# INTEGRATED DESIGN AND VERIFICATION: AN OUNCE OF PREVENTION IS WORTH A POUND OF CURE

CARL SEGER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

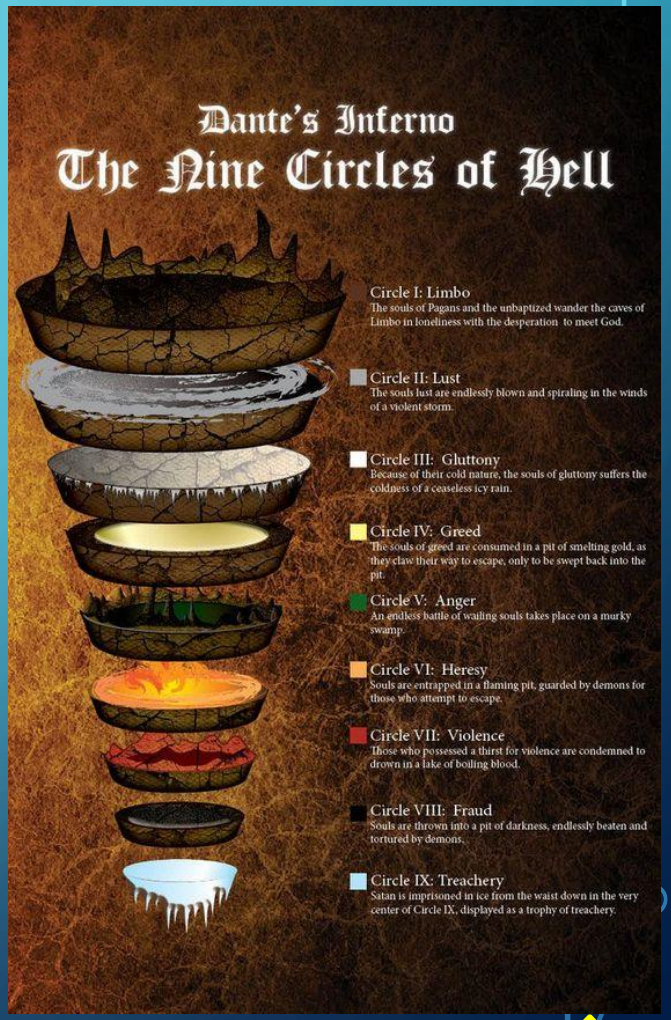
CHALMERS UNIVERSITY OF TECHNOLOGY, GOTHENBURG, SWEDEN

SECARL@CHALMERS.SE

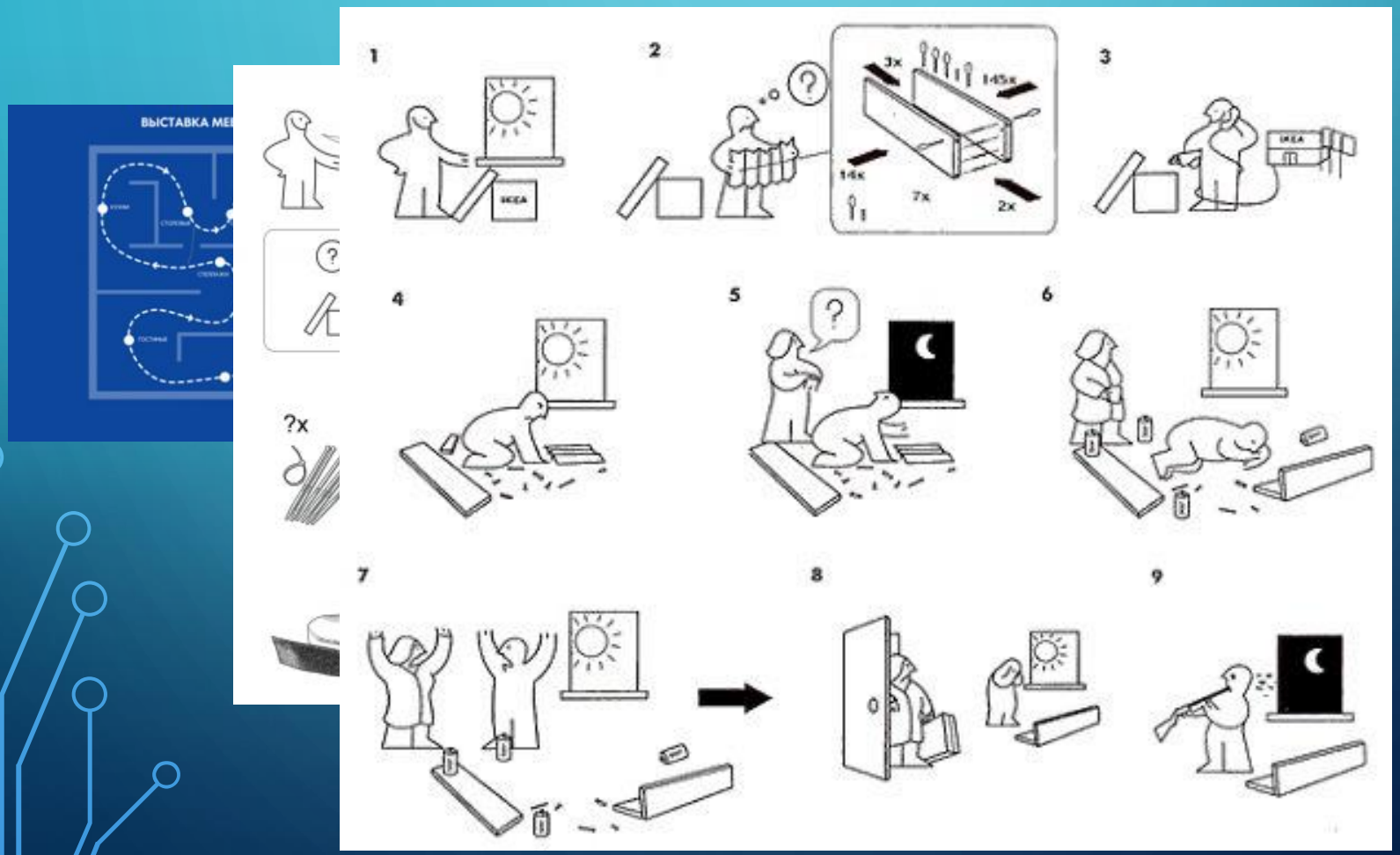
SEPTEMBER 6, 2022

# BAD NEWS

- To verify a HW design is:
  - Hard (60-80% of ASIC design effort)
  - Time consuming
- To debug a HW design:
  - Is even worse!
- To debug combined SW/HW:
  - Is cause of short life span…
  - ..and lots of grey hair!



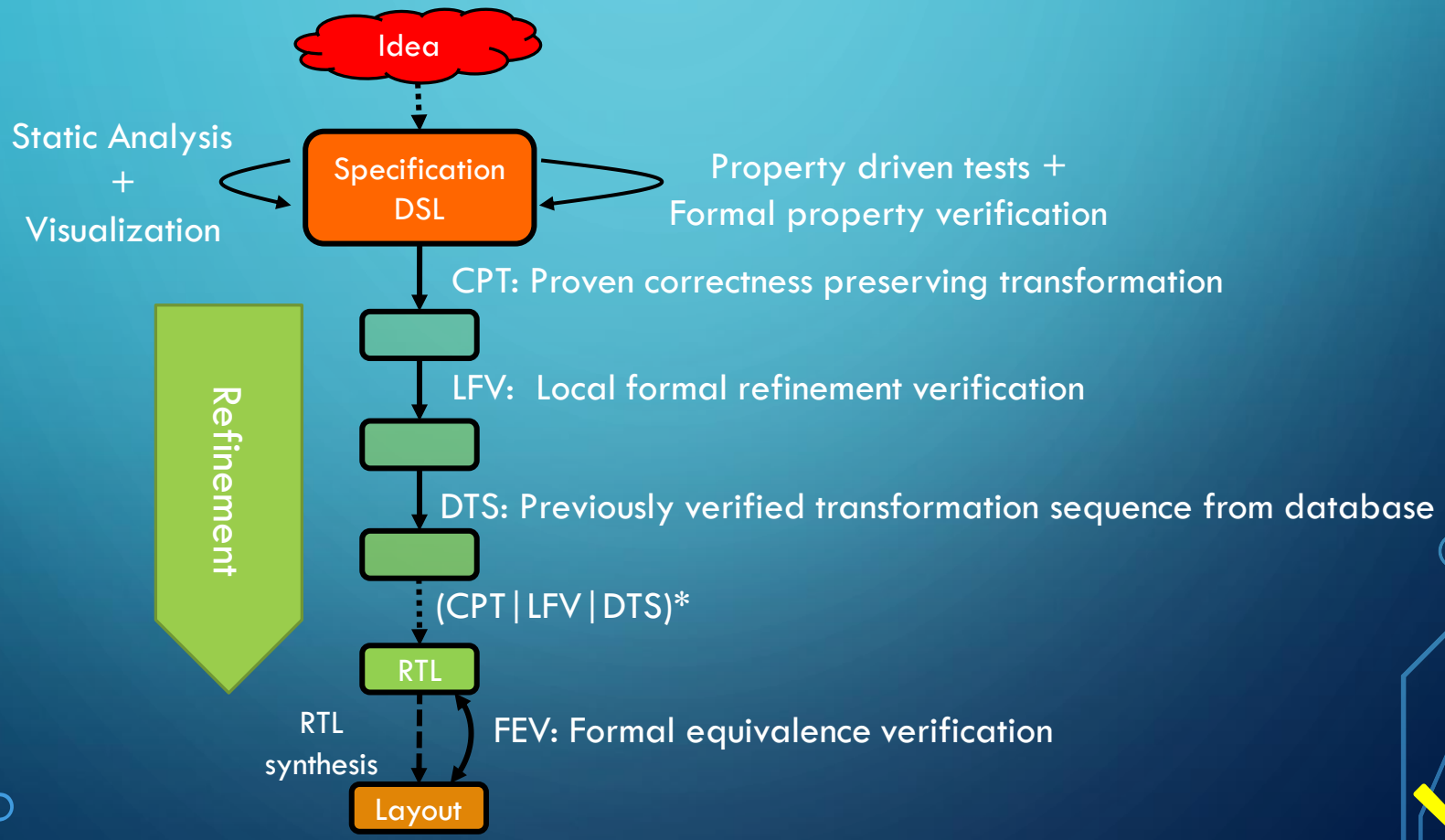
# GOOD NEWS! It could be worse…



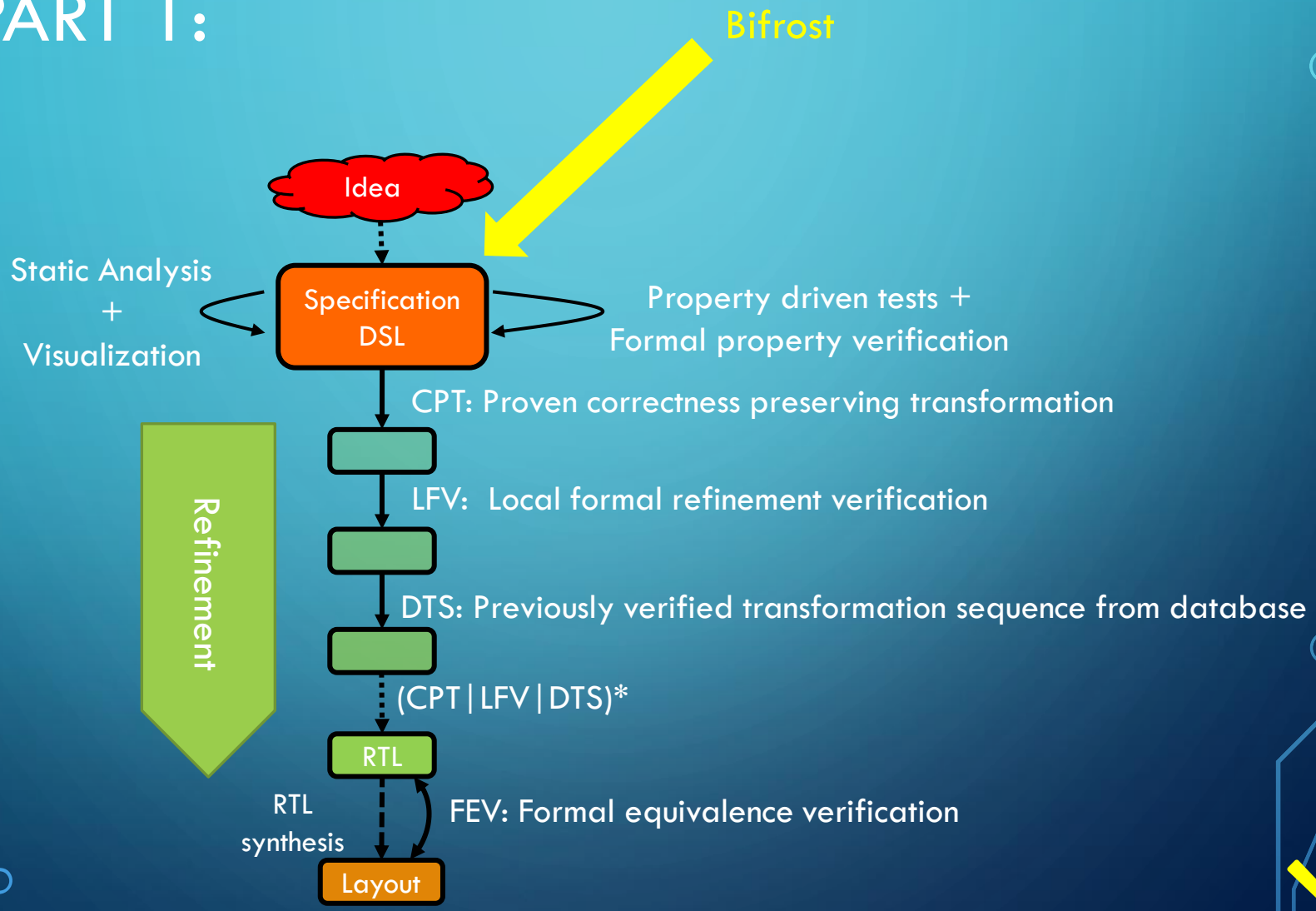
# RESEARCH HYPOTHESES

- Hypothesis 1:
  - There are way more SW engineer than HW design engineers.
  - Therefore, specification language should be “similar” to traditional SW languages.
- Hypothesis 2:
  - Ad-hoc/post design verification is not feasible
  - Therefore, design and verification must be tightly integrated
- Hypothesis 3:
  - Design is a highly interactive activity
  - Therefore, the design environment must be a highly interactive with fast interactions, much visualization and with plenty of guidance as well as re-use of earlier work.

# THOR PROJECT: BIG PICTURE



# PART 1:


Bifrost
Idea
Static Analysis
+
Visualization
Specification
DSL
Property driven tests +
Formal property verification
CPT: Proven correctness preserving transformation
Refinement
LFV: Local formal refinement verification
DTS: Previously verified transformation sequence from database
(CPT | LFV | DTS)*
RTL
RTL
synthesis
FEV: Formal equivalence verification
Layout

# HIGH-LEVEL SPEC. LANG. GOALS

- Separate “what” from “how”
  - The goal is to succinctly specify what is needed.
- Allow algorithmic specifications (“software like”)
  - Most natural specification for many problems.
- Replace timing with protocols
  - Isolate the specifier from the subtleties of communication.
- Make validation as easy as possible
  - Clean (simple) semantics, strong typing, visualization, …

# BIFROST

- Aimed at iterative algorithms and interaction between multiple modules
- Imperative language that is compiled into hardware
  - Both control machine and data path control is created.
- Protocols between units are user selectable
  - Can change protocol by changing 1-2 lines.
- Protocols include not only functional control but also power/voltage control.

English/pseudo code

```
for each chunk
    create a 64-entry messa                                    s
    (The initial values in                                     tions zero them
    copy chunk into first 16 words w[0..15] of the message schedule array

    Extend the first 16 words into the remaining 48 words w[16..63] of the message s
    for i from 16 to 63
        s0 := (w[i-15] rightrotate  7) xor (w[i-15] rightrotate 18) xor (w[i-15] rig
        s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] rights
        w[i] := w[i-16] + s0 + w[i-7] + s1

    Initialize working variables to current hash value:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    f := h5
    g := h6
    h := h7

    Compression function main loop:
    for i from 0 to 63
        S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
        ch := (e and f) xor ((not e) and g)
        temp1 := h + S1 + ch + k[i] + w[i]
        S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
        maj := (a and b) xor (a and c) xor (b and c)
        temp2 := S0 + maj

        h := g
        g := f
        f := e
        e := d + temp1
        d := c
        c := b
        b := a
        a := temp1 + temp2

    Add the compressed chunk to the current hash value:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h

Produce the final hash value (big-endian):
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 app
```

Bifrost

```
    // Initialize Wmem
    for(i = 0; i < 16; i = i+1) {
        cur_addr = read_addr + zx(i);
        w = do mem_read cur_addr;
        do w_write i w;
    }
    for(i = 16; i < 64; i = i+1) {
        w15 = do w_read (i-15);
        w2  = do w_read (i-2);
        w16 = do w_read (i-16);
        w7  = do w_read (i-7);
        s0  = (rrot7 w15) ^ (rrot18 w15) ^ (rshift3 w15);
        s1  = (rrot17 w2) ^ (rrot19 w2) ^ (rshift10 w2);
        sum1 = do add w16 s0;
        sum2 = do add w7 s1;
        sum =  do add sum1 sum2;
        do w_write i sum;
    }
    a = h0;
    b = h1;
    c = h2;
    d = h3;
    e = h4;
    f = h5;
    g = h6;
    h = h7;
    // Main compression loop
    for(i = 0; i < 64; i = i+1) {
        wi = do w_read i;
        ki = do k_lookup i;
        S1 = (rrot6 e) ^ (rrot11 e) ^ (rrot25 e);
        ch = (e & f)^(~e & g);
        t4 = do add h S1;
        t5 = do add ch ki;
        t6 = do add t4 wi;
        temp1 = do add t5 t6;
        S0 = (rrot2 a) ^ (rrot13 a) ^ (rrot22 a);
        maj = (a & b) ^ (a & c) ^ (b & c);
        temp2 = do add S0 maj;
        h = g;
        g = f;
        f = e;
        e = do add d temp1;
        d = c;
        c = b;
        b = a;
        a = do add temp1 temp2;
    }
    // Update hashes
    h0 = do add h0 a;
    h1 = do add h1 b;
    h2 = do add h2 c;
    h3 = do add h3 d;
    h4 = do add h4 e;
    h5 = do add h5 f;
    h6 = do add h6 g;
    h7 = do add h7 h;
```

Source: https://en.wikipedia.org/wiki/SHA-2

# UNITS & PROTOCOL DECLARATIONS

```
name sha256;

protocol fourphase alwayson;

#include "types.inc"

// Macros to access HFL functions
define zx = "\\z. ZX z";
// Right shift by constant
define rshift3  = "\\w. do_rshift w '3";
define rshift10 = "\\w. do_rshift w '10";
// Rotate by constant
define rrot2  = "\\w. do_rrot w '2";
define rrot6  = "\\w. do_rrot w '6";
define rrot7  = "\\w. do_rrot w '7";
define rrot11 = "\\w. do_rrot w '11";
define rrot13 = "\\w. do_rrot w '13";
define rrot17 = "\\w. do_rrot w '17";
define rrot18 = "\\w. do_rrot w '18";
define rrot19 = "\\w. do_rrot w '19";
define rrot22 = "\\w. do_rrot w '22";
define rrot25 = "\\w. do_rrot w '25";

// Memory interface
action mem_read:MemRead   provided by external via pulseecho alwayson;
action mem_write:MemRead  provided by external via pulseecho alwayson;

action w_read:Wread   provided by external via pulseecho alwayson;
action w_write:Wwrite provided by external via pulseecho alwayson;

action k_lookup:Ktbl provided by external via combinational alwayson;

define N = 10;      // Max number of big adders to use
action add:Add[N]    provided by "add" via combinational alwayson;

subroutine main : read_addr:addr -> res:signature
{
#include "declarations.inc"
#include "constants.inc"
    // Initialize Wmem
```

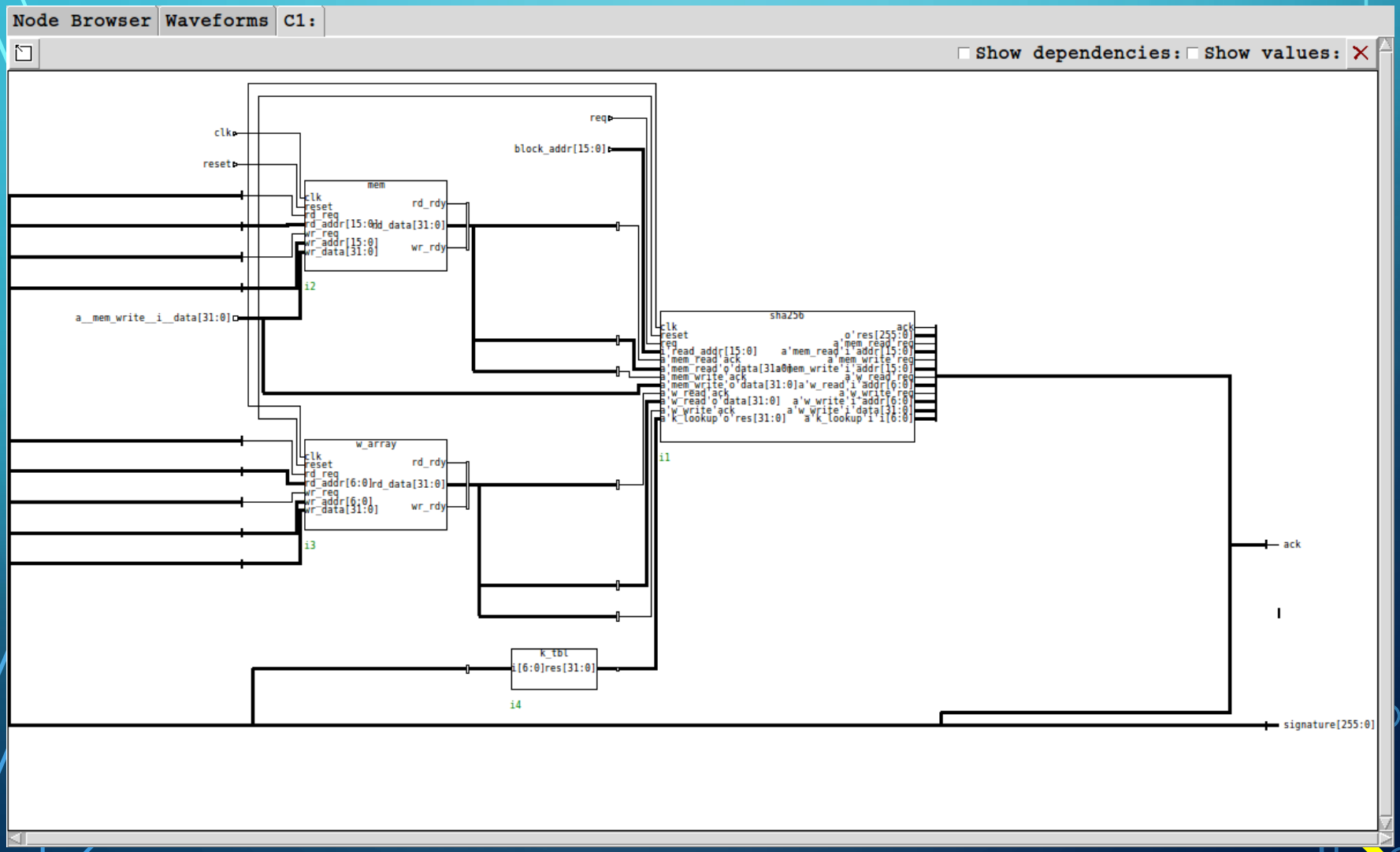
Protocol this unit talks

Protocols the subunits talk

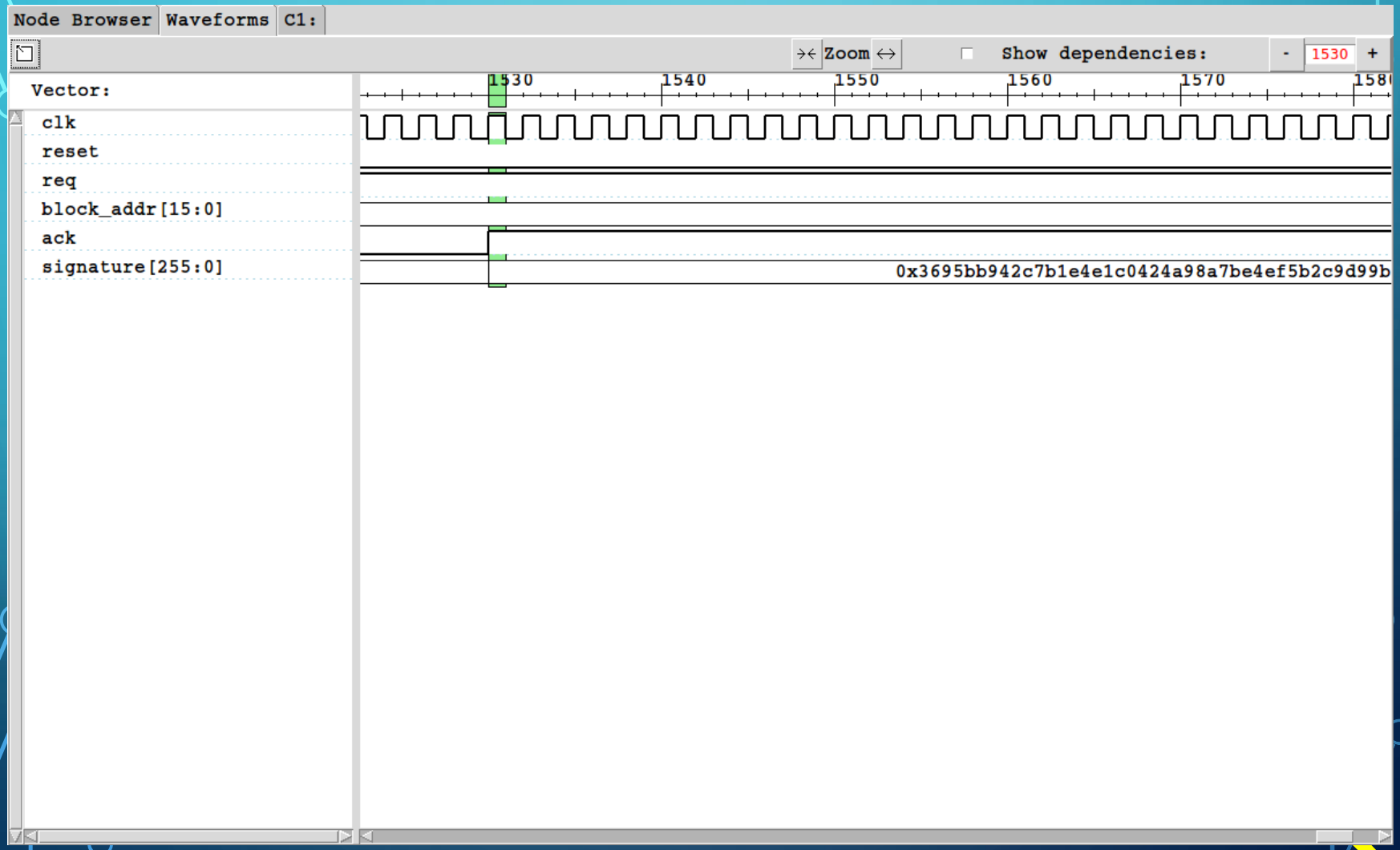
Number of units available to the scheduler

# DEMO OF BIFROST CAPTURE OF SHA256

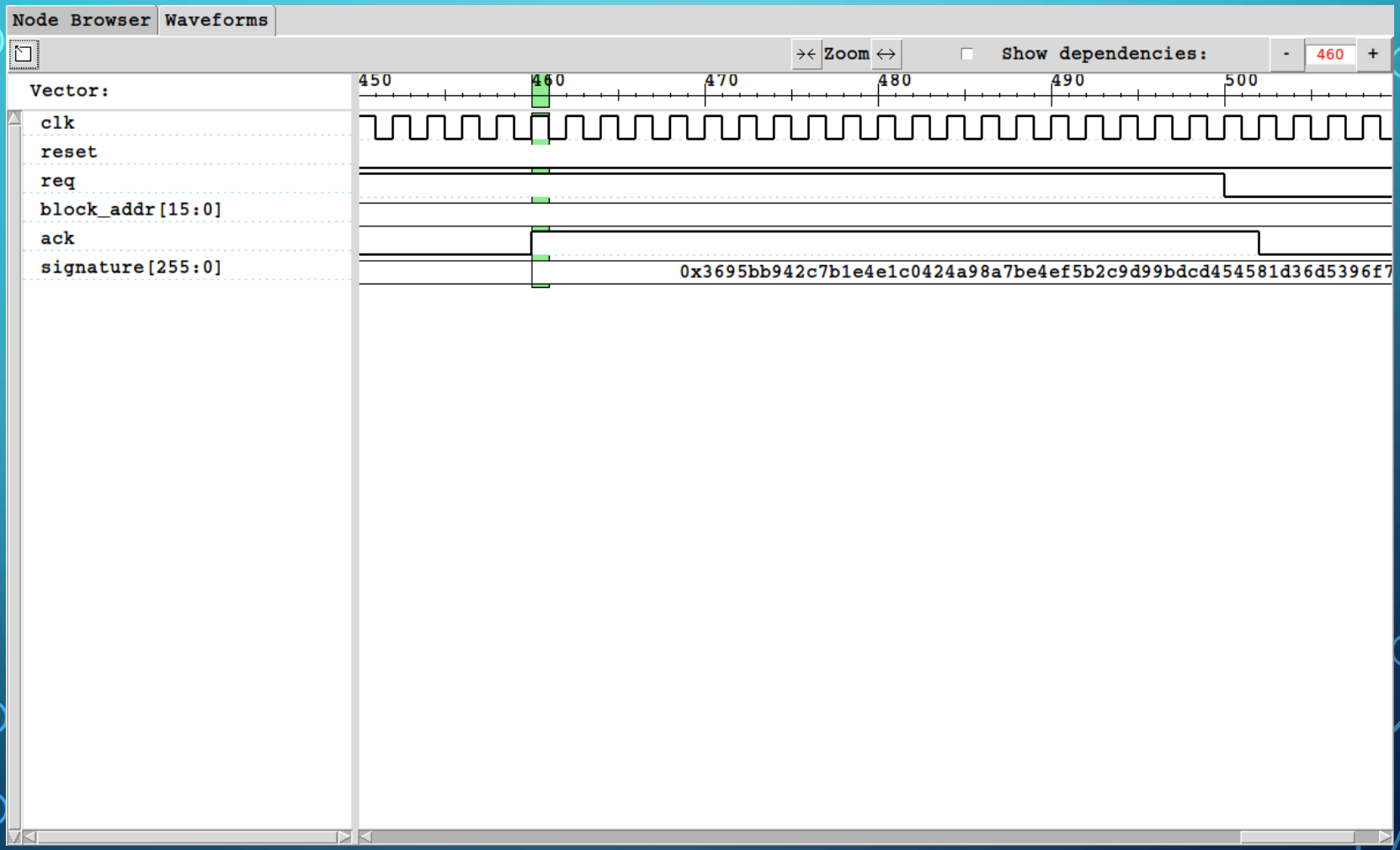
# Slow SHA256 version



# Slow SHA256 version



# Fast SHA256 version

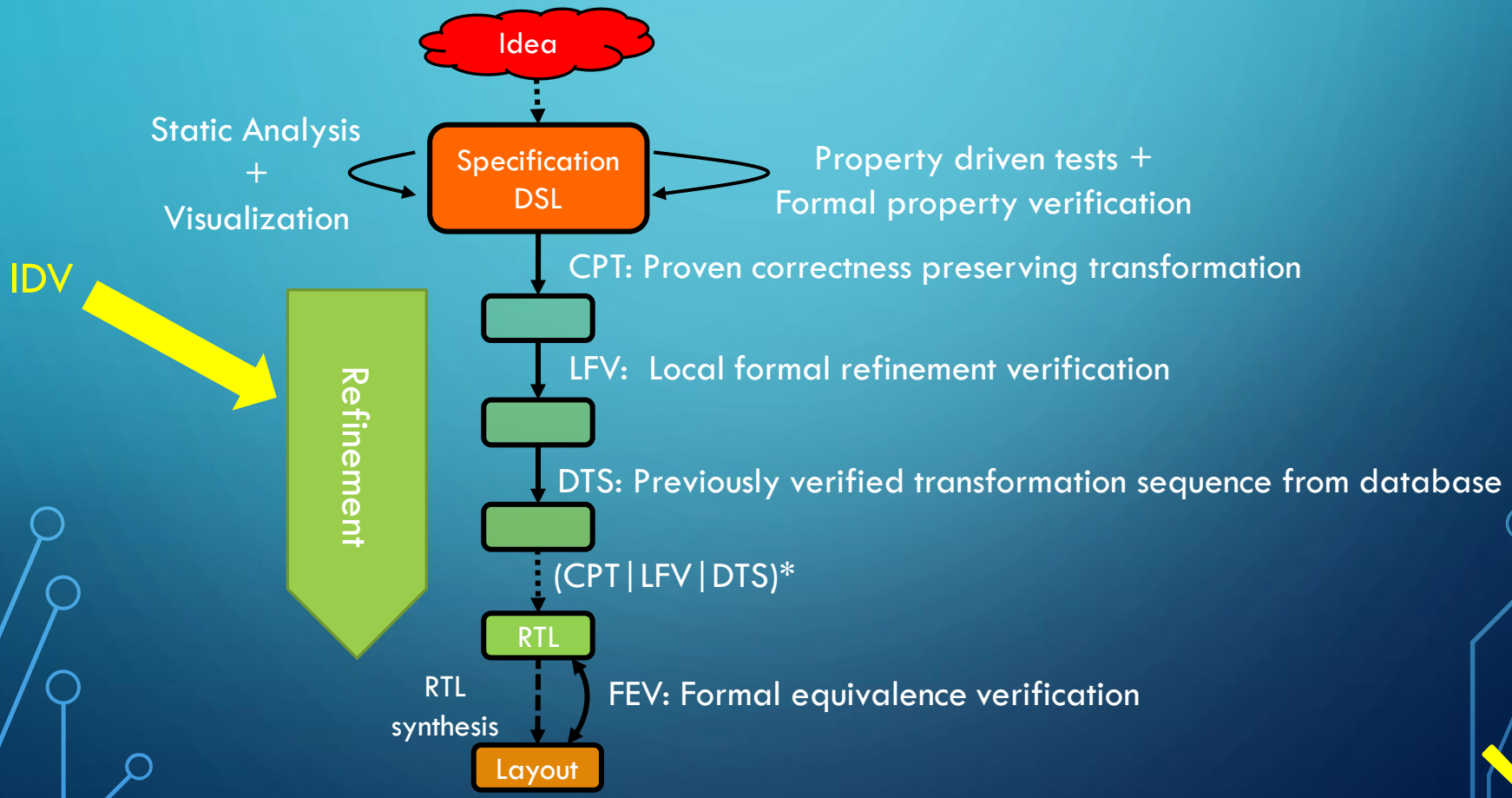


# EXAMPLE OF RESULTS: SHA256

| Name | Reg.file #rd-ports | Constant memory #rd-ports | # adders | Memory protocol | Cycles |
|---|---|---|---|---|---|
| slow | 1 | 1 | 1 | Pulse | 765 |
| medium | 1 | 1 | 7 | Pulse | 327 |
| 4phase | 1 | 1 | 7 | 4-phase | 342 |
| fast | 2 | 2 | 14 | Pulse | 230 |

NOTE: Only the number of units and protocols were changed*.

# PART II:

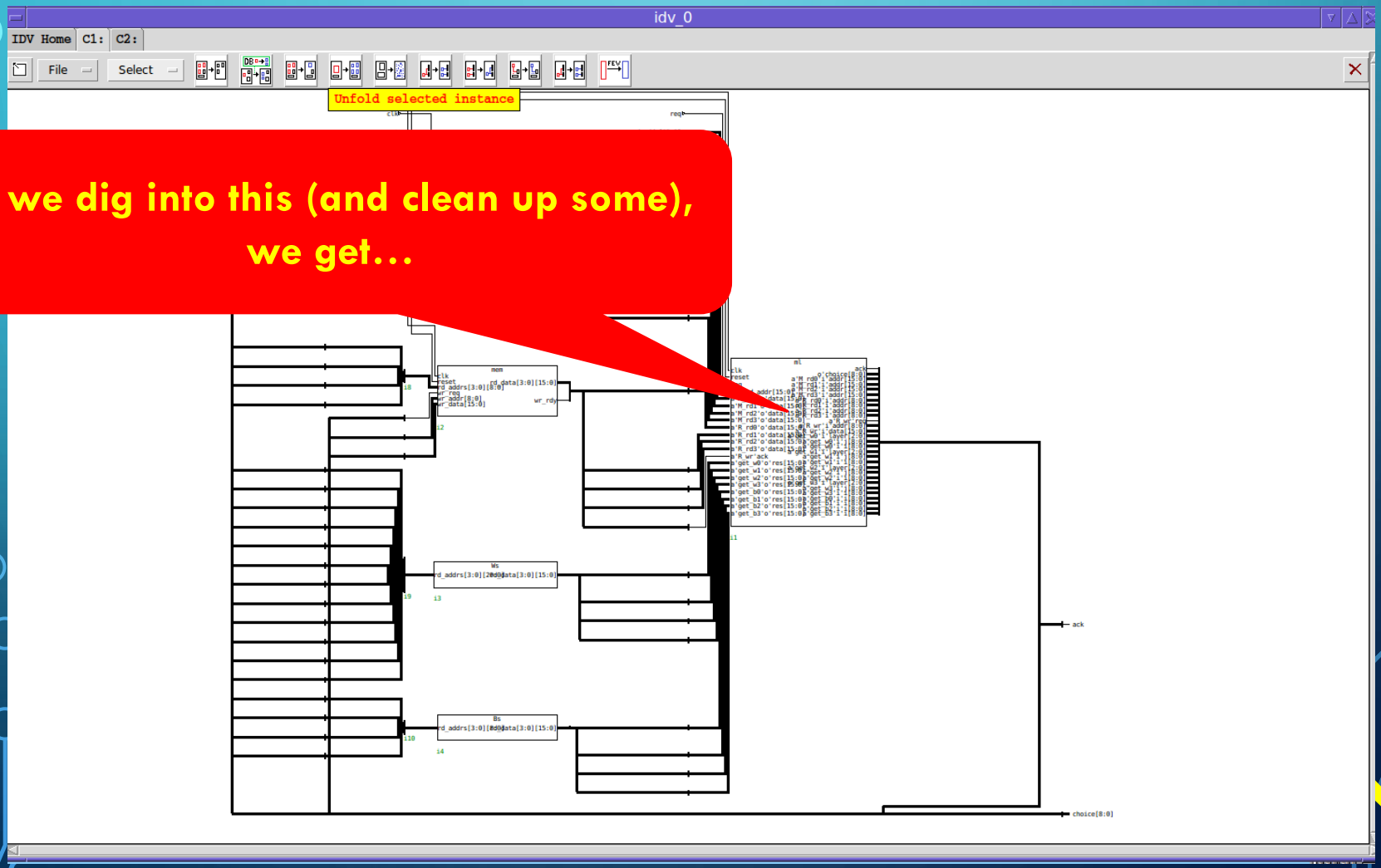

Idea
Static Analysis
+
Visualization
Specification
DSL
Property driven tests +
Formal property verification
IDV
CPT: Proven correctness preserving transformation
Refinement
LFV: Local formal refinement verification
DTS: Previously verified transformation sequence from database
(CPT | LFV | DTS)*
RTL
RTL
synthesis
FEV: Formal equivalence verification
Layout

# SIMPLE NEURAL NETWORK EVALUATOR FOR LOW-PWR IOT

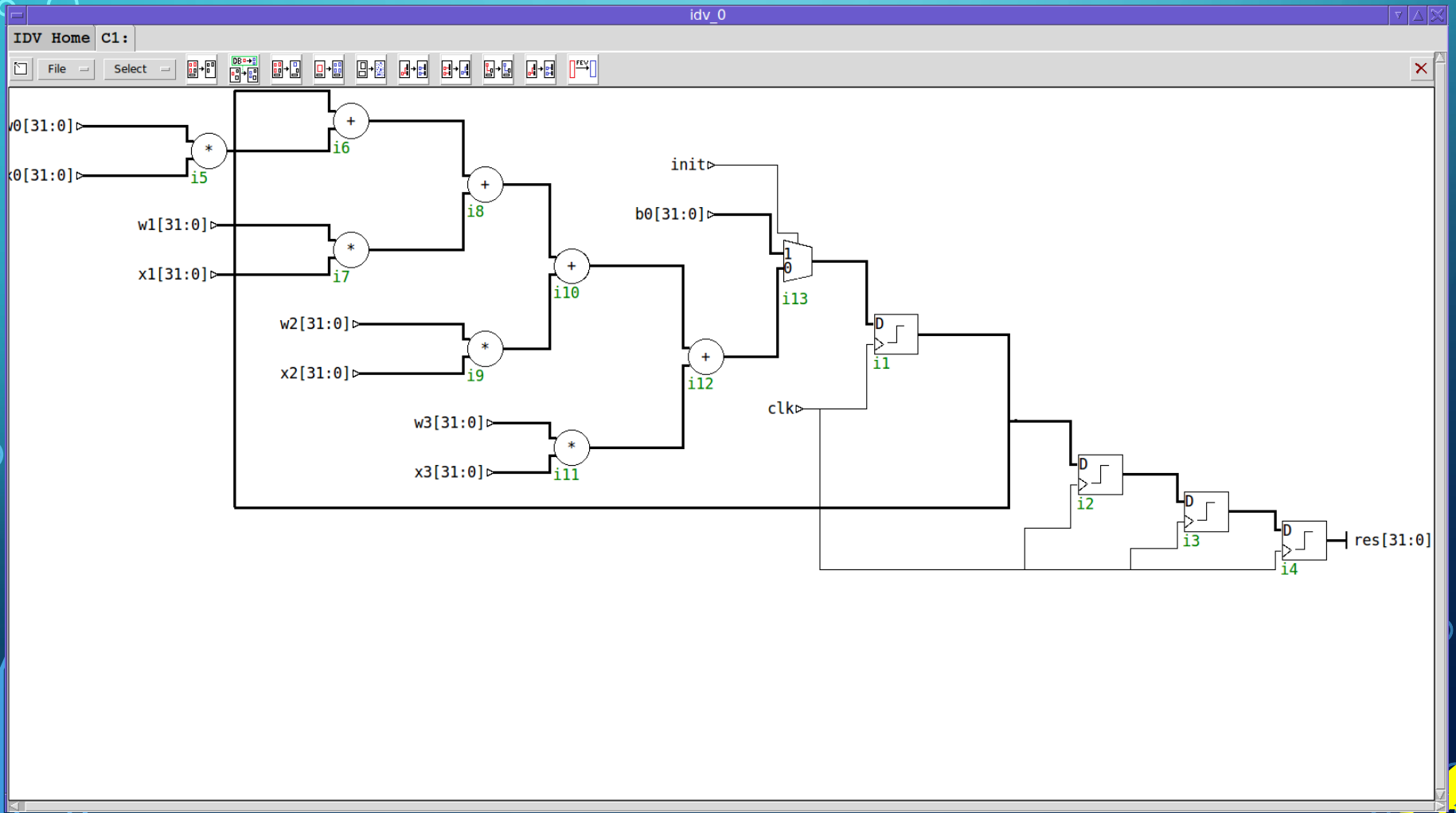
```
name ml;
protocol fourphase alwayson;

#include "types.inc"

// Macros to access HFL functions
define zx = "\\z. ZX z";

define N = 4; // Number of external units to use
define WID = 28;
define HT = 28;
define HIDDEN_LAYERS = 3;
define DEPTH = 16;
define OUTPUTS = 10;

// External unit interfaces
action M_rd:MRead[N] provided by external via combinational alwayson;
action R_rd:RRead[N] provided by external via combinational alwayson;
action R_wr:RWrite   provided by external via pulseecho alwayson;
action get_w:Wtbl[N] provided by external via combinational alwayson;
action get_b:Btbl[N] provided by external via combinational alwayson;
action act:Act       provided by "act" via combinational alwayson;
action add:Add[N]    provided by "add" via combinational alwayson;
action mul:Mul[N]     provided by "mul" via combinational alwayson;

subroutine main : read_addr:addr -> choice:lindex
{
#include "declarations.inc"
```

```
    // Layer 1
    for(l = 0; l < DEPTH; l = l+1) {
        sum = do get_b l;
        for(x = 0; x < WID; x = x+1) {
            for(y = 0; y < HT; y = y+1) {
                j = zx (y*WID+x);
                a0 = do M_rd (read_addr+(zx j));
                w = do get_w 1 j l;
                sum = do add sum (do mul w a0);
            }
        }
        res = do relu sum;
        do R_wr l res;
    }

    // For hidden layers
    N = HIDDEN_LAYERS + 2;
    for(layer = 2; layer < N; layer = layer+1) {
        for(l = 0; l < DEPTH; l = l+1) {
            sum = do get_b (((zx layer)-1)*DEPTH+l);
            for(j = 0; j < DEPTH; j = j+1) {
                aj = do R_rd (((zx layer)-2)*DEPTH+j);
                w  = do get_w layer j l;
                sum = do add sum (do mul w aj);
            }
            res = do relu sum;
            do R_wr (((zx layer)-1)*DEPTH+l) res;
        }
    }

    // Output layer: determine max
    max = 0;
    choice = 0;
    for(l = 0; l < OUTPUTS; l = l+1) {
        sum = do get_b ((N-1)*DEPTH+l);
        for(j = 0; j < DEPTH; j = j+1) {
            aj = do R_rd ((N-2)*DEPTH+j);
            w  = do get_w N j l;
            sum = do add sum (do mul w aj);
        }
        res = do relu sum;
        if( res > max ) {
            choice = l;
            max = res;
        }
    }
    return;
}
```

# RESULT OF BIFROST



# CORE DATAPATH FROM BIFROST



# DEMO OF IDV

IDV Home

Search:

Select database: DB_demo

Model class: Imported

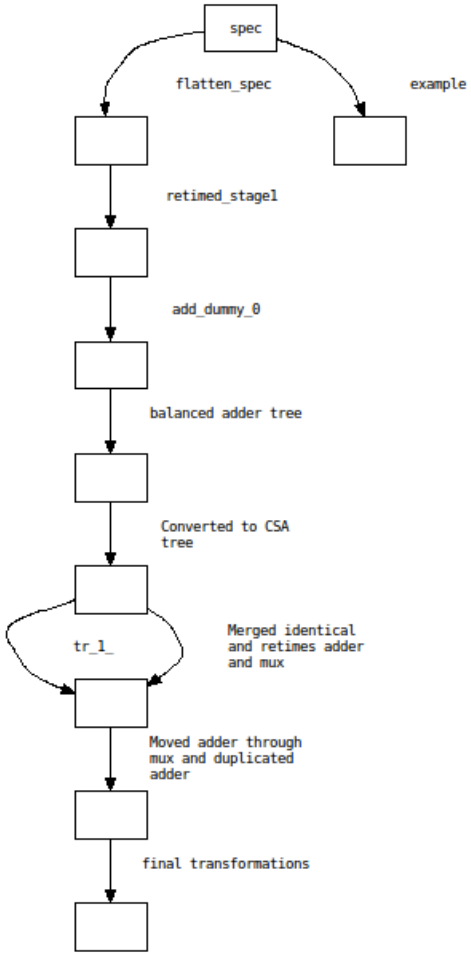
Pattern: *

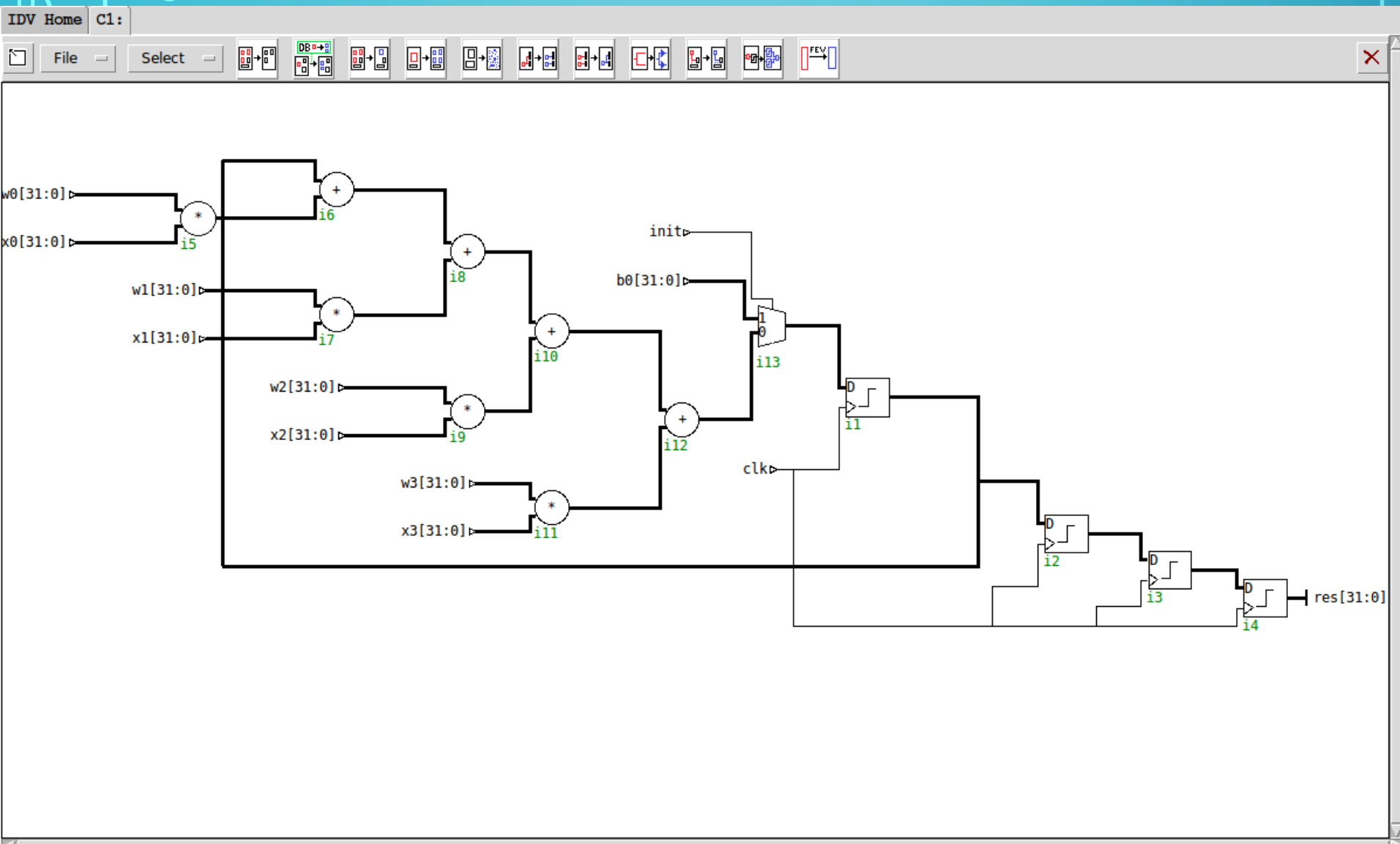
Refresh

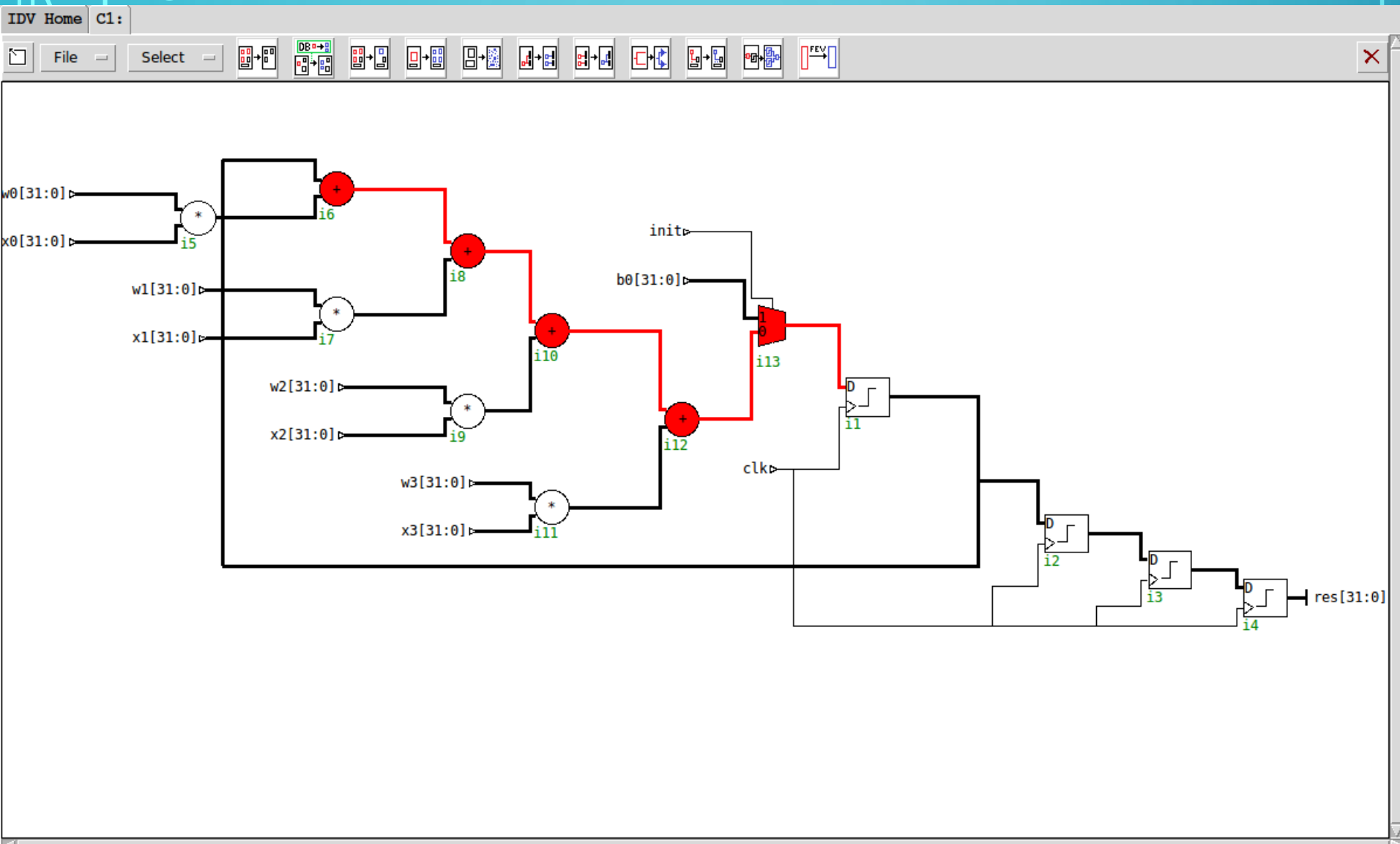
spec

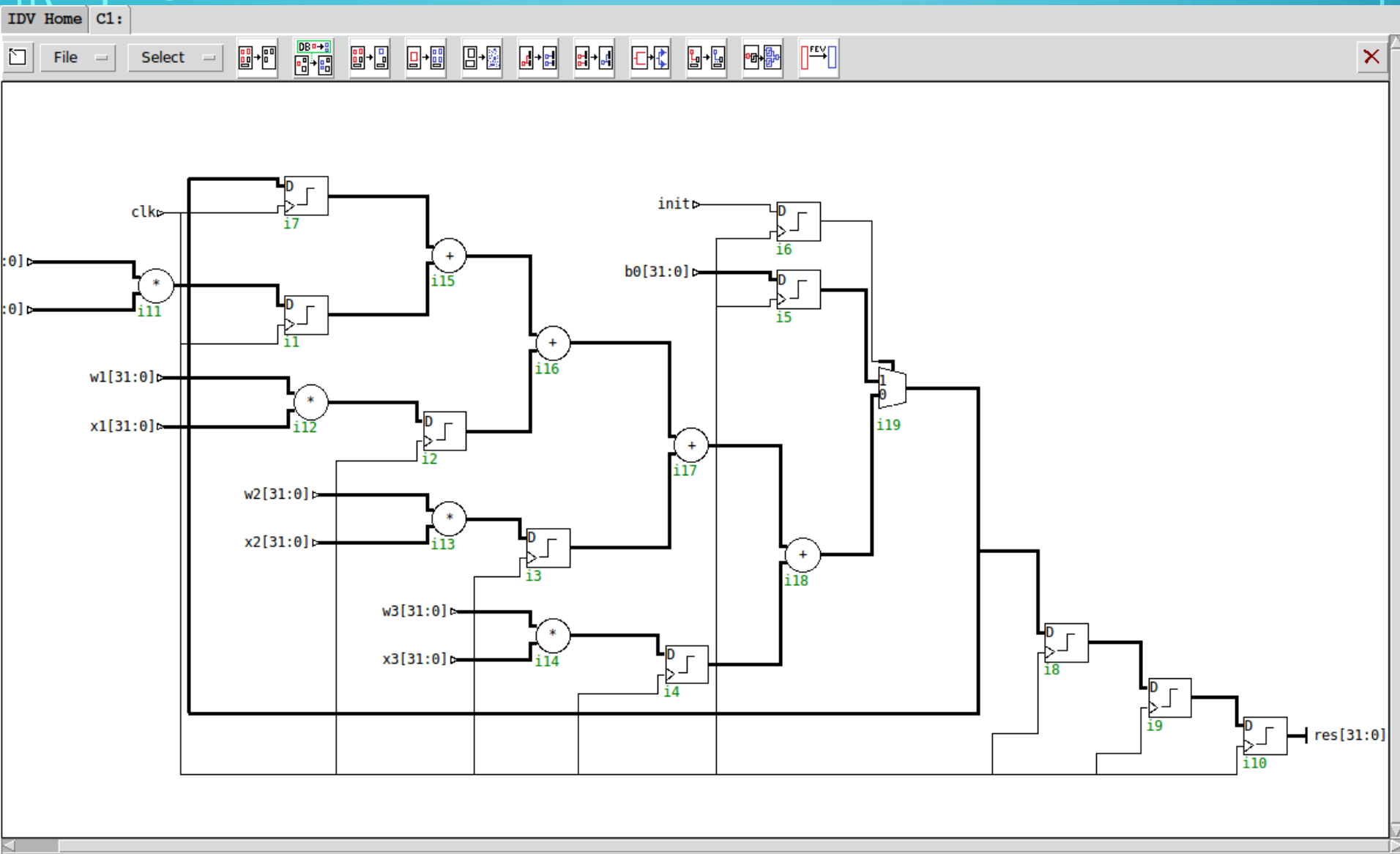
Exit | Save | Import model | New transform

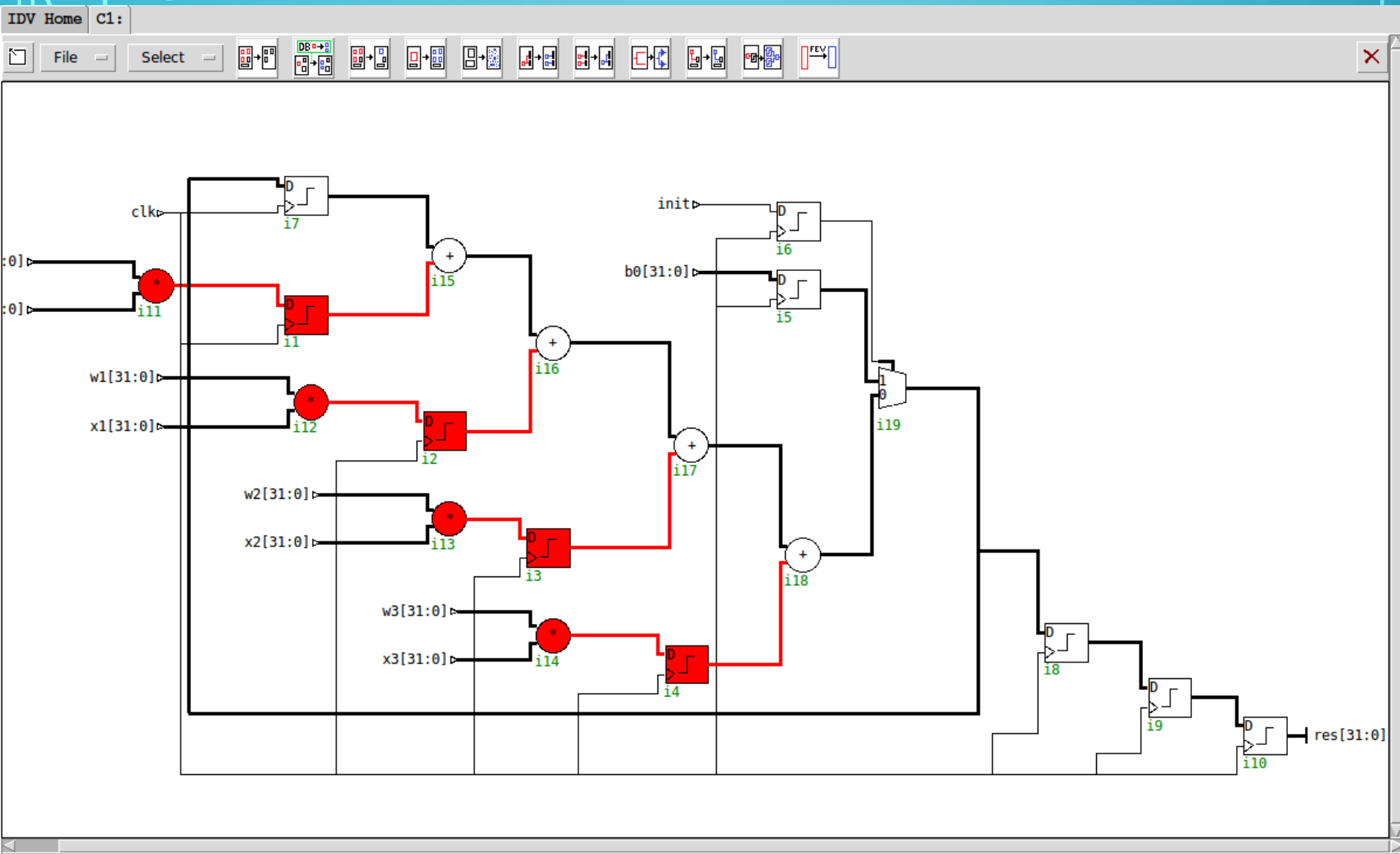
Show anon model names ☐ Show transformation name ■

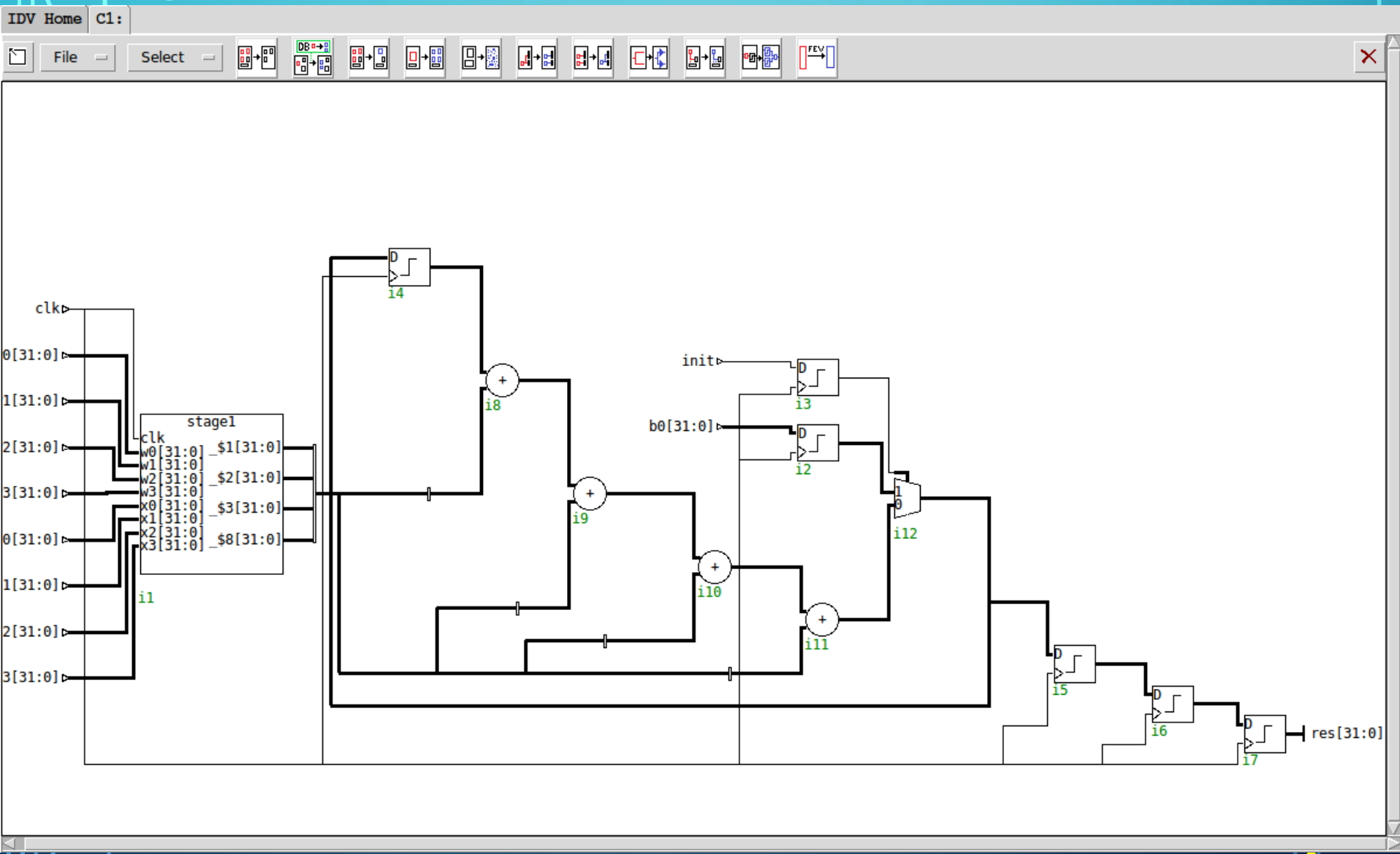


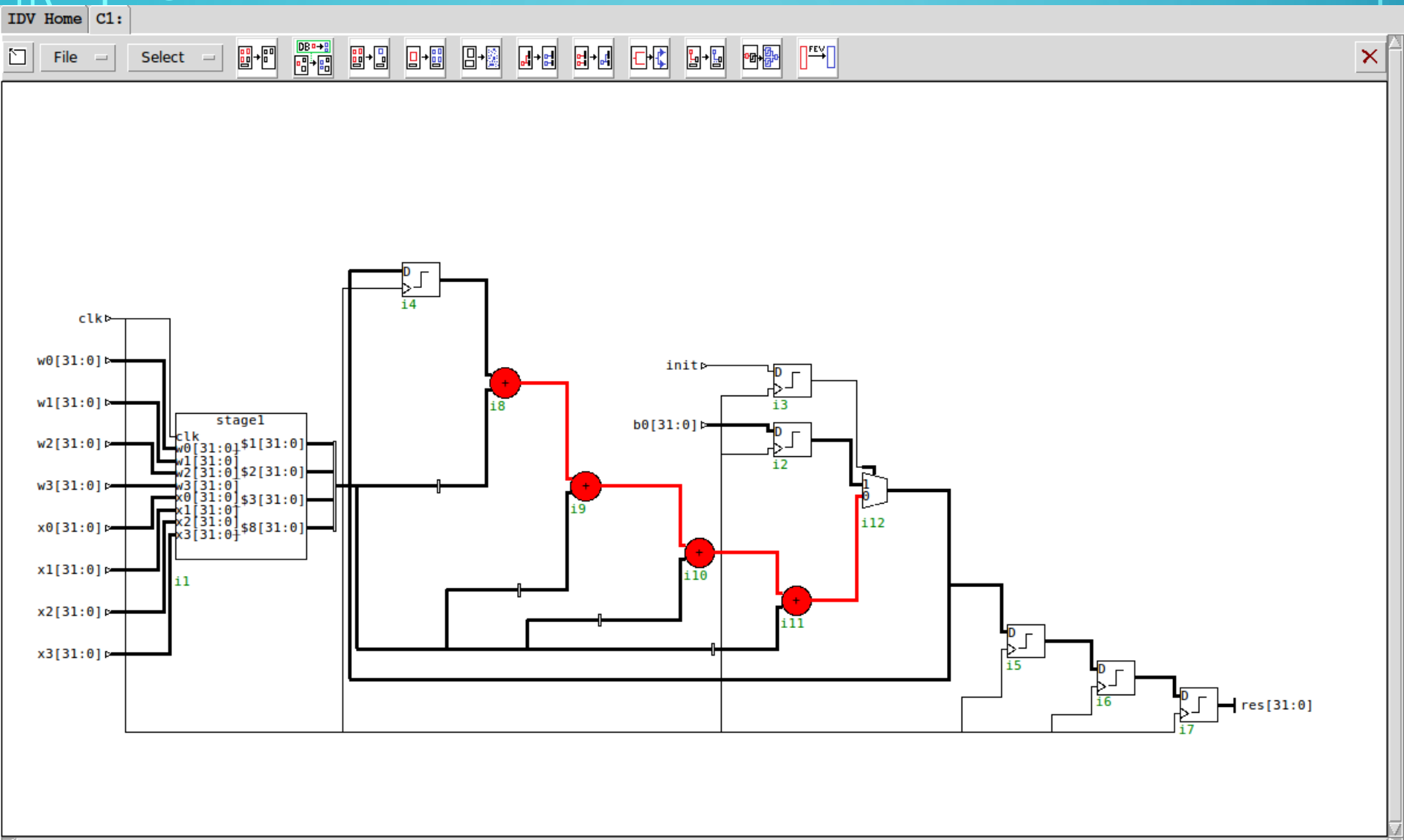

IDV Home
C1:
File
Select
w0[31:0]
x0[31:0]
i5
i6
w1[31:0]
x1[31:0]
i7
i8
w2[31:0]
x2[31:0]
i9
i10
w3[31:0]
x3[31:0]
i11
i12
init
b0[31:0]
i13
clk
i1
i2
i3
i4
res[31:0]

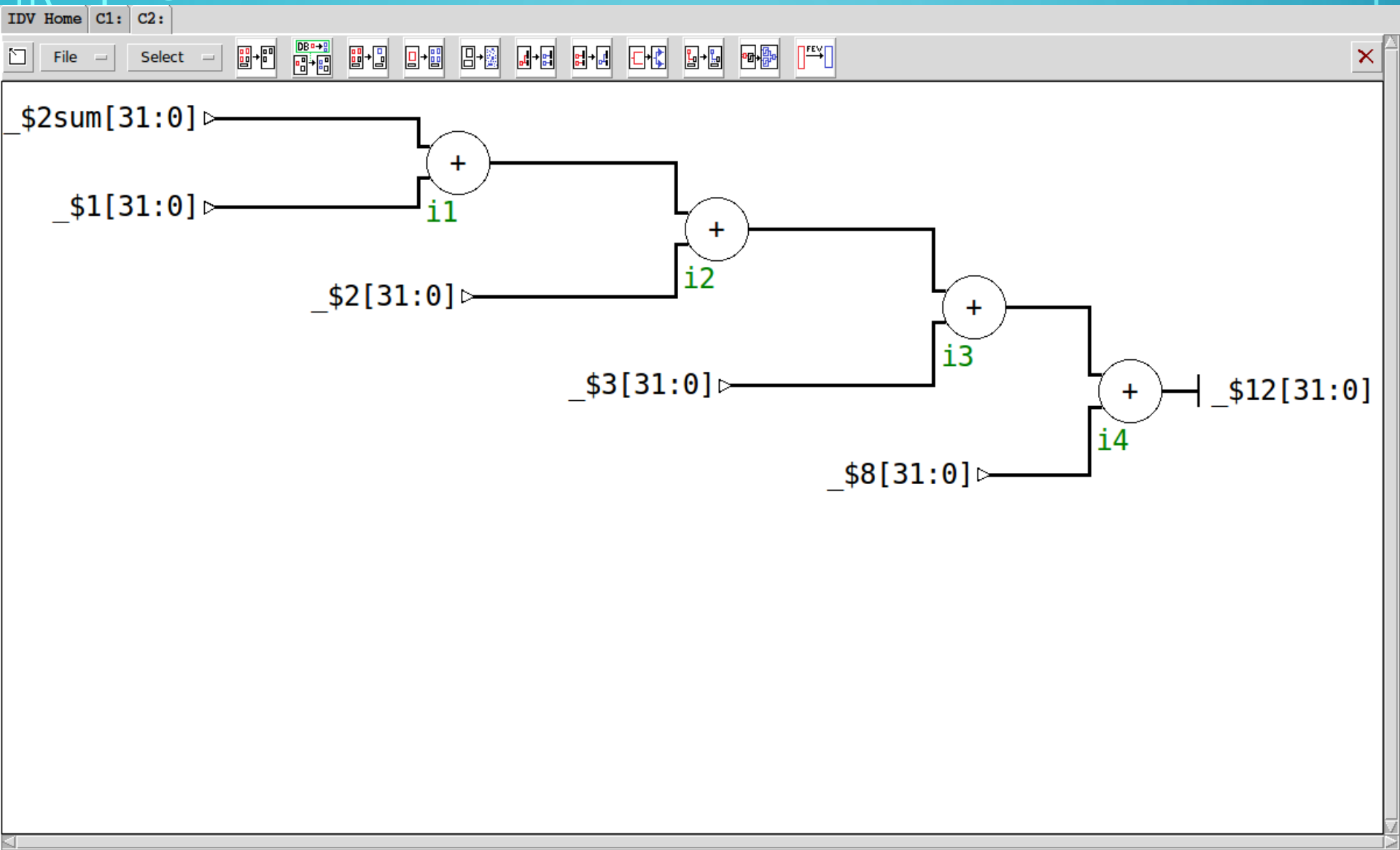

IDV Home
C1:
File
Select
w0[31:0]
x0[31:0]
i5
i6
w1[31:0]
x1[31:0]
i7
i8
w2[31:0]
x2[31:0]
i9
i10
w3[31:0]
x3[31:0]
i11
i12
init
b0[31:0]
i13
clk
i1
i2
i3
i4
res[31:0]

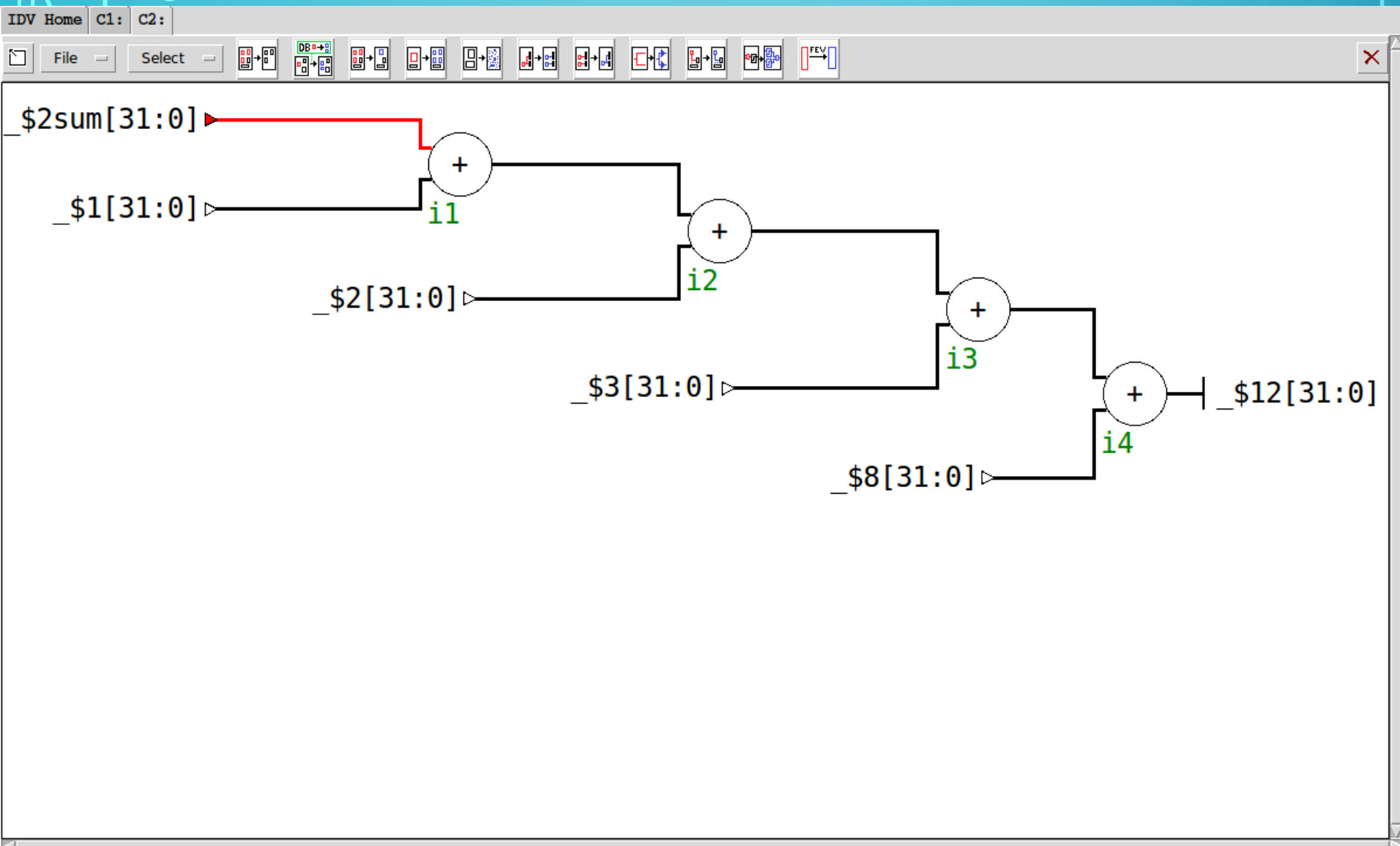

IDV Home
C1:
File
Select
clk
:0]
:0]
w1[31:0]
x1[31:0]
w2[31:0]
x2[31:0]
w3[31:0]
x3[31:0]
init
b0[31:0]
res[31:0]
i1
i2
i3
i4
i5
i6
i7
i8
i9
i10
i11
i12
i13
i14
i15
i16
i17
i18
i19

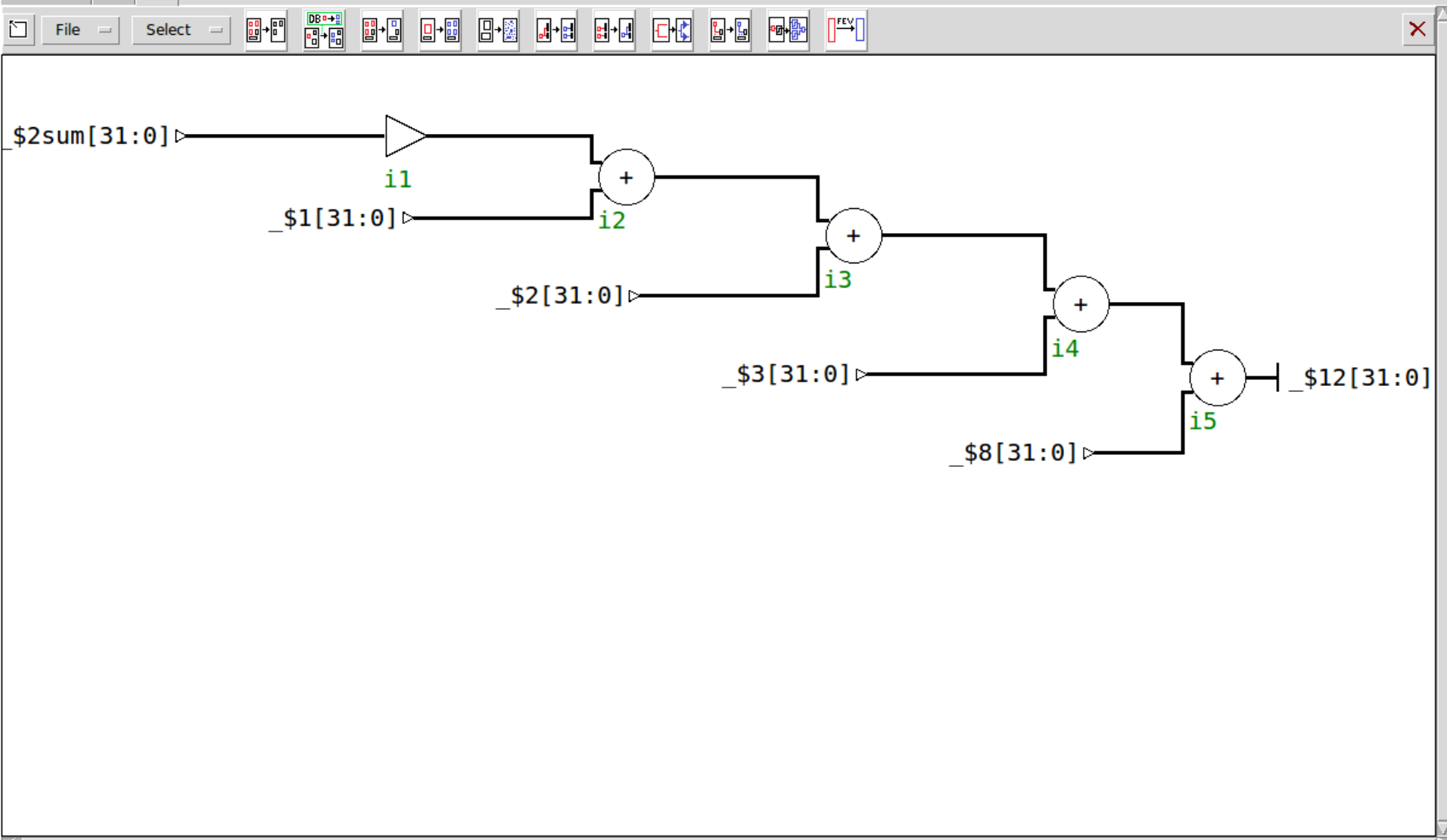

IDV Home
C1:
File
Select
clk
:0]
:0]
w1[31:0]
x1[31:0]
w2[31:0]
x2[31:0]
w3[31:0]
x3[31:0]
init
b0[31:0]
i7
i11
i1
i15
i16
i12
i2
i13
i3
i14
i4
i6
i5
i17
i18
i19
i8
i9
i10
res[31:0]

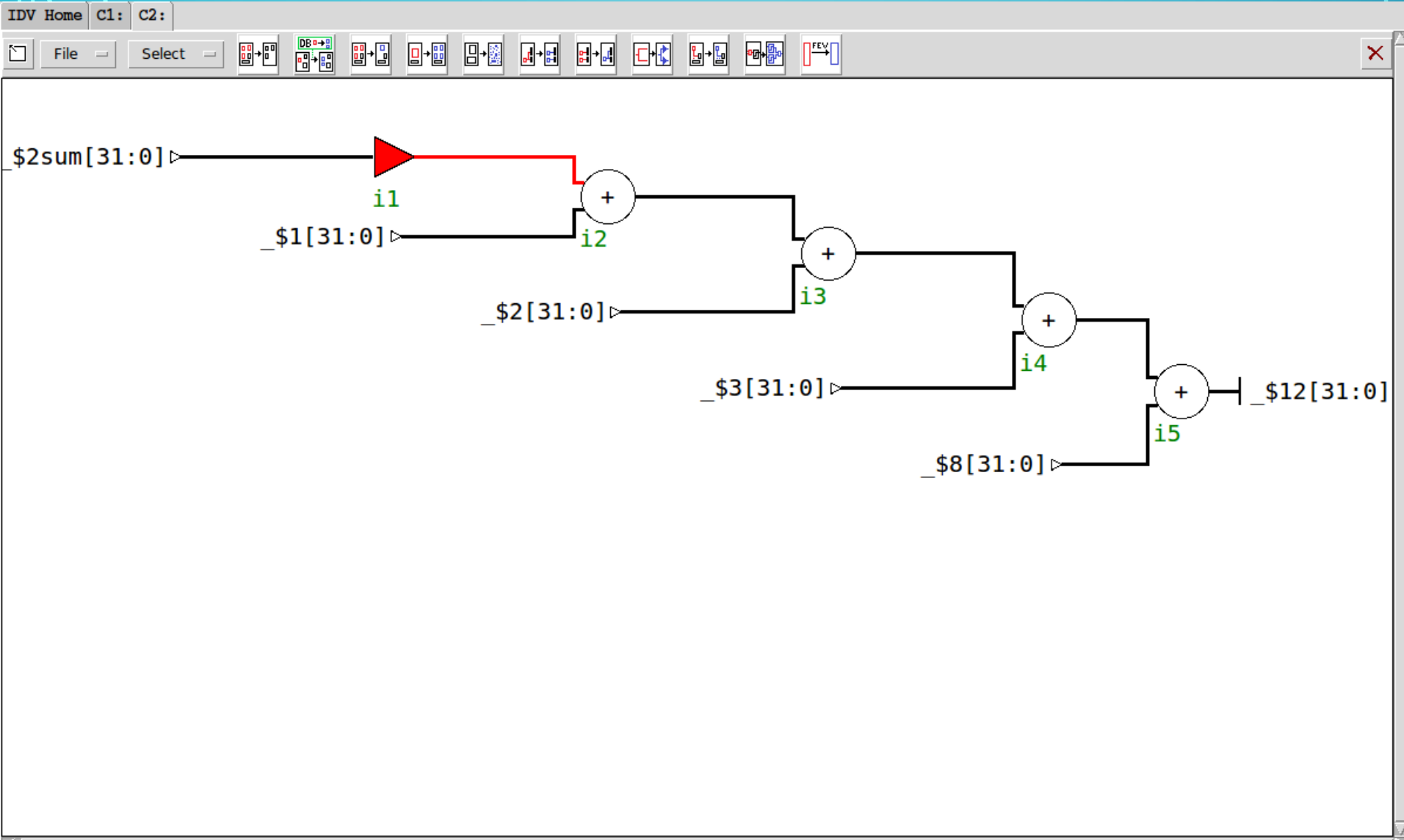

IDV Home
C1:
File
Select
clk
0[31:0]
1[31:0]
2[31:0]
3[31:0]
0[31:0]
1[31:0]
2[31:0]
3[31:0]
stage1
clk
w0[31:0] _$1[31:0]
w1[31:0]
w2[31:0] _$2[31:0]
w3[31:0]
x0[31:0] _$3[31:0]
x1[31:0]
x2[31:0]
x3[31:0] _$8[31:0]
i1
i14
i8
i9
i10
i11
init
b0[31:0]
i3
i2
i12
i5
i6
i7
res[31:0]

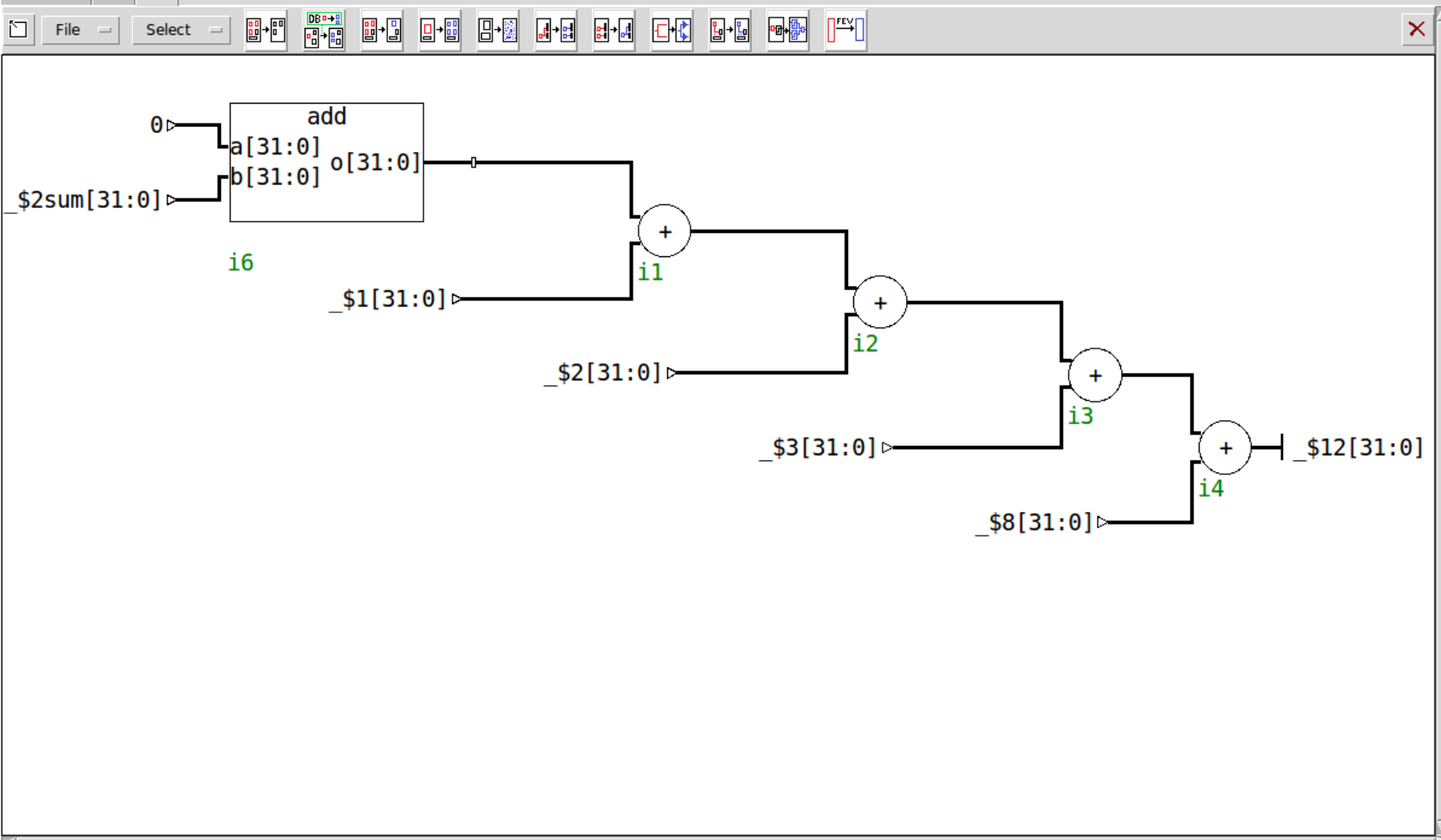

IDV Home
C1:
File
Select
clk
w0[31:0]
w1[31:0]
w2[31:0]
w3[31:0]
x0[31:0]
x1[31:0]
x2[31:0]
x3[31:0]
stage1
clk
w0[31:0]
w1[31:0]
w2[31:0]
w3[31:0]
x0[31:0]
x1[31:0]
x2[31:0]
x3[31:0]
$1[31:0]
$2[31:0]
$3[31:0]
$8[31:0]
i1
i4
i8
i9
i10
i11
i12
init
b0[31:0]
i3
i2
i5
i6
i7
res[31:0]

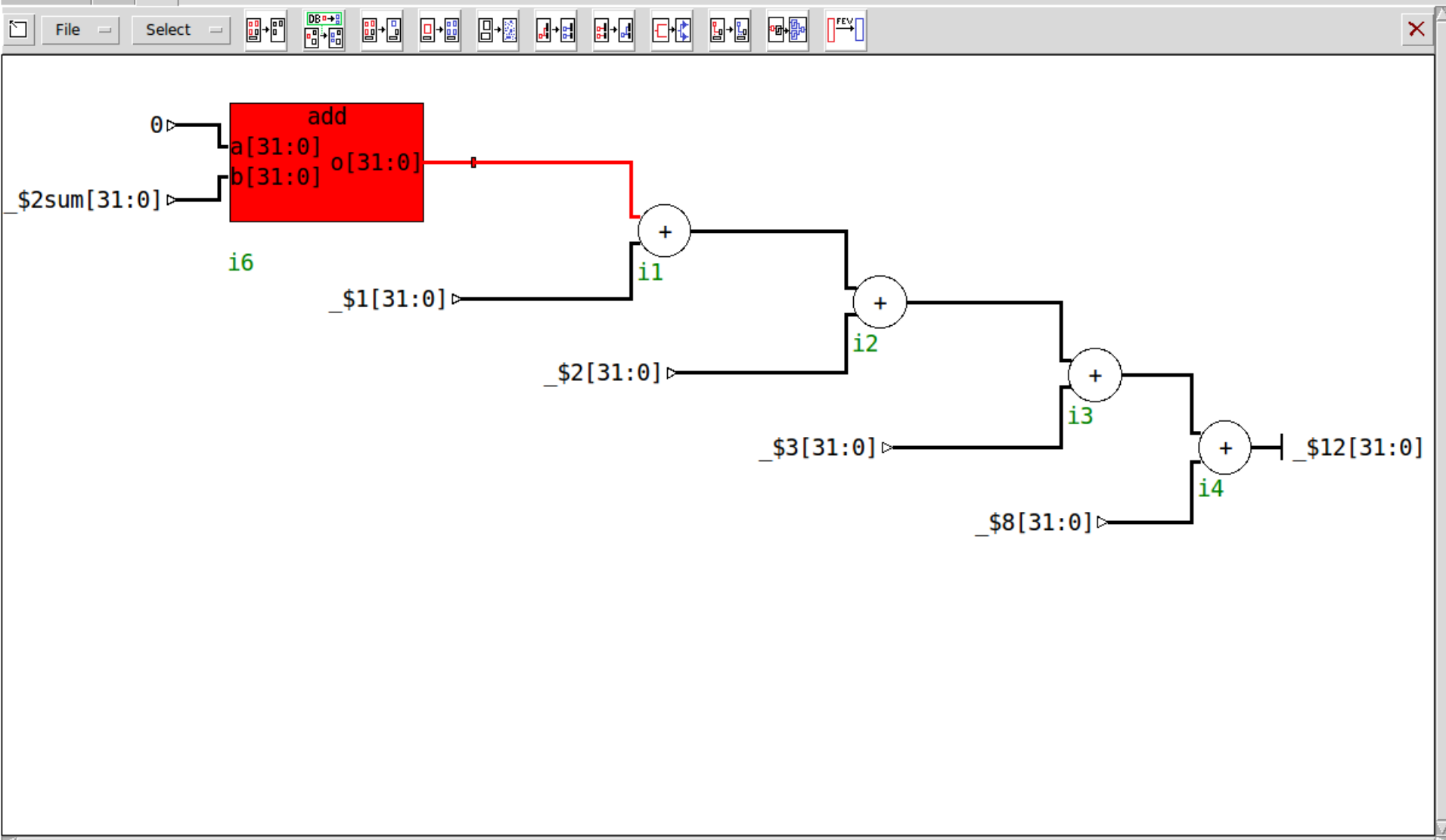

IDV Home
C1:
C2:
File
Select
_$2sum[31:0]
_$1[31:0]
+
i1
_$2[31:0]
+
i2
_$3[31:0]
+
i3
_$8[31:0]
+
i4
_$12[31:0]

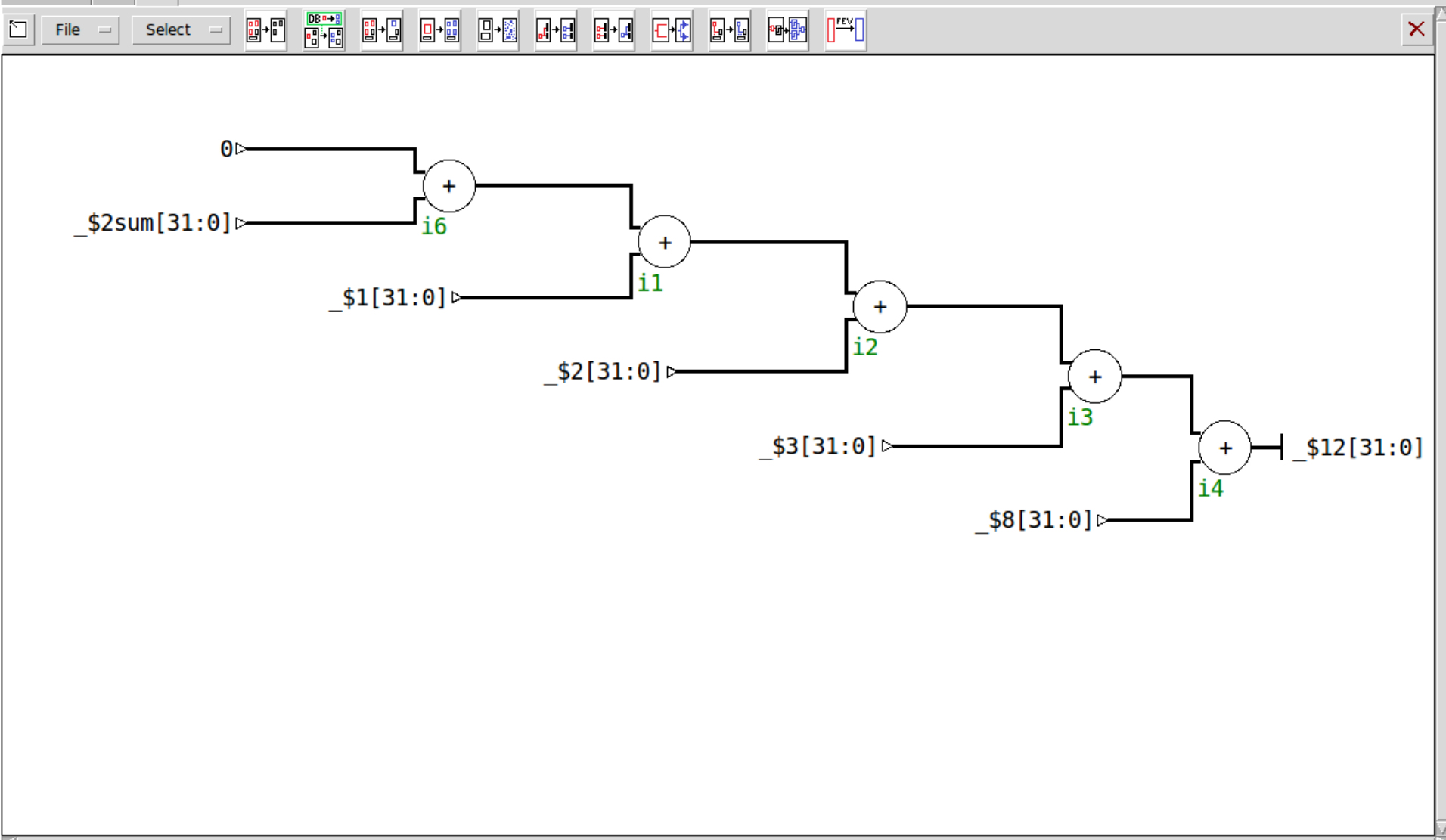

IDV Home
C1:
C2:
File
Select
_$2sum[31:0]
_$1[31:0]
+
i1
_$2[31:0]
+
i2
_$3[31:0]
+
i3
_$8[31:0]
+
i4
_$12[31:0]

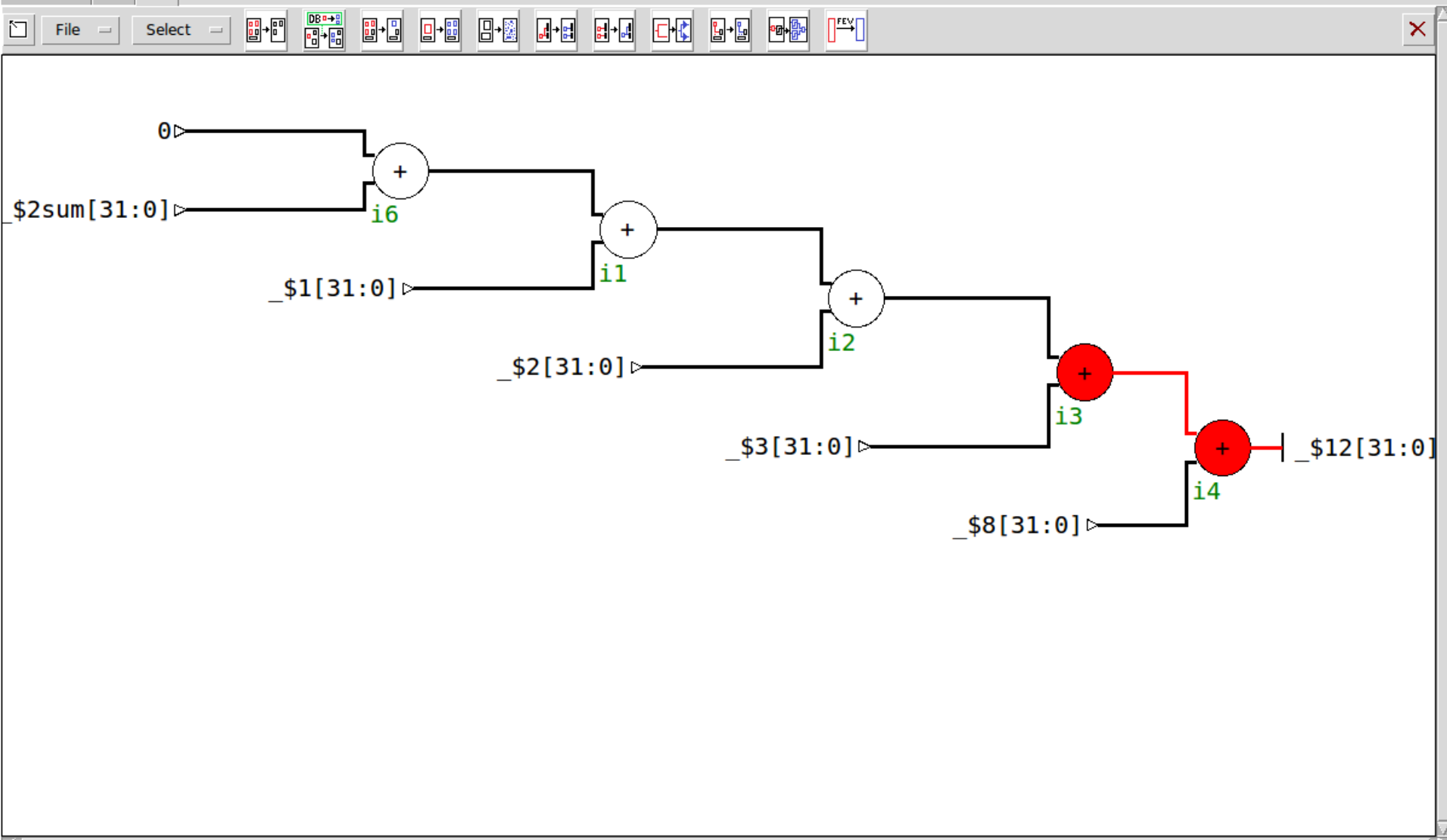

IDV Home
C1:
C2:
File
Select
_$2sum[31:0]
i1
_$1[31:0]
i2
+
i3
_$2[31:0]
i4
_$3[31:0]
i5
_$8[31:0]
_$12[31:0]

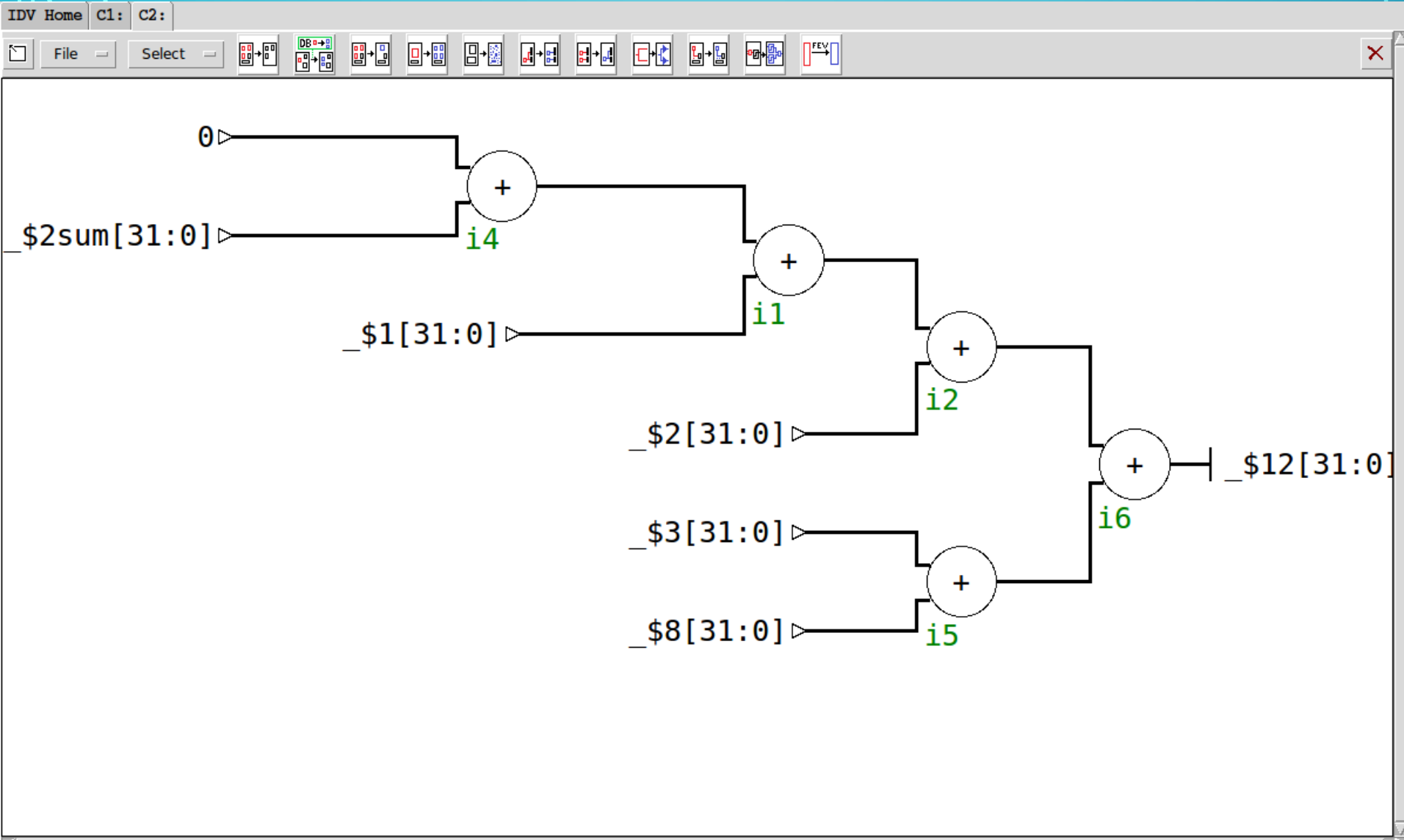

IDV Home
C1:
C2:
File
Select
_$2sum[31:0]
i1
_$1[31:0]
i2
+
i3
_$2[31:0]
i4
_$3[31:0]
i5
_$8[31:0]
_$12[31:0]

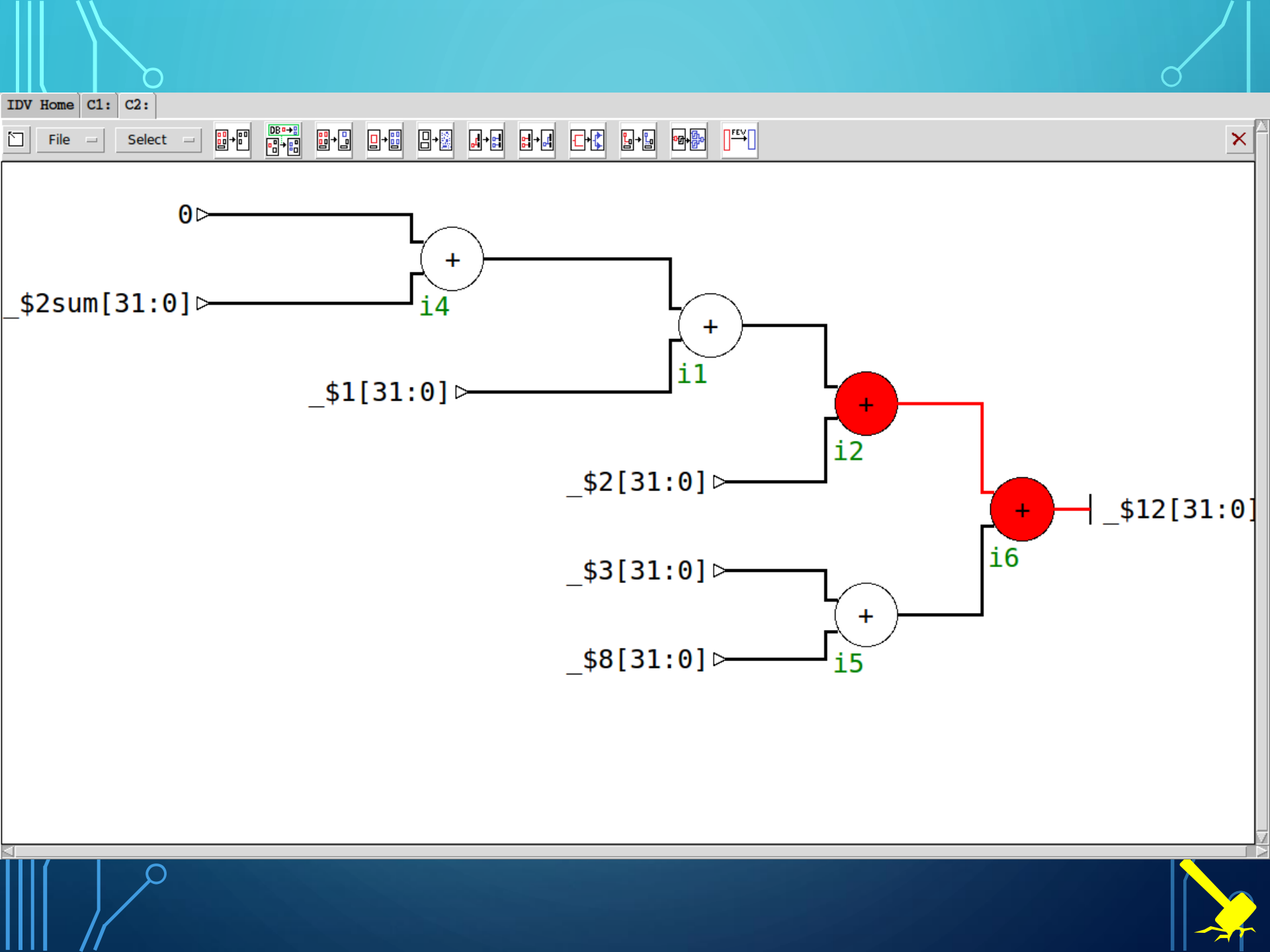

IDV Home
C1:
C2:
File
Select
add
0
a[31:0]
b[31:0]
o[31:0]
_$2sum[31:0]
i6
+
i1
_$1[31:0]
+
i2
_$2[31:0]
+
i3
_$3[31:0]
+
i4
_$8[31:0]
_$12[31:0]

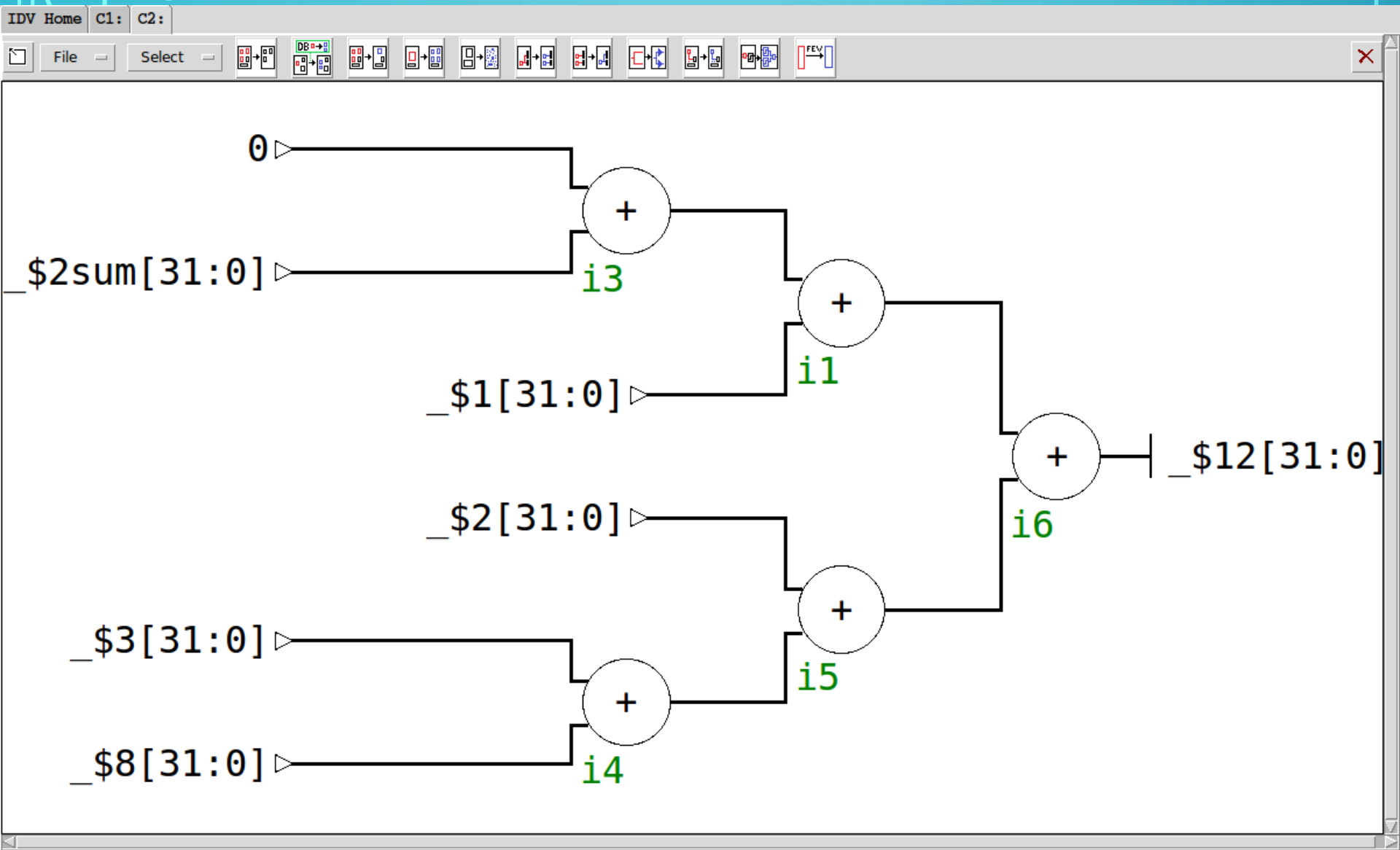

IDV Home
C1:
C2:
File
Select
0
add
a[31:0]
b[31:0]
o[31:0]
_$2sum[31:0]
i6
+
i1
_$1[31:0]
+
i2
_$2[31:0]
+
i3
_$3[31:0]
+
i4
_$12[31:0]
_$8[31:0]

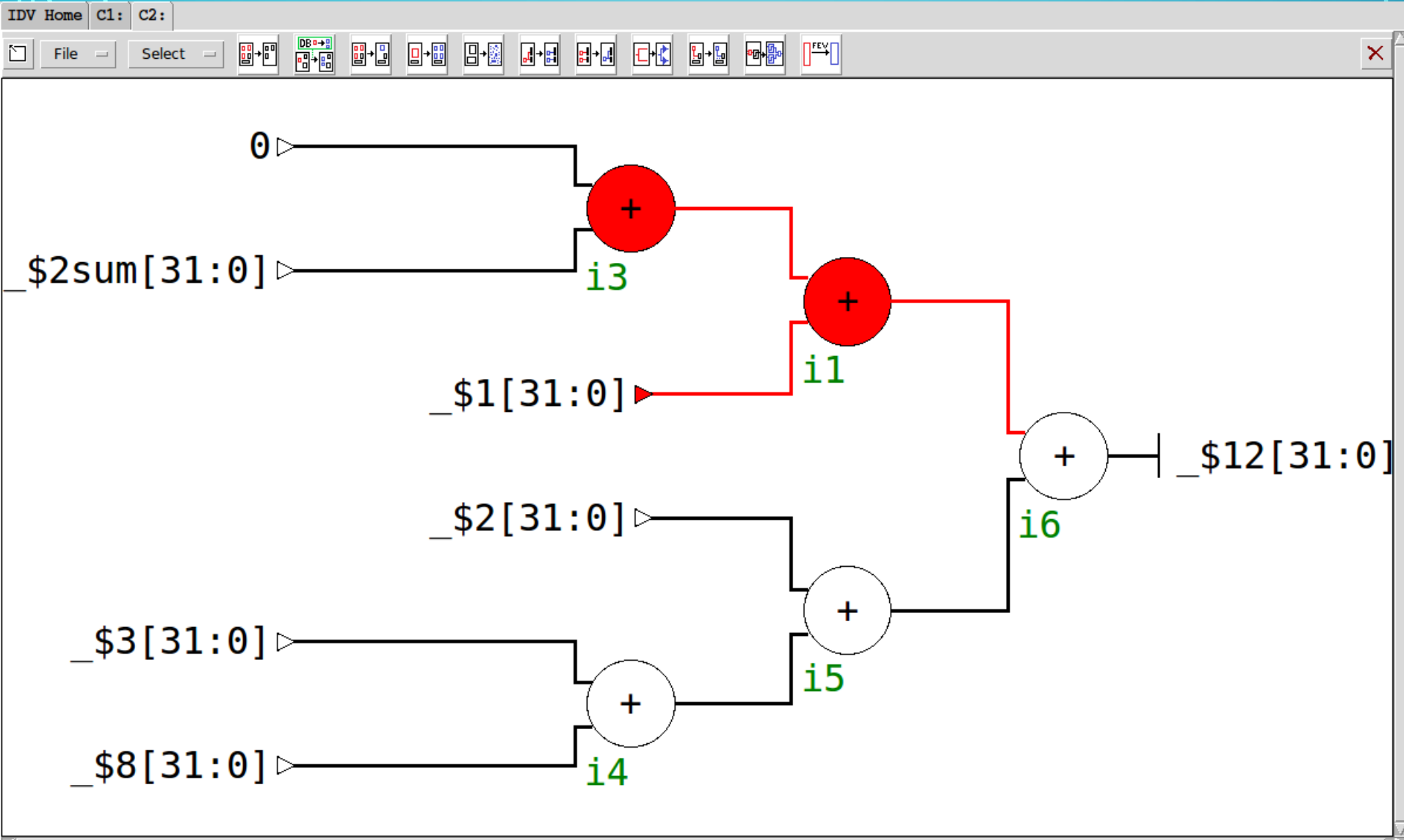

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
i6
_$1[31:0]
i1
_$2[31:0]
i2
_$3[31:0]
i3
_$8[31:0]
i4
_$12[31:0]

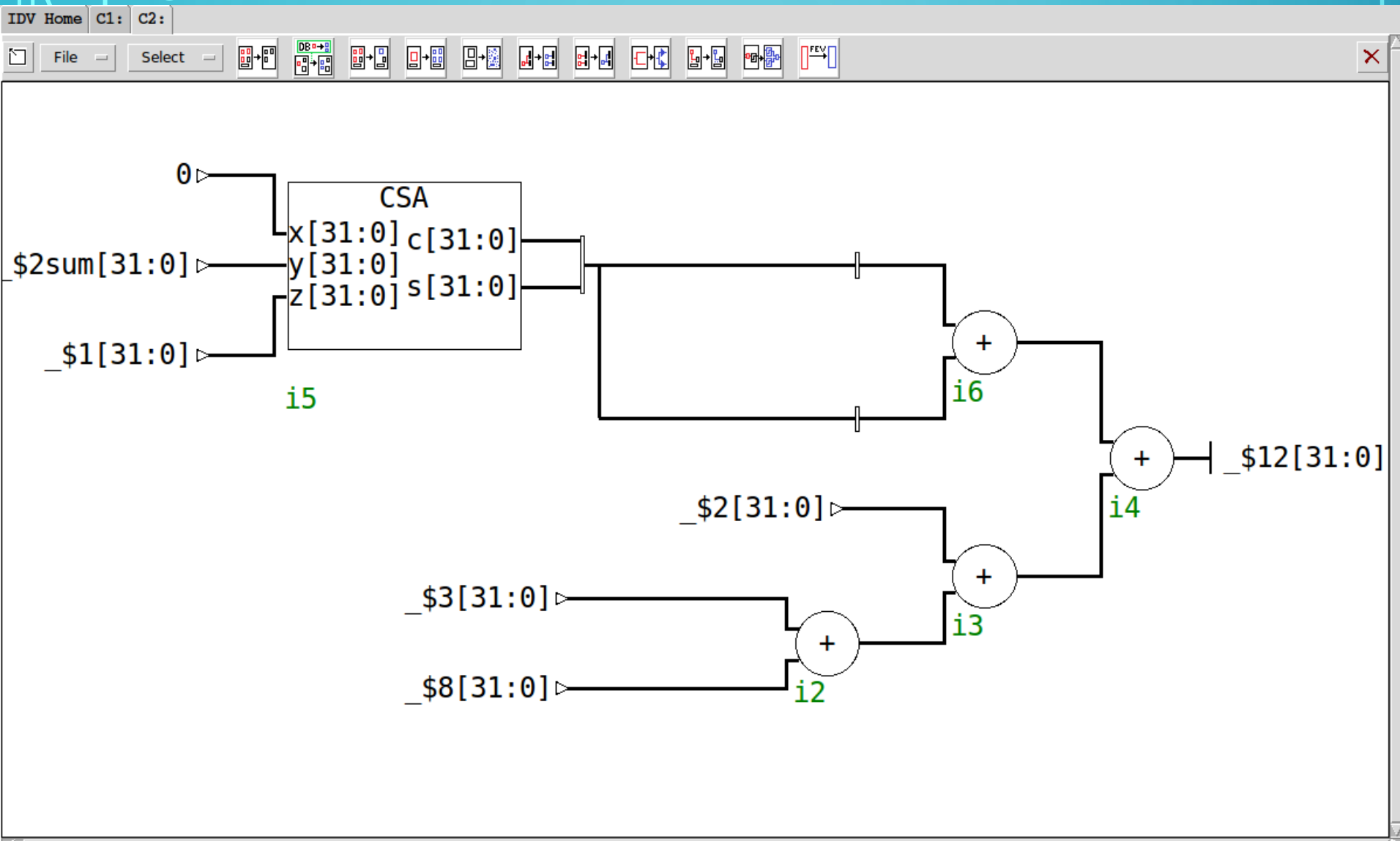

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
i6
_$1[31:0]
i1
_$2[31:0]
i2
_$3[31:0]
i3
_$8[31:0]
i4
_$12[31:0]

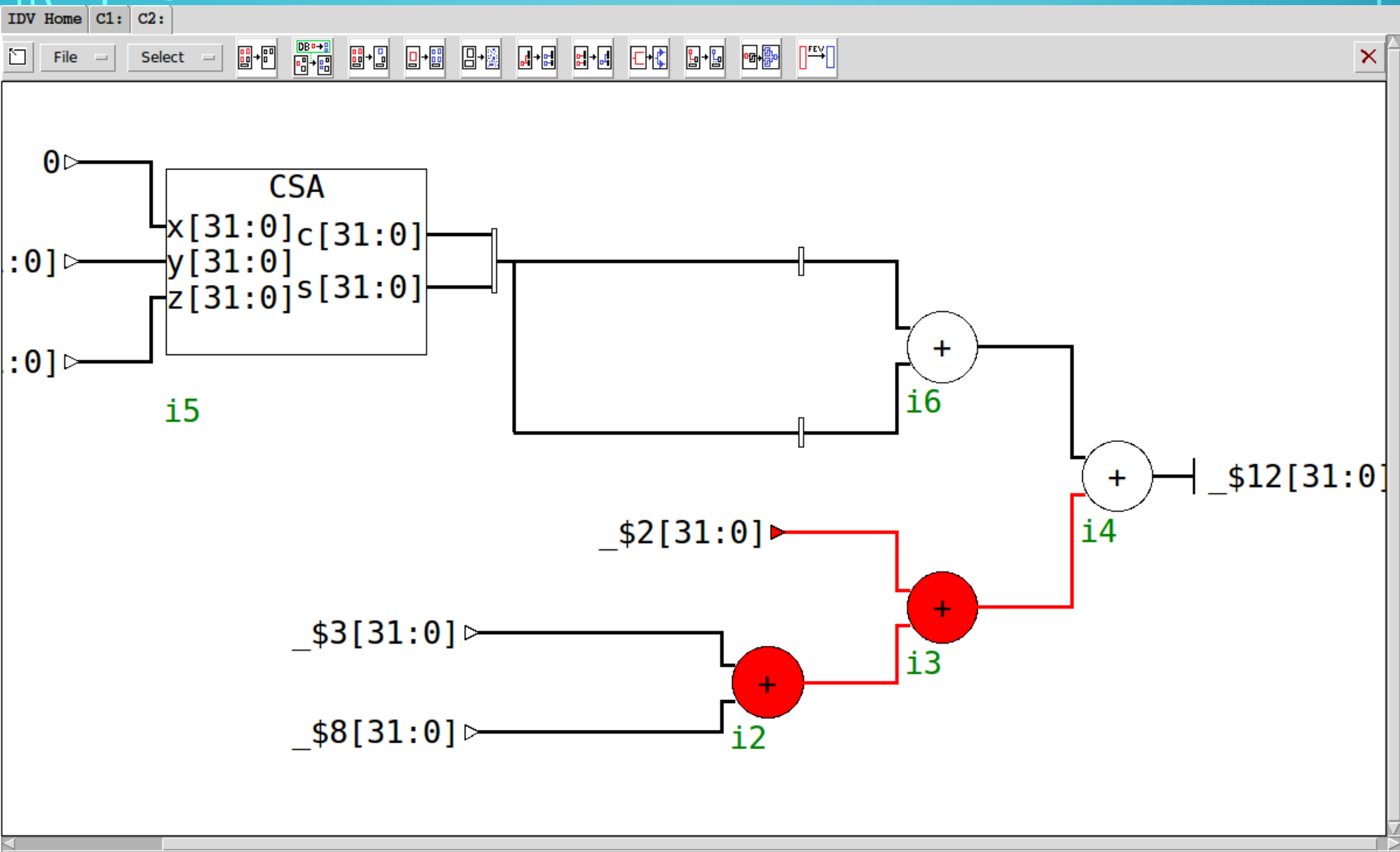

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
+
i4
_$1[31:0]
+
i1
_$2[31:0]
+
i2
_$3[31:0]
_$8[31:0]
+
i5
+
i6
_$12[31:0]

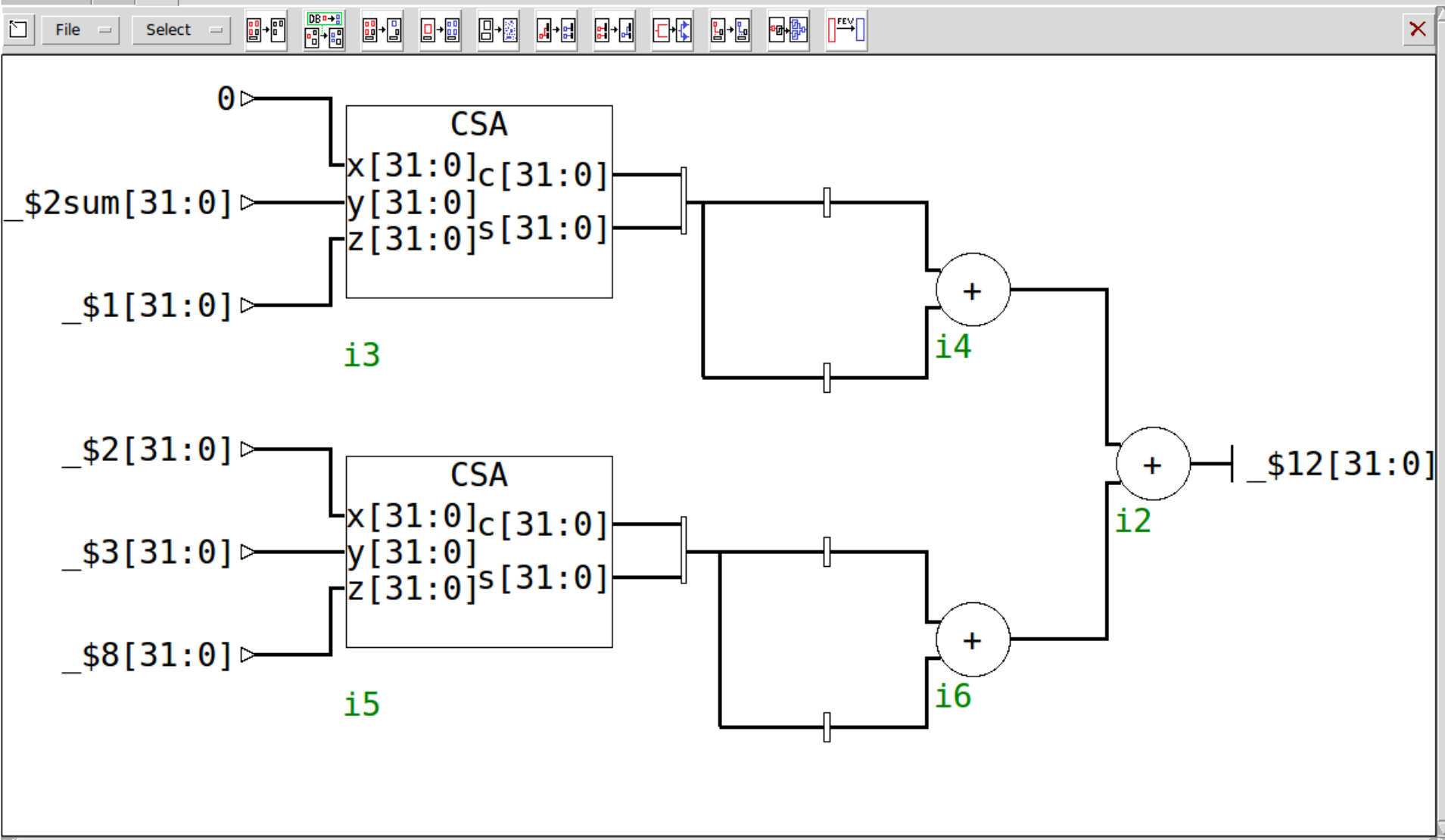

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
i4
_$1[31:0]
i1
_$2[31:0]
i2
_$3[31:0]
_$8[31:0]
i5
i6
_$12[31:0]

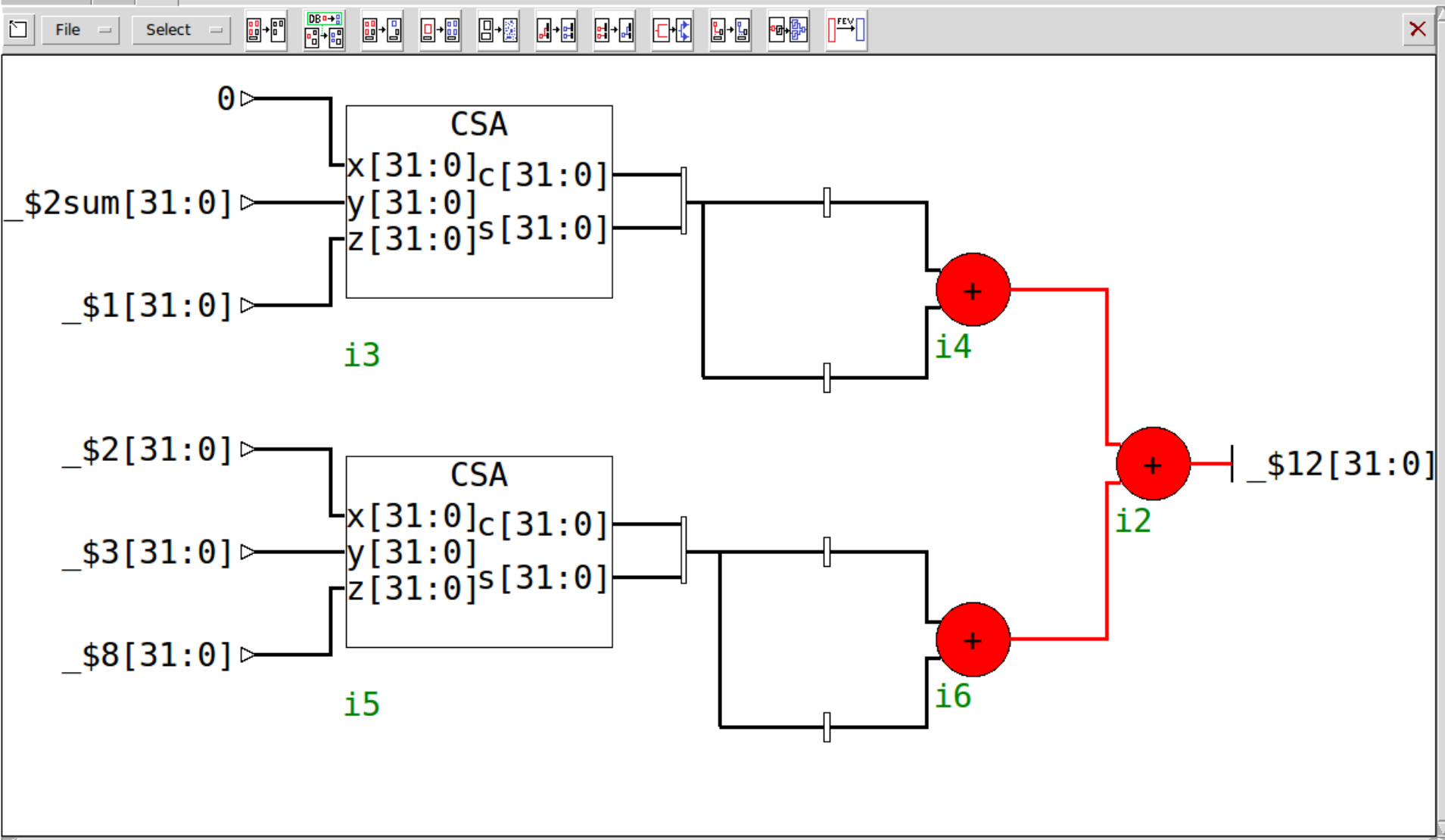

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
+
i3
_$1[31:0]
+
i1
_$2[31:0]
+
i6
_$12[31:0]
_$3[31:0]
+
i5
_$8[31:0]
+
i4

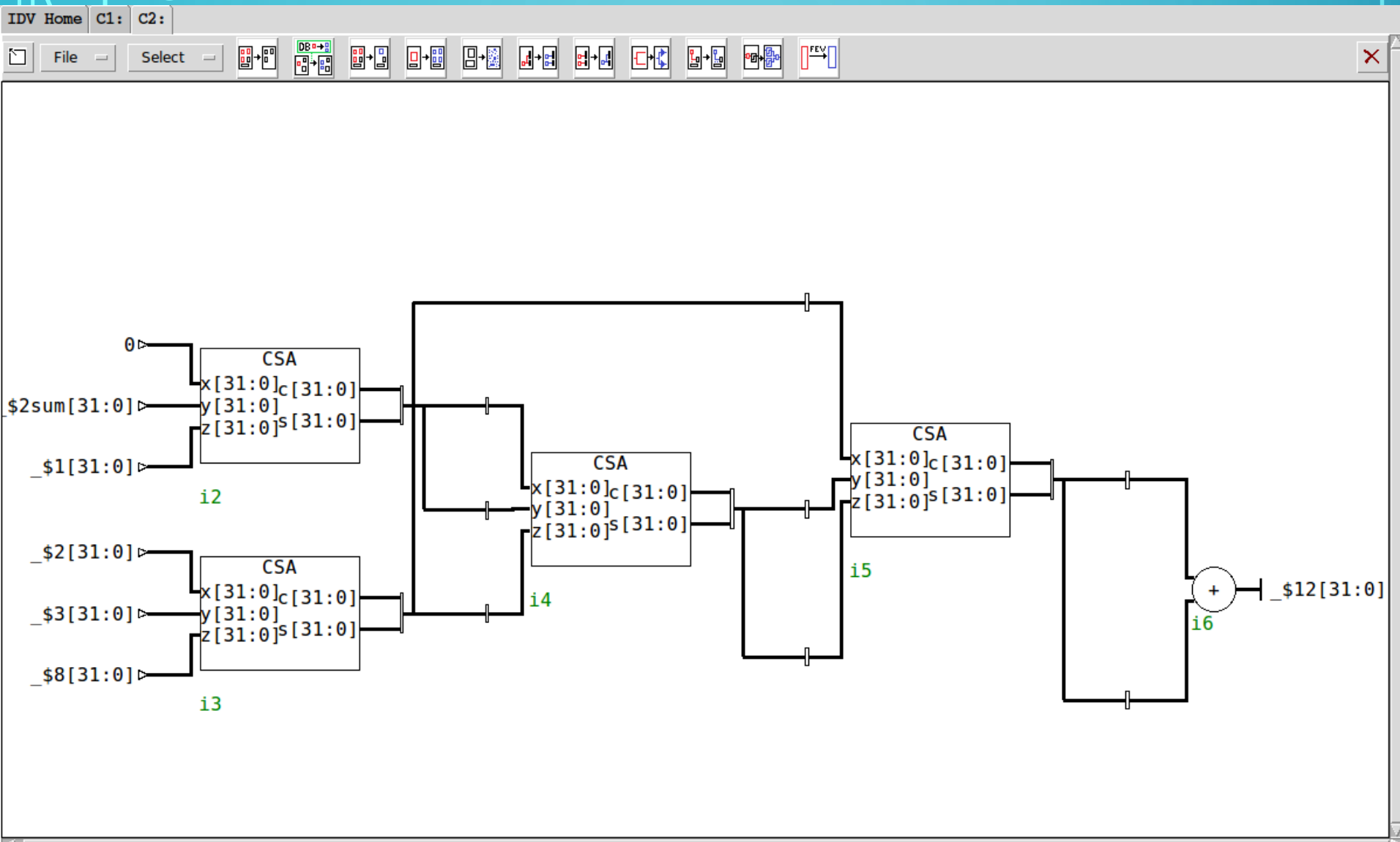

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
i3
_$1[31:0]
i1
_$2[31:0]
i6
_$12[31:0]
_$3[31:0]
i5
_$8[31:0]
i4

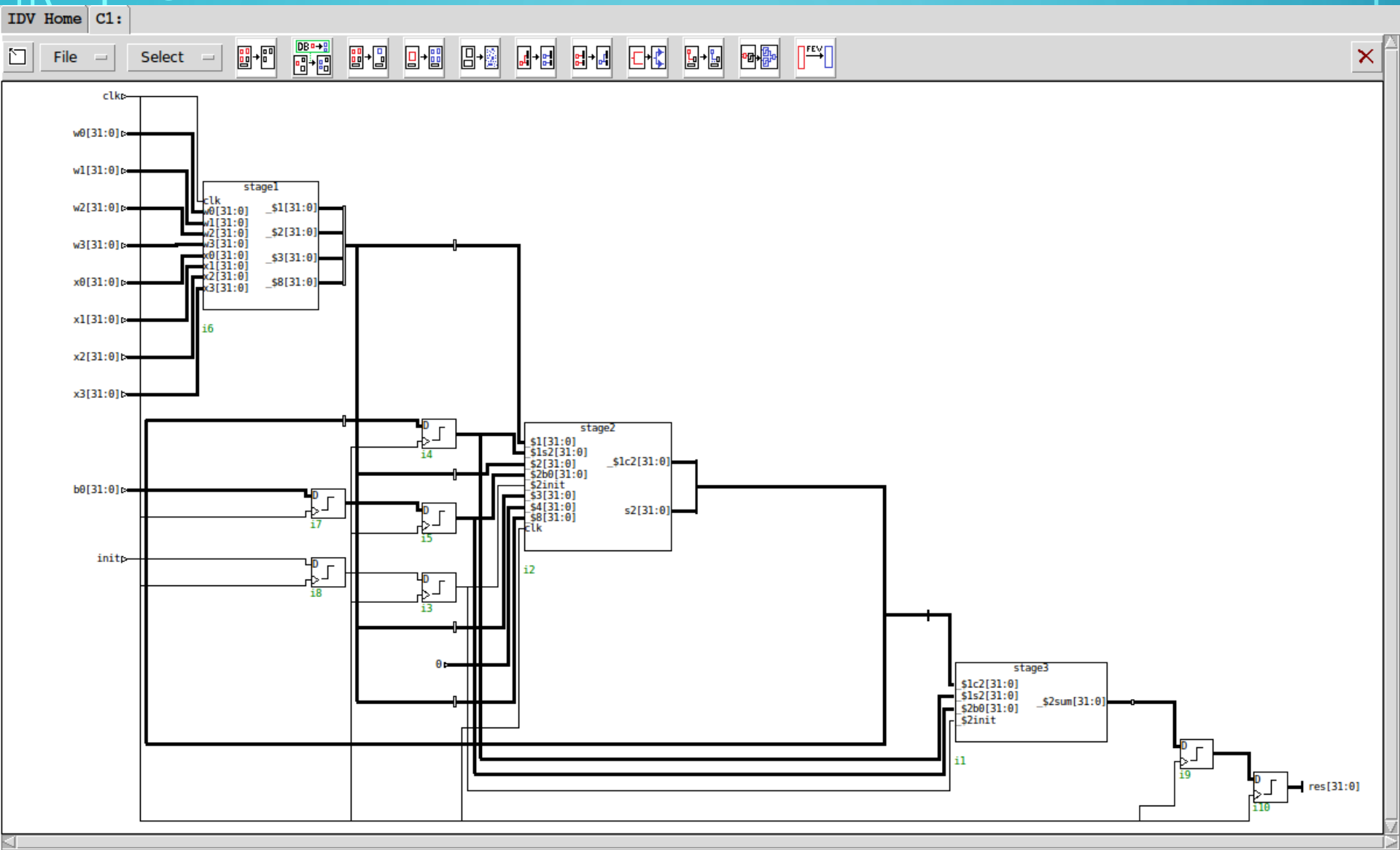

IDV Home
C1:
C2:
File
Select
0
$2sum[31:0]
_$1[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i5
i6
i4
i3
i2
_$12[31:0]
_$2[31:0]
_$3[31:0]
_$8[31:0]


IDV Home
C1:
C2:
File
Select
0
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
:0]
:0]
i5
i6
i4
_$12[31:0]
_$2[31:0]
_$3[31:0]
_$8[31:0]
i3
i2

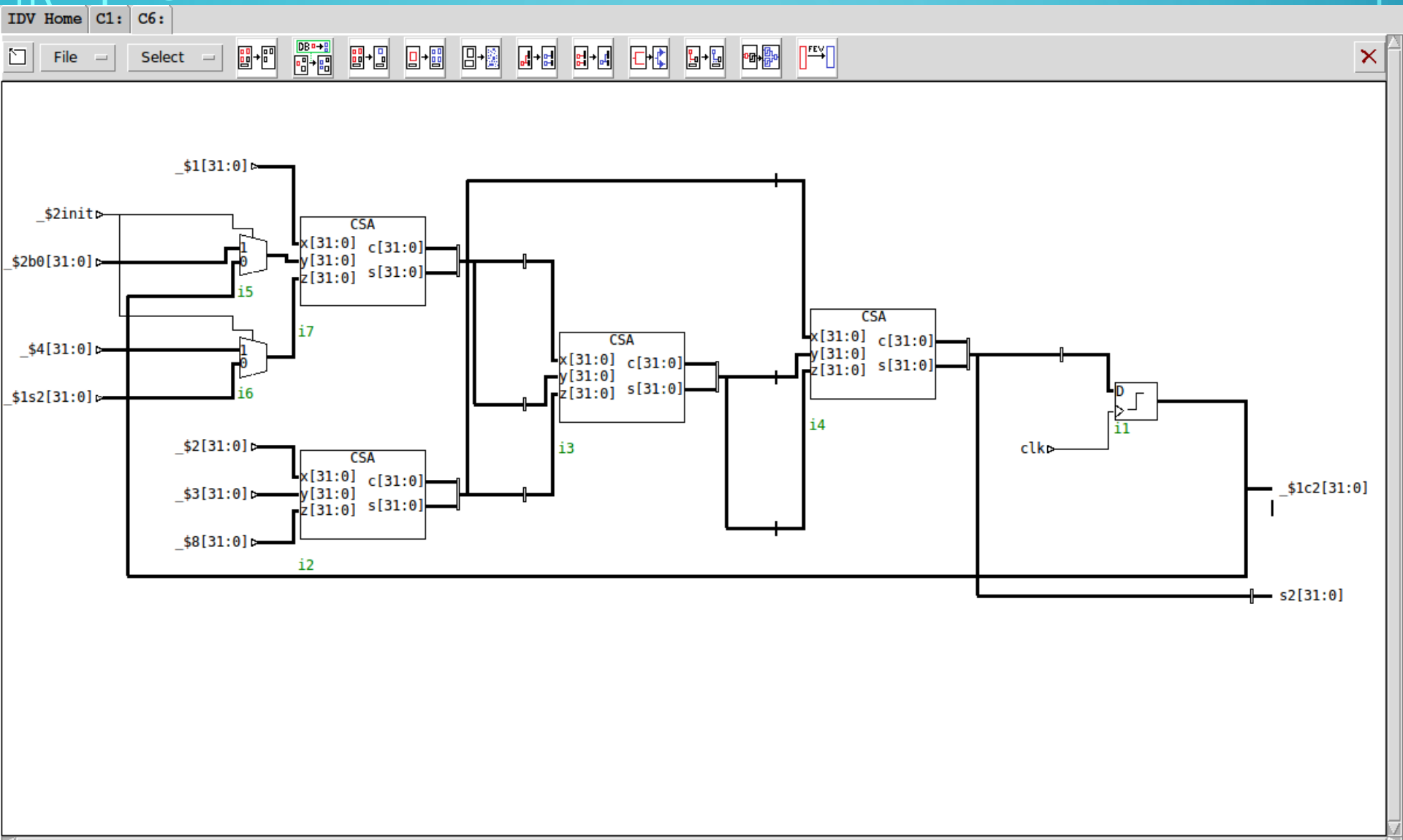

IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
_$1[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i3
i4
_$2[31:0]
_$3[31:0]
_$8[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i5
i6
i2
_$12[31:0]


IDV Home
C1:
C2:
File
Select
0
_$2sum[31:0]
_$1[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i3
i4
_$2[31:0]
_$3[31:0]
_$8[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i5
i6
i2
_$12[31:0]


IDV Home
C1:
C2:
File
Select
0
$2sum[31:0]
_$1[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i2
_$2[31:0]
_$3[31:0]
_$8[31:0]
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i3
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i4
CSA
x[31:0]
y[31:0]
z[31:0]
c[31:0]
s[31:0]
i5
+
i6
_$12[31:0]


IDV Home
C1:
File
Select
clk
w0[31:0]
w1[31:0]
w2[31:0]
w3[31:0]
x0[31:0]
x1[31:0]
x2[31:0]
x3[31:0]
b0[31:0]
init
stage1
_$1[31:0]
_$2[31:0]
_$3[31:0]
_$8[31:0]
i6
i4
i7
i5
i8
i3
stage2
$1[31:0]
$1s2[31:0]
$2[31:0]
$2b0[31:0]
$2init
$3[31:0]
$4[31:0]
$8[31:0]
_$1c2[31:0]
s2[31:0]
i2
0
stage3
$1c2[31:0]
$1s2[31:0]
$2b0[31:0]
$2init
_$2sum[31:0]
i1
i9
i10
res[31:0]

# Details of Stage 2



# STATUS

- Thor is a less than one year old project, but many of the ingredients have been in development for many years as part of other projects.
- For Thor, we are currently proceeding on two fronts:
  - Developing the system (coding, coding, testing, testing, …)
  - Using it designing an IoT processor (eat your own dog food)
    - Working on deciding our next domain to apply Thor to

# SUMMARY

- To enable domain-specific hardware we need:
  - A method for describing the circuit as “SW”
  - A method to ensure the “SW” model is correct.

**CORRECTLY** A method for compiling the “SW” to “HW”

**CORRECTLY** A method for refining the “HW” to a realistic HW implementation.

- Thor provides a proof-of-concept for such a system

VossII and IDV available at: https://github.com/TeamVoss/VossII

# THANK YOU

## QUESTIONS?