



دانشگاه اصفهان
دانشکده مهندسی کامپیوتر

درس هم طراحی سخت افزار نرم افزار

راهنمای نصب برنامه gezel

به همراه چند مثال

خرداد ۱۴۰۱

فهرست

۳	نصب یک سیستم عامل debian based مانند اوبونتو:
۴	روش های نصب gezel:
۴	روش اول) نصب gezel با bash کد:
۵	روش دوم) نصب gezel به صورت عادی:
۶	جدول راهنما:
۷	مثال ها:
۷	مثال ۰) چاپ یک پیام ساده
۸	مثال ۱) طراحی یک گیت and با ۴ ورودی (با استفاده از and ۲ ورودی)
۱۰	مثال ۲) UpDown counter
۱۳	مثال ۳) sequence

نصب یک سیستم عامل debian based مانند اوبونتو:

ابتدا یک سیستم عامل debian based مانند اوبونتو دانلود کنید. مراحل نصب برای همه اوبونتو ها یکی است.

یکی از ورژن هایی که عمل می کند، اوبونتو دسکتاپ ۳۲ بیتی (ورژن ۱۷،۰۴ - حجم فایل حدود ۱۵۰۰ مگ) است که

از طریق سایت روبهرو دانلود شده است. <https://p30download.ir/fa/entry/35912/>

سپس VMWare را باز کنید و مراحل زیر را طی کنید:

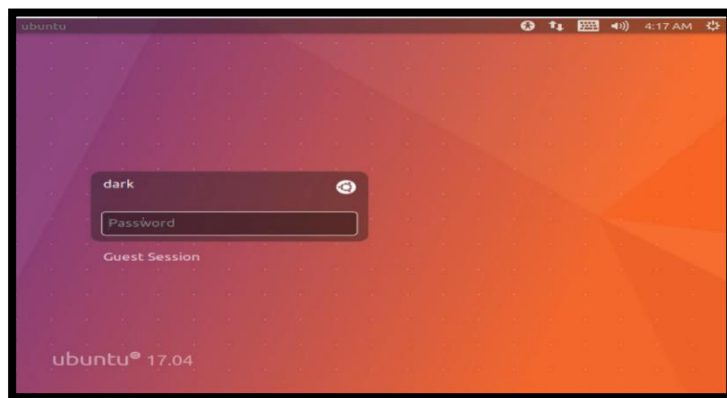
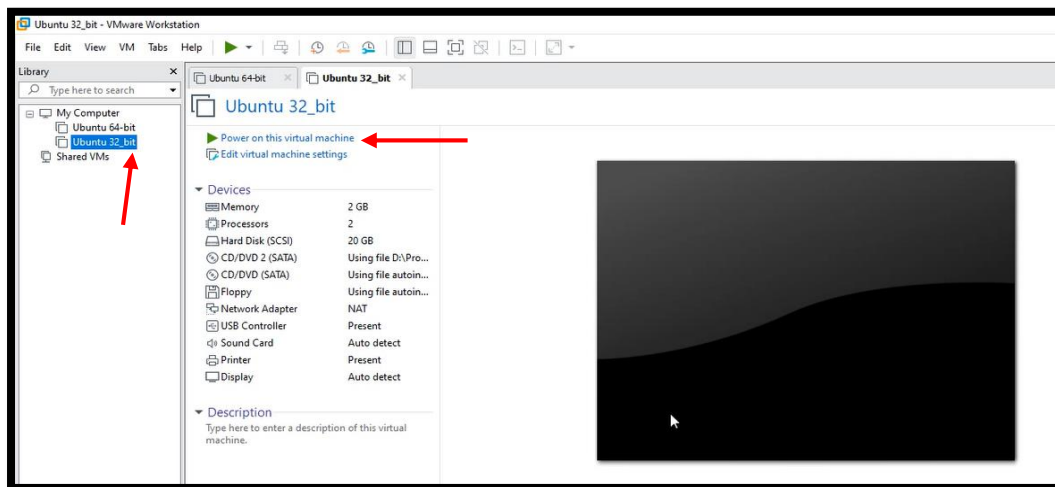
File >> New virtual machine >> Typical >> Next >>

Next >> Browse: محل قرارگیری فایل دانلود شده >> Next >>

Next >> به سلیقه خودتان پر کنید: Full name – user name – password >> Next >>

Next >> Finish Virtual machine name: به سلیقه خودتان پر کنید >> Next >>

سپس گزینه Power on this virtual machine را اجرا کنید.



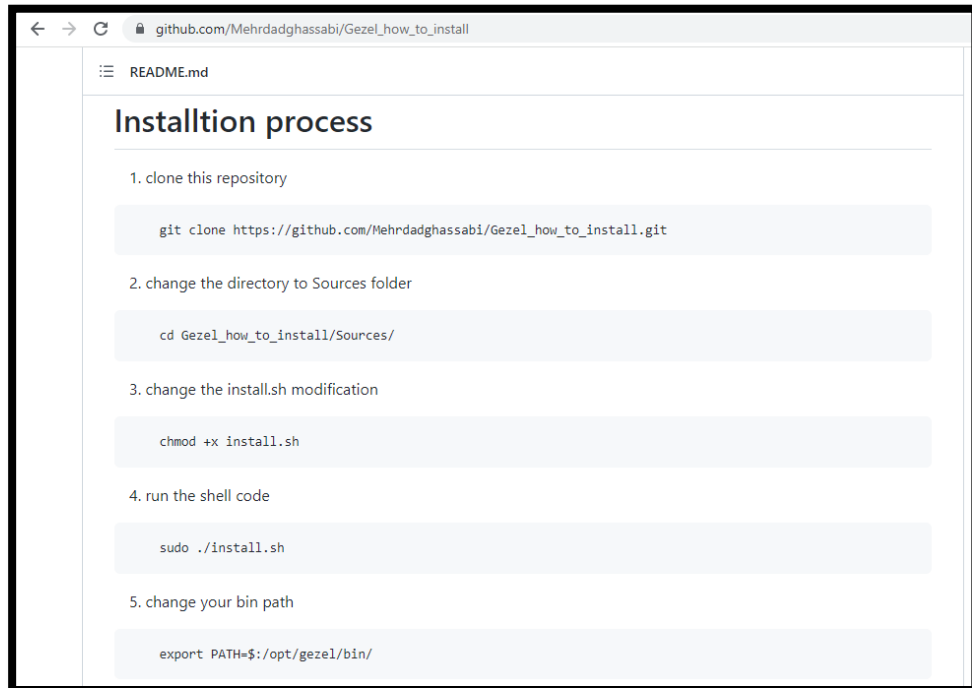
صبر کنید تا به صفحه ورود (login) برسید.

روش های نصب gezel:

روش اول) نصب gezel با bash کد:

ابتدا وارد لینک زیر شوید و مراحل را طبق ترتیب اجرا کنید.

https://github.com/Mehrdadghassabi/Gezel_how_to_install



اگر موفق به اجرای دستور git clone نشدید، ابتدا با دستور زیر آن را نصب کنید:

```
sudo apt install git
```

اگر باز هم نتوانستید، ممکن است ورژن نصب شده پایین باشد و این دستور را اجرا نکند. در چنین حالتی از روش دوم استفاده کنید.

روش دوم) نصب gezel به صورت عادی:

می توانید فایل gezel.zip را از طریق لینک زیر دانلود کنید:

<https://drive.google.com/file/d/111aEM3TZpM7EJirjMMQhBgeixBKLmnSa/view?usp=sharing>

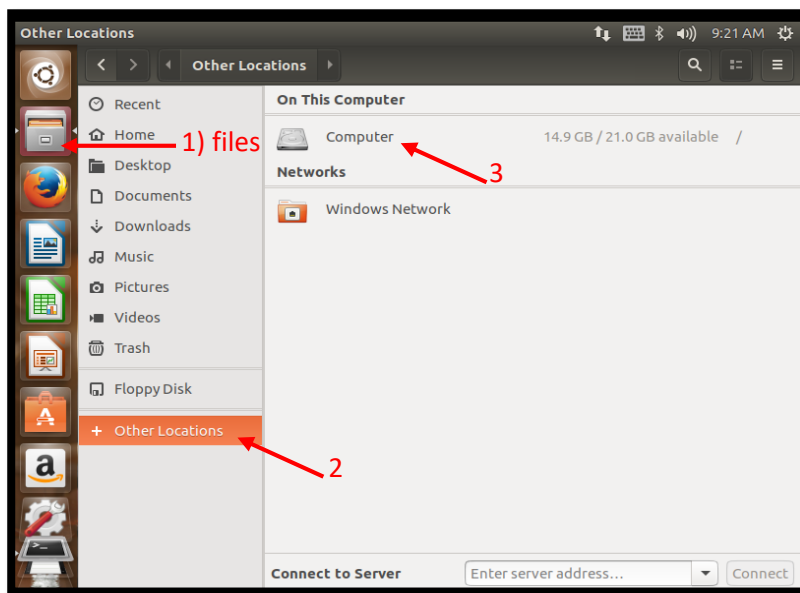
ابتدا فایل های پوشه gezel.zip را به اوبونتو منتقل کنید. روش های زیادی وجود دارد که به دو راه اشاره شده است:

(۱) یک پوشه در هرجایی از سیستم اصلی (Host) که مدنظرتان است ایجاد کنید. داخل آن فایل های gezel.zip را قرار داده و Extract کنید. سپس به VMWare بازگردید.

VM >> Settings >> Options >> Shared Folders >> Always Enabled >> Add

پوشه ایجاد شده را انتخاب کنید و در نهایت Ok را کلیک کنید.

برروی آیکون Files کلیک کنید. از بخش Other Locations گزینه Computer را دبل کلیک کنید.



پوشه اشتراکی بین سیستم و اوبونتو >> Folder hgfs >> Folder mnt

اگر پوشه وجود نداشت، از راه دوم اقدام کنید.

(۲) فایل زیپ را به ایمیل خود بفرستید و سپس ایمیل خود را روی سیستم اوبونتو باز کنید و فایل ها را دانلود

کنید. فولدر Extract شده را روی صفحه Home کپی کنید.

دستورات زیر را در ترمینال اوبونتو به ترتیب اجرا کنید: (توجه: نام فولدر Extract شده gezel است و username

انتخاب شده dark است)

```

sudo apt-get update
cd /home/dark/gezel/
sudo dpkg -i libgmp3c2_4.3.2+dfsg-2ubuntu1_i386.deb
sudo dpkg -i gezel-base-2.5.deb
sudo dpkg -i gezel-simulavr-2.5.deb
sudo dpkg -i gezel-simitarm-2.5.deb
sudo dpkg -i gezel-sources-2.5.deb
sudo dpkg -i gezel-debug-2.5.deb
sudo dpkg -i gezel-examples-2.5.deb
export PATH=$PATH:/opt/gezel/bin/
fdlsim Hello_World.fdl

```

دستور fdlsim بایستی اجرا شود و یک سری اطلاعات را نمایش دهد.

در صورتی که ترمینال را باز کردید و دستور را شناخت، ابتدا دستور `cd /home/dark/gezel/` و سپس دستور `export Path` را دوباره تکرار کنید. برای اجرای هر فایل fdl باید حتما وارد پوشه ذخیره شده شوید.

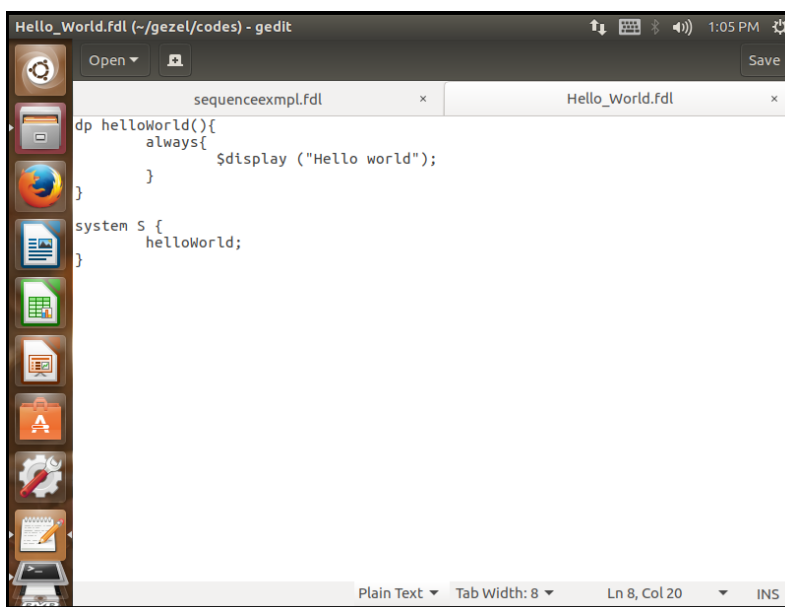
جدول راهنما:

نماد	مخفف
dp	Data path
sfg	Signal flow graph
fsm	Finite state machine
use	Instantiate another module
lookup	Lookup table (LUT)
sig	signal
reg	Register

مثال ها:

مثال ۰) چاپ یک پیام ساده

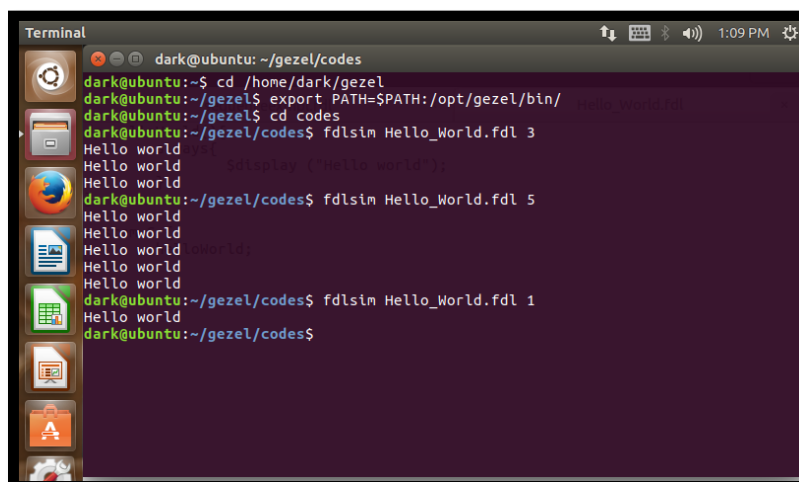
فرض کنید می خواهیم یک پیام ساده “Hello world” چاپ کنیم. از آنجایی که ورودی نداریم، پس test bench هم نداریم و می توانیم با یک دستور \$display پیام را چاپ کنیم.



```
Hello_World.fdl (~/.gezeli/codes) - gedit
sequenceexmpl.fdl x Hello_World.fdl x
dp helloWorld(){
  always{
    $display ("Hello world");
  }
}
system s {
  helloWorld;
}
```

به محیط ترمینال باز می گردیم.

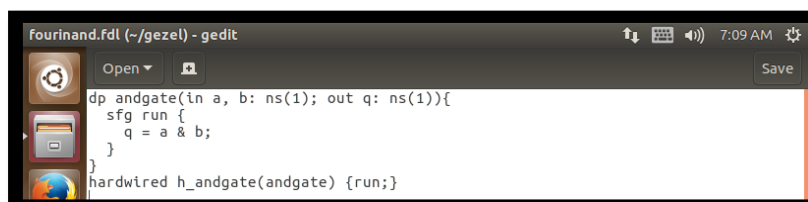
نتیجه نهایی برای دوره/سیکل های ۳، ۵ و ۱ قابل مشاهده است.



```
Terminal
dark@ubuntu: ~/.gezeli/codes
dark@ubuntu:~$ cd /home/dark/gezel
dark@ubuntu:~/gezel$ export PATH=$PATH:/opt/gezel/bin/
dark@ubuntu:~/gezel$ cd codes
dark@ubuntu:~/gezeli/codes$ fdisim Hello_World.fdl 3
Hello world
Hello world
Hello world
dark@ubuntu:~/gezeli/codes$ fdisim Hello_World.fdl 5
Hello world
Hello world
Hello world
Hello world
Hello world
dark@ubuntu:~/gezeli/codes$ fdisim Hello_World.fdl 1
Hello world
dark@ubuntu:~/gezeli/codes$
```

مثال ۱) طراحی یک گیت and با ۴ ورودی (با استفاده از and ۲ ورودی)

ابتدا باید گیت and با ۲ ورودی را طراحی کنیم:



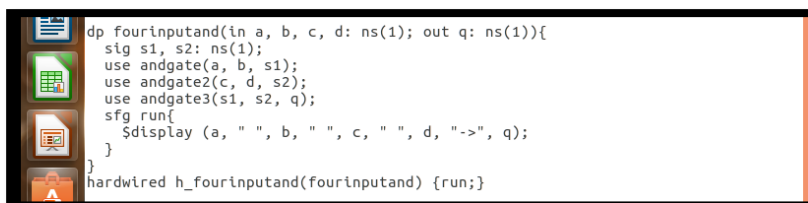
```
Fourinand.fdl (~/.gezel) - gedit
Open Save
dp andgate(in a, b: ns(1); out q: ns(1)){
  sfg run {
    q = a & b;
  }
}
hardwired h_andgate(andgate) {run;}
```

سپس از آنجایی که می خواهیم یک ماژول گیت and با ۴ ورودی طراحی کنیم، به ۳ ماژول گیت and با ۲ ورودی نیاز داریم. پس باید ۲ ماژول دیگر اضافه کنیم:



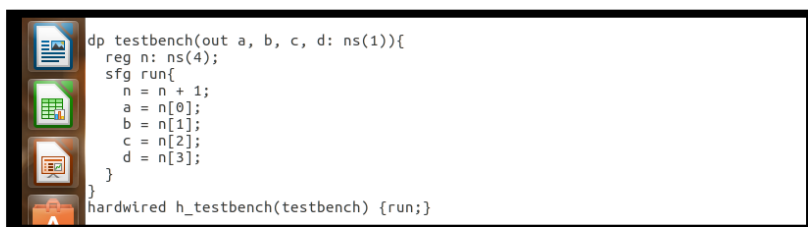
```
dp andgate2: andgate
dp andgate3: andgate
```

سپس ماژول گیت and با ۴ ورودی را تعریف می کنیم:



```
dp fourinputand(in a, b, c, d: ns(1); out q: ns(1)){
  sig s1, s2: ns(1);
  use andgate(a, b, s1);
  use andgate2(c, d, s2);
  use andgate3(s1, s2, q);
  sfg run{
    $display (a, " ", b, " ", c, " ", d, "->", q);
  }
}
hardwired h_fourinputand(fourinputand) {run;}
```

حال که تعریف کد تمام شده است، زمان تعریف testbench می باشد.



```
dp testbench(out a, b, c, d: ns(1)){
  reg n: ns(4);
  sfg run{
    n = n + 1;
    a = n[0];
    b = n[1];
    c = n[2];
    d = n[3];
  }
}
hardwired h_testbench(testbench) {run;}
```

Testbench را به این گونه نوشته ایم که یک عدد به نام n داریم. n هم یک عدد سخت افزاری است و هم نرم افزاری.

به این گونه که هردفعه n را بعلاوه یک می کنیم و بیت های آن را به متغیرها منتقل می کنیم.

در نهایت ماژول نهایی برای اجرا را می نویسیم:

```
dp sysandgate{
  sig a, b, c, d, q: ns(1);
  use testbench(a, b, c, d);
  use fourinputand(a, b, c, d, q);
}

system S {
  sysandgate;
}
```

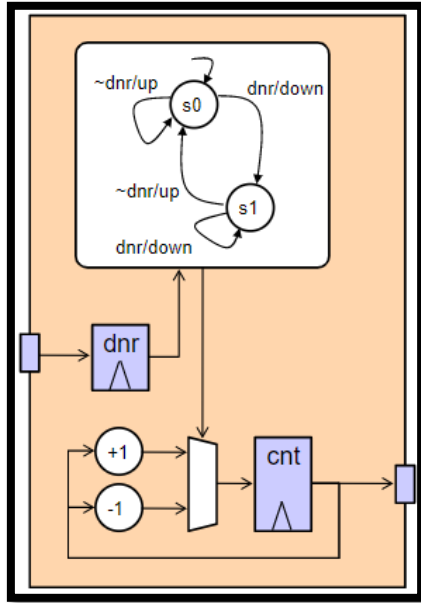
فایل نوشته شده را با نام fourinand.fdl ذخیره می کنیم و به محیط ترمینال برمی گردیم.

```
Terminal
dark@ubuntu: ~/gezel
dark@ubuntu:~$ cd /home/dark/gezel/
dark@ubuntu:~/gezel$ fdlsim
fdlsim: command not found
dark@ubuntu:~/gezel$ export PATH=$PATH:/opt/gezel/bin/
dark@ubuntu:~/gezel$ fdlsim
Usage: fdlsim [-d] [<filename>] <cycles>
-h          print this message
-v          value-change trace mode
-d          optional file name (default=stdin)
[filename] optional file name (default=stdin)
cycles     number of cycles to simulate (-1=forever)
```

برنامه را یکبار با دوره (cycle) ۳ و بار دیگر ۱۶ اجرا می کنیم.

```
dark@ubuntu:~/gezel$ fdlsim fourinand.fdl 3
0 0 0 0->0
1 0 0 0->0
0 1 0 0->0
dark@ubuntu:~/gezel$ fdlsim fourinand.fdl 16
0 0 0 0->0
1 0 0 0->0
0 1 0 0->0
1 1 0 0->0
0 0 1 0->0
1 0 1 0->0
0 1 1 0->0
1 1 1 0->0
0 0 0 1->0
1 0 0 1->0
0 1 0 1->0
1 1 0 1->0
0 0 1 1->0
1 0 1 1->0
0 1 1 1->0
1 1 1 1->1
dark@ubuntu:~/gezel$
```

مثال ۲) UpDown counter



می خواهیم کدی را بنویسیم که کنترل کننده شمارنده باشد.

اگر سیگنال کنترلی برابر ۱ شد، عدد مدنظر را یک واحد افزایش دهد و

اگر ۰ شد، یک واحد کاهش دهد.

همچنین فرض کردیم مقدار اولیه شمارنده عدد صفر باشد.

بلوک بالایی بخش کنترل را برعهده دارد و بلوک پایینی باتوجه به بخش

کنترلر، تصمیم می گیرد کدام یک از عملیات را اجرا کند.

پس بخش fsm به صورت زیر است:

S0 = up counter

if (S0 and up) >> S0

if (S1 and up) >> S0

S1 = down counter

if (S0 and down) >> S1

if (S1 and down) >> S1

حال باید کد fdl نوشته شود.

ابتدا بخش data path را می نویسیم.

```
updowncounter.fdl (~/gezel) - gedit
dp updowncounter(in dpth_in: ns(1); out output: ns(3)){
  reg cnt: ns(3);
  reg dnr: ns(1);

  always {
    dnr = dpth_in;
    output = cnt;
  }

  sfg up {
    cnt = cnt + 1;
    $display("number of counter: ",cnt, " up");
  }

  sfg down {
    cnt = cnt - 1;
    $display("number of counter: ",cnt, " down");
  }
}
```

• نکته: توجه شود که فقط یکبار می توان از always در هر dp استفاده کرد.

• نکته: توجه شود که مقدار اولیه تمامی رجیسترها صفر است.

سپس بخش کنترلی (fsm) را می نویسیم.

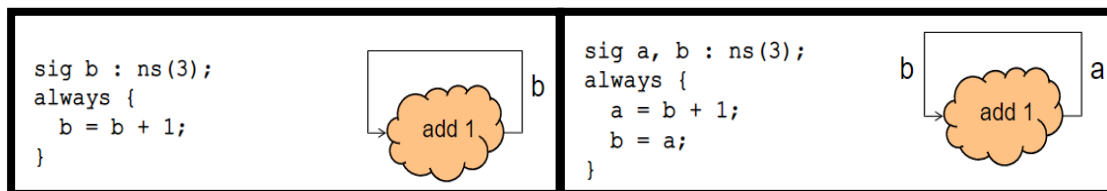
```
fsm cntrol_updowncounter (updowncounter){
  initial s0;
  state s1;
  @s0 if (dnr) then (down) -> s1;
                        else (up) -> s0;
  @s1 if (~dnr) then (up) -> s0;
                        else (down) -> s1;
}
```

در نهایت بخش test bench و ماژول اصلی برای ران شدن نوشته می شود.

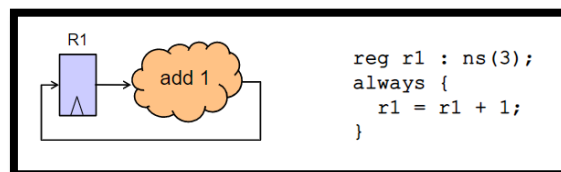
برای نوشتن بخش test bench از یک LUT کمک گرفتیم. متغیر i به عنوان رجیستر تعریف شده است و هر دفعه به مقدار آن افزوده می شود تا بتوانیم در LUT حرکت کنیم.

```
dp tb(out dnr: ns(1)){
  lookup exmpl: ns(8) = {1, 0, 0, 0, 0, 0, 0, 1};
  reg i: ns(8);
  always {
    dnr = exmpl(i);
    i = i + 1;
  }
}
```

• **نکته:** توجه شود که متغیر i باید حتما رجیستر تعریف شود و نه سیگنال.



اشتباه:



درست:

در انتها، بخش نهایی نوشته می شود.

```
dp sysupdowncounter{
  sig dnr: ns(1);
  sig output: ns(3);
  use tb(dnr);
  use updowncounter(dnr, output);
}

system S{
  sysupdowncounter;
}
```

حال به محیط ترمینال رفته و اجرا می کنیم. از آنجایی که LUT ما دارای ۸ عدد است، اگر مقدار i برابر ۸ شود، LUT در حافظه خود مقداری ندارد. پس خطا می دهد. پس می توانیم این کد را برای دوره/سیکل های ۸ و کمتر از آن اجرا کنیم. برای اینکه دوره را افزایش دهیم، کد باید تغییر کند. $\text{If } (i > 7) \text{ then } i = 0$

```

dark@ubuntu: ~/gezel
dark@ubuntu:~$ fdlsim updowncounter.fdl 8
fdlsim: command not found
dark@ubuntu:~$ cd /home/dark/gezel/
dark@ubuntu:~/gezel$ fdlsim updowncounter.fdl 8
fdlsim: command not found
dark@ubuntu:~/gezel$ export PATH=$PATH:/opt/gezel/bin/
dark@ubuntu:~/gezel$ fdlsim updowncounter.fdl 8
number of counter: 0/1 up
number of counter: 1/0 down
number of counter: 0/1 up
number of counter: 1/2 up
number of counter: 2/3 up
number of counter: 3/4 up
number of counter: 4/5 up
number of counter: 5/6 up
dark@ubuntu:~/gezel$ fdlsim updowncounter.fdl 2
number of counter: 0/1 up
number of counter: 1/0 down
dark@ubuntu:~/gezel$

```

دلیل اینکه مقدار خروجی را با علامت / نشان داده است، این است که متوجه نشده که display (cnt) مقدار قبل از تغییر یا بعد از تغییر را می خواهد.

مفهوم خط اول: مقدار اولیه cnt صفر بوده است. دستور up اجرا شده است. مقدار cnt به یک تغییر کرده است. پس

first (0) changed (/) then(1) چاپ می کند:

مثال ۳) sequence

$$A = (A + 2) \bmod 2;$$

فرض کنید همچنین کاری را می‌خواهیم انجام دهیم:

$$A = ((A + 1) + 1) \bmod 2;$$

اما یک واحد یک واحد می‌توانیم به A اضافه کنیم. پس داریم:

add – add – mod

حال اگر بخواهیم با state machine پیاده کنیم، داریم:

پس فایل fdl خود را اینگونه می‌نویسیم:

```
sequenceexmpl.fdl (~/.gezel/codes) - gedit
dp sequenceexmpl (in data: ns(4); out output: ns(8)){
  reg acc: ns(8);
  sfg phase1 {
    acc = data;
    $display("phase1(initialize): ", acc);
    output = 0;
  }

  sfg phase2 {
    acc = acc + 1;
    $display("phase2(++): ", acc);
    output = 0;
  }

  sfg phase3 {
    acc = acc % 2;
    $display("phase3(divide by 2): ", acc);
    output = 0;
  }

  sfg phase4 {
    output = acc;
    $display("phase4(output): ", output);
  }
}
sequencer h_sequenceexmpl(sequenceexmpl) {phase1; phase2; phase2; phase3; phase4;}
```

از sequencer استفاده می‌کنیم تا این مراحل را به ترتیب انجام دهد.

و اما مابقی کد:

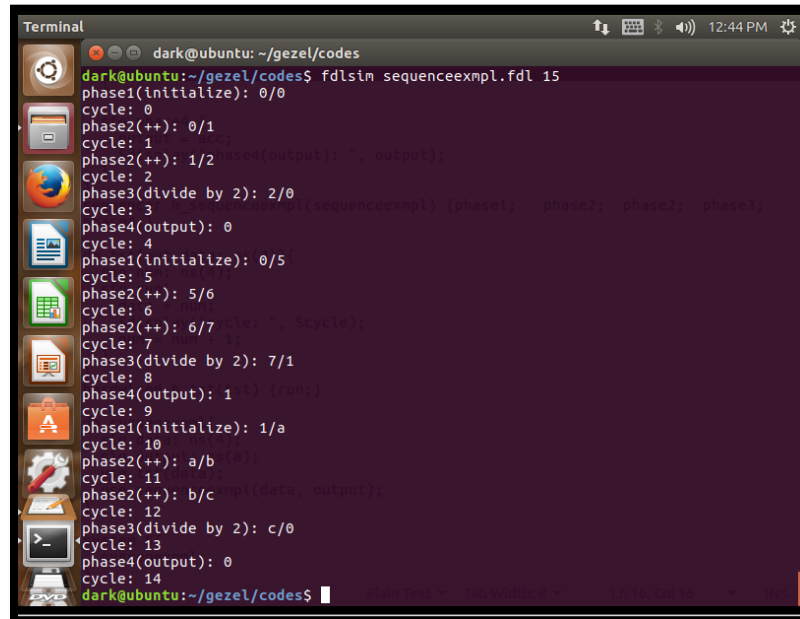
```
dp tst(out data: ns(4)){
  reg num: ns(4);
  sfg run{
    data = num;
    $display("cycle: ", $cycle);
    num = num + 1;
  }
}
hardwired h_tst(tst) {run;}

dp sysseqexmpl{
  sig data: ns(4);
  sig output: ns(8);
  use tst(data);
  use sequenceexmpl(data, output);
}

system S{
  sysseqexmpl;
}
```

اگر بخواهیم شماره دوره/سیکل را نمایش دهیم، از دستور $\$display(\$cycle)$ استفاده می‌کنیم.

در محیط ترمینال خروجی را مشاهده می کنید:



```
dark@ubuntu: ~/gezel/codes
dark@ubuntu:~/gezel/codes$ fdlsin sequenceexmpl.fdl 15
phase1(initialize): 0/0
cycle: 0
phase2(++): 0/1
cycle: 1
phase2(++): 1/2 phase4(output): ", output);
cycle: 2
phase3(divide by 2): 2/0
cycle: 3
phase4(output): 0
cycle: 4
phase1(initialize): 0/5
cycle: 5
phase2(++): 5/6
cycle: 6
phase2(++): 6/7 phase1(output): ", 5cycle);
cycle: 7
phase3(divide by 2): 7/1
cycle: 8
phase4(output): 1 1) (run);
cycle: 9
phase1(initialize): 1/a
cycle: 10
phase2(++): a/b (1);
cycle: 11
phase2(++): b/c mul(data, output);
cycle: 12
phase3(divide by 2): c/0
cycle: 13
phase4(output): 0
cycle: 14
dark@ubuntu:~/gezel/codes$
```

در سیکل صفر، متغیر عدد صفر را به مرحله بعدی منتقل می کند. $data = A$

در سیکل یک و دو، در نهایت ۲ واحد به متغیر اضافه می شود. $A = A + 2$

در سیکل سه، متغیر تقسیم بر دو می شود و باقی مانده ذخیره می شود. $A = A \% 2$

در سیکل چهارم، باقی مانده به خروجی منتقل می شود. $Output = A$

در سیکل پنجم، چون شمارنده به عدد ۵ رسیده است، عدد ۵ به عنوان ورودی داده شده است. به همین جهت نمایش داده شده است 0/5.

پس در سیکل ششم و هفتم بایستی عدد ۷ را داشته باشیم.

الی آخر.