Nama	Edgrant Henderson Suryajaya	**Kode Asisten**	…………………………
NPM	2206025016	**Jenis Tugas**	TP/CS…………………

Jawaban

1. Code

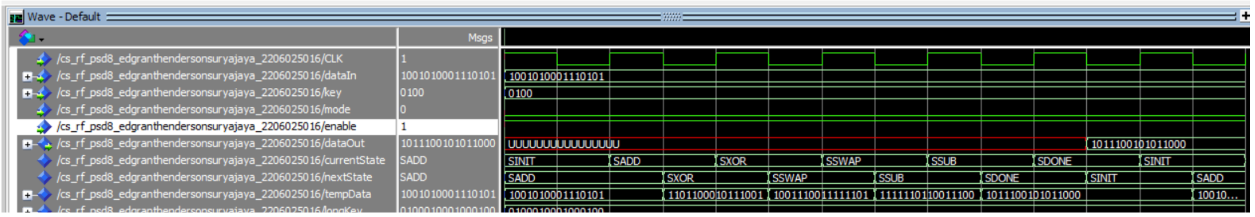
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.MATH_REAL.ALL;


entity CS_RF_PSD8_EdgrantHendersonSuryajaya_2206025016 is
    port (
        CLK : in std_logic;
        dataIn : in std_logic_vector (15 downto 0);
        key : in std_logic_vector (3 downto 0);
        mode : in std_logic;
        enable : in std_logic;
        dataOut : out std_logic_vector (15 downto 0)
    );
end CS_RF_PSD8_EdgrantHendersonSuryajaya_2206025016;

architecture rtl of CS_RF_PSD8_EdgrantHendersonSuryajaya_2206025016 is
    type state is (SINIT, SADD, SXOR, SSWAP, SSUB, SDONE);
    signal currentState, nextState : state := SINIT;

    -- init
    function fInit(oldKey : std_logic_vector) return std_logic_vector  is
    begin
        return oldKey & oldKey & oldKey & oldKey;
    end function;

    -- add
    function fAdd(data, newKey : std_logic_vector ) return std_logic_vector  is
        variable oldKey : unsigned (3 downto 0) := unsigned(newKey(3 downto 0));
```

```vhdl
    variable block0 : unsigned (3 downto 0) := unsigned(data(3 downto 0));
    variable block1 : unsigned (3 downto 0) := unsigned(data(7 downto 4));
    variable block2 : unsigned (3 downto 0) := unsigned(data(11 downto 8));
    variable block3 : unsigned (3 downto 0) := unsigned(data(15 downto 12));
begin
    block0 := block0 + oldKey;
    block1 := block1 + oldKey;
    block2 := block2 + oldKey;
    block3 := block3 + oldKey;

    return std_logic_vector(block3 & block2 & block1 & block0);
end function;

-- xor
function fXor(data, newKey : std_logic_vector ) return std_logic_vector  is
begin
    return data xor newKey;
end function;

-- swap
function fSwap(data: std_logic_vector ) return std_logic_vector  is
begin
    return data(7 downto 0) & data(15 downto 8);
end function;

-- subtract
function fSub(data, newKey : std_logic_vector) return std_logic_vector  is
    variable oldKey : unsigned (3 downto 0) := unsigned(newKey(3 downto 0));
    variable block0 : unsigned (3 downto 0) := unsigned(data(3 downto 0));
    variable block1 : unsigned (3 downto 0) := unsigned(data(7 downto 4));
    variable block2 : unsigned (3 downto 0) := unsigned(data(11 downto 8));
    variable block3 : unsigned (3 downto 0) := unsigned(data(15 downto 12));
begin
    block0 := block0 - oldKey;
    block1 := block1 - oldKey;
    block2 := block2 - oldKey;
    block3 := block3 - oldKey;

    return std_logic_vector(block3 & block2 & block1 & block0);
end function;
```

```vhdl
    -- Done
    function fDone(data : std_logic_vector) return std_logic_vector  is
    begin
        return data;
    end function;

    signal tempData : std_logic_vector (15 downto 0);
    signal longKey : std_logic_vector (15 downto 0);
begin
    -- Mengupdate state setiap clocktick
    clock_upadte: process(clk)
    begin
        if rising_edge(clk) then
            currentState <= nextState;
        end if;
    end process clock_upadte;

    -- state is (SINIT, SADD, SXOR, SSWAP, SSUB, SDONE);
    -- Mengupdate state
    state_update: process
    begin
        case currentState is
            when SINIT =>
                tempData <= dataIn;
                nextState <= SADD;
                if enable = '1' then
                    longKey <= fInit(key);
                end if;

            when SADD =>
                tempData <= fAdd(tempData, longKey);
                if mode = '0' then
                    nextState <= SXOR;
                else
                    nextState <= SSWAP;
                end if;

            when SXOR =>
                tempData <= fXor(tempData, longKey);
                if mode = '0' then
                    nextState <= SSWAP;
                else
                    nextState <= SSUB;
                end if;

            when SSWAP =>
                tempData <= fSwap(tempData);
                if mode = '0' then
                    nextState <= SSUB;
                else
                    nextState <= SXOR;
                end if;

            when SSUB =>
                nextState <= SDONE;
                tempData <= fSub(tempData, longKey);

            when SDONE =>
                nextState <= SINIT;
                dataOut <= fDone(tempData);
        end case;
        wait until falling_edge(clk);
    end process state_update;
end rtl;
```

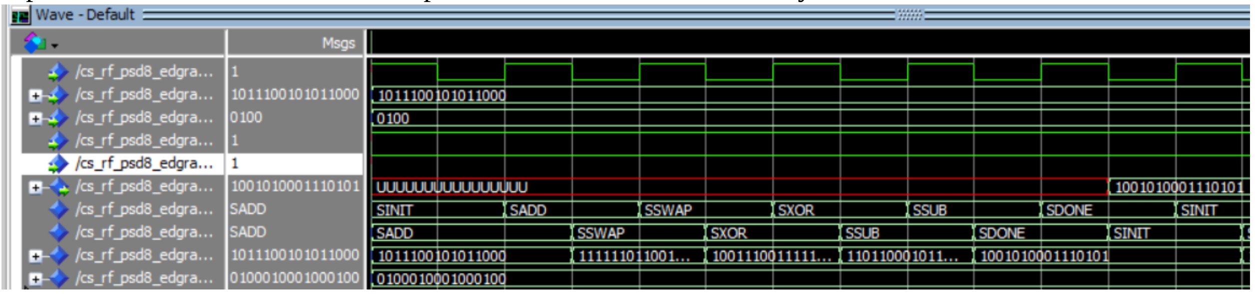
2. Simulasi.

Input 1001010001110101 output 1011100101011000 key 0100



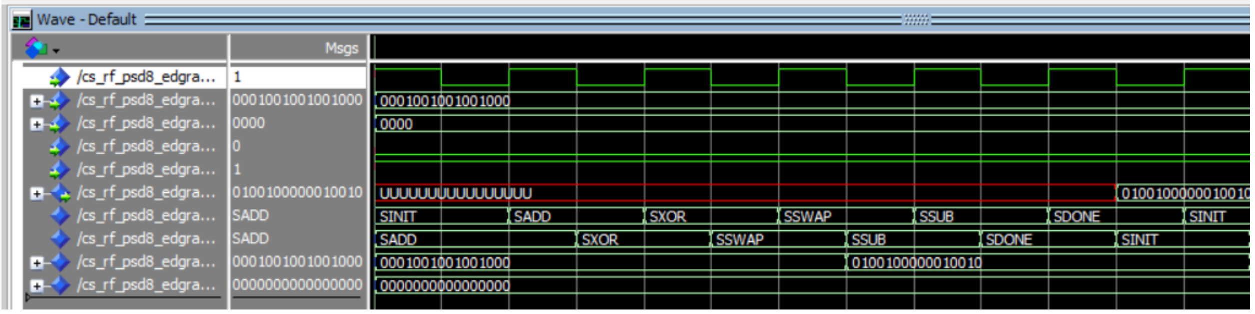
```
Wave - Default
                                                        Msgs
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/CLK          1
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/dataIn       1001010001110101   1001010001110101
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/key          0100               0100
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/mode         0
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/enable       1
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/dataOut      1011100101011000   UUUUUUUUUUUUUUUU  1011100101011000
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/currentState SADD               SINIT SADD SXOR SSWAP SSUB SDONE SINIT
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/nextState    SADD               SADD SXOR SSWAP SSUB SDONE SINIT SADD
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/tempData     1001010001110101   1001010001110101 1101100010111001 1001110011111101 1111110110011100 1011100101011000 10010...
/cs_rf_psd8_edgranthendersonsuryajaya_2206025016/longKey      0100010001000100   0100010001000100
```

Input 1011100101011000 output 1001010001110101 key 0100



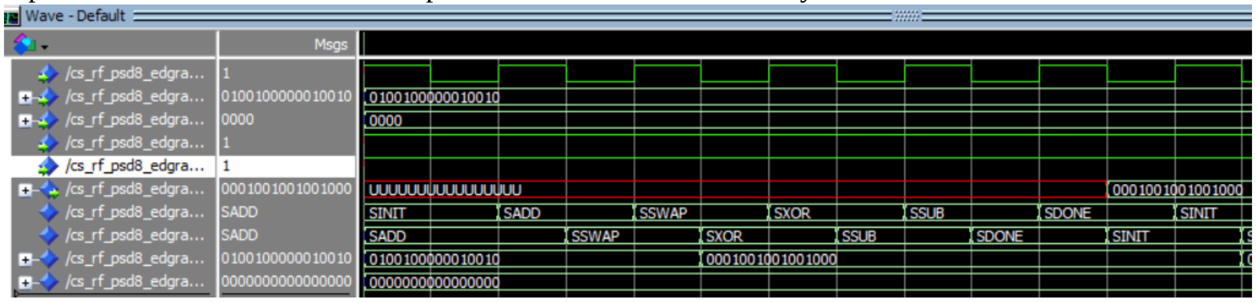
```
Wave - Default
                        Msgs
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    1011100101011000   1011100101011000
/cs_rf_psd8_edgra...    0100               0100
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    1001010001110101   UUUUUUUUUUUUUUUU  1001010001110101
/cs_rf_psd8_edgra...    SADD               SINIT SADD SSWAP SXOR SSUB SDONE SINIT
/cs_rf_psd8_edgra...    SADD               SADD SSWAP SXOR SSUB SDONE SINIT
/cs_rf_psd8_edgra...    1011100101011000   1011100101011000 111111011001... 1001110011111... 1101100010 11... 1001010001110101
/cs_rf_psd8_edgra...    0100010001000100   0100010001000100
```

Input 0001001001001000 output 0100100000010010 key 0000



```
Wave - Default
                        Msgs
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    0001001001001000   0001001001001000
/cs_rf_psd8_edgra...    0000               0000
/cs_rf_psd8_edgra...    0
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    0100100000010010   UUUUUUUUUUUUUUUU  0100100000010010
/cs_rf_psd8_edgra...    SADD               SINIT SADD SXOR SSWAP SSUB SDONE SINIT
/cs_rf_psd8_edgra...    SADD               SADD SXOR SSWAP SSUB SDONE SINIT
/cs_rf_psd8_edgra...    0001001001001000   0001001001001000 0100100000010010
/cs_rf_psd8_edgra...    0000000000000000   0000000000000000
```

Input 0100100000010010 output 0001001001001000 key 0000



```
Wave - Default
                        Msgs
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    0100100000010010   0100100000010010
/cs_rf_psd8_edgra...    0000               0000
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    1
/cs_rf_psd8_edgra...    0001001001001000   UUUUUUUUUUUUUUUU  0001001001001000
/cs_rf_psd8_edgra...    SADD               SINIT SADD SSWAP SXOR SSUB SDONE SINIT
/cs_rf_psd8_edgra...    SADD               SADD SSWAP SXOR SSUB SDONE SINIT
/cs_rf_psd8_edgra...    0100100000010010   0100100000010010 0001001001001000
/cs_rf_psd8_edgra...    0000000000000000   0000000000000000
```

3. **Urutan state untuk dekripsi**
   INIT → ADD → SWAP → XOR → SUB → DONE → INIT

4. **Apakah hasil output dekripsi sama dengan input enkripsi?**
   Sama, karena urutan dekripsi adalah kebalikan dari enkripsi, enkripsi dibalik dari akhir, jadinya

   - DONE → SUB → SWAP → XOR → ADD → INIT.

   Done dan Init tidak melakukan operasi, jadi urutannya tetap sama, proses jadinya

   - INIT → SUB → SWAP → XOR → ADD → DONE.

   XOR dan SWAP adalah operasi simetris, artinya kalau dilakukan 2 kali akan balik ke nilai awal. Namun add dan sub tidak simetris, jika ingin balik ke nilai awal, sub harus diganti ke add dan add harus ganti ke sub. Jadinya proses jadinya

   - INIT → ADD → SWAP → XOR → SUB → DONE.

   Karena hal tersebut, dekripsi nilai hasil enkripsi akan mendapatkan nilai awal.