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

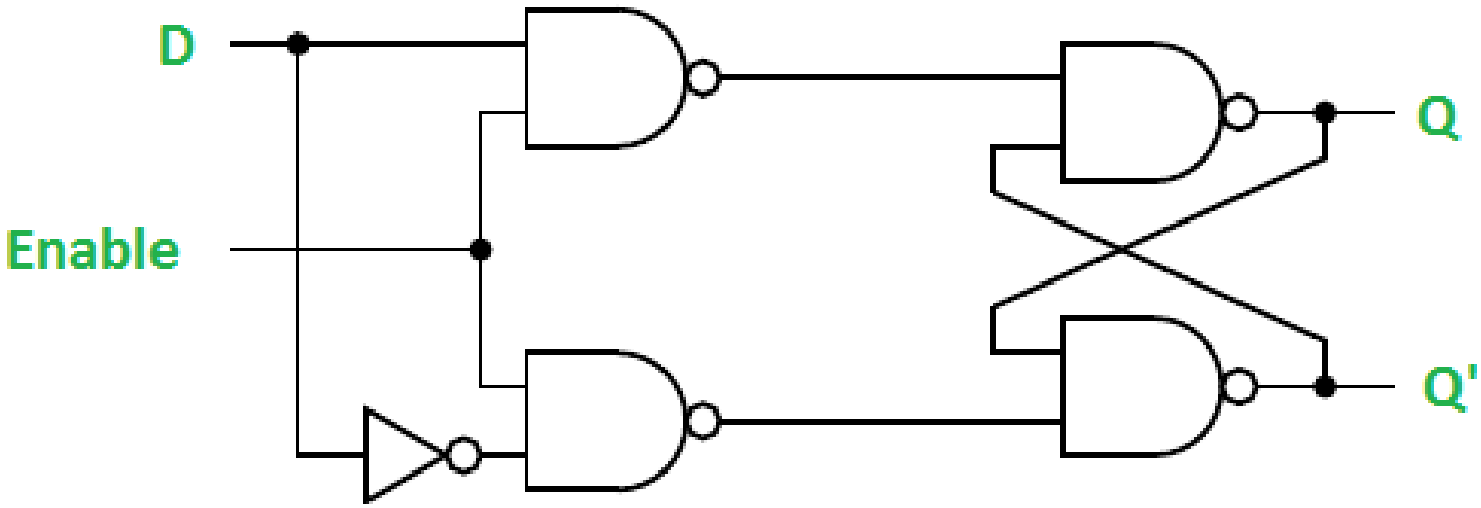
```
// out = (a + b)^2
module squaredsum (
    input  [7:0] a,
    input  [7:0] b,
    output [8:0] out
);

    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 (
    output [7:0] out
);

    // Ikke tillatt
    wire count = count + 1;

    assign out = count;

endmodule
```

Sekvensiell logikk



Sekvensiell logikk

```
module test (
    output [7:0] out
);

    reg    [7:0] count = 0;

    assign out = count;

endmodule
```

Sekvensiell logikk

```
module test (
    output [7:0] out
);

    reg    [7:0] count = 0;

    assign out = count;

    // Ikke tillatt
    assign count = count + 1;

endmodule
```

Sekvensiell logikk

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

    reg     [7:0] count = 0;

    assign out = count;

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

endmodule
```

Sekvensiell logikk reset

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

    reg     [7:0] count = 0;

    assign out = count;

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

endmodule
```

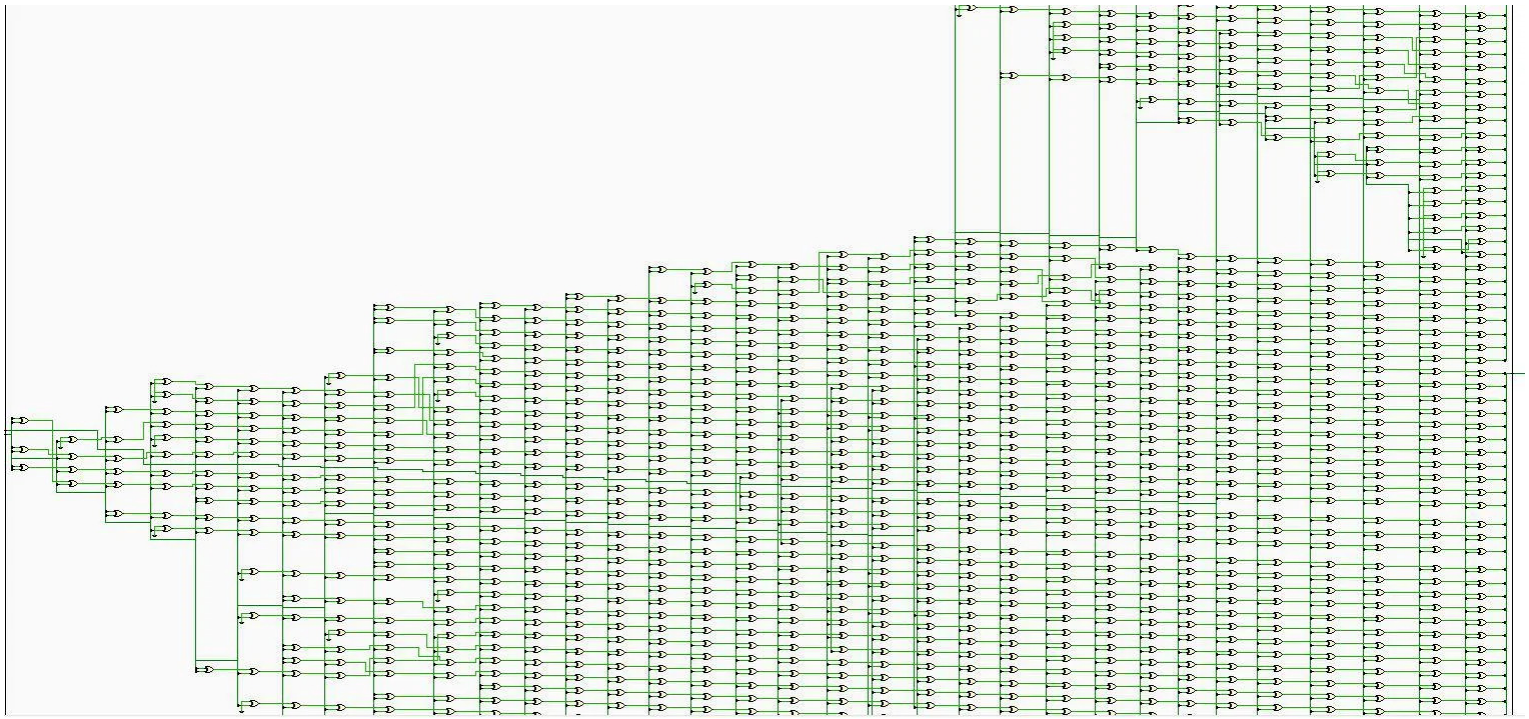
Fibonacci

```
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
```

Fibonacci blocking

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

   reg     [7:0] tmp = 0;
   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
         tmp = b;
         b = a + b;
         a = tmp;
      end
   end

endmodule
```

Lengste sti



Saktere klokke

```
module top (
  input         clk,
  output [7:0] out
);

   reg    [7:0] ticks = 0;
   reg          counter_clk = 0;

   counter counter_mod (
      .clk (counter_clk),
      .count (out)
   );

   always @(posedge clk) begin
      if (ticks == 100) begin
         ticks <= 0;
         counter_clk <= ~counter_clk;
      end else begin
        ticks <= ticks + 1;
      end
   end

endmodule
```

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

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

endmodule
```

Oppsett

- Klon repoet
- Installer iverilog
- VSCode plugin
  - Verilog-HDL/SystemVerilog/Bluespec SystemVerilog
    - Sett Settings->Verilog->Linting->Linter = iverilog
- Kjør `sudo ./run.sh task1_led`
  - Compilerer og flasher til FPGA

Oppgave 1

Få lysene til å blinke i et mønster.

F.eks.

- En teller
- En klokke
- Et fast mønster

Oppgave 2

FPGAen har fire seven-segment display. Bruk dette til å vise en teller i base 16.

Steg:

- Koble opp displayet og få det til å reagere
- Vis en teller på ett display
- Utvid telleren til hele displayet

Ekstra utfordring: tell i base 10.

I/O porter

Mappingen mellom de fysiske portene på FPGAen og navnene vi bruker i Verilog er definert av filen `cu.pcf`. Det er mulig å samle flere porter til en array, uavhengig av deres fysiske plassering.

- Finn signalet dere vil styre på diagrammet for Alchitry Io
- Spor signalet tilbake til en pin på B1A
- Finn tilsvarende pin på diagrammet for Alchitry Cu
- Spor signalet tilbake til en port på FPGAen