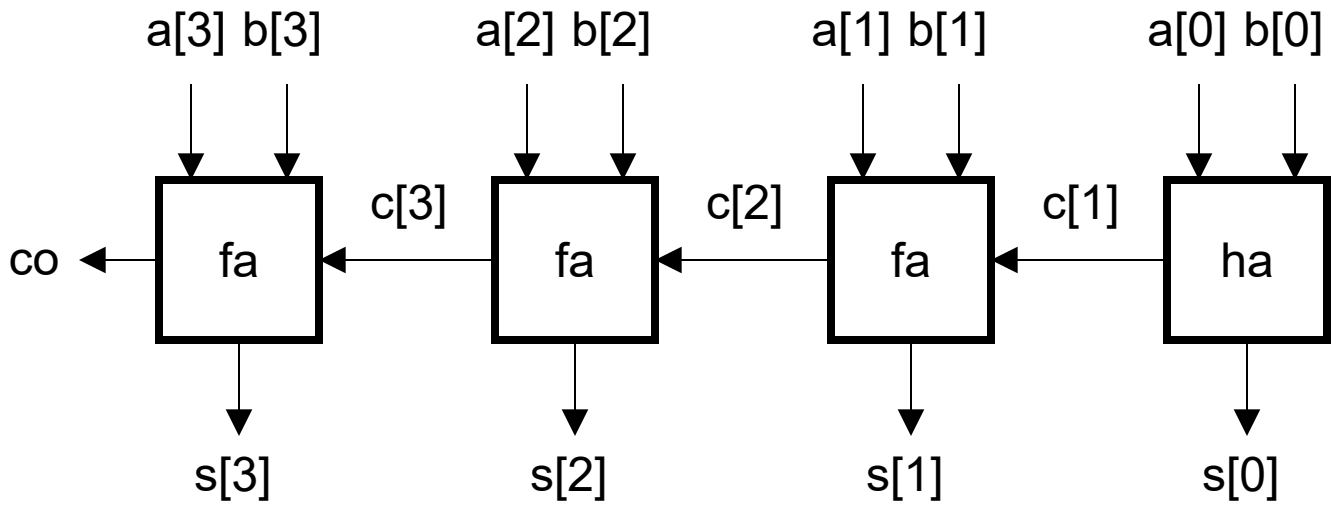
Structured design with verilog

4 bit ripple adder



- The last adder has no carry input. It is either a half-adder or a full adder with its carry input tied to 0.

4 bit ripple adder

```
module ripple_adder_4 (
      input  [3:0] a,
      input  [3:0] b,
      output [3:0] sum,
      output co
);

wire [3:0] c; // internal carry bits

full_adder fa0(a[0], b[0], 1'b0, c[1], sum[0]);
full_adder fa1(a[1], b[1], c[1], c[2], sum[1]);
full_adder fa2(a[2], b[2], c[2], c[3], sum[2]);
full_adder fa3(a[3], b[3], c[3],   co, sum[3]);

endmodule
```

- impossible to parameterize the number of instances
- hardwired structure and wiring

generate block

- allows flexibility over instantiations
  - multiple instantiations
    - iterative
    - recursive
  - conditional instantiations
- A generate block is defined inside a module, and is used only during compilation/synthesis.
- Template:

```
genvar var1, var2, ... // only if genvars are needed
generate
    generate item
endgenerate
```

Iterative design

```
module ripple_adder_4 (
    input  [3:0] a,
    input  [3:0] b,
    output [3:0] sum,
    output co
);

wire [3:0] c; // internal carry bits

full_adder fa0(a[0], b[0], 1'b0, c[1], sum[0]);
genvar g;
generate
    for(g = 1; g <= 2; g = g + 1)
        full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate
full_adder fa3(a[3], b[3], c[3], co, sum[3]);

endmodule
```

- use **generate** to repeatedly instantiate a module.

Iterative design

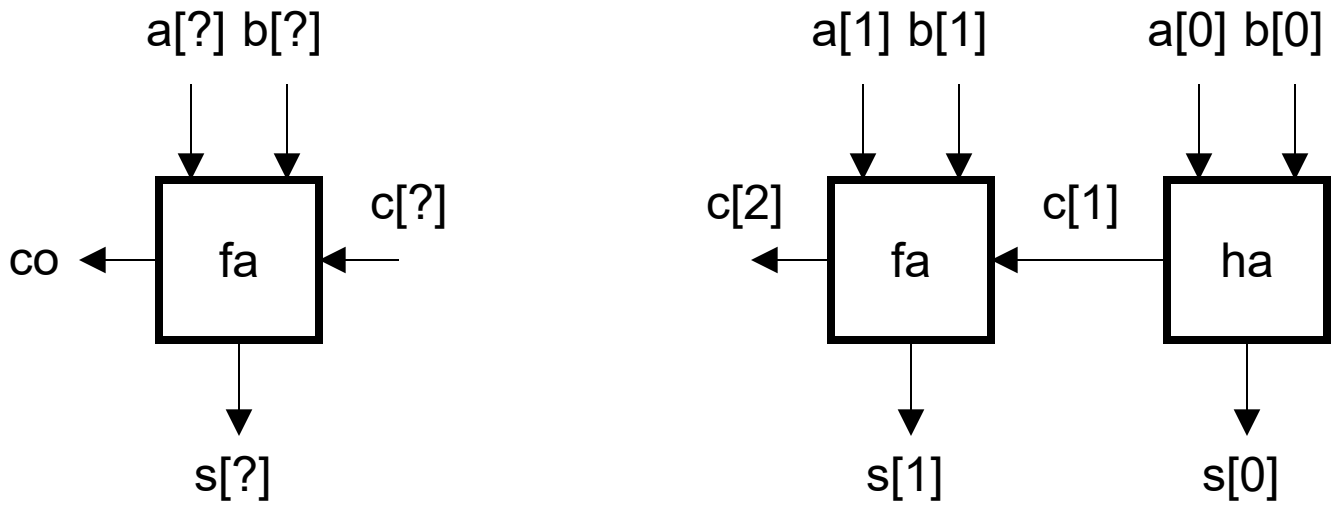
```
module ripple_adder_4 (
    input  [3:0] a,
    input  [3:0] b,
    output [3:0] sum,
    output co
);

wire [4:0] c; // includes the external carry input and output
assign co = c[4];
assign c[0] = 1'b0;
genvar g;
generate
    for(g = 0; g < 4; g = g + 1)
        full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate


endmodule
```

- use **generate** smartly.

Parameterizable ripple adder



- The number of full adders is fixed when the ripple adder is instantiated.
- Wire widths also depend on the number of full adders.

Parameterizable module

```
module ripple_adder_N #(parameter N = 4) (
    input  [N-1:0] a,
    input  [N-1:0] b,
    output [N-1:0] sum,
    output co
);

wire [N:0] c; // includes the external carry input and output
assign co = c[N];
assign c[0] = 1'b0;
genvar g;
generate
    for(g = 0; g < N; g = g + 1)
        full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate


endmodule
```

- **generate** makes parameterization possible and easy.

Para ble module

```
module ripple_adder_N #(parameter N = 4) (
    input  [N-1:0] a,
    input  [N-1:0] b,
    output [N-1:0] sum,
    output co
);

wire [N:0] c; // includes the external carry input and output
assign co = c[N];
assign c[0] = 1'b0;
genvar g;
generate
    for(g = 0; g < N; g = g + 1)
        full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate


endmo
```

parameter declaration block

declaration

the default value

parameterizable width

parameterizable bit selection

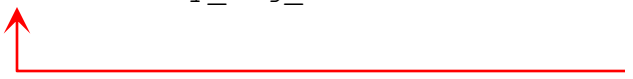
parameterizable number of instatiations

- ge neterization possible and easy.

Parameterizable module

- Parameterized instantiation:

```
ripple_adder_N #(8) my_adder(...
ripple_adder_N #(32) your_adder(...
ripple_adder_N #(128) very_big_adder(..
```



inline parameter redefinition

- The parameter value must be an integer or an integer literal.
- When the parameter is not specified at instantiation, the default value is used:

```
ripple_adder_N 4bit_adder (...
```

Conditional instantiation

```
module ripple_adder_N #(parameter N = 4) (
    input  [N-1:0] a,
    input  [N-1:0] b,
    output [N-1:0] sum,
    output co
);

wire [N:0] c; // includes the external carry input and output
assign co = c[N];

genvar g;
generate
    for(g = 0; g < N; g = g + 1)
        if(g == 0)
            full_adder fa(a[g], b[g], 1'b0, c[g+1], sum[g]); // first adder
        else
            full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate
endmodule
```

Conditional instantiation

```
module ripple_adder_N #(parameter N = 4) (
    input  [N-1:0] a,
    input  [N-1:0] b,
    output [N-1:0] sum,
    output co
);

wire [N:0] c; // includes the external carry input and output
assign co = c[N];

genvar g;
generate
    for(g = 0; g < N; g = g + 1)
        if(g == 0)
            full_adder fa(a[g], b[g], 1'b0, c[g+1], sum[g]); // first adder
        else
            full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate
endmodule
```

- **genvar** value selects between different wirings.

Conditional instantiation

```
module ripple_adder_N #(parameter N = 4) (
    input  [N-1:0] a,
    input  [N-1:0] b,
    output [N-1:0] sum,
    output co
);

wire [N:0] c; // includes the external carry input and output
assign co = c[N];

genvar g;
generate
    for(g = 0; g < N; g = g + 1)
        if(g == 0)
            half_adder fa(a[g], b[g], c[g+1], sum[g]); // first adder
        else
            full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate
endmodule
```

Conditional instantiation

```
module ripple_adder_N #(parameter N = 4) (
    input  [N-1:0] a,
    input  [N-1:0] b,
    output [N-1:0] sum,
    output co
);

wire [N:0] c; // includes the external carry input and output
assign co = c[N];

genvar g;
generate
    for(g = 0; g < N; g = g + 1)
        if(g == 0)
            half_adder fa(a[g], b[g], c[g+1], sum[g]); // first adder
        else
            full_adder fa(a[g], b[g], c[g], c[g+1], sum[g]);
endgenerate
endmodule
```

- **genvar** value selects between different module types.