Intro til FPGA

Modul

```
module test ();
endmodule
```

Modul I/O

```
module test (
   input in,
   output out
);

   assign out = in;

endmodule
```

Modul I/O

```
module sum (
   input  a,
   input  b,
   output out
);

   assign out = a + b;

endmodule
```

Modul I/O

```
module sum (
   input  a,
   input  b,
   output out
);

   wire   result = a + b;

   assign out = result;

endmodule
```

Modul I/O bredde

```
module sum (
   input  [7:0] a,
   input  [7:0] b,
   output [8:0] out
);

   assign out = a + b;

endmodule
```

Submodul

```
// out = (a + b)^2
module squaredsum (
  input  [7:0] a,
  input  [7:0] b,
  output [8:0] out
);

   wire [14:0] sum;

   sum sum_mod (
      .a   (a),
      .b   (b),
      .sum (sum)
   );

   multiply square_mod (
      .a   (sum),
      .b   (sum),
      .sum (out)
   );

endmodule
```

```
module sum (
   input  [7:0] a,
   input  [7:0] b,
   output [8:0] sum
);

   assign sum = a + b;

endmodule

module multiply (
   input  [7:0] a,
   input  [7:0] b,
   output [14:0] product
);

   assign product = a * b;

endmodule
```

Sekvensiell logikk

```
module test ();

   wire count = count + 1;

endmodule
```

Sekvensiell logikk

```
module test (
   input         clk,
   output [7:0] count
);

   reg [7:0] count_reg = 0;

   assign count = reg;

   always @(posedge clk) begin
      count_reg <= count_reg + 1;
   end

endmodule
```

Sekvensiell logikk

```
module test (
   input             clk,
   output reg [7:0] count
);

   always @(posedge clk) begin
      count <= count + 1;
   end

endmodule
```

Sekvensiell logikk reset

```
module test (
   input              clk,
   input              rst_n,
   output reg [7:0] count
);

   always @(posedge clk or negedge rst_n) begin
      if (~rst_n) begin
         count <= 0;
      end else begin
         count <= count + 1;
      end
   end

endmodule
```

Fibonacci

```
module test (
   input        clk,
   input        rst_n,
   output [7:0] fib
);

   reg [7:0] a = 0;
   reg [7:0] b = 0;

   assign fib = b;

   always @(posedge clk or negedge rst_n) begin
      if (~rst_n) begin
         a <= 0;
         b <= 0;
      end else begin
         a <= b;
         b <= a + b;
      end
   end

endmodule
```