n is the number of qubits;
- p is the number of bits of the program memory addresses, chosen in such a way that 2^p is greater than or equal to the number of instructions;
- *logn* is a natural greater than or equal to $log_2 n$.

Those parameters are defined within the package `qubits`.

The processor must be able to control the execution of four types of instructions.

- *INIT*: set the quantum state to $|00...0\rangle$. For that, all dual port ram memory words are set to $0+0i$, excepted the first one, set to $1+0i$.
- $U(k)$: execute a unitary operation on the qubit number k, where k belongs to the set $\{0, 1, 2, \ldots, n\text{-}1\}$.
- $CU(l,k)$: execute a unitary operation on the qubit number k, under the control of the qubit number $l \neq k$, where k and l belong to the set $\{0, 1, 2, \ldots, n\text{-}1\}$.
- *END*: execute the identity operation on the quantum state.

Furthermore, in the case of *INIT*, $U(k)$ and $CU(l,k)$ instructions, the program memory address is incremented. The unitary operations U are defined by the corresponding complex coefficients u_{00}, u_{01}, u_{10}, u_{11}. The instruction format is shown in Fig.1, in the particular case where n = 5, t = 10 and *logn* = 3. The first two bits encode the four instruction types *INIT*, *U*, *CU*, *END*.

87..86	85..83	82..80	79..70	69..60	59..50	49..40	39..30	29..20	19..10	9..0
type	l	k	u_{00}_re	u_{00}_im	u_{01}_re	u_{01}_im	u_{10}_re	u_{10}_im	u_{11}_re	u_{11}_im

Figure 1 Instruction format

With this instruction format, the block diagram of Fig.2 is defined. The processor generates the dual-port RAM addresses and the write commands, as well as the *output*01 signal. The u_{ij} coefficients, that define a particular unitary operation, are directly transmitted to the arithmetic unit.

The following entity defines the processor behavior. It mainly consists of a process `processor` that translates to VHDL the functions `U()` and

`CU()` defined in [1], Sec.2.1 and 2.2, taking into account that the coefficient u_{ij} are immediate values included in the instruction (Fig.1). The *opcode* input includes the instruction type (first 2 bits) and the values of *l* and *k* (next 2·*logn* bits).

```
entity processor is
port (
  clk, reset:in std_logic;
  opcode: in std_logic_vector(cs-1 downto 0);
  writeA, writeB,output01: out std_logic;
  address_A,address_B : out std_logic_vector(n-1 downto 0);
  address_M: inout std_logic_vector(p-1 downto 0)
);
end processor;

architecture behavior of processor is
begin
  processor: process
   variable s, k, l, m, q: integer;
   variable addr: std_logic_vector(n-1 downto 0);
  begin
    address_M <= first;
    wait until clk'event and clk = '1';
    loop
      case opcode(cs-1 downto cs-2) is
        when "00" =>
          output01 <= '1';
          address_A <= addr_zero;
          address_B <= addr_zero;
          writeA <= '0';
          writeB <= '1';
          wait until clk'event and clk = '1';
          for i in 1 to d-1 loop
              address_A <= conv_std_logic_vector(i, n);
              writeA <= '1';
              writeB <= '0';
              wait until clk'event and clk = '1';
          end loop;
          writeA <= '0';
          writeB <= '0';
          output01 <= '0';
          address_M <= address_M + 1;
        when "01" =>
          k := conv_integer(opcode(logn-1 downto 0));
          m := 2**(n-k-1);
          q := 2**k;
          for j in 0 to q-1 loop
            s := 2*m*j;
            for i in s to s+m-1 loop
              address_A <= conv_std_logic_vector(i, n);
              address_B <= conv_std_logic_vector(i+m, n);
              writeA <= '1';
              writeB <= '1';
              wait until clk'event and clk = '1';
            end loop;
           end loop;
           writeA <= '0';
           writeB <= '0';
          address_M <= address_M + 1;
        when "11" =>
          writeA <= '0';
          writeB <= '0';
          output01 <= '0';
        when "10" =>
          k := conv_integer(opcode(logn-1 downto 0));
          l := conv_integer(opcode(cs-3 downto logn));
          m := 2**(n-k-1);
          q := 2**k;
          for j in 0 to q-1 loop
            s := 2*m*j;
            for i in s to s+m-1 loop
              addr := conv_std_logic_vector(i, n);
              address_A <= addr;
              address_B <= conv_std_logic_vector(i+m, n);
              if addr(n-1-l) = '1' then
                writeA <= '1';
                writeB <= '1';
              else
                writeA <= '0';
                writeB <= '0';
              end if;
              wait until clk'event and clk = '1';
            end loop;
           end loop;
           writeA <= '0';
           writeB <= '0';
          address_M <= address_M + 1;
        when others =>
          writeA <= '0';
          writeB <= '0';
          output01 <= '0';
         end case;
       wait until clk'event and clk = '1';
      end loop;
    end process;
end behavior;
```

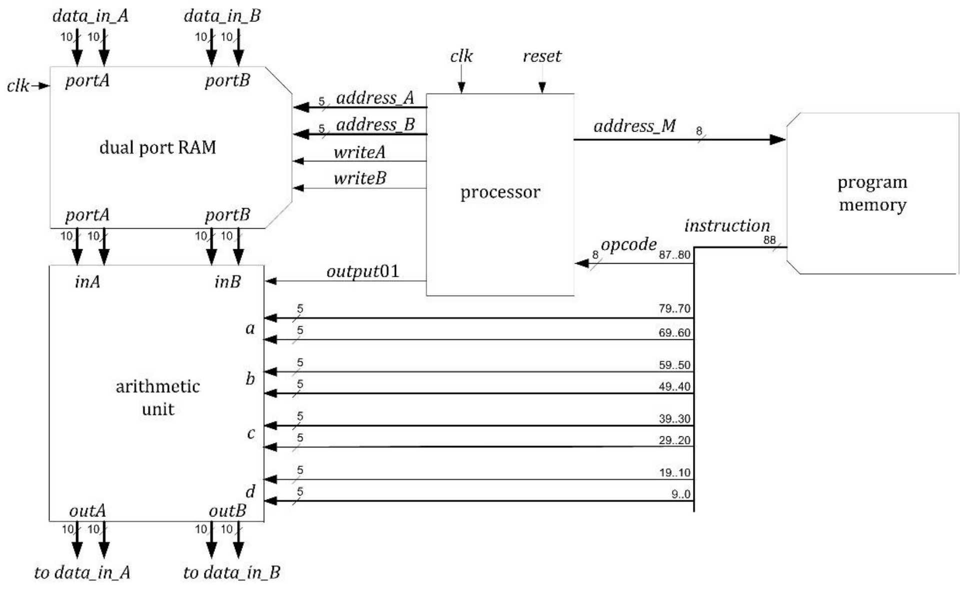


Figure 2 Block diagram

5. Example

The program memory contents are defined in order to execute the quantum Fourier transform on qubits number 1 to 4 of a 5-qubit register, with initial state equal to $|00101\rangle$. This is the same example as in [1], Sec.2.4, without the final permutation. The parameter values are $n = 5$, $t = 10$, $logn = 3$, $p = 8$, so that the instruction size is equal to $ws = 2 + 2 \cdot 3 + 8 \cdot 10 = 88$.

The numbers belonging to the interval [-1,1] are represented as follows:

- if $x \geq 0$, compute $y = round(x \cdot 2^{t-2})$ and express y in hexadecimal,
- if $x < 0$, compute $y = round((4-x) \cdot 2^{t-2})$ and express y in hexadecimal,

being *round* a rounding function. For example,

if $x = 1/2^{0.5} = 1.707\ldots$ and $t = 10$, then $round(x \cdot 2^8) = 181 = 0BA$,

if $x = -1/2^{0.5}$ and $t = 10$, then $round((4-x) \cdot 2^8) = 843 = 34B$.

```
entity program_memory is
port (
  address_M: in std_logic_vector(p-1 downto 0);
  instruction: out std_logic_vector(ws-1 downto 0)
);
end program_memory;

architecture behavior of program_memory is
  type format is record
    op: std_logic_vector(1 downto 0);
    l: std_logic_vector(logn-1 downto 0);
    k: std_logic_vector(logn-1 downto 0);
    ar: std_logic_vector(t-1 downto 0);
    ai: std_logic_vector(t-1 downto 0);
    br: std_logic_vector(t-1 downto 0);
    bi: std_logic_vector(t-1 downto 0);
    cr: std_logic_vector(t-1 downto 0);
    ci: std_logic_vector(t-1 downto 0);
    dr: std_logic_vector(t-1 downto 0);
    di: std_logic_vector(t-1 downto 0);
  end record;
  type memory is array (0 to 2**p-1) of format;
  signal contents: memory := (
  ("00", "000", "000","00"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"00","00"&x"00"), -- init
  ("01", "000", "001","00"&x"00","00"&x"00","01"&x"00","00"&x"00",
   "01"&x"00","00"&x"00","00"&x"00","00"&x"00"), -- X(1)
  ("01", "000", "011","00"&x"00","00"&x"00","01"&x"00","00"&x"00",
   "01"&x"00","00"&x"00","00"&x"00","00"&x"00"), -- X(3)
  ("01", "000", "000","00"&x"b5","00"&x"00","00"&x"b5","00"&x"00",
   "00"&x"b5","00"&x"00","11"&x"4b","00"&x"00"), -- H(0)
  ("10", "001", "000","01"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"00","01"&x"00"), -- CRphi(1,0, pi/2)
  ("10", "010", "000","01"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"b5","00"&x"b5"), -- CRphi(2,0, pi/4)
  ("10", "011", "000","01"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"ec","00"&x"62"), -- CRphi(3,0, pi/8)
  ("01", "000", "001","00"&x"b5","00"&x"00","00"&x"b5","00"&x"00",
   "00"&x"b5","00"&x"00","11"&x"4b","00"&x"00"), -- H(1)
  ("10", "010", "001","01"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"00","01"&x"00"), -- CRphi(2,1, pi/2)
  ("10", "011", "001","01"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"b5","00"&x"b5"), -- CRphi(3,1, pi/4)
  ("01", "000", "010","00"&x"b5","00"&x"00","00"&x"b5","00"&x"00",
   "00"&x"b5","00"&x"00","11"&x"4b","00"&x"00"), -- H(2)
  ("10", "011", "010","01"&x"00","00"&x"00","00"&x"00","00"&x"00",
   "00"&x"00","00"&x"00","00"&x"00","01"&x"00"), -- CRphi(3,2, pi/2)
  ("01", "000", "011","00"&x"b5","00"&x"00","00"&x"b5","00"&x"00",
   "00"&x"b5","00"&x"00","11"&x"4b","00"&x"00"), -- H(3)
  others => ("11","000","000","00"&x"00","00"&x"00","00"&x"00","00"&x"00",
    "00"&x"00","00"&x"00","00"&x"00","00"&x"00") -- end
  );
begin
   rom: process(address_M)
     variable i: natural;
     variable form_instruction: format;
   begin
     i := conv_integer(address_M);
     form_instruction := contents(i);
     instruction <= form_instruction.op & form_instruction.l
      &  form_instruction.k & form_instruction.ar & form_instruction.ai
      & form_instruction.br & form_instruction.bi & form_instruction.cr
      & form_instruction.ci & form_instruction.dr & form_instruction.di;
   end process;
end behavior;
```

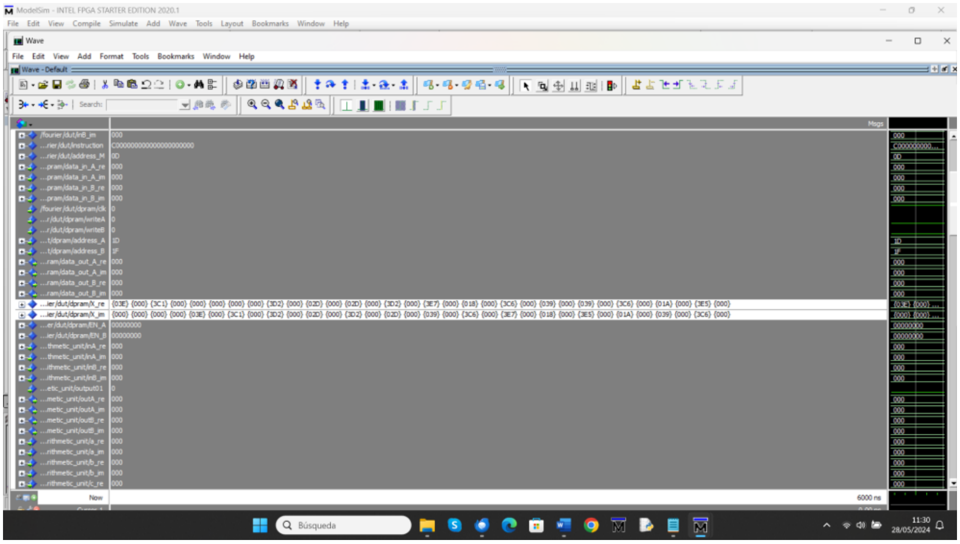
The simulation result is shown in Fig.3. The final state is

```
0.2421875 |00000⟩
+ (- 0.24609375) |00010⟩
+ (0.2421875i) |00100⟩
+ (0.24609375i) |00110⟩
+ (- 0.1796875 - 0.1796875i) |01000⟩
+ (0.17578125 + 0.17578125i) |01010⟩
+ (0.17578125 - 0.1796875i)  |01100⟩
+ (- 0.1796875 + 0.17578125 i) |01110⟩
+ (- 0.09765625 + 0.22265625i) |10000⟩
+ (0.09375 − 0.2265625i) |10010⟩
+ (- 0.2265625 - 0.09765625i) |10100⟩
+ (0.22265625 + 0.09765625i) |10110⟩
+ (0.22265625 − 0.10546875i) |11000⟩
+ (-0.2265625 + 0.1015625i) |11010⟩
+ (0.1015625 + 0.22265625i) |11100⟩
+ (- 0.10546875 − 2265625i) |11110⟩
```

Equivalently (by permuting the rows), the final state is

```
0.2421875 |00000⟩
+ (- 0.09765625 + 0.22265625i) |10000⟩
+ (- 0.1796875 - 0.1796875i) |01000⟩
+ (0.22265625 − 0.10546875i) |11000⟩
+ (0.2421875i) |00100⟩
+ (- 0.2265625 - 0.09765625i) |10100⟩
+ (0.17578125 - 0.1796875i)  |01100⟩
+ (0.1015625 + 0.22265625i) |11100⟩
+ (- 0.24609375) |00010⟩
+ (0.09375 − 0.2265625i) |10010⟩
+ (0.17578125 + 0.17578125i) |01010⟩
+ (-0.2265625 + 0.1015625i) |11010⟩
+ (- 0.24609375i) |00110⟩
+ (0.22265625 + 0.09765625i) |10110⟩
+ (- 0.1796875 + 0.17578125 i) |01110⟩
+ (- 0.10546875 − 2265625i) |11110⟩
```

Apart from small inaccuracy errors – the computations are performed with only eight fractional bits – the results are the same as in [1], Sec.2.4.



```
/fourier/dut/inB_im            000
...rier/dut/instruction        C0000000000000000000
...rier/dut/address_M          0D
...pram/data_in_A_re           000
...pram/data_in_A_im           000
...pram/data_in_B_re           000
...pram/data_in_B_im           000
/fourier/dut/dpram/clk         0
...r/dut/dpram/writeA          0
...r/dut/dpram/writeB          0
...t/dpram/address_A           1D
...t/dpram/address_B           1F
...ram/data_out_A_re           000
...ram/data_out_A_im           000
...ram/data_out_B_re           000
...ram/data_out_B_im           000
...ier/dut/dpram/X_re          {03E} {000} {3C1} {000} {000} {000} {000} {000} {3D2} {000} {02D} {000} {02D} {000} {3D2} {000} {3E7} {000} {018} {000} {3C6} {000} {039} {000} {039} {000} {3C6} {000} {01A} {000} {3E5} {000}
...ier/dut/dpram/X_im          {000} {000} {000} {000} {03E} {000} {3C1} {000} {3D2} {000} {02D} {000} {3D2} {000} {02D} {000} {039} {000} {3C6} {000} {3E7} {000} {018} {000} {3E5} {000} {01A} {000} {039} {000} {3C6} {000}
...er/dut/dpram/EN_A           00000000
...ier/dut/dpram/EN_B          00000000
...thmetic_unit/inA_re         000
...thmetic_unit/inA_im         000
...ithmetic_unit/inB_re        000
...ithmetic_unit/inB_im        000
...etic_unit/output01          0
...metic_unit/outA_re          000
...metic_unit/outA_im          000
...metic_unit/outB_re          000
...metic_unit/outB_im          000
...rithmetic_unit/a_re         000
...rithmetic_unit/a_im         000
...rithmetic_unit/b_re         000
...rithmetic_unit/b_im         000
...rithmetic_unit/c_re         000
Now                            6000 ns
```

Figure 3 Fourier transform simulation

References

[1] Nota 10, Digital emulation of quantum circuits, TR 1.

[2] Nota 9, Digital emulation of 3-qubit quantum circuits.