Универзитет у Нишу
Електронски факултет
Катедра за рачунарство

Архитектура и организација рачунара
Вежбе, VHDL

Део 1

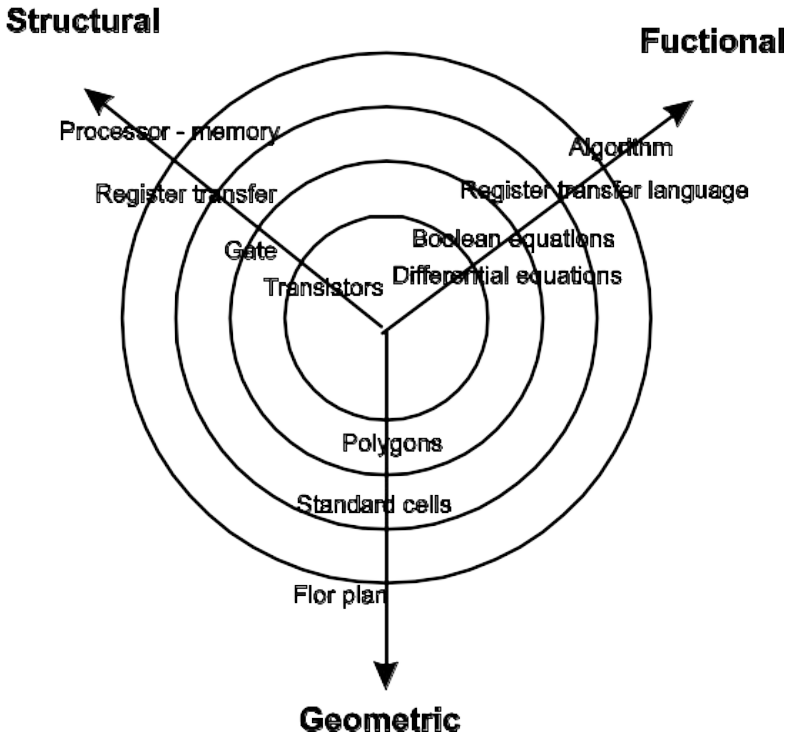
Увод

Овај материјал представља полазну основу за изучавање области VHDL у оквиру рачунских вежби на курсу Архитектура и организација рачунара.
Замишљено је да се студенти самостално упознају са изложеном материјом, проуче и пробају изложене примере пре часова вежби, а да на часовима вежби дискутују решења, затраже појашњења и додатна објашњења у вези са изложеним материјалом.
Материјал је настао као резултат напора да се у кратком времену припреми што комплетнији садржај за учење на даљину. Аутори су се трудили да објашњења буду што концизнија, а да покрију све предвиђене елементе. Материјал може да садржи ненамерне (и/или намерне !) грешке; на часовима ће ове појаве бити разматране. Потпун материјал за учење треба изградити од овог материјала и од личних бележака са часова и у току самосталног учења.
Додатна литература : Pedroni, Volnei A. *Circuit design with VHDL*. MIT press, 2004.

Циљ ове области у оквиру курса је да се студенти упознају са ”филозофијом” језика за опис хардвера (*Hardware Description Language* – HDL). *И*ако синтакса HDL подсећа на синтаксу програмских језика, ефекти ”извршавања” описа на HDL-у су потпуно другачији од ефеката извршавања програма написаних на програмским језицима.
Програмски језици за циљ имају компајлирање на машински језик и извршавање на процесорима, док описи на HDL-у имају за циљ симулацију и синтетисање хардверских компонената од клаузула HDL-а. Код програмских језика, свака наредба се преводи у низ машинских инструкција циљног процесора; код HDL описа, алати за синтезу препознају карактеристичне конструкције (величине једне или више клаузула) и замењују их одређеним елементима хардвера циљне технологије. На пример, за *if* клаузулу у HDL опису се генерише мултиплексер. Алати за ситезу не могу да препознају све синтаксно исправне описе на HDL-у, и зато се говори о *синтетизабилном подскупу језика*, у који спадају одређени типови клаузула, али и само одређене конструкције написане помоћу иначе синтетизабилних клаузула. Другим речима, креативност описивања на HDL-у је ограничена само на оне конструкције које подржавају алати за синтезу. Ова чињеница отежава учење HDL-ова и треба је имати у виду, иако се у току овог курса нећемо стриктно и експлицитно држати само синтетизабилног подскупа језика.
Друга кључна разлика у односу на програмске језике лежи у томе да процесори извршавају програме секвенцијално (инструкцију по инструкцију), док код хардвера (било ког хардвера, па и оног синтетисаног на основу HDL) све компоненте раде истовремено. То за последицу има да при пројектовању хардвера помоћу HDL-а треба размишљати на ортогоналан начин у односу на размишљање при програмирању на програмским језицима.

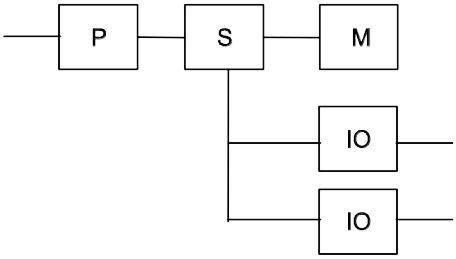
Моделовање хардвера

За моделовање хардвера се примењују три глобално различите парадигме: структурална, функционална и геометријска, и у свакој од њих, могу се креирати модели на више различитих нивоа апстракције (слика 1).



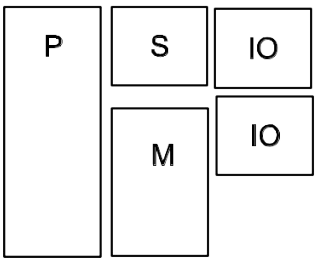
Слика 1. Парадигме и нивои апстракције за моделовање хардвера

Функционални приступ подразумева да се описује понашање кола.
Пример функционалног моделовања, на псеудојезику и алгоритамском нивоу апстракције:
loop // бесконачна петља
for *za svaki ulaz* loop
očitati vrednost ulaza;
obraditi podatak sa ulaza;
poslati rezultat obrade na izlazne pinove;
end loop;
wait for 10ms; // периода такта
end loop
Код структуралног приступа, описује се структура кола од делова и њихових међусобних веза (слика 2).



Слика 2. Пример структуралног модела

По геометријском приступу, одређује се геометријски распоред појединих елемената кола на површини хардвера (штампаној плочици, силицијумском wafer-у…, слика 3).



Слика 3. Пример геометријског модела

Помоћу језика за опис хардвера могу се креирати функционални и структурални модели.

Нивои апстракције

У дизајну хардверских система, нивои апстракције представљају различите погледе на модел система који се разликују по нивоу детаљности, начинима како се систем представља, али и по намени специфичног описа. Најчешће разматрани нивои апстракције (од вишег ка нижем степену): модел понашања (бихејвиорални модел), *register-transfer level* (RTL), модел тока података (*dataflow*) и модел на нивоу гејтова (*gate-level*).

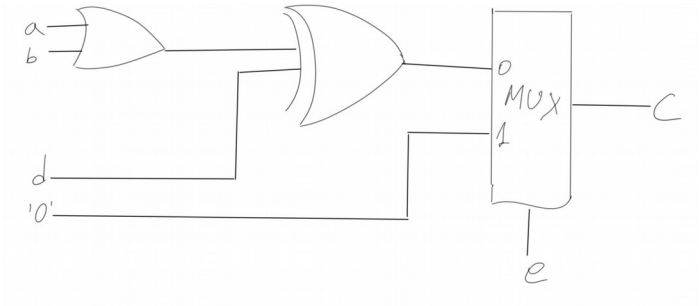
На бихејвиоралном нивоу, описују се понашање и интефејс система. Користи се функционална парадигма, као у примеру датом раније у овом документу.

RTL ниво апстракције подразумева опис система као низа меморијских елемената (регистара) које повезују трансфер функције (које се имплементирају комбинационим мрежама). Иако је опис дат функционалном парадигмом, јасно се може идентификовати структура кола потребних за имплементацију датог описа. Пример:

```
c = (a or b) xor d when e = '0' else '0'
```

Једноставном анализом се може схватити да је овде потребно једно AND, једно XOR коло и један мултиплексер.

При моделовању на нивоу гејтова, представљају се појединачна логичка кола (гејтови) и начин њиховог повезивања. На пример, за функционалност представлљену током података у претходном примеру, може се креирати шема као на слици 4. Подсећања ради, на курсу Дигитална електроника се већина кола представља овим начином моделовања.



Слика 4. Пример gate-level модела

Могу се симулирати модели представљени на било ком нивоу апстракције, али нису сви описи синтетизабилни. Типично се при елаборацији кола креће са виших нивоа апстракције, на којима је лакше развити модел, а циљ је елаборирати даље делове модела нижим нивоима апстракције док се не дође до синтетизабилног описа. У основи, алати за синтезу препознају и успешно обрађују RTL, *dataflow*, i naravno *gate level* опис. Од бихејвиоралног описа, постоји синтетизабилни подскуп: како елемената језика, тако и

конструкција. Алати препознају шаблоне (крупније од једне наредбе у програмском језику) и шаблоне замењују хардверским елементима (нпр. одређена структура if блока се замењује мултиплексером)

VHDL

VHDL је један од најкоришћенијих језика за опис хардвера. О историјату и особинама може се укратко упознати на Википедији.

У оквиру овог курса, обрадићемо само подскуп VHDL-а. При анализи (на лабораторијским вежбама), задржаћемо се на логичкој (*behavioral, HDL*) simulaciji. Користићемо идетичан пакет алата који је рокишћен и на курсу Дигитална електроника, са једином разликом да ће се описи компонената задавати VHDL описом, уместо шемом. Кратко упутство за коришћење алата је дато на страници курса, у документу AOR_lab2-4-uputstvo.pdf.

Основна структура VHDL дизајна

VHDL дизајн једне хардверске компоненте (кола) се састоји од два основна елемента:
а. *entity* statement, који описује спољњи изглед (интерфејс) дизајна;
б. *architecture* statement, који описује архитектуру (функционални опис или структуру) дизајна.
У наредном примеру моделовано је двоулазно коло за конјункцију. Искористићемо га да укратко уведемо основне појмове језика, при чему ће поједини елементи језика који су коришћени у примеру бити разјашњавани касније у материјалу.

Z ПРИМЕР 1, двоулазно И коло

```
01  entity e_and2 is             -- VHDL je case insensitive!!!
02    port (in1,in2 : in bit; -- <име(на) порт(ова)> : <мод> <тип>
03          out1 : out bit); -- постоји и мод inout. nema ";" iza poslednje deklaracije porta
04    -- + generic декларације, декларација типова и константи. Касније...
05  end entity e_and2;
06
07  architecture a1_and2 of e_and2 is
08     -- deklaracije  signala. Primer: SIGNAL a:bit;
09  begin
10     out1 <= in1 and in2;  -- тзв. конкурентна клаузула доделе вредности сигналу
11  end architecture a1_and2;
12
13  architecture a2_and2 of e_and2 is
14  begin
15    process (in1,in2) -- ili, od VHDL2008: process (all). Објашњење процеса касније
16      -- + deklaracije promenljivih
17    begin
18      out1 <= in1 and in2;
```

```
19    end process;
20  end architecture a2_and2;
```

За један *entity* може да се дефинише више различитих архитектура, свака добије своје име (л. 07 и 13). При коришћењу моделоване компоненте (при референцирању на њу у другим компонентама), референцира се преко имена архитектруре и имена ентитета коме припада.
Основни преносиоци података кроз дизајн су ***сигнали***. Сигнали се могу сматрати водовима (жицама, проводницима). Треба имати у виду да концепт сигнала није исто што и концепт променљиве у програмским језицима (детаљи касније). Сигнали се могу дефинисати као портови компоненте (линије 02 и 03) или као интерни сигнали (л. 08). Оператор доделе вредности сигналу је <=
За интерне сигнале се дефинише тип (у примеру у л. 08 сигнал a је типа bit), а за сигнале који су портови се, поред типа, дефинише и *мод порта* (смер).

Tipovi

VHDL је јако типизиран језик. То, између осталог, значи да се не ради имплицитна конверзија типа између у суштини истих, али различито названих типова (конверзија између типа и његових типова се, са друге стране, спроводи).
Основни, уграђени типови су: `bit` за једнобитне и `bit_vector` за вишебитне сигнале. Литерали (константе) за ове типове су '0' и '1' (за једнобитне) и "0000", "011001" и сл. (за вишебитне сигнале). Ове константе се не могу поистовећивати са нумеричким вредностима 0, 1, бинарним бројевима и сл.

ŠTO ZNAČI?
Zato što je tipiziran, ili šta?

На слици 5 су дати неки од често коришћених типова. Типови за које за пакет стоји "стандард" су уграђени и могу се користити без посебних декларација. Остали типови се могу користити након декларације библиотеке и пакета у коме су дефинисани:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

или

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

KOJA JE RAZLIKA?

Типови `std_logic` и `std_logic_vector` су стандардизовани, синтетизабилни типови који представљају проширење типова `bit` и `bit_vector` вредностима које верније моделују понашање дигиталних кола у реалности. Осим високог и ниског нивоа сигнала ('1' и '0'), постоје и вредности које означавају да сигналу није постављена вредност ('U'), да на линији постоји конфликт ('X', више извора сигнала постоји, извори у исто време покушавају да поставе различите вредности), да је линија у стању високе импедансе ('Z'), и тд.
Низови битова се могу интерпретирати као бројеви (типови `signed`, `unsigned`) и повезивање сигнала ових типова операторима је синтетизабилно. (На пример, за израз `a+b` ће се генерисани *carry look-ahead* сабирач одговарајуће ширине.) И типови `integer` и `real` су синтетизабилни, али нпр. тип `time` није.
Поред уграђених, могу се креирати и кориснички типови, типови набрајања, подтипови и сл. О свему овоме ће се више говорити у наставку, када се укаже потреба.

Type	Values	Package
std_logic	'0', '1', 'Z', 'U', 'H', 'L', 'W', 'X', '-'	std_logic_1164
std_logic_vector	Array of std_logic	std_logic_1164
unsigned	Array of std_logic	numeric_std
signed	Array of std_logic	numeric_std
integer	$-(2^{31}-1)$ to $(2^{31}-1)$	standard
real	−1.0E38 to 1.0E38	standard
time	1 fs to 1 h	standard
Boolean	true, false	standard
character	256 characters as defined in	standard
String	Array of characters	standard

Слика 5. Често коришћени типови

Како јеVHDL јако типизиран, неподударање типова података у изразима и доделама ће резултовати грешкама приликом процеса анализе. Чак и када желимо просту конверзију типова података (нпр. unsigned_int у int) морамо извести **експлицитну конверзију типа**. Конверзија се може извршити на два начина, у зависности од тога колико су типови података блиски. Ако су блиски типови података (имају исти скуп вредности), онда конверзија изгледа: **ime_željenog_tipa(izraz_u_originalnom_tipu)**. Ако типови нису блиски, мора да се користи функција конверзије која дефинише како ће се вредност једног типа превести у други (неке функције за конверзију ће бити дате касније).

Опис функционисања

Функционисања кола се унутар архитектуре може описивати на неколико начина (који се могу слободно комбиновати, чак и унутар једне архитектуре):

- *Dataflow* стил – тзв. конкурентне клаузуле доделе вредности сигналу
- Бихејвиорални опис – секвенца програмских конструкција сличних онима у програмским језицима (доделе, петље, гранања), унутар `process` клаузуле
- Структурални опис – коришћење инстанци других модула/компонената који ће радити делове посла, међусобно повезивање њихових портова. Овај опис је у ствари вербализација (представљање текстуалним језиком) шеме кола (каква је нпр. креирана у курсу Дигитална електроника).

Клаузуле (*statements*) у VHDL

- **Конкурентне клаузуле** – извршавају се паралелно, истовремено. Могу се јавити само унутар архитектуре. У конкурентне клаузуле спадају:
  - процеси,
  - конкурентне клаузуле доделе вредности сигналу,
  - инстанце компонената, generate клаузуле... (И још три врсте клаузула које се неће обрађивати на овом курсу.)

- **Секвенцијалне клаузуле** – извршавају се редом како су наведене. Могу се јавити само унутар процеса (или процедура). Секвенцијалне клаузуле које ће се обрађивати у овом курсу су:
    - додела вредности сигналима и променљивама,
    - клаузуле контроле тока (if, case, loop, next, exit, for, while),
    - позив процедуре,
    - *wait* клаузула,
    - ...

Конкурентна клаузула доделе вредности сигналу

Позиција: унутар тела архитектуре, ван процеса.
У симулацији, све ове клаузуле се извршавају непрестано и симултано, без обзира на редослед којим су написане. Ово је концепт различит у односу на програме на програмским језицима!
Три врсте:

- једноставна додела
- условна додела
- селективна додела

Једноставна додела

Код **једноставне доделе** (пример 1 л. 10), сигнал са леве стране оператора доделе добија вредност једнаку резултату израза написаном са десне стране. У изразу могу учествовати сви оператори дефинисани за дате типове сигнала (понекад се за примену оператора морају укључити специфични пакети). Као ефекат, генерише се комбинационо коло.

Условна додела

Условна додела омогућава избор и доделу једне између више понуђених вредности, зависно од тога који је услов испуњен:

```
1   <target signal> <=
2       <izraz 1> when <uslov 1> else
3       <izraz 2> when <uslov 2> else
            ...
4       [else <podrazumevana vrednost>];
```

(Ова клаузула се донекле може посматрати као низ угњеждених if-elseif-else структура у програмским језицима.)

Z ПРИМЕР 2, компаратор означених бројева

Уводе се: условна конкурентна клаузула доделе вредности сигналу, вишебитни сигнали, релационе операције над вишебитним подацима.

Компаратор добија два вишебитна податка на улазу које треба сматрати означеним целим бројевима. Треба да генерише 1 на једном од три излаза, зависно од релационог односа улаза (веће, једнако или мање), а на остала два излаза треба да буде 0. Коло не треба да има такт.

```
01  LIBRARY ieee;
02   USE ieee.std_logic_1164.ALL; -- za std_logic
03   USE ieee.std_logic_arith.ALL; -- za relacione operacije nad signed
04   ---------------------------------------
05   ENTITY komparator IS
06         GENERIC (n: INTEGER := 7);
07         PORT (a, b: IN SIGNED (n DOWNTO 0);-- za višebitne tipove se
navodi broj i redosled bitova
08                     x1, x2, x3: OUT STD_LOGIC);
09   END komparator;
10   ---------------------------------------
11   ARCHITECTURE signed OF komparator IS
12   BEGIN
13          x1 <= '1' WHEN a > b ELSE '0';
14          x2 <= '1' WHEN a = b ELSE '0';
15          x3 <= '1' WHEN a < b ELSE '0';
16   END signed;
```

KAKO?

л. 06: Generic константе су у ентитету константе, али се вредност може предефинисати при коришћењу компоненте у другим компонентама. Користе се да би се постигла флексибилност. У овом случају, описан је компаратор са произвољним бројем битова на улазу; при коришћењу кола ће се предефинисањем константе `n` одредити број битова.

л. 07: Вишебитни сигнали се декларишу навођењем броја и редоследа битова. синтакса `(n DOWNTO 0)` одређује да је бит највеће тежине лево. Насупрот овоме, би `(0 TO n)` би значило да је бит највеће тежине десно.

Тип `SIGNED` подразумева податке у потпуном комплементу. Зато ће, за n=8, biti 127>0, ali 128<0 i 255<0 (127 je decimalno 127, ali 128 je decimalno -128, a 255 je decimalno -1).

Да би коло радило са неозначеним бројевима, требало би линију `07` заменити са:
`PORT (a, b: IN UNSIGNED (n DOWNTO 0);`
У том случају би било 128>127.

У примеру 2, л. `13`, `x1` добија вредност `'1'` ili `'0'`, зависно од тога да ли је испуњен услов `a>b`. Може се закључити да ово производи мултиплексер на чији се контролни улаз доводи резултат компаратора `a>b`, а на улазе података 0 и 1.

Условна додела се може завршити са `ELSE` (као у примеру о коме се говори у претходном пасусу), и тада за ефекат има генерисање чисто комбинационих мрежа. У случају да се клаузула завршава са `WHEN`, генеришу се секвенцијалне мреже. На пример: `Q<= '0' WHEN reset='1' ELSE D WHEN enable='1'`, генерисаће меморисјки елемент (леч), пошто нема `ELSE` на крају, нису описани сви могући случајеви вредности `reset` и `enable`, па ће се вредности задржавати до настанка неког другог од описаних случајева.

Ово је типичан пример који илуструје како ситне разлике у VHDL опису производе врло различите имплементације, што чини оптимално пројектовање коришћењем HDL врло сложеним задатком. Премда су сви овакви случајеви експлицитно наведени у стандарду, број случајева је јако велик и ван опсега овог курса.

Селективна додела

Селективна додела омогућава доделу једне од више вредности, зависно од вредности израза који представља услов:

```
1   with <izraz> select <target signal> <=
2       <vrednost 1> when <izbor 1>,
3       <vrednost 2> when <izbor 2> | <izbor 3>, -- "ili"
4       <vrednost 3> when <izbor 2> to <izbor 3>, -- opseg, za uređene
tipove
            ...
5       <podrazumevana vrednost> when others;
```

kao na primer?

Испитује се вредност израза `<izraz>` и сигналу `<target signal>` се додељује вредност `<vrednost i>` која стоји уз једну од предвиђених вредности израза `<izbor i>`. Изборима се морају покрити све могуће вредности израза, или да се на крају наведе `when others` (л. 5). Избори се не смеју преклапати.
(Ова клаузула се може разумети као switch структура у програмским језицима.)
Више детаља на линку: https://insights.sigasi.com/tech/signal-assignments-vhdl-withselect-whenelse-and-case/ .

Агрегација

Користан механизам који повећава могућности конкурентних клаузула јесте агрегација.

Агрегат је структура која представља комбинацију одвојених вредности које су постављене у неки композитни објекат (нпр. поље или рекорд). Агрегати спадају у композитне типове података, што значи да се могу састојати од више типова података. Овако дефиниција се може учинити гломазном због појмова у њој, због тога дајемо изглед агрегата као структуре када је агрегат поље: (izraz_1, izraz_2 ...) или: (mesto_u_polju => izraz, mesto_u_polju => izraz, ... ***others*** => izraz).

???????

По овоме, може се видети да агрегат агрегира вредности у низ. Додатно , агрегати могу представљати и рекорде, где се мора дефинисати рекорд као тип података. Агрегација се може применити са обе стране у клаузулама доделе вредности, као што ће бити показано у следећем примеру.

Пример 3 илуструје коришћење селективне доделе, заједно са агрегацијом и експлицитном конверзијом типова.

Z ПРИМЕР 3, једнобитни потпуни сабирач

Уводе се: селективна конкурентна клаузула доделе, експлицитна конверзија типа, агрегација.

```
01 ENTITY full_adder IS
02       PORT (a, b, c_in: IN BIT; s, c_out: OUT BIT);
03 END ENTITY full_adder;
04 ---
05 ARCHITECTURE truth_table OF full_adder IS
06 BEGIN
07       WITH BIT_VECTOR'(a, b, c_in) SELECT
08              (c_out, s) <=    BIT_VECTOR'("00") WHEN "000",
09                               BIT_VECTOR'("01") WHEN "001",
10                               BIT_VECTOR'("01") WHEN "010",
11                               BIT_VECTOR'("10") WHEN "011",
12                               BIT_VECTOR'("01") WHEN "100",
13                               BIT_VECTOR'("10") WHEN "101",
14                               BIT_VECTOR'("10") WHEN "110",
15                               BIT_VECTOR'("11") WHEN "111";
16 END ARCHITECTURE truth_table;
```

zašto (') ?

Ентитет full_adder представља потпуни сабирач. Функционисање потпуног сабирача би требало да буде познато, а детаљи се могу видети и на:
https://www.geeksforgeeks.org/full-adder-in-digital-logic/ .
У Archtecture delu, сабирач је описан табелом истинитости.

л. 07: сви улазни портови су агрегирани у један податак типа BIT_VECTOR. Оваква агрегација је била могућа зато што су сви сигнали били типа BIT, па нису биле потребне додатне функције за конверзију типа.

л. 08: излазни портови су агрегирани заједно (агрегација је могућа и с леве стране!). Тиме може да се искористи једна селекциона клаузула за доделу вредности на више сигнала истовремено.