Verilog Modelling: 1-Gate Level Modelling: Primitives (and, or, not..), VDP, 2-Dataflow Modelling: Continuous assignment, logic, arithmetic and conditional operators. 3-Behavirioral Modelling: always @*, if-else, case. | Behavioral modelling used for complex circuit, and primilary used for sequential circuit. ⇒ 1-Initial Statement, 2-Always statement. | Procedural Statement: 1-always(signal, signal) infer combinational logic 2-always @* sensetive the all signals read within procedure. 3-always @(posedge signal, negedge signal) infer sequential logic.

```
module mux2_1_if_else(
  input wire a,
   "     "  b,
   "     "  sel,
  output reg y
);
always @*begin
  if(sel==1) begin
    y=b;
  end
  else begin
    y=a;
  end
end//always
endmodule
```

Procedural statement if-else

! if output are assigned within a procedural block it must be set to reg.

```
always @* begin
  case(sel)
    1'b0: y=a;
    1'b1: y=b
    default: y=0;
  endcase
end//alway end
endmodule
```

4 bit adder

```
assign {cout, sum}=a+b+cin;
```

concatenation

VERILOG FOR SEQUENTIAL CIRCUIT
• behavior controlled by a positive or negative edge, and clock
• sequential element are Latches and Flip-flops
• flip-flops are edgetrigerred storage device.
• Inputs are data, clocks and reset

Port range-vector

```
module Mux4to1(
  input wire [1:0] sel,
   "     "   a,
   "     "   b,
   "     "   c,
   "     "   d,
  output reg y
);
always @* begin
  case(sel)
    2'b00: y=a;
    2'b01: y=b;
    2'b10: y=c;
    2'b11: y=d;
  endcase
end
endmodule
```

```
`timescale 1ns/ 1ns
`include "Mux4to1.v"
module tb_mux4to1();
reg[1:0] sel;
reg a,b,c,d;
wire y;
Mux4to1 M0(.sel(sel),
.a(a), .b(b).c(c)----);
initial begin
  $monitor("Time,%0t sel=%b....", $time,
  sel, a, b, c, d, y);
  a=1'b0; b=1'b;
  sel=2'b00; #10;
  sel=2'b01; #10
  $finish;
end
endmodule
```

D-flip-flop witout reset

```
module DFF(
  input D,
  input clk,
  output reg Q
);
always @(posedge clk)
begin
  Q<=D;
end
endmodule
```

1-Instantiating: generate the clock at every #5 toggle, means 10ns clk period, 2-initial: blocks process statements one time, 3-forever: Infinite loop continuosly executes the statement or stat group.

```
module tb_dff();
reg d, clk;
wire Q;
//instantiate
DFF uut(.clk(clk),---);
```

clk-TB

```
//clock generation
initial begin
  clk=0;
  forever #1 clk=~clk;
end

initial begin
  d=0; #10;
  d=1; #10;
  d=0; #10;
  d=0; #10;
  $finish;
end
//dump
initial begin
  $dumpfile("dump.vcd");
  $dumpvars;
end
endmodule
```

blocking-nonblocking assignment: (=)blocking, (<=)non-blocking. Blocking statements are executed sequentially, one after the other, within the same time step. We use this typically at combinational circuit. Nonblocking execute concurrently at the end of time step. Usage typically sequential logic, such as ff capture values simultaneously.

Example design flow: High-Leveland Logic Design: Create the architecture, specify and implement required system behaviour., Verification: Confirm that design meets the specification., ★Logic Synthesis: Convert high-level description into a gate level logical net-list Physical Implementiation. ⇒ Specification ⇒ functional design ⇒ Synthesis | Synthesis tool convert the behavioral description into a gate-level logical netlist. (Yosys), (Gowin Synthesis), (Vivado), (Quartus)

Finite State Machine: • FSM: A computational model with finite state machine • Transition: A move between states based on inputs • Outputs: Generate based on current state (Moore) or state+inputs (Mealy) | Moore Machine: • Outputs depends only on current state • More stable, output changes only on clk edges. Mealy Machine: • Outputs depends on current state and inputs, • Faster response, but can have glitches.

HDL: 1-Initial idea was to create a language that could simulate digital circuit, 2-Later verilog used design logic circuits using synthesis of HDL code. 3-Finally, developed for fpga application. | Logic Circuit VS FPGA: FPGA: are semiconductor devices that are based around a matrix of configurable (CLB's) connected via programmable interconnects. • FPGA can be reprogrammed after manufacturing. | CLB (Configurable logic block): • Three essential element consist of LUT's, multiplexer, flip flopes • LUT's the primary elements that can implement the logical function (combinatorial). • Multiplexer is used to select data output between combinational and sequential logic. • So, Comb. and sequential, ff together we can implement all type of logic functions.

! Leds are on if values are zero. Ledleri yakmak için sıfır yapacaksın. Common anode (1,8V) (Pinler ⇒ 16-15-14-13-11-10) ! Button pins are S2=3.pin S3=4.pin

! 27MHz clock pin in the xtal (crystal oscillator) pin=52 ! 32 sinyal çıkışı ! tx⇒17pin ! rx=18pin

! Tang Nano 9K-menus • new project • select fpga • Design Tab: New file-Verilog - Write code inside • Process tab (Reset layout if need) • Synthesis—configuration—Dual purpose—use Done—Click synthesis • Floor planer—CST file YES—Package view and I-O constraints • select pins—save • Place & route • Programmer—USB cable detect—Embedded flash mode • upload flash

```
module and_gate(
  input wire a,b,
  output wire out
);
  and G0(out,a,b);
endmodule
```

```
module halfadder(
  input wire a,b,
  output wire sum, carry
);
assign sum=~(~a^~b);
assign carry=~(~a&~b);
endmodule
```

```
module if_else(
  input wire a,
  output reg [5:0]b
);
always @* begin
  if(a==0)
    b=6'b000000
  else
    b=6'b111111
end
endmodule
```

```
module conditional_assign(
  input wire a,
  output reg [5:0]b
);
always @* begin
  case(a)
    1'b0: b=6'b000000
    1'b1: b=6'b111111;
    default: b=6'b000000;
  endcase
endmodule
```

```
//Equality check
module button_led(
  input wire button3,
   "     "   button4,
  output reg led
);
always @* begin
  case(button3)
    button4: led=0; //equal
    default: led=1; //non-equal
  endcase
end
endmodule
```

```
module decoder2to4(
  input wire button3,
   "     "   button4,
  output reg [3:0] led
);
always @(*) begin //active low
  case({~button4, ~button3})
    2'b00: led=4'b1110;
    2'b01: led=4'b1101;
    2'b10: led=4'b1011;
    2'b11: led=4'b1111;
  endcase
end
endmodule
```

```
★module blinkled(
  input clk,
  output reg[5:0]leds      ⟶ output reg pin32
localparam WAIT_TIME=27000000;
reg [31:0] clockCounter=0;
always @(posedge clk) begin
  clockCounter <=clockCounter+1;
  if(clockCounter=WAIT_TIME)begin
    clockCounter <=0;
    leds<=~leds;          ⟶ pin32 <=~pin32
  end
end
endmodule
```

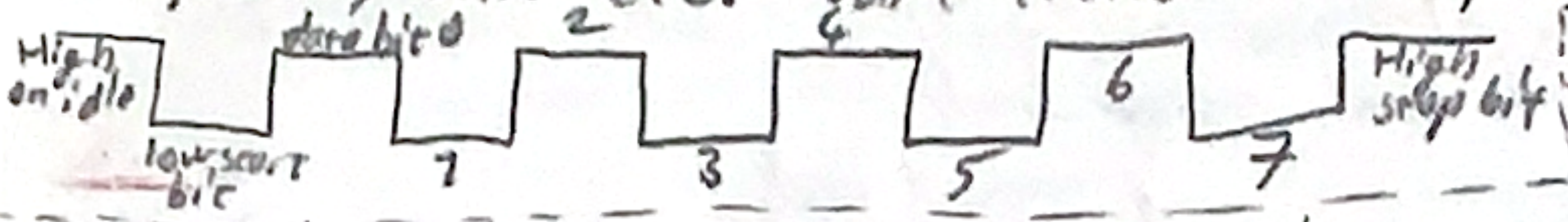
! sol alttan devam ediyor.

↳ = hoca böyle yapmış

```
module clockdivider(
  input wire clk,
  output wire out
);
reg[3:0] c1=4'b0;
always @(posedge clk) begin
  c1<=c1+1'b1;
end
assign out
=c1[3];
endmodule
```

```
module counter6bit(
  input clk,
   "   rst, //active-low
  output [5:0] led
);
localparam wait=27Mega;
reg[5:0]led_counter;
reg [31:0] clkcter=0;
always @(posedge or negedge) begin
  if(!rst) begin //active low
    clkcter<=0;
    led_counter<=0;
  end else begin
    clkcter <= clkcter +1; if(clkcter == wait) begin clkcter<=0; led_counter<=0; end end end assign led=~led_counter;
endmodule
```

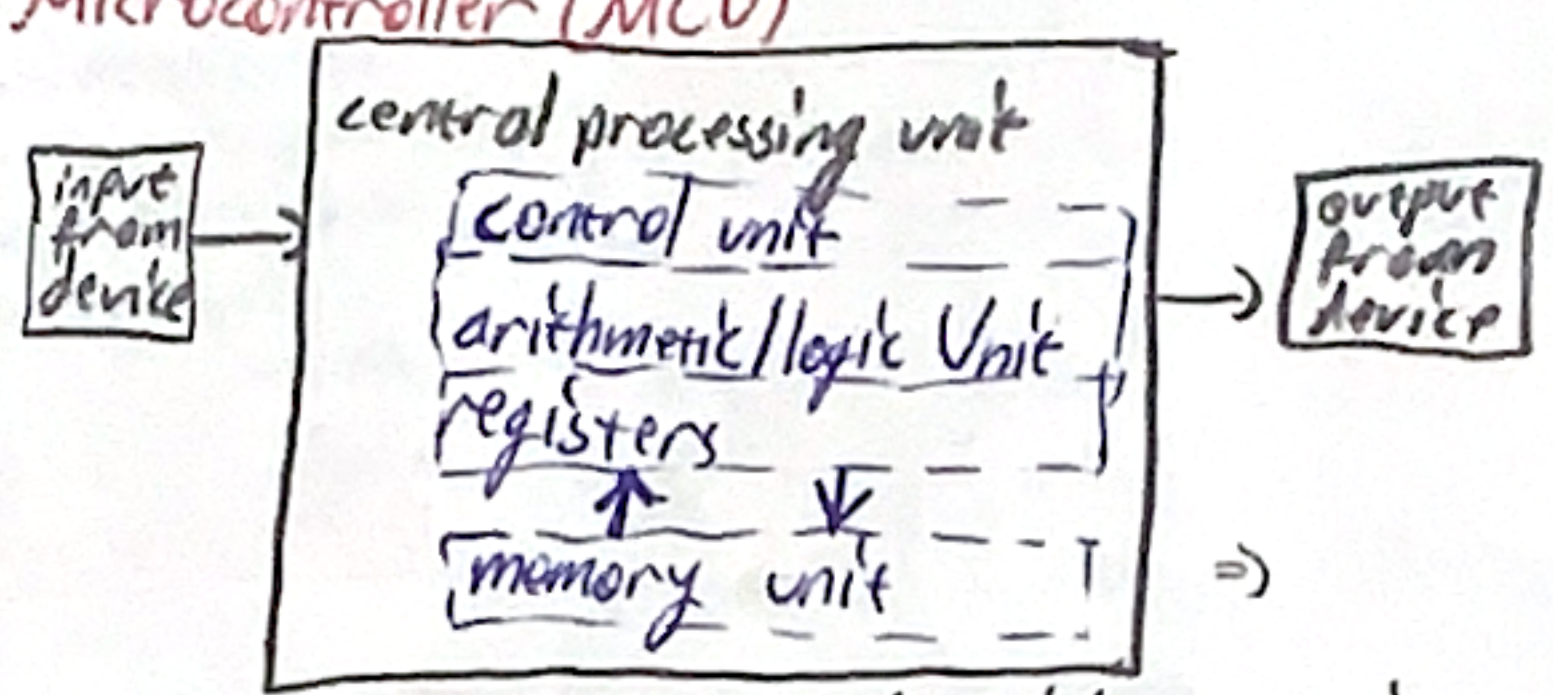
Tang Nano UART • FPGA's are also used to test communication protocols. We can generate any digital communication UART, I2C, CAN etc. • Uart transmit example ⇒ 8N1, 0start – 8bit data – 1 stop total 10 bits



Tang Nano IP cores Gowin and our design partner provide proven intellectual property (IP) for various market segment and application to accelerate your design innovation, simply your work and let you focus on your key competence. | rPLL (Phase Locked Loop), which is used to generate multiple clocks with defined phase and frequency relationship to a given input clk. ex ⇒ 6MHz clk at pin 32 from internal 27MHz clk.

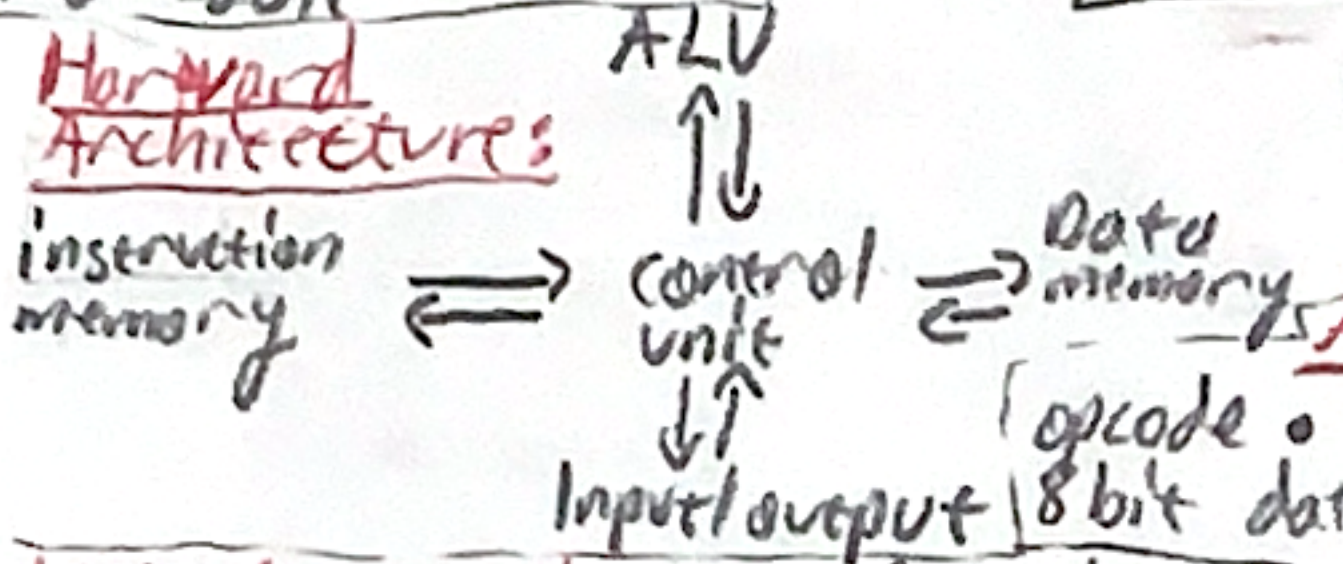
```
module ~~top~~(          rPLL
 input clkin,            örnek
 output clkout
);
 Gowin_rPLL name(
  .clkout(clkout),
  .clkin(clkin)
 );
endmodule
```

Mikrocontroller (MCU)



The Von Neumann architecture or Princeton architecture – is a computer architecture based on 1945 discription by von Neumann, and by others, in the first draft of a report on the EDVAC. The document describe a design architecture for an electronic digital computer with these components.

Harward Architecture:



• Harward architecture is named after the "Harward Mark I" which was an IBM computer in Harvard. • Two buses, one for data transfer and one for instruction fetches, thus faster. | MCU Instruction • assembly instruction ADD, SUB, LDI, NOP, JMP etc. converted into 16 bit opcode • 4–6 bit instruction, 2x5 bit register adress (ADD) • general purpose register : 5 bit adress 8 bit data

MCU: Program Memory • General purpose registers for fast add-sub operation. • Opcode kept in flash memory ex ⇒ 16 bit address | MYMCU DESIGN MCU design approaches: Architectural vs Behavioral

1– Architectural (Structural) Design: • Describe hardware structure (how compenent are connected) • Uses modules and instances to build hierarchy. • Focus on physical implementation | 2– Behavioral Design: • Describe functionality – what the circuit does • Uses algorithm and procedures • Focus on system behaviour.

– Combinational | • Mux • Decoder • Rom • Async Ram • ALU – Sequential | • Program counter • Register • Synch Ram

```
module ROM(
 input [3:0] address,
 output reg [7:0] data
);
 always @(*) begin
  case(address)
   4'h0: data=8'h13;
   4'h1: data=8'h22;
     :
   default: data=8'h00;
  endcase
 end
endmodule
```

MYMCU: FETCH → DECODE → EXECUTE • Fetch: Fetch state, the CPU retrives the next instruction from memory • The address of instruction held in Program Counter (PC) • After fetching instruction, the PC is incremented to point to the next instruction in memory. | Decode: Decode state, CPU interprets the fetched instruction. • The control unit decodes the instruction to determine what action are required. • This involves identifying the opcode and any operands (data or addresses) that are part of the instruction. | Execute: Execute state, CPU performs the operation specified by the decoded instruction. • This could involve arithmetic or logical operation, data transfer between registers, memory access, or control operations like jumps or branches. • The results of the operation are stored in the appropriate register, or memory locations.

AVR – RISCV: Riscv is an open-source instruction set architecture (ISA) based on the principle of reduced istruction set computing (RISCV). 1– Open Source, 2– Costumizeable, 3– Wide range of application, 4– Growing ecosystem.

```
module pwm(
input wire clk,
output reg pwm_out);
//system clk 100MHz
//Target freq 1032
//(100MHz/1032). 25/100
parameter freq_div=96899;
parameter duty_limit=24225;
reg [16:0] counter=0;
always @(posedge clk or
posedge rst) begin
 if(rst) begin
  counter<=0;
  pwm_out<=0;
 end else begin
  if(counter >= freq_div) begin
   counter <=0;
  end else begin
   counter <= counter+1'b1;
  end
  if(counter < duty_limit) begin
   pwm_out <=1'b1;
  else
   pwm_out <=1'b0;
  end
 end
endmodule
```

SİNAV

```
//Testbench
`timescale 1ns/1ns
module testbench();
reg a,b;
wire s,c;
//instantiation
halfadder ... GO(.S(s),
.c(c)---);
$dumpfile("dump.vcd"); $dumpvars;
$display("ab:cs",
a, b, c, s); ...
$monitor("%b%b", a,b); end
#1 a=0; b=0;
#1 a=0; b=1;
  :
$finish;
endmodule
```

• assign wire EXPRESSION end the wire.
• net data type=wire, variable data type=reg, integer
• ~=not, &=and, |=or, ^=xor

assign y=sel ? b : a;
true false

★ In the fetch stage, where the cpu retrieves an instruction from memory ★ In the decode stage, CPU interprets the instruction and prepares the neccessary signal ★ In the execute stage is when the cpu performs the operation specified by instruction. ★ PC is the address of the next instruction.

```
module led_toggle(
 input wire clk,
 output reg led
);
reg [2:0] counter; //3bit counter
always @(posedge clk) begin
 counter <=counter+1;
 //Increment counter
 if(counter == 3'b111) begin
   led<=~led;
 end
end
endmodule
```

```
`timescale 1ns/1ps
module testbench;
reg clk;
wire led;
led_toggle  ut(.clk(clk),
.led(led));
initial begin
 clk=0;
 forever #1 clk=~clk;
end
initial begin
 $dumpfile("dump.vcd");
 $dumpvars;
 #100; //Run all sim 100ns
 $finish;
end
endmodule
```

⇒ iverilog compiler
⇒ vpp sim run engine