# DHDL

## The D Hardware Design Language

Luís Marques
<luis@luismarques.eu>

http://live.dhdlang.org

# What is hardware?

# What is hardware?

You compile this code with DMD:

```
uint mul(uint a, uint b)
{
    return a * b;
}
```

**Question 1**:
Is this multiplication implemented in hardware?
(i.e., is it hardware accelerated?)

# What is hardware?

**Question 2**:
What about this one?

```
uint mul(uint a, uint b)
{
    uint sum = 0;

    while(a != 0)
    {
        if((a & 1) != 0)
            sum += b;

        b <<= 1;
        a >>= 1;
    }

    return sum;
}
```

# What is hardware?

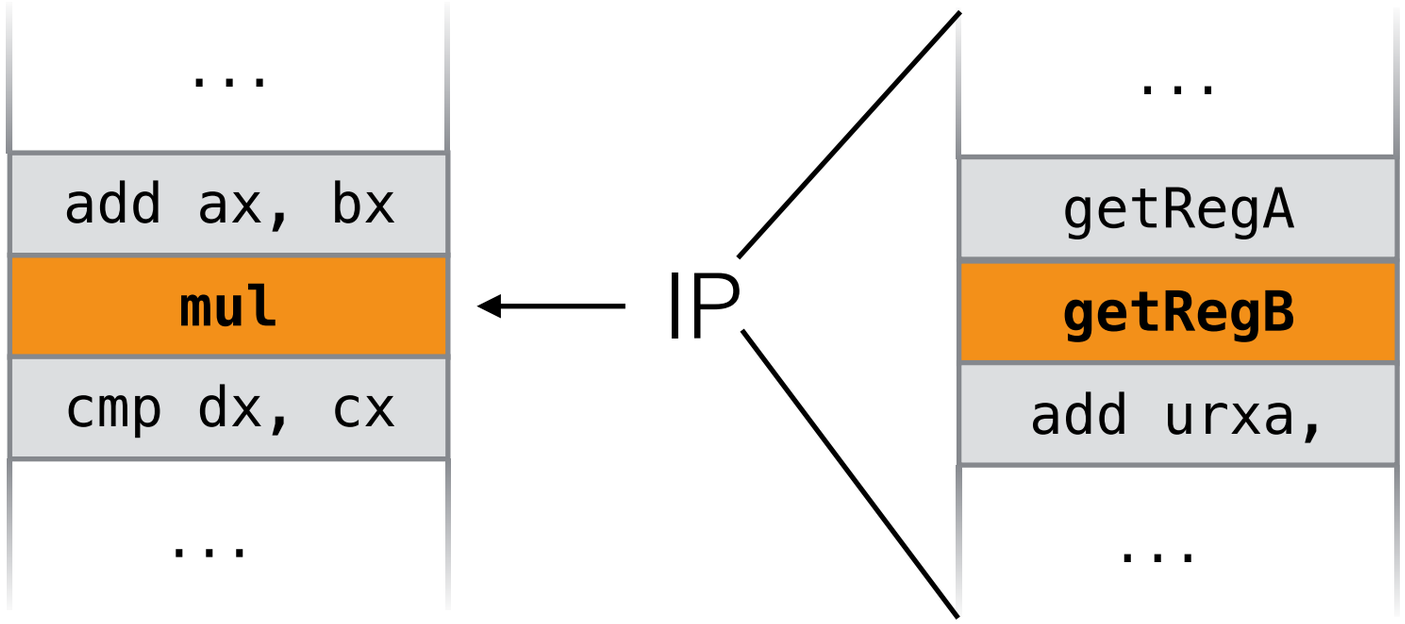
You compile this code with ~~DMD~~ DMC

```
uint mul(uint a, uint b)
{
    return a * b;
}
```

…and you run it on an original Intel 8086.

**Question 3**:
Is this multiplication implemented in hardware?

# What is hardware?




```
uint mul(uint a, uint b)
{
    uint sum = 0;

    while(a != 0)
    {
        if((a & 1) != 0)
            sum += b;

        b <<= 1;
        a >>= 1;
    }

    return sum;
}
```

- Each 8086 instruction is executed as *n* µinstructions
- What if I tell you that the µcode for the `mul` instruction executes *this* algorithm?
- **Question 4**: Is the `mul` instruction implemented in HW?

# What is hardware?

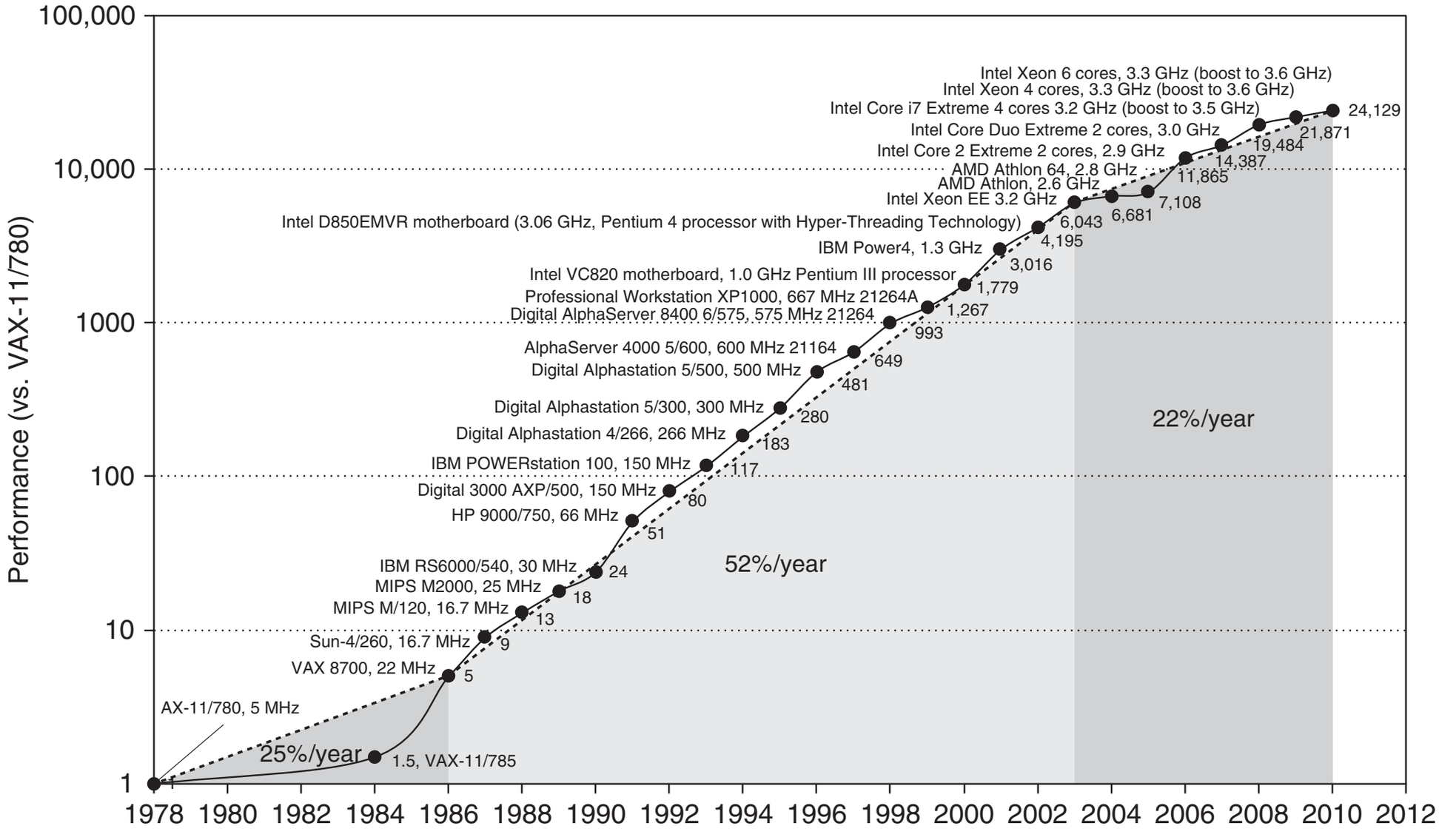
- What if there is no μcode, but the multiplication circuit works through a series of additions and shifts?

- **Question 5**: Does it make a difference? Is it (still?) hardware?

# What is hardware?

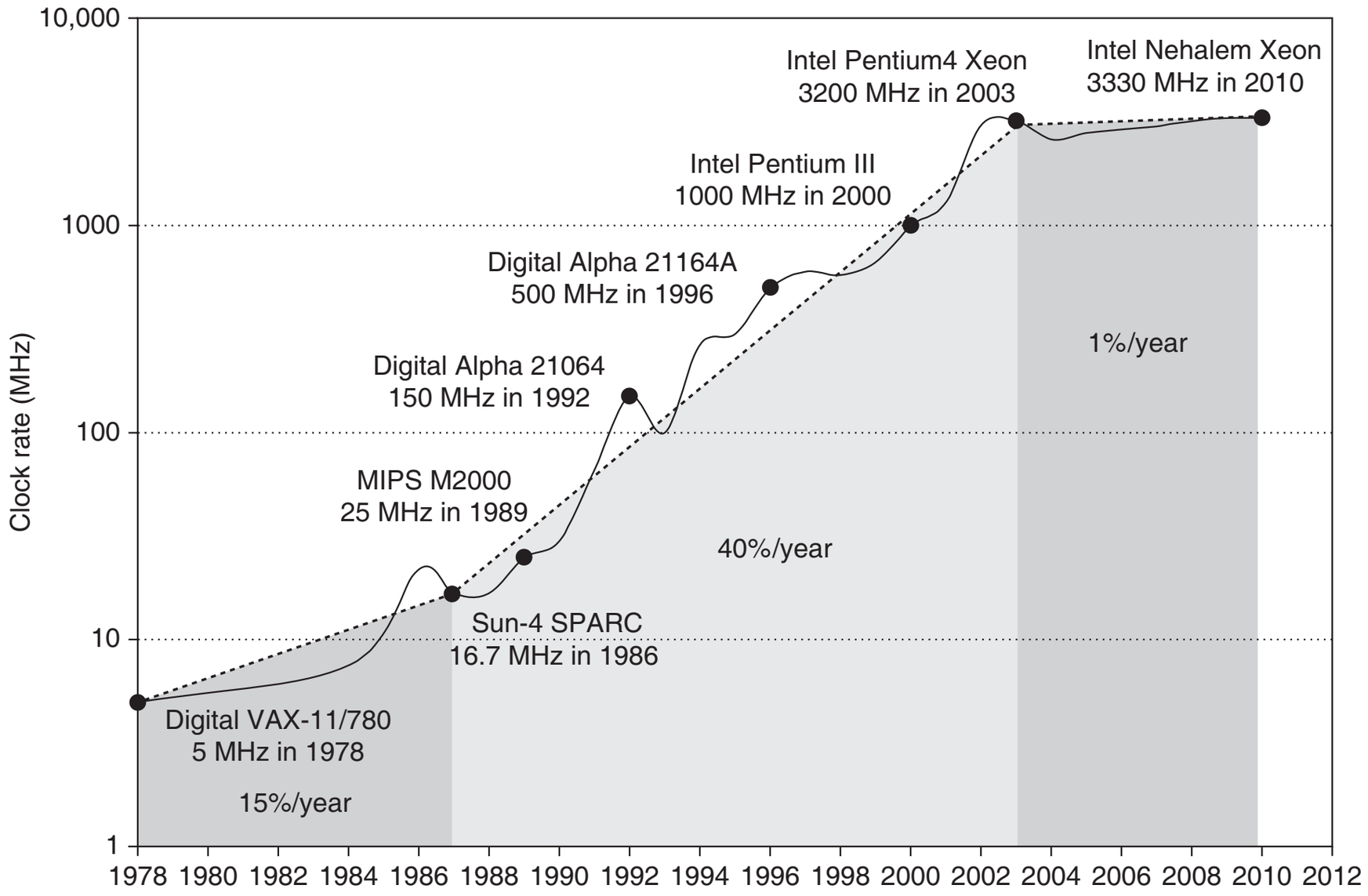
- Software
  - Dynamic allocation of resources
  - Generic
  - Flexible
  - Serial
- Hardware
  - Fixed resources for each function
  - Specialized
  - Hardcoded
  - Parallel

# Why Hardware Matters




Computer Architecture: A Quantitative Approach, Fifth Edition

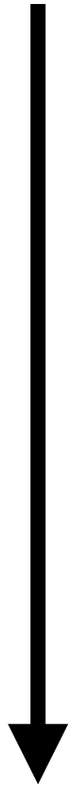
# Why Hardware Matters




Computer Architecture: A Quantitative Approach, Fifth Edition

# Why Hardware Matters

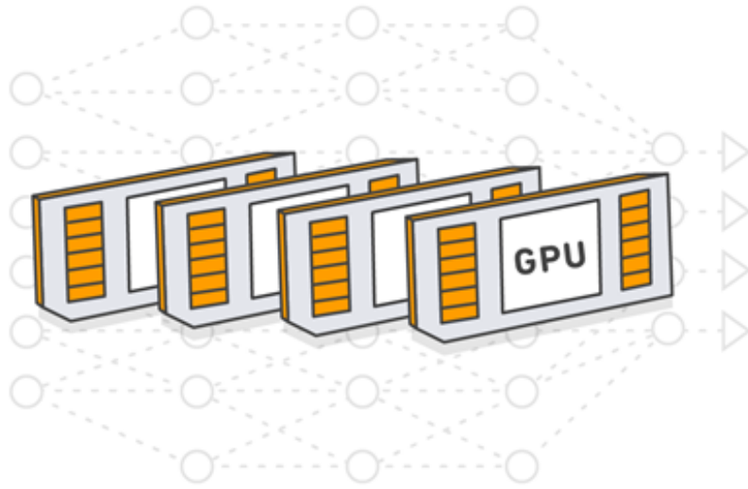
- Clock frequency has stopped increasing
- Solutions:
  - Multiple cores
  - GPUs
  - FPGAs
  - ASICs



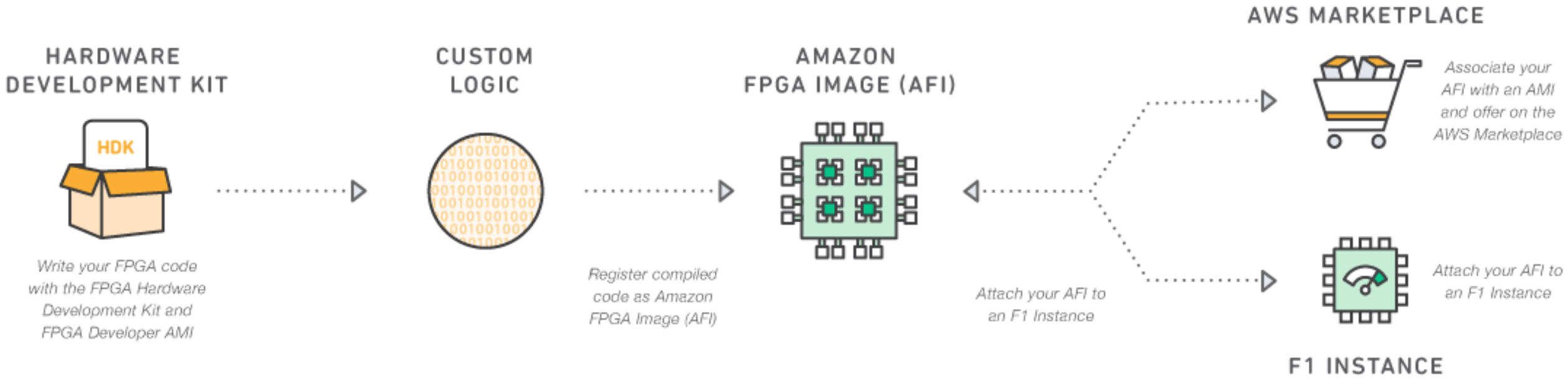
+parallel
+specialized
+harder

# Why Hardware Matters

- The datacenter doesn't lie
- E.g., Amazon
  - GPU instances




  - FPGA instances (f1)



# Creating Hardware

# Creating Hardware

```d
import core.stdc.stdio;

void main()
{
    int a, op, b, v;

    while(true)
    {
        a  = getchar();
        op = getchar();
        b  = getchar();
        if(op == '+') v = a+b-'0';
        if(op == '-') v = a-b+'0';
        putchar(v);
        putchar('\n'); getchar();
    }
}
```

```
$ rdmd -version=v1 calc.d
3+4
7
8-5
3
```

# Creating Hardware

```d
import core.stdc.stdio;

void main()
{
    int a, op, b, v, ip;

    while(true)
    {
        if(ip == 0) { a  = getchar(); ++ip;                }
        if(ip == 1) { op = getchar(); ++ip;                }
        if(ip == 2) { b  = getchar(); ++ip;                }
        if(ip == 3) { if(op == '+') v = a+b-'0'; ++ip; }
        if(ip == 4) { if(op == '-') v = a-b+'0'; ++ip; }
        if(ip == 5) { putchar(v); ++ip;                    }
        if(ip == 6) { putchar('\n'); getchar(); ip=0;  }
    }
}
```

```
$ rdmd -version=v2 calc.d
3+4
7
8-5
3
```

# Creating Hardware

```
        int a, op, b, v, ip;

        while(true)
        {
ip0:        if(ip == 0) { a  = getchar(); ++ip; goto d; }
ip1:        if(ip == 1) { op = getchar(); ++ip; goto d; }
ip2:        if(ip == 2) { b  = getchar(); ++ip; goto d; }
ip3:        if(ip == 3) { if(op == '+') v = a+b-'0'; ++ip; goto d; }
ip4:        if(ip == 4) { if(op == '-') v = a-b+'0'; ++ip; goto d; }
ip5:        if(ip == 5) { putchar(v); ++ip; goto d; }
ip6:        if(ip == 6) { putchar('\n'); getchar(); ip=0; goto d; }
        }

d:      final switch(uniform(0, 7))
        {
            case 0: goto ip0;
            case 1: goto ip1;
            case 2: goto ip2;
            case 3: goto ip3;
            case 4: goto ip4;
            case 5: goto ip5;
            case 6: goto ip6;
        }
```

```
$ rdmd -version=v3 calc.d
3+4
7
8-5
3
```

# Creating Hardware

```d
import core.stdc.stdio;

void main()
{
    int a, op, b, v, ip;

    while(true)
    {
        if(ip == 0) { a  = getchar(); ++ip;                }
        if(ip == 1) { op = getchar(); ++ip;                }
        if(ip == 2) { b  = getchar(); ++ip;                }
        if(ip == 3) { if(op == '+') v = a+b-'0'; ++ip; }
        if(ip == 4) { if(op == '-') v = a-b+'0'; ++ip; }
        if(ip == 5) { putchar(v); ++ip;                    }
        if(ip == 6) { putchar('\n'); getchar(); ip=0;  }
    }
}
```

```
$ rdmd -version=v2 calc.d
3+4
7
8-5
3
```

# Creating Hardware

```d
import core.stdc.stdio;

void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 7;
        if(ip == 0) a  = getchar();
        if(ip == 1) op = getchar();
        if(ip == 2) b  = getchar();
        if(ip == 3) if(op == '+') v = a+b-'0';
        if(ip == 4) if(op == '-') v = a-b+'0';
        if(ip == 5) putchar(v);
        if(ip == 6) putchar('\n'); getchar();
    }
}
```

# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 5;
        if(ip == 0) a  = getchar();
        if(ip == 1) op = getchar();
        if(ip == 2) b  = getchar();
        if(ip == 3) if(op == '+') v = a+b-'0';
        if(ip == 4) if(op == '-') v = a-b+'0';
    }
}
```

# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = getchar();
        if(ip == 1) op = getchar();
        if(ip == 2) b  = getchar();
        if(ip == 3) op == '+' ? v = a+b-'0' : v = a-b+'0';
    }
}
```

# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

adder subcircuit

# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

adder subcircuit

**Question 6**: How many subcircuits are there in here?

# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
        ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

adder

modulus

comparator

comparator

comparator

comparator

multiplexer

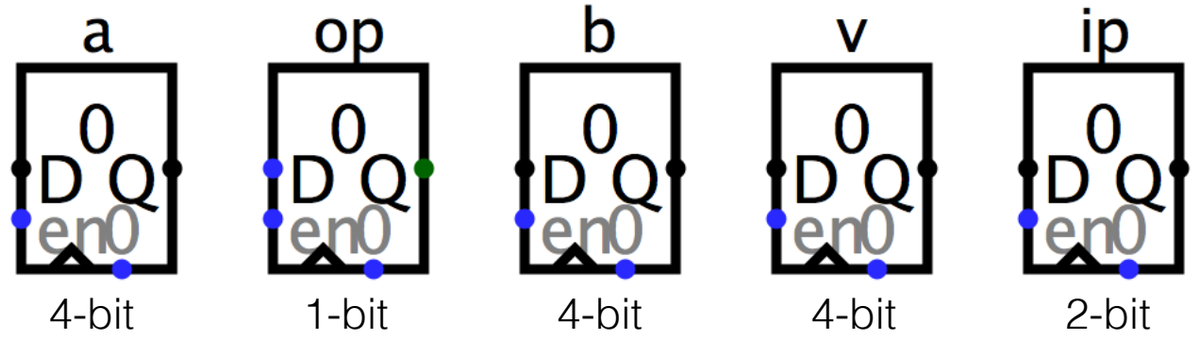
subtractor

comparator

adder subcircuit

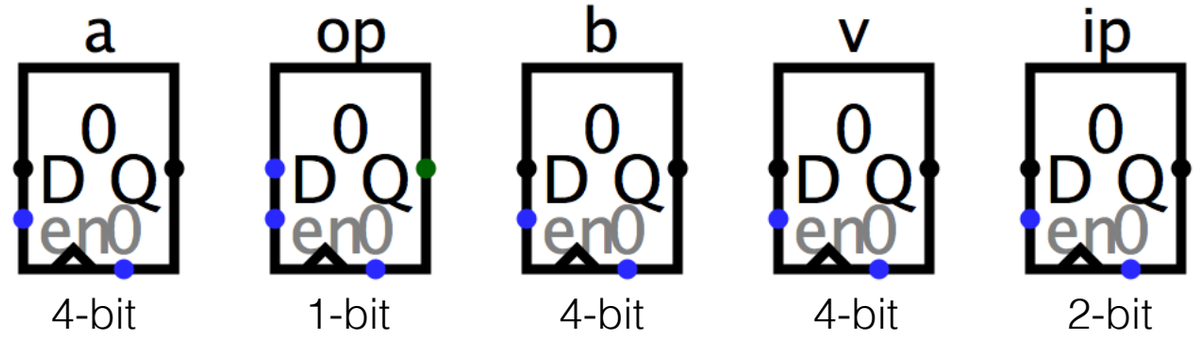
**Question 6**: How many subcircuits are there in here?

# Creating Hardware




```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

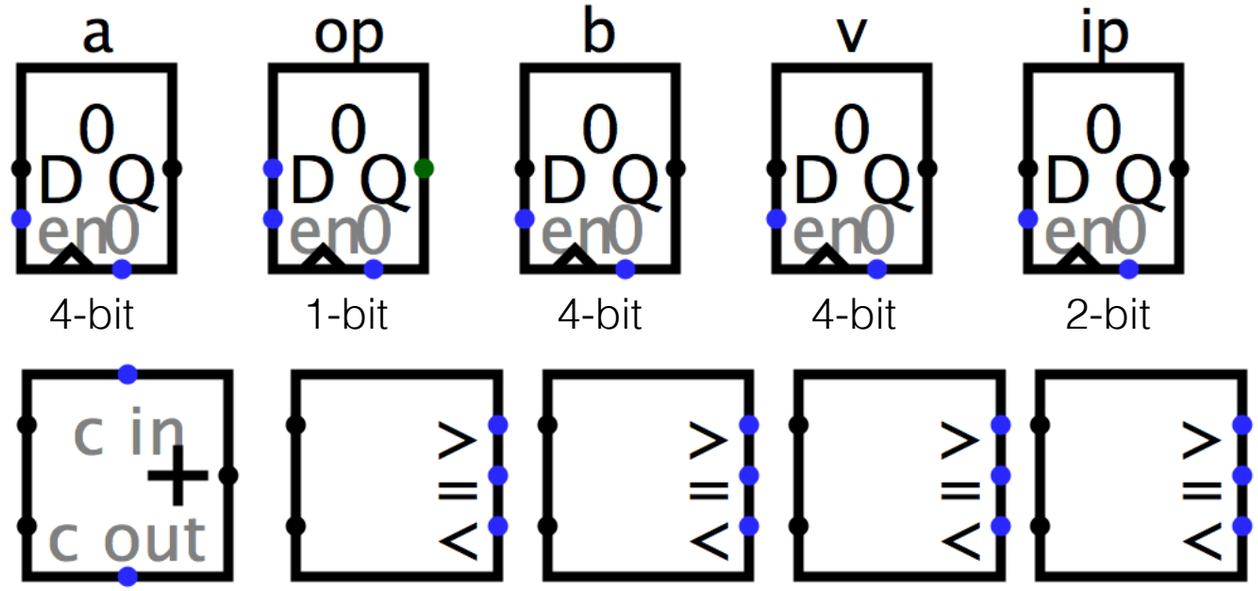
# Creating Hardware




```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

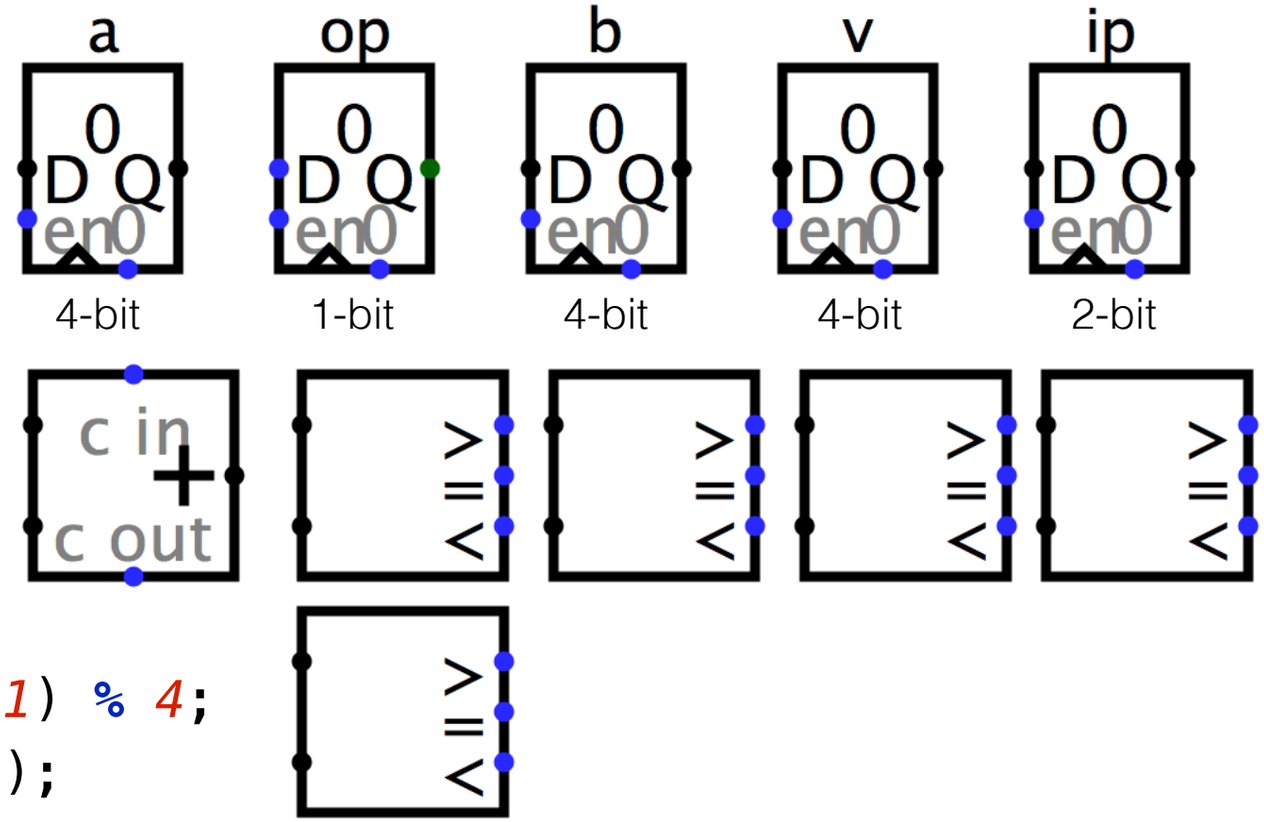
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



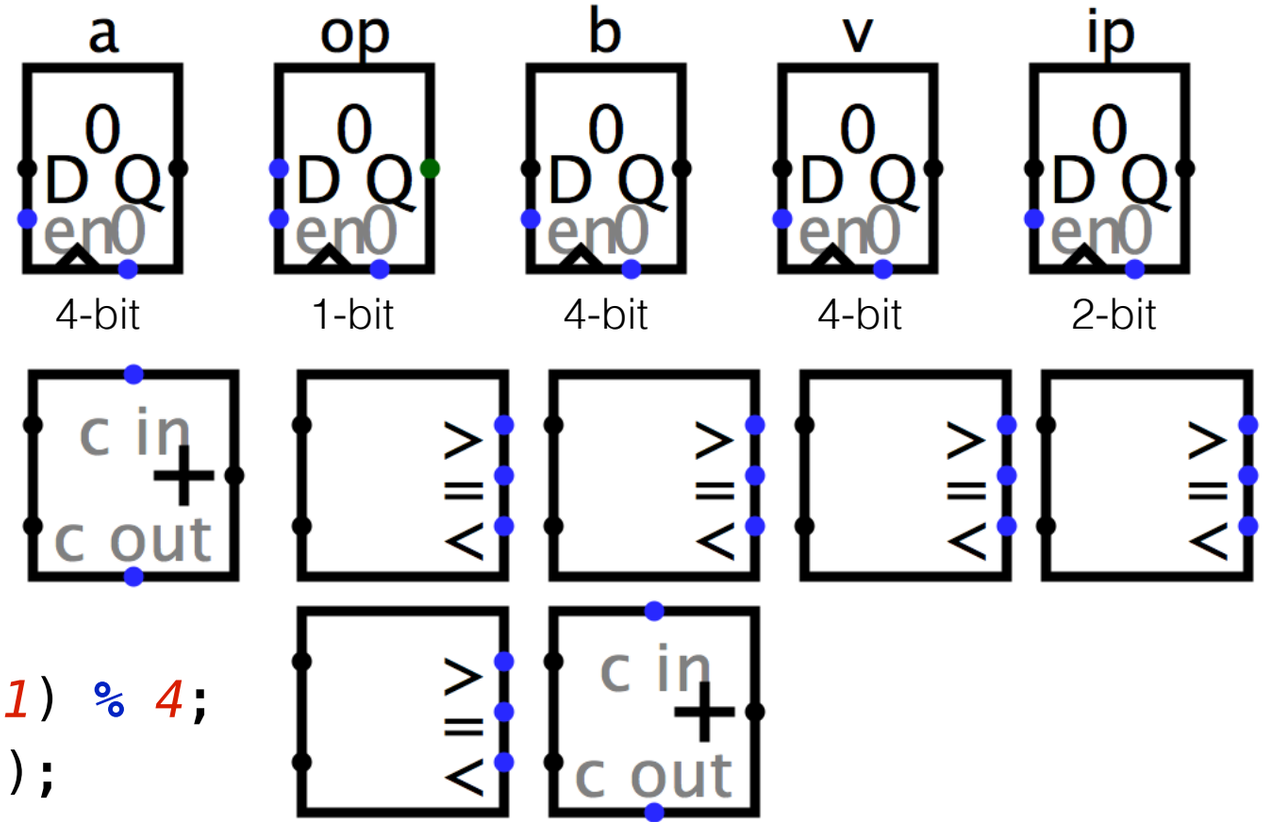
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



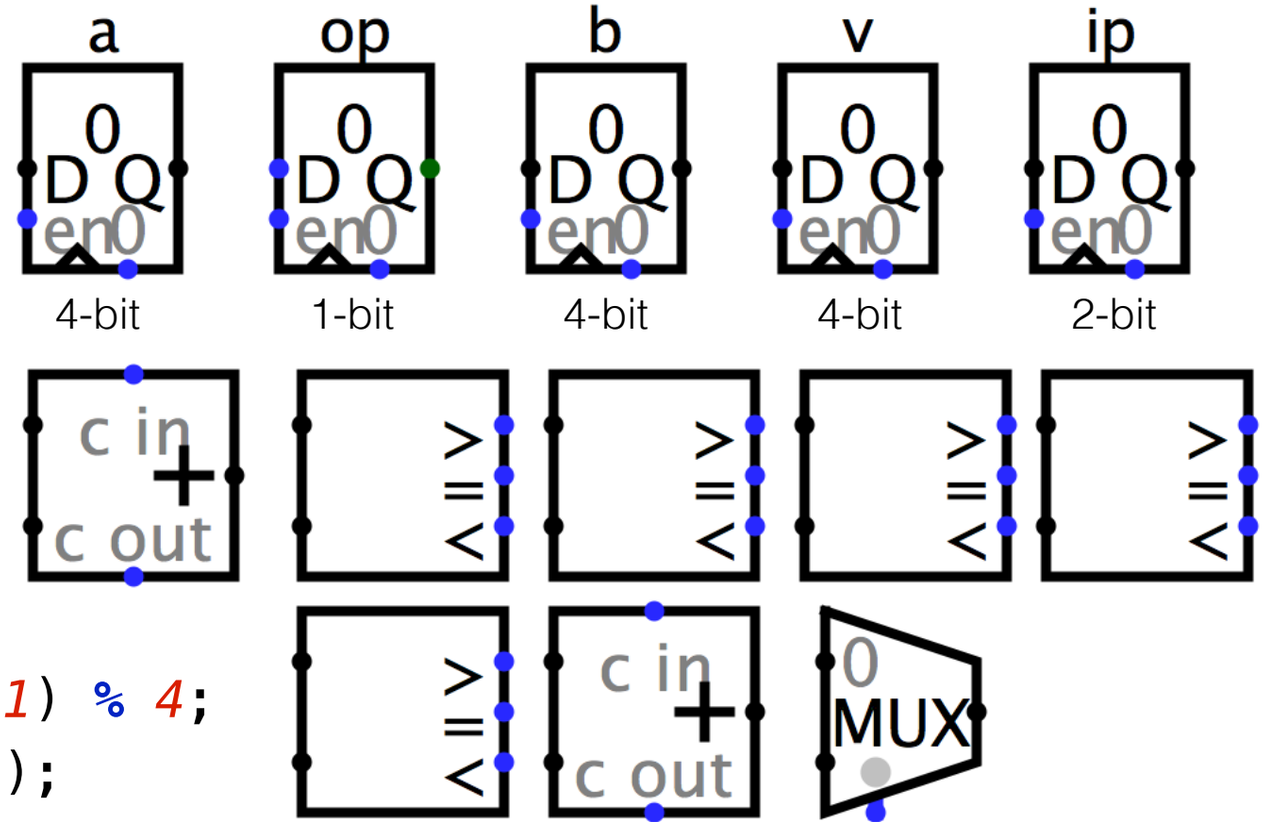
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



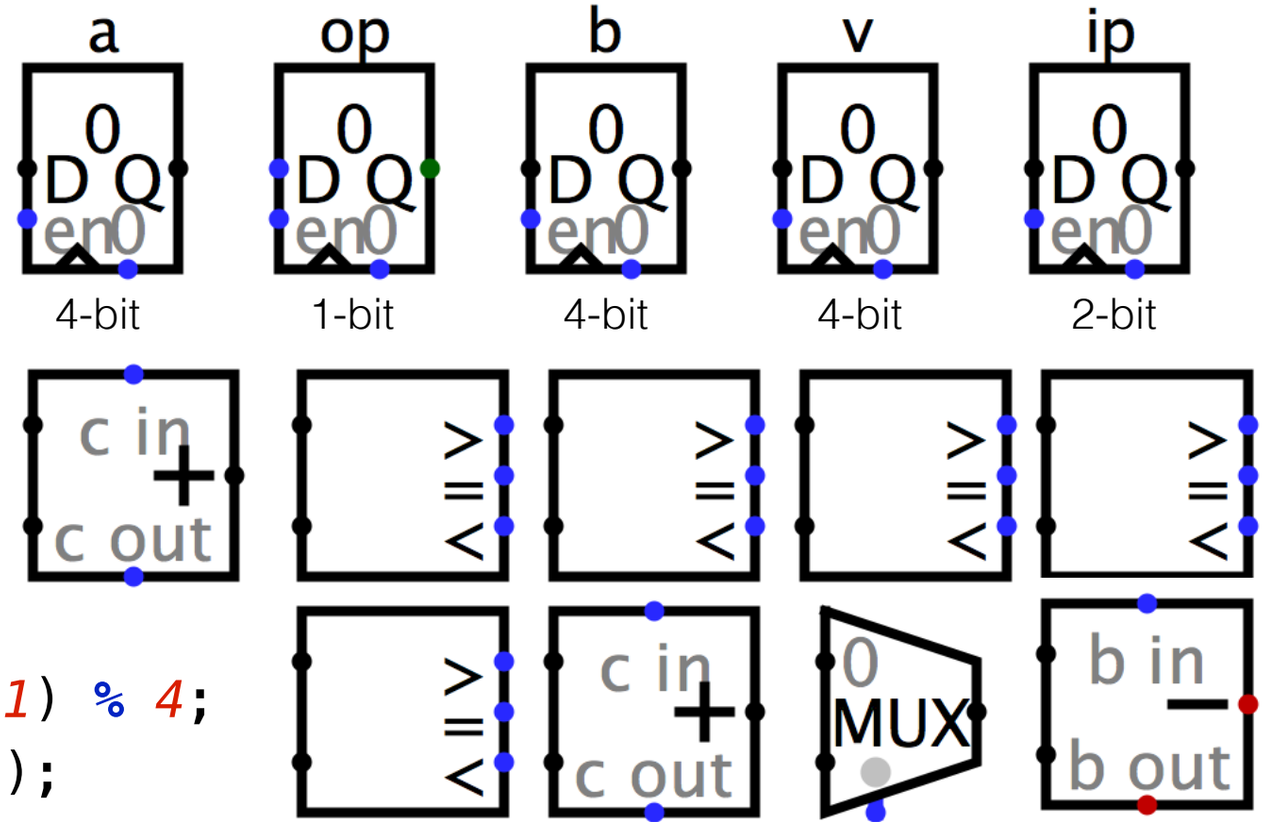
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



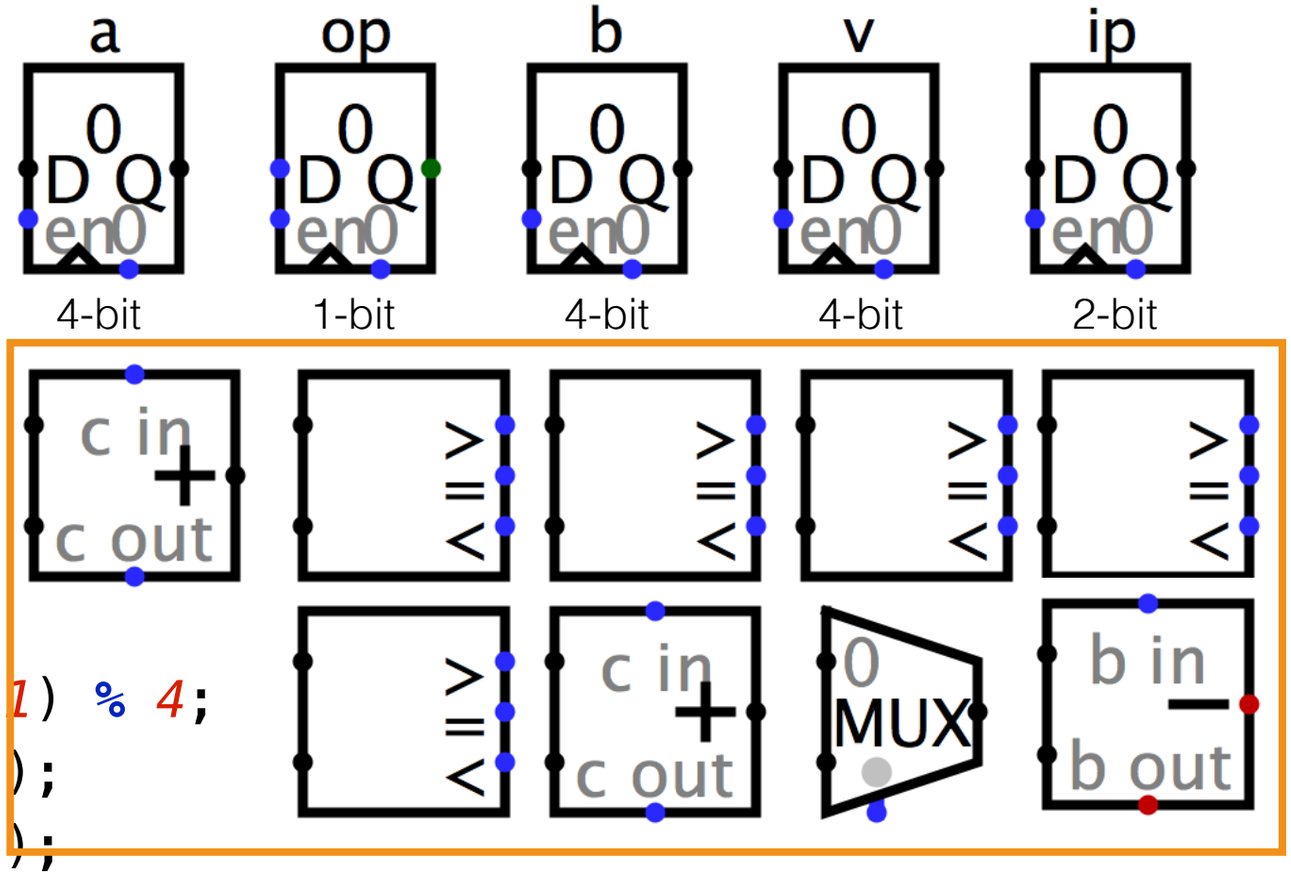
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



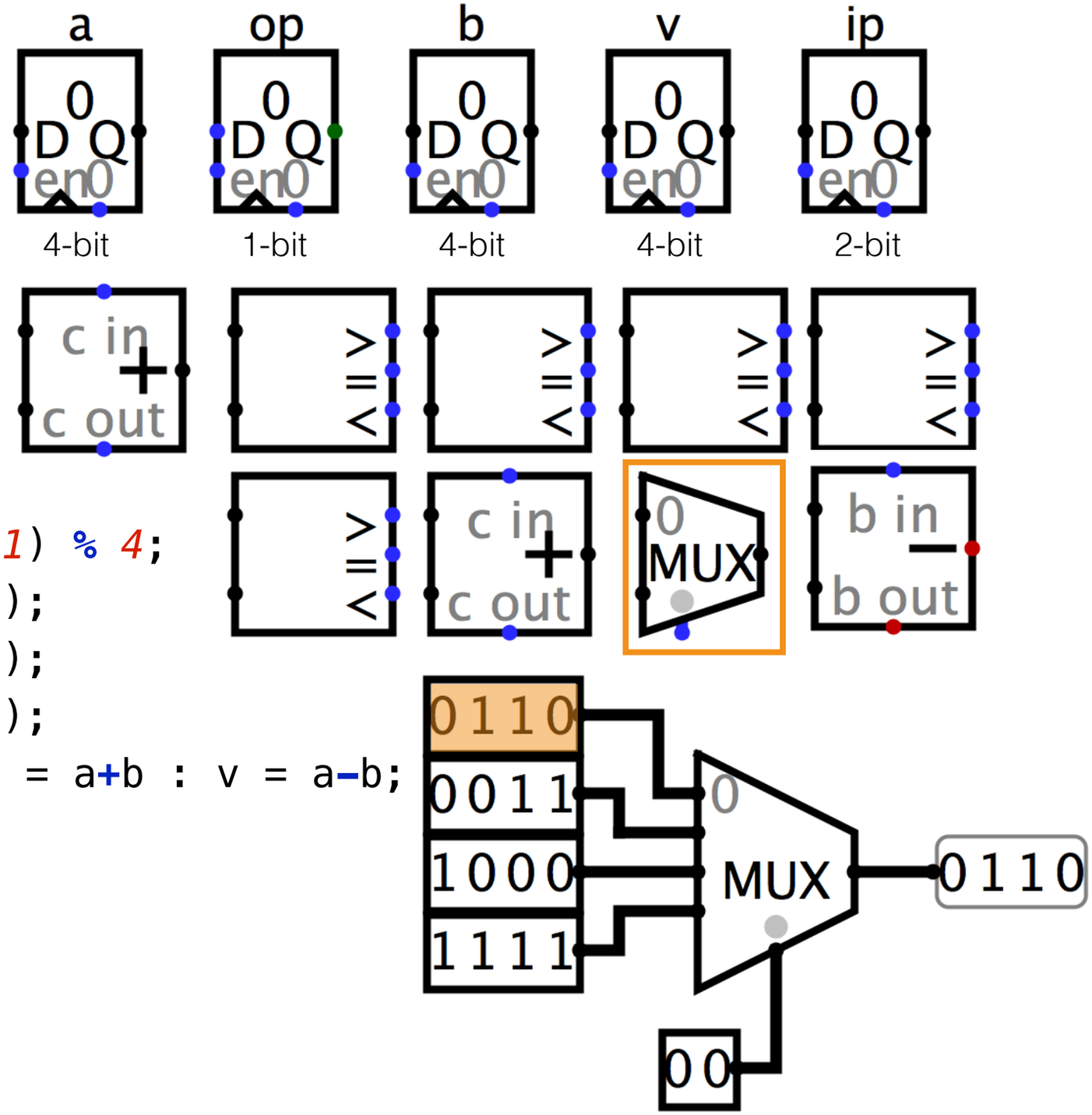
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



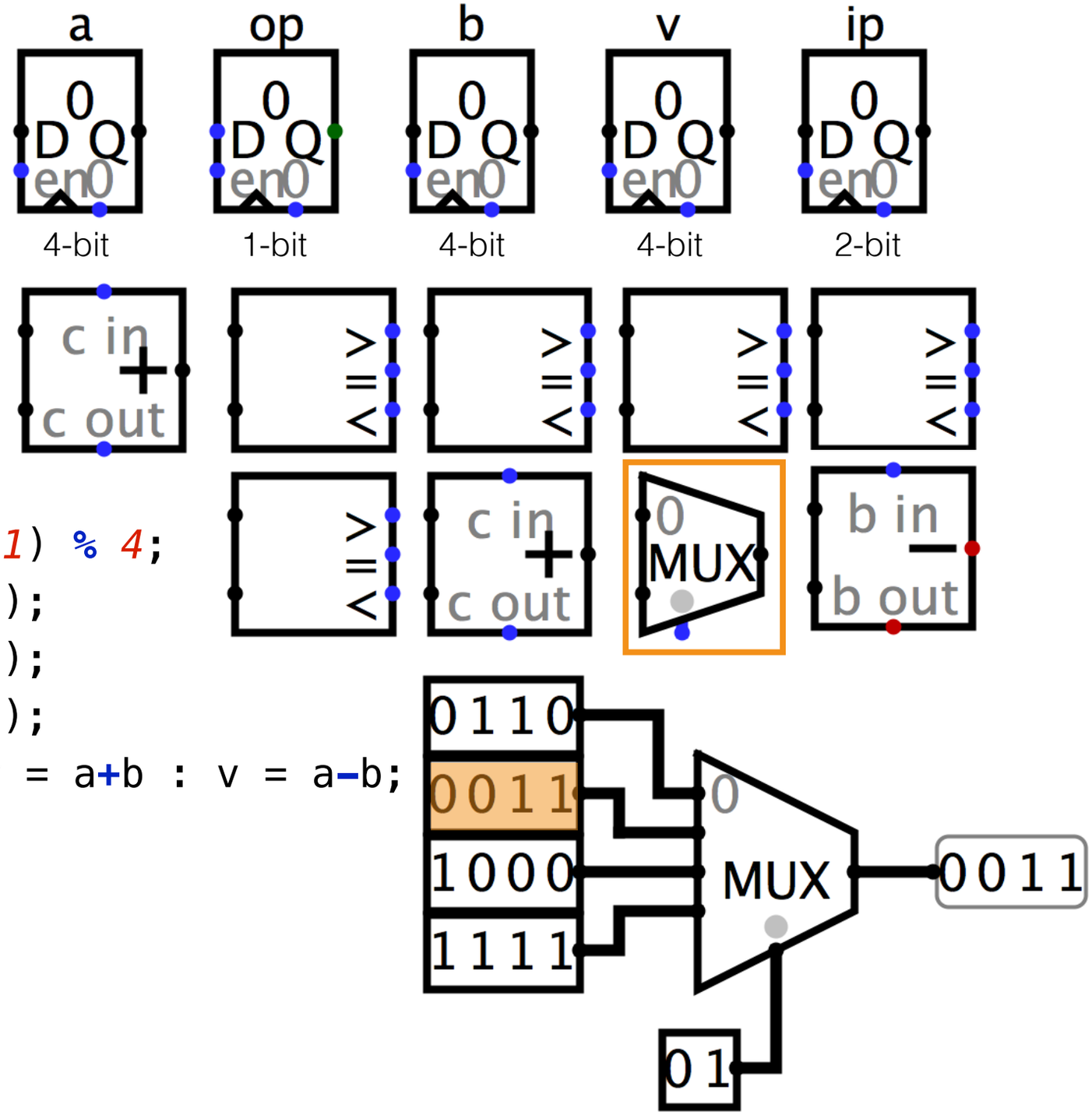
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



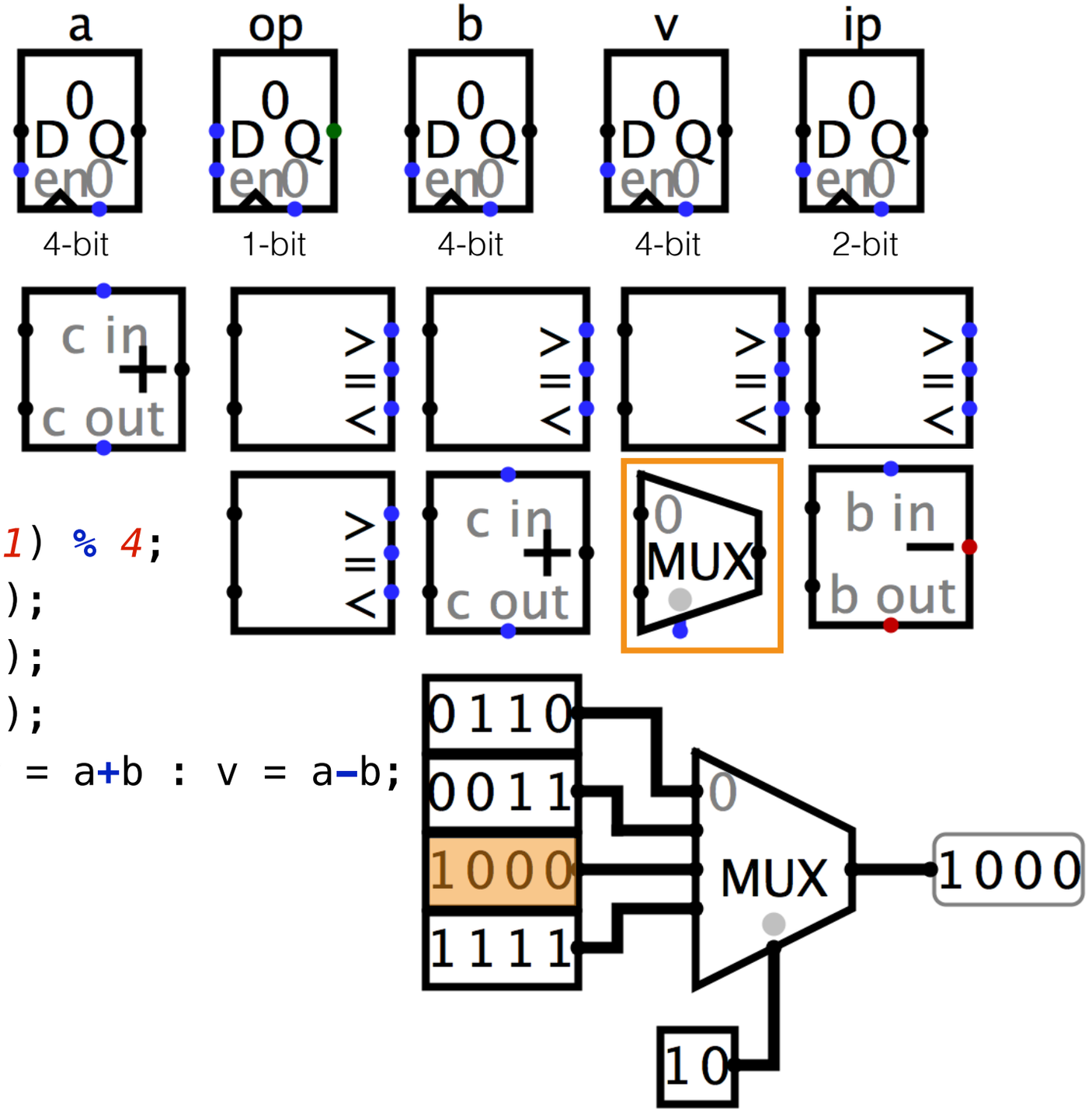
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```



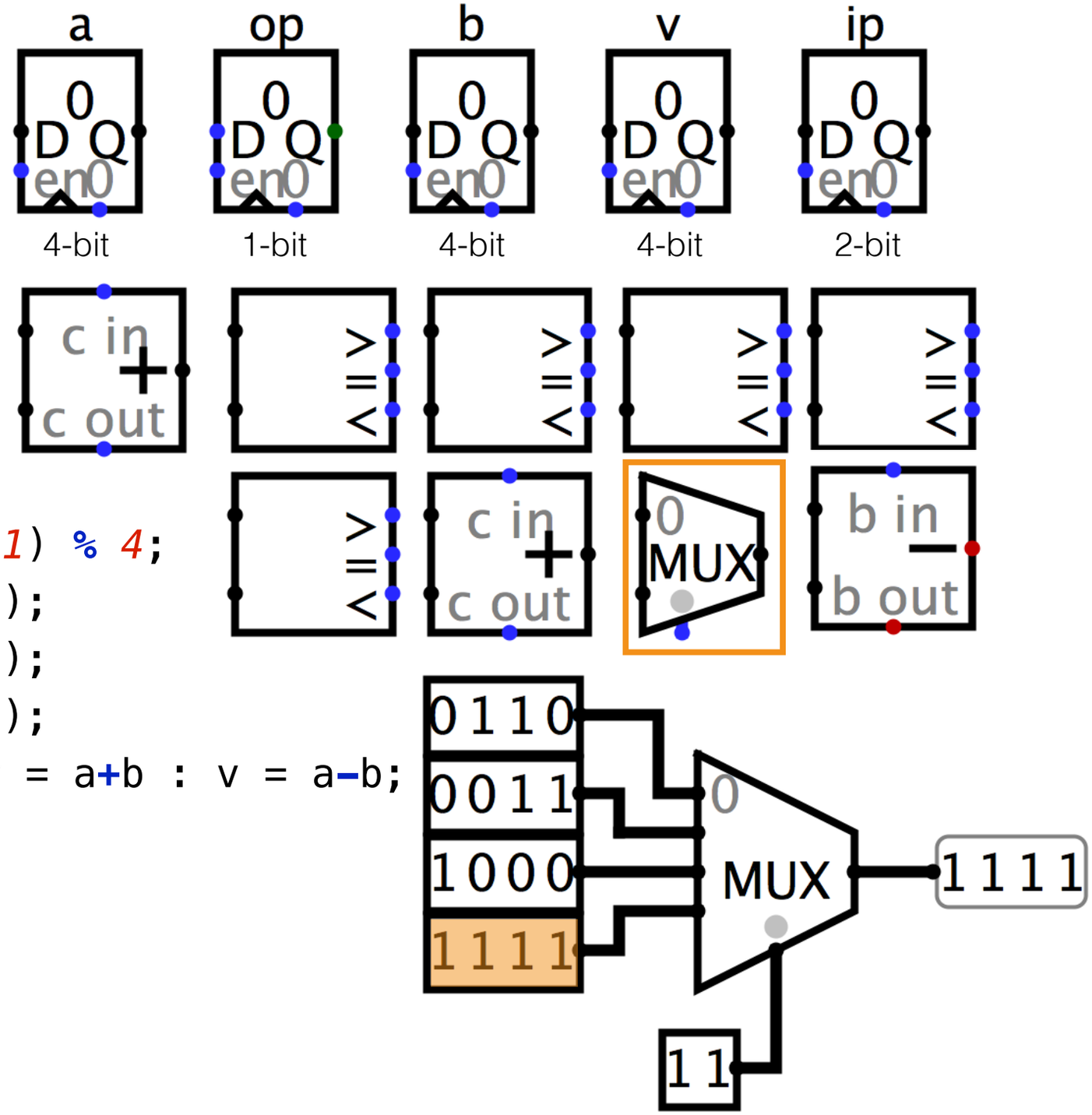
# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

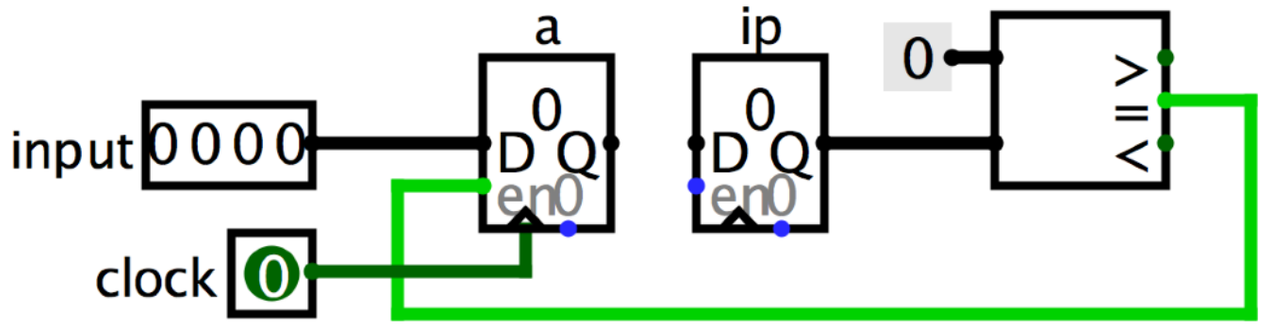


# Creating Hardware

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

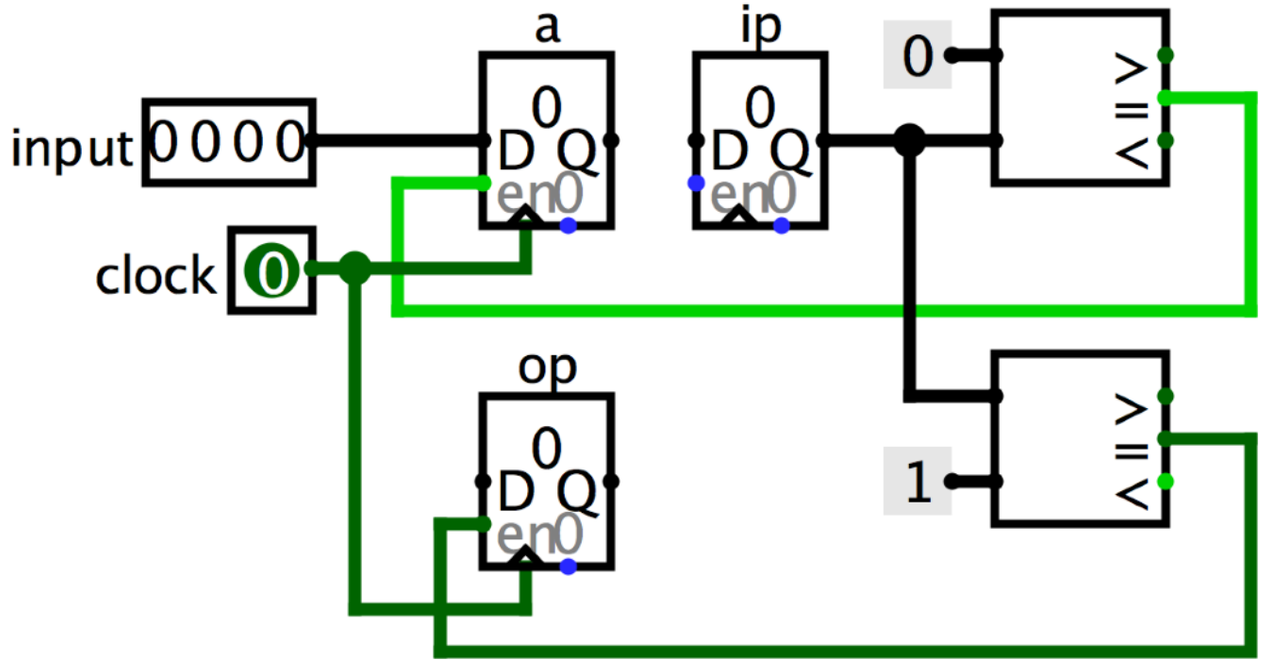


# Creating Hardware



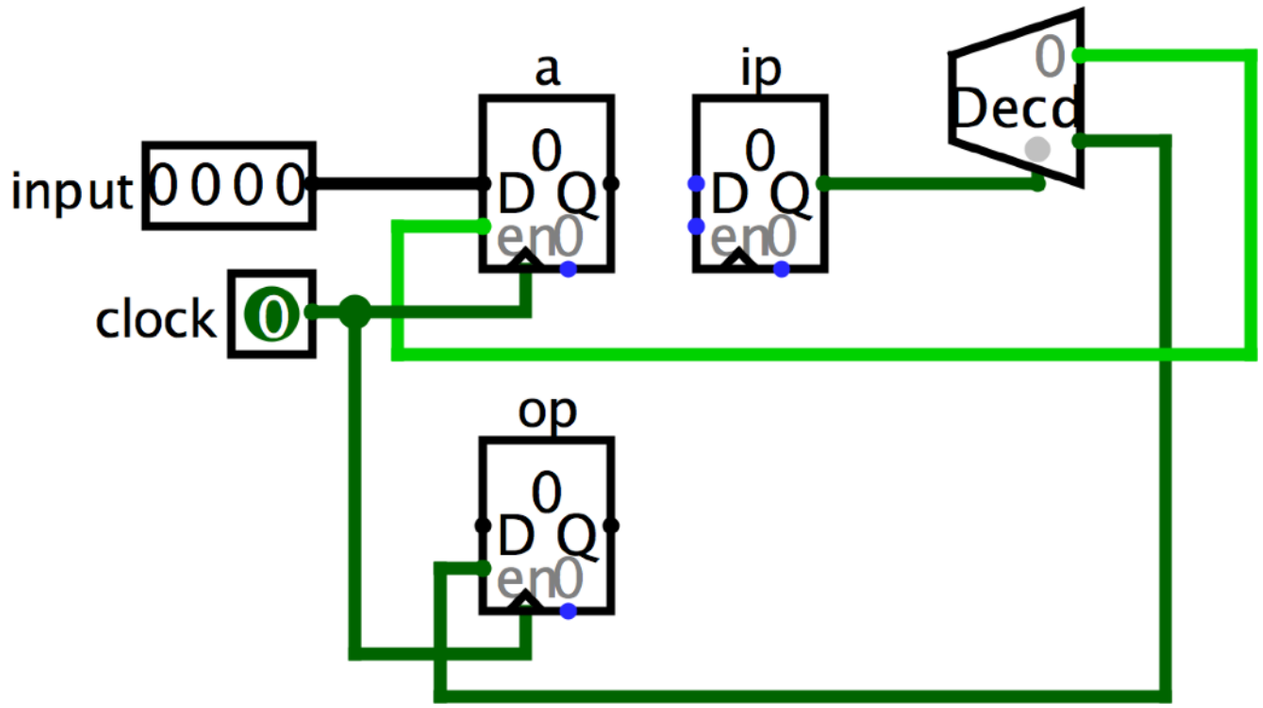

```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

# Creating Hardware




```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

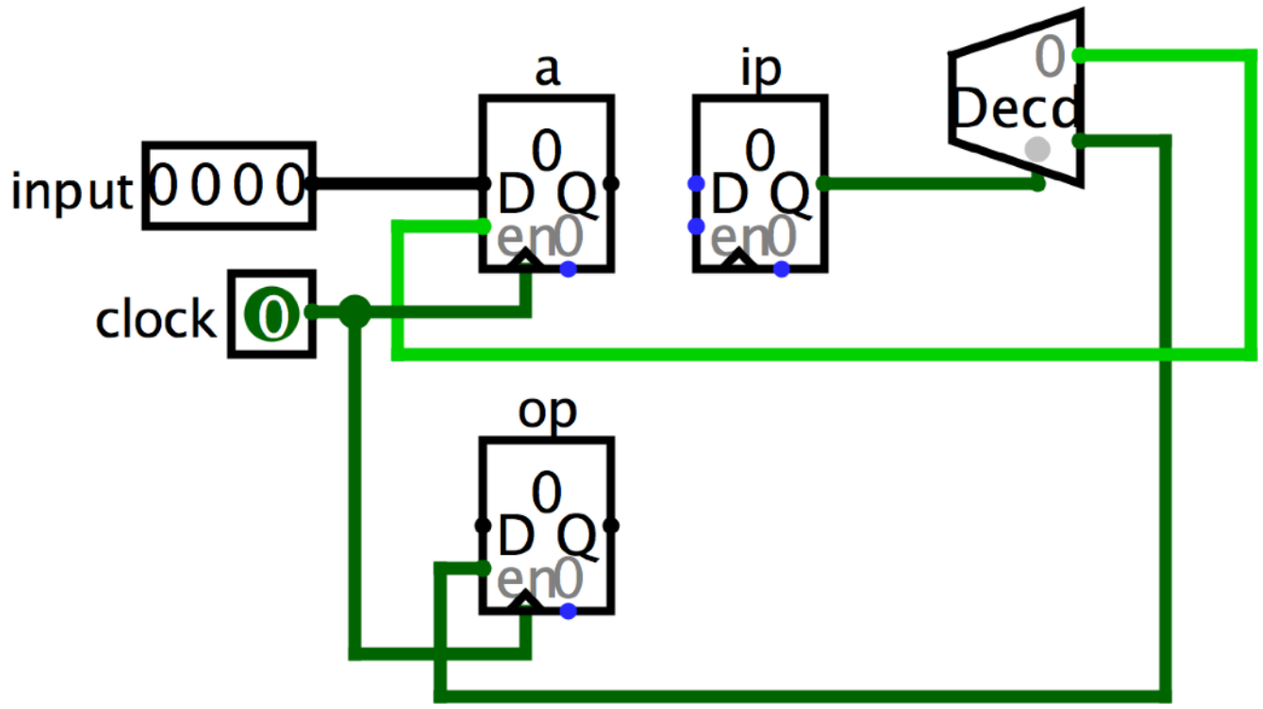
# Creating Hardware

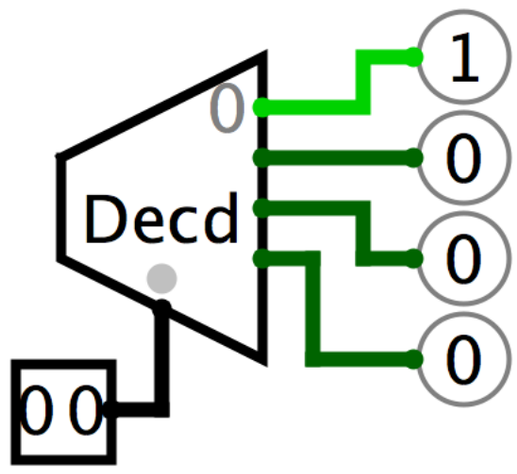



```
void main()
{
     int a, op, b, v, ip;

     while(true)
     {
                    ip = (ip + 1) % 4;
          if(ip == 0) a  = input();
          if(ip == 1) op = input();
          if(ip == 2) b  = input();
          if(ip == 3) op == 0 ? v = a+b : v = a-b;
     }
}
```

# Creating Hardware

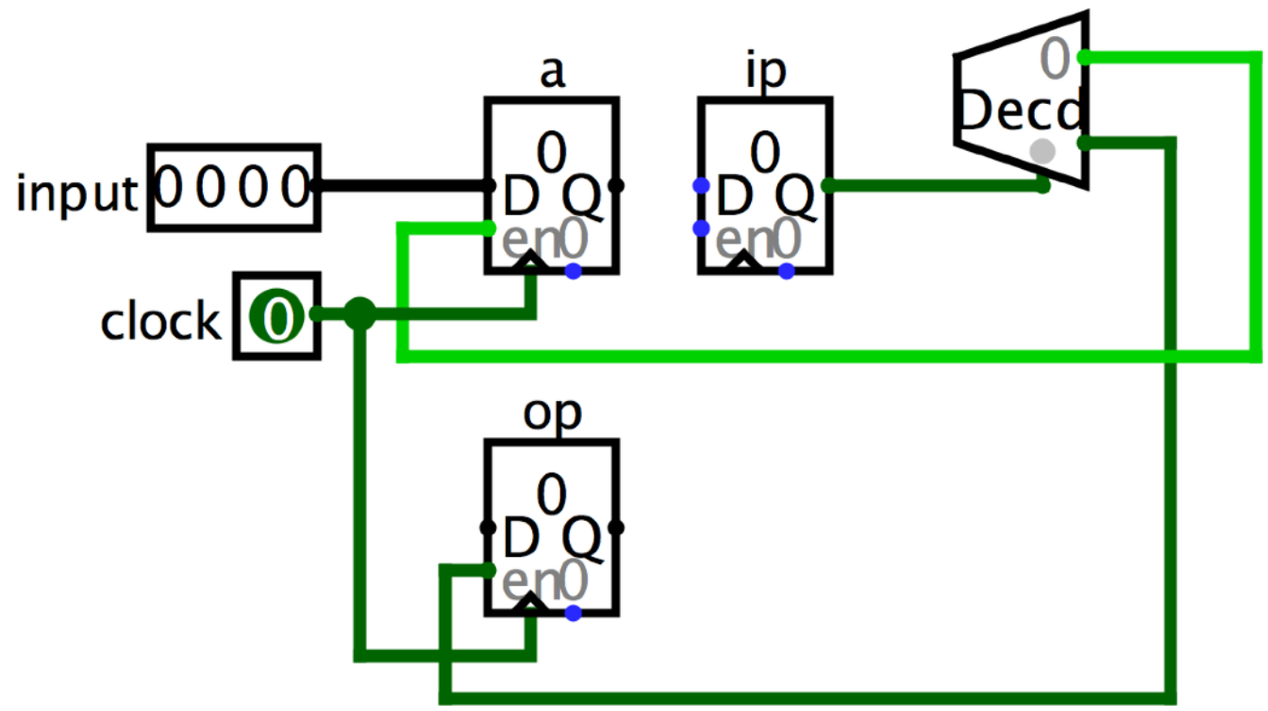
```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

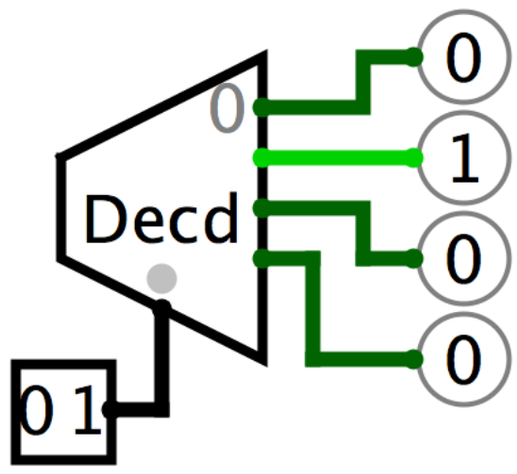


# Creating Hardware

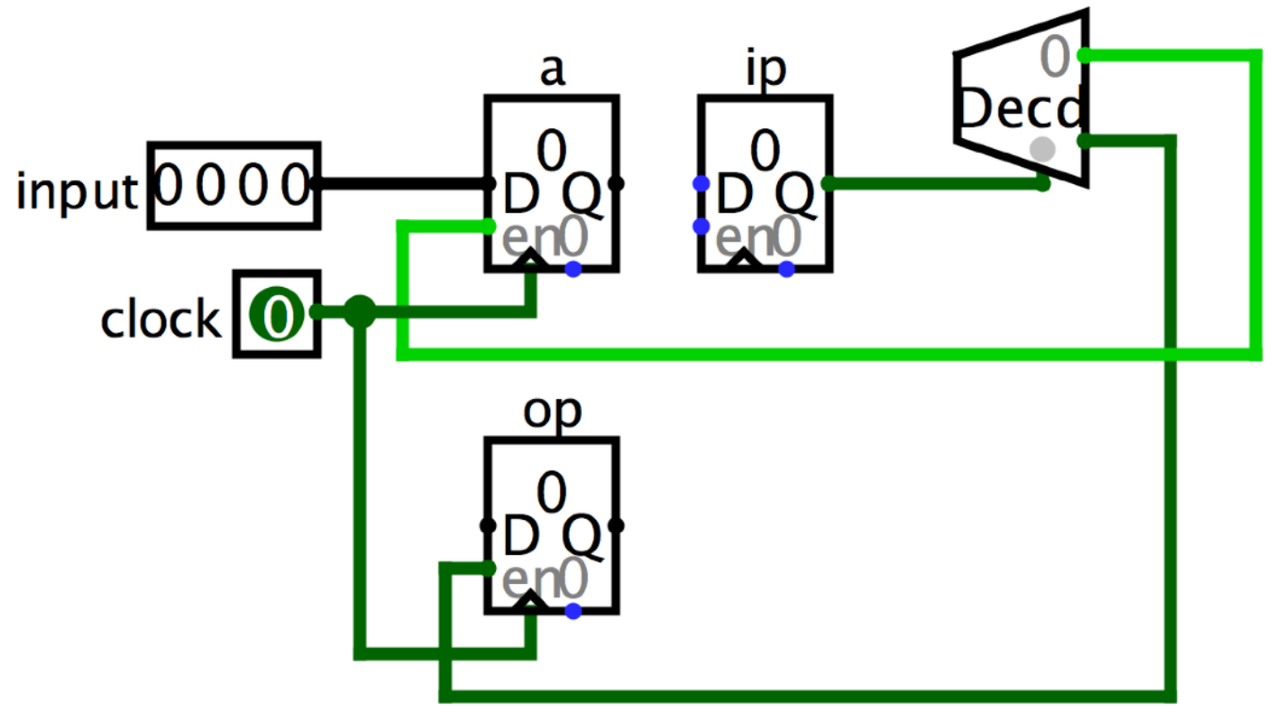
```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

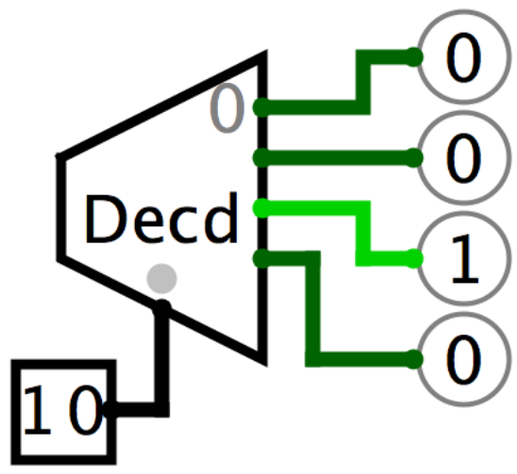


# Creating Hardware

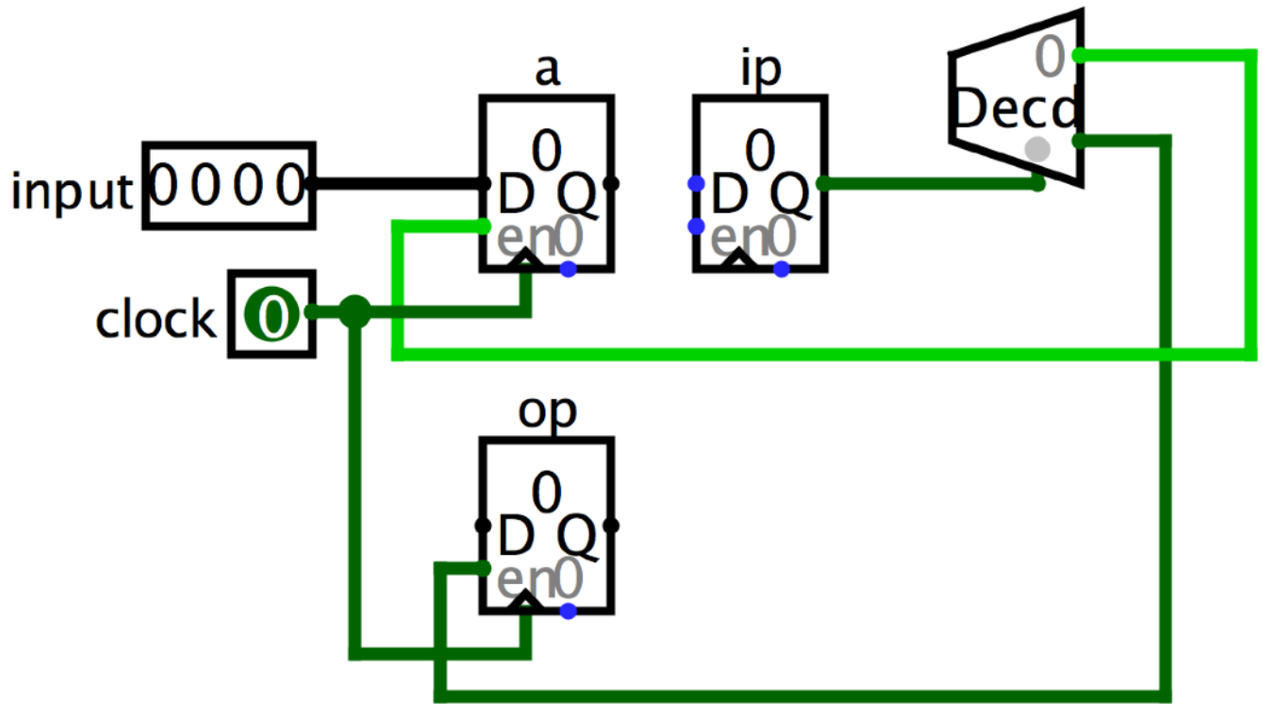
```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```

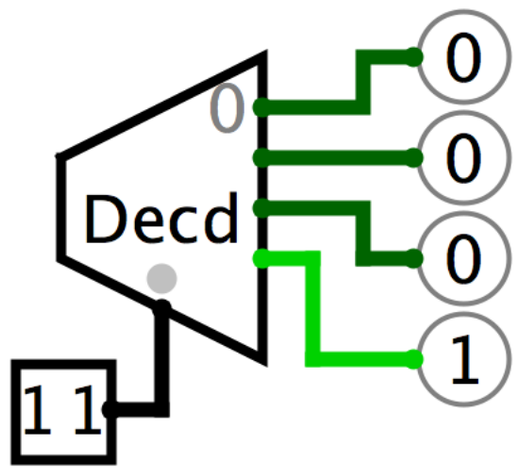


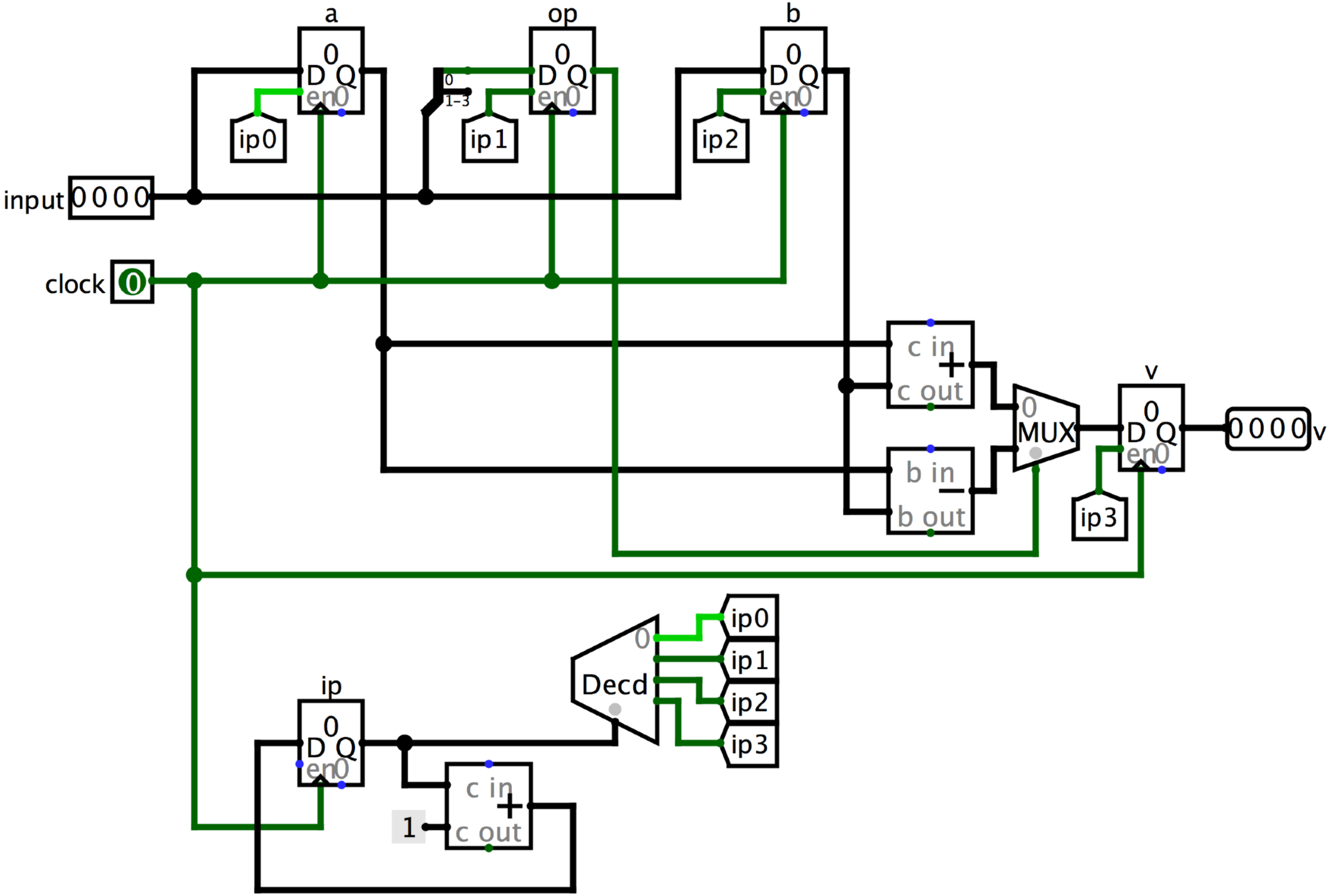
# Creating Hardware

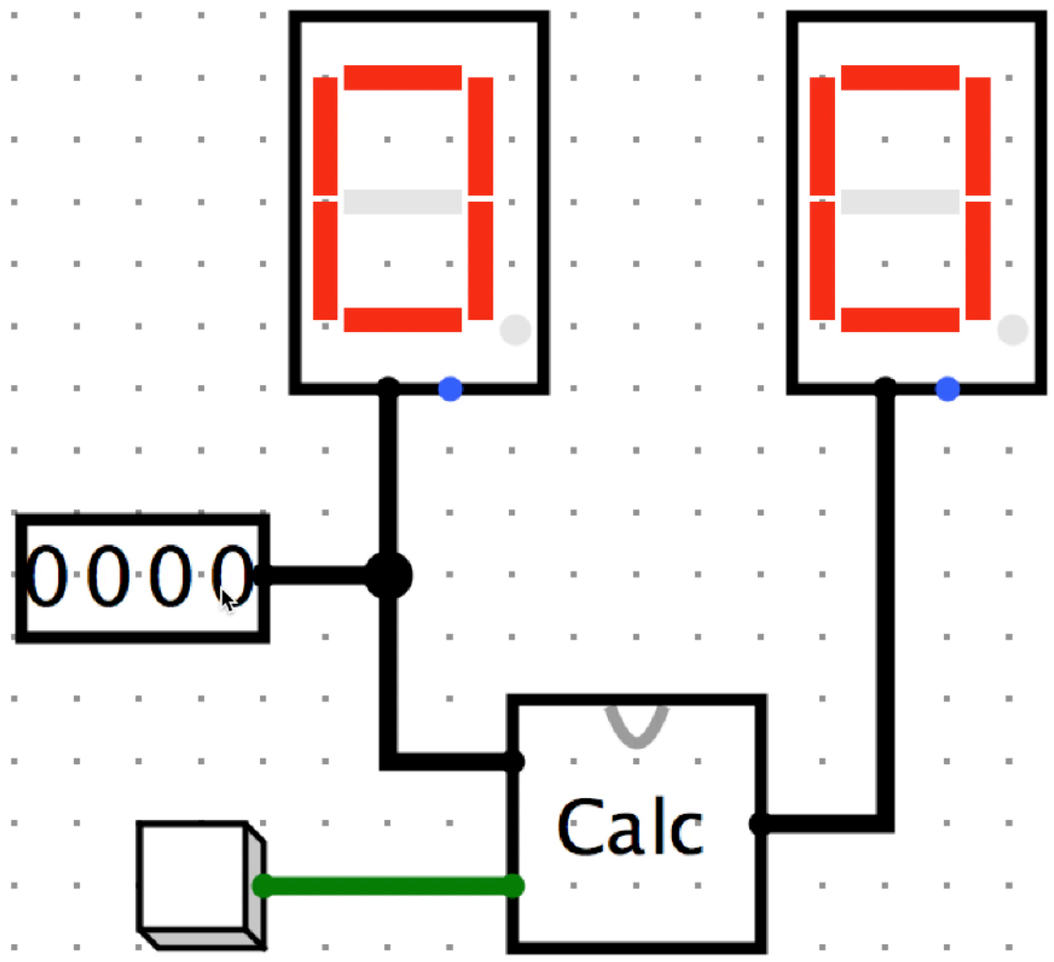
```
void main()
{
    int a, op, b, v, ip;

    while(true)
    {
                    ip = (ip + 1) % 4;
        if(ip == 0) a  = input();
        if(ip == 1) op = input();
        if(ip == 2) b  = input();
        if(ip == 3) op == 0 ? v = a+b : v = a-b;
    }
}
```




(one-hot encoding)


a
op
b
0
D Q
en
ip0
ip1
ip2
0
1-3
input
0000
clock
c in
c out
b in
b out
MUX
v
0000
v
ip3
ip
1
Decd
ip0
ip1
ip2
ip3

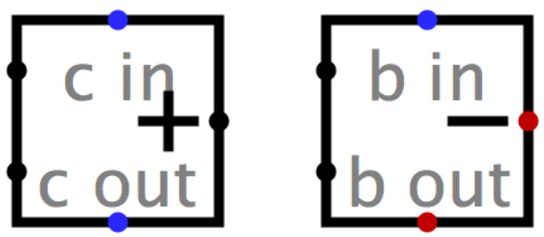



**Question 7**: There's an important optimization left to do in this circuit. What is it?

# Question 7

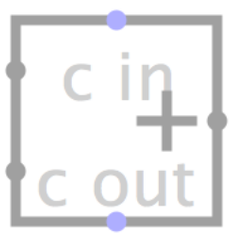
- Optimization: –x == ~x + 1; a+b == a+(~b)+1

```
if(ip == 3) op == 0 ? v = a+b : v = a-b;
```





```
if(ip == 3) v = a + (op == 0 ? b : ~b + 1);
```



# Question 8

- **Question 8**: do you know what an exclusive or (XOR) is?

  [ ] Yes
  [ ] No
  [ ] Is this a trick question?

# Question 8

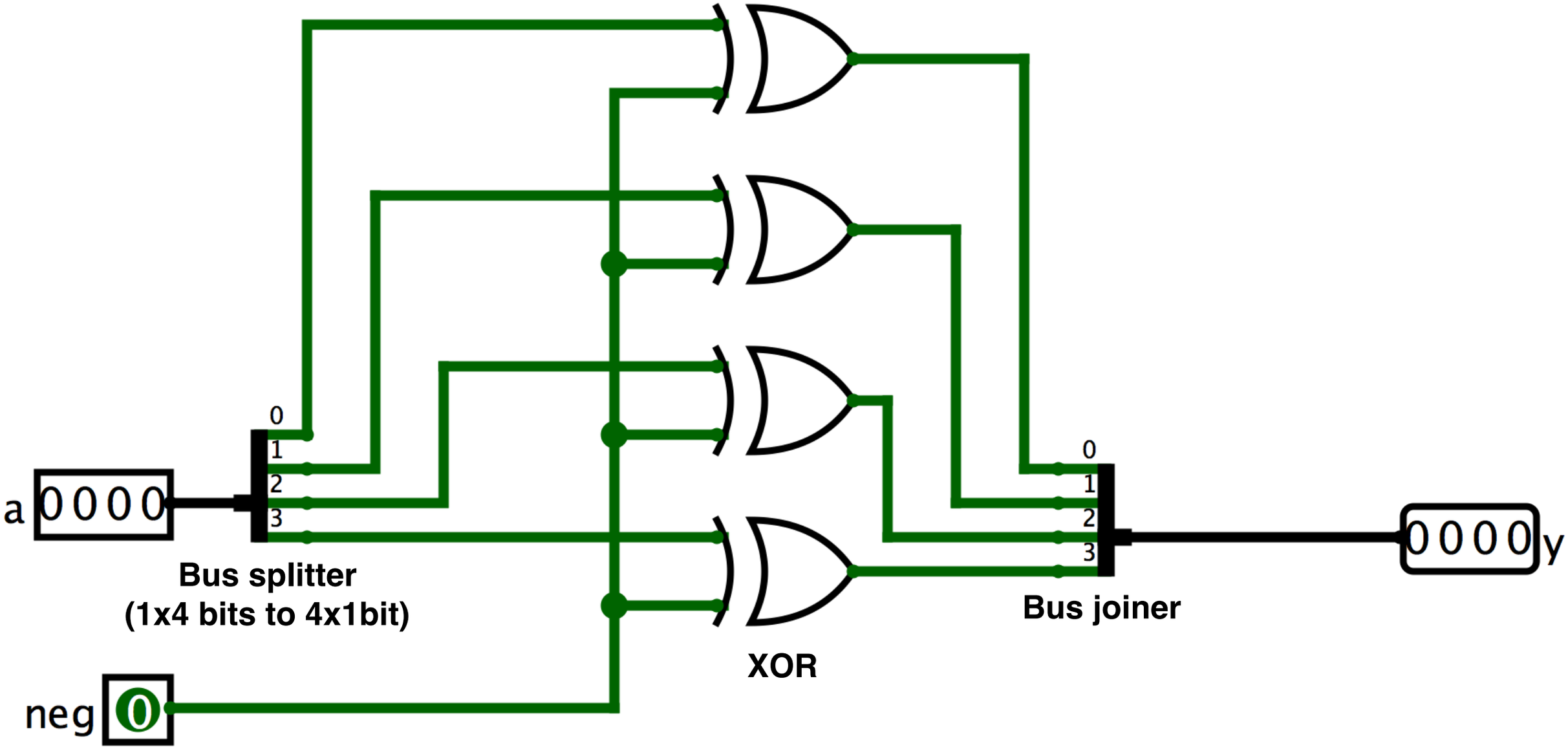
- **Question 8**: do you know what an exclusive or (XOR) is?

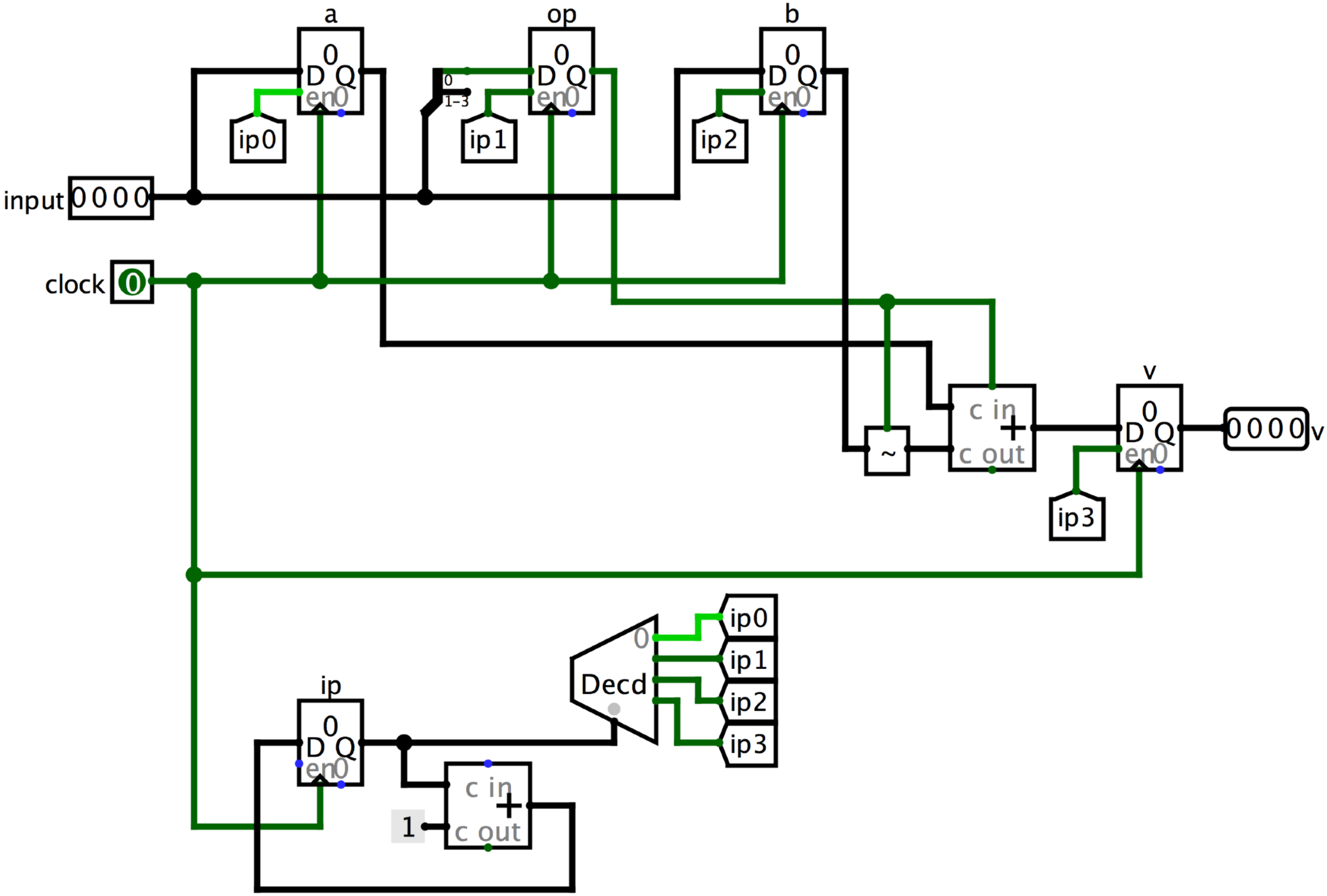
[ ] Yes
[ ] No
[ ] Is this a trick question?

```
if(ip == 3) v = a + (op == 0 ? b : ~b + 1);
```

invert B?

| INPUT | | OUTPUT |
|---|---|---|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

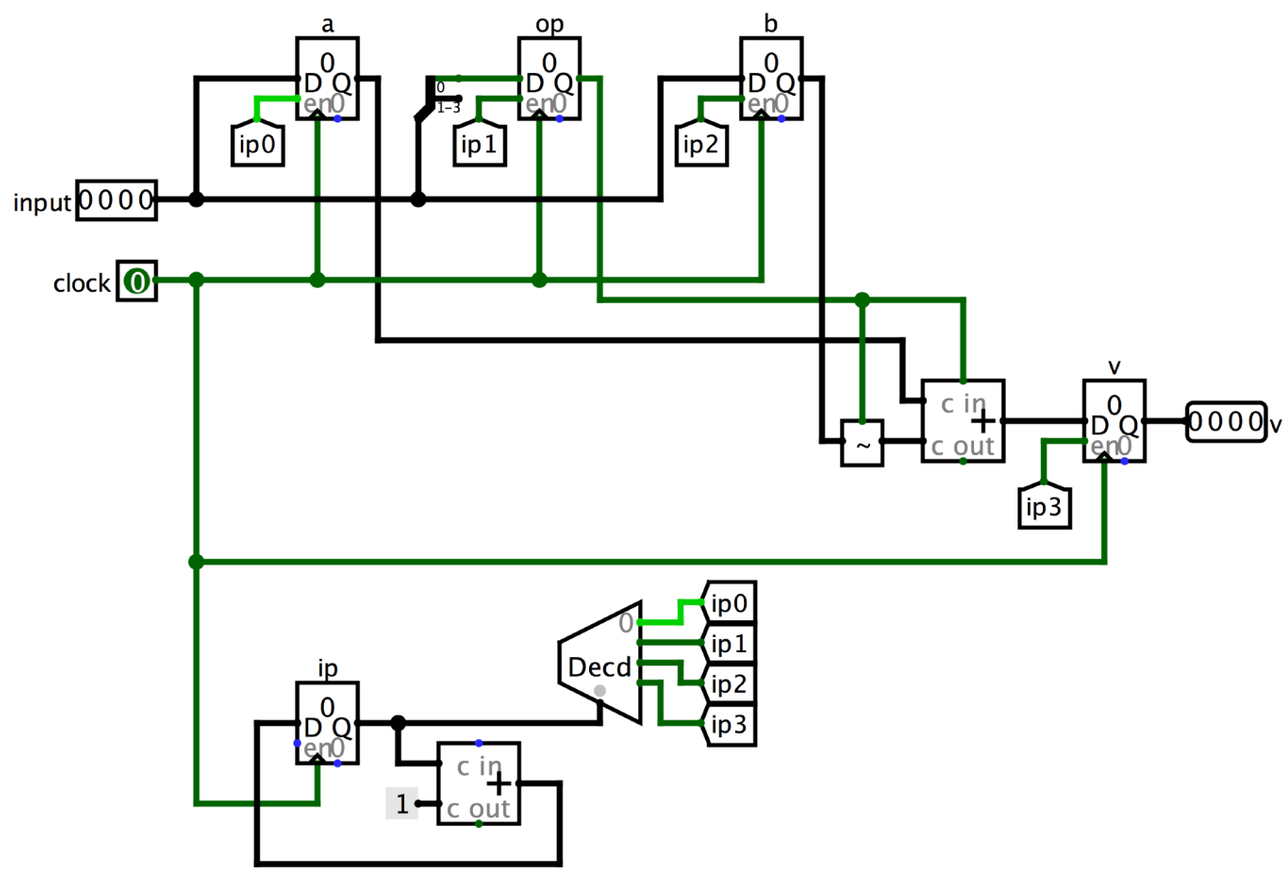

0
1
2
3
a
0 0 0 0
Bus splitter
(1x4 bits to 4x1bit)
neg
0
XOR
0
1
2
3
0 0 0 0
y
Bus joiner


a
op
b
0
D Q
en
ip0
ip1
ip2
ip3
input
0000
clock
0
1-3
~
c in
c out
+
v
0000
v
ip
1
Decd

# Creating Hardware

- Approach 1: Describe the algorithm, derive the circuit from the description (high level synthesis)

```
int a, op, b, v;

while(true)
{
     a  = getchar();
     op = getchar();
     b  = getchar();
     if(op == '+') v = a+b-'0';
     if(op == '-') v = a-b+'0';
     putchar(v);
     putchar('\n'); getchar();
}
```



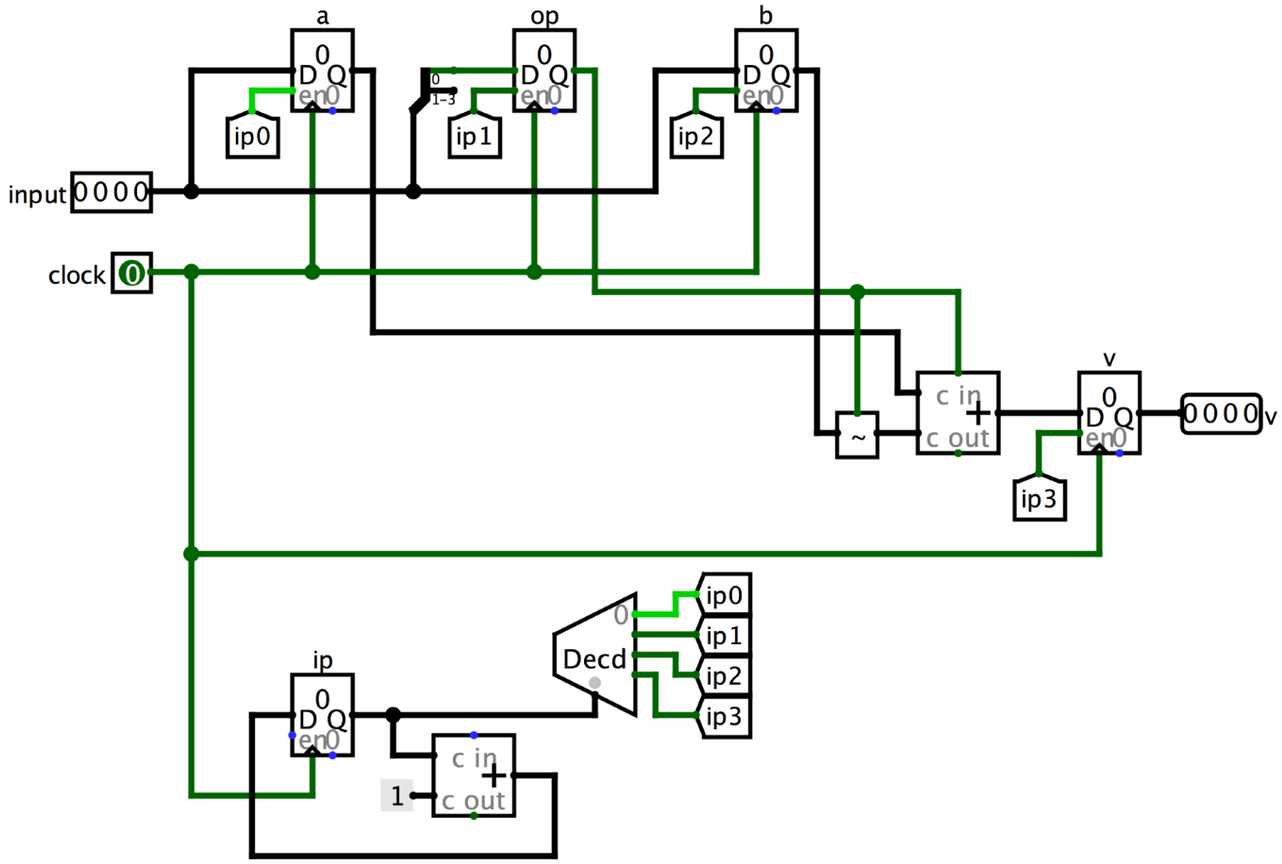
# Creating Hardware

- Approach 2: Write an event-based simulation of your design. Let the tools infer what hardware would be consistent with that simulation.
  - In practise, only the simulations for which there is an obvious mapping to hardware work. That's the so-called synthesizable subset.

# Creating Hardware

- Approach 3: Describe the circuit (RTL generator)

```
void main()
{
    auto a = new Reg();
    auto op = new Reg();
    auto b = new Reg();
    auto ip = new Reg();
    auto adder = new Adder;
    adder.connect(a, b);
    auto dec = new Decoder();
    dec.connect(ip);
    // ... etc
}
```



# Creating Hardware

- Approach 1: High-level synthesis
  Approach 2: Event-based
  Approach 3: RTL builder

- VHDL & Verilog: Follow mostly approach 2

- DHDL (and Chisel, SpinalHDL, etc.): Follows mostly approach 3

- Not pure: You don't have to specify events / gates for combinational circuits, just write `(a + b) * 7` etc.

# DHDL

- What is DHDL?
  - A hardware design language
  - D Language extension + D library
  - Falls more on the RTL generator side (currently)
- Why DHDL?
  - Modern design

# Modern Design

- `int sum(int[] values);`

Level 0: one tool does one job

- `int mul(int[] values);`

Level 0: one tool does one job

- `auto accumulate(alias op, T)(T[] values);`

Level 1: one tool does multiple jobs

Level 1: generates multiple tools, each of which do one job

# Modern Design

**Question 9**:
At what level do hardware designers work?

# Level -1

(sometimes…)

# Modern Design

- Level -1:

- Level 0: one tool does one job
  (e.g. max computes maximums)

- Level 1: one tool does multiple jobs
  (e.g. accumulate computes sums, maximums, etc.)

# Modern Design

- Level -1: multiple tools needed to do one job (`sum_part1()`, `sum_part2()`?)

- Level 0: one tool does one job
  (e.g. max computes maximums)

- Level 1: one tool does multiple jobs
  (e.g. accumulate computes sums, maximums, etc.)

# HDL Woes

- VHDL was created to create simulations of hardware
- After debugging the design, the hardware was built by converting to a schematic, where you wired simple gates, etc.
- Why not make the conversion automatic?
- Hardware engineers were used to schematic entry as a design method. VHDL was nailed with that hammer.
- Lots of little blocks without obvious relationship

# HDL Woes

- void sum_part1()
  {
  thread
  }

- void sum_part2()
  {
  thread
  }

# HDL Woes

```
CBandDatat_LatchPROC9F: process(MDLE, CB_In, Reset_Out_N)
 begin
  if Reset_Out_N = '0' then
     CBLatch_F_1       <= "0000";
  elsif MDLE = '1' then
     CBLatch_F_1       <= CB_In(3 downto 0);
  end if;
 end process;


 CBandDatat_LatchPROC10F: process(MDLE, CB_In, DParIO_In, Reset_Out_N)
 begin
  if Reset_Out_N = '0' then
     CBLatch_F_2       <= "0000";
  elsif MDLE = '1' then
     CBLatch_F_2(6 downto 4) <= CB_In(6 downto 4);
     CBLatch_F_2(7)          <= DParIO_In;
  end if;
 end process;

CBLatch_F <= CBLatch_F_2 & CBLatch_F_1;


  BullEn_PROC: process(Mem_Test, ByteSel)
  begin
   BullEn <= not (ByteSel(0) and ByteSel(1) and ByteSel(2) and ByteSel(3)) or Mem_Test;
  end process;

  IUChk_Out_Gen: process (IUDataLatch_F, ChkGen_Data, BullEn, CB_bull)
   Variable IUGen_Chk  : std_logic_vector(7 downto 0);

  begin
    IUGen_Chk(6 downto 0) := ChkGen (IUDataLatch_F);
    IUGen_Chk(7)          := ChkGen_Data(32);
   CB_Out_Int <= mux2 (BullEn, IUGen_Chk, CB_bull);
  end process;
```

# HDL Woes

- The argument is:
  - VHDL was originally created for simulation
    - Synthesis was added later
    - Problems arose from that
- But then at least things related to simulation should be pretty good, right?

# HDL Woes

```
entity hello_world is
end hello_world;

architecture behaviour of hello_world is
    signal foo : std_logic;
begin
    process begin
        foo <= '1';
        wait for 1 ns;
        report std_logic'image(foo);
        foo <= '0';
        wait for 1 ns;
        report std_logic'image(foo);
       wait;
    end process;
end behaviour;
```

```
$ ghdl -a hello.vhdl && ghdl -e hello_world
$ ./hello_wold
hello.vhdl:13:8:@1ns:(report note): '1'
hello.vhdl:16:8:@2ns:(report note): '0'
```

# HDL Woes

```
entity hello_world is
end hello_world;

architecture behaviour of hello_world is
    signal foo : std_logic_vector(7 downto 0);
begin
    process begin
        foo <= "11111111";
        wait for 1 ns;
        report std_logic_vector'image(foo);
        foo <= "00000000";
        wait for 1 ns;
        report std_logic_vector'image(foo);
       wait;
    end process;
end behaviour;
```

```
$ ghdl -a hello.vhdl && ghdl -e hello_world
hello.vhdl:13:32: prefix of 'image
attribute must be a scalar type
hello.vhdl:13:32: found array type
"std_logic_vector" defined at ../../../src/
ieee/std_logic_1164.v93:69:30
```

# HDL Woes


vhdl print std_logic_vector

Google Search  I'm Feeling Lucky

Google

vhdl print std_logic_vector

All Images News Videos More Settings Tools

About 11,300 results (0.68 seconds)

## How to Display std_logic_vector Value? - Google Groups

https://groups.google.com/d/topic/comp.lang.vhdl/ZkyIUFlulj4

Jan 15, 2001 - comp.lang.**vhdl** › ... How can I display the value of **std_logic_vector**? ... a) Both compilers complain data_out is incorrect type for **REPORT**.

## VHDL BLOG: Print to STDOUT using REPORT Statement in VHDL

sagekingthegreat.blogspot.com/2013/08/report-statement-report-statement-is.html

Aug 12, 2013 - variable B : std_logic := '1'; **report** "B value is" & std_logic'image(B); **output** > B value is 1. (C) To **print std_logic_vector** type values to STDOUT ...

## How do I use VHDL write() function to print std_logic_vector type ...

www.edaboard.com › ... › PLD, SPLD, GAL, CPLD, FPGA Design

May 23, 2012 - 3 posts - 2 authors

Can anyone tell me how to do it? I used to use **report**() function, but recently learned that **write**() can be used for **printing** things on a screen.

how to **write** ucf file in xilinx for **vhdl** code for an **...** 6 posts 28 Apr 2014
[SOLVED] decimal numbers in **vhdl** - EDA Board 12 posts 3 Aug 2011
More results from www.edaboard.com

## How to print std_logic_vector variable into hex string in VHDL ...

https://www.thecodingforums.com › Archive › Archive › VHDL

Aug 31, 2005 - 5 posts - 4 authors

How to **print std_logic_vector** variable into hex string in **VHDL** ... Is there any command like "**report** integer'image(myvalue)"; so that it will **print** ...

## vhdl - Is there a way to print the values of a signal to a file from a ...

stackoverflow.com/.../is-there-a-way-to-print-the-values-of-a-signal-to-a-file-from-a-...

Jun 20, 2014 - First make functions that convert std_logic and **std_logic_vector** to string like: function to_bstring(sl : std_logic) return string is variable sl_str_v ...

**pin...@my-deja.com** 1/16/01

You may want use this function:

```
-- By PK debug mean for printing std_logic
function prtstd( v : std_logic_vector ) return integer is
variable s : string( 3 downto 1 );
variable r : string( (v'left+1) downto (v'right+1) );
begin
  for i in v'left downto v'right loop
  --report std_logic'image(v(i));
    s := std_logic'image(v(i));
    --string must start/stop at 1
    --          '1' we need only the second character
     r(i+1) := s(2);
   end loop;
   report "dbg by pk " & r & " dbg end";
   return 0;
 end prtstd;
```

Call it by:
junk := prtstd( FileDesc.CurrentWord );

In article <940isc$qfe$1...@nnrp1.deja.com>,

- show quoted text -

vhdl print std_logic_vector

All Images News Videos More Settings Tools

About 11,300 results (0.68 seconds)

## How to Display std_logic_vector Value? - Google Groups

https://groups.google.com/d/topic/comp.lang.vhdl/ZkyIUFlulj4

Jan 15, 2001 - comp.lang.**vhdl** › ... How can I display the value of **std_logic_vector**? ... a) Both compilers complain data_out is incorrect type for **REPORT**.

## VHDL BLOG: Print to STDOUT using REPORT Statement in VHDL

sagekingthegreat.blogspot.com/2013/08/report-statement-report-statement-is.html

Aug 12, 2013 - variable B : std_logic := '1'; **report** "B value is" & std_logic'image(B); **output** > B value is 1. (C) To **print std_logic_vector** type values to STDOUT ...

## How do I use VHDL write() function to print std_logic_vector type ...

www.edaboard.com › ... › PLD, SPLD, GAL, CPLD, FPGA Design

May 23, 2012 - 3 posts - 2 authors

Can anyone tell me how to do it? I used to use **report**() function, but recently learned that **write**() can be used for **printing** things on a screen.

how to **write** ucf file in xilinx for **vhdl** code for an ... 6 posts 28 Apr 2014
[SOLVED] decimal numbers in **vhdl** - EDA Board 12 posts 3 Aug 2011
More results from www.edaboard.com

## How to print std_logic_vector variable into hex string in VHDL ...

https://www.thecodingforums.com › Archive › Archive › VHDL

Aug 31, 2005 - 5 posts - 4 authors

How to **print std_logic_vector** variable into hex string in **VHDL** ... Is there any command like "**report** integer'image(myvalue)"; so that it will **print** ...

## vhdl - Is there a way to print the values of a signal to a file from a ...

stackoverflow.com.../is-there-a-way-to-print-the-values-of-a-signal-to-a-file-from-a-...

Jun 20, 2014 - First make functions that convert std_logic and **std_logic_vector** to string like: function to_bstring(sl : std_logic) return string is variable sl_str_v ...

# HDL Woes

3

As you have discovered, the 'image attribute is only declared for scalar types, not arrays or records : the usual approach is to create one's own library of test utilities including `to_string` or `image` functions in a package at the start of a design, and use it throughout.

It would be perfectly possible to standardise a library of these, and you will probably find many potential "test utility" packages but none has really caught on well enough to deserve becoming a standard.

That being said, you might find the following package a useful starting point.

- wat?
- Is this what the competition looks like?
  You have to create your own writeln?

# HDL Woes

- More seriously, here are the fundamental problems:
  - VHDL & Verilog are event-based
    - Only a subset is synthesizable
      - Only a subset of that subset actually works in practise.
      - Which sub-sub-set? Oh, it depends on the tool vendor!
  - No OOP, FP or generics. Everything is verbose, repetitive and error prone

# HDL Woes

- Only VHDL and Verilog are mainstream, but there are some competitors:
  - Bluespec, Lava, Hydra, ForSyDe, etc. (Haskell)
  - Chisel / SpinalHDL (Scala)
  - MyHDL (Python)
  - CLaSH
  - Lucid

# Why DHDL

- DHDL
  - Chisel's hardware construction model…
  - …without the Scala DSL tax
  - …with D's modeling power

# Why DHDL

- Scala DSL tax

```
if(a == false)
     b := 42;
else
     c := 0;
```

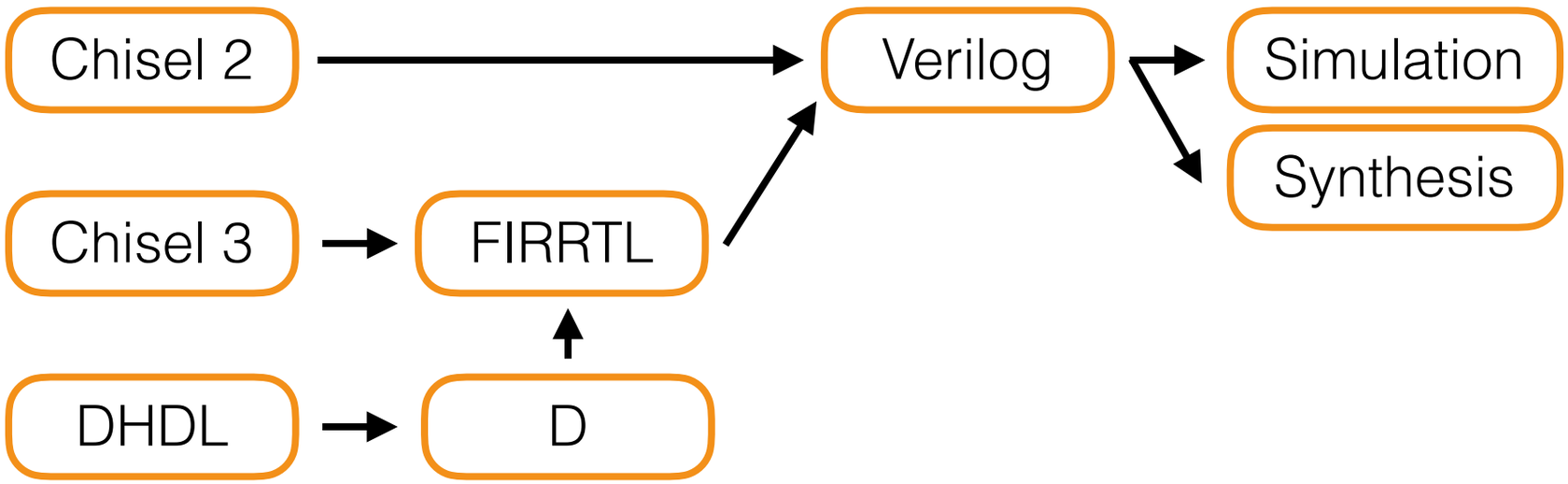
(DHDL)

```
when(a === false.B)
{
     b := 42.U
}
.otherwise
{
     c := 0.U
}
```

(Chisel)

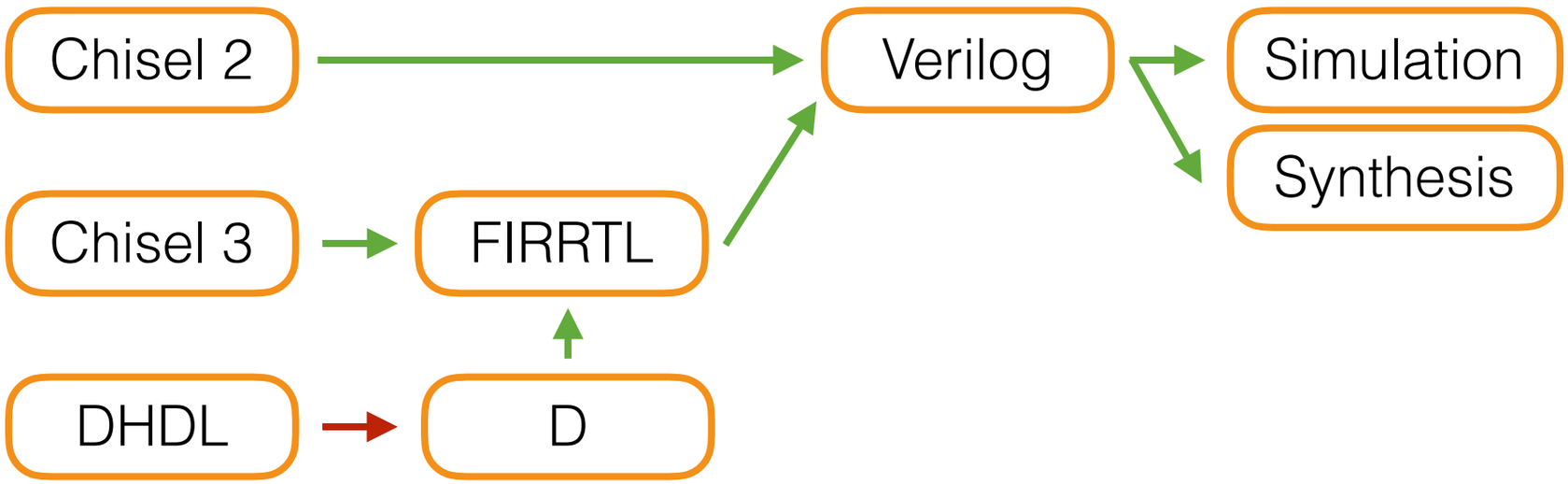
# Why DHDL

- Compiling DHDL



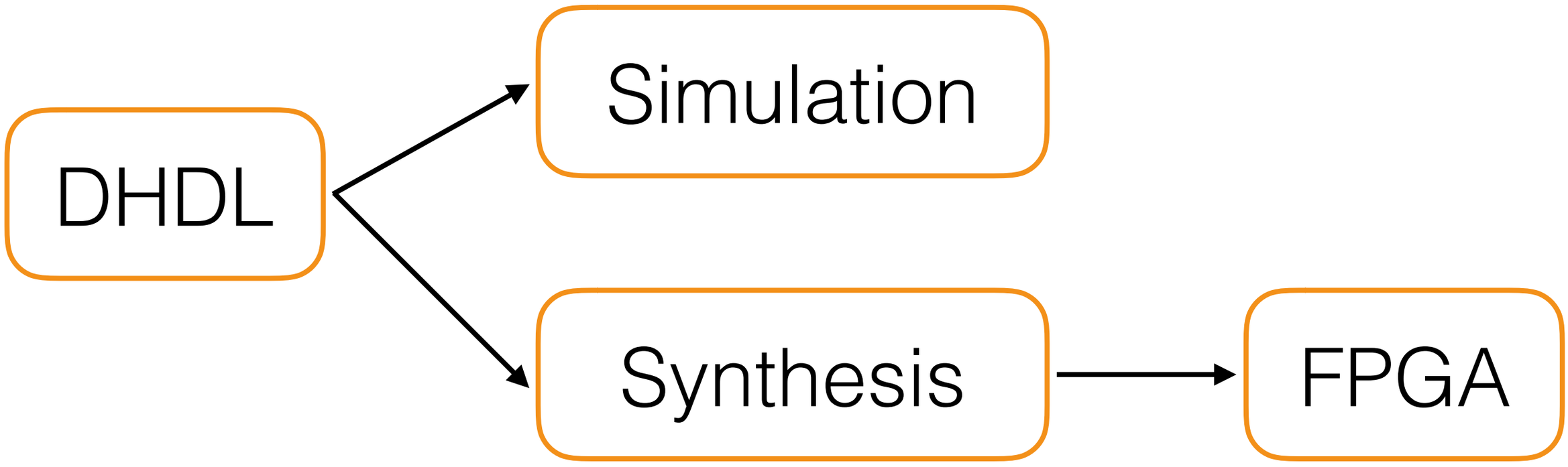
# Why DHDL

- Compiling DHDL



# DHDL

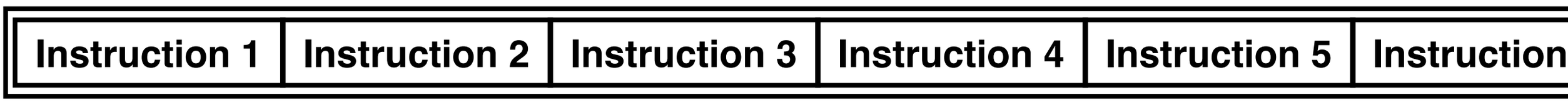
- What we'll see today



# FPGA

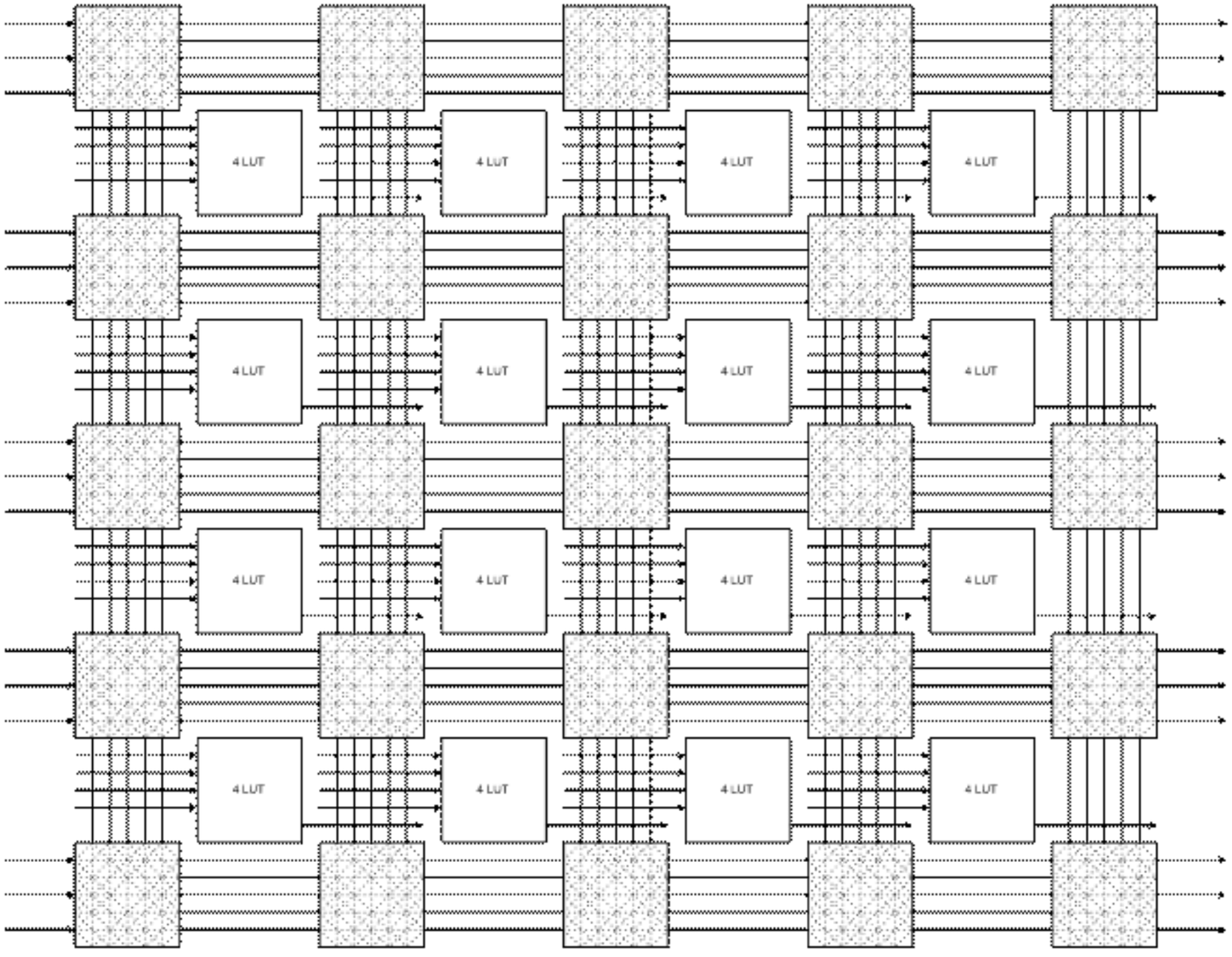
- What's an FPGA?
  - The largest VLIW CPU in the world
  - An interpreter for hardware


(VLIW)

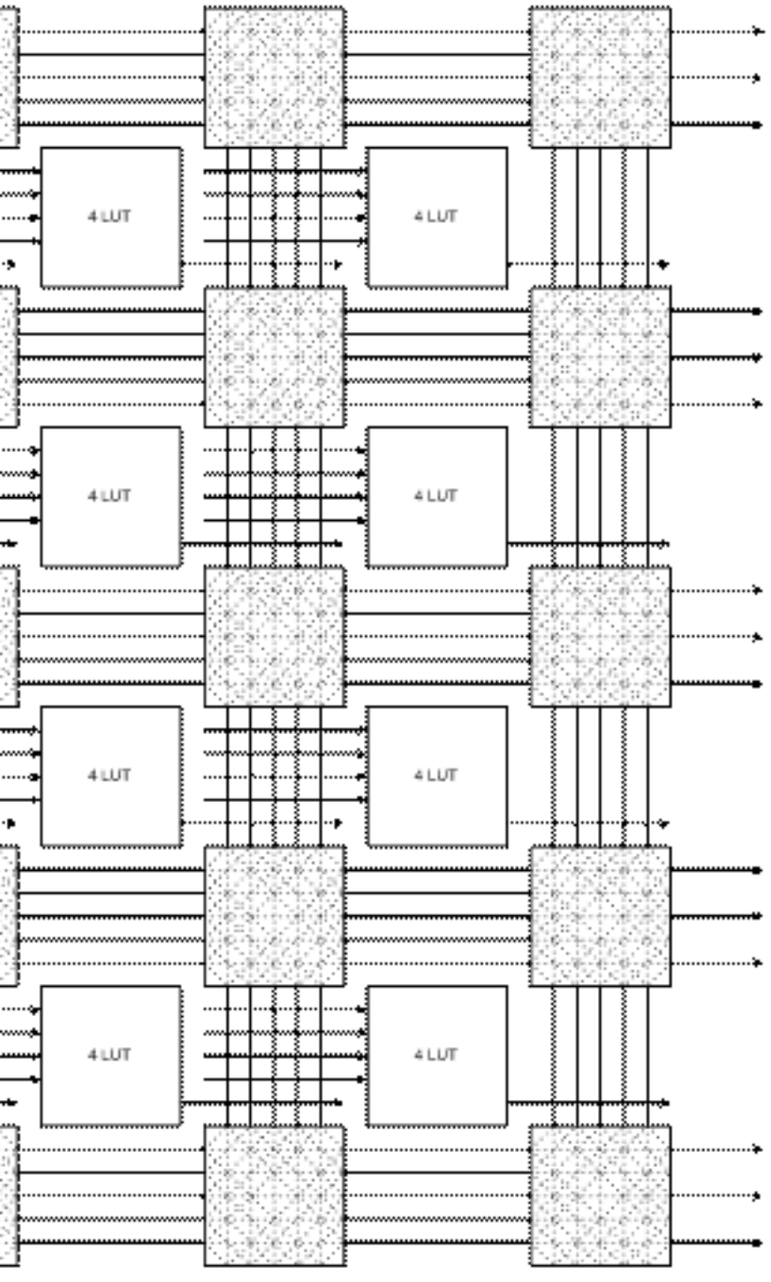


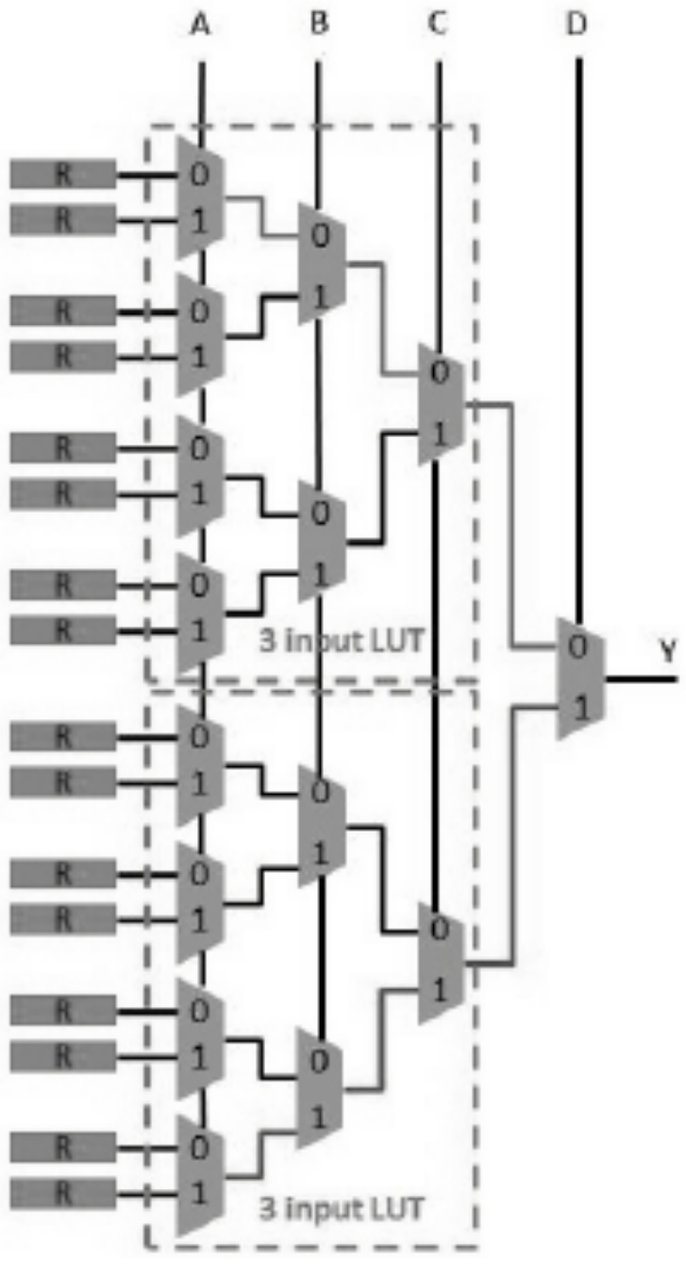

(FPGA)

# FPGA


4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT
4 LUT

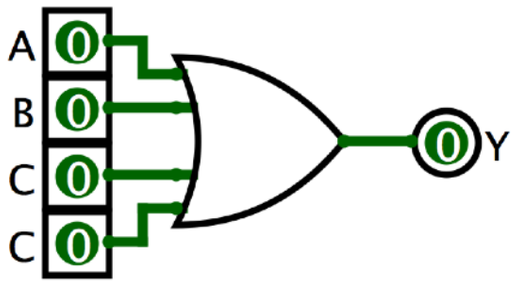
# FPGA



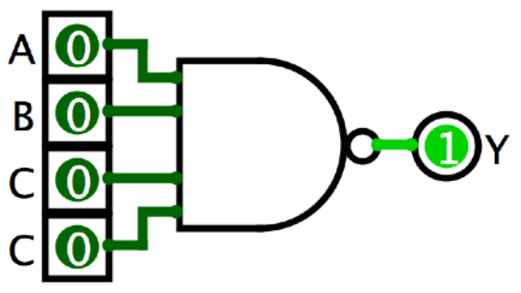

AB

| CD | 0 | 1 | 1 | 1 |
|---|---|---|---|---|
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 1 |



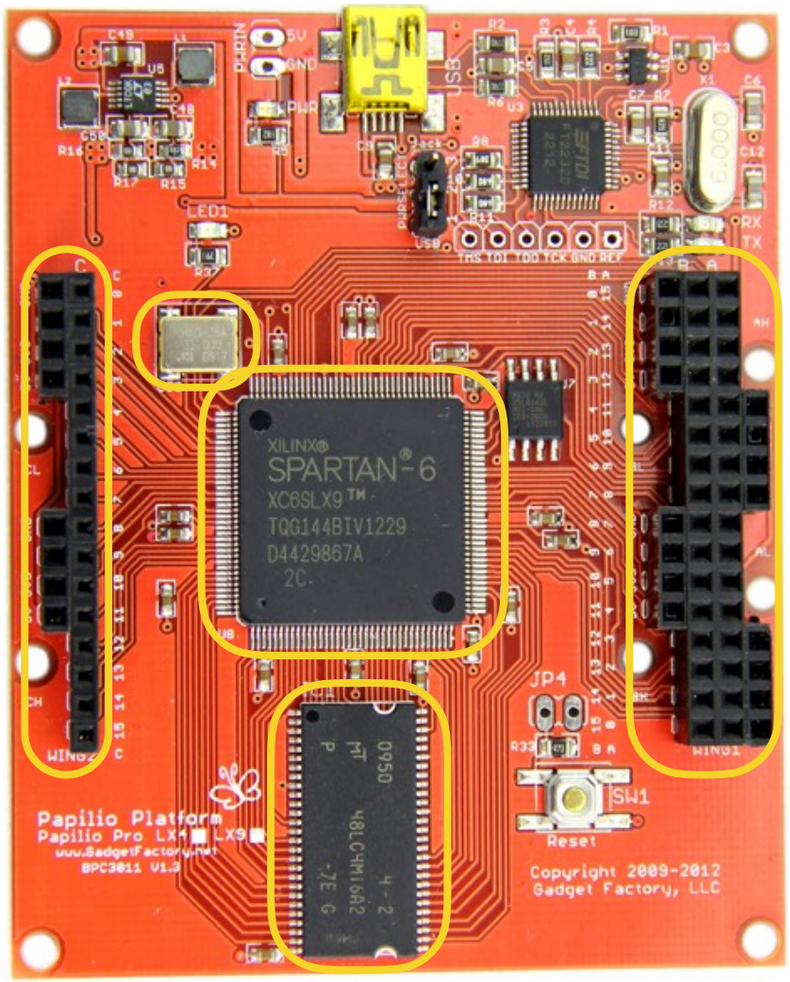

AB

| CD | 1 | 1 | 1 | 1 |
|---|---|---|---|---|
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 1 |
| | 1 | 1 | 1 | 0 |



# FPGA

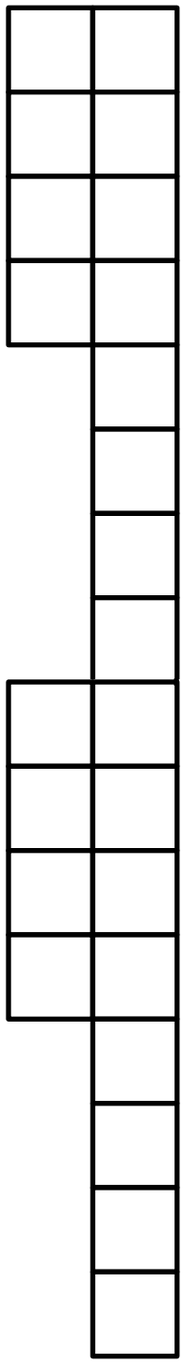
- We'll be using a Papilio Pro in this talk

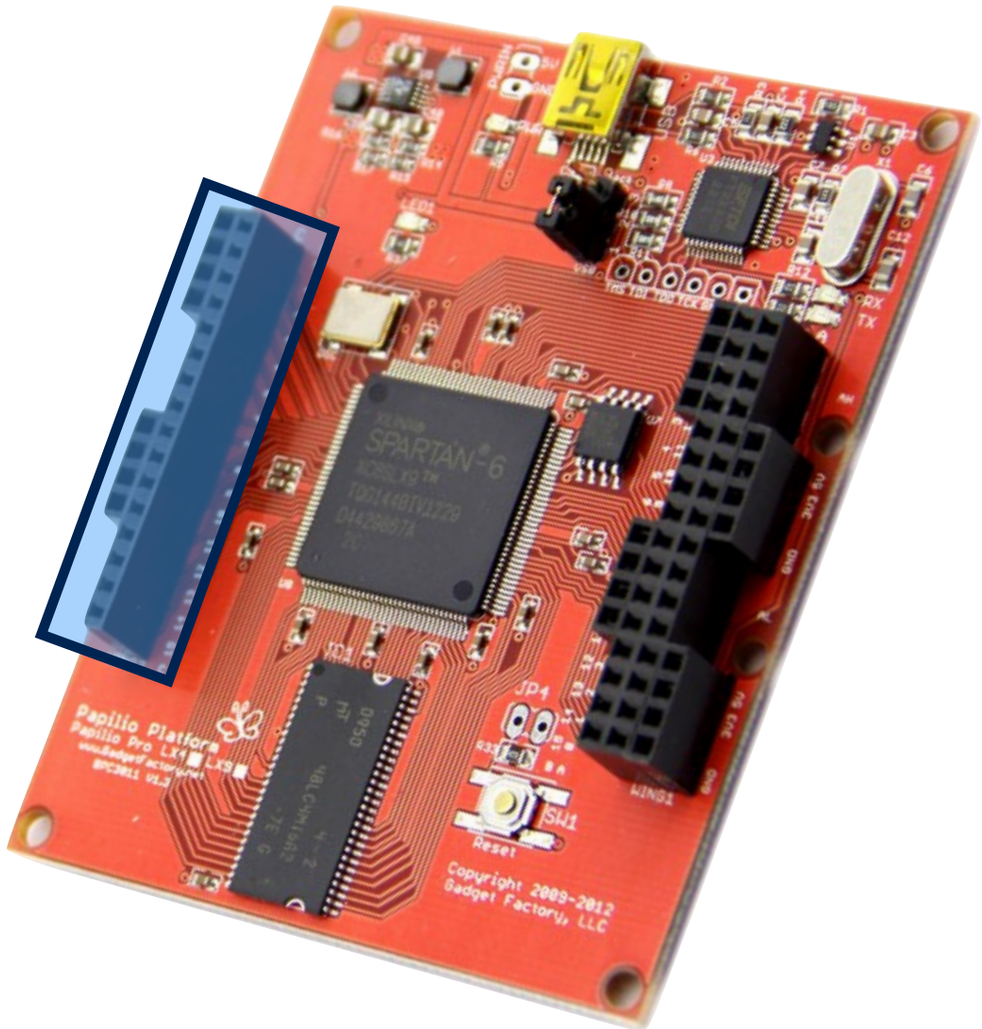



GADGET FACTORY

  - Spartan 6 LX9 FPGA
    - 1,430 slices
    - 32 x 18 Kb block RAM
    - 16 x DSP48A1 (18x18 mul)
  - 8 Megabytes SDRAM
  - 32 MHz crystal oscillator
  - 48 I/O pins
    - Papilio Wing form factor

# Papilio Wings

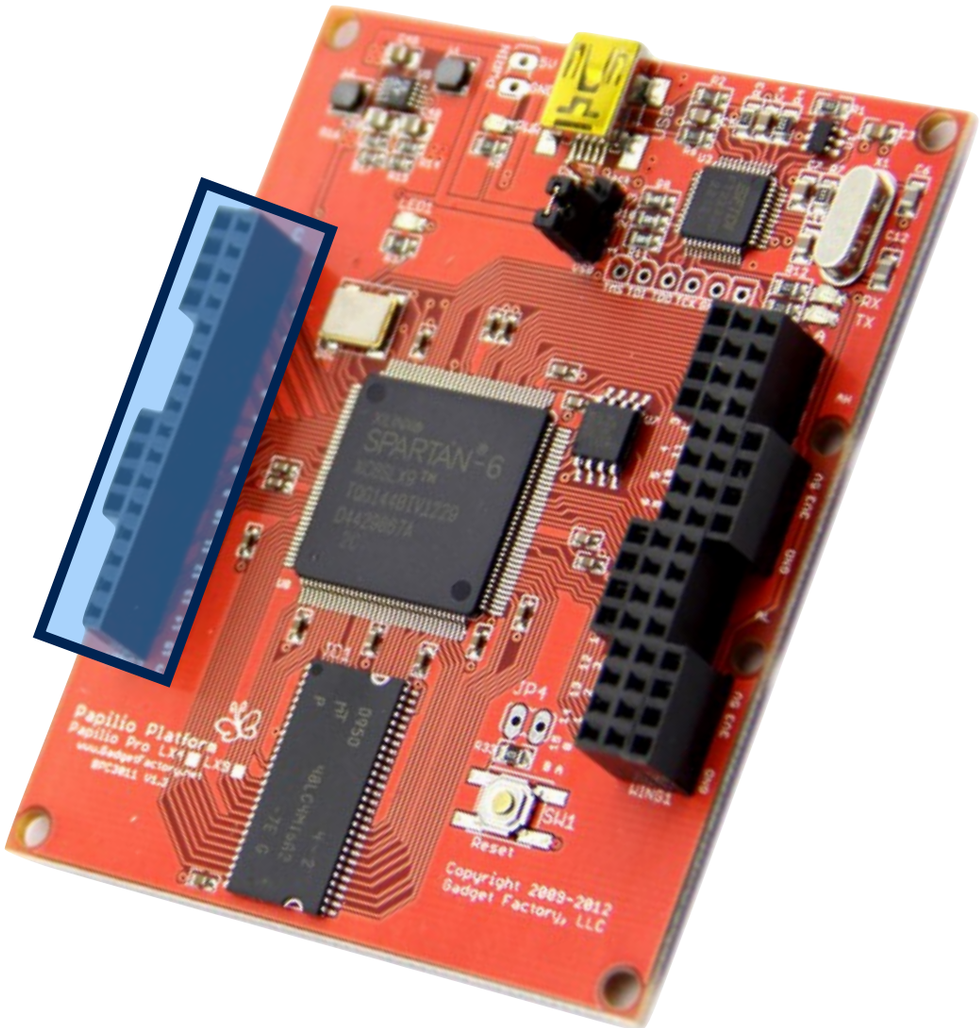




# Papilio Wings

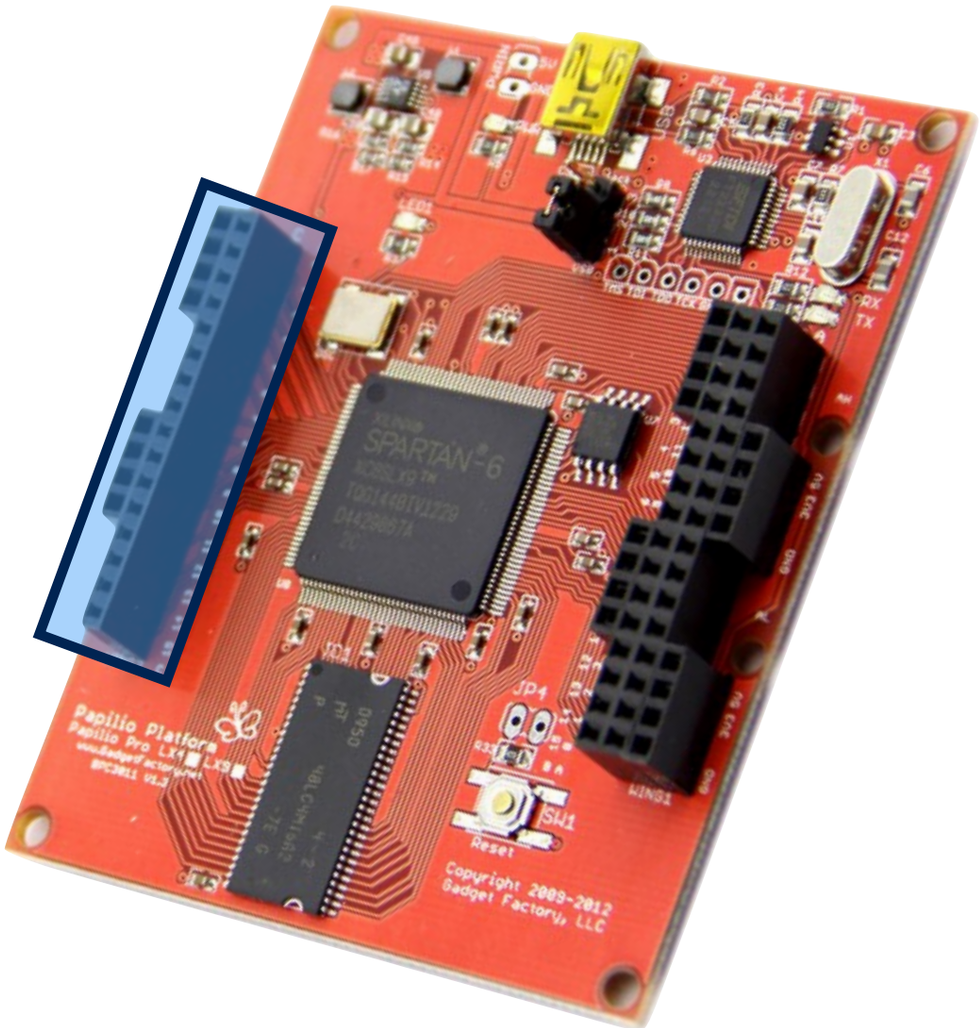
| | |
|---|---|
| GND | 0 |
| | 1 |
| 3V3 | 2 |
| 5V | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |

| | |
|---|---|
| GND | 0 |
| | 1 |
| 3V3 | 2 |
| 5V | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |



# Papilio Wings

| | |
|---|---|
| GND | 0 |
| | 1 |
| 3V3 | 2 |
| 5V | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |
| GND | 8 |
| | 9 |
| 3V3 | 10 |
| 5V | 11 |
| | 12 |
| | 13 |
| | 14 |
| | 15 |

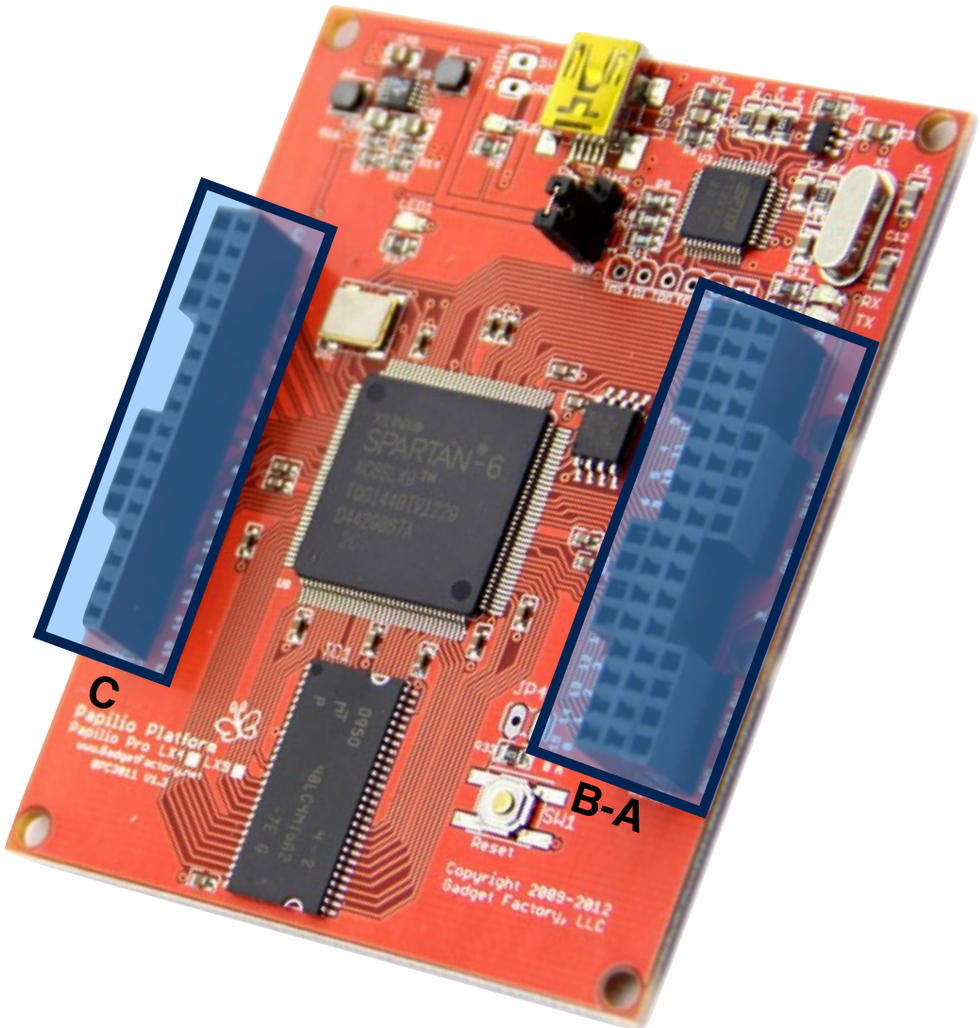


C

| | |
|---|---|
| GND | 0 |
| | 1 |
| 3V3 | 2 |
| 5V | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |
| GND | 8 |
| | 9 |
| 3V3 | 10 |
| 5V | 11 |
| | 12 |
| | 13 |
| | 14 |
| | 15 |

B A

| | | | |
|---|---|---|---|
| GND | 0 | 15 | |
| | 1 | 14 | |
| 3V3 | 2 | 13 | |
| 5V | 3 | 12 | |
| | 4 | 11 | 5V |
| | 5 | 10 | 3V3 |
| | 6 | 9 | |
| | 7 | 8 | GND |
| GND | 8 | 7 | |
| | 9 | 6 | |
| 3V3 | 10 | 5 | |
| 5V | 11 | 4 | |
| | 12 | 3 | 5V |
| | 13 | 2 | 3V3 |
| | 14 | 1 | |
| | 15 | 0 | GND |

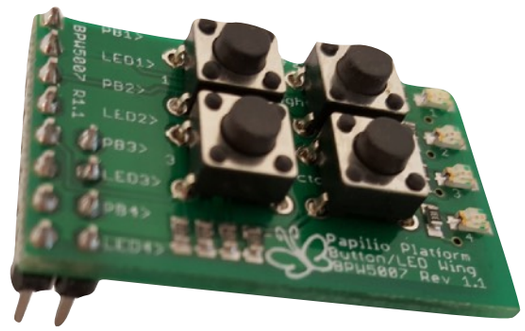


# Papilio Wings

**A-B-C x 2**

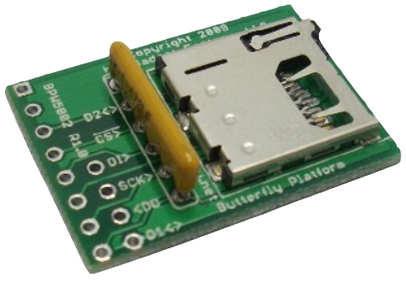
| | |
|---|---|
| GND | 0 |
| | 1 |
| 3V3 | 2 |
| 5V | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |


Audio

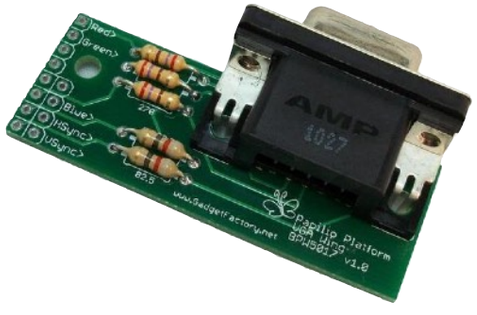



Buttons+LEDs

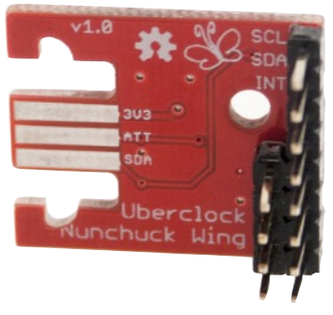

Joystick




SD Card



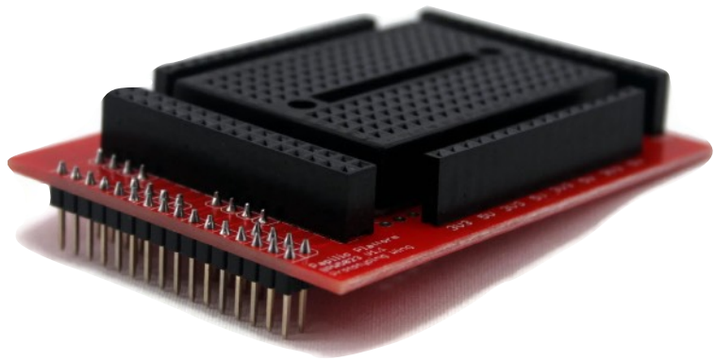

VGA



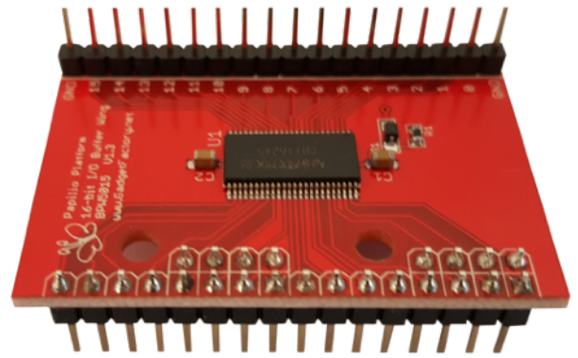

Wii Controller

# Papilio Wings

| C | |
|---|---|
| GND | 0 |
| | 1 |
| 3V3 | 2 |
| 5V | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |
| GND | 8 |
| | 9 |
| 3V3 | 10 |
| 5V | 11 |
| | 12 |
| | 13 |
| | 14 |
| | 15 |



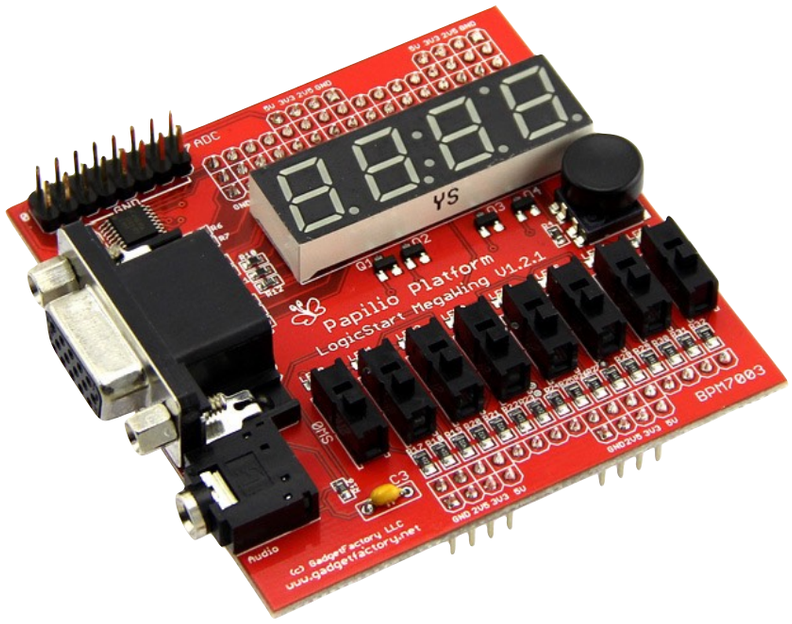
Breadboard



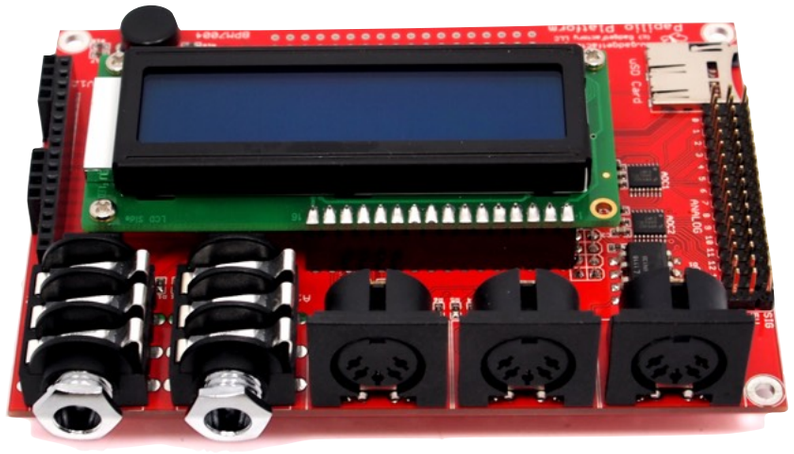
I/O Buffer
(Logic Analyser)

| A | |
|---|---|
| 15 | |
| 14 | |
| 13 | |
| 12 | |
| 11 | 5V |
| 10 | 3V3 |
| 9 | |
| 8 | GND |
| 7 | |
| 6 | |
| 5 | |
| 4 | |
| 3 | 5V |
| 2 | 3V3 |
| 1 | |
| 0 | GND |

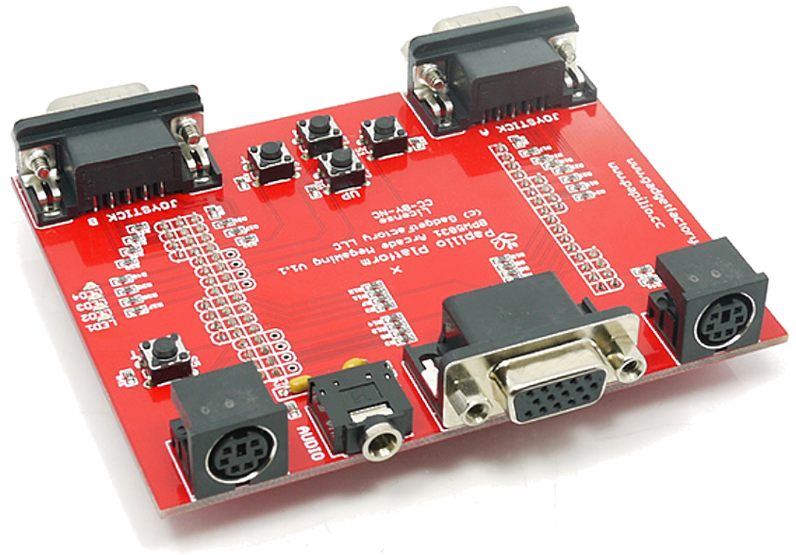
# Papilio MegaWings




LogicStart

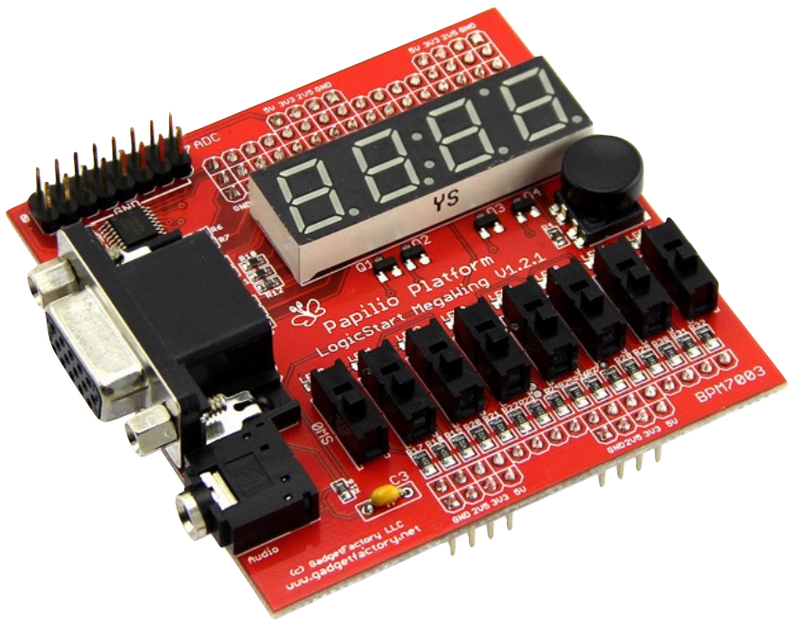


RetroCade

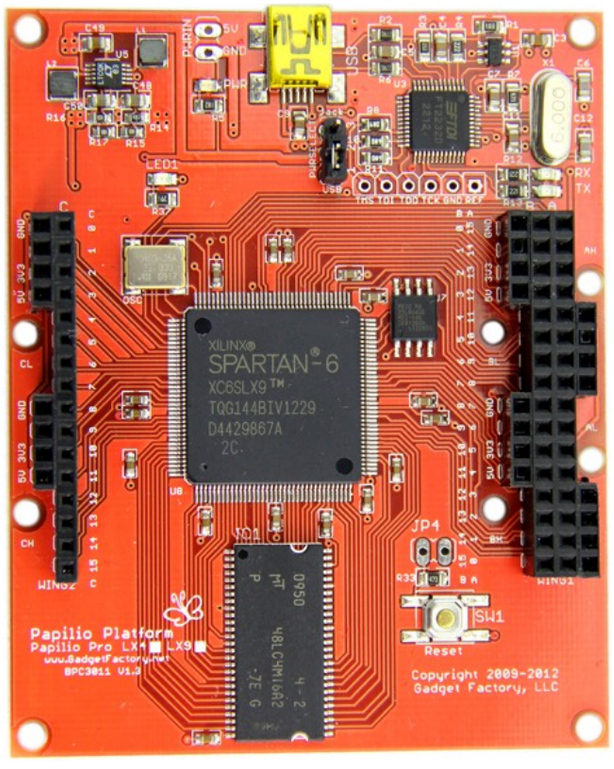



Arcade

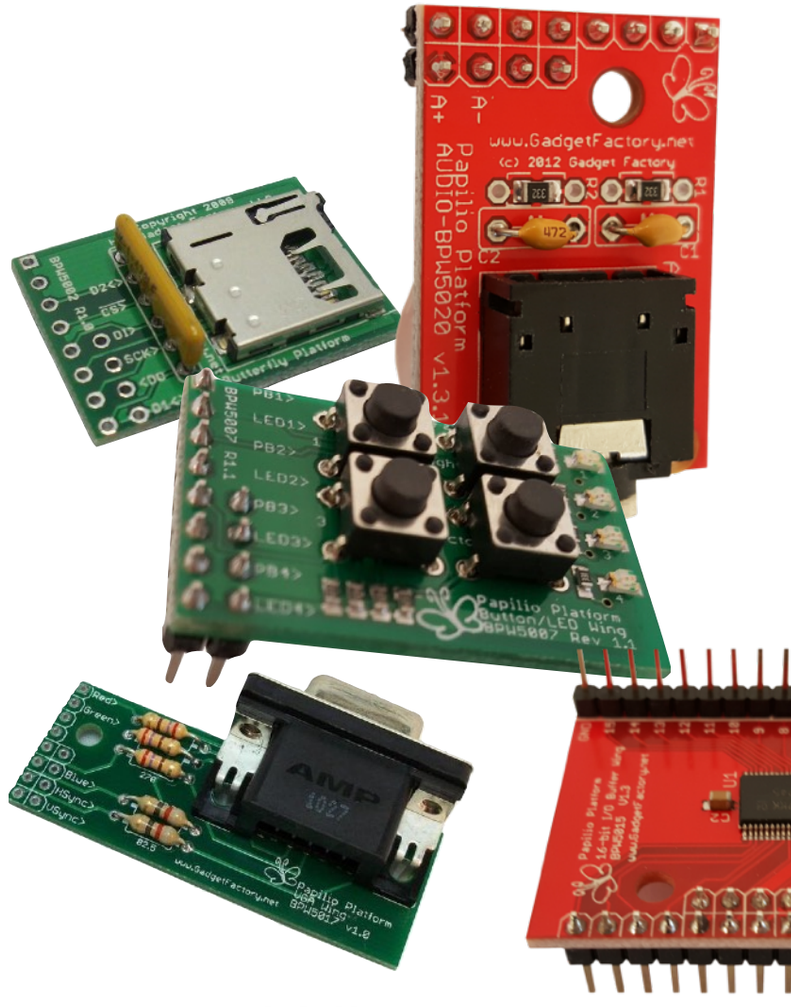
# How to Win an FPGA and Influence Hardware




LogicStart




Papilio Pro




Wings

# Hello World

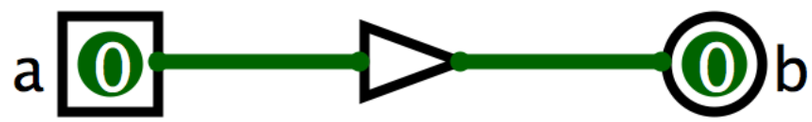
helloworld.dhdl

```
circuit HelloWorld
{
    port in bool a;
    port out bool b;

    b := a;
}
```

# Hello World

helloworld.dhdl

```
circuit HelloWorld
{
    port in bool a;
    port out bool b;

    b := a;
}
```



# Hello World

helloworld.dhdl

```
circuit HelloWorld
{
    port in bool a;
    port out bool b;

    b := a;
}
```

```
unittest
{
    auto hello = peekPokeTester!HelloWorld;

    hello.a = false;
    hello.eval();
    assert(hello.b == false);

    hello.a = true;
    hello.eval();
    assert(hello.b == true);
}
```

# Hello World

helloworld.dhdl

```
circuit HelloWorld
{
    port in bool a;
    port out bool b;

    b := a;
}
```

HelloWorld.d

```
class HelloWorld : Circuit
{
    Bool a;
    Bool b;

    this()
    {
        a = input("a", new Bool);
        b = output("b", new Bool);

        assign(b, a);
    }
}
```

# Hello World

HelloWorld.fir

```
circuit HelloWorld :
    module HelloWorld :
        input reset : UInt<1>
        input clock : Clock
        input a : UInt<1>
        output b : UInt<1>

        b is invalid

        b <= a
```

HelloWorld.d

```
class HelloWorld : Circuit
{
    Bool a;
    Bool b;

    this()
    {
        a = input("a", new Bool);
        b = output("b", new Bool);

        assign(b, a);
    }
}
```

# Hello World

HelloWorld.fir

```
circuit HelloWorld :
    module HelloWorld :
        input reset : UInt<1>
        input clock : Clock
        input a : UInt<1>
        output b : UInt<1>

        b is invalid

        b <= a
```

HelloWorld.v

```
module HelloWorld(
  input   reset,
  input   clock,
  input   a,
  output  b
);
  assign b = a;
endmodule
```

# Hello World

VHelloWorld.h

```
VL_MODULE(VHelloWorld) {
  public:
    VL_IN8(reset,0,0);
    VL_IN8(clock,0,0);
    VL_IN8(a,0,0);
    VL_OUT8(b,0,0);
```

(…)

VHelloWorld.cpp

(…)

VHelloWorld (shared library)

(…)

dhdl\testing.d

(…)

HelloWorld.v

```
module HelloWorld(
  input   reset,
  input   clock,
  input   a,
  output  b
);
  assign b = a;
endmodule
```

```
auto r = ctRegex!r"\s*VL_([A-Z]+[0-9]*)\(([a-z]+),([0-9]+),";
```

# Hello World

helloworld.dhdl

```
circuit HelloWorld
{
    port in bool a;
    port out bool b;

    b := a;
}
```

```
unittest
{
    auto hello = peekPokeTester!HelloWorld;

    hello.a = false;
    hello.eval();
    assert(hello.b == false);

    hello.a = true;
    hello.eval();
    assert(hello.b == true);
}
```

# Hello World

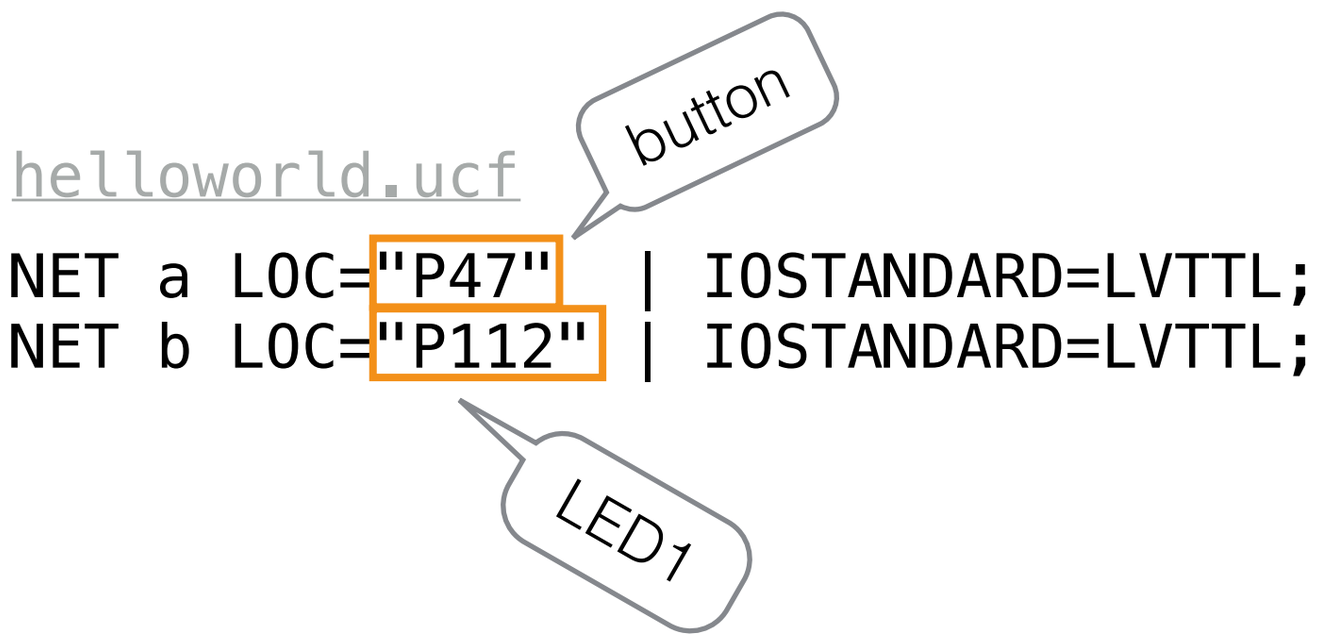
helloworld.dhdl

```
circuit HelloWorld
{
    port in bool a;
    port out bool b;

    b := a;
}
```

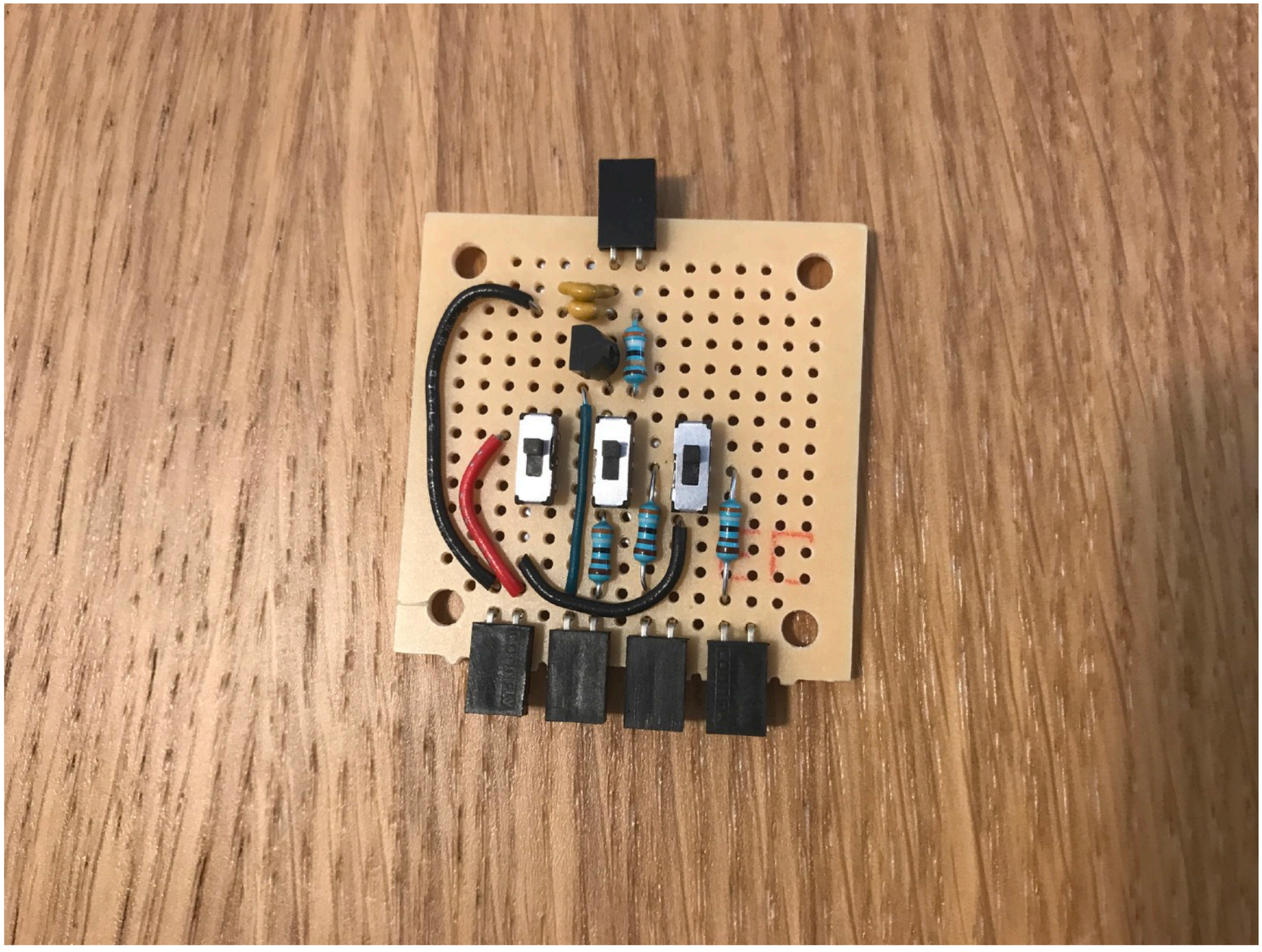
HelloWorld.v

```
module HelloWorld(
  input   reset,
  input   clock,
  input   a,
  output  b
);
  assign b = a;
endmodule
```

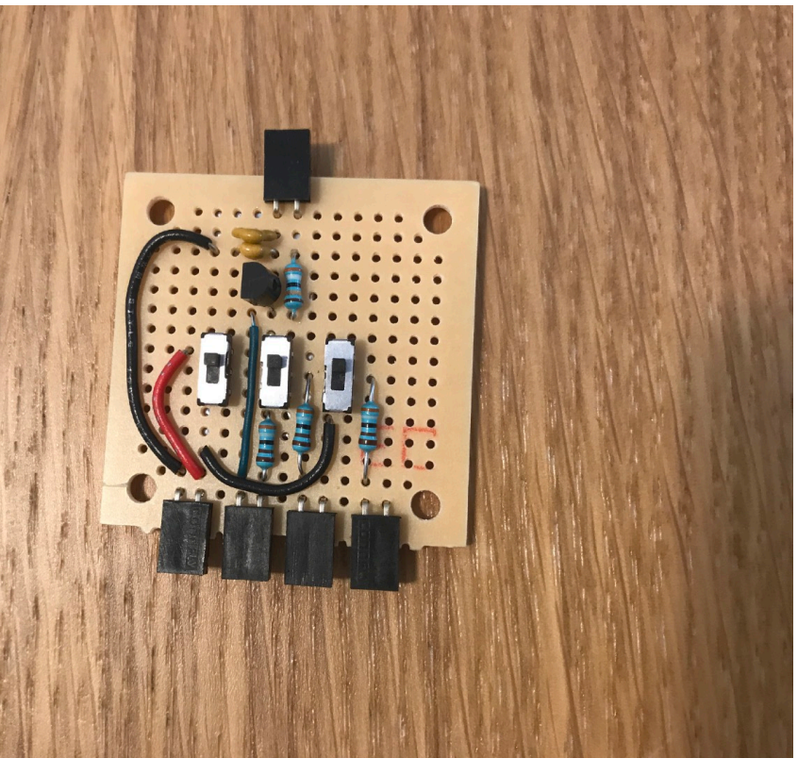
helloworld.ucf

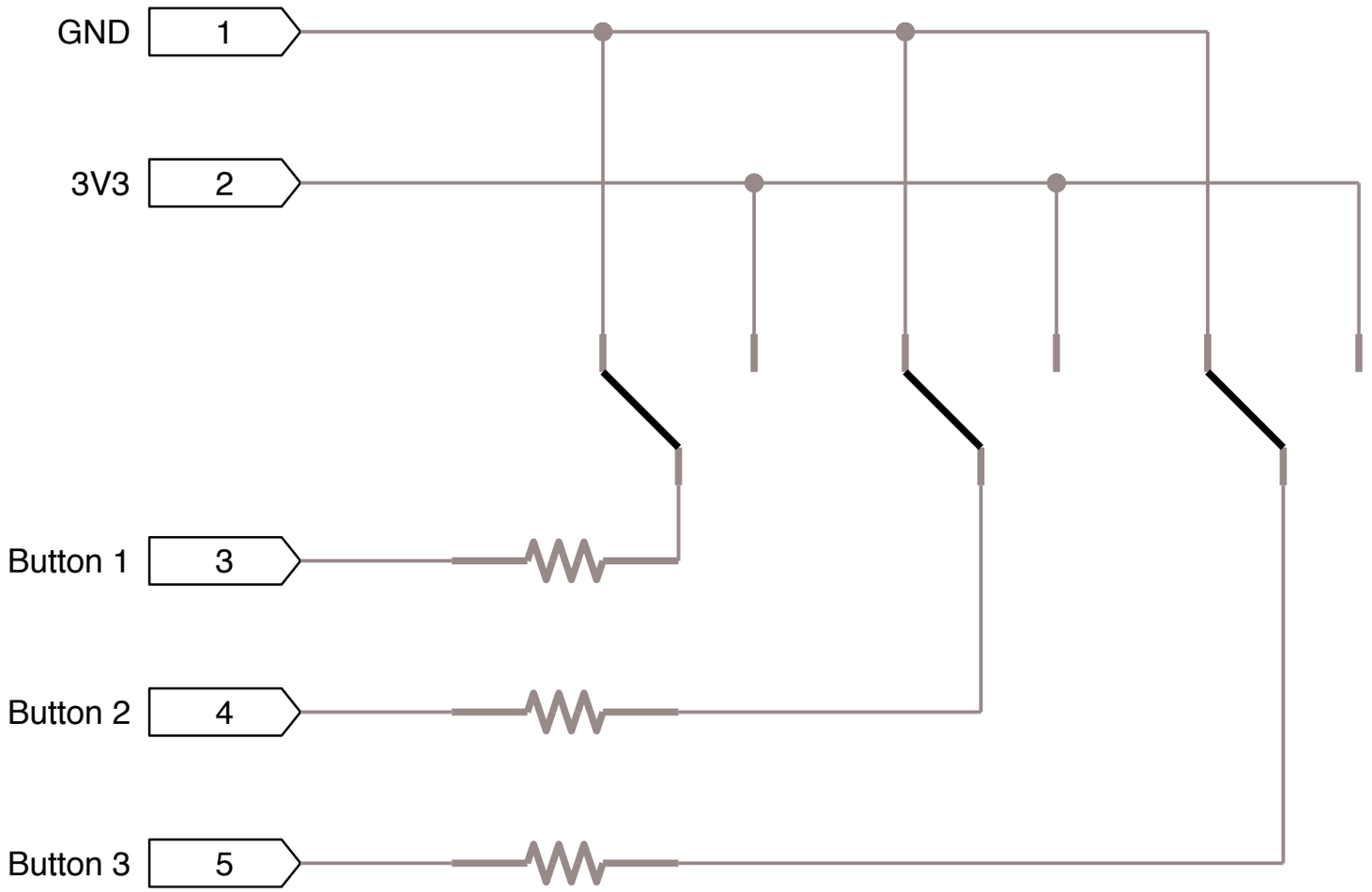


# Hello World

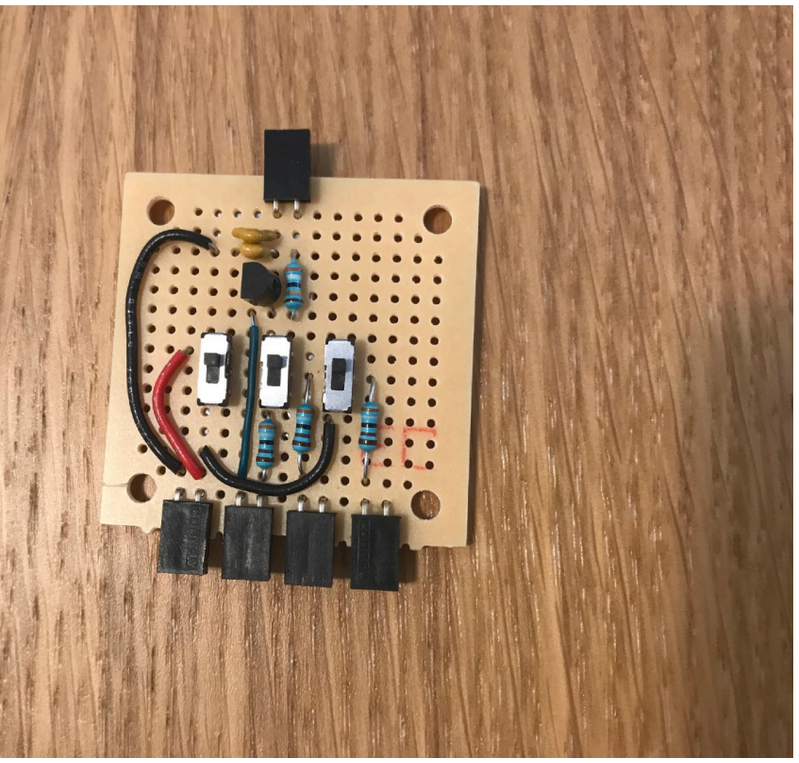


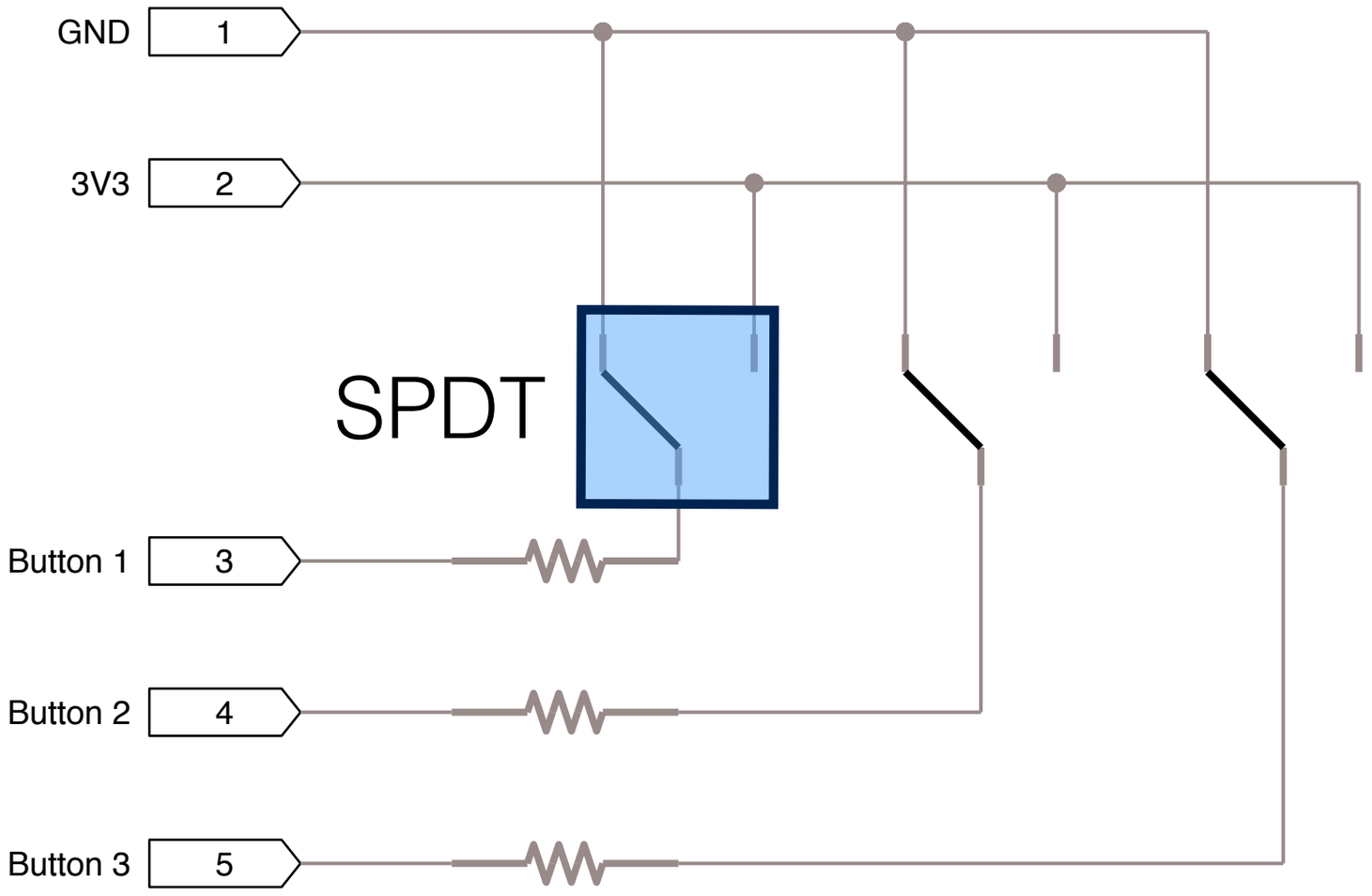
# Hello World



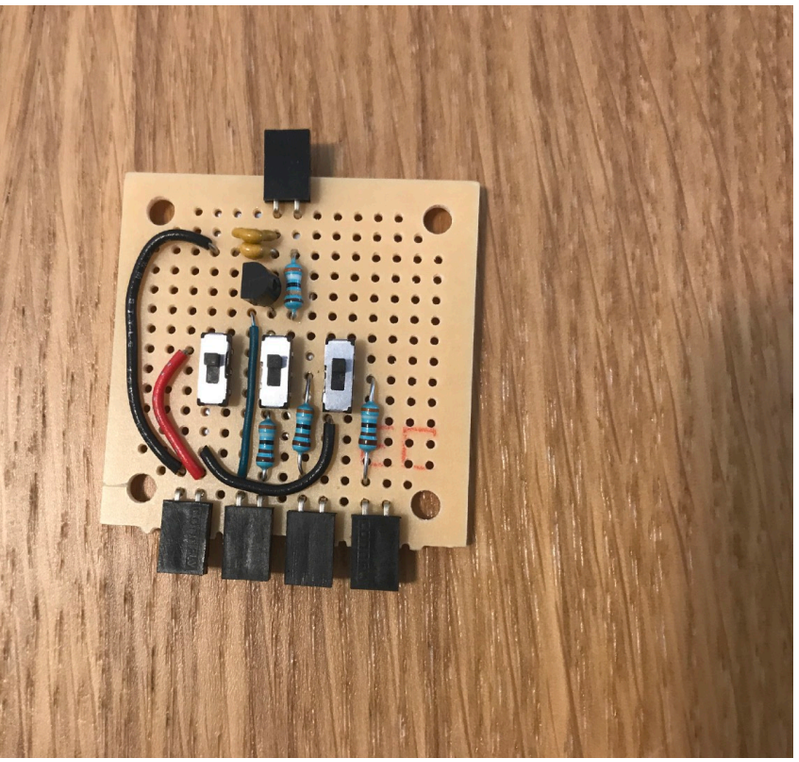


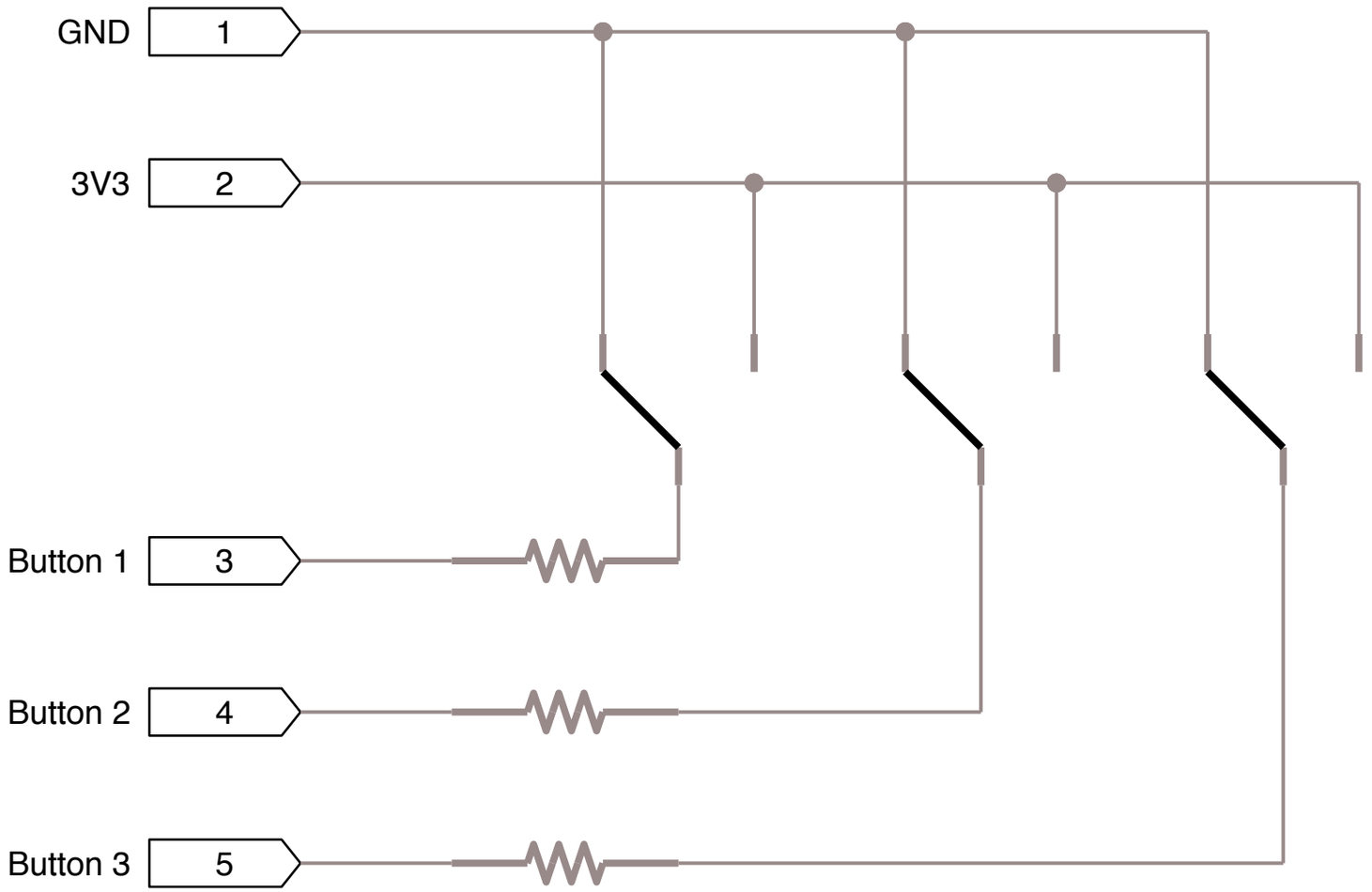
# Hello World



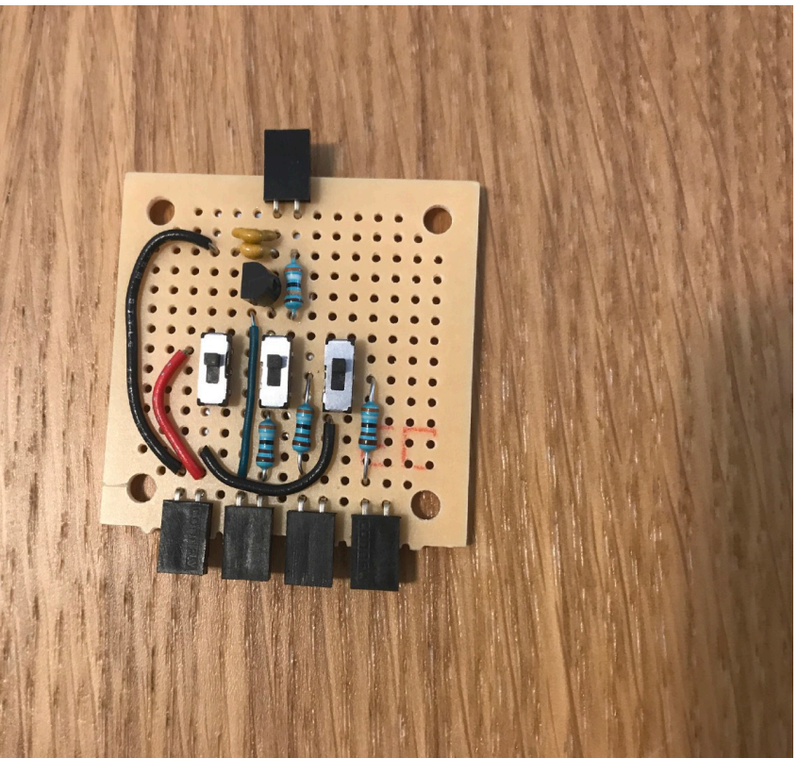


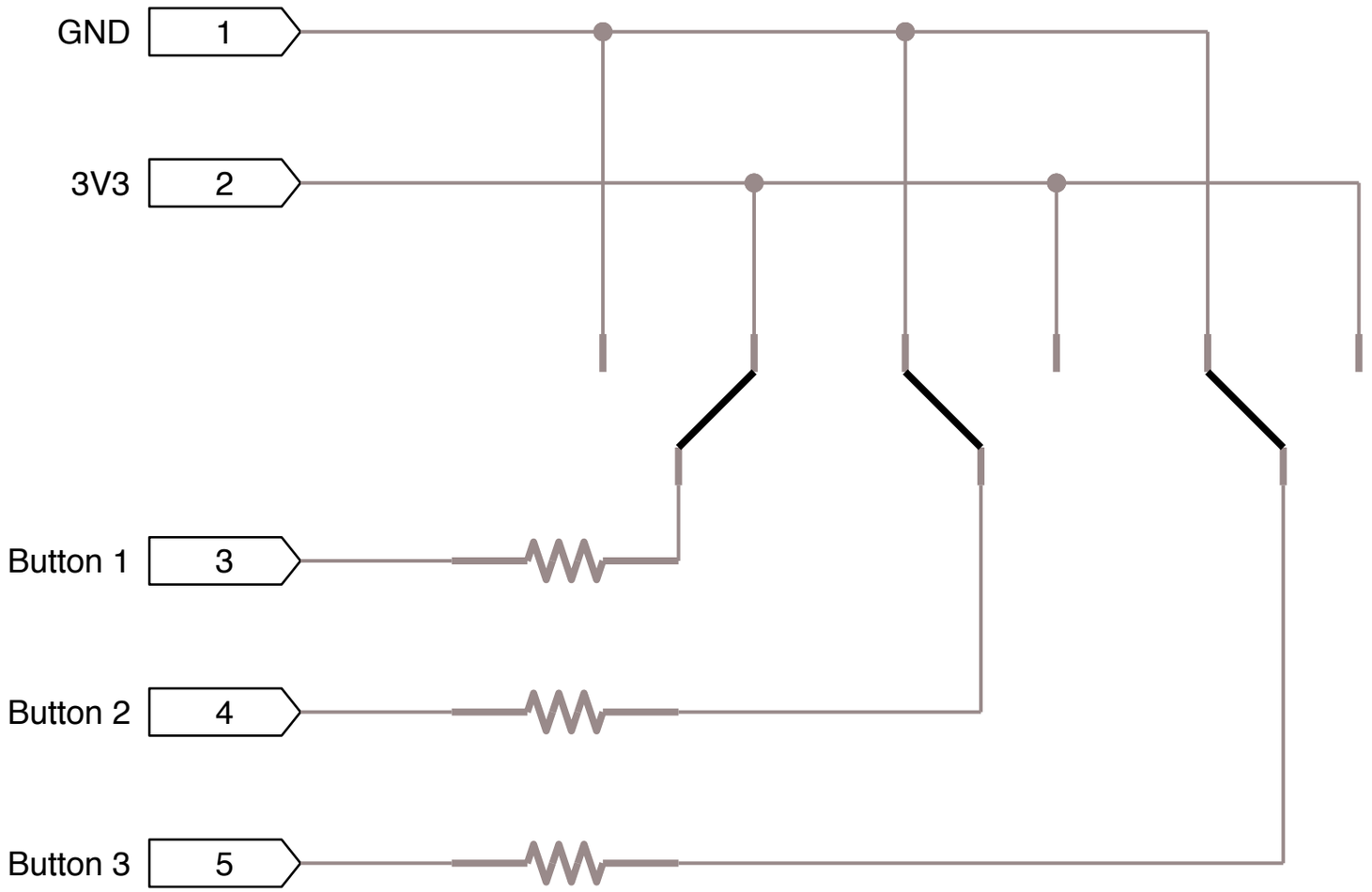
# Hello World



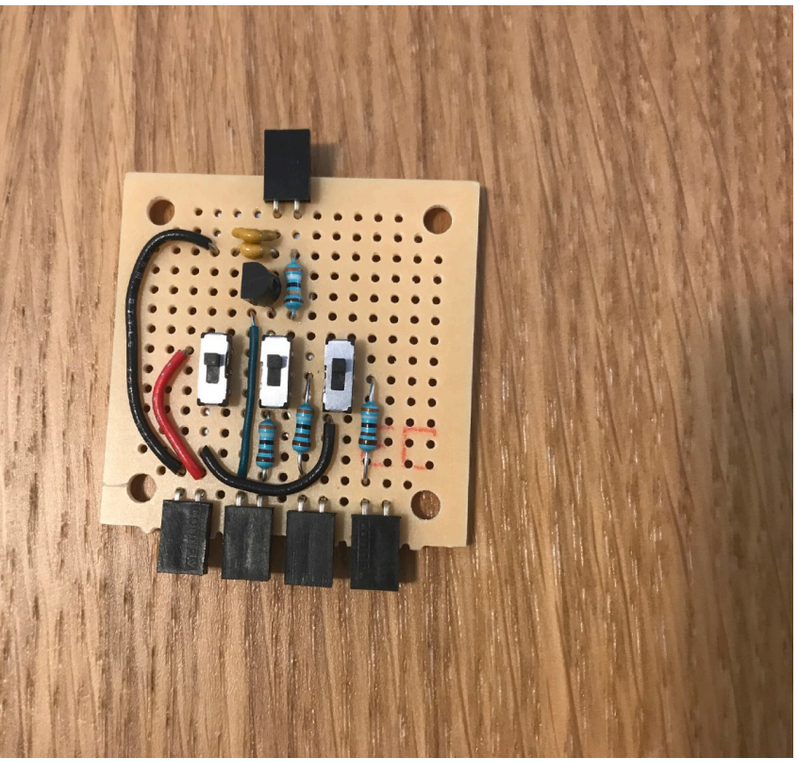


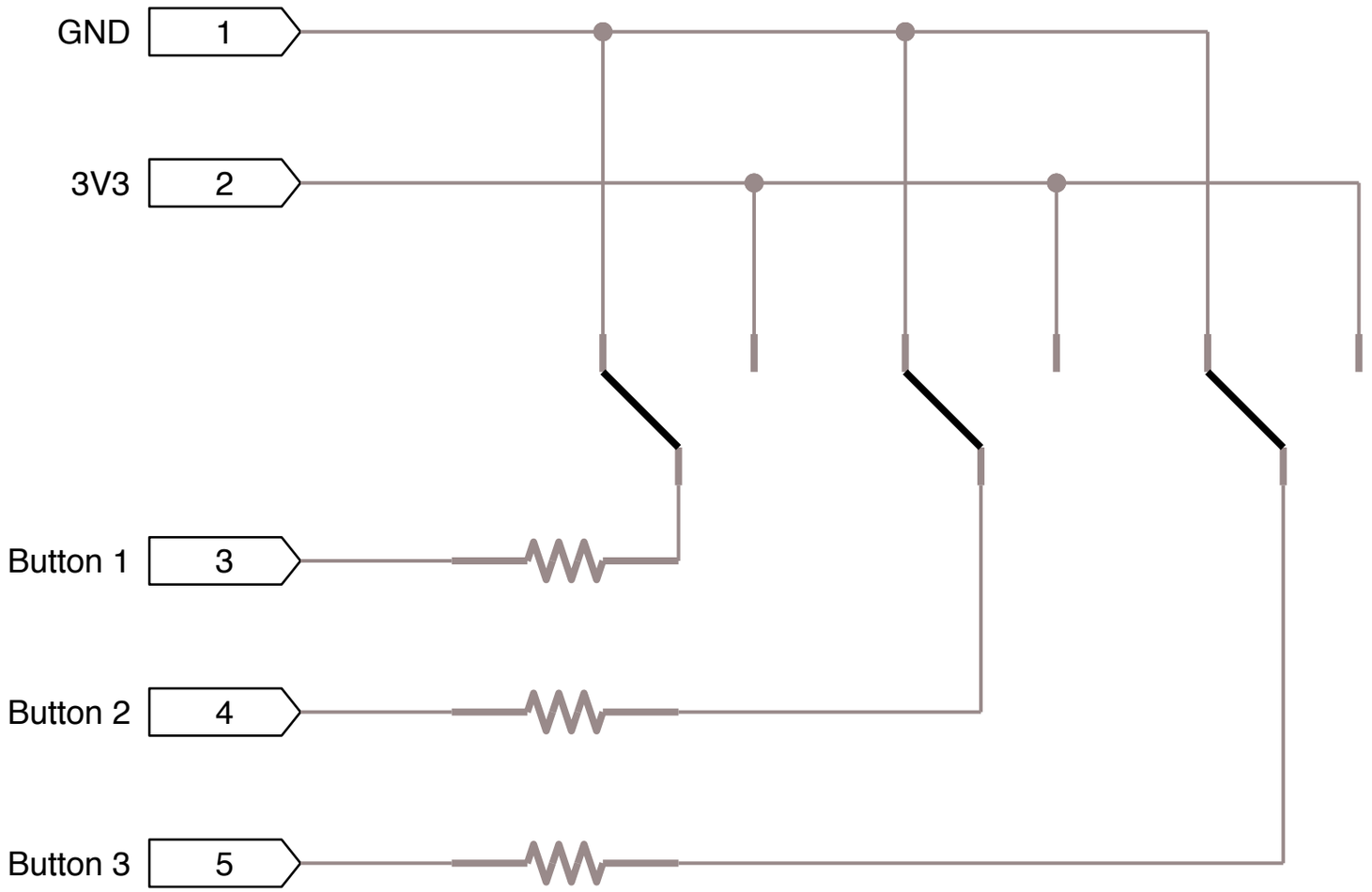
# Hello World



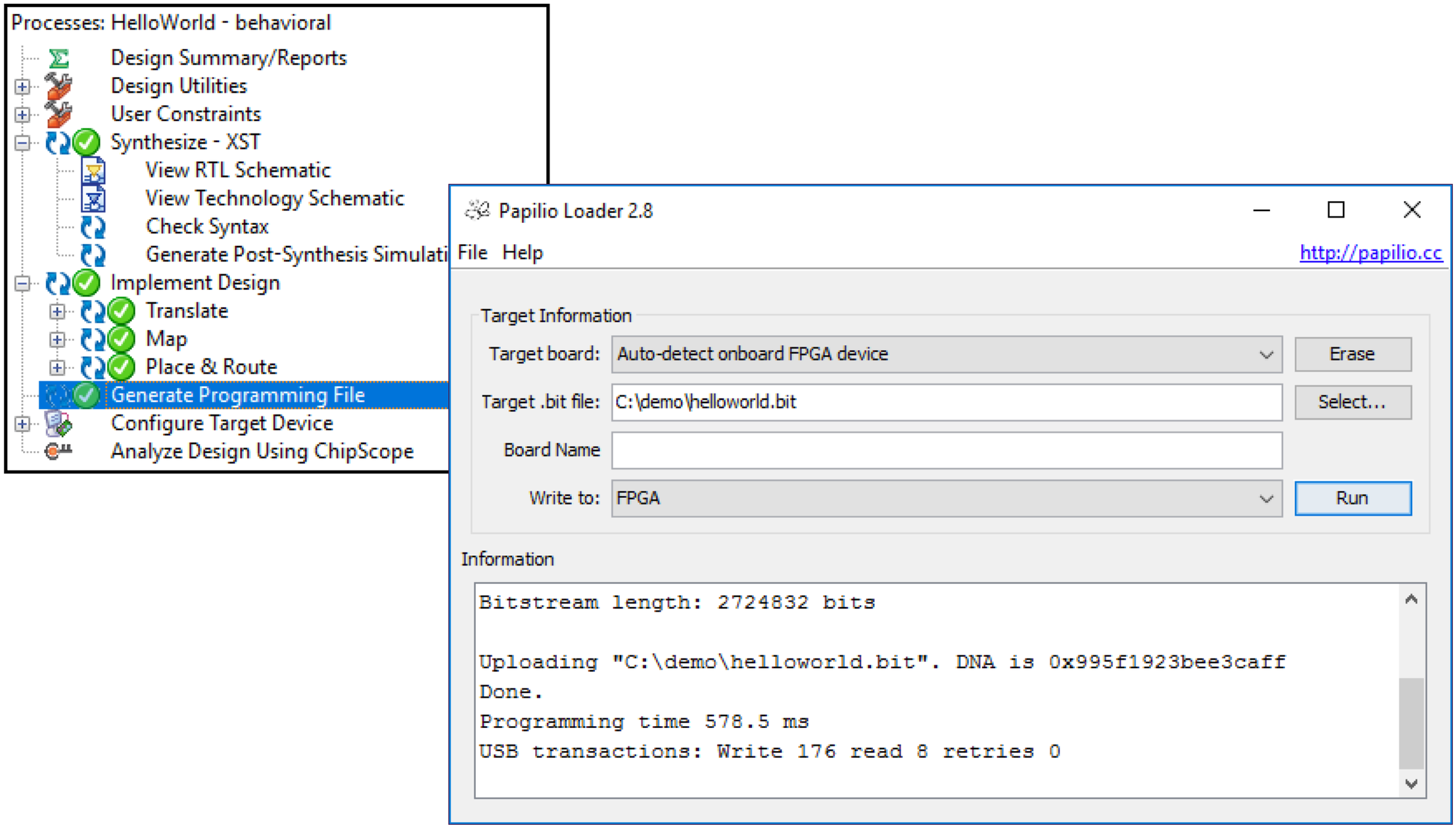


# Hello World

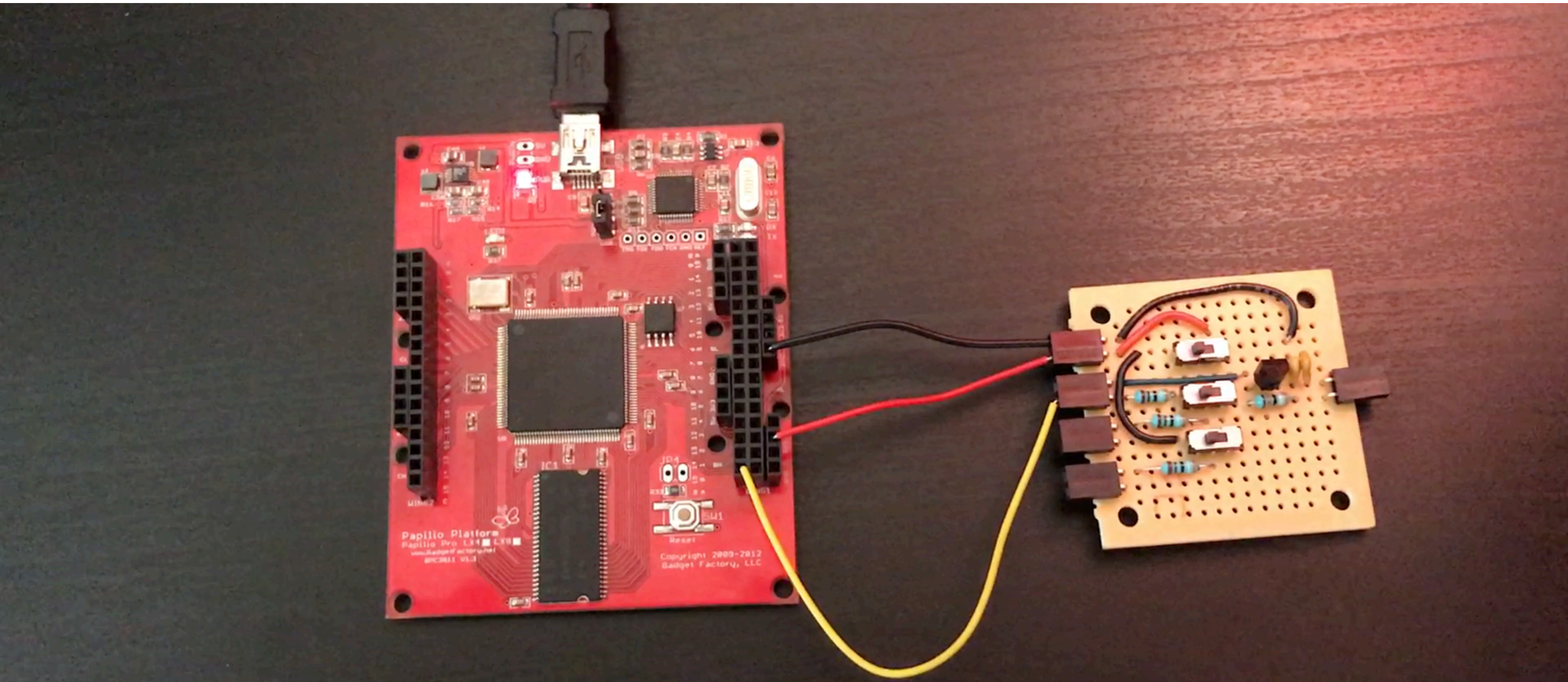




# Hello World


Processes: HelloWorld - behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulati
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Papilio Loader 2.8
File Help
http://papilio.cc
Target Information
Target board:
Auto-detect onboard FPGA device
Erase
Target .bit file:
C:\demo\helloworld.bit
Select...
Board Name
Write to:
FPGA
Run
Information
Bitstream length: 2724832 bits
Uploading "C:\demo\helloworld.bit". DNA is 0x995f1923bee3caff
Done.
Programming time 578.5 ms
USB transactions: Write 176 read 8 retries 0

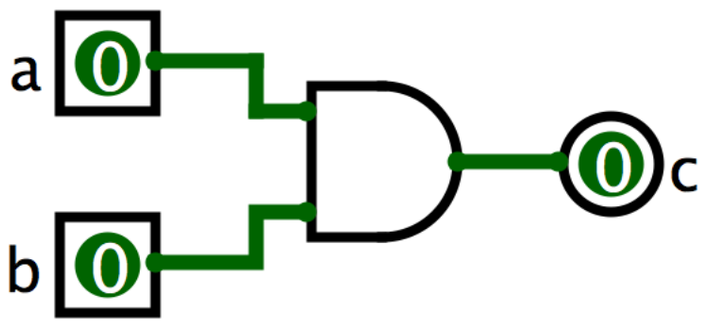
# Hello World



# Operators

operators.dhdl

```
circuit Operators
{
    port in bool a;
    port in bool b;
    port out bool c;

    b := a & b;
}
```



# Operators

operators.dhdl

```
circuit Operators
{
    port in bool a;
    port in bool b;
    port out bool c;

    b := a & b;
}
```

```
unittest
{
    auto op = peekPokeTester!Operators;

    op.a = false;
    op.b = true;
    op.eval();
    assert(op.c == false);

    op.a = true;
    op.eval();
    assert(op.c == true);
}
```

# Operators

operators.dhdl

```
circuit Operators
{
    port in bool a;
    port in bool b;
    port out bool c;

    b := a & b;
}
```

Operators.d

```
class Operators : Circuit
{
    Bool a;
    Bool b;
    Bool c;

    this()
    {
        a = input("a", new Bool);
        b = input("b", new Bool);
        c = output("c", new Bool);

        assign(c, a.and(b));
    }
}
```

# Operators

Operators.fir

```
circuit Operators :
    module Operators :
        input reset : UInt<1>
        input clock : Clock
        output c : UInt<1>
        input a : UInt<1>
        input b : UInt<1>

        c is invalid

        node _T_14 = and(a, b)
        c <= _T_14
```

Operators.d

```
class Operators : Circuit
{
    Bool a;
    Bool b;
    Bool c;


    this()
    {
        a = input("a", new Bool);
        b = input("b", new Bool);
        c = output("c", new Bool);

        assign(c, a.and(b));
    }
}
```

# Operators

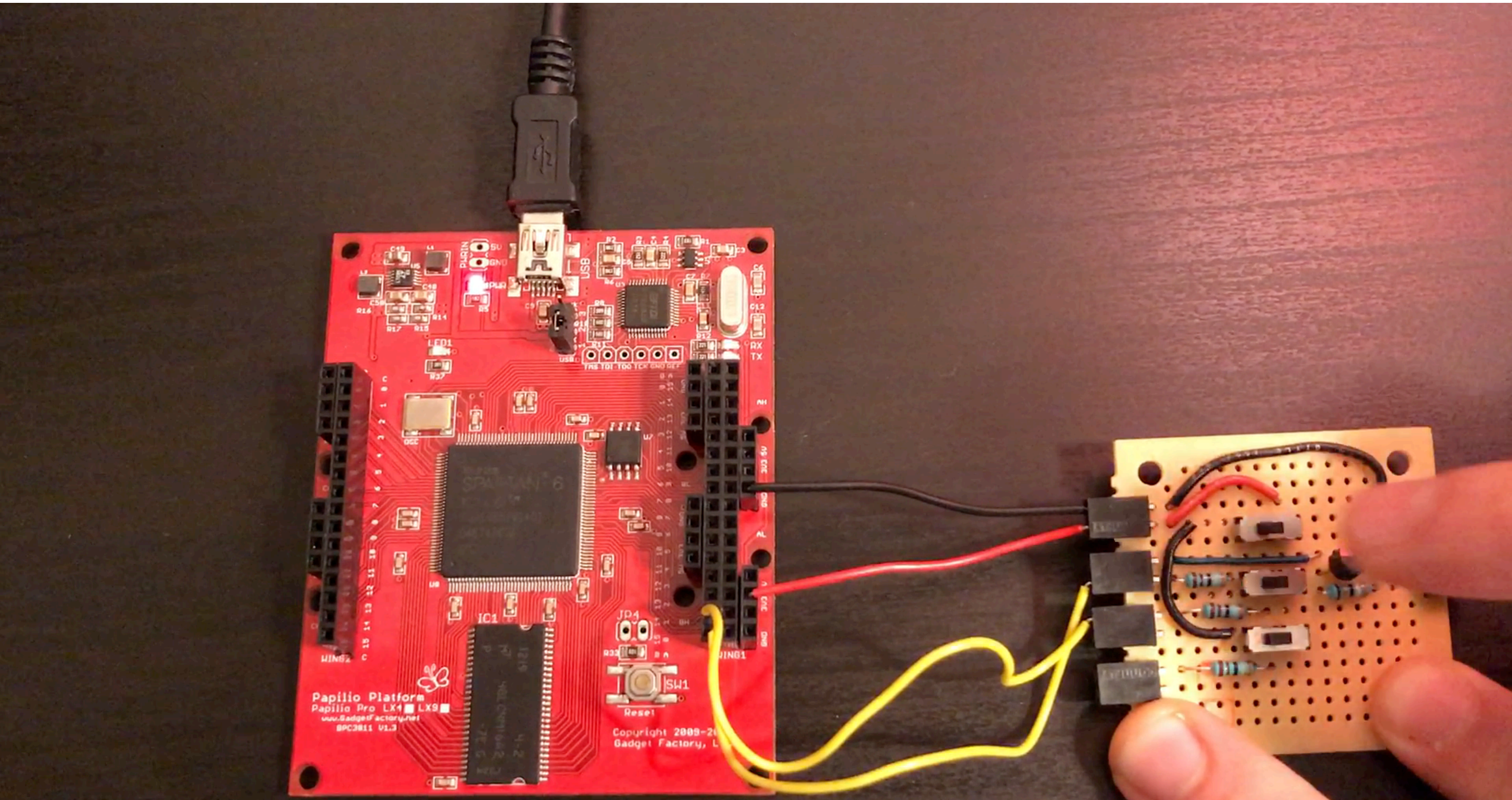
Operators.fir

```
circuit Operators :
    module Operators :
        input reset : UInt<1>
        input clock : Clock
        output c : UInt<1>
        input a : UInt<1>
        input b : UInt<1>

        c is invalid

        node _T_14 = and(a, b)
        c <= _T_14
```

Operators.v

```
module Operators(
  input   reset,
  input   clock,
  output  c,
  input   a,
  input   b
);
  wire  _T_14;
  assign c = _T_14;
  assign _T_14 = a & b;
endmodule
```

# Operators


Papilio Platform
Papilio Pro LX4 LX9
www.GadgetFactory.net
BPC3011 V1.3
Reset
SW1
JP4
IC1
WING2
WING1
USB
RX
TX

# State

blink.dhdl

```
circuit Blink
{
    port out bool a;

    reg bool state;

    state := !state;
    a := state;
}
```

# State

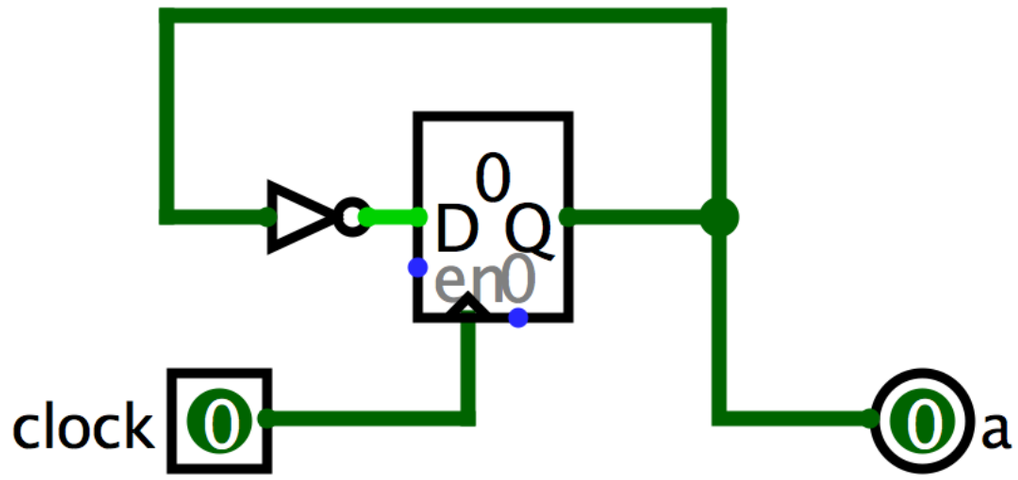
blink.dhdl

```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```

# State

blink.dhdl

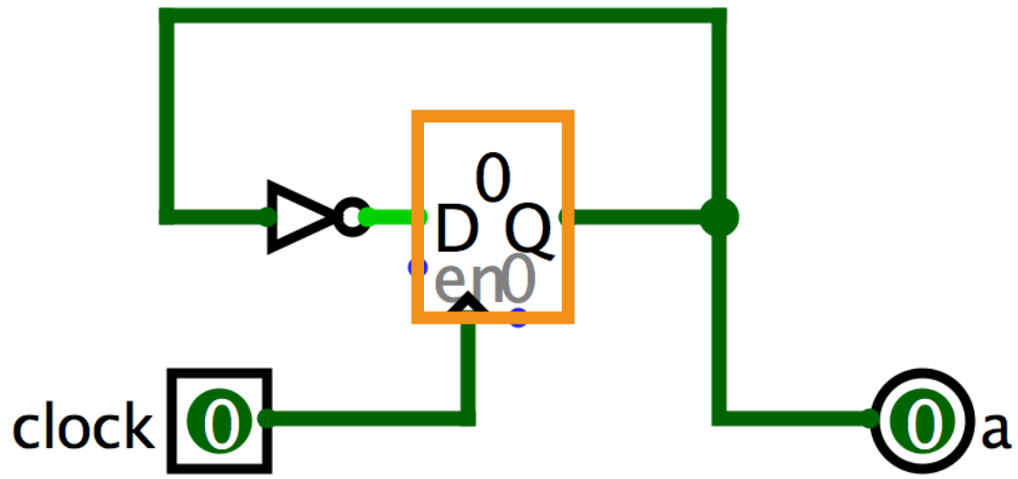
```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```



# State

blink.dhdl

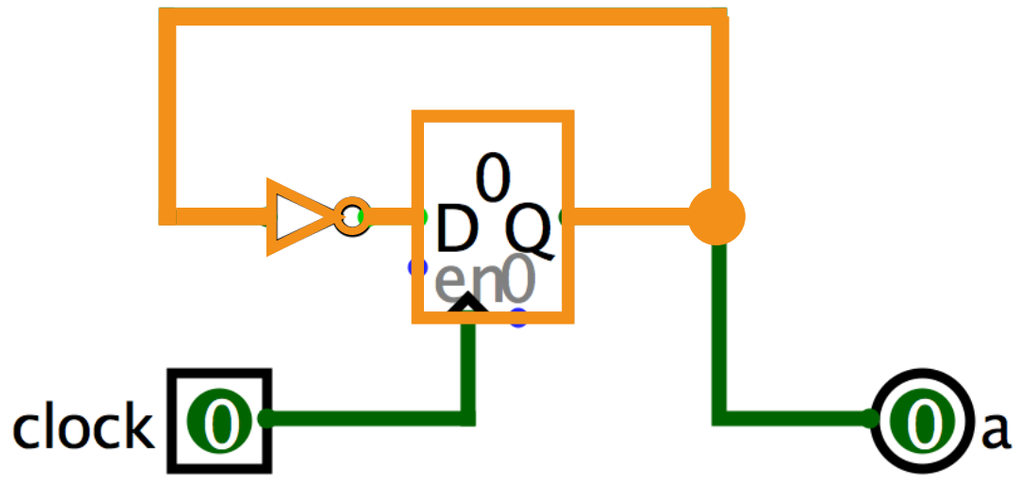
```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```



# State

blink.dhdl

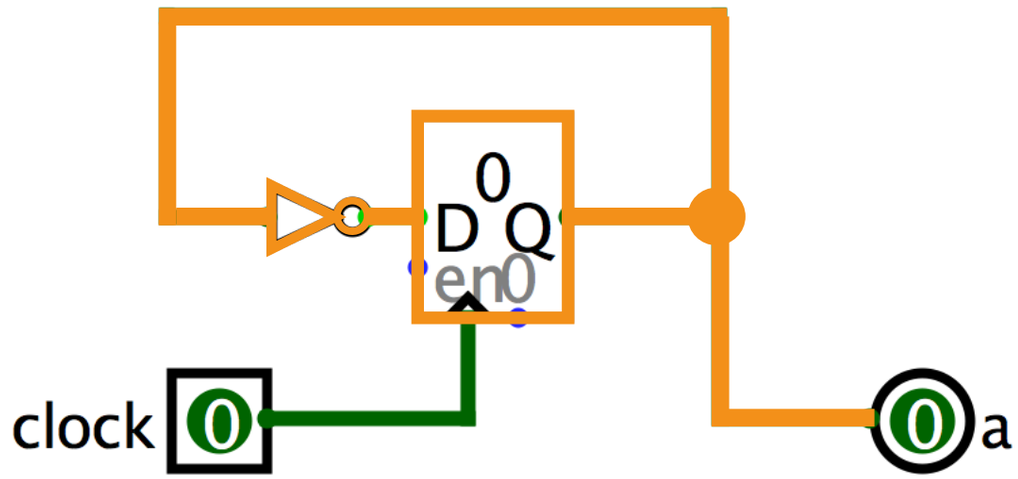
```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```



# State

blink.dhdl

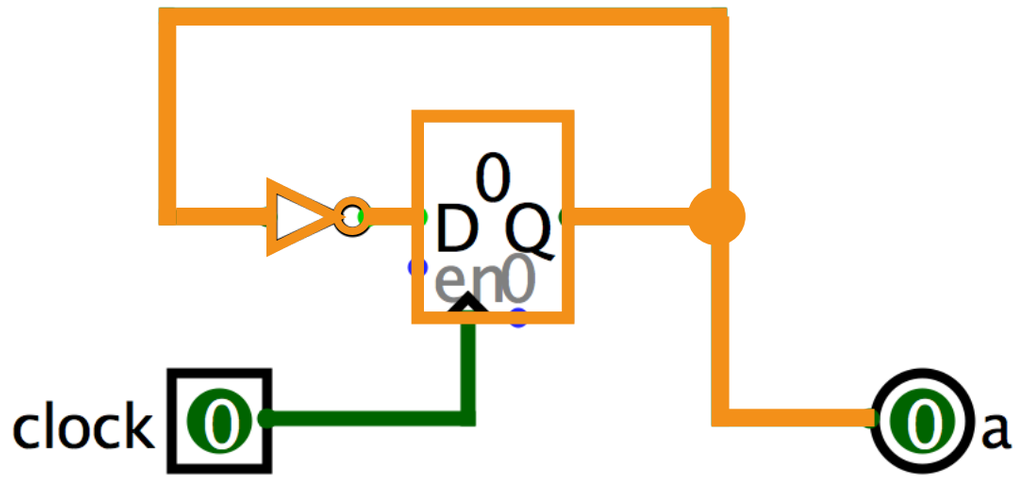
```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```



# State

blink.dhdl

```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```

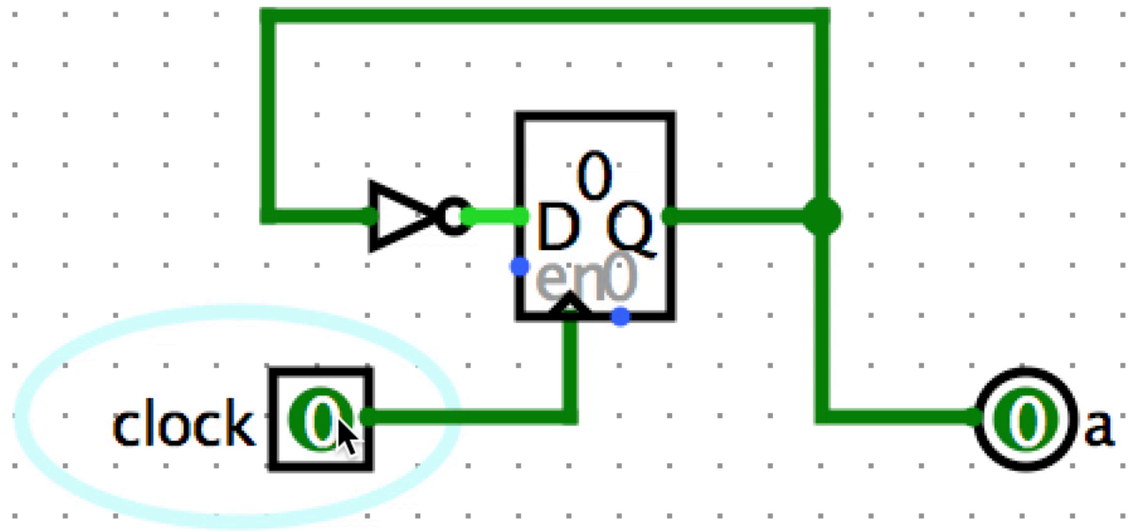



**Question 10**: Will port a blink?

# State

blink.dhdl

```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```



# State

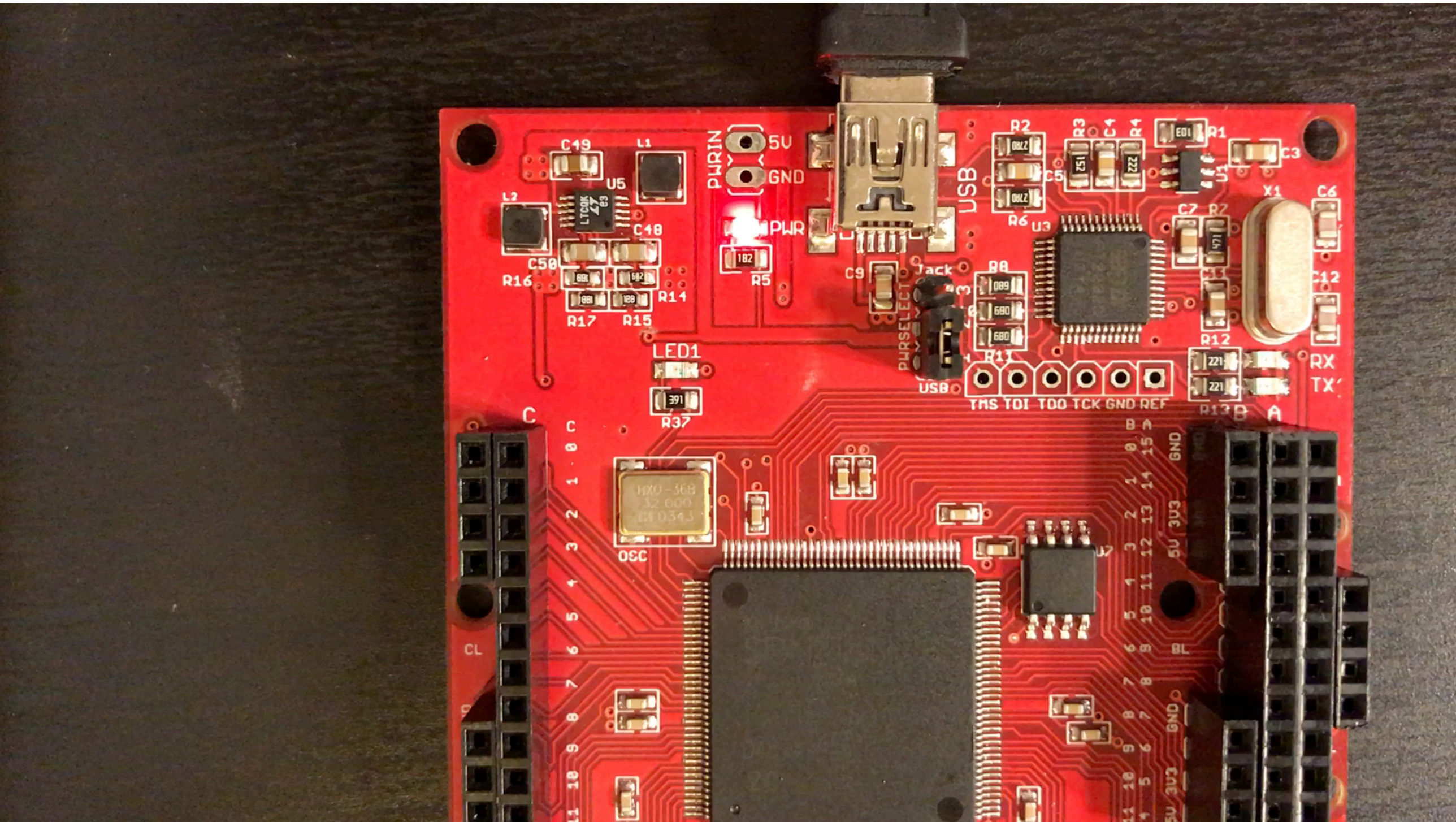
blink.dhdl

```
circuit Blink
{
    port out bool a;

    reg bool state;

    a := state;
    state := !state;
}
```

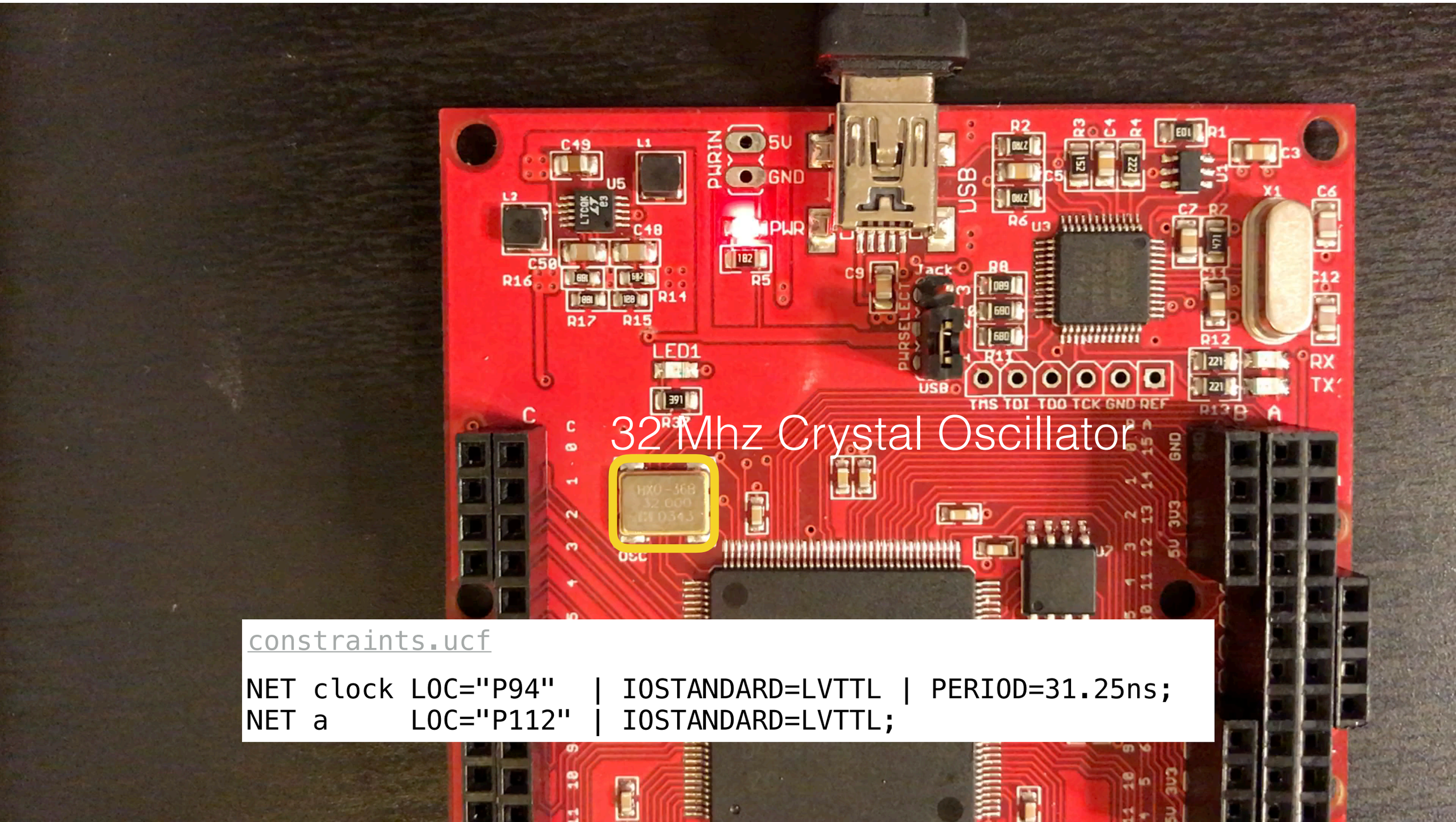
```
unittest
{
    auto blink = peekPokeTester!Blink;

    foreach(i; 0..8)
    {
        assert(blink.a == i % 2);
        blink.step();
    }
}
```

**Question 11**: Will the LED (on port a) blink?

# State


C19
L1
U5
L2
C18
C50
R16
R14
R17
R15
LED1
R37
PWRIN
5V
GND
PWR
R5
C9
Jack
SELECT
USB
R2
R6
U3
R8
R11
R3
C1
R4
R5
C5
R1
U1
C3
X1
C6
C7
R7
C12
R12
R13
RX
TX
USB
TMS TDI TDO TCK GND REF
C
B
A
CL
BL
OSC
HXO-368
32.000
GND
3V3
5V

# State


32 Mhz Crystal Oscillator
constraints.ucf
NET clock LOC="P94" | IOSTANDARD=LVTTL | PERIOD=31.25ns;
NET a LOC="P112" | IOSTANDARD=LVTTL;

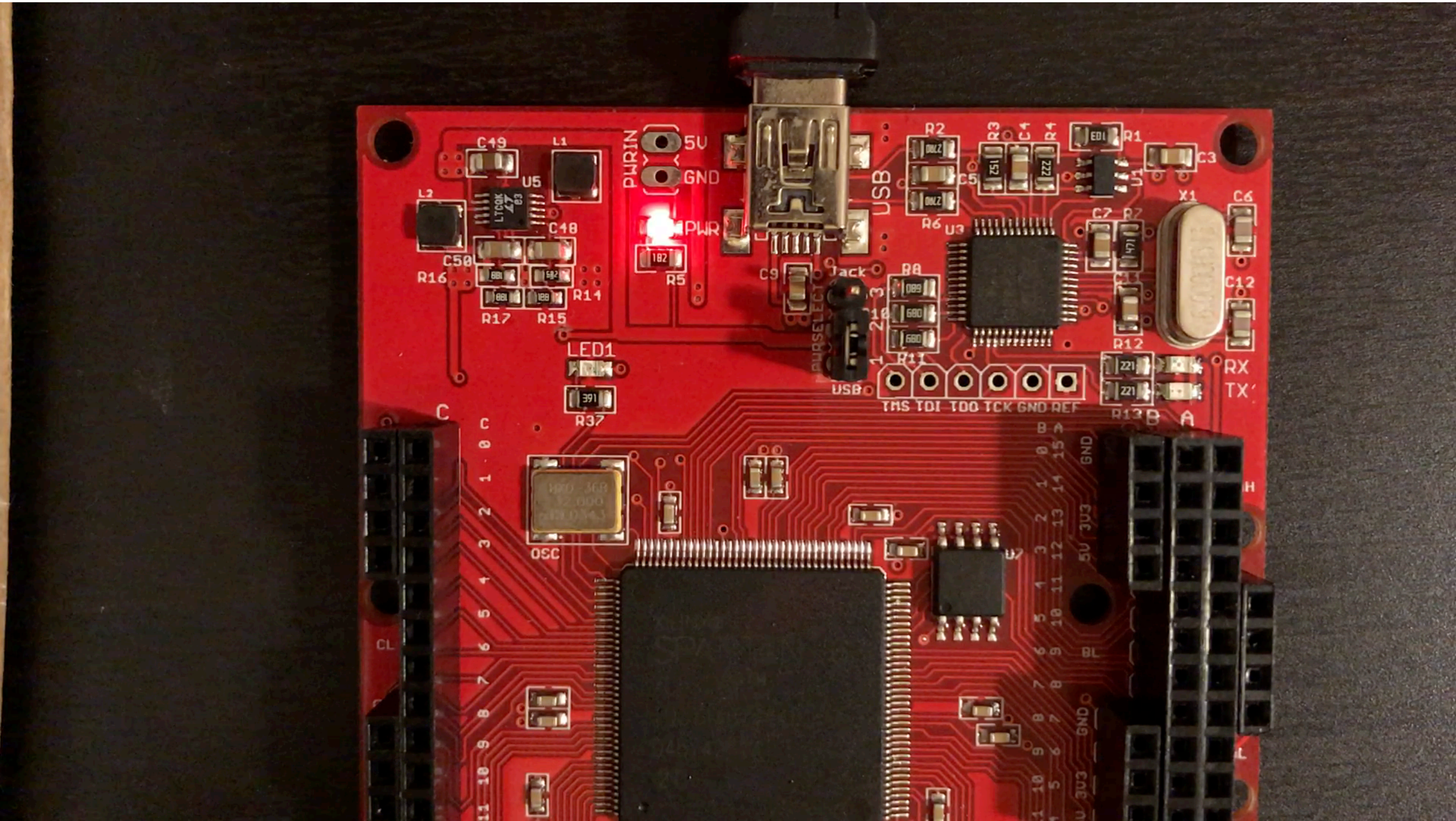
# State

blink2.dhdl

```
enum clockRate = 32_000_000;

circuit Blink2
{
    port out bool a;

    reg bool state;
    reg uint counter = clockRate;

    a := state;

    if(counter != 0)
        counter := counter - 1;
    else
    {
        counter := clockRate
        state := !state;
    }
}
```

**Question 12**: Now, will the LED blink? (at 1 Hz)

# State


C49
L1
U5
L2
C48
C50
R16
R14
R17
R15
PWRIN
5V
GND
PWR
R5
LED1
R37
USB
R2
R3
C1
R4
R1
C5
C3
U1
X1
C6
C7
R7
R6
U3
C9
Jack
R8
R12
R11
C12
RX
TX
TMS TDI TDO TCK GND REF
R13
B
A
OSC
U7
5V 3V3
GND
C
CL
BL
H
L

# State - width inference

blink2.dhdl

```
enum clockRate = 32_000_000;

circuit Blink2
{
    port out bool a;

    reg bool state;
    reg uint counter = clockRate;

    a := state;

    if(counter != 0)
        counter := counter - 1;
    else
    {
        counter := clockRate
        state := !state;
    }
}
```

# State - width inference

blink2.dhdl

```
enum clockRate = 32_000_000;

circuit Blink2
{
    port out bool a;

    reg bool state;
    reg uint!25 counter = clockRate;

    a := state;

    if(counter != 0)
        counter := counter - 1;
    else
    {
        counter := clockRate
        state := !state;
    }
}
```

# State - width inference

blink2.dhdl

```
enum clockRate = 32_000_000;

circuit Blink2
{
    port out bool a;

    reg bool state;
    reg uint!(clockRate.log2Up) counter = clockRate;

    a := state;

    if(counter != 0)
        counter := counter - 1;
    else
    {
        counter := clockRate
        state := !state;
    }
}
```

# IO - UART

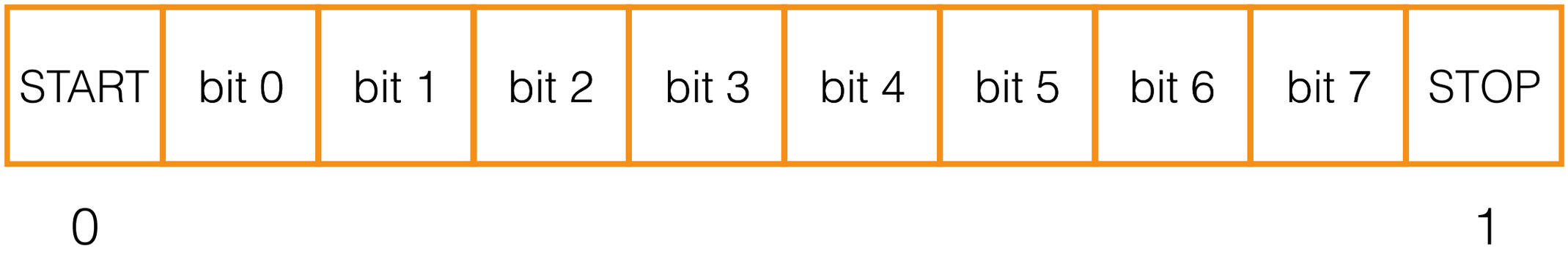
# IO - UART


byte 0
byte 2
byte 3
...
bit 7


bit 0
bit 1
bit 2
bit 3
bit 4
bit 5
bit 6
bit 7

# IO - UART


byte 0
byte 2
byte 3
...
bit 7


START
bit 0
bit 1
bit 2
bit 3
bit 4
bit 5
bit 6
bit 7
STOP
0
1

# IO - UART

serial.dhdl

```
circuit UARTTx(int symbolTime)
{
    port out bool tdx;
    port in bits!8 enq;
    port in bool valid;
    port out bool ready;

    reg bits buffer = 0b111111111;
    reg uint state = 10;
    reg uint counter = symbolTime;

    txd := buffer[0];

    ...

    ready := state == 0;
}
```

# IO - UART

```
if(state == 0)
{
    if(valid)
    {
        buffer := enq ~ 0;
        counter := time;
        state := 10;
    }
}
else
{
    if(count == 0)
    {
        buffer := 1 ~ buffer[9..1];
        counter := time;
        state := state-1;
    }
    else
        counter := counter-1;
}
```

# IO - UART

serial.dhdl

```
circuit UARTTx(int symbolTime)
{
    port out bool tdx;
    port in bits!8 enq;
    port in bool valid;
    port out bool ready;

    reg bits buffer = 0b111111111;
    reg uint state = 10;
    reg uint counter = symbolTime;

    txd := buffer[0];

    ...

    ready := state == 0;
}
```

```
port Decoupled(T)
{
    port in bool valid;
    port out bool ready;
    port in T payload;
}
```

# IO - UART

serial.dhdl

```
circuit UARTTx(int symbolTime)
{
    port out bool tdx;
    port in Decoupled!(bits!8) enq;

    reg bits buffer = 0b111111111;
    reg uint state = 10;
    reg uint counter = symbolTime;

    txd := buffer[0];

    ...

    enq.ready := state == 0;
}
```

# IO - UART

```
if(state == 0)
{
    if(enq.valid)
    {
        buffer := enq.payload ~ 0;
        counter := time;
        state := 10;
    }
}
else
{
    if(count == 0)
    {
        buffer := 1 ~ buffer[9..1];
        counter := time;
        state := state-1;
    }
    else
        counter := counter-1;
}
```

# IO - UART

serial.dhdl

```
port UARTPeripheral
{
    port in rxd;
    port out txd;
}

port UARTCtl
{
    port in Decoupled!(bits!8) enq;
    port out Decoupled!(bits!8) deq;
}

circuit UART(int symbolTime)
{
    port in UARTCtl ctl;
    port in UARTPeripheral pins;
    auto tx = new UARTTx!symbolTime;
    auto rx = new UARTRx!symbolTime;
    tx.txd <> pins.txd
    // ...
}
```

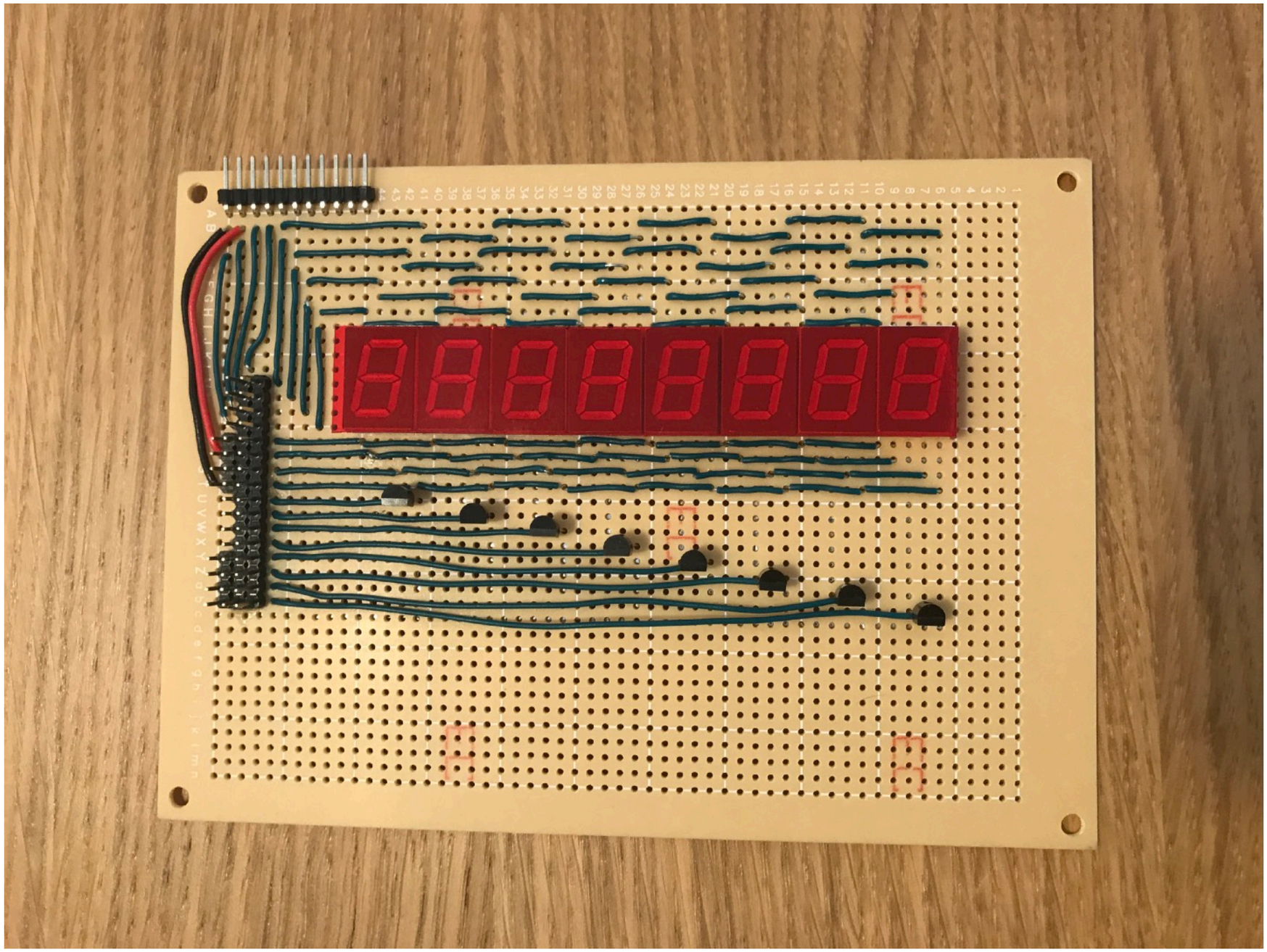
# IO - UART

test-serial.dhdl

```
circuit TestSerial
{
    port in UARTPeripheral uartPins;

    auto uart = new UART!230400;
    uart.ctl.enq.valid := true;
    uart.ctl.enq.payload := ‘D’;
    uartPins.txd = uart.pins.txd;
}
```

# IO - UART

# IO - 7 segment



# FPGA IO - 7 segment

sevensegment.dhdl

```
switch(digit)
{
    case 0x0: pins.segments := 0b1000000; break;
    case 0x1: pins.segments := 0b1111001; break;
    case 0x2: pins.segments := 0b0100100; break;
    case 0x3: pins.segments := 0b0110000; break;
    case 0x4: pins.segments := 0b0011001; break;
    case 0x5: pins.segments := 0b0010010; break;
    case 0x6: pins.segments := 0b0000010; break;
    case 0x7: pins.segments := 0b1111000; break;
    case 0x8: pins.segments := 0b0000000; break;
    case 0x9: pins.segments := 0b0011000; break;
    case 0xA: pins.segments := 0b0001000; break;
    case 0xB: pins.segments := 0b0000011; break;
    case 0xC: pins.segments := 0b1000110; break;
    case 0xD: pins.segments := 0b0100001; break;
    case 0xE: pins.segments := 0b0000110; break;
    case 0xF: pins.segments := 0b0001110; break;
}
```

# FPGA IO - 7 segment

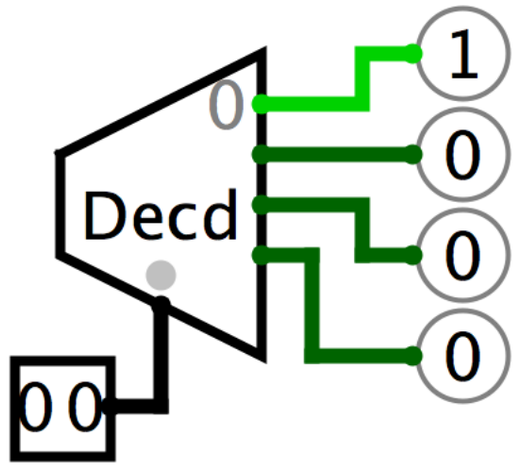
sevensegment.dhdl

```
switch(activeDigit)
{
    case 0: pins.anodes := 0b00000001; break;
    case 1: pins.anodes := 0b00000010; break;
    case 2: pins.anodes := 0b00000100; break;
    case 3: pins.anodes := 0b00001000; break;
    case 4: pins.anodes := 0b00010000; break;
    case 5: pins.anodes := 0b00100000; break;
    case 6: pins.anodes := 0b01000000; break;
    case 7: pins.anodes := 0b10000000; break;
}
```

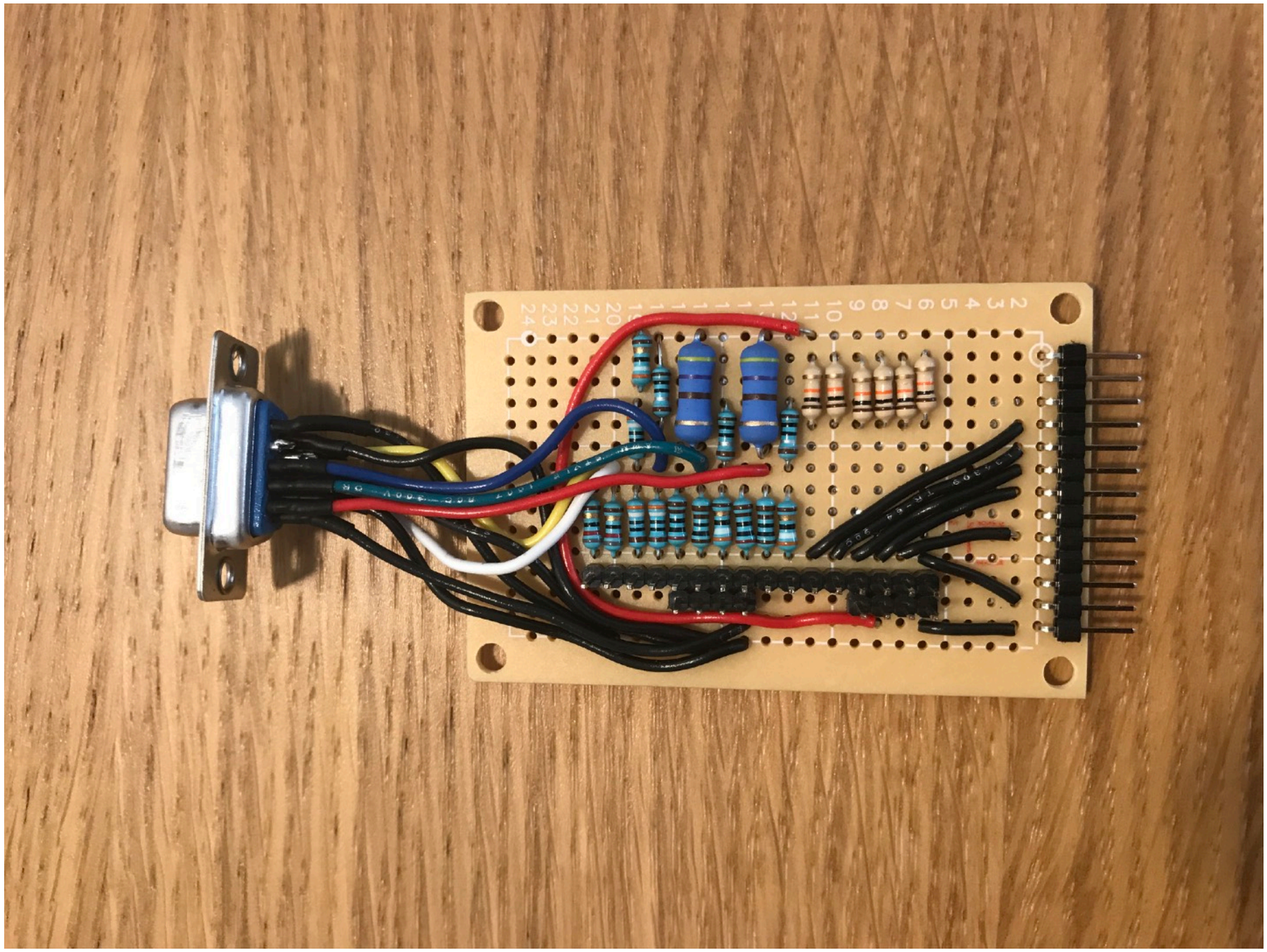
# FPGA IO - 7 segment

sevensegment.dhdl

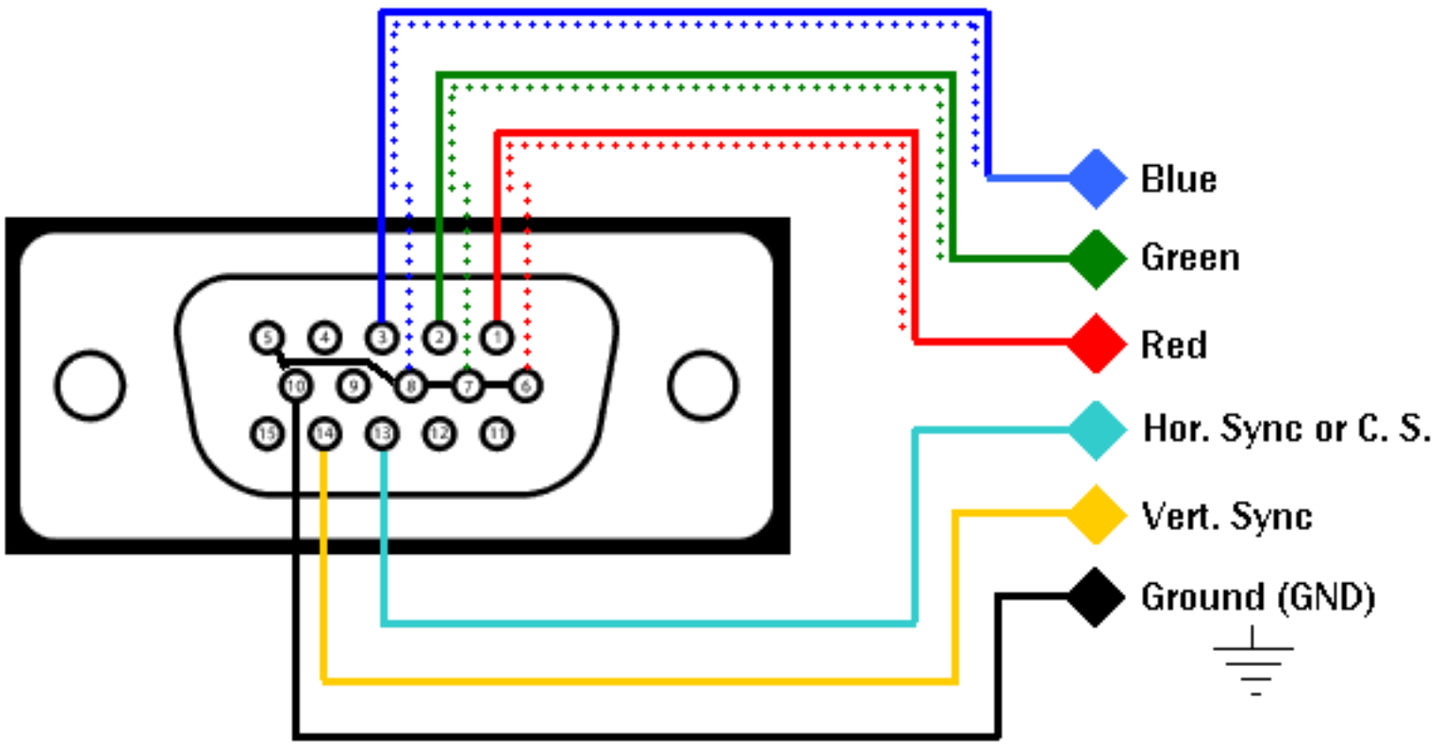
```
switch(activeDigit)
{
    case 0: pins.anodes := 0b00000001; break;
    case 1: pins.anodes := 0b00000010; break;
    case 2: pins.anodes := 0b00000100; break;
    case 3: pins.anodes := 0b00001000; break;
    case 4: pins.anodes := 0b00010000; break;
    case 5: pins.anodes := 0b00100000; break;
    case 6: pins.anodes := 0b01000000; break;
    case 7: pins.anodes := 0b10000000; break;
}
```

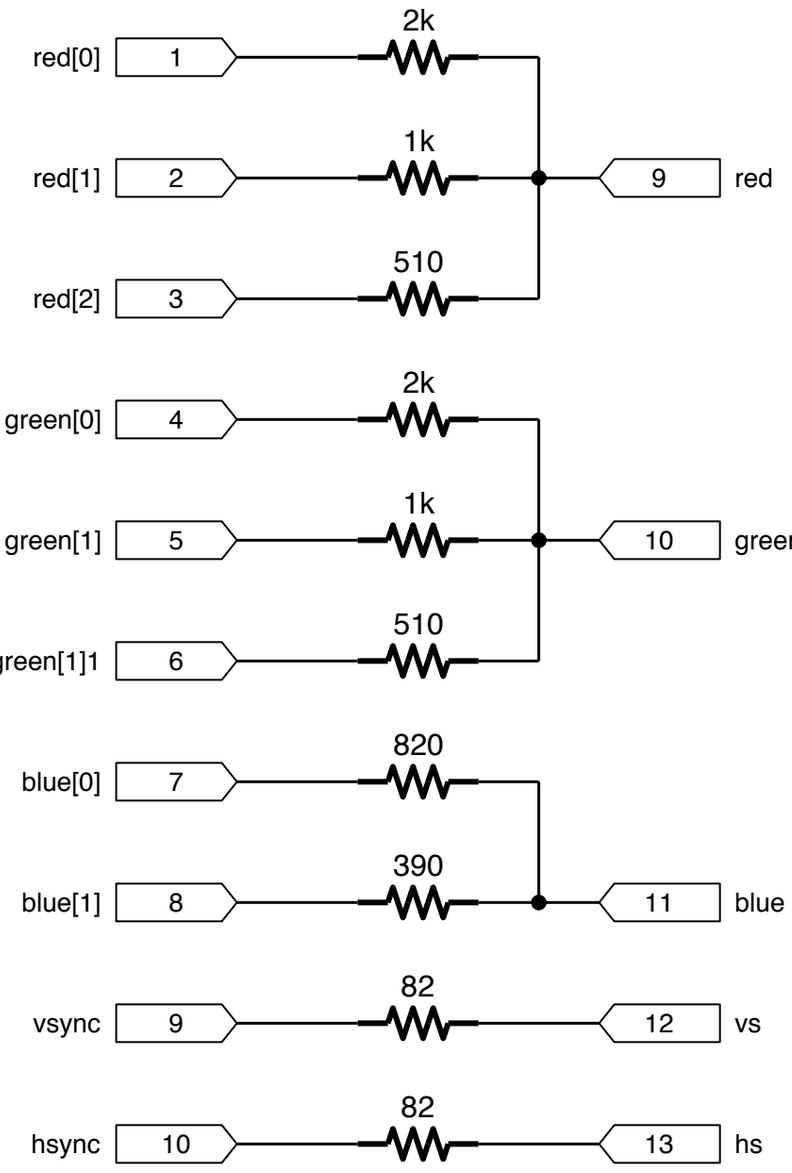


# IO - VGA

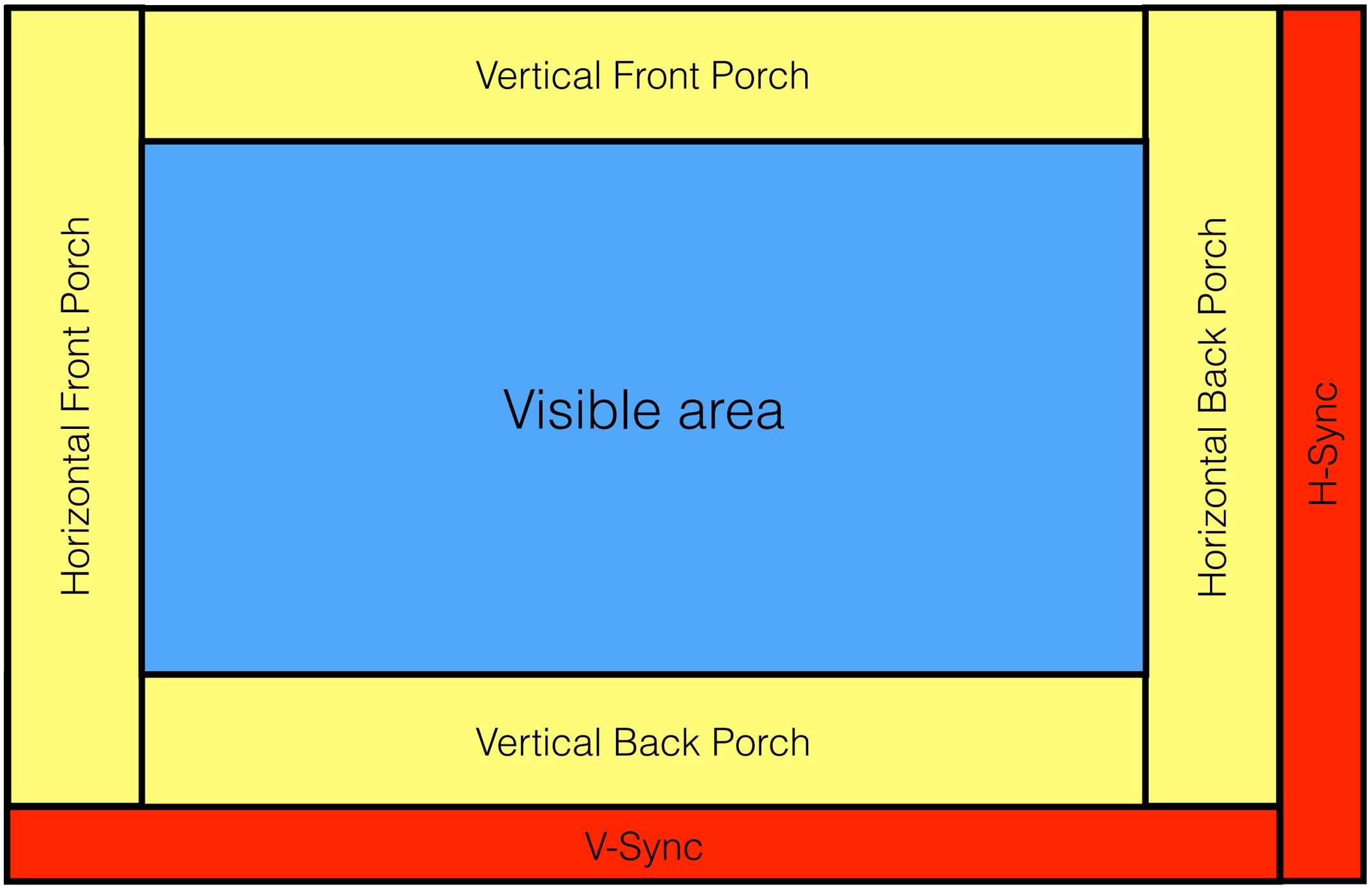


# IO - VGA

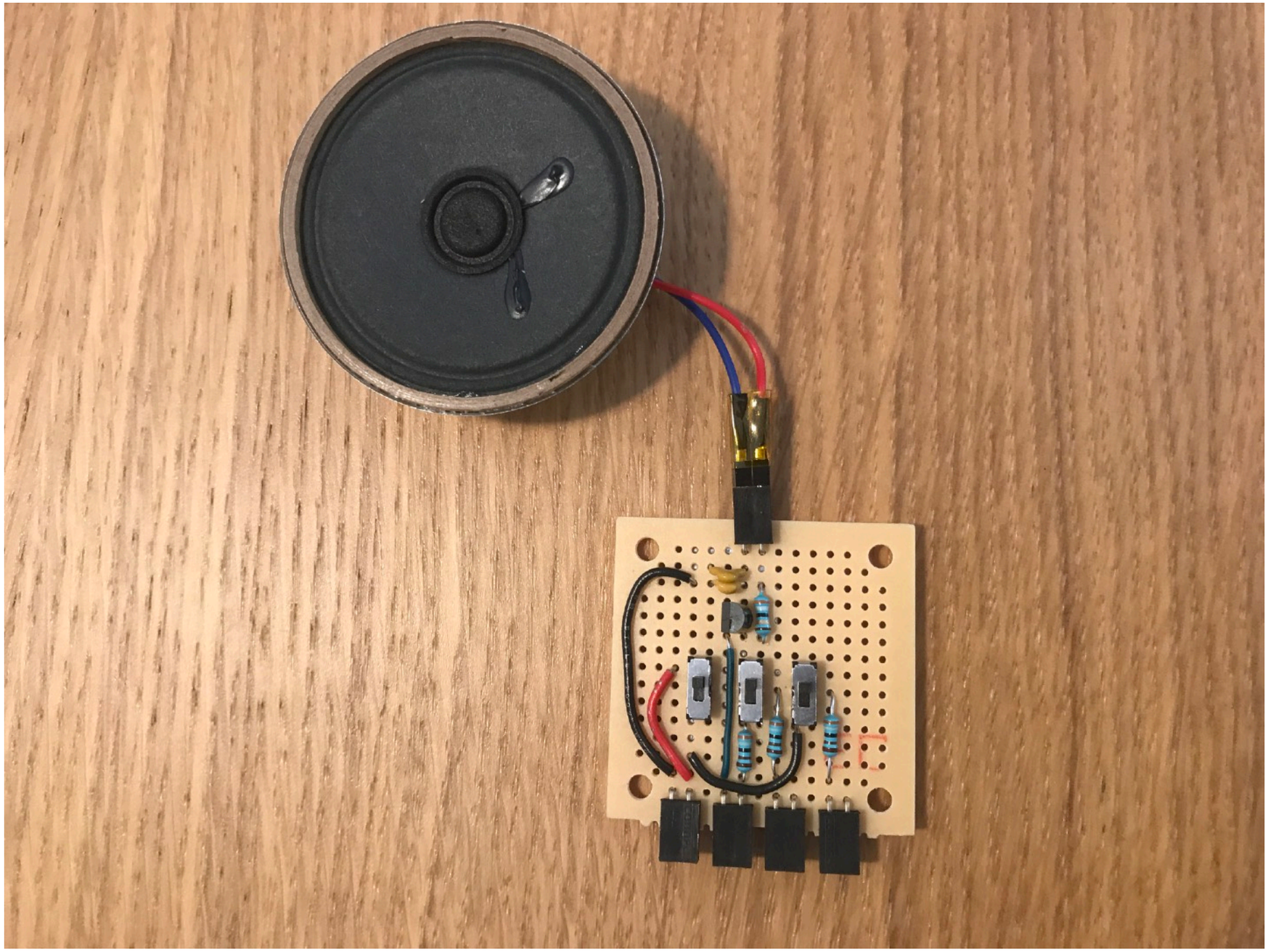

Blue
Green
Red
Hor. Sync or C. S.
Vert. Sync
Ground (GND)


red[0]
1
2k
red[1]
2
1k
9
red
red[2]
3
510
green[0]
4
2k
green[1]
5
1k
10
green
green[1]1
6
510
blue[0]
7
820
blue[1]
8
390
11
blue
vsync
9
82
12
vs
hsync
10
82
13
hs

# IO - VGA

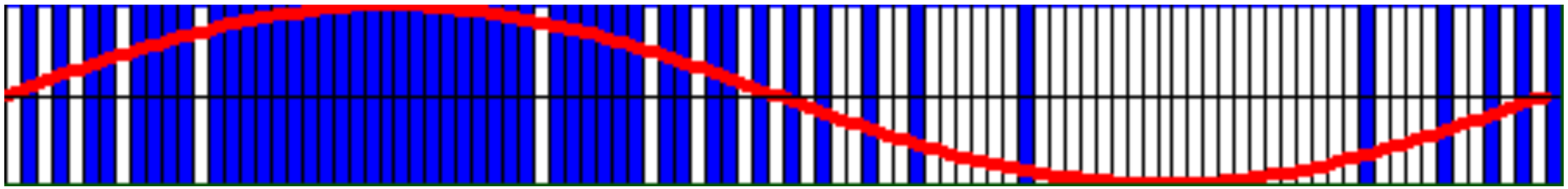

Vertical Front Porch
Horizontal Front Porch
Visible area
Horizontal Back Porch
H-Sync
Vertical Back Porch
V-Sync

# IO - Audio



# IO - Audio

- Delta-sigma DAC



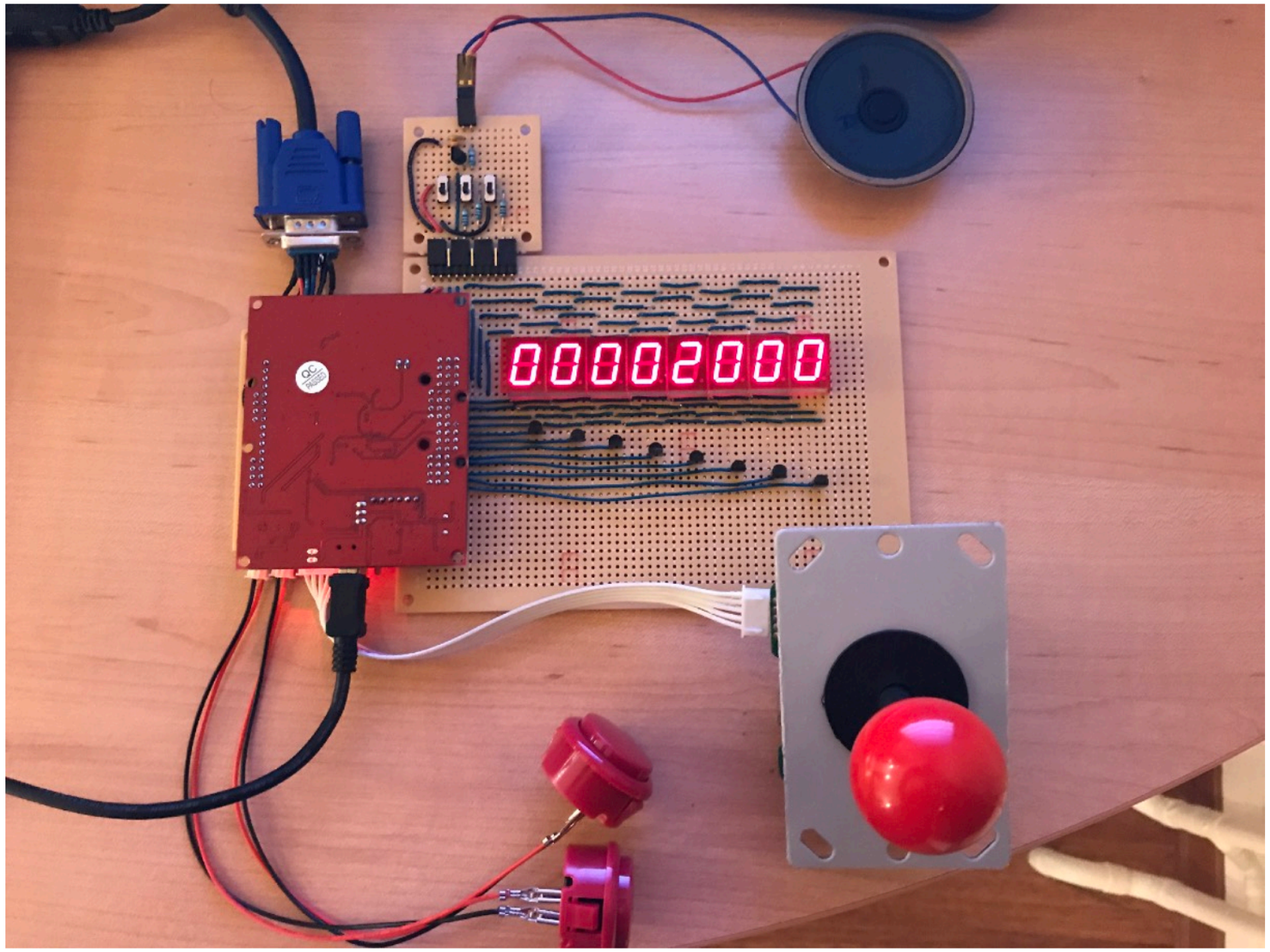
audio.dhdl

```
port out bool audio;

reg uint!8 sample;
reg uint!9 deltaSigma;

deltaSigma := deltaSigma + sample;

audio := deltaSigma[8];
```

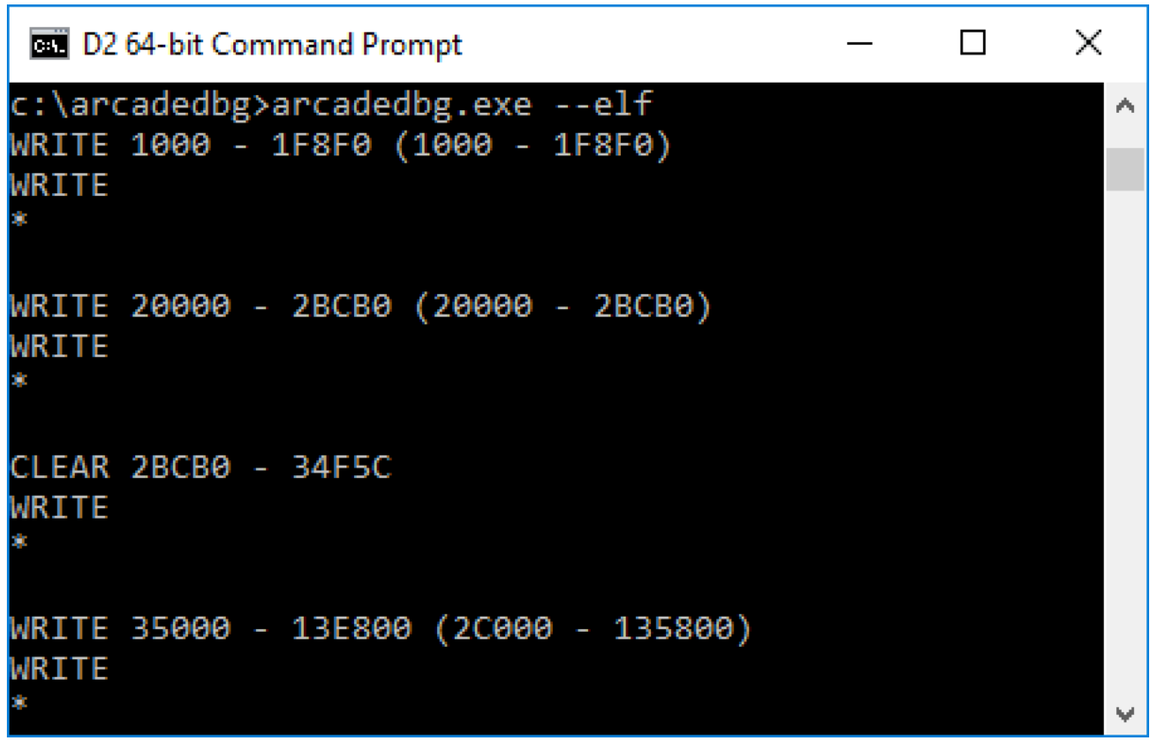
# Demo

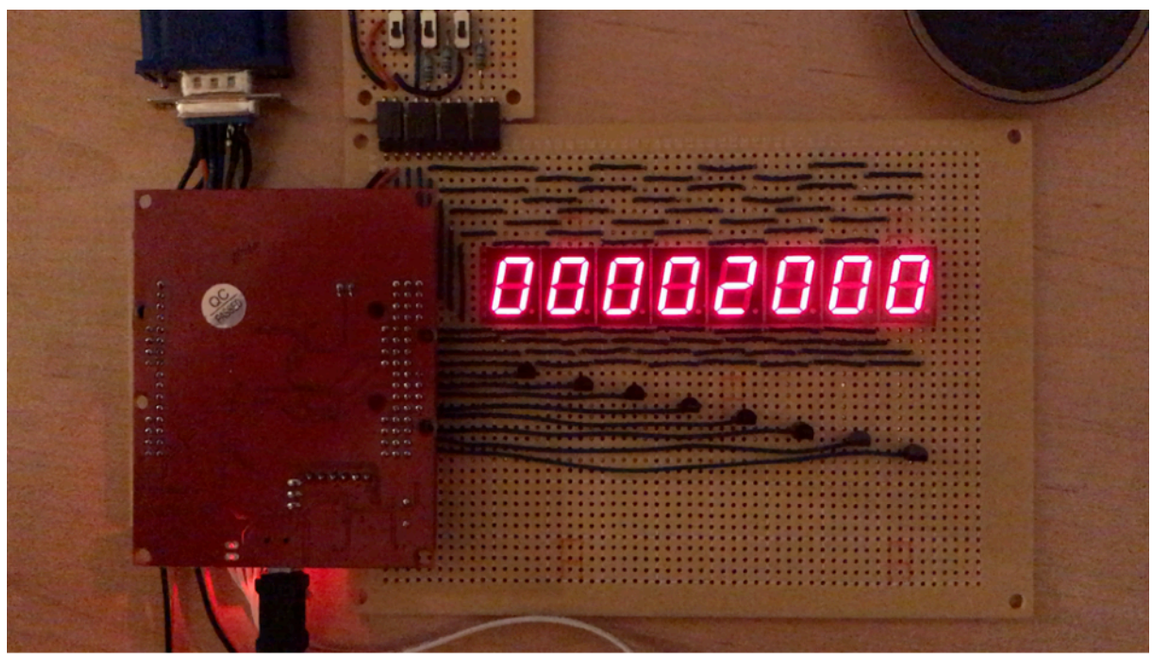
- Let’s build an arcade with these components
  - What game should we use?
  - Something somewhat related to D…

# Demo

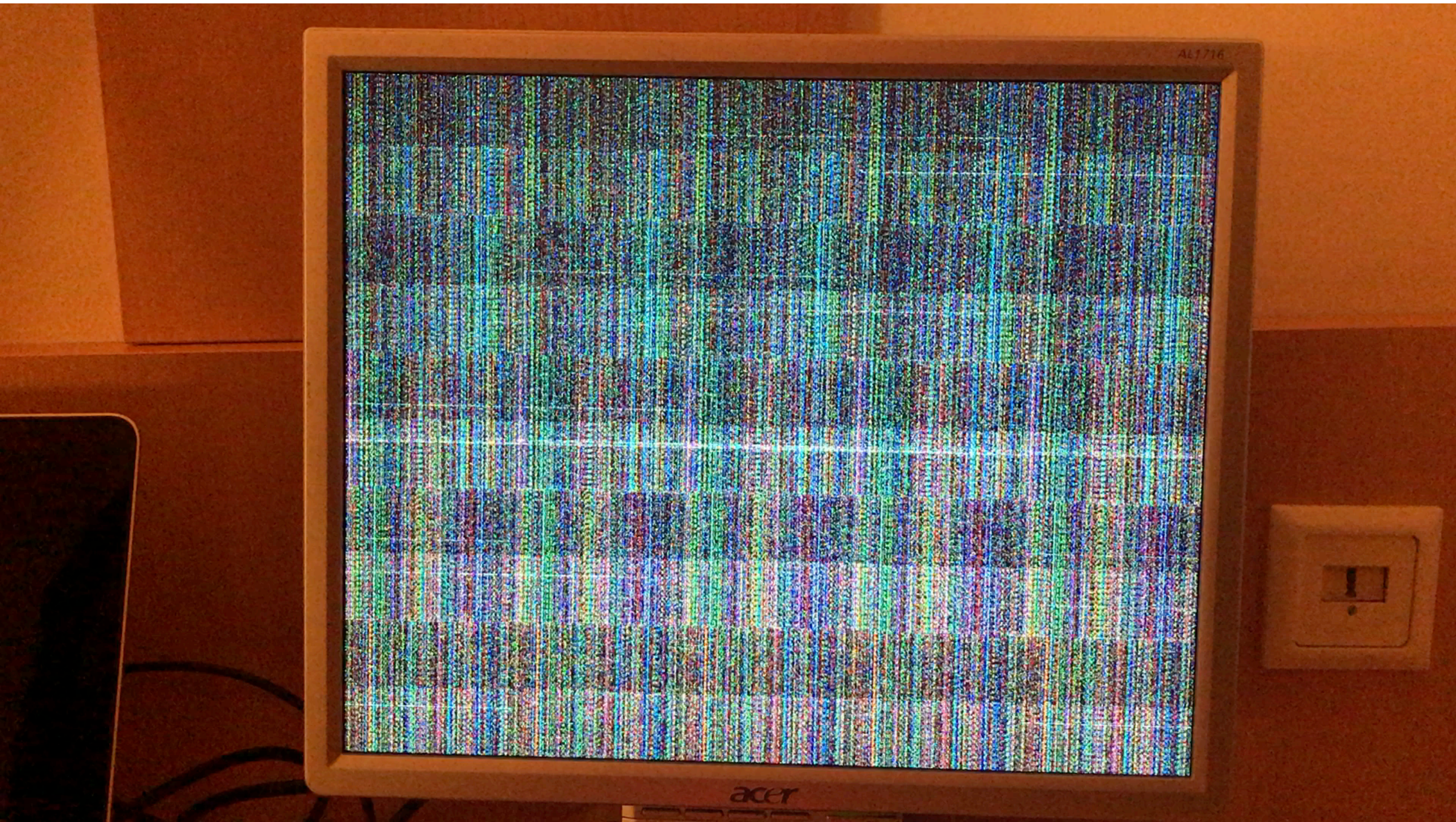


# Demo


D2 64-bit Command Prompt
c:\arcadedbg>arcadedbg.exe --elf
WRITE 1000 - 1F8F0 (1000 - 1F8F0)
WRITE
*
WRITE 20000 - 2BCB0 (20000 - 2BCB0)
WRITE
*
CLEAR 2BCB0 - 34F5C
WRITE
*
WRITE 35000 - 13E800 (2C000 - 135800)
WRITE
*


00002000

# Demo



# Demo

- Not shown for lack of time:
  - SDRAM Controller
  - RISC-V CPU
  - Cache
- LDC2 compiled with RISC-V backend
  - Lots of issues here…

Thank you