3. VHDL lengoaia

VHDL (Very High Speed Integrates Circuit Hardware Description Language) lengoaia sistema digitalak deskribatzeko lengoaia estandarra da. Atal honetan, deskripzioa garatzeko gidalerro batzuk aurkezten dira.

3.1. Deskribapena egiteko osagaiak

Zirkuitu edo azpizirkuitu bat VHDL lengoaia erabiliz desbribatzen denean, esaten da entitatearen diseinua (*design entity*) egin dela. Bi ataletan banatzen da diseinu hori: alde batetik, entitatearen deklarazioa, *entity.* Atal horretan, sarrera- eta irteera-seinaleak deklaratzen dira; izan ere, kanpoaldearekiko interfazea izango da. Bestalde, arkitektura dago, *architecture,* non zirkuituaren xehetasun guztiak adierazten diren, hau da, entitatearen funtzionamenduaren zehaztapenak.

Halaber, diseinurako elementuak (datu motak, eragileak, osagaiak, funtzioak,...) iturburu-liburutegietan, *libraries*, aurkitu daitezke. Iturburu-liburutegi horien elementuak, paketa bidez (*Packages*) antolatzen dira, eta erabiligarri izango dira deklaratu ostean.

Besterik adierazi ezean, *std* (estandarra) eta *work* (lana garatzekoa) iturburu-liburutegiak, ez dira deklaratu behar.

3.2. Diseinu unitateak eta sintaxia

Paketeen deklarazioa

Packagea sartzeko modua:

```
LIBRARY library_name;
USE library_name.package_name.all
```

Library_name, sisteman eskuragarri dagoen liburutegia, edo erabiltzaileak sortutako bat izan daiteke. Paketea deklaratzeko sintaxia hau da:

```
PACKAGE package_name IS
        [TYPE declarations]
        [SIGNAL declarations]
        [COMPONENT declarations]
END package_name;
```

Entity-aren deklarazioa

Entitatearen deklarazioan definitu behar diren ezaugarriak dira zirkuituaren sarrerak, irteerak, haien tamaina (0tik n-ra bitarteko bitak), modua (sarrera edo irteera den), eta mota (integer, bit).

`entity zirkuitu_a is`	Programaren goiburua
`port(`	Sarrera- eta irteera-portuak hitz horren ondoren adierazten dira.
	Sarrerak eta/edo irteerak deklaratzen dira sintaxi egokia erabiliz.
-- sarrerako portuak -- irteerako portuak -- I/O portuak -- buffers portuak	Gidoi batekin hasten diren lerroak, konpiladoreak ez ditu aintzat hartzen. Konpiladoreak ez du bereizten letra maiuskula eta minuskularen artean.
`);`	Sarrera- eta/edo irteera-portuen deklarazioa bukatu da; orduan, entitatea ere bai.
`end circuito_a;`	

Entitatearen deklarazioan agertzen den seinale bakoitza portu bati dagokio; era berean, portu bakoitza ikur eskematikoaren hankatxo bati (edo bat baino gehiagori) dagokio. Portu bakoitza informazio-objektua da; hortaz, edozein adierazpenetan erabil daiteke, eta baita portu bakoitzari balio berriak esleitu.

Portu bakoitzak izen bat behar du. Izena egokia izan dadin, portuaren izenaren ostean, eta bi puntuak jarri ondoren, *modua* adierazi behar da. Informazioak portuan zehar zer noranzko duen definitzeko erabiltzen da modua: *in (*irakurri*)*, *out* (idatzi), *buffer* eta *inou (*irakurri eta idatzi*)*. Ezer adierazten ez bada, modua *in* moduan hartuko da. Moduaz gainera, seinale batek har ditzakeen balioak adierazi behar dira *datu mota*-ren bitartez. Hauek izan daitezke:

a) Eskalarea: atxikitako balioa besterik ezin du izan.

b) Matriziala (*array*): anitz elementu ditu, eta guztiak datu mota berdina dira.

c) Erregistroa (*records*): anitz elementu ditu, baina horiek datu mota ezberdinetakoak izan daitezke.

Portuetan defini daitezkeen moduak:

- ***In*** Modua: portuari dagokion informazioa entitatean "sartzen" bada, orduan portua *in* modukoa izango da.
- ***Out*** Modua: portuari dagokion informazioa entitatearen kanporanzko noranzkoa hartzen badu, orduan portua *out* modukoa izango da. *Out* moduan deklaratzen denean portu bat, konpiladoreak irakurgaitz moduan ulertuko du; hortaz, ez du berrelikadurarik onartuko. Ezaugarri horrek, nahiz eta desabantaila izan, gailu programagarriaren baliabide gutxiago kontsumitzea dakar.
- ***Buffer*** Modua: barne-berrelikadurak garatzeko erabiltzen da. Portu mota hau *out*aren antzekoa da; izan ere, noranzko bakarrekoa da, eta berrelikadura onartzen du. Halaber, zuzeneko konexio bat egin nahi izanez gero, ezinestekoa da barne-seinale batera edo beste entitate baten *buffer* moduko portu batera konektatzea. Horren aplikazio arrunta kontadore baten irteera da; izan ere, hurrengo unean irteerak zer balio behar duen kalkulatzeko, kontagailuak jakin behar du oraingo unean zer balio duen irteerak.
- ***Inout*** Modua: bi noranzkoko seinaleak dira; hortaz, portutik informazioa atera edo sar daiteke. Barne-elikadura onartzen du, eta gainerako moduen ordez erabil daiteke.

Datu mota eskalarra:

- ***Boolean*** datu mota: bi balio ezberdin har ditzake: egia (true) edo faltsua (false)
- ***Bit*** datu mota: bi balio har ditzake: 0 edo 1.
- ***Integer*** datu mota: zenbaki bitarra adierazten du. Zenbaki lehenetsia 32 bitekoa da (negatiboak barne), baina tartea murriztu daiteke *Range* erabiliz, adibidez,

SIGNAL X: INTEGER RANGE –127 TO 127

Eragiketa matematikoak garatzeko erabiltzen bada, *std_logic_vector* ere erabil daiteke horrelakoetan.

- **Constant data object** datu mota: balioa aldaezina da, eta adierazteko modua hau da.

CONSTANT constant_name : type_name :=constant_value

- ***bit_vector*** datu mota: Bitez osaturiko bektorea da. Nahitaezkoa da bektorea osatzen duten biten maila zehaztea, hots, *downto* edo *to,* pisu handieneko bita zenbakirik handiena dela adierazteko, edo pisu txikieneko bita zenbakirik handiena dela adierazteko, hurrenez hurren.

`zenbakia: bit_vector (0 to 7);`	MSB zenbakia (0) da, eta LSB zenbakia(7) da
`zenbakia: bit_vector (7 downto 0);`	MSB zenbakia (7) da, eta LSB zenbakia(0) da

- ***std_logic*** datu mota*: bit* datu mota baino moldagarriagoa da balio gehiago har ditzakeelako: 0, 1, Z (inpedantzia altua), - (ez du axola), L, H, U, X eta W. Zirkuitu digitalak garatzeko, lehendabiziko laurak erabiltzen dira gehienbat.
- **std_logic_vector** datu mota. *std_logic*-ez osatutako *array*a da. Zirkuitu aritmetikoetan zenbaki bitarrekin erabil daiteke.

Azaldutako azken bi datu mota horiek erabili ahal izateko, IEEE liburutegia sartu behar da, eta baita hurrengo *package*-ak ere:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all
USE ieee.std_logic_signed.all
```

- Datu mota ***enumeratu***: erabiltzaileak definitutako datu mota da. Karaktereak erabiliz, edo komenentziako izenak erabiliz defini daiteke.

```
TYPE mota_izena IS mota_definizioa;
```

```
ARCHITECTURE rtl OF fsm IS
TYPE egoera IS (reset,libre,okupatuta);
SIGNAL aurrekoa, ondorengoa: egoera;
BEGIN
--arkitekturari dagozkion sententziak
....
END rtl;
```

Datu mota hori erabiltzeko, lehenago pakete batean deklaratu behar da, eta pakete horri dagokion liburutegia erabili.

Matrize datu mota (array)

Array-a datu mota berdineko elementuz osatuta dago, eta definitzeko modua hau da:

TYPE bit_vector IS ARRAY (tartea) OF bit;

TYPE std_logic_vector **IS ARRAY** (tartea) **OF** std_logic;

Erabiltzaileak edozein motatako arrayak defini ditzake. Adibidez:

TYPE bit **IS ARRAY** (7 DOWNTO 0) **OF** std_logic;

SIGNAL X: Byte;

Non X seinalea byte motakoa deklaratu den.

Arkitekturaren deklarazioa (*architecture*)

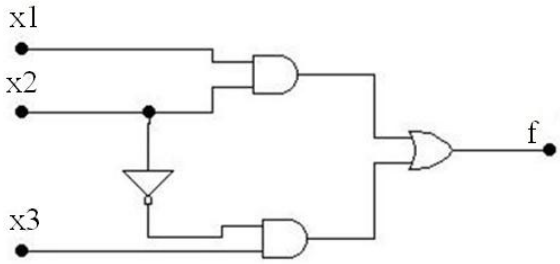
Entitate baten funtzionalitatea definitzen du, eta bi atal ditu: deklarazioari dagokiona (*declarative region*), eta arkitekturaren gorputzari (*architecture body*) dagokiona.

Deklaratu beharreko seinaleak, erabiltzaileak definitutako motak, konstanteak, atributuak, eta osagaiak deklarazioan sar daitezke; izan ere, erabiliko diren terminoak aurreratzen dira hor.

Funtzionalitatea arkitekturaren gorputzaren barnean eta BEGIN hitzaren ondoren zehazten da.

```
ARCHITECTURA architecture_name OF entity_name IS
        [SIGNAL declarations]
        [CONSTANT declarations]
        [TYPE declarations]
        [COMPONENT declarations]
        [ATTRIBUTE declarations]
BEGIN
        {COMPONENT instantiation statement;}
        {CONCURRENT ASSIGNMENT statement;}
        {PROCESS statement;}
        {GENERATE statement;}
END [architecture_name];
```

Adibidea: AND_OR Zirkuitua (78. Irudia)



78. Irudia

```
ENTITY adibide1 IS
        PORT (x1, x2, x3  : IN BIT;
                f             : OUT BIT);
END adibide1;
ARCHITECTURE LogicFunc OF adibide1 IS
BEGIN
        f <= (x1 AND x2) OR (NOT x2 AND x3);
END LogicFunc;
```

Adibidea: Full adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY fullsdd IS
        PORT (Cin, x, y : IN STD_LOGIC;
                s, Cout   : OUT STD_LOGIC);
END fulladd;
ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
        s <= x XOR y XOR Cin;
        Cout <= (x AND y) OR (x AND Cin) OR (y AND Cin);
END LogicFunc;
```

3.3. Eragileak

VHDL lengoaiak baditu funtzio bereziak garatzen dituzten ikurrak. Erabilienak dira:

"+" ikurra, batuketa eragiketa egiteko erabiltzen da.

"--" , deskripzioa ulergarri izan dadin erabiltzaileak oharrak sar ditzake ikur horren ondoren.

" ; ", kodearen sententzia bakoitza bukatutzat emateko erabiltzen da.

Halaber, beste ikur bereziak laburbildu dira 23. Taulan:

ERAGILE LOGIKOAK	NOT, AND, OR, NAND, NOR, XOR	Eragile mota: boolean Emaitza mota: boolean
HARREMAN-ERAGILEAK	= / < <= > >=	Eragile mota: edozein mota Emaitza mota: boolean
ERAGILE ARITMETIKOAK	+ - * / ** MOD, REM, ABS	Eragile mota: integer, erreal, signal Emaitza mota: integer, erreal,signal
KATEATZE-ERAGILEA	&	Eragile mota: array Emaitza mota: array

23. Taula

Adibidea: $f = \left(x_1 \overline{x}_6 + x_2 x_7\right)\left(x_3 + x_4 x_5\right)$

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adibide2 IS
        PORT (x1, x2, x3, x4, x5, x6, x7  : IN STD_LOGIC;
                f               : OUT STD_LOGIC);
END adibide2;

ARCHITECTURE LogicFunc OF adibide2 IS
BEGIN
        f <= (x1 AND x3 AND NOT x6) OR
                (x1 AND x4 AND x5 AND NOT x6) OR
                (x2 AND x3 AND x7) OR
                (x2 AND x4 AND x5 AND x7);
END LogicFunc;
```

3.4. Lengoaiaren gramatika

Deskripzio-lengoaietan programaren gauzatzea goitik behera egiten da, hau da, sententziak idatzi diren ordenan exekutatuko dira. Hortaz, sententzien ordena oso garrantzitsua da. Halaber, VHDL lengoaiak prozesu baten gorputzean zehar eta sekuentzialki esleitzen dituenez seinaleen balioak, han ere ordena kontuan hartu beharreko parametroa da. Izan ere, ordena hori konpilazioan isladatuko da.

Seinale bati esleipen bat egiteko modua: Prozesuaren seinale bati esleipena egiteko erabili behar den eragilea <= da. Seinale hori eragilearen ezkerraldean jartzen da, eta seinale horrek hartu behar duen balioa, berriz, eragilearen eskualdean.

signal **<=** signal1 + signal2;

Gogoratu behar da seinale baten balioa ez dela aldatuko seinale bera dagoen prozesua bukatu arte. Taula honetan adibide sinplea erakusten da.

```
process
begin
a <= b;
b <= a;
wait on a,b;
end process;
```

a seinaleak **b** seinalearen balioa izango du
b seinaleak **a** seinalearen balioa izango du

Aldaketak HEMEN egiten dira.

Aldagai bati esleipen bat egiteko modua: Seinaleekin gertatzen den ez bezala, aldagai bati balio bat esleitzen zaionean, momentuan hartzen du balio berria. Hortaz, balio berri hori hurrengo sententzietan aplikagarria izango da beste balio berri bat agertu arte.
Beraz, seinaleen ordez aldagaiak berridazten badizkiogu aurreko adibideari, programaren emaitza ezberdina izango da:

```
a := b;
b := a;
```

a-k b-ren balioa hartuko du
b-k a-ren balio berria hartuko du

IF Sententzia

Sententzia honen eragina hau da; baldintza bat betetzen denean, ekintza bat garatzen da, eta betetzen ez bada, beste ekintza bat garatzen da. Horri dagokion idazkera lerro hauetan azaltzen da:

```
if (baldintza) then
  ekintza garatu;
else
beste ekintza bat, ezberdina, garatu;
end if;
```

Halaber, baldintza ezberdinak kontuan hartu behar badira:

```
if (baldintza) then
 ekintza garatu;
elsif (beste baldintza) then
 beste ekintza bat, ezberdina, garatu;
else
 beste ekintza bat, ezberdina, garatu;
end if;
```

CASE Sententzia

Kasu honetan, baldintza ezberdinen aurrean ekintza ezberdinak garatzen dira, eta idazkera hau da:

```
case (ebaluatzeko seinalea) is
  when (balio 1) => ekintza garatu;
  when (balio 2) => beste ekintza bat garatu;
  ...
  when (azkenengo balioa) => beste ekintza bat garatu;
end case;
```

Adibidez:

```
case kontrol is          Kontrol-seinalea ebaluatzen da
  when "00" => d <= a;   kontrol = "00" bada, orduan d<=a
  when "01" => d <= b;   kontrol = "01" bada, orduan d<=b
  when "10" => d <= c;   kontrol = "10" bada, orduan d<=c
  when others => d <=    kontrolaren balioa aurretik
"1111";                  azaldutakoa bezalakoa ez bada,
                         orduan "1111" balioa izango du
end case;                arkitekturaren amaiera
```

LOOP Sententzia

Loop (zikloa) sententzia erabiltzen da, sententzia talde bat *n* aldiz errepikatzea beharrezkoa denean. Loop sententziak *for* edo *while* zikloak du barne.

```
process (a)
begin
ziklo1: for i in 7 downto 0
loop                          zikoaren burua
    sarrera(i) <= ( others => 8 aldiz errepikatu beharreko
'0' )                         ekintza
end loop;                     zikloaren bukaera
end process;
```

```
process (a)
aldagaia i: integer := 0;
begin
ziklo2: while i < 7 loop
   sarrera(i) <= (others =>   i 7 baino txikiagoa den
'0');                         bitartean => zikloa
   i := i + 1;
end loop;
                              zikloaren amaiera
end process;
```

EXIT Sententzia

Exit sententziak ahalbideratzen du loop zikloa buka dadin. Horretarako, ezinbestekoa da programatzaileak jarritako baldintzaren bat betetzea.

```
process (a)
begin
ziklo1: for i in 7 downto 0 loop
  if a'length < i then exit ziklo1;
  sarrera(i) <= ( others => '0' );
end loop;
end process;
```

NEXT Sententzia

Sententzia hau erabili ahal izateko, nahitaezkoa da *loop* ziklo baten barruan kokatzea eta programatutako instrukzio bat, edo gehiago, ez gauzatzea eragiten du.

```
process (a)
begin
ziklo1: for i in 7 downto 0
loop
    if i=4 then next;
    else
    sarrera(i) <= ( others =>
'0' );
   end if;
end loop;
end process;
```

```
zikoaren burua
i-ren balioa 4 bada, ez du
zikloa egingo.
4 ez bada,...
... sarrera hasieratzen da

zikloaren amaiera
```

NULL Sententzia

Egoera berezi baten aurrean ekintzarik gerta ez dadin erabiltzen da; hortaz, seinaleek eta aldagaiek ez dute balioa aldatuko, eta programak jarraituko du. Loop ziklo baten barruan *next* sententziak duen portaeraren antzekoa da.