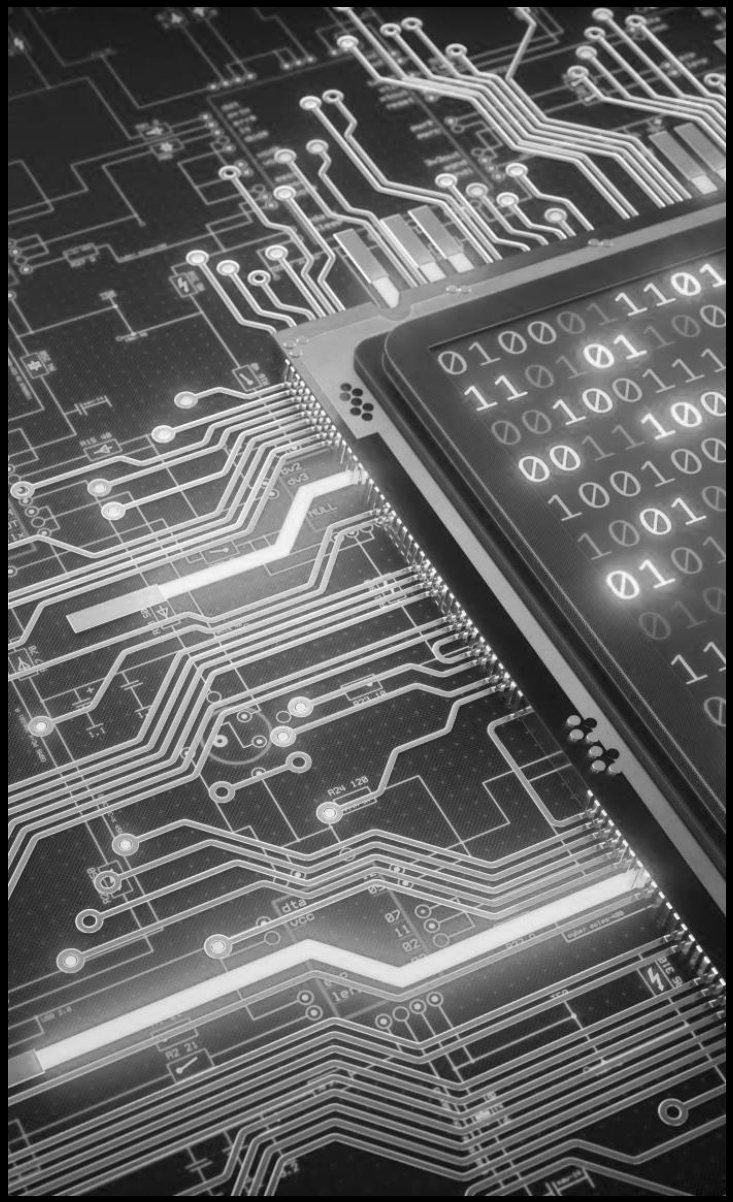
Artgraphics



Build Netlist
論理合成システム

性能予測のためのRTL論理合成機能

はじめに

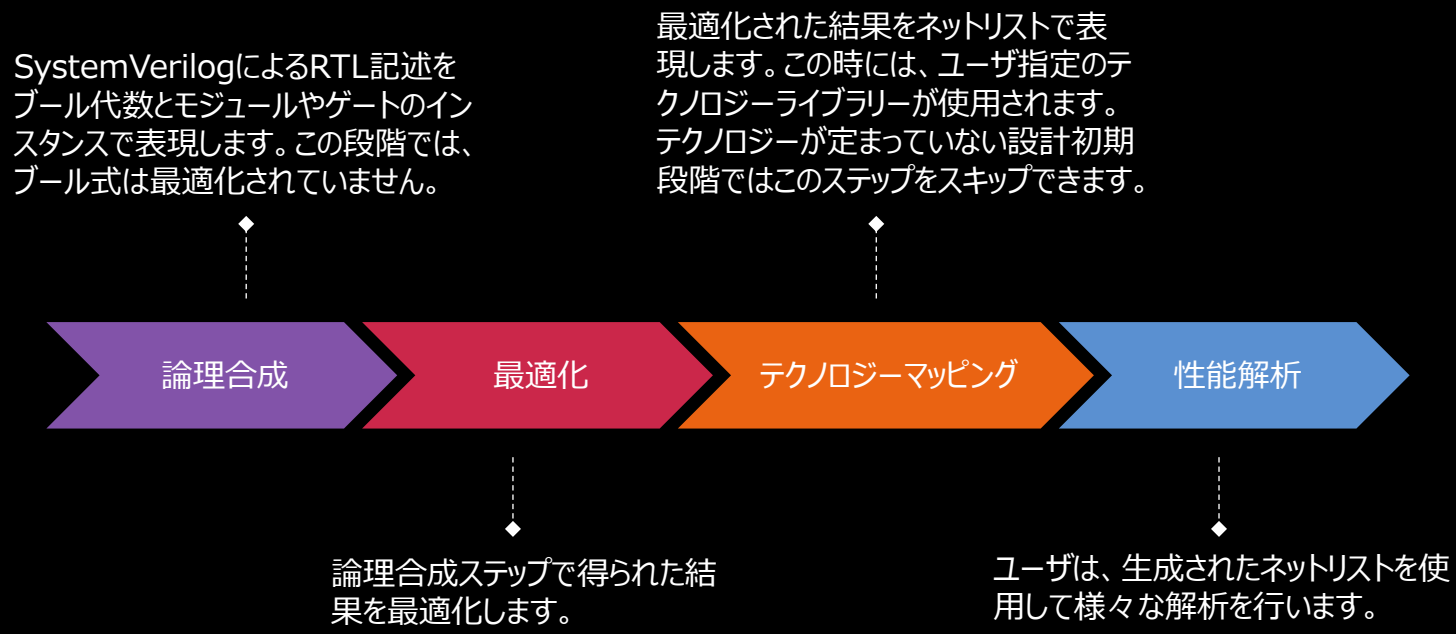
- 論理合成システムはRTL記述によるデザインの性能をいち早く予測するための機能です。
- 論理合成システムは、ユーザ指定のライブラリーを使用してRTL記述からネットリストを生成します。ユーザは、生成されたネットリストを基にして様々な解析ツールによりデザインの性能を測定する事ができます。
- 論理合成システムには制約が少ないので、RTL設計の初期段階から論理合成を行えます。したがって、早期に論理合成可能か否かを判定する事ができます。また、予期しないラッチ生成を早期に発見できる機会にも恵まれます。
- 論理合成システムは、GUIを持たないターミナルでも使用できますが、以下では SystemVerilog IDE の一機能としての論理合成システムの概要を解説します。
- 論理合成システムは Windows の環境で動作します。

論理合成システム概説

論理合成システムは設計の初期段階における性能予測機能として位置付けられています。論理合成システムはネットリストを生成しますが、所謂論理合成としてのスタンスをとっていません。

論理合成システムの構成と役割

- 論理合成システムは、合成、最適化、テクノロジーマッピングから構成されています。それぞれの過程の終了時には処理結果がモデル形式で出力されます。最後のステップではモデル形式の代わりに Verilog または SystemVerilog 形式として出力されます。

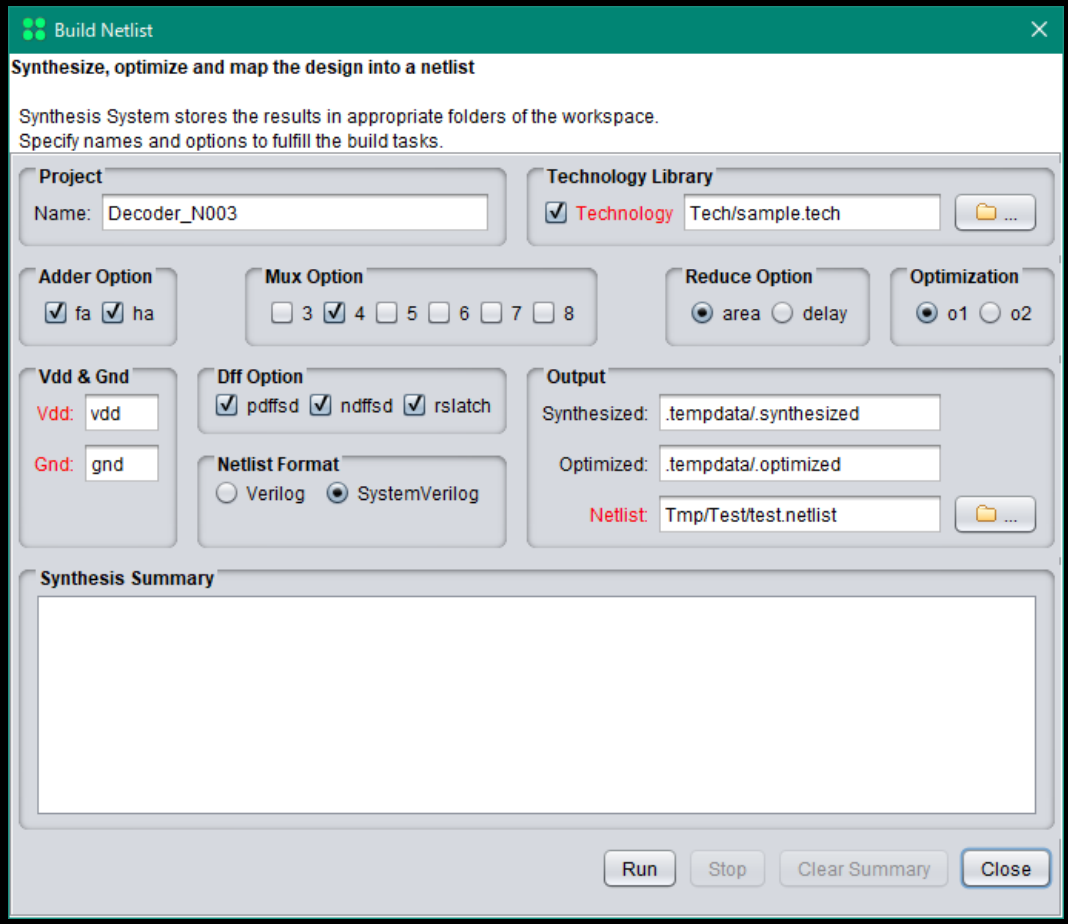


論理合成、最適化、テクノロジーマッピング

- 論理合成に必要なステップは、Build Netlist ダイアログを使用すると簡単に行えます。
- ダイアログでは、テクノロジーライブラリーやネットリスト名称を指定します。その他、特別な回路の使用を指示します。
- 指定が完了したら、Runボタンをクリックするだけで論理合成が行われます。
- 論理合成が終了すると Synthesis Summary パネルに合成結果の総括情報が表示されます。また、生成されたネットリストはプロジェクトの Synthesis フォルダーに登録されるので、テキストエディタで開く事ができます。

Build Netlist ダイアログ

- 論理合成の対象となるソースファイルは自動的に収集されます。
- Technologyチェックボックスが選択されていなければテクノロジーマッピングはスキップされます。
- ネットリストの形式としてVerilogまたはSystemVerilogの選択をできるので、使用する解析ツールに合わせて選べます。



合成例

簡単な回路記述を使用して論理合成機能を紹介します。

モデル形式

- テクノロジーが定まっていない設計初期段階ではテクノロジーライブラリーを指定せずに合成できます。この場合には、合成結果はモデル形式で出力されます。
- モデル形式はシンプルなシンタックスなので、解析は容易です。

```
module Pdff_N004(input clk,set,reset,d,output logic q);
always_ff @(posedge clk,posedge reset,posedge set)
        if( reset )
                q <= 1'b0;
        else if( set )
                q <= 1'b1;
        else
                q <= d;
endmodule
```

フリップフロップのRTL記述

```
model Pdff_N004;
input   clk;
input   set;
input   reset;
input   d;
output  q;
//      dataflow section
assign  t$n1 = ~reset*set;
//      instance section
m$pdff v$i1 d=d, ck=clk, pr=t$n1, cl=reset, q=q;
endmodel
```

モデル形式による合成結果

マルチプレクサ

マルチプレクサのRTL記述

```
module mux4_if(input a,b,c,d,sl,s0,output logic out);

always @(a,b,c,d,sl,s0)
        if( {sl,s0} == 0 )
                out = a;
        else if( {sl,s0} == 1 )
                out = b;
        else if( {sl,s0} == 2 )
                out = c;
        else
                out = d;

endmodule
```

合成されたネットリスト

```
module mux4_if(input a,b,c,d,sl,s0,output logic out);
//      instance section
mux4 v$il (.d0(a), .dl(b), .d2(c), .d3(d), .s0(s0), .sl(sl), .z(out));
endmodule
```

プライオリティコーディング

プライオリティコーディング
のRTL記述

```
module mux_priority(input a,b,c,d,sa,sb,sc,output logic out);

always @(a,b,c,d,sa,sb,sc)
        if( sa == 1'b1 )
                out = a;
        else if( sb == 1'b1 )
                out = b;
        else if( sc == 1'b1 )
                out = c;
        else
                out = d;

endmodule
```

合成されたネットリスト

```
module mux_priority(input a,b,c,d,sa,sb,sc,output logic out);
//      instance section
mux v$i1 (.d0(d), .d1(c), .s(sc), .z(t$n1));
mux v$i2 (.d0(t$n1), .d1(b), .s(sb), .z(t$n2));
mux v$i3 (.d0(t$n2), .d1(a), .s(sa), .z(out));
endmodule
```

プライオリティコーディング

プライオリティコーディング
のRTL記述

```
module mux_priority(input a,b,c,d,sa,sb,sc,output logic out);

always @(a,b,c,d,sa,sb,sc)
        casex ({sa,sb,sc})
        3'b1xx: out = a;
        3'bx1x: out = b;
        3'bxx1: out = c;
        default: out = d;
        endcase

endmodule
```

合成されたネットリスト

```
module mux_priority(input a,b,c,d,sa,sb,sc,output logic out);
//      instance section
mux v$i1 (.d0(d), .d1(c), .s(sc), .z(t$n2));
mux v$i2 (.d0(t$n2), .d1(b), .s(sb), .z(t$n1));
mux v$i3 (.d0(t$n1), .d1(a), .s(sa), .z(out));
endmodule
```

デコーダー

デコーダーのRTL記述

```
module decoder(input logic [2:0] code,output logic [7:0] data);

always_comb
        foreach(data[i])
                data[i] = code == i;

endmodule
```

合成されたネットリスト

```
module decoder(input [2:0] code,output logic [7:0] data);
//      instance section
inv t$i10 (.a(code[1]), .z(t$n2));
inv t$i11 (.a(code[0]), .z(t$n3));
and3 t$i1 (.a1(t$n3), .a2(code[1]), .a3(t$n1), .z(data[2]));
and3 t$i2 (.a1(t$n3), .a2(t$n2), .a3(t$n1), .z(data[0]));
and3 t$i3 (.a1(code[0]), .a2(code[1]), .a3(t$n1), .z(data[3]));
and3 t$i4 (.a1(code[0]), .a2(t$n2), .a3(t$n1), .z(data[1]));
inv t$i5 (.a(code[2]), .z(t$n1));
and3 t$i6 (.a1(t$n3), .a2(code[1]), .a3(code[2]), .z(data[6]));
and3 t$i7 (.a1(t$n3), .a2(t$n2), .a3(code[2]), .z(data[4]));
and3 t$i8 (.a1(code[0]), .a2(code[1]), .a3(code[2]), .z(data[7]));
and3 t$i9 (.a1(code[0]), .a2(t$n2), .a3(code[2]), .z(data[5]));
endmodule
```

カウンター

カウンターのRTL記述

```
module counter #(NBITS=4)
        (input clk,reset,output logic [NBITS-1:0] q);

always @(posedge clk,posedge reset)
        if( reset == 1 )
                q <= '0;
        else
                q <= q+1;

endmodule
```

合成されたネットリスト

```
module counter(input clk,reset,output logic [3:0] q);
//      instance section
pdffc v$i1 (.d(t$n1), .ck(clk), .cl(reset), .q(q[0]), .qbar(t$n1));
pdffc v$i2 (.d(t$n2), .ck(clk), .cl(reset), .q(q[1]));
pdffc v$i3 (.d(t$n3), .ck(clk), .cl(reset), .q(q[2]));
pdffc v$i4 (.d(t$n4), .ck(clk), .cl(reset), .q(q[3]));
ha v$i5 (.a(q[0]), .b(q[1]), .co(t$n5), .sum(t$n2));
ha v$i6 (.a(t$n5), .b(q[2]), .co(t$n6), .sum(t$n3));
ha v$i7 (.a(t$n6), .b(q[3]), .co(), .sum(t$n4));
endmodule
```


Grayカウンター

```
module gray_counter #(NBITS=4)
        (input clk,reset,output logic [NBITS-1:0] q);
logic [NBITS-1:0]       binary;

always @(posedge clk,posedge reset)
        if( reset == 1 )
                binary <= 0;
        else
                binary <= binary+1;

always @(binary) begin
        q[NBITS-1] = binary[NBITS-1];

        for( int i = NBITS-2; i >= 0; i-- )
                q[i] = binary[i+1] ^ binary[i];
end

endmodule
```

GrayカウンターのRTL記述

```
module gray_counter(input clk,reset,output logic [3:0] q);
//      instance section
pdffc v$i1 (.d(t$n4), .ck(clk), .cl(reset), .q(t$n2), .qbar(t$n4));
pdffc v$i2 (.d(t$n5), .ck(clk), .cl(reset), .q(t$n3));
pdffc v$i3 (.d(t$n6), .ck(clk), .cl(reset), .q(t$n1));
pdffc v$i4 (.d(t$n7), .ck(clk), .cl(reset), .q(q[3]));
ha v$i5 (.a(t$n2), .b(t$n3), .co(t$n8), .sum(t$n5));
ha v$i6 (.a(t$n8), .b(t$n1), .co(t$n9), .sum(t$n6));
ha v$i7 (.a(t$n9), .b(q[3]), .co(), .sum(t$n7));
xor2 t$i1 (.a(t$n1), .b(q[3]), .z(q[2]));
xor2 t$i2 (.a(t$n2), .b(t$n3), .z(q[0]));
xor2 t$i3 (.a(t$n3), .b(t$n1), .z(q[1]));
endmodule
```

合成されたネットリスト

RSラッチ

RSラッチのRTL記述

```
module RSLatch_N001(input a,en,reset,set,d,output logic z);

always_latch
        if( reset )
                z = 1'b0;
        else if( set )
                z = 1'b1;
        else if( en )
                z = d;

endmodule
```

合成されたネットリスト

```
module RSLatch_N001(input a,en,reset,set,d,output logic z);
//      instance section
rslatch v$i1 (.d(d), .en(t$n1), .pr(t$n2), .cl(reset), .q(z));
and2 t$i1 (.a1(t$n3), .a2(set), .z(t$n2));
inv t$i2 (.a(reset), .z(t$n3));
nor2 t$i3 (.a1(set), .a2(t$n4), .z(t$n1));
nand2 t$i4 (.a1(en), .a2(t$n3), .z(t$n4));
endmodule
```

非同期セットとリセット付きのフリップフロップ

フリップフロップのRTL記述

```
module Pdff_N004(input clk,set,reset,d,output logic q);
always_ff @(posedge clk,posedge reset,posedge set)
        if( reset )
                q <= 1'b0;
        else if( set )
                q <= 1'b1;
        else
                q <= d;
endmodule
```

合成されたネットリスト

```
module Pdff_N004(input clk,set,reset,d,output logic q);
//      instance section
pdffpc v$i1 (.d(d), .ck(clk), .pr(t$n1), .cl(reset), .q(q));
and2 t$i1 (.a1(t$n2), .a2(set), .z(t$n1));
inv t$i2 (.a(reset), .z(t$n2));
endmodule
```

JK-フリップフロップ

JK-フリップフロップのRTL記述

```
typedef enum logic [1:0] { HOLD=2'b00, RESET, SET, TOGGLE }     jk_e;

module jk_ff(input logic clk,j,k,output logic q,qbar);

assign qbar = ~q;

always @(posedge clk) begin
        case ({j,k})
        HOLD:   q <= q;                 // HOLD
        RESET:  q <= 0;                 // RESET
        SET:    q <= 1;                 // SET
        TOGGLE: q <= ~q;                // TOGGLE
        endcase
end

endmodule
```

合成されたネットリスト

```
module jk_ff(input clk,j,k,output logic q,qbar);
//      instance section
pdff v$i1 (.d(t$n1), .ck(clk), .q(q), .qbar(qbar));
mux t$i1 (.d0(j), .d1(t$n2), .s(q), .z(t$n1));
inv t$i2 (.a(k), .z(t$n2));
endmodule
```

おわりに

プレゼンテーションをご覧頂きまして有難うございます。
論理合成システムの製品評価は近々可能になります。
ご興味のある方は、是非、ご連絡下さい。