



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 نصب gezel:gezel
۶	جدول راهنما:
۶	مثال ها:
۶	مثال ۰) چاپ یک پیام ساده
۷	مثال ۱) طراحی یک گیت and با ۴ ورودی (با استفاده از and ۲ ورودی)
۹	مثال ۲) UpDown counter
۱۲	مثال ۳) sequence
۱۴	طراحی توام سخت افزار / نرم افزار:
۱۴	مثال ۰) چاپ یک پیام ساده با عملیات ضرب
۱۷	مثال ۱) Encoder Priority
۲۰	مثال ۲) تبادل

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 ۱۶،۰۴ - حجم فایل حدود ۱۵۶۰ مگ) است که از

<http://releases.ubuntu.com/16.04/>

طریق سایت روبهرو دانلود شده است.

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

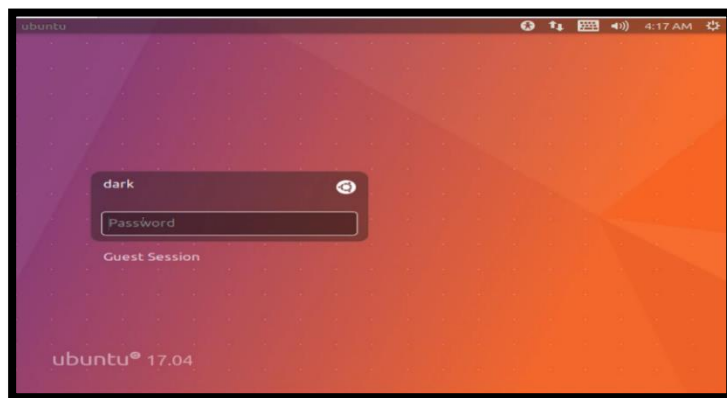
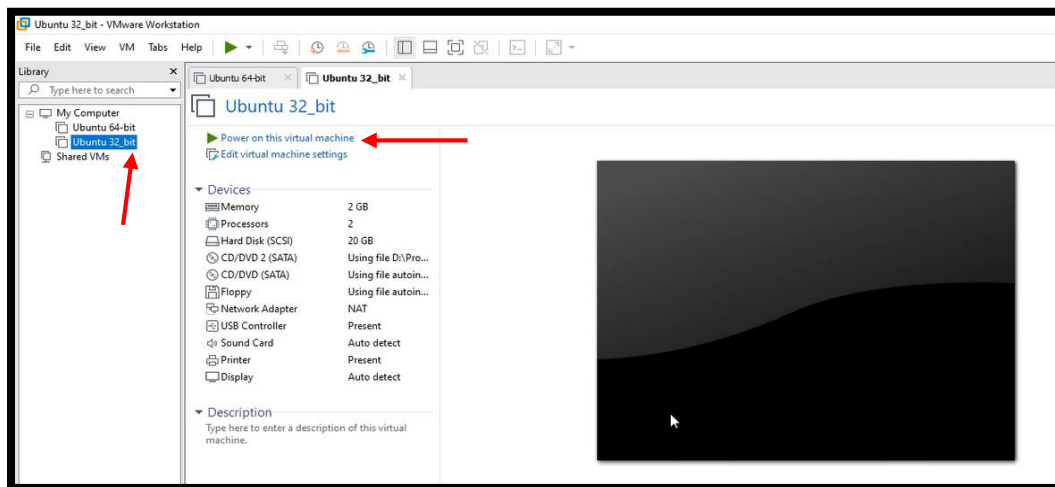
File >> New virtual machine >> Typical >> Next >>

Installer disc image file (iso) >> Browse: محل قرارگیری فایل دانلود شده >> Next >>

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

Virtual machine name: >> Next >> Next >> Finish

سپس گزینه Power on this virtual machine را اجرا کنید.



صبر کنید تا به صفحه ورود (login) برسید.

روش نصب gezel:

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

https://drive.google.com/file/d/1aH_PBkrl2OxNbOJNJYBl-ax2e8la5hAV/view?usp=sharing

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

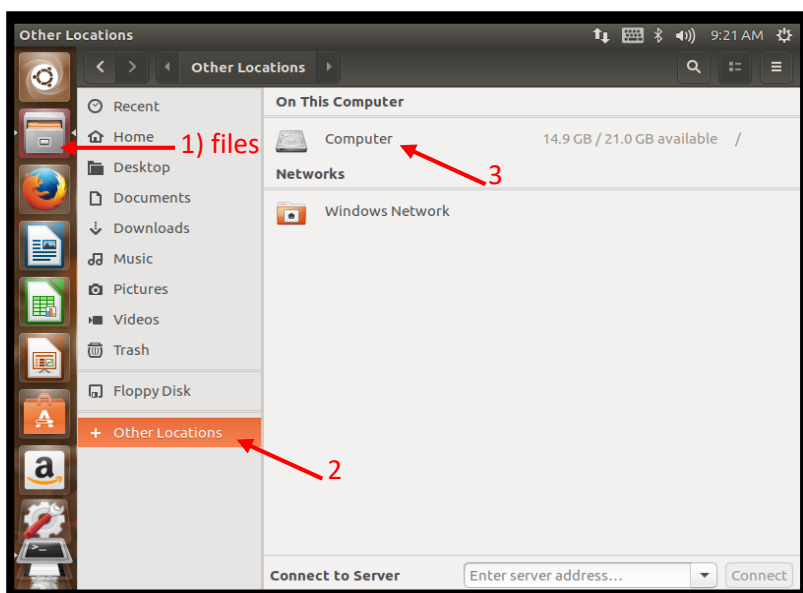
(۱) یک پوشه در هرجایی از سیستم اصلی (Host) که مدنظرتان است ایجاد کنید. داخل آن فایل های gezel.zip

را قرار داده و Extract کنید. سپس به VMWare بازگردید.

Add >> Always Enabled >> Shared Folders >> Options >> Settings >> VM از بخش منو

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

(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.

(۳) کپی کردن لینک در مرورگر و دانلود مستقیم.

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
sudo dpkg -i arm-linux-gcc-3.2.deb
export PATH=$PATH:/opt/gezel/bin/
fdlsim Hello_World.fdl

```

دستور fdlsim بایستی اجرا شود و یک سری اطلاعات را نمایش دهد. دستورات در فایل info.txt وجود دارند.

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.

برای نصب دستور gplatform، دستورهای زیر را وارد کنید:

```

sudo apt-get update
sudo apt-cache search libbfd
sudo apt-get -y install binutils-multiarch-dev
sudo ln /usr/lib/i386-linux-gnu/libbfd-2.26.1-system.so /usr/lib/i386-linux-gnu/libbfd-
2.20.1-system.20100303.so

```

سپس دستور زیر بایستی اجرا شود و یک سری اطلاعات به شما بدهد: gplatform

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.

به محیط ترمینال باز می گردیم. نتیجه نهایی برای دوره/سیکل های ۳، ۵ و ۱ قابل مشاهده است.

Hello_World.fdl (~/.gez) - gedit

```

dp helloWorld(){
    always {
        $display("Hello World");
    }
}

system S {
    helloWorld;
}

```

Terminal
dark@ubuntu: ~/gez/codes

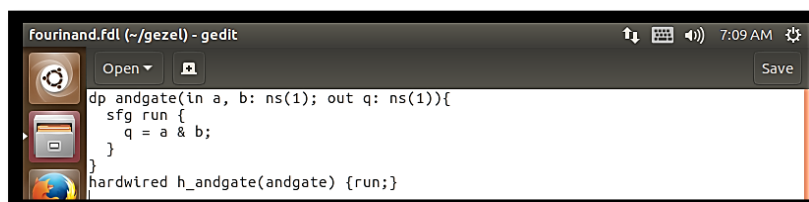
```

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



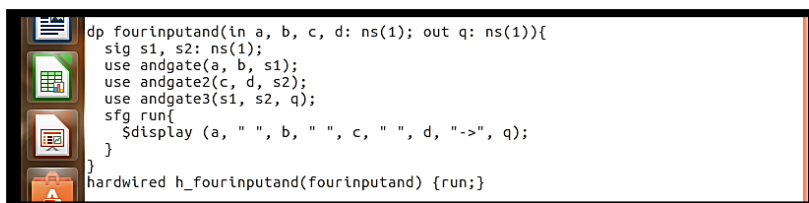
```
fourinand.fdl (~/.gezel) - gedit
Open Save
dp andgate(in a, b: ns(1); out q: ns(1)){
  sfg run {
    q = a & b;
  }
}
hardwired h_andgate(andgate) {run;}
```

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:



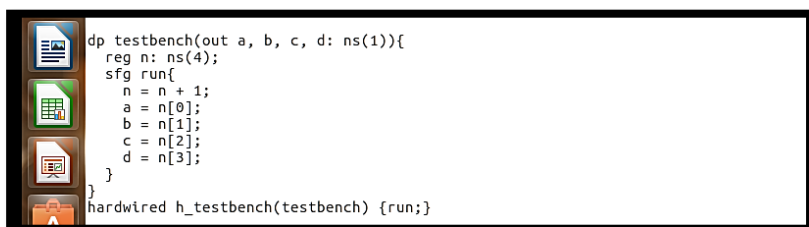
```
dp andgate2: andgate
dp andgate3: andgate
```

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;
}
```

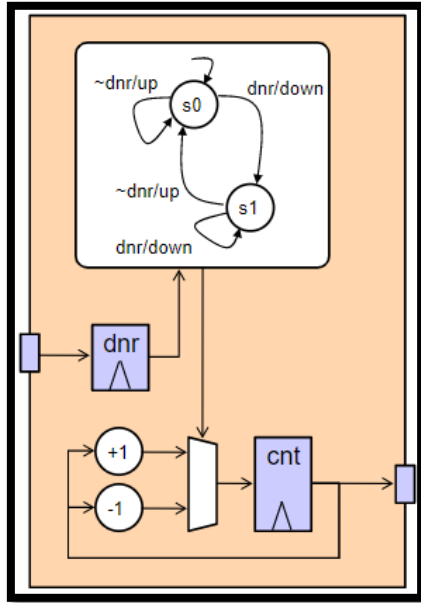
فایل نوشته شده را با نام fourinand.fdl ذخیره می کنیم و به محیط ترمینال برمی گردیم.

```
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
-d          value-change trace mode
[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 0 0 1->0
1 1 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.

```
sig b : ns(3);
always {
  b = b + 1;
}
```

```
sig a, b : ns(3);
always {
  a = b + 1;
  b = a;
}
```

اشتباه:

```
reg r1 : ns(3);
always {
  r1 = r1 + 1;
}
```

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. $\text{If } (i > 7) \text{ then } i = 0$

```

Terminal
dark@ubuntu: ~/gezel
dark@ubuntu:~$ fdisim updowncounter.fdl 8
fdisim: command not found
dark@ubuntu:~$ cd /home/dark/gezel/
dark@ubuntu:~/gezel$ fdisim updowncounter.fdl 8
fdisim: command not found
dark@ubuntu:~/gezel$ export PATH=$PATH:/opt/gezel/bin/
dark@ubuntu:~/gezel$ fdisim 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$ fdisim 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 (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

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

مثال ۳) sequence

فرض کنید همچنین کاری را می‌خواهیم انجام دهیم:

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

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

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

حال اگر بخواهیم با state machine پیاده کنیم، داریم:

add – add – mod

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:

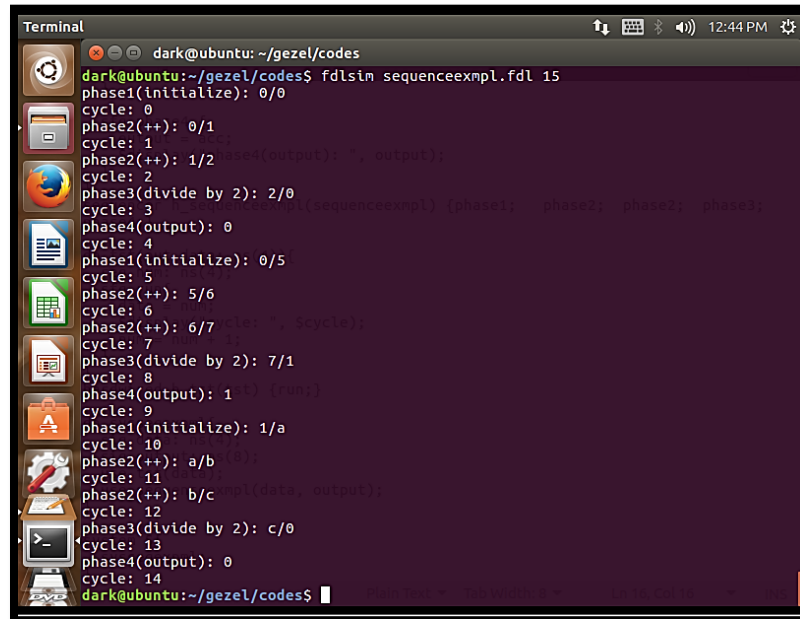
```
dp tst(out data: ns(4)){
  reg num: ns(4);
  sfg run{
    data = num;
    $display("cycle: ", $cycle);
    num = num + 1;
  }
}
hardwired h_tst(tst) {run;}

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 phase4(output): ", output);
cycle: 7
phase3(divide by 2): 7/1
cycle: 8
phase4(output): 1
cycle: 9
phase1(initialize): 1/a
cycle: 10
phase2(++): a/b
cycle: 11
phase2(++): b/c phase4(output): ", 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.

طراحی توام سخت افزار / نرم افزار:

مثال ۰) چاپ یک پیام ساده با عملیات ضرب

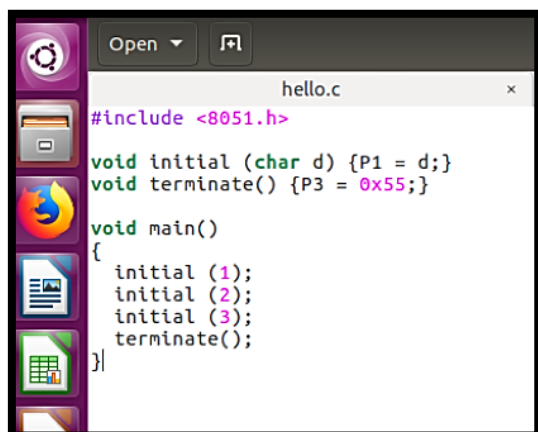
برای بخش نرم افزار یک تابع ساده نوشتیم که تنها ورودی بگیرد.

ورودی ها از طریق پورت P1 وارد سخت افزار می شوند.

تابع مقداردهی به پورت P1 را سه بار در تابع main فراخوانی کردیم.

تابع terminate که با پورت P3 کار می کند، جهت خاتمه دادن به

عملیات نوشته شده است. نام فایل نرم افزاری hello.c ذخیره کردیم.



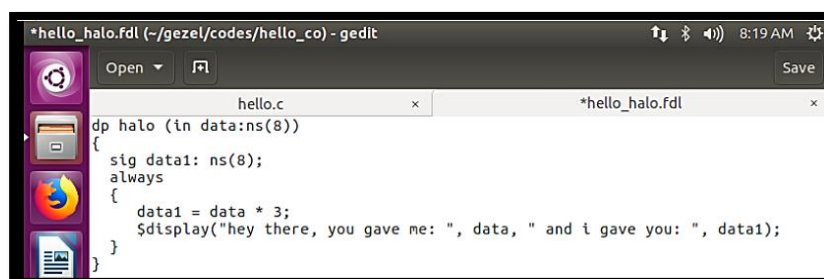
```
hello.c
#include <8051.h>

void initial (char d) {P1 = d;}
void terminate() {P3 = 0x55;}

void main()
{
    initial (1);
    initial (2);
    initial (3);
    terminate();
}
```

برای بخش سخت افزار ابتدا یک data path به نام halo تعریف کردیم که صرفاً ورودی را سه برابر کرده و تحویل

خروجی می دهد. همچنین مقدار ورودی و خروجی را چاپ می کند. نام فایل hello_halo.fdl است.



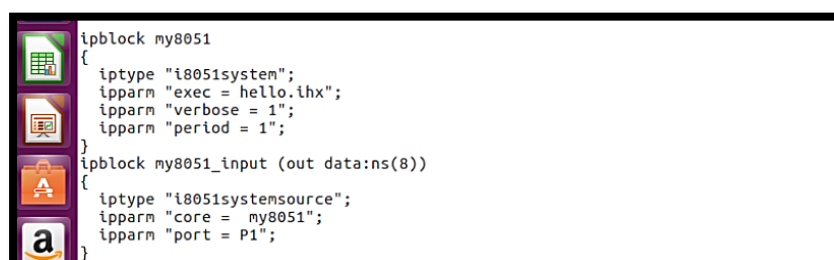
```
*hello_halo.fdl (-/gezel/codes/hello_co) - gedit
dp halo (in data:ns(8))
{
    sig data1: ns(8);
    always
    {
        data1 = data * 3;
        $display("hey there, you gave me: ", data, " and i gave you: ", data1);
    }
}
```

حال باید قسمت core و ports را پیاده سازی کنیم. برای پیاده سازی این بخش ها باید از ipblock استفاده شود.

ipblock my8051 برای نوشتن بخش core است. ipblock my8051_input برای بخش پورت است. اگر چندین

پورت استفاده شده باشد، این تابع بایستی برای هر پورت به صورت جداگانه تکرار شود.

مثلاً؛ پورت P0 << P0 my8051_p0 ipblock پورت P1 << P1 my8051_p1 ipblock و ...



```
ipblock my8051
{
    iptype "i8051system";
    ipparm "exec = hello.ihx";
    ipparm "verbose = 1";
    ipparm "period = 1";
}

ipblock my8051_input (out data:ns(8))
{
    iptype "i8051systemsource";
    ipparm "core = my8051";
    ipparm "port = P1";
}
```

اینکه پارامتر های تابع چه چیزهایی هستند، طبق manual جزل مشخص می شود که در صفحه بعد مشاهده می کنید.

Library Blocks in **gplatform** (Section 6.5 on page 65)

armsystem	Function	ARM Core + program memory. The ISS is SimIt-ARM.	
	IO		
	Parameters	exec	Name of the statically linked ELF binary to be executed on the ARM
		verbose	When set to 1, this ARM will execute in verbose (debug) mode, visualizing all system calls as they proceed.
		period	Relative clock period, default 1. When set to e.g. 2, the ARM will run at half speed relative to the system (gezel) clock.
armsystemsouce	Function	Memory-mapped cosimulation interface for an ARM core intercepting memory writes on this core.	
	IO	data	data output, ns(32)
	Parameters	core	Name of the armsystem block this cosimulation interface is connected to.
		address	Address decoded by this cosimulation interface.
armsystemsink	Function	Memory-mapped cosimulation interface for an ARM core intercepting memory reads from this core.	
	IO	data	data input, ns(32)
	Parameters	core	Name of the armsystem block this cosimulation interface is connected to.
		address	Address decoded by this cosimulation interface.
i8051system	Function	i8051 core + program memory	
	IO		
	Parameters	exec	Name of the intel-hex formatted i8051 binary to execute
		verbose	When set to 1, run the ISS in verbose (debug) mode.
		period	Relative clock period, default 1. When set to e.g. 2, the 8051 will run at half speed relative to the system (gezel) clock.
i8051systemsouce	Function	Port-mapped cosimulation interface to transport data from 8051 to GEZEL.	
	IO	data	output, ns(32), data output.
	Parameters	core	name of the i8051system core this port-mapped interface belongs to.
		port	quoted string, one of P0, P1, P2, P3.
i8051systemsink	Function	Port-mapped cosimulation interface to transport data from GEZEL to 8051 to GEZEL.	
	IO	data	input, ns(32), data input.
	Parameters	core	name of the i8051system core this port-mapped interface belongs to.
		port	quoted string, one of P0, P1, P2, P3.

حال فقط بخش تاپ ماژول مانده است.

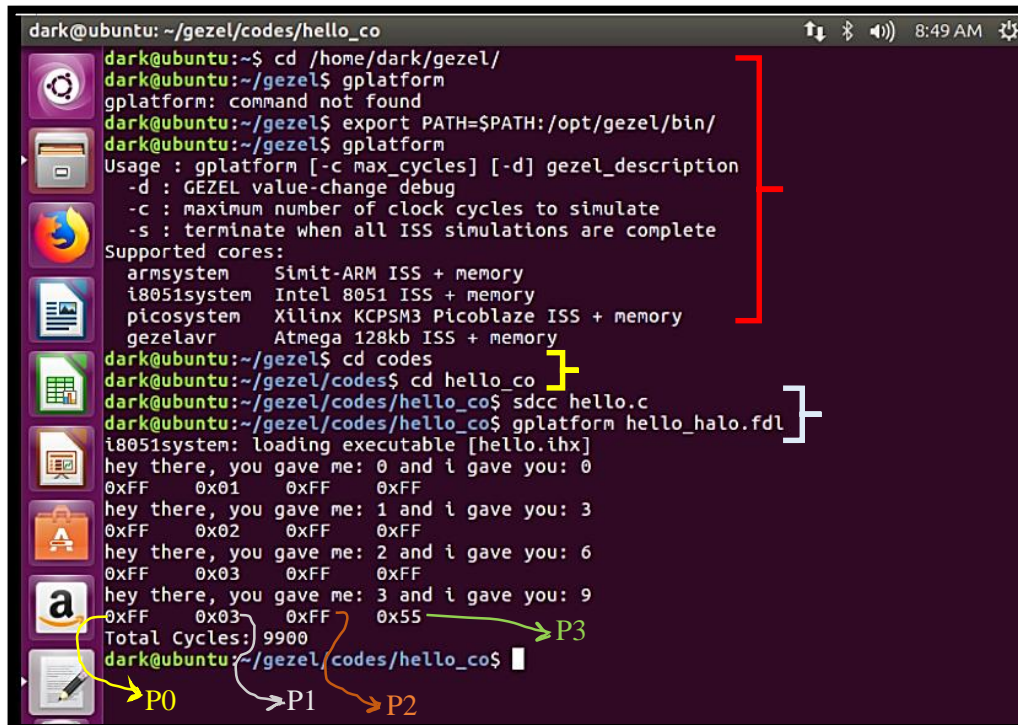


```
dp sys
{
  sig data: ns(8);
  use halo(data);
  use my8051;
  use my8051_input(data);
}
system S {sys;}
```

ابتدا تابع halo و سپس بخش core و در نهایت بخش ports فراخوانی می شود.

حال وارد ترمینال می شویم. ابتدا باید فایل hello.c را به hello.ihx تبدیل کنیم. همچنین در بخش core اشاره شده

که فایل نرم افزار ۷۸ به نام hello.ihx است. (ihx مخفف intel hex است)



```
dark@ubuntu: ~/gezel/codes/hello_co
dark@ubuntu:~$ cd /home/dark/gezel/
dark@ubuntu:~/gezel$ gplatform
gplatform: command not found
dark@ubuntu:~/gezel$ export PATH=$PATH:/opt/gezel/bin/
dark@ubuntu:~/gezel$ gplatform
Usage : gplatform [-c max_cycles] [-d] gezel_description
-d : GEZEL value-change debug
-c : maximum number of clock cycles to simulate
-s : terminate when all ISS simulations are complete
Supported cores:
armsystem      Simit-ARM ISS + memory
i8051system     Intel 8051 ISS + memory
picosystem     Xilinx KCPSM3 Picoblaze ISS + memory
gezelavr       Atmega 128kb ISS + memory
dark@ubuntu:~/gezel$ cd codes
dark@ubuntu:~/gezel/codes$ cd hello_co
dark@ubuntu:~/gezel/codes/hello_co$ sdcc hello.c
dark@ubuntu:~/gezel/codes/hello_co$ gplatform hello_halo.fdl
i8051system: loading executable [hello.ihx]
hey there, you gave me: 0 and i gave you: 0
0xFF 0x01 0xFF 0xFF
hey there, you gave me: 1 and i gave you: 3
0xFF 0x02 0xFF 0xFF
hey there, you gave me: 2 and i gave you: 6
0xFF 0x03 0xFF 0xFF
hey there, you gave me: 3 and i gave you: 9
0xFF 0x03 0xFF 0x55
Total Cycles: 9900
dark@ubuntu:~/gezel/codes/hello_co$
```

بخش قرمز، بخش تکراری شناسایی دستورات جزل است. بخش زرد برای ورود به پوشه کدهاست. (توجه کنید که دستور

export باید در پوشه gezel اجرا شود و سپس وارد پوشه کد ها شوید)

بخش آبی، بخش جدید برای دستورات هم طراحی است: sdcc hello.c و gplatform hello_halo.fdl

خروجی کد قابل مشاهده است. توجه کنید طبق manual ما می توانیم تنها از چهار پورت P0، P1، P2 و P3 استفاده

کنیم که به ترتیب از چپ به راست در هر مرحله نوشته می شوند.

مثال ۱) Encoder Priority

در این مثال می خواهیم یک انکودر اولویت دار پیاده سازی کنیم.

جدول درستی انکودر اولویت دار ۴ ورودی به ۲ خروجی در تصویر روبه‌رو

قابل مشاهده است. انکودر در بخش سخت افزار پیاده سازی می شود.

برای بخش نرم افزار از سه پورت P0، P1 و P3 استفاده کردیم. پورت

P3 برای خاتمه دادن به برنامه، پورت P1 برای اعمال ورودی و پورت P0

سیگنال کنترلی سیستم می باشد. پورت P0 با توجه به تعریف enum در

یک فرکانس ۱ و سپس در فرکانس بعدی ۰ می شود.

نام فایل نرم افزاری را encyp.c ذخیره کردیم.

Truth table

```

#include <8051.h>
enum {ins_0, ins_1};
void start(char inp)
{
    P1 = inp;
    P0 = ins_1;
    P0 = ins_0;
}
void terminate() {P3 = 0x55;}
void main()
{
    start(1);
    start(2);
    start(3);
    start(4);
    start(5);
    start(6);
    start(7);
    start(8);
    start(9);
    terminate();
}
    
```

حال برای بخش سخت افزاری بایستی چندین نکته را توجه کرد:

۱- غیر از پورت P3 (خاتمه دهنده) از دو پورت سیگنالی استفاده شده است. پس باید ipblock p0 و ipblock p1

در فایل fdl تعریف شوند.

۲- دو پورت سیگنالی استفاده شده است. پس ورودی فایل fdl دو سیگنال خواهد بود.

```

encyprio.fdl (~/.gezeli/codes/encoderpriority) - gedit
Open [F]

encyp.c x encyprio.fdl

dp ency (in data:ns(4); in ins:ns(4))
{
    reg inreg: ns(4);
    reg insreg: ns(4);
    sfg start { $display("start, your data is: ", data); }
    sfg encode { inreg = data; insreg = ins; }
    sfg ency0 { $display("output is 0"); }
    sfg ency1 { $display("output is 1"); }
    sfg ency2 { $display("output is 2"); }
    sfg ency3 { $display("output is 3"); }
}

fsm f_ency (ency)
{
    initial s0;
    state s1, s2, s3, s4;
    @s0 if (insreg == 1) then (start, encode) -> s1;
    else (encode) -> s0;
    @s1 if (inreg[3]) then (ency3, encode) -> s4;
    else (encode) -> s2;
    @s2 if (inreg[2]) then (ency2, encode) -> s4;
    else (encode) -> s3;
    @s3 if (inreg[1]) then (ency1, encode) -> s4;
    else (ency0, encode) -> s4;
    @s4 if (insreg == 0) then (encode) -> s0;
    else (encode) -> s4;
}
    
```

دو شرط $insreg = 0$ و $insreg = 1$ در S0 و S4 برای این هستند که مطمئن شویم در بخش نرم افزار پورت P0

تغییر کرده است. با برداشتن این دو شرط، خروجی به صورت زیر خواهد بود (داده ورودی عدد یک است):

قبل از فلش زرد، خط اول پورت P0 مقدار 0xFF را دارد. پس ۵ بار با این مقدار خروجی تولید کرده است.

بعد از فلش زرد، عدد اول از سمت چپ 0x01 است. به عبارتی پورت P0

مقدار یک را گرفته است و حال ۵ بار دیگر اجرا می کند.

بعد از فلش آبی، پورت P0 مقدار صفر را گرفته است و ۲۰ بار اجرا می

کند که ادامه اجراها تصویر قرار داده نشده است. در نهایت بعد از ۳۰ بار

اجرا، ورودی بعدی را می گیرد و همین روند تکرار می شود.

```
i8051system: loading executable [encyp.ihx]
0xFF 0x01 0xFF 0xFF
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
0x01 0x01 0xFF 0xFF
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
0x00 0x01 0xFF 0xFF
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
output is 0
start, your data is: 1
```

```
start, your data is: 1
output is 0
start, your data is: 1
0x00 0x02 0xFF 0xFF
output is 0
0x01 0x02 0xFF 0xFF
0x00 0x02 0xFF 0xFF
0x00 0x03 0xFF 0xFF
start, your data is: 3
output is 1
```

البته ناگفته نماند ورودی ها یک در میان دچار مشکل می شوند. همانند

تصویر زیر که برای ورودی ۲ خروجی نداده است و ورودی ۳ را گرفته است.

ipblock my8051 برای بخش core و دو ipblock دیگر برای پورت ها هستند.

اگرچه ورودی هر دو ipblock پورت ها سیگنال out data: ns(4) است، اما در تاپ ماژول مشخص می کنیم که هر

پورت برای چه داده ای است. (در dp ency نوشته شده که سیگنال data و ins هر دو ۴ بیتی بدون علامت هستند)

```
ipblock my8051
{
  iptype "i8051system";
  ipparam "exec = encyp.ihx";
  ipparam "verbose = 1";
  ipparam "period = 1";
}

ipblock my8051_p1 (out data:ns(4))
{
  iptype "i8051systemsource";
  ipparam "core = my8051";
  ipparam "port = P1";
}

ipblock my8051_p0 (out data:ns(4))
{
  iptype "i8051systemsource";
  ipparam "core = my8051";
  ipparam "port = P0";
}
```

در نهایت در بخش تاپ ماژول داریم:

```
dp sys
{
  sig data: ns(4);
  sig ins: ns(4);
  use my8051;
  use my8051_p0(ins);
  use my8051_p1(data);
  use ency(data, ins);
}
system S {sys;}
```

سیگنال کنترلی ins برای پورت صفر و سیگنال داده data برای پورت

۱ است. در نهایت تابع ency فراخوانی می شود.

حال به محیط ترمینال رفته و کد را اجرا می کنیم.

دستور `gplatform -s` زمانی به کار می رود که نخواهیم تعداد دوره/سیکل مشخص کنیم. به عبارتی تا زمانی اجرا می

شود که تمام ورودی های بخش نرم افزار را اجرا کرده باشد و به دستور `terminate` رسیده باشد.

```
dark@ubuntu: ~/gezel/codes/encoderpriority
dark@ubuntu:~$ cd /home/dark/gezel/
dark@ubuntu:~/gezel$ export PATH=$PATH:/opt/gezel/bin/
dark@ubuntu:~/gezel$ cd codes
dark@ubuntu:~/gezel/codes$ cd encoderpriority
dark@ubuntu:~/gezel/codes/encoderpriority$ sdcc encyp.c
dark@ubuntu:~/gezel/codes/encoderpriority$ gplatform -s encyprio.fdl
```

خروجی در تصویر زیر قابل مشاهده است:

```
dark@ubuntu:~/gezel/codes/encoderpriority$ gplatform -s encyprio.fdl
i8051system: loading executable [encyp.ihx]
0xFF 0x01 0xFF 0xFF
0x01 0x01 0xFF 0xFF
start, your data is: 1
output is 0
0x00 0x01 0xFF 0xFF
0x00 0x02 0xFF 0xFF
0x01 0x02 0xFF 0xFF
start, your data is: 2
output is 1
0x00 0x02 0xFF 0xFF
0x00 0x03 0xFF 0xFF
0x01 0x03 0xFF 0xFF
start, your data is: 3
output is 1
0x00 0x03 0xFF 0xFF
0x00 0x04 0xFF 0xFF
0x01 0x04 0xFF 0xFF
start, your data is: 4
output is 2
0x00 0x04 0xFF 0xFF
0x00 0x05 0xFF 0xFF
0x01 0x05 0xFF 0xFF
start, your data is: 5
output is 2
0x00 0x05 0xFF 0xFF
0x00 0x06 0xFF 0xFF
0x01 0x06 0xFF 0xFF
start, your data is: 6
output is 2
0x00 0x06 0xFF 0xFF
0x00 0x07 0xFF 0xFF
0x01 0x07 0xFF 0xFF
start, your data is: 7
output is 2
0x00 0x07 0xFF 0xFF
0x00 0x08 0xFF 0xFF
0x01 0x08 0xFF 0xFF
start, your data is: 8
output is 3
0x00 0x08 0xFF 0xFF
0x00 0x09 0xFF 0xFF
0x01 0x09 0xFF 0xFF
start, your data is: 9
output is 3
0x00 0x09 0xFF 0xFF
0x00 0x09 0xFF 0x55
Total Cycles: 10908
dark@ubuntu:~/gezel/codes/encoderpriority$
```

مثال ۲) تبادل

فرض کنیم قرار است از یک پورت عددی را از 8051 به gezel فرستاده و نتیجه را از gezel به 8051 منتقل کنیم.

```
ex.c
#include <8051.h>

void start(char input)
{
    char output;
    P0 = input;
    output = P1;
    P2 = output + 1;
}

void terminate() {P3 = 0x55;}

void main()
{
    start(3);
    start(6);
    start(9);
    terminate();
}
```

در بخش نرم افزار داده بر روی پورت P0 نوشته شده و به سخت افزار فرستاده می شود. سخت افزار عملیات را انجام داده و خروجی را روی پورت P1 قرار می دهد. سپس نرم افزار به ادامه کار خود می پردازد و بر روی پورت P2 مقدار P1+1 قرار می دهد.

برای بخش سخت افزار یک کد ساده نوشتیم که عدد را گرفته و در ۲ ضرب کند.

```
dp double(in data_in:ns(8); out data_out:ns(8))
{
    sig outy: ns(8);
    always
    {
        data_out = data_in << 1;
        $display("input: ", data_in, " and output: ", data_out); }
}
```

سپس باید پورت ها را تعریف کرد. از آنجایی که P1 برای انتقال داده از سخت افزار به نرم افزار است، از نوع i8051systemsink تعریف می شود.

```
ipblock my8051
{
    iptype "i8051system";
    ipparm "exec = ex.lhx";
    ipparm "verbose = 1";
    ipparm "period = 1";
}

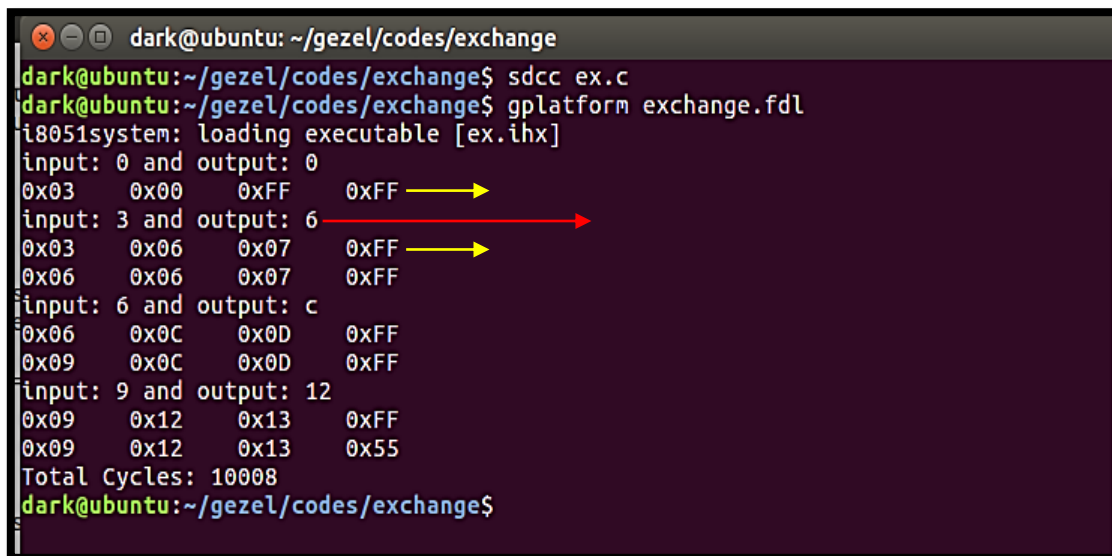
ipblock my8051_p0 (out data:ns(8))
{
    iptype "i8051systemsource";
    ipparm "core = my8051";
    ipparm "port = P0";
}

ipblock my8051_p1 (in data:ns(8))
{
    iptype "i8051systemsink";
    ipparm "core = my8051";
    ipparm "port = P1";
}
```

و در نهایت ماژول نهایی را داریم:

```
dp sys
{
  sig data_in: ns(8);
  sig data_out: ns(8);
  use my8051;
  use my8051_p0(data_in);
  use double(data_in, data_out);
  use my8051_p1(data_out);
}
system S {sys;}
```

حال وارد ترمینال شده و اجرا می کنیم.



```
dark@ubuntu: ~/gezel/codes/exchange
dark@ubuntu:~/gezel/codes/exchange$ sdcc ex.c
dark@ubuntu:~/gezel/codes/exchange$ gplatform exchange.fdl
i8051system: loading executable [ex.ihx]
input: 0 and output: 0
0x03  0x00  0xFF  0xFF →
input: 3 and output: 6 →
0x03  0x06  0x07  0xFF →
0x06  0x06  0x07  0xFF
input: 6 and output: c
0x06  0x0C  0x0D  0xFF
0x09  0x0C  0x0D  0xFF
input: 9 and output: 12
0x09  0x12  0x13  0xFF
0x09  0x12  0x13  0x55
Total Cycles: 10008
dark@ubuntu:~/gezel/codes/exchange$
```

تحلیل:

فلش زرد اول: پورت P0 مقدار ۳ را گرفته است و به سخت افزار منتقل کرده است.

فلش قرمز: عملیات سخت افزاری انجام گرفته شده است و خروجی ۶ بایستی به نرم افزار بازگردانده شود.

فلش زرد دوم: پورت P0 همچنان مقدار ۳ دارد. پورت P1 که از سخت افزار آمده مقدار ۶ را گرفته است. پورت P2 در

بخش نرم افزاری قصد انجام عملیات P1+1 دارد که نتیجه ۷ است و درست می باشد.

دقت کنید که اعداد به صورت Hex می باشند.