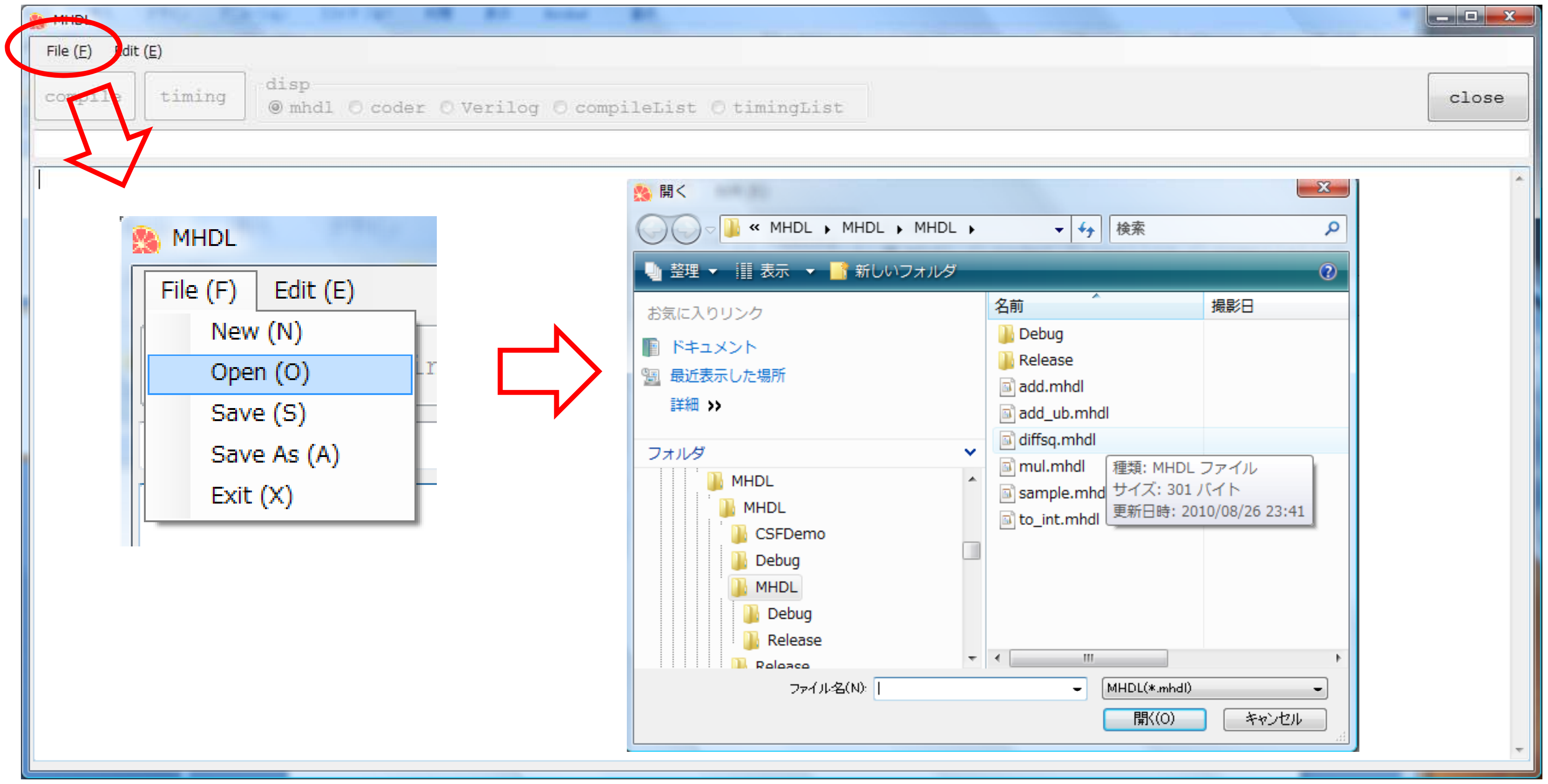
“***CodeSqueezer Floating***” デモソフト

操作方法および結果例

2010.09.02 : DASにおけるデモ内容

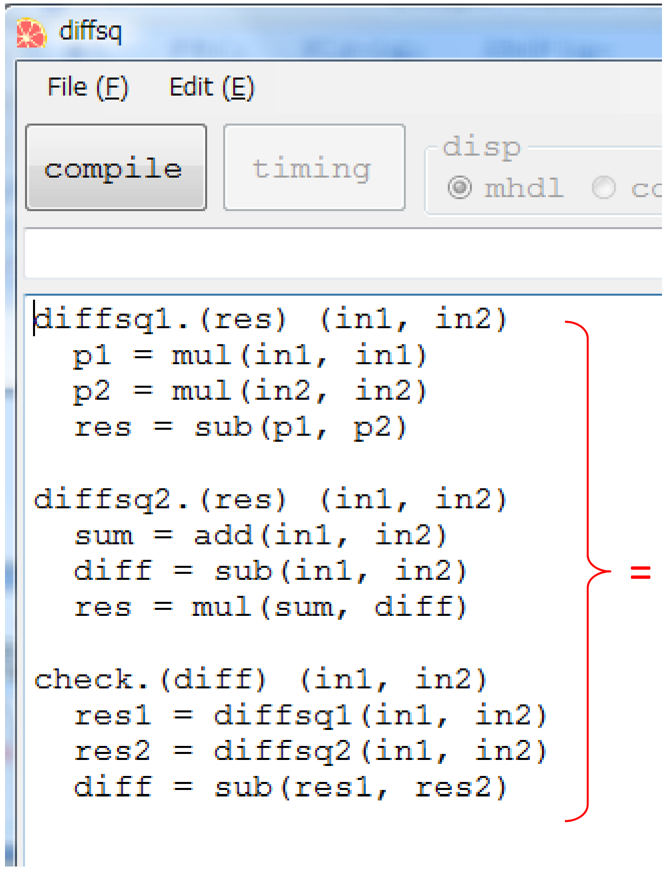
起動画面

"CSFDemo" (CodeSqueezer Floating Demo) のアイコンをクリックすると初期画面(下図)が現れる
Fileプルダウンメニューで"Open"を選びMHDLソースファイルを選択する
ここでは"diffsq.mhdl"を選択する



mhdlソースコード画面

mhdlファイルを開くとその内容がテキストボックスに表示される。
mhdlファイルはファンクション・ダイアグラムを表現したもの



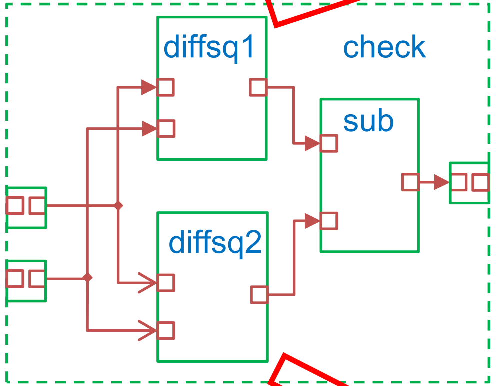
```
diffsq1.(res) (in1, in2)
  p1 = mul(in1, in1)
  p2 = mul(in2, in2)
  res = sub(p1, p2)

diffsq2.(res) (in1, in2)
  sum = add(in1, in2)
  diff = sub(in1, in2)
  res = mul(sum, diff)

check.(diff) (in1, in2)
  res1 = diffsq1(in1, in2)
  res2 = diffsq2(in1, in2)
  diff = sub(res1, res2)
```

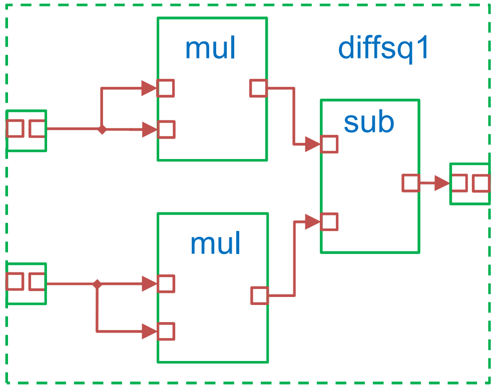
=

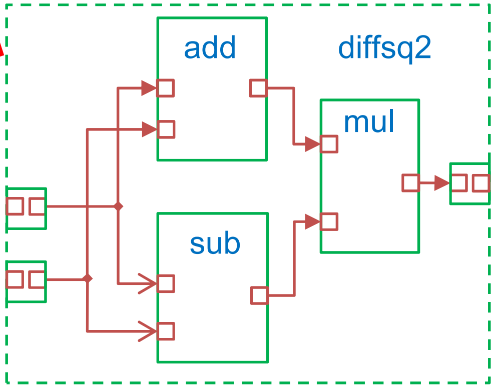
diffsq1(in1, in2)
 res = in1 * in1 – in2 * in2

diffsq2(in1, in2)
 res = (in1 + in2) * (in1 – in2)

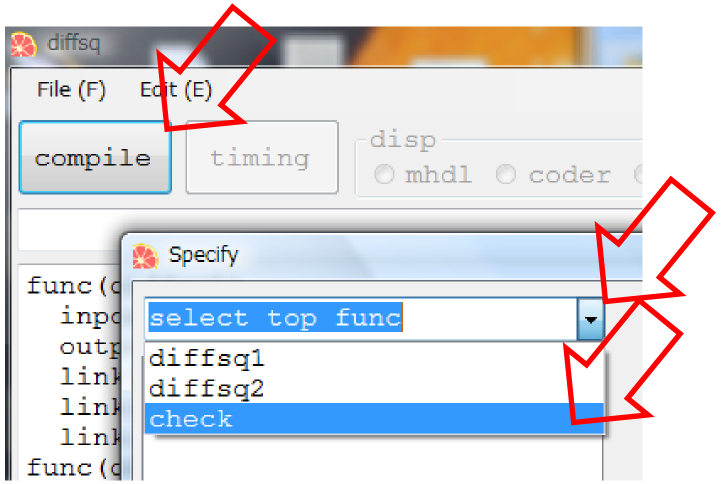
check(in1, in2)
 diff = diffsq1(in1, in2) –
  diffsq2(in1, in2)



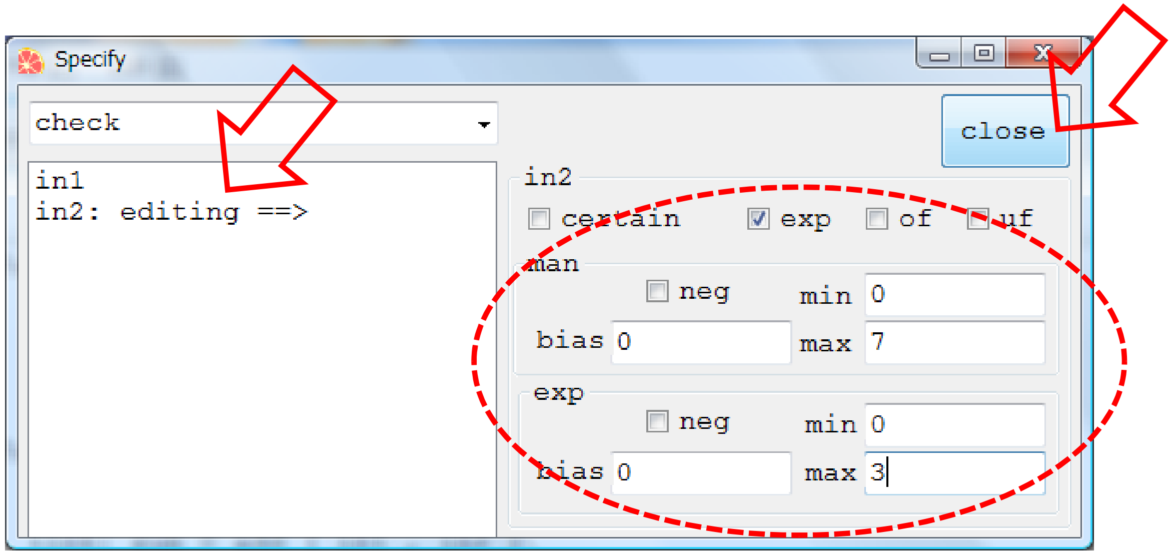




mhdlソースコードのコンパイル



```
func (
  inp
  outp
  link
  link
  link
func (
```



check

```
in1
in2: editing ==>
```

in2

☐ certain ☑ exp ☐ of ☐ uf

man: ☐ neg, min 0, bias 0, max 7

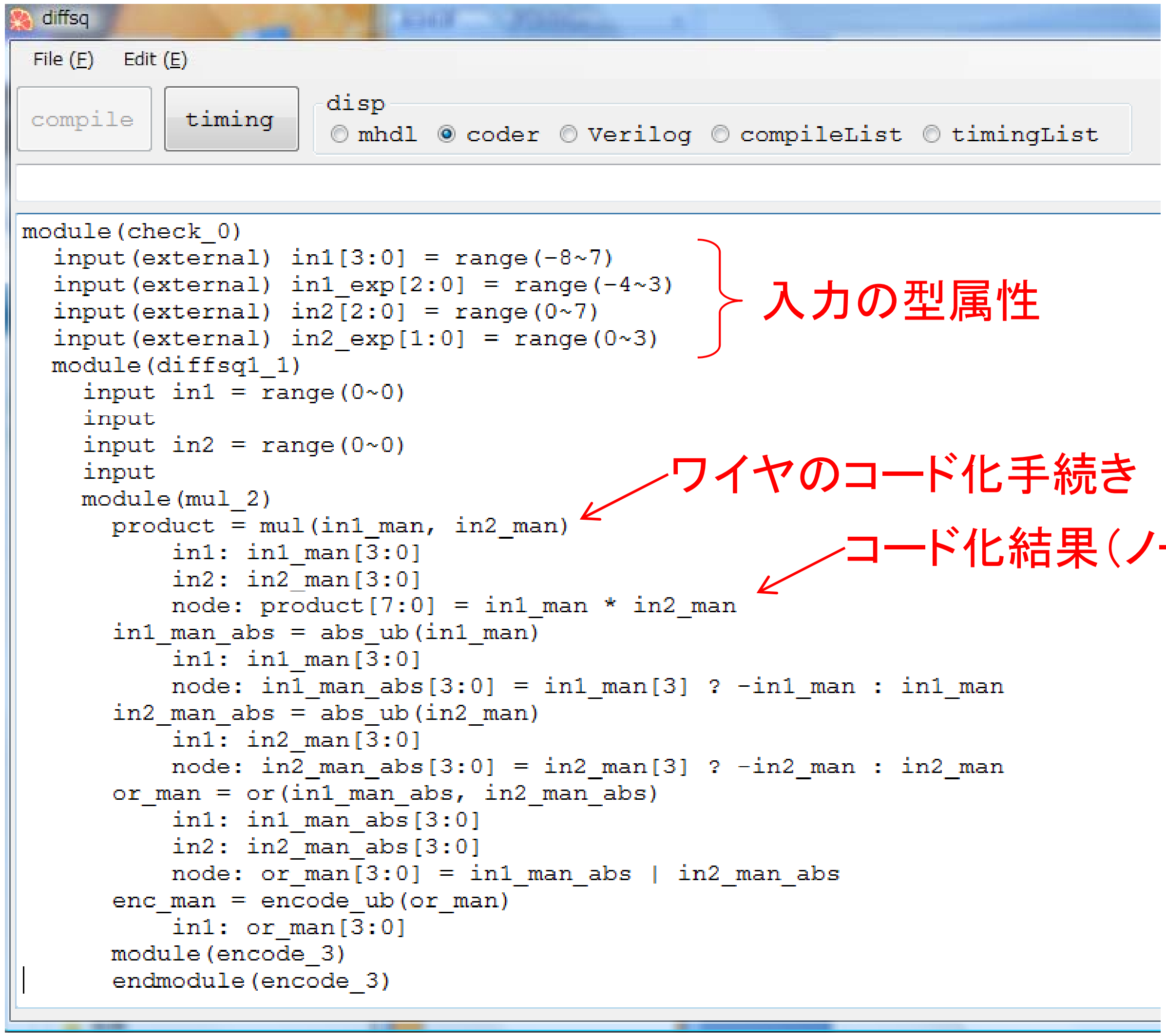
exp: ☐ neg, min 0, bias 0, max 3

“compile”ボタンを押すと“Specify”ウィンドウが現れる。まず、コンパイルしたいファンクションを選択する

次に、ファンクションの入力リンクの各々を選択し、数値の型属性を指定する

最後にcloseボタンを押す

mhdlソースコードのコンパイル結果



disp: ○ mhdl [✓ coder] ○ Verilog ○ compileList ○ timingList

```
module(check_0)
  input(external) in1[3:0] = range(-8~7)
  input(external) in1_exp[2:0] = range(-4~3)
  input(external) in2[2:0] = range(0~7)
  input(external) in2_exp[1:0] = range(0~3)
  module(diffsq1_1)
    input in1 = range(0~0)
    input
    input in2 = range(0~0)
    input
    module(mul_2)
      product = mul(in1_man, in2_man)
          in1: in1_man[3:0]
          in2: in2_man[3:0]
          node: product[7:0] = in1_man * in2_man
      in1_man_abs = abs_ub(in1_man)
          in1: in1_man[3:0]
          node: in1_man_abs[3:0] = in1_man[3] ? -in1_man : in1_man
      in2_man_abs = abs_ub(in2_man)
          in1: in2_man[3:0]
          node: in2_man_abs[3:0] = in2_man[3] ? -in2_man : in2_man
      or_man = or(in1_man_abs, in2_man_abs)
          in1: in1_man_abs[3:0]
          in2: in2_man_abs[3:0]
          node: or_man[3:0] = in1_man_abs | in2_man_abs
      enc_man = encode_ub(or_man)
          in1: or_man[3:0]
      module(encode_3)
      endmodule(encode_3)
```

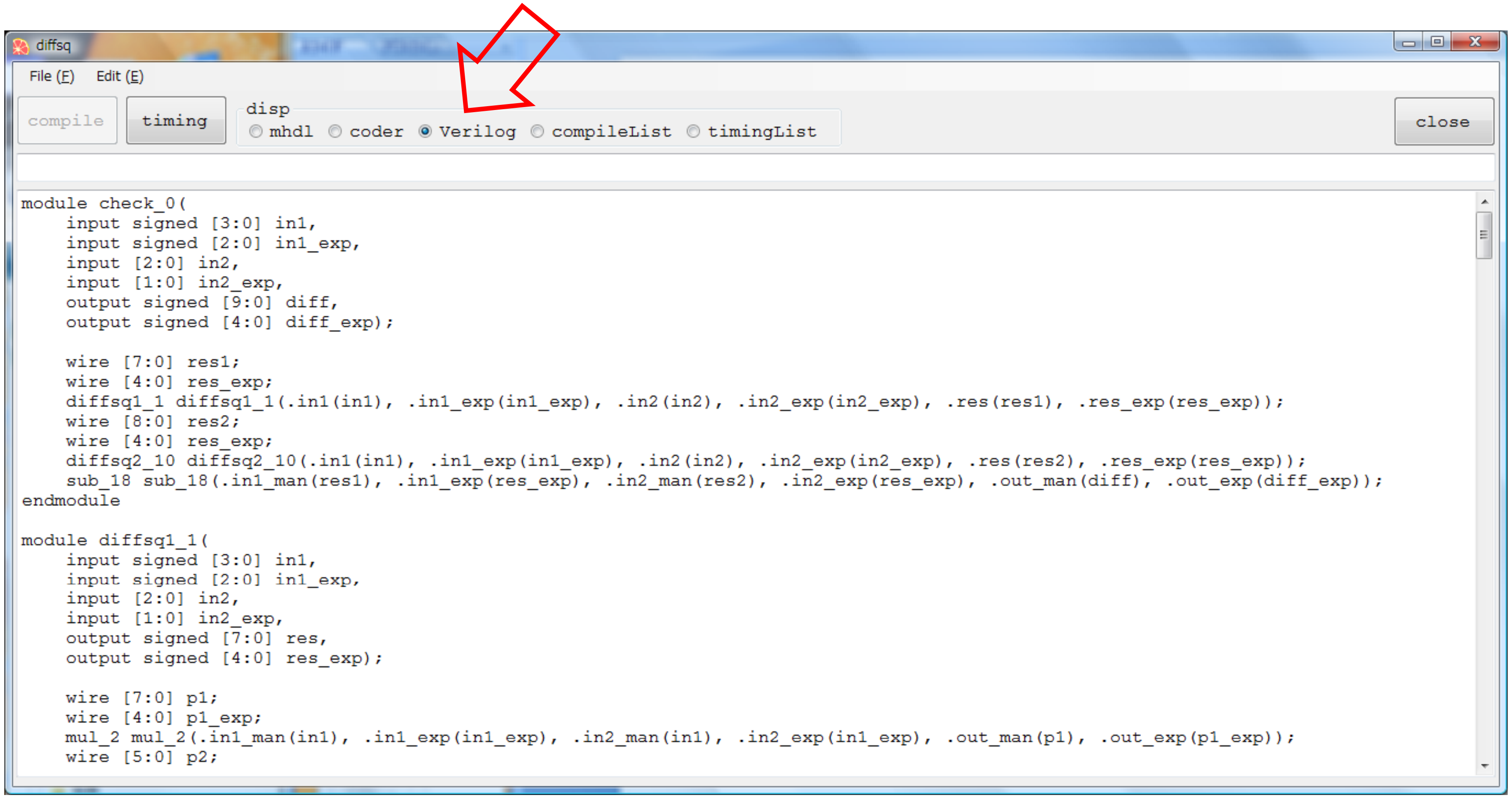
入力の型属性

ワイヤのコード化手続き

コード化結果（ノード）

コンパイル結果の Verilog HDL 表示

“Verilog”ラジオボタンを押すとノードを Verilog HDL 形式に変換して表示する
Verilog HDLに変換された結果を次ページ以降に示す



disp
mhdl coder [✓ Verilog] compileList timingList

```
module check_0(
    input signed [3:0] in1,
    input signed [2:0] in1_exp,
    input [2:0] in2,
    input [1:0] in2_exp,
    output signed [9:0] diff,
    output signed [4:0] diff_exp);

    wire [7:0] res1;
    wire [4:0] res_exp;
    diffsq1_1 diffsq1_1(.in1(in1), .in1_exp(in1_exp), .in2(in2), .in2_exp(in2_exp), .res(res1), .res_exp(res_exp));
    wire [8:0] res2;
    wire [4:0] res_exp;
    diffsq2_10 diffsq2_10(.in1(in1), .in1_exp(in1_exp), .in2(in2), .in2_exp(in2_exp), .res(res2), .res_exp(res_exp));
    sub_18 sub_18(.in1_man(res1), .in1_exp(res_exp), .in2_man(res2), .in2_exp(res_exp), .out_man(diff), .out_exp(diff_exp));
endmodule

module diffsq1_1(
    input signed [3:0] in1,
    input signed [2:0] in1_exp,
    input [2:0] in2,
    input [1:0] in2_exp,
    output signed [7:0] res,
    output signed [4:0] res_exp);

    wire [7:0] p1;
    wire [4:0] p1_exp;
    mul_2 mul_2(.in1_man(in1), .in1_exp(in1_exp), .in2_man(in1), .in2_exp(in1_exp), .out_man(p1), .out_exp(p1_exp));
    wire [5:0] p2;
```

変換結果（その１）

以下の単純な仕様記述が右および次ページに示す長大なVerilogコードに変換される

mhdlコード

```
diffsq1.(res) (in1, in2)
  p1 = mul(in1, in1)
  p2 = mul(in2, in2)
  res = sub(p1, p2)

diffsq2.(res) (in1, in2)
  sum = add(in1, in2)
  diff = sub(in1, in2)
  res = mul(sum, diff)

check.(diff) (in1, in2)
  res1 = diffsq1(in1, in2)
  res2 = diffsq2(in1, in2)
  diff = sub(res1, res2)
```

入力型属性の規定

```
in1[3:0] = range(-8~7)
in1_exp[2:0] = range(-4~3)
in2[2:0] = range(0~7)
in2_exp[1:0] = range(0~3)
```

コーディングの手間もさることながら、適切な型属性を求める工程が大幅に簡素化されている。

```
module check_0(
   input signed [3:0] in1,
   input signed [2:0] in1_exp,
   input [2:0] in2,
   input [1:0] in2_exp,
   output signed [9:0] diff,
   output signed [4:0] diff_exp);

   wire [7:0] res1;
   wire [4:0] res_exp;
   diffsq1_1 diffsq1_1(.in1(in1), .in1_exp(in1_exp), .in2(in2), .
        in2_exp(in2_exp), .res(res1), .res_exp(res_exp));
   wire [8:0] res2;
   wire [4:0] res_exp;
   diffsq2_10 diffsq2_10(.in1(in1), .in1_exp(in1_exp), .in2(in2), .
        in2_exp(in2_exp), .res(res2), .res_exp(res_exp));
   sub_18 sub_18(.in1_man(res1), .in1_exp(res_exp), .in2_man(res2),
        .in2_exp(res_exp), .out_man(diff), .out_exp(diff_exp));
endmodule

module diffsq1_1(
   input signed [3:0] in1,
   input signed [2:0] in1_exp,
   input [2:0] in2,
   input [1:0] in2_exp,
   output signed [7:0] res,
   output signed [4:0] res_exp);

   wire [7:0] p1;
   wire [4:0] p1_exp;
   mul_2 mul_2(.in1_man(in1), .in1_exp(in1_exp), .in2_man(in1),
        .in2_exp(in1_exp), .out_man(p1), .out_exp(p1_exp));
   wire [5:0] p2;
   wire [3:0] p2_exp;
   mul_5 mul_5(.in1_man(in2), .in1_exp(in2_exp), .in2_man(in2),
        .in2_exp(in2_exp), .out_man(p2), .out_exp(p2_exp));
   sub_8 sub_8(.in1_man(p1), .in1_exp(p1_exp), .in2_man(p2), .
        in2_exp(p2_exp), .out_man(res), .out_exp(res_exp));
endmodule

module mul_2(
   input signed [3:0] in1_man,
   input signed [2:0] in1_exp,
   input signed [3:0] in2_man,
   input signed [2:0] in2_exp,
   output signed [7:0] out_man,
   output signed [4:0] out_exp);

   wire [1:0] enc_man;
   encode_3 encode_3(.dat(or_man), .res(enc_man));
   shift_4 shift_4(.dat(product), .cnt(enc_man), .res(out_man));

   wire [7:0] product = in1_man * in2_man;
   wire [3:0] in1_man_abs = in1_man[3] ? -in1_man : in1_man;
   wire [3:0] in2_man_abs = in2_man[3] ? -in2_man : in2_man;
   wire [3:0] or_man = in1_man_abs | in2_man_abs;
   wire [3:0] sum_exp = in1_exp + in2_exp;
   assign out_exp = sum_exp + enc_man;
endmodule
module encode_3(
   input [3:0] dat,
   output [1:0] res);

   wire [1:0] cur2 = |dat[3:2] ? dat[3:2] : dat[1:0];
   wire  cnt2 = |dat[3:2];
   wire  cur1 = cur2[1] ? cur2[1] : cur2[0];
   assign res = {cnt2, cur2[1]};
endmodule
```

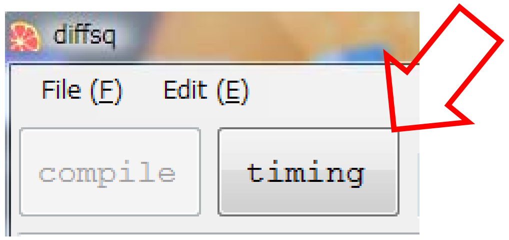
```
module shift_4(
   input signed [7:0] dat,
   input [1:0] cnt,
   output signed [7:0] res);

   wire  cond0 = cnt[0];
   wire [7:0] dat_0 = cond0 ? {{1{dat[7]}}, dat[7:1]} : dat;
   wire  cond1 = cnt[1];
   assign res = cond1 ? {{2{dat_0[7]}}, dat_0[7:2]} : dat_0;
endmodule

module mul_5(
   input [2:0] in1_man,
   input [1:0] in1_exp,
   input [2:0] in2_man,
   input [1:0] in2_exp,
   output [5:0] out_man,
   output [3:0] out_exp);

   wire [1:0] enc_man;
   encode_6 encode_6(.dat(or_man), .res(enc_man));
   shift_7 shift_7(.dat(product), .cnt(enc_man), .res(out_man));

   wire [5:0] product = in1_man * in2_man;
   wire [2:0] or_man = in1_man | in2_man;
   wire [2:0] sum_exp = in1_exp + in2_exp;
   assign out_exp = sum_exp + enc_man;
endmodule

module encode_6(
   input [2:0] dat,
   output [1:0] res);

   wire [1:0] cur2 = dat[2] ? dat[2] : dat[1:0];
   wire  cnt2 = dat[2];
   wire  cur1 = cur2[1] ? cur2[1] : cur2[0];
   assign res = {cnt2, cur2[1]};
endmodule

module shift_7(
   input [5:0] dat,
   input [1:0] cnt,
   output [5:0] res);

   wire  cond0 = cnt[0];
   wire [5:0] dat_0 = cond0 ? {1'h0, dat[5:1]} : dat;
   wire  cond1 = cnt[1];
   assign res = cond1 ? {2'h0, dat_0[5:2]} : dat_0;
endmodule

module sub_8(
   input signed [7:0] in1_man,
   input signed [4:0] in1_exp,
   input [5:0] in2_man,
   input [3:0] in2_exp,
   output signed [7:0] out_man,
   output signed [4:0] out_exp);

   wire [7:0] adj_man1;
   shift_9 shift_9(.dat(man1), .cnt(abs_dexp), .res(adj_man1));

   wire [5:0] dexp = in1_exp - in2_exp;
   wire  sgn_dexp = dexp[5];
   assign out_exp = sgn_dexp ? in1_exp : in2_exp;
   wire [4:0] abs_dexp = dexp[5] ? -dexp : dexp;
   wire [7:0] man1 = sgn_dexp ? in1_man : in2_man;
   wire [7:0] man2 = sgn_dexp ? in2_man : in1_man;
   assign out_man = adj_man1 - man2;
endmodule
```

変換結果（その2）

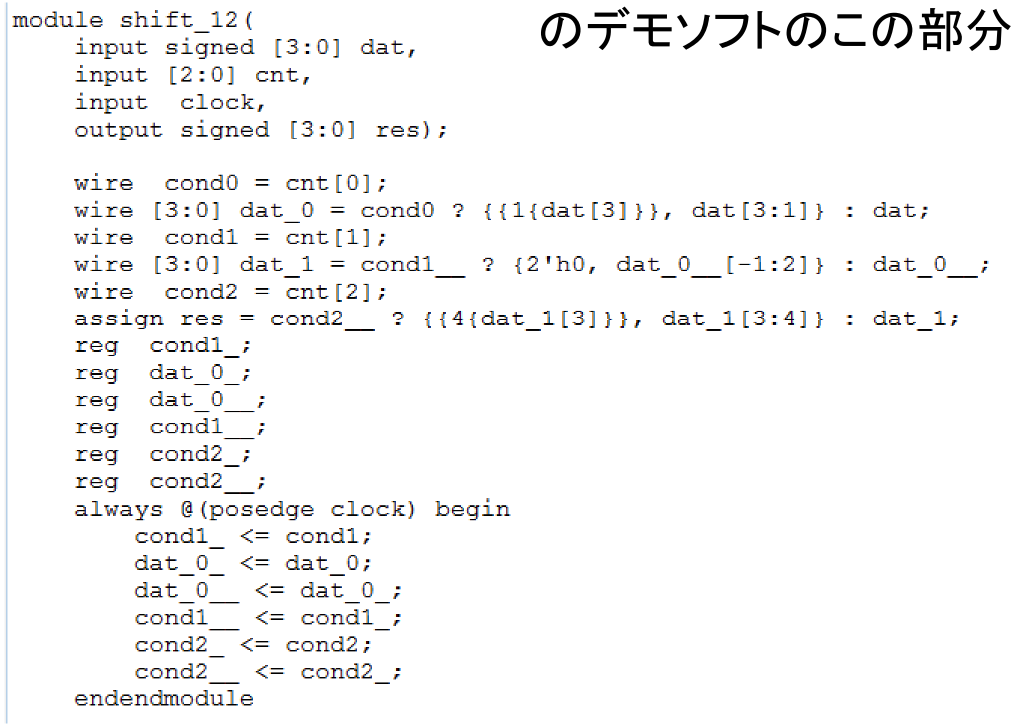
```
module shift_9(
    input signed [7:0] dat,
    input [4:0] cnt,
    output signed [7:0] res);

    wire  cond0 = cnt[0];
    wire [7:0] dat_0 = cond0 ? {{1{dat[7]}}, dat[7:1]} : dat;
    wire  cond1 = cnt[1];
    wire [7:0] dat_1 = cond1 ? {{2{dat_0[7]}}, dat_0[7:2]} : dat_0;
    wire  cond2 = cnt[2];
    wire [7:0] dat_2 = cond2 ? {{4{dat_1[7]}}, dat_1[7:4]} : dat_1;
    wire  cond3 = cnt[3];
    wire [7:0] dat_3 = cond3 ? {{8{dat_2[7]}}, dat_2[7:8]} : dat_2;
    wire  cond4 = cnt[4];
    assign res = cond4 ? {{16{dat_3[7]}}, dat_3[7:16]} : dat_3;
endmodule

module diffsq2_10(
    input signed [3:0] in1,
    input signed [2:0] in1_exp,
    input [2:0] in2,
    input [1:0] in2_exp,
    output signed [8:0] res,
    output signed [4:0] res_exp);

    wire [4:0] sum;
    wire [2:0] sum_exp;
    add_11 add_11(.in1_man(in1), .in1_exp(in1_exp), .in2_man(in2), .
        in2_exp(in2_exp), .out_man(sum), .out_exp(sum_exp));
    wire [4:0] diff;
    wire [2:0] diff_exp;
    sub_13 sub_13(.in1_man(in1), .in1_exp(in1_exp), .in2_man(in2),
        .in2_exp(in2_exp), .out_man(diff), .out_exp(diff_exp));
    mul_15 mul_15(.in1_man(sum), .in1_exp(sum_exp), .in2_man(diff),
        .in2_exp(diff_exp), .out_man(res), .out_exp(res_exp));
endmodule

module add_11(
    input signed [3:0] in1_man,
    input signed [2:0] in1_exp,
    input [2:0] in2_man,
    input [1:0] in2_exp,
    output signed [4:0] out_man,
    output signed [2:0] out_exp);

    wire [3:0] adj_man1;
    shift_12 shift_12(.dat(man1), .cnt(abs_dexp), .res(adj_man1));

    wire [3:0] dexp = in1_exp - in2_exp;
    wire  sgn_dexp = dexp[3];
    assign out_exp = sgn_dexp ? in1_exp : in2_exp;
    wire [2:0] abs_dexp = dexp[3] ? -dexp : dexp;
    wire [3:0] man1 = sgn_dexp ? in1_man : in2_man;
    wire [3:0] man2 = sgn_dexp ? in2_man : in1_man;
    assign out_man = adj_man1 + man2;
endmodule

module shift_12(
    input signed [3:0] dat,
    input [2:0] cnt,
    output signed [3:0] res);

    wire  cond0 = cnt[0];
    wire [3:0] dat_0 = cond0 ? {{1{dat[3]}}, dat[3:1]} : dat;
    wire  cond1 = cnt[1];
    wire [3:0] dat_1 = cond1 ? {{2{dat_0[3]}}, dat_0[3:2]} : dat_0;
    wire  cond2 = cnt[2];
    assign res = cond2 ? {{4{dat_1[3]}}, dat_1[3:4]} : dat_1;
endmodule

module sub_13(
    input signed [3:0] in1_man,
    input signed [2:0] in1_exp,
    input [2:0] in2_man,
    input [1:0] in2_exp,
    output signed [4:0] out_man,
    output signed [2:0] out_exp);

    wire [3:0] adj_man1;
    shift_14 shift_14(.dat(man1), .cnt(abs_dexp), .res(adj_man1));

    wire [3:0] dexp = in1_exp - in2_exp;
    wire  sgn_dexp = dexp[3];
    assign out_exp = sgn_dexp ? in1_exp : in2_exp;
    wire [2:0] abs_dexp = dexp[3] ? -dexp : dexp;
    wire [3:0] man1 = sgn_dexp ? in1_man : in2_man;
    wire [3:0] man2 = sgn_dexp ? in2_man : in1_man;
    assign out_man = adj_man1 - man2;
endmodule

module shift_14(
    input signed [3:0] dat,
    input [2:0] cnt,
    output signed [3:0] res);

    wire  cond0 = cnt[0];
    wire [3:0] dat_0 = cond0 ? {{1{dat[3]}}, dat[3:1]} : dat;
    wire  cond1 = cnt[1];
    wire [3:0] dat_1 = cond1 ? {{2{dat_0[3]}}, dat_0[3:2]} : dat_0;
    wire  cond2 = cnt[2];
    assign res = cond2 ? {{4{dat_1[3]}}, dat_1[3:4]} : dat_1;
endmodule

module mul_15(
    input signed [4:0] in1_man,
    input signed [2:0] in1_exp,
    input signed [4:0] in2_man,
    input signed [2:0] in2_exp,
    output signed [8:0] out_man,
    output signed [4:0] out_exp);

    wire [2:0] enc_man;
    encode_16 encode_16(.dat(or_man), .res(enc_man));
    shift_17 shift_17(.dat(product), .cnt(enc_man), .res(out_man));

    wire [8:0] product = in1_man * in2_man;
    wire [4:0] in1_man_abs = in1_man[4] ? -in1_man : in1_man;
    wire [3:0] in2_man_abs = in2_man[4] ? -in2_man : in2_man;
    wire [4:0] or_man = in1_man_abs | {1'h0, in2_man_abs};
    wire [3:0] sum_exp = in1_exp + in2_exp;
    assign out_exp = sum_exp + enc_man;
endmodule

module encode_16(
    input [4:0] dat,
    output [2:0] res);

    wire [3:0] cur4 = dat[4] ? dat[4] : dat[3:0];
    wire  cnt4 = dat[4];
    wire [1:0] cur2 = |cur4[3:2] ? cur4[3:2] : cur4[1:0];
    wire [1:0] cnt2 = {cnt4, |cur4[3:2]};
    wire  cur1 = cur2[1] ? cur2[1] : cur2[0];
    assign res = {cnt2, cur2[1]};
endmodule

module shift_17(
    input signed [8:0] dat,
    input [2:0] cnt,
    output signed [8:0] res);

    wire  cond0 = cnt[0];
    wire [8:0] dat_0 = cond0 ? {{1{dat[8]}}, dat[8:1]} : dat;
    wire  cond1 = cnt[1];
    wire [8:0] dat_1 = cond1 ? {{2{dat_0[8]}}, dat_0[8:2]} : dat_0;
    wire  cond2 = cnt[2];
    assign res = cond2 ? {{4{dat_1[8]}}, dat_1[8:4]} : dat_1;
endmodule

module sub_18(
    input signed [7:0] in1_man,
    input signed [4:0] in1_exp,
    input signed [8:0] in2_man,
    input signed [4:0] in2_exp,
    output signed [9:0] out_man,
    output signed [4:0] out_exp);

    wire [8:0] adj_man1;
    shift_19 shift_19(.dat(man1), .cnt(abs_dexp), .res(adj_man1));

    wire [5:0] dexp = in1_exp - in2_exp;
    wire  sgn_dexp = dexp[5];
    assign out_exp = sgn_dexp ? in1_exp : in2_exp;
    wire [4:0] abs_dexp = dexp[5] ? -dexp : dexp;
    wire [8:0] man1 = sgn_dexp ? in1_man : in2_man;
    wire [8:0] man2 = sgn_dexp ? in2_man : in1_man;
    assign out_man = adj_man1 - man2;
endmodule

module shift_19(
    input signed [8:0] dat,
    input [4:0] cnt,
    output signed [8:0] res);

    wire  cond0 = cnt[0];
    wire [8:0] dat_0 = cond0 ? {{1{dat[8]}}, dat[8:1]} : dat;
    wire  cond1 = cnt[1];
    wire [8:0] dat_1 = cond1 ? {{2{dat_0[8]}}, dat_0[8:2]} : dat_0;
    wire  cond2 = cnt[2];
    wire [8:0] dat_2 = cond2 ? {{4{dat_1[8]}}, dat_1[8:4]} : dat_1;
    wire  cond3 = cnt[3];
    wire [8:0] dat_3 = cond3 ? {{8{dat_2[8]}}, dat_2[8]} : dat_2;
    wire  cond4 = cnt[4];
    assign res = cond4 ? {{16{dat_3[8]}}, dat_3[8:16]} : dat_3;
endmodule
```

タイミング制約の解決

“timing”ボタンを押すとタイミング制約を解決するようにレジスタが挿入される



“Verilog”ラジオボタンを押すことによりVerilog HDLのソースコードを表示させることができる。
“clock”入力、レジスタ宣言およびクロックに同期する代入文が形成されていることがわかる。(なお、現在のデモソフトのこの部分にはバグがある様子である)



```
module shift_12(
    input signed [3:0] dat,
    input [2:0] cnt,
    input  clock,
    output signed [3:0] res);

    wire  cond0 = cnt[0];
    wire [3:0] dat_0 = cond0 ? {{1{dat[3]}}, dat[3:1]} : dat;
    wire  cond1 = cnt[1];
    wire [3:0] dat_1 = cond1__ ? {2'h0, dat_0__[-1:2]} : dat_0__;
    wire  cond2 = cnt[2];
    assign res = cond2__ ? {{4{dat_1[3]}}, dat_1[3:4]} : dat_1;
    reg  cond1_;
    reg  dat_0_;
    reg  dat_0__;
    reg  cond1__;
    reg  cond2_;
    reg  cond2__;
    always @(posedge clock) begin
        cond1_ <= cond1;
        dat_0_ <= dat_0;
        dat_0__ <= dat_0_;
        cond1__ <= cond1_;
        cond2_ <= cond2;
        cond2__ <= cond2_;
    endendmodule
```

まとめ

- デモソフトを用いることで、今回紹介したアルゴリズムが期待したとおりに動作することが確認された
- 抽象論理仕様記述言語“mhdl”を用いると、Verilog HDLでは複雑となる論理仕様を簡潔に記述することが可能である
- この信号処理論理設計支援システムは、浮動小数点を取扱い可能であることを特徴としているが、誤差を含まない整数や誤差を含む固定小数点数などの取り扱いも可能である
- タイミング制約の解決のためのレジスタ挿入機能の動作も確認されているが、正しく挿入されているかについては疑問がある
- 今後は、本ソフトウエアの機能拡充と動作検証をおこない、信号処理論理設計支援システムとして製品化したい