Praktikum Sistem Digital Lanjut

Percobaan 3: Dekoder 3-ke-8 dan Demultiplekser 1-ke-8

1 Tujuan dan Sasaran

Kegiatan praktikum ini bertujuan untuk mengimplementasikan blok rangkaian kombinasional di board Xilinx Spartan-3E Starter Kit. Blok rangkaian kombinasional yang akan dibuat adalah dekoder 3-ke-8 dan demultiplekser 1-ke-8.

Sasaran kegiatan praktikum adalah:

1. Praktikan dapat memahami blok rangkaian kombinasional dekoder 3-ke-8 dan demux 1-ke-8;
2. Praktikan dapat mengimplementasikan blok rangkaian tersebut sebagai modul menggunakan desain secara prosedural maupun struktural;
3. Praktikan dapat mengaplikasikan modul (*reusable*) tersebut untuk membuat suatu rangkaian berbasis komponen;
4. Praktikan dapat memahami (dan membedakan) hasil sintesis rangkaian RTL (register transfer level) kedua bentuk desain tersebut serta utilisasi/penggunaan device untuk desain tersebut;
5. Praktikan dapat menganalisis perilaku masukan-keluaran desain di board Starter Kit;

Sumber referensi yang bisa digunakan:

1. UG230: Spartan-3E FPGA Starter Kit Board User Guide, Xilinx, June 2008
2. Spartan-3E Starter Board Schematic, Digilent, Feb 2006
3. Verilog Tutorial (online): http://www.asic-world.com/verilog/veritut.html

2 Alat dan Bahan

Alat dan bahan yang digunakan adalah:

1. Board Starter Kit Spartan-3E berbasis Xilinx FPGA XC3S500E-4FG320C;

   Modul (komponen) I/O yang akan digunakan dalam praktikum:

   a) 4 buah tombol push-button (BTN North, BTN East, BTN South, BTN West);

   b) 4 buah saklar geser (SW0, SW1, SW2, SW3);

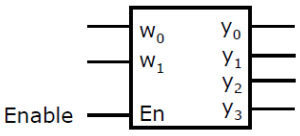
   c) 8 buah LED (LD0-7);
2. Kabel USB dengan konektor tipe-B;
3. Adaptor sumber daya DC 5 Volt;
4. Software Xilinx ISE Webpack 11.1;

3 Dasar Teori

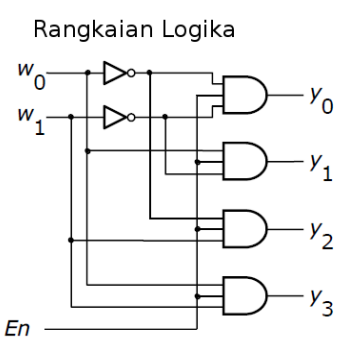
Board Starter Kit beserta komponennya dan Xilinx ISE Webpack telah dijelaskan dalam modul praktikum 1. Di bab ini akan dijelaskan tentang blok rangkaian kombinasional yang akan digunakan dalam praktikum ini. Blok rangkaian kombinasional yang akan dibahas adalah dekoder 3-ke-8 dan demux 1-ke-8.

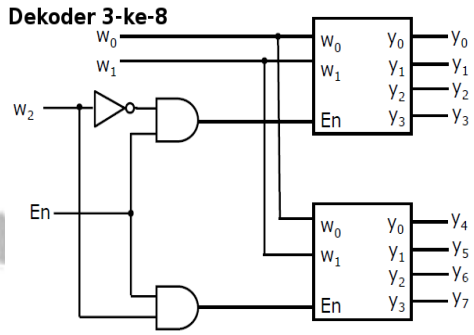
3.1 Dekoder 3-ke-8

Rangkaian dekoder 3-ke-8 mendekodekan data masukan 3-bit menjadi data 8-keluaran. Hanya satu keluaran yang diaktifkan (*one-hot encoding*) dan tiap keluaran menyatakan valuasi nilai masukannya. Dekoder 3-ke-8 dapat tersusun atas 2 buah dekoder 2-ke-4 (Gambar 1). Salah satu aplikasi dekoder adalah untuk mendekodekan jalur alamat di chip memori.



En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

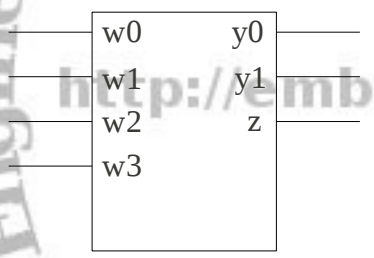




Gambar 1: Blok rangkaian dekoder 3-ke-8 tersusun atas 2 buah dekoder 2-ke-4

3.2 Demux 1-ke-8

Demux 1-ke-8 dapat diimplementasikan dengan menggunakan dekoder 3-ke-8. Jalur data {w0,w1,w2} digunakan **sebagai jalur kontrol** untuk menempatkan data 1 masukan En ke salah satu dari 8 jalur keluaran.



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	D	D	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Gambar 2: Enkoder 4-ke-2 dengan w3 mempunyai prioritas tertinggi. Keluaran z mengindikasikan semua masukan bernilai '0'

4 Cara Kerja

Kegiatan praktikum dilakukan untuk memenuhi kebutuhan desain yang diinginkan. Setiap tahap dilakukan berdasarkan cara kerja yang diuraikan dalam Subbab 4.2. Hasil kegiatan praktikum yang ditandai dengan ikon dituliskan dalam Lembar Isian Kegiatan. Laporan akhir disusun dengan menyertakan hasil kegiatan praktikum.

4.1 Kebutuhan Desain

Desain yang akan diimplementasikan dalam praktikum ini adalah:

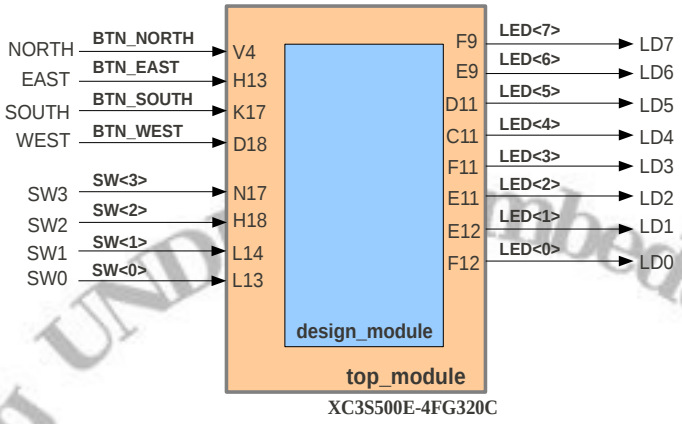
1. Dekoder 3-ke-8 (DEC_3TO8)
2. Demultiplekser 1-ke-8 (DEMUX_8)

Tiap desain akan dibuat secara prosedural dan struktural untuk dibedakan bagaimana rangkaian logikanya setelah sintesis.

Jelaskan tentang desain secara prosedural dan struktural sehingga terlihat perbedaan keduanya. Apa kelebihan dan kekurangannya masing-masing

Semua desain dimasukkan dalam proyek di praktikum sebelumnya (lihat modul praktikum 2).

Desain/fungsi akan dibuat sebagai modul, yang akan kemudian dipanggil oleh top_module (Gambar 3). Diagram blok top_module (yang menggambarkan port masukan dan keluaran) untuk semua fungsi/desain **sama**, sehingga bisa disalin untuk menghemat waktu praktikum. Perilaku tiap modul mengikuti tabel kebenaran fungsi yang dijabarkan dalam bab 3. Desain-desain tersebut di atas menggunakan file konstrain *rangkaian_kombinasional.ucf* yang telah ada di proyek. Koneksi sinyal dari tiap modul dengan top_module dijabarkan dalam Tabel 1.



Gambar 3: Diagram blok rancangan sistem masukan-keluaran (top_modul)

Sinyal	Desain Modul			
	MUX_4	DEC_3TO8	DEMUX_8	ENC_4TO2
BTN_NORTH	W<0>	**NC**	**DATA_IN**	NC
BTN_EAST	W<1>	**NC**	**NC**	NC
BTN_SOUTH	W<2>	**NC**	**NC**	NC
BTN_WEST	W<3>	**NC**	**NC**	NC
SW<3>	NC	**ENABLE**	**NC**	W<3>
SW<2>	NC	**W<2>**	**SEL<2>**	W<2>
SW<1>	S<1>	**W<1>**	**SEL<1>**	W<1>
SW<0>	S<0>	**W<0>**	**SEL<0>**	W<0>
LED<7>	NC	**Y<7>**	**Y<7>**	ZERO
LED<6>	NC	**Y<6>**	**Y<6>**	NC
LED<5>	State S<1>	**Y<5>**	**Y<5>**	NC
LED<4>	State S<0>	**Y<4>**	**Y<4>**	NC
LED<3>	NC	**Y<3>**	**Y<3>**	NC
LED<2>	NC	**Y<2>**	**Y<2>**	NC
LED<1>	NC	**Y<1>**	**Y<1>**	Y<1>
LED<0>	f (OUT)	**Y<0>**	**Y<0>**	Y<0>

Tabel 1: Koneksi sinyal tiap modul dengan top_modulnya. NC menandakan no-connect. Baris yang diarsir merupakan keluaran

4.2 Langkah Kerja

Kegiatan praktikum meliputi hal-hal sebagai berikut:

1. Menulis kode HDL untuk keempat blok sistem di atas sebagai modul, masing-masing dengan desain prosedural dan struktural, sehingga total ada 4 modul. Masukkan modul-modul tersebut dalam proyek *rangkaian_kombinasional*;
2. Mengaplikasikan modul-modul tersebut dalam top_modulnya membentuk satu desain;

3. Menambah satu file konstrain yang digunakan oleh semua desain;
4. Mensintesis dan mengimplementasikan untuk tiap desain;
   a) Melihat skematik RTL dari tiap desain;
   b) Melihat utilisasi device dari tiap desain;
5. Membangkitkan file programming *.bit untuk tiap desain;
6. Memprogram file *.bit tersebut dan mengamati perilaku sistem;

4.2.1 Membuat Proyek Baru

Proyek baru tidak perlu dibuat. Masukkan semua modul ke proyek di praktikum sebelumnya.

4.2.2 Menambah Modul Desain dan Konstrain

Modul yang perlu ditambahkan ada 4 buah. Langkah-langkahnya adalah sebagai berikut:

1. Buat modul-modul verilog seperti dalam Tabel 2. Masukkan dalam proyek;

No	Nama Modul	Masukan	Keluaran	Keterangan
1.	dec_3to8_proc	w_in[2:0] enable	y_out[7:0]	prosedural
2.	dec_3to8_struc			struktural
3.	demux_8_proc	data_in sel[2:0]	data_out[7:0]	prosedural
4.	demux_8_struc			struktural

Tabel 2: Nama modul-modul desain yang akan dibuat

2. Edit file dec_3to8_proc.v, dec_3to8_struc.v, demux_8_proc.v dan demux_8_struc.v. Listing kode sumber HDL untuk semua desain modul ada dalam Lampiran;

   Catatan: seringkali untuk membuat desain struktural diperlukan bantuan K-map untuk mensintesis fungsi menjadi ekspresi paling sederhana.

   Tuliskan persamaan fungsi untuk dec_3to8 dan demux_8 dalam lembar kegiatan;

   Jelaskan perbedaan fungsi dan aplikasi antara decoder 3-to-8 dengan demultiplekser 1-ke-8!

3. Cek file konstrain rangkaian_kombinasional.ucf. Isi file tersebut ada dalam Lampiran;

4.2.3 Mengaplikasikan Modul dalam Desain (Top Module)

Tiap modul/komponen akan diaplikasikan dalam top_modulenya masing-masing (desain berbasis komponen). Langkah yang diperlukan adalah:

1. Buat top_module untuk tiap-tiap modul. Berikan nama filenya <nama_modul>_top.v, sehingga akan ada 4 buah file *_top.v. **Ingat**: semua *_top.v mempunyai port masukan dan keluaran yang sama (Gambar 3), sehingga bisa disalin dari satu *_top.v ke lainnya;
2. Panggil modul yang bersesuaian dari modul *_top ini dan interkoneksikan port-portnya. Listing kode sumber HDL untuk semua *_top.v ada dalam Lampiran;

   Catatan: demux_8_proc_top dan demux_8_struc_top hampir mirip, yang membedakan adalah pemanggilan modul di dalamnya. Modul demux_8_proc_top memanggil modul demux_8_proc, sedangkan demux_8_struc_top memanggil modul demux_8_struc

No	Nama Modul	Masukan	Keluaran	Keterangan
1.	dec_3to8_proc_top	BTN_NORTH, BTN_EAST, BTN_SOUTH, BTN_WEST, SW[3:0]	LED[7:0]	prosedural
2.	dec_3to8_struc_top			struktural
3.	demux_8_proc_top			prosedural
4.	demux_8_struc_top			struktural

Tabel 3: Nama top_module dari desain yang akan dibuat

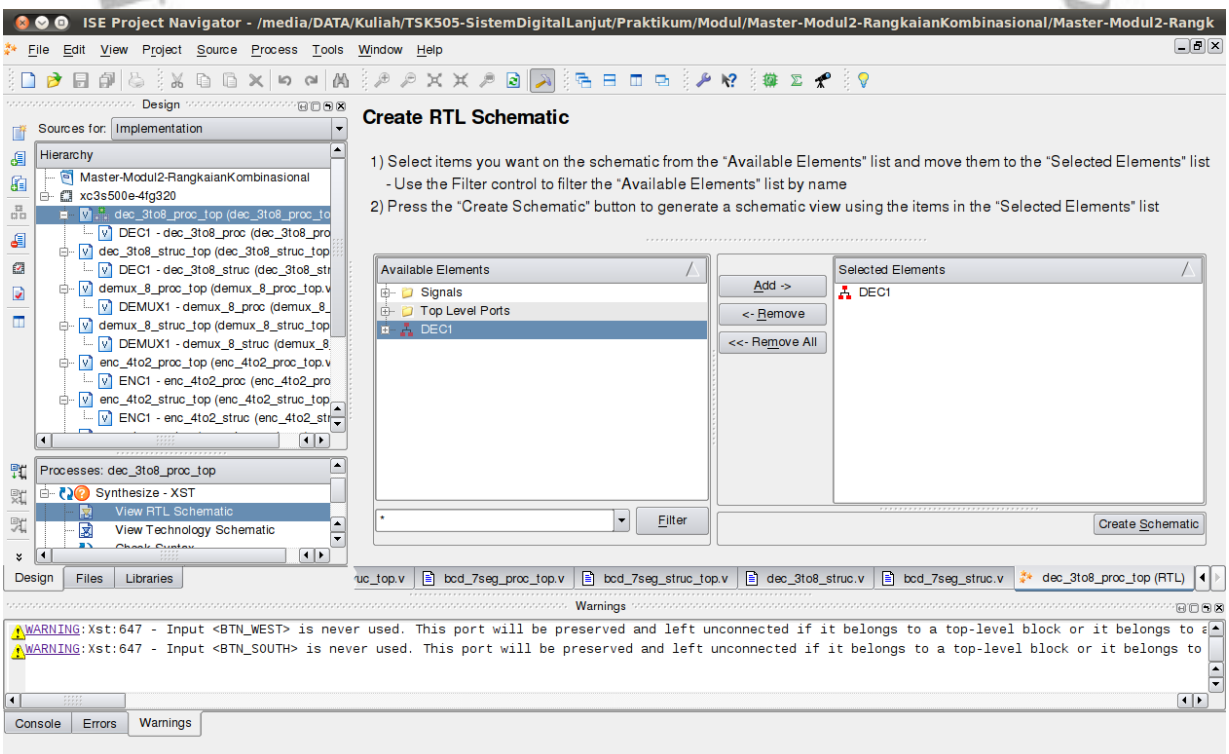
4.2.4 Sintesis dan Implementasikan Top Modul

Tiap modul *_top (4 modul) akan disintesis dan diimplementasikan. Sebaiknya melakukan langkah di 4.2.4 dan 4.2.5 secara berurutan untuk tiap desain (modul *_top). Ini dapat menghemat waktu Anda. Langkah-langkahnya adalah sebagai berikut:

1. Set *_top modul yang akan disintesis dan diimplementasikan sebagai top_module. Klik kanan *_top modul tersebut dan pilih 'Set as Top Module'. Misalnya: untuk mensintesis modul demux_8_proc_top, klik kanan modul tersebut dan pilih 'Set as Top Module'. Modul demux_8_proc_top akan menjadi top_module dan siap untuk disintesis;
2. Bangkitkan skematik RTL dari top_module. Klik “View RTL Schematic” dari proses Synthesize. Tambahkan elemen yang tersedia dan klik tombol “Create Schematic”. Lihat blok dari elemen dan bagaimana isinya (rangkaian logikanya). Misalnya: untuk melihat skematik RTL dari DEC1 (Gambar 4);

   Simpan gambar blok elemen beserta rangkaiannya dalam file gambar (dengan snapshot). Skematik ini beserta penjelasannya harus dilampirkan di laporan
3. Sintesis dan implemen top_module dengan mengklik kanan proses “Implement Design” dan pilih “Rerun All”;

   Lihat ringkasan laporan desain dari implementasi top_module tersebut dengan melihat tab Design Summary. Jika belum ada, klik menu Project->Design Summary/Report. Catat hasilnya untuk mengisi tabel utilisasi device di lembar kegiatan;



Gambar 4: Membangkitkan skematik RTL

4.2.5 Memprogram FPGA dan Pengamatan Perilaku Sistem

Langkah percobaan:

1. Bangkitkan file programming *.bit untuk top_module;
2. Jalankan Xilinx iMPACT jika belum dijalankan. Pastikan kabel USB telah terpasang di starter kit dan komputer;
3. Dari Xilinx iMPACT, klik ganda “Boundary Scan”. Kemudian klik kanan dan pilih Initialize Chain (Ctrl+I) untuk menginisialisasi Boundary Scan. Langkah inisialisasi chain hanya dilakukan sekali untuk semua top_module;
4. Yang diperlukan adalah mengkonfigurasi FPGA, sedangkan flash XCF04S dan CPLD XC2C64 dibiarkan bypass. Klik kanan device FPGA dan pilih “Assign New Configuration File”. Pilih file *.bit yang ingin diprogram ke FPGA. Nama file konfigurasi akan tampil di bawah device FPGA;
5. Klik kanan device FPGA dan pilih “Program” untuk mengkonfigurasi FPGA;
6. Perilaku sistem dapat dilihat dengan memberikan nilai masukan dari BTN_* dan SW[3:0] serta mengamati keluarannya berupa penyalaan LED

Amati perilaku sistem dan isi tabel di lembar kegiatan;

4.3 Lampiran Kode HDL

Nama file modul bersesuaian dengan nama modulnya. Misalnya: modul demux_4_proc berada dalam file demux_4_proc.v. Modul dec_3to8_proc_top akan berada dalam file dec_3to8_proc_top.v.

4.3.1 DEC_3TO8: dec_3to8_proc dan dec_3to8_struc

```
module dec_3to8_proc(
    input [2:0] w_in,
    input enable,
    output [7:0] y_out
    );
      reg [7:0] y_out_reg;

      /*Active high decoder*/
      always @(w_in, enable) begin
            case (w_in)
               /*Z=high-impedance*/
               3'b000: y_out_reg <= (enable == 1)? 8'b00000001:8'bZ;
               3'b001: y_out_reg <= (enable == 1)? 8'b00000010:8'bZ;
                3'b010: y_out_reg <= (enable == 1)? 8'b00000100:8'bZ;
                3'b011: y_out_reg <= (enable == 1)? 8'b00001000:8'bZ;
                3'b100: y_out_reg <= (enable == 1)? 8'b00010000:8'bZ;
                3'b101: y_out_reg <= (enable == 1)? 8'b00100000:8'bZ;
                3'b110: y_out_reg <= (enable == 1)? 8'b01000000:8'bZ;
                3'b111: y_out_reg <= (enable == 1)? 8'b10000000:8'bZ;
                default: y_out_reg <= (enable == 1)? 8'b00000001:8'bZ;
            endcase
      end

      assign y_out = y_out_reg;

endmodule
```

```
module dec_3to8_proc_top(
    input BTN_NORTH,
    input BTN_EAST,
    input BTN_SOUTH,
    input BTN_WEST,
    input [3:0] SW,
    output [7:0] LED
    );

      wire [7:0] y_out;

      dec_3to8_proc DEC1 (
            .w_in (SW[2:0]),
            .enable(SW[3]),
            .y_out (y_out)
      );

      assign LED = y_out;

endmodule
```

```
module dec_3to8_struc(
    input [2:0] w_in,
    input enable,
    output [7:0] y_out
    );

      assign y_out = (enable==1)?(8'b00000001<<w_in):8'bZ;

endmodule
```

```
module dec_3to8_struc_top(
    input BTN_NORTH,
    input BTN_EAST,
    input BTN_SOUTH,
    input BTN_WEST,
    input [3:0] SW,
    output [7:0] LED
    );

      wire [7:0] y_out;

      dec_3to8_struc DEC1 (
            .w_in (SW[2:0]),
            .enable(SW[3]),
            .y_out (y_out)
     );

      assign LED = y_out;

endmodule
```


4.3.2 DEMUX_8: demux_8_proc dan demux_8_struc

```
module demux_8_proc(
    input data_in,
    input [2:0] sel,
    output [7:0] data_out
    );

      reg [7:0] data_out_reg;

      always @(data_in, sel) begin
            data_out_reg = 8'b000000000; /*blocking assignment*/
            case (sel)
                  3'b000: data_out_reg[0] = data_in;
                  3'b001: data_out_reg[1] = data_in;
                  3'b010: data_out_reg[2] = data_in;
                  3'b011: data_out_reg[3] = data_in;
                  3'b100: data_out_reg[4] = data_in;
                  3'b101: data_out_reg[5] = data_in;
                  3'b110: data_out_reg[6] = data_in;
                  3'b111: data_out_reg[7] = data_in;
                  default: data_out_reg = 8'bZ;
            endcase
      end

      assign data_out = data_out_reg;

endmodule
```

```
module demux_8_proc_top(
    input BTN_NORTH,
    input BTN_EAST,
    input BTN_SOUTH,
    input BTN_WEST,
    input [3:0] SW,
    output [7:0] LED
    );

      wire [7:0] data_out;

      demux_8_proc DEMUX1 (
            .data_in (BTN_NORTH),
            .sel (SW[2:0]),
            .data_out (data_out)
      );

      assign LED = data_out;

endmodule
```

```
module demux_8_struc(
    input data_in,
    input [2:0] sel,
    output [7:0] data_out
    );

      assign data_out[7] = sel[2] & sel[1] & sel[0] & data_in;
      assign data_out[6] = sel[2] & sel[1] & ~sel[0] & data_in;
      assign data_out[5] = sel[2] & ~sel[1] & sel[0] & data_in;
      assign data_out[4] = sel[2] & ~sel[1] & ~sel[0] & data_in;
      assign data_out[3] = ~sel[2] & sel[1] & sel[0] & data_in;
      assign data_out[2] = ~sel[2] & sel[1] & ~sel[0] & data_in;
      assign data_out[1] = ~sel[2] & ~sel[1] & sel[0] & data_in;
      assign data_out[0] = ~sel[2] & ~sel[1] & ~sel[0] & data_in;

endmodule
```

```
module demux_8_struc_top(
    input BTN_NORTH,
    input BTN_EAST,
    input BTN_SOUTH,
    input BTN_WEST,
    input [3:0] SW,
    output [7:0] LED
    );

      wire [7:0] data_out;

      demux_8_struc DEMUX1 (
            .data_in (BTN_NORTH),
            .sel (SW[2:0]),
            .data_out (data_out)
      );

      assign LED = data_out;

endmodule
```

4.3.3 File rangkaian_kombinasional.ucf

```
# Koneksi tombol tekan
NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
# Koneksi SW
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
# Koneksi LED
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```