

## 1. Kode Verilog

```
module MicroProcessorDesign(  
/*I/O utama*/clock, Instruksi, Result  
/*untuk penjelas*/, Op1, Op2);  
input clock;  
input [31:0] Instruksi;  
output reg [31:0] Result;  
  
reg [3:0] RsSetunggal,RsKaleh,Rd4bit,OpCode;  
output reg [31:0] Op1, Op2;  
reg [15:0] isiRegister[31:0];  
  
always @(clock)  
begin  
/*Control Unit*/  
integer i, k;  
for(i = 8; i<12; i = i+1)  
OpCode[i-8] = Instruksi[i];  
for(i = 12; i<16; i = i+1)  
RsKaleh[i-12] = Instruksi[i];  
for(i = 16; i<20; i = i+1)  
RsSetunggal[i-16] = Instruksi[i];  
for(i = 20; i<24; i = i+1)  
Rd4bit[i-20] = Instruksi[i];  
  
/*Register*/  
for(k=0; k<16; k = k+1)  
begin  
isiRegister[k] = (k << 2) + 3;  
end  
Op1 = isiRegister[RsSetunggal];  
Op2 = isiRegister[RsKaleh];  
  
/*Arithmetic Logic Unit (ALU) */  
case(OpCode)  
4'h0: Result = (Op1 + Op2);  
4'h1: Result = (Op1 - Op2);  
4'h2: Result = (Op1 >> Op2);  
4'h3: Result = (Op1 << Op2);  
4'h4: // Perkalian Bit  
begin  
integer i,j;  
reg [31:0] temp, tempAnd = 32'h0;  
Result = 32'h0;  
  
for(i=0; i < 32; i = i+1)  
begin  
temp = (Op2 << i);  
for(j = 0; j <32;j = j+1)  
begin  
tempAnd[j] = (temp[j] & Op1[i]);  
end  
Result = (Result + tempAnd);  
end  
end  
4'h5: // Pembagian Bit  
begin  
reg [63:0]tOp1=64'h0, tOp2=64'h0, hdiff, shdiff;
```

```



integer v, w, wmax;
Result = 32'hFFFF;

for(v = 0; v < 32; v = v+1)
begin
    tOp1[v] = Op1[v];
    tOp2[32+v] = Op2[v];
end
wmax = 32;
hdiff = tOp1;
if(tOp2 == 64'h0) Result = 32'hFFFFFFFF;
else
begin
    for(w = 0; w < wmax; w = w+1)
    begin
        shdiff = hdiff;
        hdiff = hdiff - (tOp2 >> (w+1));
        if((hdiff >= 64'h0) && (hdiff < 64'hFFFFFFFF)) Result[wmax-w-1] = 1'b1;
        else if(hdiff >= 64'hFFFFFFFF)
        begin
            Result[wmax-w-1] = 1'b0;
            hdiff = shdiff;
        end
    end
end
end
endcase

/*Rdestination = Result ----> Proses pada Register*/
isiRegister[Rd4bit] = Result;
end
endmodule

```

## 2. Hardware Utilization Report , speed, power estimation

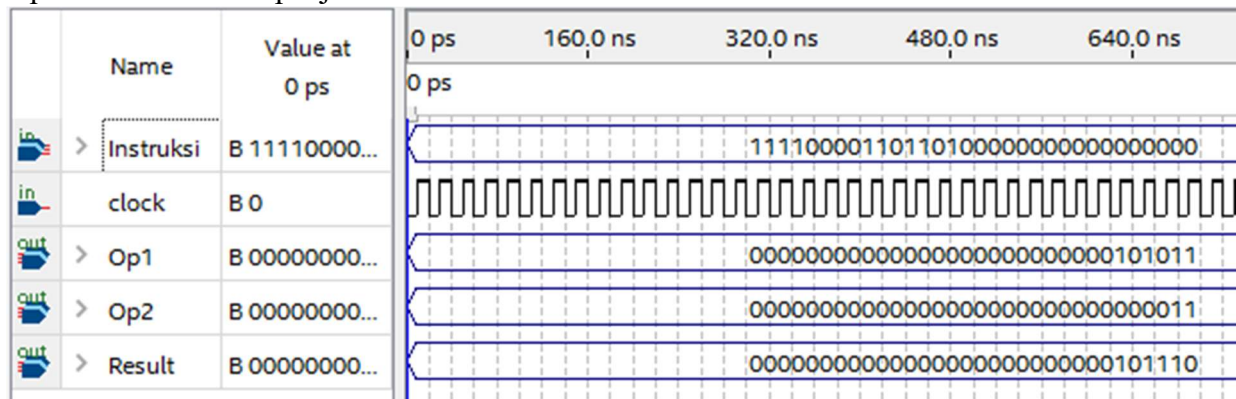
Flow Summary		Power Analyzer Summary	
 <<Filter>>		 <<Filter>>	
Flow Status	Successful - Tue Nov 27 18:17:38 2018	Power Analyzer Status	Successful - Tue Nov 27 18:17:38 2018
Quartus Prime Version	18.0.0 Build 614 04/24/2018 SJ Lite Edition	Quartus Prime Version	18.0.0 Build 614 04/24/2018 SJ Lite Edition
Revision Name	MicroProcessorDesign	Revision Name	MicroProcessorDesign
Top-level Entity Name	MicroProcessorDesign	Top-level Entity Name	MicroProcessorDesign
Family	Cyclone V	Family	Cyclone V
Device	5CSEMA5F31C6	Device	5CSEMA5F31C6
Timing Models	Final	Power Models	Final
Logic utilization (in ALMs)	1,780 / 32,070 ( 6 % )	Total Thermal Power Dissipation	422.32 mW
Total registers	0	Core Dynamic Thermal Power Dissipation	0.00 mW
Total pins	129 / 457 ( 28 % )	Core Static Thermal Power Dissipation	411.24 mW
Total virtual pins	0	I/O Thermal Power Dissipation	11.08 mW
Total block memory bits	0 / 4,065,280 ( 0 % )	Power Estimation Confidence	Low: user provided insufficient toggle rate data
Total DSP Blocks	0 / 87 ( 0 % )		
Total HSSI RX PCSs	0		
Total HSSI PMA RX Deserializers	0		
Total HSSI TX PCSs	0		
Total HSSI PMA TX Serializers	0		
Total PLLs	0 / 6 ( 0 % )		
Total DLLs	0 / 4 ( 0 % )		

Slow 1100mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	113.05 MHz	113.05 MHz	Instruksi[10]	

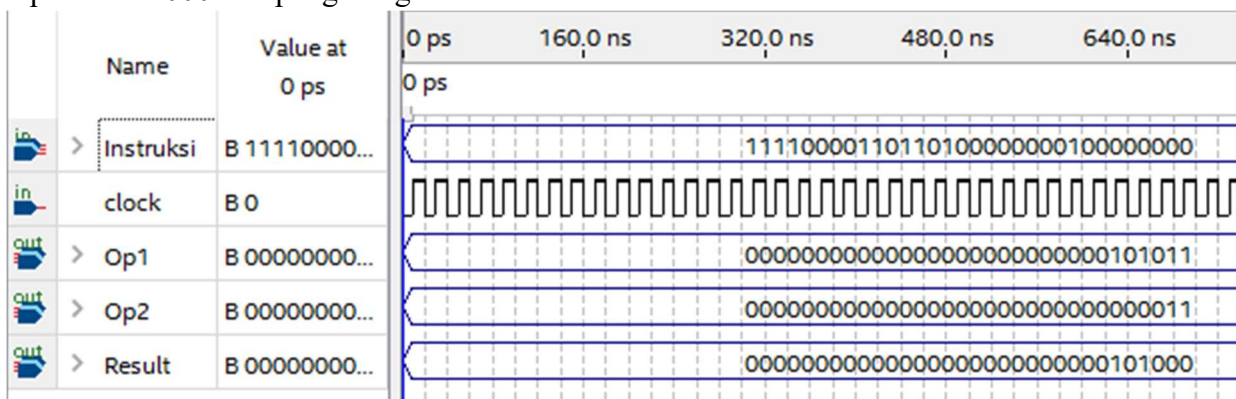
Slow 1100mV 0C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	114.44 MHz	114.44 MHz	Instruksi[10]	

### 3. Simulasi menggunakan Model-Sim

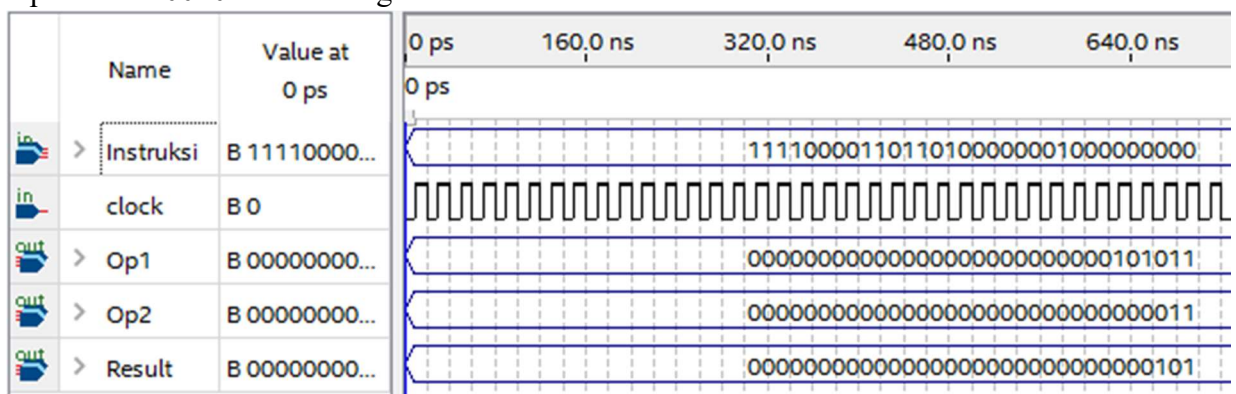
OpCode 4'b0000 → penjumlahan



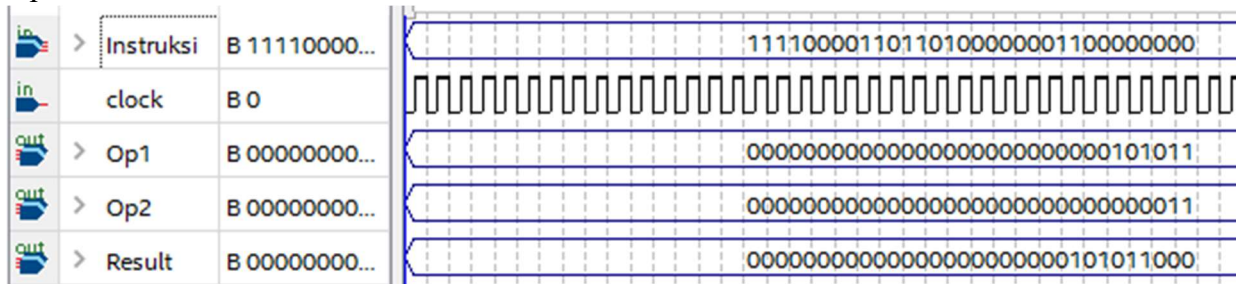
OpCode 4'b0001 → pengurangan



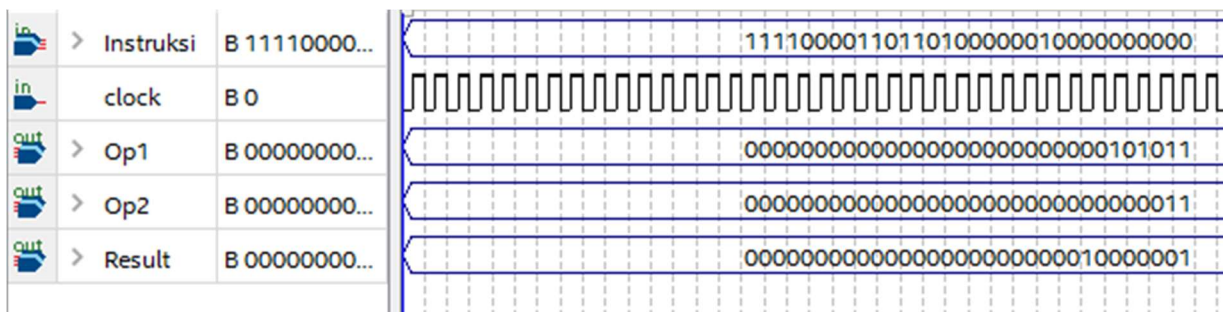
OpCode 4'b0010 → Shift Right



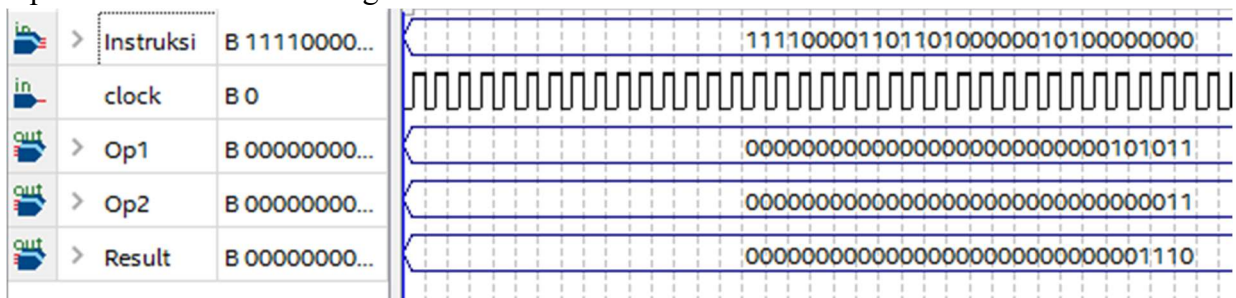
OpCode 4'b0011 → Shift Left



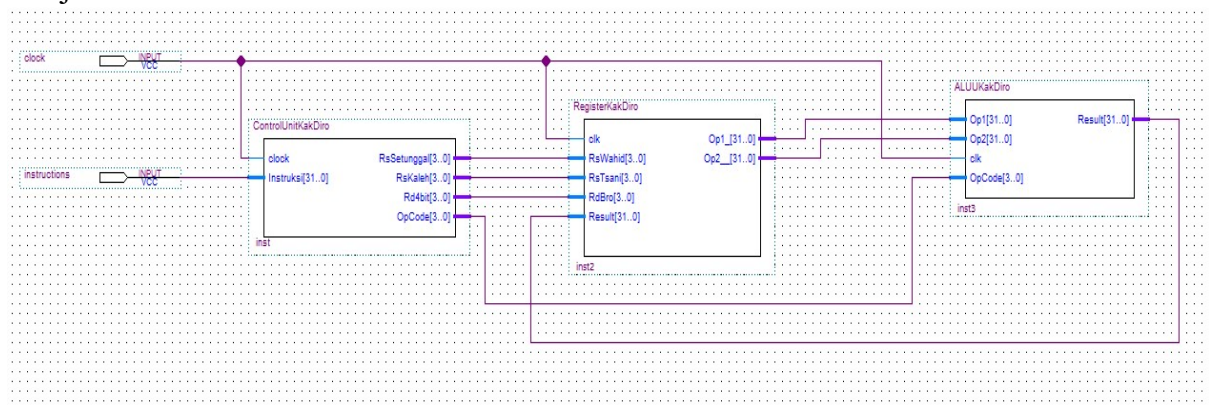
OpCode 4'b0100 → Perkalian



OpCode 4'b0101 → Pembagian



Sebenarnya, kode Verilog awal saya memisahkan antara CU, ALU, dan REGISTER dalam module yang berbeda, namun karena sering error saat linking ketiganya dalam meng-compile, akhir nya saya putuskan untuk menjadikannya dalam satu module bernama MicroProcessorDesign. Oleh karena itu pada awalnya, sudah saya generate symbol nya menjadi berikut:



#### 4. Penjelasan

Module MicroProcessorDesign ini mencakup dari Control Unit, Register, serta Arithmetic Logic Unit.

Untuk Control Unit, bekerja mendekomposisi instruksi menjadi OpCode (yang berguna untuk menentukan operasi apa yang akan dilakukan), Rs1 (alamat register sebagai operand pertama), Rs2 (alamat register sebagai operand kedua), serta Rd (alamat register tujuan, tempat menyimpan nilai hasil operasi pada ALU), yang masing – masing memiliki panjang 4 bit.

Instruksi yang saya gunakan pada uji coba tugas ini adalah:

1111 0000 Rd(1101) Rs1(1010) Rs2(0000) OpCode(0\*\*\*) 0000 0000

Dengan OpCode 0\*\*\* maksudnya adalah dapat berubah nilai dari 0 hingga 5, sesuai operasi yang akan dilakukan.

Pada bagian Register, saya menginisialisasi nilai tiap alamat Register ke- $k$  sebagai  $(k \ll 2) + 3$ , atau dalam kata lain  $(4k + 3)$ , sedangkan yang menjadi operand pertama adalah register ke-10, (sehingga nilainya adalah 43), sedangkan operand kedua adalah register ke-0 (sehingga nilainya adalah 3). Pemilihan nilai operand kedua kecil, agar nilai saat Shift Right ataupun pembagian cukup mudah diamati. Sedangkan register tujuan sebagai tempat menyimpan nilai perhitungan ALU dipilih pada register ke-13.

Bagian berikut nya adalah ALU. Pada bagian ini, operand satu dan dua akan dioperasikan sesuai dengan nilai OpCode yang digunakan. Untuk 4'h0, adalah penjumlahan, 4'h1 pengurangan, 4'h2 shift right, 4'h3 shift left, sedangkan 4'h4 dan 4'h5 secara berurutan adalah perkalian dan pembagian.

Untuk perkalian, di sini saya mencoba menggunakan perkalian bit, yakni seperti halnya proses perkalian bilangan puluhan ataupun ratusan dan keatas secara manual. Mudahnya berikut contohnya  $1101 \times 0011 = (1 \times 0011) + ((0 \times 0011) \ll 2) + (1 \times 0011) \ll 3) + (1 \times 0011) \ll 4)$ , sehingga secara mudahnya kita menshift left dan mengalikan dengan tiap – tiap bit operand pertama, namun lebih efektif jika hanya dilakukan pada saat bit nya 1, namun saat saya coba di computer saya mengalami error, maka saya inisiasi dengan tetap menshift pada setiap bit nya, namun kemudian meng-and kan nya, dan menjumlahkan kesemua nilai hasil operasi tersebut.

Begitu juga dengan pembagian, saya belajar untuk membuat sendiri berdasarkan algoritma pembagaian pada bit, yakni dengan mengurangi operand 2 dari operand 1 yang awalnya dishift left sedemikian sehingga LSB operand 2 berada tepat di MSB operand 1, kemudian mengurangi operand 2 dari operand 1. Dari nilai pengurangan tersebut, jika nilainya lebih atau sama dengan nol, maka diperoleh bit 1, langkah berikutnya adalah dengan meng-right shift kan operand 2 tersebut dan mengulangnya tersu hingga hasil pengurangan terhadap operand 1, bernilai negative, untuk kondisi ini, maka diisi dengan bit nol.

Namun karena di Quartus, bit- bit nol justru direpresentasikan sebagai rentetan bit – bit nol, dengan MSB 1 namun tersembunyi, sehingga saat diharapkan bernilai negative, justru menghasilkan nilai yang sangat besar, sehingga strateginya yakni, tidak hanya mempertimbangkn nilai lebih atau sama dengan nol, untuk hasil bit 1, namun juga saat nilai kurang dari nilai maksimum (sebagai representasi bit – bit negative di Quartus).

Langkah terakhir, yang sebenarnya terjadi di Register yakni menyimpan nilai hasil perhitungan ALU, ke dalam Register tujuan (Rd).