

64点高速フーリエ変換回路

～ Level1 ～

金城佑典（チーム：）

所属：琉球大学工学部情報工学科学部 2 年次

金城佑典（学籍番号：055717A）j05017@ie.u-ryukyu.ac.jp

T シャツ希望サイズ：L

池野谷（学籍番号：）j05002@ie.u-ryukyu.ac.jp

T シャツ希望サイズ：L

目 次

1 構成と動作

1.1 FFT64

$$X(k) = \sum_{n=0}^{63} x(n)e^{-j\left(\frac{2\pi}{64}\right)nk} = \sum_{n=0}^{63} x(n)W_{64}^{nk} \quad (k = 0, 1, 2, \dots, 63)$$

の演算を行う

FFT64は64点高速フーリエ変換を行う回路で、複素数の演算を行う「Stage1」「Stage2」「Stage3」と演算中に入れ代わってしまった順番をもとに戻す「REORDER」から構成されている。

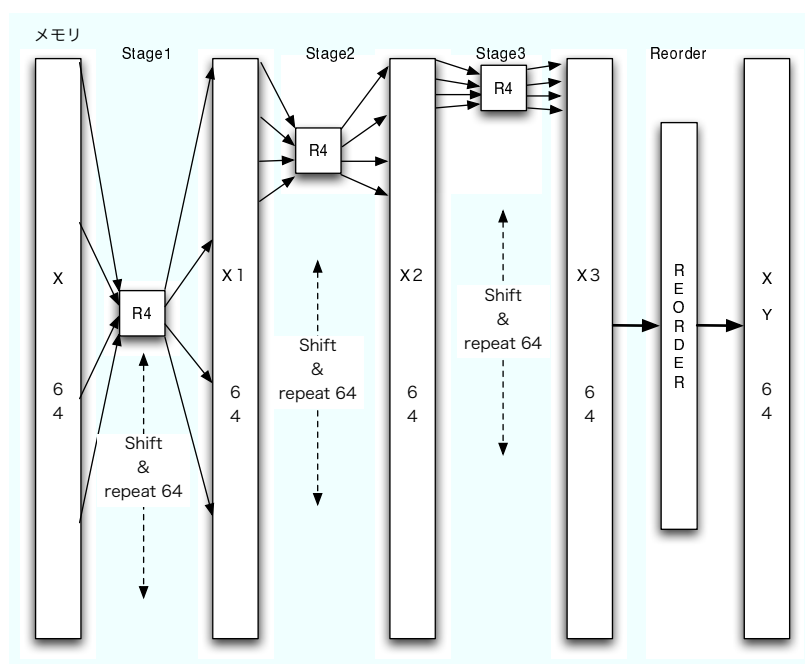


図 1: FFT64 のブロック図

「Stage1」「Stage2」「Stage3」は図のようにそれぞれ4つの複素数から4つの演算結果を出すことができるが、上から順番に計算していきたいので shift 動作を繰り返すことで0～63まで合計64点の演算を行い、「REORDER」によって順序を入れ替えた結果を出力として返す。

1.2 TWIDDLE

Twiddle の生成を行う回路、Stage1, Stage2, Stage3 で使用する

$$W_{64} = e^{-j\frac{2\pi}{64}}$$

の演算を行うが、ここで

$$e^{j\theta} = \cos\theta + j\sin\theta$$

より

$$W_{64} = e^{-j\frac{2\pi}{64}} = \cos\frac{2\pi}{64} + j\sin\frac{2\pi}{64}$$

なので、実際にはあらかじめ計算しておいた \cos の数値のリストから要求されたものを出力するだけの回路になっている (\sin の数値は \cos の数値から得られる)

1.3 複素乗算器 (Cmult と multar)

複素数の乗算を行う回路、Stage1,Stage2,Stage3 で使用する

$$(AIN_I + jAIN_Q) * (BIN_I + jBIN_Q) \\ = \{(AIN_I * BIN_I) - (AIN_Q * BIN_Q)\} + j\{(AIN_Q * BIN_I) + (AIN_I * BIN_Q)\}$$

の計算を行う

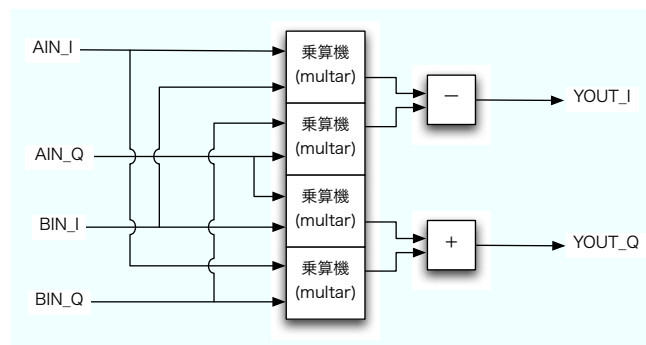


図 2: Cmult のブロック図

図中の乗算機は multar(multar.vhd) で実装されている。

1.4 Stage1

x 信号を用いて

$$x1(n_0 + 4n_1 + 16k_0) = \begin{pmatrix} x(n_0 + 4n_1) \\ +(-j)^{k_1} x(n_0 + 4n_1 + 16) \\ +(-1)^{k_1} x(n_0 + 4n_1 + 32) \\ +(j)^{k_1} x(n_0 + 4n_1 + 48) \end{pmatrix} W_{64}^{k_0(n_0+4n_1)}$$

の演算を受け持つ回路

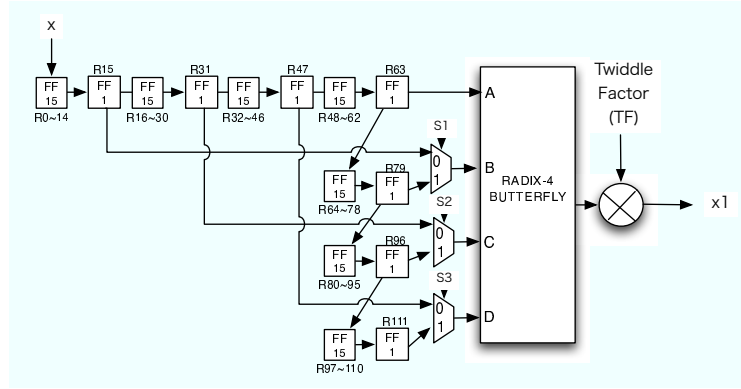


図 3: Stage1 のブロック図

1.5 Stage2

x1 信号を用いて

$$x2(n_0 + 4k_1 + 16k_0) = \begin{pmatrix} x1(n_0 + 0 + 16k_0) \\ +(-j)^{k_1} x1(n_0 + 4 + 16k_0) \\ +(-1)^{k_1} x1(n_0 + 8 + 16k_0) \\ +(j)^{k_1} x1(n_0 + 12 + 16k_0) \end{pmatrix} W_{16}^{(k_1 n_0)}$$

の演算を行う回路

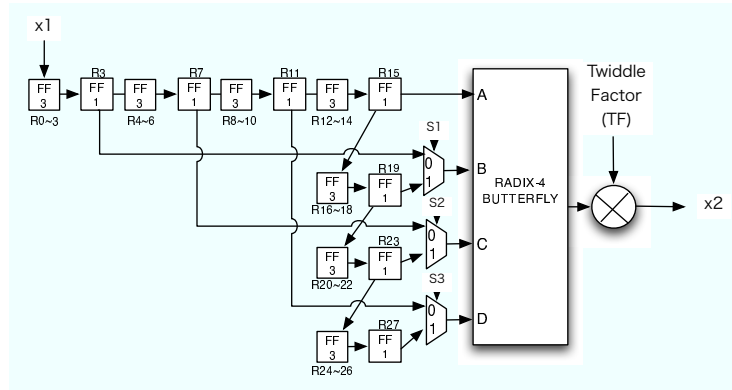


図 4: Stage2 のブロック図

1.6 Stage3

x2 信号を用いて

$$\begin{aligned}
 x3(k_2 + 4k_1 + 16k_0) &= \sum_{n_0=0}^3 x2(n_0 + 4k_1 + 16k_0) W_4^{n_0 k_2} \\
 &= \begin{pmatrix} x2(0 + 4k_1 + 16k_0) \\ +x2(1 + 4k_1 + 16k_0) W_4^{k_2} \\ +x2(2 + 4k_1 + 16k_0) W_4^{2k_2} \\ +x2(3 + 4k_1 + 16k_0) W_4^{3k_2} \end{pmatrix} \\
 &= \begin{pmatrix} x2(0 + 4k_1 + 16k_0) \\ +(-j)^{k_2} x2(1 + 4k_1 + 16k_0) \\ +(-1)^{k_1} x2(2 + 4k_1 + 16k_0) \\ +(j)^{k_2} x2(3 + 4k_1 + 16k_0) \end{pmatrix}
 \end{aligned}$$

の演算を行う回路

論理合成後の回路規模

$Totalcellarea : 95521.000000$ なので $95521/3 = 31840.3333$ より回路面積は $31840.34UNIT_AREA$

2.2 動作確認

2.2.1 1) ALL1 入力

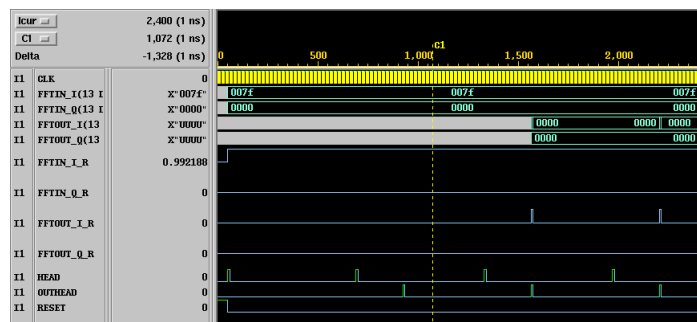


図 8: ALL1 入力の動作波形

2.2.2 2) 1 周期する複素回転信号 初期値=1+0j

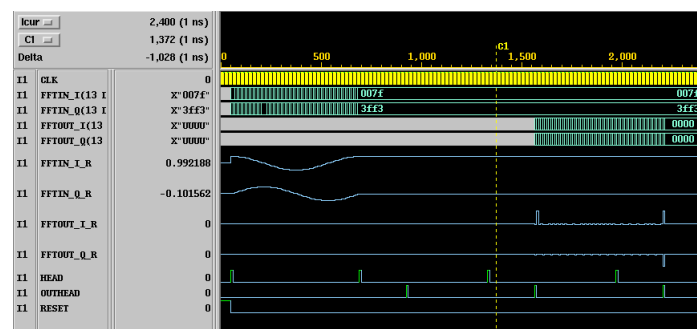


図 9: 1 周期する複素回転信号 初期値=1+0j の動作波形

2.2.3 3) 1 周期する複素回転信号 初期値=0-1j

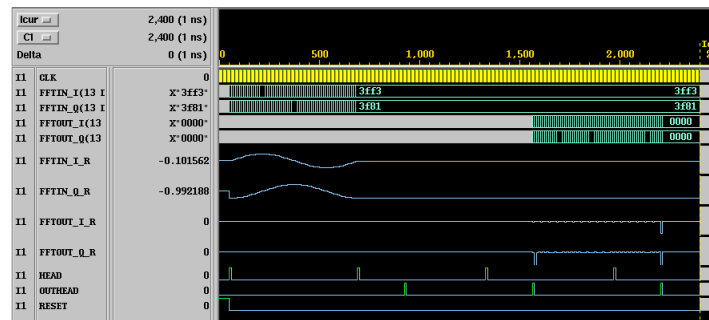


図 10: 1 周期する複素回転信号 初期値=0-1j の動作波形

2.2.4 4) 1 周期する COS 信号

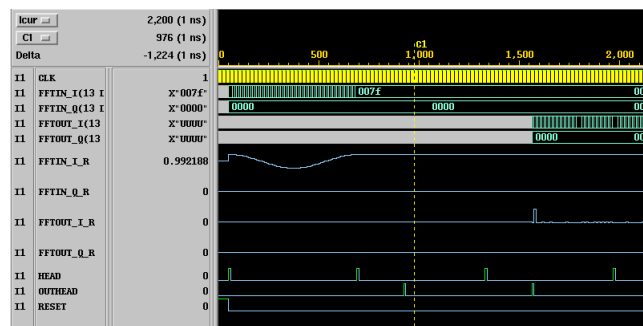


図 11: 1 周期する COS 信号の動作波形

2.2.5 5) 15 周期する複素回転信号

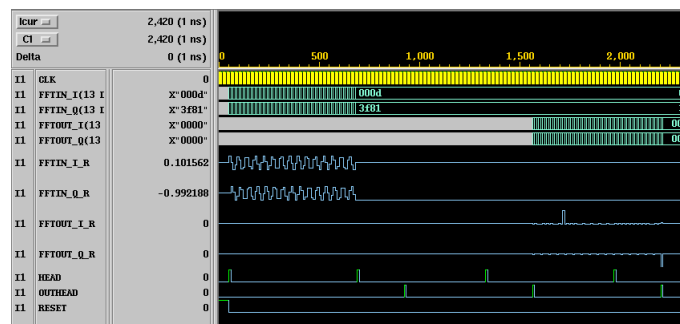


図 12: 15 周期する複素回転信号の動作波形

3 工夫した点

stage1.vhd と stage2.vhd において、PHASE GENERATOR と TWID-DLE FACTOR GENERATOR を if 文や case 文などを用いずに、64 CYCLE COUNTER の信号 (COUNT) 等を用いて、一行で表すように工夫した。

4 自由意見

5 VHDL のコード (全文)

5.1 fft64.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity FFT64 is
  port (
    RESET      : in  std_logic;
    CLK        : in  std_logic;
    HEAD       : in  std_logic;           -- <1,1,u>
    FFTIN_I    : in  std_logic_vector(13 downto 0); -- <14,6,t>
    FFTIN_Q    : in  std_logic_vector(13 downto 0); -- <14,6,t>
    OUTHEAD    : out std_logic;           -- <1,1,u>
    FFTOUT_I   : out std_logic_vector(13 downto 0); -- <14,6,t>
    FFTOUT_Q   : out std_logic_vector(13 downto 0)); -- <14,6,t>
end;

architecture RTL of FFT64 is

  component STAGE1 is
    port (
      RESET      : in  std_logic;
      CLK        : in  std_logic;
      S1HEAD     : in  std_logic;           -- <1,1,u>
      S1IN_I     : in  std_logic_vector(13 downto 0); -- <14,6,t>
      S1IN_Q     : in  std_logic_vector(13 downto 0); -- <14,6,t>
      S1OUTHEAD   : out std_logic;           -- <1,1,u>
      S1OUT_I    : out std_logic_vector(13 downto 0); -- <14,6,t>
      S1OUT_Q    : out std_logic_vector(13 downto 0)); -- <14,6,t>
  end component;

  component STAGE2
  port (
    RESET      : in  std_logic;
    CLK        : in  std_logic;
    S2HEAD     : in  std_logic;           -- <1,1,u>
    S2IN_I     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S2IN_Q     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S2OUTHEAD   : out std_logic;           -- <1,1,u>
    S2OUT_I    : out std_logic_vector(13 downto 0); -- <14,6,t>
    S2OUT_Q    : out std_logic_vector(13 downto 0)); -- <14,6,t>
  end component;

  component STAGE3
  port (
    RESET      : in  std_logic;
    CLK        : in  std_logic;
    S3HEAD     : in  std_logic;           -- <1,1,u>
    S3IN_I     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S3IN_Q     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S3OUTHEAD   : out std_logic;           -- <1,1,u>
    S3OUT_I    : out std_logic_vector(13 downto 0); -- <14,6,t>
    S3OUT_Q    : out std_logic_vector(13 downto 0)); -- <14,6,t>
  end component;

  component REORDER
  port (
    RESET      : in  std_logic;
```

```

        CLK      : in  std_logic;
        ROHEAD   : in  std_logic; -- <1,1,u>
        ROIN_I   : in  std_logic_vector(13 downto 0); -- <14,6,t>
        ROIN_Q   : in  std_logic_vector(13 downto 0); -- <14,6,t>
        ROOUTHEAD : out std_logic; -- <1,1,u>
        ROOUT_I  : out std_logic_vector(13 downto 0); -- <14,6,t>
        ROOUT_Q  : out std_logic_vector(13 downto 0)); -- <14,6,t>
end component;

signal S1HEAD, S1OUTHEAD : std_logic;
signal S2HEAD, S2OUTHEAD : std_logic;
signal S3HEAD, S3OUTHEAD : std_logic;
signal ROHEAD, ROOUTHEAD : std_logic;
signal S1IN_I, S1IN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal S1OUT_I, S1OUT_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal S2IN_I, S2IN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal S2OUT_I, S2OUT_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal S3IN_I, S3IN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal S3OUT_I, S3OUT_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal ROIN_I, ROIN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal ROOUT_I, ROOUT_Q : std_logic_vector (13 downto 0); -- <14,6,t>

begin

S1HEAD <= HEAD; S1IN_I <= FFTIN_I; S1IN_Q <= FFTIN_Q;

ST1: STAGE1 port map (RESET, CLK, S1HEAD, S1IN_I, S1IN_Q, S1OUTHEAD, S1OUT_I, S1OUT_Q);

S2HEAD <= S1OUTHEAD; S2IN_I <= S1OUT_I; S2IN_Q <= S1OUT_Q;

ST2: STAGE2 port map (RESET, CLK, S2HEAD, S2IN_I, S2IN_Q, S2OUTHEAD, S2OUT_I, S2OUT_Q);

S3HEAD <= S2OUTHEAD; S3IN_I <= S2OUT_I; S3IN_Q <= S2OUT_Q;

ST3: STAGE3 port map (RESET, CLK, S3HEAD, S3IN_I, S3IN_Q, S3OUTHEAD, S3OUT_I, S3OUT_Q);

ROHEAD <= S3OUTHEAD; ROIN_I <= S3OUT_I; ROIN_Q <= S3OUT_Q;

R00: REORDER port map (RESET, CLK, ROHEAD, ROIN_I, ROIN_Q, ROOUTHEAD, ROOUT_I, ROOUT_Q);

OUTHEAD <= ROOUTHEAD; FFTOUT_I <= ROOUT_I; FFTOUT_Q <= ROOUT_Q;

end;

```

5.2 twiddle.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TWIDDLE is
    port (
        ADDR : in  std_logic_vector(5 downto 0); -- <6,6,u>
        W_I  : out std_logic_vector(9 downto 0); -- <10,0,t>
        W_Q  : out std_logic_vector(9 downto 0)); -- <10,0,t>
end;

architecture RTL of TWIDDLE is
begin
    process (ADDR)

```

```

variable iaddr : integer;
variable qaddr : integer;
type rom_type is array(0 to 63) of std_logic_vector(9 downto 0);
constant cos_table : rom_type := (
"011111111", "011111110", "011110110", "011101010",
"011101001", "0111000100", "0110101010", "0110001100",
"0101101010", "0101000101", "0100011100", "0011110001",
"0011000100", "0010010101", "0001100100", "0000110010",
"0000000000", "1111001110", "1110011100", "1101101011",
"1100111100", "1100001111", "1011100100", "1010111011",
"1010010110", "1001110100", "1001010110", "1000111100",
"1000100111", "1000010110", "1000001010", "1000000010",
"1000000000", "1000000010", "1000001010", "1000010110",
"1000100111", "1000111100", "1001010110", "1001110100",
"1010010110", "1010111011", "1011100100", "1100001111",
"1100111100", "1101101011", "1110011100", "1111001110",
"0000000000", "0000110010", "0001100100", "0010010101",
"0011000100", "0011110001", "0100011100", "0101000101",
"0101101010", "0110001100", "0110101010", "0111000100",
"0111011001", "0111101010", "0111110110", "0111111110"
);

begin
iaddr := conv_integer(unsigned(ADDR));
qaddr := conv_integer(unsigned(ADDR)+16);
W_I <= cos_table(iaddr);
W_Q <= cos_table(qaddr);
end process;

end;

```

5.3 cmult.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity CMULT is
port(AIN_I : in std_logic_vector(13 downto 0); -- <14,6,t>
AIN_Q : in std_logic_vector(13 downto 0); -- <14,6,t>
BIN_I : in std_logic_vector(9 downto 0); -- <10,0,t>
BIN_Q : in std_logic_vector(9 downto 0); -- <10,0,t>
YOUT_I : out std_logic_vector(13 downto 0); -- <14,6,t>
YOUT_Q : out std_logic_vector(13 downto 0)); -- <14,6,t>
end CMULT;

architecture RTL of CMULT is

-- component declaration
component MULTAR
port(in1 : in std_logic_vector(13 downto 0); -- <14,6,t>
in2 : in std_logic_vector(9 downto 0); -- <10,0,t>
outp : out std_logic_vector(13 downto 0)); -- <14,6,t>
end component;

signal S1, S2, S3, S4 : std_logic_vector(13 downto 0); -- <14,6,t>

begin

M1: MULTAR port map (AIN_I, BIN_I, S1);

```

```

M2: MULTAR port map (AIN_Q, BIN_Q, S2);
M3: MULTAR port map (AIN_Q, BIN_I, S3);
M4: MULTAR port map (AIN_I, BIN_Q, S4);

YOUT_I <= signed(S1) - signed(S2);
YOUT_Q <= signed(S3) + signed(S4);

end RTL;

```

5.4 multar.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity MULTAR is
    port(in1      : in  std_logic_vector(13 downto 0); -- <14,6,t>
          in2      : in  std_logic_vector(9  downto 0); -- <10,0,t>
          outp     : out std_logic_vector(13 downto 0)); -- <14,6,t>
end MULTAR;

architecture RTL of MULTAR is

begin

    MULTAR : process(in1,in2)
        variable var_tprod : std_logic_vector(23 downto 0); -- <24,7,t>
        variable var_round : std_logic_vector(13 downto 0); -- <14,6,t>
    begin
        var_tprod := signed(in1) * signed(in2);

        if (var_tprod(8) = '1') then
            var_round := signed(var_tprod(22 downto 9)) + '1';
        else
            var_round := var_tprod(22 downto 9);
        end if;

        outp <= var_round;

    end process MULTAR;

end RTL;

```

5.5 stage1.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity STAGE1 is
    port (
        RESET      : in  std_logic;
        CLK        : in  std_logic;
        S1HEAD      : in  std_logic;                -- <1,1,u>
        S1IN_I      : in  std_logic_vector(13 downto 0); -- <14,6,t>
        S1IN_Q      : in  std_logic_vector(13 downto 0); -- <14,6,t>
        S1OUTHEAD   : out std_logic;                -- <1,1,u>
        S1OUT_I     : out std_logic_vector(13 downto 0); -- <14,6,t>
    );
end STAGE1;

```

```

        S1OUT_Q : out std_logic_vector(13 downto 0)); -- <14,6,t>
end;

architecture RTL of STAGE1 is

component CMULT is
    port(AIN_I      : in  std_logic_vector(13 downto 0); -- <14,6,t>
          AIN_Q      : in  std_logic_vector(13 downto 0); -- <14,6,t>
          BIN_I      : in  std_logic_vector(9  downto 0); -- <10,0,t>
          BIN_Q      : in  std_logic_vector(9  downto 0); -- <10,0,t>
          YOUT_I     : out std_logic_vector(13 downto 0); -- <14,6,t>
          YOUT_Q     : out std_logic_vector(13 downto 0)); -- <14,6,t>
end component;

component TWIDDLE
    port (
        ADDR : in  std_logic_vector(5 downto 0); -- <6,6,u>
        W_I  : out std_logic_vector(9  downto 0); -- <10,0,t>
        W_Q  : out std_logic_vector(9  downto 0)); -- <10,0,t>
end component;

signal HEAD : std_logic_vector(64 downto 0); -- 65 FFs for Head Signal Delay
signal COUNT : std_logic_vector(5 downto 0); -- count 0 to 63 cycles
signal PHASE : std_logic_vector(1 downto 0); -- PHASE 0 to 3
signal TF : std_logic_vector(5 downto 0); -- TWIDDLE FACTOR INDEX
signal S1, S2, S3 : std_logic;

signal REGIN_I : std_logic_vector (13 downto 0); -- <14,6,t>
signal REGIN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
type shiftreg112 is array (0 to 111) of std_logic_vector (13 downto 0); -- <14,6,t>
signal REG_I : shiftreg112;
signal REG_Q : shiftreg112;
signal A_I, A_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal B_I, B_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal C_I, C_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal D_I, D_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal Y_I, Y_Q : std_logic_vector (13 downto 0); -- <14,6,t> butterfly output
signal W_I, W_Q : std_logic_vector(9  downto 0); -- <10,0,t>

begin

-- OUTHEAD GENERATION
process ( CLK, RESET ) begin
    if (RESET = '1') then
        for I in 0 to 64 loop
            HEAD(I) <= '0';
        end loop;
    elsif ( CLK'event and CLK = '1' ) then
        HEAD(0) <= S1HEAD;
        for I in 1 to 64 loop
            HEAD(I) <= HEAD(I-1);
        end loop;
    end if;
end process;
S1OUTHEAD <= HEAD(64);

-- 64 CYCLE COUNTER
process ( CLK, RESET ) begin
    if (RESET = '1') then
        COUNT <= "000000";
    elsif ( CLK'event and CLK = '1' ) then
        if (S1HEAD = '1') then

```

```

        COUNT <= "000000";
    else
        COUNT <= unsigned(COUNT) + '1';
    end if;
end if;
end process;

-- PHASE GENERATOR
PHASE <= COUNT(5 downto 4); --henkou

-- TWIDDLE FACTOR GENERATOR
TF <= unsigned(COUNT(3 downto 0)) * unsigned(PHASE(1 downto 0)); --henkou

-- SELECT SIGNAL GENERATOR
S1 <= PHASE(0) or PHASE(1);
S2 <= PHASE(1);
S3 <= PHASE(0) and PHASE(1);

-- SHIFT REGISTER
process ( CLK ) begin
    if ( CLK'event and CLK = '1' ) then
        REGIN_I <= S1IN_I;    REGIN_Q <= S1IN_Q;
        REG_I(0) <= REGIN_I;  REG_Q(0) <= REGIN_Q;
        for I in 1 to 111 loop
            REG_I(I) <= REG_I(I-1); REG_Q(I) <= REG_Q(I-1);
        end loop;
    end if;
end process;

-- A,B,C,D GENERATION
A_I <= REG_I(63);
A_Q <= REG_Q(63);
B_I <= REG_I(15) when S1 = '0' else REG_I(79);
B_Q <= REG_Q(15) when S1 = '0' else REG_Q(79);
C_I <= REG_I(31) when S2 = '0' else REG_I(95);
C_Q <= REG_Q(31) when S2 = '0' else REG_Q(95);
D_I <= REG_I(47) when S3 = '0' else REG_I(111);
D_Q <= REG_Q(47) when S3 = '0' else REG_Q(111);

-- RADIX-4 BUTTERFLY
process (PHASE, A_I, B_I, C_I, D_I, A_Q, B_Q, C_Q, D_Q) begin
    case PHASE is
        when "00" =>
            Y_I <= signed(A_I) + signed(B_I) + signed(C_I) + signed (D_I);
            Y_Q <= signed(A_Q) + signed(B_Q) + signed(C_Q) + signed (D_Q);
        when "01" =>
            Y_I <= signed(A_Q) + signed(B_I) - signed(C_Q) - signed (D_I);
            Y_Q <= - signed(A_I) + signed(B_Q) + signed(C_I) - signed (D_Q);
        when "10" =>
            Y_I <= signed(A_I) - signed(B_I) + signed(C_I) - signed (D_I);
            Y_Q <= signed(A_Q) - signed(B_Q) + signed(C_Q) - signed (D_Q);
        when "11" =>
            Y_I <= signed(A_Q) - signed(B_I) - signed(C_Q) + signed (D_I);
            Y_Q <= - signed(A_I) - signed(B_Q) + signed(C_I) + signed (D_Q);
        when others =>
            Y_I <= (others => 'X');
            Y_Q <= (others => 'X');
    end case;
end process;

TWO: TWIDDLE port map (TF, W_I, W_Q);

```



```
CM0: CMULT port map (Y_I, Y_Q, W_I, W_Q, S1OUT_I, S1OUT_Q);
```

```
end;
```

5.6 stage2.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
entity STAGE2 is
  port (
    RESET      : in  std_logic;
    CLK        : in  std_logic;
    S2HEAD     : in  std_logic;           -- <1,1,u>
    S2IN_I     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S2IN_Q     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S2OUTHEAD  : out std_logic;           -- <1,1,u>
    S2OUT_I    : out std_logic_vector(13 downto 0); -- <14,6,t>
    S2OUT_Q    : out std_logic_vector(13 downto 0)); -- <14,6,t>
end;
```

```
architecture RTL of STAGE2 is
```

```
  component CMULT is
    port(AIN_I      : in  std_logic_vector(13 downto 0); -- <14,6,t>
         AIN_Q      : in  std_logic_vector(13 downto 0); -- <14,6,t>
         BIN_I      : in  std_logic_vector(9  downto 0); -- <10,0,t>
         BIN_Q      : in  std_logic_vector(9  downto 0); -- <10,0,t>
         YOUT_I     : out std_logic_vector(13 downto 0); -- <14,6,t>
         YOUT_Q     : out std_logic_vector(13 downto 0)); -- <14,6,t>
  end component;
```

```
  component TWIDDLE
    port (
      ADDR : in  std_logic_vector(5 downto 0); -- <6,6,u>
      W_I  : out std_logic_vector(9  downto 0); -- <10,0,t>
      W_Q  : out std_logic_vector(9  downto 0)); -- <10,0,t>
  end component;
```

```
  signal HEAD : std_logic_vector(16 downto 0); -- 17 FFs for Head Signal Delay
  signal COUNT : std_logic_vector(5 downto 0); -- count 0 to 63 cycles
  signal PHASE : std_logic_vector(1 downto 0); -- PHASE 0 to 3
  signal TF    : std_logic_vector(5 downto 0); -- TWIDDLE FACTOR INDEX
  signal S1, S2, S3 : std_logic;
```

```
  signal REGIN_I : std_logic_vector (13 downto 0); -- <14,6,t>
  signal REGIN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  type shiftreg28 is array (0 to 27) of std_logic_vector (13 downto 0); -- <14,6,t>
  signal REG_I   : shiftreg28;
  signal REG_Q   : shiftreg28;
  signal A_I, A_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  signal B_I, B_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  signal C_I, C_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  signal D_I, D_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  signal Y_I, Y_Q : std_logic_vector (13 downto 0); -- <14,6,t>  butterfly output
  signal W_I, W_Q : std_logic_vector(9  downto 0); -- <10,0,t>
```

```
begin
```

```

-- OUTHEAD GENERATION
process ( CLK, RESET ) begin
    if (RESET = '1') then
        for I in 0 to 16 loop
            HEAD(I) <= '0';
        end loop;
    elsif ( CLK'event and CLK = '1' ) then
        HEAD(0) <= S2HEAD;
        for I in 1 to 16 loop
            HEAD(I) <= HEAD(I-1);
        end loop;
    end if;
end process;
S2OUTHEAD <= HEAD(16);

-- 64 CYCLE COUNTER
process ( CLK, RESET ) begin
    if (RESET = '1') then
        COUNT <= "000000";
    elsif ( CLK'event and CLK = '1' ) then
        if (S2HEAD = '1') then
            COUNT <= "000000";
        else
            COUNT <= unsigned(COUNT) + '1';
        end if;
    end if;
end process;

-- PHASE GENERATOR
PHASE <= COUNT(3 downto 2); --henkou

-- TWIDDLE FACTOR GENERATOR
TF <= unsigned(COUNT(1 downto 0)) * (unsigned(PHASE(1 downto 0)) & "00"); --henkou

-- SELECT SIGNAL GENERATOR
S1 <= PHASE(0) or PHASE(1);
S2 <= PHASE(1);
S3 <= PHASE(0) and PHASE(1);

-- SHIFT REGISTER
process ( CLK ) begin
    if ( CLK'event and CLK = '1' ) then
        REGIN_I <= S2IN_I;    REGIN_Q <= S2IN_Q;
        REG_I(0) <= REGIN_I;  REG_Q(0) <= REGIN_Q;
        for I in 1 to 27 loop
            REG_I(I) <= REG_I(I-1);  REG_Q(I) <= REG_Q(I-1);
        end loop;
    end if;
end process;

-- A,B,C,D GENERATION
A_I <= REG_I(15);
A_Q <= REG_Q(15);
B_I <= REG_I(3) when S1 = '0' else REG_I(19);
B_Q <= REG_Q(3) when S1 = '0' else REG_Q(19);
C_I <= REG_I(7) when S2 = '0' else REG_I(23);
C_Q <= REG_Q(7) when S2 = '0' else REG_Q(23);
D_I <= REG_I(11) when S3 = '0' else REG_I(27);
D_Q <= REG_Q(11) when S3 = '0' else REG_Q(27);

```

```

-- RADIX-4 BUTTERFLY

process (PHASE, A_I, B_I, C_I, D_I, A_Q, B_Q, C_Q, D_Q) begin
  case PHASE is
    when "00" =>
      Y_I <= signed(A_I) + signed(B_I) + signed(C_I) + signed (D_I);
      Y_Q <= signed(A_Q) + signed(B_Q) + signed(C_Q) + signed (D_Q);
    when "01" =>
      Y_I <= signed(A_Q) + signed(B_I) - signed(C_Q) - signed (D_I);
      Y_Q <= - signed(A_I) + signed(B_Q) + signed(C_I) - signed (D_Q);
    when "10" =>
      Y_I <= signed(A_I) - signed(B_I) + signed(C_I) - signed (D_I);
      Y_Q <= signed(A_Q) - signed(B_Q) + signed(C_Q) - signed (D_Q);
    when "11" =>
      Y_I <= signed(A_Q) - signed(B_I) - signed(C_Q) + signed (D_I);
      Y_Q <= - signed(A_I) - signed(B_Q) + signed(C_I) + signed (D_Q);
    when others =>
      Y_I <= (others => 'X');
      Y_Q <= (others => 'X');
  end case;
end process;

TWO: TWIDDLE port map (TF, W_I, W_Q);

CM0: CMULT port map (Y_I, Y_Q, W_I, W_Q, S2OUT_I, S2OUT_Q);

end;

```

5.7 stage3.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity STAGE3 is
  port (
    RESET      : in  std_logic;
    CLK        : in  std_logic;
    S3HEAD     : in  std_logic;                -- <1,1,u>
    S3IN_I     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S3IN_Q     : in  std_logic_vector(13 downto 0); -- <14,6,t>
    S3OUTHEAD  : out std_logic;                -- <1,1,u>
    S3OUT_I    : out std_logic_vector(13 downto 0); -- <14,6,t>
    S3OUT_Q    : out std_logic_vector(13 downto 0)); -- <14,6,t>
end;

architecture RTL of STAGE3 is

  signal HEAD : std_logic_vector(4 downto 0); -- 5 FFs for Head Signal Delay
  signal COUNT : std_logic_vector(5 downto 0); -- count 0 to 63 cycles
  signal PHASE : std_logic_vector(1 downto 0); -- PHASE 0 to 3
  signal S1, S2, S3 : std_logic;

  signal REGIN_I : std_logic_vector (13 downto 0); -- <14,6,t>
  signal REGIN_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  type shiftreg7 is array (0 to 6) of std_logic_vector (13 downto 0); -- <14,6,t>
  signal REG_I : shiftreg7;
  signal REG_Q : shiftreg7;
  signal A_I, A_Q : std_logic_vector (13 downto 0); -- <14,6,t>
  signal B_I, B_Q : std_logic_vector (13 downto 0); -- <14,6,t>

```

```

signal C_I, C_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal D_I, D_Q : std_logic_vector (13 downto 0); -- <14,6,t>
signal Y_I, Y_Q : std_logic_vector (13 downto 0); -- <14,6,t>  butterfly output

begin

-- OUTHEAD GENERATION
process ( CLK, RESET ) begin
    if (RESET = '1') then
        HEAD(0) <= '0';
        HEAD(1) <= '0';
        HEAD(2) <= '0';
        HEAD(3) <= '0';
        HEAD(4) <= '0';
    elsif ( CLK'event and CLK = '1' ) then
        HEAD(0) <= S3HEAD;
        HEAD(1) <= HEAD(0);
        HEAD(2) <= HEAD(1);
        HEAD(3) <= HEAD(2);
        HEAD(4) <= HEAD(3);
    end if;
end process;
S3OUTHEAD <= HEAD(4);

-- 64 CYCLE COUNTER
process ( CLK, RESET ) begin
    if (RESET = '1') then
        COUNT <= "000000";
    elsif ( CLK'event and CLK = '1' ) then
        if (S3HEAD = '1') then
            COUNT <= "000000";
        else
            COUNT <= unsigned(COUNT) + '1';
        end if;
    end if;
end process;

-- PHASE GENERATOR
PHASE <= COUNT(1 downto 0);

-- SELECT SIGNAL GENERATOR
S1 <= PHASE(0) or PHASE(1);
S2 <= PHASE(1);
S3 <= PHASE(0) and PHASE(1);

-- SHIFT REGISTER
process ( CLK ) begin
    if ( CLK'event and CLK = '1' ) then
        REGIN_I <= S3IN_I;  REGIN_Q <= S3IN_Q;
        REG_I(0) <= REGIN_I;  REG_Q(0) <= REGIN_Q;
        REG_I(1) <= REG_I(0);  REG_Q(1) <= REG_Q(0);
        REG_I(2) <= REG_I(1);  REG_Q(2) <= REG_Q(1);
        REG_I(3) <= REG_I(2);  REG_Q(3) <= REG_Q(2);
        REG_I(4) <= REG_I(3);  REG_Q(4) <= REG_Q(3);
        REG_I(5) <= REG_I(4);  REG_Q(5) <= REG_Q(4);
        REG_I(6) <= REG_I(5);  REG_Q(6) <= REG_Q(5);
    end if;
end process;

-- A,B,C,D GENERATION
A_I <= REG_I(3);
A_Q <= REG_Q(3);

```

```

        B_I <= REG_I(0) when S1 = '0' else REG_I(4);
        B_Q <= REG_Q(0) when S1 = '0' else REG_Q(4);
        C_I <= REG_I(1) when S2 = '0' else REG_I(5);
        C_Q <= REG_Q(1) when S2 = '0' else REG_Q(5);
        D_I <= REG_I(2) when S3 = '0' else REG_I(6);
        D_Q <= REG_Q(2) when S3 = '0' else REG_Q(6);

-- RADIX-4 BUTTERFLY

process (PHASE, A_I, B_I, C_I, D_I, A_Q, B_Q, C_Q, D_Q) begin
    case PHASE is
        when "00" =>
            Y_I <= signed(A_I) + signed(B_I) + signed(C_I) + signed (D_I);
            Y_Q <= signed(A_Q) + signed(B_Q) + signed(C_Q) + signed (D_Q);
        when "01" =>
            Y_I <= signed(A_Q) + signed(B_I) - signed(C_Q) - signed (D_I);
            Y_Q <= - signed(A_I) + signed(B_Q) + signed(C_I) - signed (D_Q);
        when "10" =>
            Y_I <= signed(A_I) - signed(B_I) + signed(C_I) - signed (D_I);
            Y_Q <= signed(A_Q) - signed(B_Q) + signed(C_Q) - signed (D_Q);
        when "11" =>
            Y_I <= signed(A_Q) - signed(B_I) - signed(C_Q) + signed (D_I);
            Y_Q <= - signed(A_I) - signed(B_Q) + signed(C_I) + signed (D_Q);
        when others =>
            Y_I <= (others => 'X');
            Y_Q <= (others => 'X');
    end case;
end process;
S3OUT_I <= Y_I;
S3OUT_Q <= Y_Q;

end;

```

5.8 reorder.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity REORDER is
    port (
        RESET      : in  std_logic;
        CLK         : in  std_logic;
        ROHEAD      : in  std_logic;                -- <1,1,u>
        ROIN_I      : in  std_logic_vector(13 downto 0); -- <14,6,t>
        ROIN_Q      : in  std_logic_vector(13 downto 0); -- <14,6,t>
        ROOUTHEAD    : out std_logic;                -- <1,1,u>
        ROOUT_I     : out std_logic_vector(13 downto 0); -- <14,6,t>
        ROOUT_Q     : out std_logic_vector(13 downto 0)); -- <14,6,t>
end;

architecture RTL of REORDER is

    signal COUNT      : std_logic_vector(5 downto 0);
    signal BANK       : std_logic;                -- Input bank select signal
    signal BANKEN     : std_logic;
    type shiftreg64 is array (0 to 63) of std_logic_vector (13 downto 0); -- <14,6,t>
    signal REG0_I, REG0_Q : shiftreg64;
    signal REG1_I, REG1_Q : shiftreg64;
    signal REGIN_I, REGIN_Q : std_logic_vector (13 downto 0); -- <14,6,t>

```

```

begin

-- 64 CYCLE COUNTER
process ( CLK ) begin
    if (RESET = '1') then
        COUNT <="000000";
        BANKEN <= '0';
    elsif ( CLK'event and CLK = '1' ) then
        if (ROHEAD = '1') then
            COUNT <= "000000";
            BANKEN <= '1';
        elsif (COUNT = "111111") then
            BANKEN <= '0';
        else
            COUNT <= unsigned(COUNT) + '1';
        end if;
    end if;
end process;

-- BANK SELECT GENERATION
process ( CLK ) begin
    if (RESET = '1') then
        BANK <= '0';
    elsif ( CLK'event and CLK = '1' ) then
        if (ROHEAD = '1') then
            BANK <= not BANK;
        end if;
    end if;
end process;

-- INPUT REGISTER
process (CLK) begin
    if (CLK'event and CLK = '1' ) then
        REGIN_I <= ROIN_I;
        REGIN_Q <= ROIN_Q;
    end if;
end process;

-- SHIFT REGISTER
process (CLK) begin
    if (CLK'event and CLK = '1' ) then
        if (BANKEN = '1')then
            if (BANK = '0') then
                --
                -- BANK 0 SERIAL INPUT
                --
                for I in 0 to 62 loop
                    REGO_I(I) <= REGO_I(I+1);
                    REGO_Q(I) <= REGO_Q(I+1);
                end loop;
                REGO_I(63) <= REGIN_I;
                REGO_Q(63) <= REGIN_Q;
                --
                -- BANK 1 REORDER OUPUT
                --
                for I in 0 to 3 loop
                    REG1_I(I+ 0) <= REG1_I(I+16);
                    REG1_I(I+16) <= REG1_I(I+32);
                    REG1_I(I+32) <= REG1_I(I+48);
                    REG1_I(I+48) <= REG1_I(I+ 4);
                    REG1_I(I+ 4) <= REG1_I(I+20);
                end loop;
            end if;
        end if;
    end if;
end process;

```

```

REG1_I(I+20) <= REG1_I(I+36);
REG1_I(I+36) <= REG1_I(I+52);
REG1_I(I+52) <= REG1_I(I+ 8);
REG1_I(I+ 8) <= REG1_I(I+24);
REG1_I(I+24) <= REG1_I(I+40);
REG1_I(I+40) <= REG1_I(I+56);
REG1_I(I+56) <= REG1_I(I+12);
REG1_I(I+12) <= REG1_I(I+28);
REG1_I(I+28) <= REG1_I(I+44);
REG1_I(I+44) <= REG1_I(I+60);
REG1_Q(I+ 0) <= REG1_Q(I+16);
REG1_Q(I+16) <= REG1_Q(I+32);
REG1_Q(I+32) <= REG1_Q(I+48);
REG1_Q(I+48) <= REG1_Q(I+ 4);
REG1_Q(I+ 4) <= REG1_Q(I+20);
REG1_Q(I+20) <= REG1_Q(I+36);
REG1_Q(I+36) <= REG1_Q(I+52);
REG1_Q(I+52) <= REG1_Q(I+ 8);
REG1_Q(I+ 8) <= REG1_Q(I+24);
REG1_Q(I+24) <= REG1_Q(I+40);
REG1_Q(I+40) <= REG1_Q(I+56);
REG1_Q(I+56) <= REG1_Q(I+12);
REG1_Q(I+12) <= REG1_Q(I+28);
REG1_Q(I+28) <= REG1_Q(I+44);
REG1_Q(I+44) <= REG1_Q(I+60);
end loop;
REG1_I(60) <= REG1_I(1);
REG1_I(61) <= REG1_I(2);
REG1_I(62) <= REG1_I(3);
REG1_Q(60) <= REG1_Q(1);
REG1_Q(61) <= REG1_Q(2);
REG1_Q(62) <= REG1_Q(3);
else
--
-- BANK 1 SERIAL INPUT
--
for I in 0 to 62 loop
    REG1_I(I) <= REG1_I(I+1);
    REG1_Q(I) <= REG1_Q(I+1);
end loop;
REG1_I(63) <= REG1_I;
REG1_Q(63) <= REG1_Q;
--
-- BANK 0 REORDER OUPUT
--
for I in 0 to 3 loop
    REG0_I(I+ 0) <= REG0_I(I+16);
    REG0_I(I+16) <= REG0_I(I+32);
    REG0_I(I+32) <= REG0_I(I+48);
    REG0_I(I+48) <= REG0_I(I+ 4);
    REG0_I(I+ 4) <= REG0_I(I+20);
    REG0_I(I+20) <= REG0_I(I+36);
    REG0_I(I+36) <= REG0_I(I+52);
    REG0_I(I+52) <= REG0_I(I+ 8);
    REG0_I(I+ 8) <= REG0_I(I+24);
    REG0_I(I+24) <= REG0_I(I+40);
    REG0_I(I+40) <= REG0_I(I+56);
    REG0_I(I+56) <= REG0_I(I+12);
    REG0_I(I+12) <= REG0_I(I+28);
    REG0_I(I+28) <= REG0_I(I+44);
    REG0_I(I+44) <= REG0_I(I+60);
    REG0_Q(I+ 0) <= REG0_Q(I+16);

```

```

        REGO_Q(I+16) <= REGO_Q(I+32);
        REGO_Q(I+32) <= REGO_Q(I+48);
        REGO_Q(I+48) <= REGO_Q(I+ 4);
        REGO_Q(I+ 4) <= REGO_Q(I+20);
        REGO_Q(I+20) <= REGO_Q(I+36);
        REGO_Q(I+36) <= REGO_Q(I+52);
        REGO_Q(I+52) <= REGO_Q(I+ 8);
        REGO_Q(I+ 8) <= REGO_Q(I+24);
        REGO_Q(I+24) <= REGO_Q(I+40);
        REGO_Q(I+40) <= REGO_Q(I+56);
        REGO_Q(I+56) <= REGO_Q(I+12);
        REGO_Q(I+12) <= REGO_Q(I+28);
        REGO_Q(I+28) <= REGO_Q(I+44);
        REGO_Q(I+44) <= REGO_Q(I+60);
    end loop;
    REGO_I(60) <= REGO_I(1);
    REGO_I(61) <= REGO_I(2);
    REGO_I(62) <= REGO_I(3);
    REGO_Q(60) <= REGO_Q(1);
    REGO_Q(61) <= REGO_Q(2);
    REGO_Q(62) <= REGO_Q(3);
end if;
end if;
end if;
end process;

-- OUTHEAD GENERATOR
OH1: process (CLK) begin
    if (RESET = '1') then
        ROOUTHEAD <= '0';
    elsif (CLK'event and CLK = '1' ) then
        if (ROHEAD = '1') then
            ROOUTHEAD <= '1';
        else
            ROOUTHEAD <= '0';
        end if;
    end if;
end process;

-- OUTPUT SELECTOR
ROOUT_I <= REG1_I(0) when BANK = '0' else
    REG0_I(0);
ROOUT_Q <= REG1_Q(0) when BANK = '0' else
    REG0_Q(0);

end;

```

参考文献

- [1] UNIV. OF THE RYUKYUS LSI DESIGN CONTEST 2007
<http://www.ie.u-ryukyu.ac.jp/wada/design07/contest2007.html> —