



دانشکده مهندسی کامپیوتر

آزمایشگاه طراحی سیستم‌های دیجیتال

آزمایش سوم - توصیف جریان داده

دکتر اجلالی، مهندس اثنی عشری

امیرمهدی کوششی — ۹۸۱۷۱۰۵۳

ایمان محمدی — ۹۹۱۰۲۲۰۷

شایان صالحی — ۹۹۱۰۵۵۶۱

۲۳ مرداد ۱۴۰۲

فهرست مطالب

۳	۱	مقدمه
۳	۲	پارت اول
۴	۱.۲	الگوریتم
۵	۲.۲	FPGA
۸	۳.۲	تست FPGA
۱۲	۳	پارت دوم
۱۳	۱.۳	FPGA
۱۴	۲.۳	تست FPGA

۱ مقدمه

در این آزمایش که در دو پارت انجام شده است ما قصد داریم با طراحی توصیفی data flow، مقایسه کننده بسازیم. در این آزمایش با کلاک و ریست کاری نداریم و با تغییر داده خروجی ما در همان لحظه تغییر می‌کند.

۲ پارت اول

در قسمت اول از این آزمایش، ما قصد داریم یک مقایسه کننده ۴-بیتی بسازیم که دو عدد را از ورودی گرفته و آن‌ها را مقایسه کند. برای این مقایسه کننده ما ابتدا یک مقایسه کننده ۱-بیتی ساخته و با استفاده از این ماژول در مقایسه کننده ۴-بیتی، پارت اول را می‌زنیم. ابتدا به مقایسه کننده ۱-بیتی می‌پردازیم.

```

1 module one_bit_comparator(
2     x,
3     y,
4     in_gt,
5     in_eq,
6     o_gt,
7     o_eq
8 );
9 input x, y, in_gt, in_eq;
10 output o_eq, o_gt;
11
12 assign o_eq = (x == y) & in_eq;
13 assign o_gt = (in_gt) | (in_eq & (x > y));
14 endmodule

```

در این ماژول ما ورودی‌های x و y و in_gt و in_eq را داریم و خروجی‌های ما o_eq و o_gt نیز خروجی‌های ما هستند.

ورودی‌های x و y همان ۱ بیت‌هایی هستند که قرار است با هم مقایسه کنند. ورودی‌های in_eq و in_gt نیز ورودی‌هایی هستند که به صورت آبشاری به بیت‌های کم ارزش‌تر برای تشخیص بزرگی یا کوچی عدد نهایی پاس داده می‌شود.

۱.۲ الگوریتم

الگوریتم کلی این مقایسه کننده به این شکل است که در ابتدای کار ورودی in_eq را برابر ۱ داده و in_gt را برابر ۰ می‌دهیم. زیرا ابتدا که بیت‌های پر ارزش قرار است مقایسه شوند، مطابق کدی که ما زدیم نیاز است تا ورودی‌ها به این شکل باشند. زیرا در کد همانطور که در بالا مشخص است، در صورتی که دو بیت ورودی به ماژول $in_eq_one_bit_comparator$ که به معنی برابر بودن دو بیت قبلی بوده است نیز ۱ باشد، پس می‌توانیم بگوییم که تا اینجا اعداد ما نیز برابر هستند. به عبارتی خط زیر همین قسمت را مشخص می‌کند.

```
assign o_eq = (x==y) & in_eq
```

همانطور که مشاهده می‌کنید ابتدا ۲ بیت ورودی چک می‌شوند و در صورتی که برابر بودند و همچنین in_eq که به معنی برابر بودن دو بیت قبلی آن است نیز ۱ بود پس می‌توان گفت که تا اینجا اعداد برابر هستند. حال به خط زیر توجه کنید.

```
assign o_gt = in_gt | (in_eq & (x > y))
```

مطابق این خط از کد می‌گوییم که اگر ورودی in_gt که نتیجه‌ی مقایسه‌ی ۲ بیت قبلی آن است، ۱ بود، پس یعنی عدد بزرگ تر بوده پس بدون توجه به مقایسه دو بیت فعلی می‌گوییم که عدد بزرگ تر است. (همانطور که در کد نیز مشخص است in_gt با قسمت مقایسه کننده OR شده است پس اگر ۱ باشد نتیجه OR نیز ۱ خواهد بود.) اما اگر در صورتی که in_gt ۱ نباشد، بررسی می‌کنیم که آیا دو بیت قبلی برابر بوده اند یا نه. در صورتی که دو بیت قبلی برابر بوده‌اند و در دوبیت فعلی x بزرگ تر از y باشد پس می‌توان گفت که عدد بزرگ تر است پس خروجی o_gt را ۱ می‌کنیم.

به عنوان مثال اعداد زیر را مشاهده کنید:

$$step0 \rightarrow A : 1101, B : 1100, in_eq = 1, in_gt = 0$$

$$step1 \rightarrow A_3 : 1, B_3 : 1, in_eq = 1, in_gt = 0 \rightarrow o_eq = 1, o_gt = 0$$

$$step2 \rightarrow A_2 : 1, B_2 : 1, in_eq = 1, in_gt = 0 \rightarrow o_eq = 1, o_gt = 0$$

$$step3 \rightarrow A_1 : 0, B_1 : 0, in_eq = 1, in_gt = 0 \rightarrow o_eq = 1, o_gt = 0$$

$$step4 \rightarrow A_0 : 1, B_0 : 0, in_eq = 1, in_gt = 0 \rightarrow o_eq = 0, o_gt = 1$$

حال در عکس زیر ماژول مقایسه کننده‌ی ۴-بیتی را مشاهده می‌کنید که با توجه به الگوریتم بالا کار خواهد کرد:

```

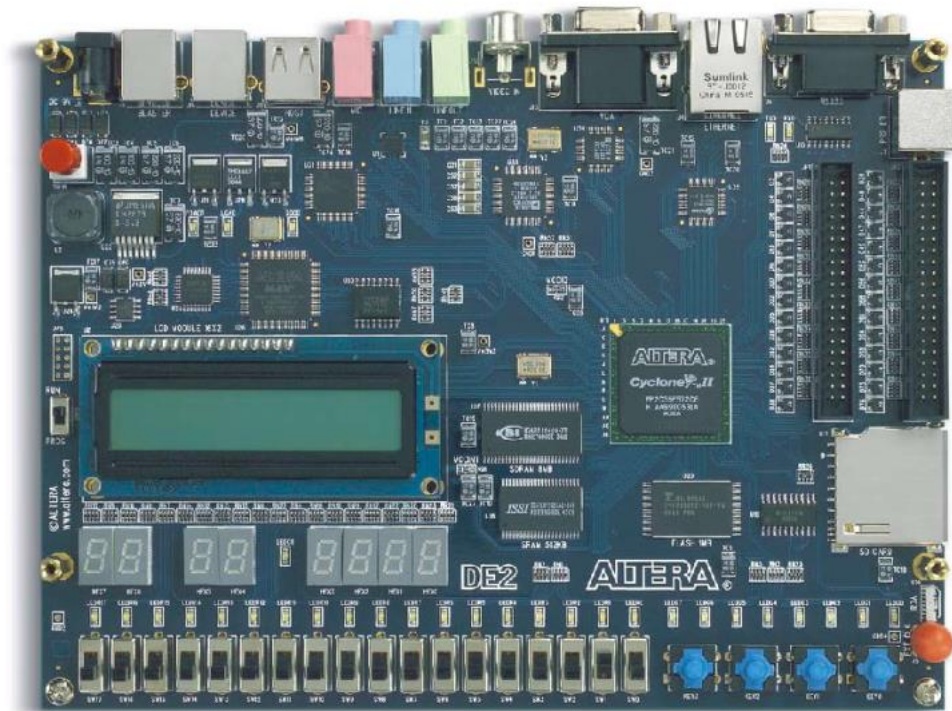
1 module four_bit_comparator(
2     input wire [3:0] x,
3     input wire [3:0] y,
4     input wire in_gt,
5     input wire in_eq,
6     output wire o_gt,
7     output wire o_eq
8 );
9
10 wire [2:0] o_eq_comps;
11 wire [2:0] o_gt_comps;
12
13 one_bit_comparator comp0(x[3], y[3], in_gt, in_eq, o_gt_comps[0], o_eq_comps[0]);
14 one_bit_comparator comp1(x[2], y[2], o_gt_comps[0], o_eq_comps[0], o_gt_comps[1], o_eq_comps[1]);
15 one_bit_comparator comp2(x[1], y[1], o_gt_comps[1], o_eq_comps[1], o_gt_comps[2], o_eq_comps[2]);
16 one_bit_comparator comp3(x[0], y[0], o_gt_comps[2], o_eq_comps[2], o_gt, o_eq);
17
18
19 endmodule

```

همانطور که در عکس بالا مشخص است، به ترتیب از بیت پر ارزش تا کم ارزش هر دو عدد را دو به دو با هم مقایسه می‌کنیم و خروجی تک بیتی‌های آن را که برابر یا بزرگ‌تر است را به بیت‌های بعدی به صورت آشناری می‌دهیم.

FPGA ۲.۲

در نهایت بعد از زدن کد، باید آن را روی FPGA پیاده کرده و تست کنیم. در ابتدا نیاز است پین‌های ورودی را مشخص کنیم و همچنین خروجی را نیز مشخص کنیم. ما برای خروجی‌ها در این آزمایش از LED ها استفاده کرده‌ایم. برای پین‌های ورودی نیز از سویچ‌های SW استفاده کرده‌ایم. در عکس زیر FPGA را مشاهده می‌کنید.



ما تنظیمات کوآرتوس‌مان را روی cyclone II تنظیم کرده‌ایم. حال از قسمت pin planner ورودی‌ها و خروجی‌های کدام را مطابق پین‌ها و LED های FPGA ست می‌کنیم.

Top View - Wire Bond
Cyclone II - EP2C35F672C6

Node	Node Name	Direction	Reserved	Location
1	GPIO	Input		PIN_A11
6	GPIO	Input		PIN_A13
7	GPIO	Input		PIN_B13
8	GPIO	Input		PIN_C13
9	GPIO	Input		PIN_A14
10	GPIO	Input		PIN_F25
11	GPIO	Input		PIN_B25
12	GPIO	Input		PIN_B25
13	<new node>			

Named: Edit:

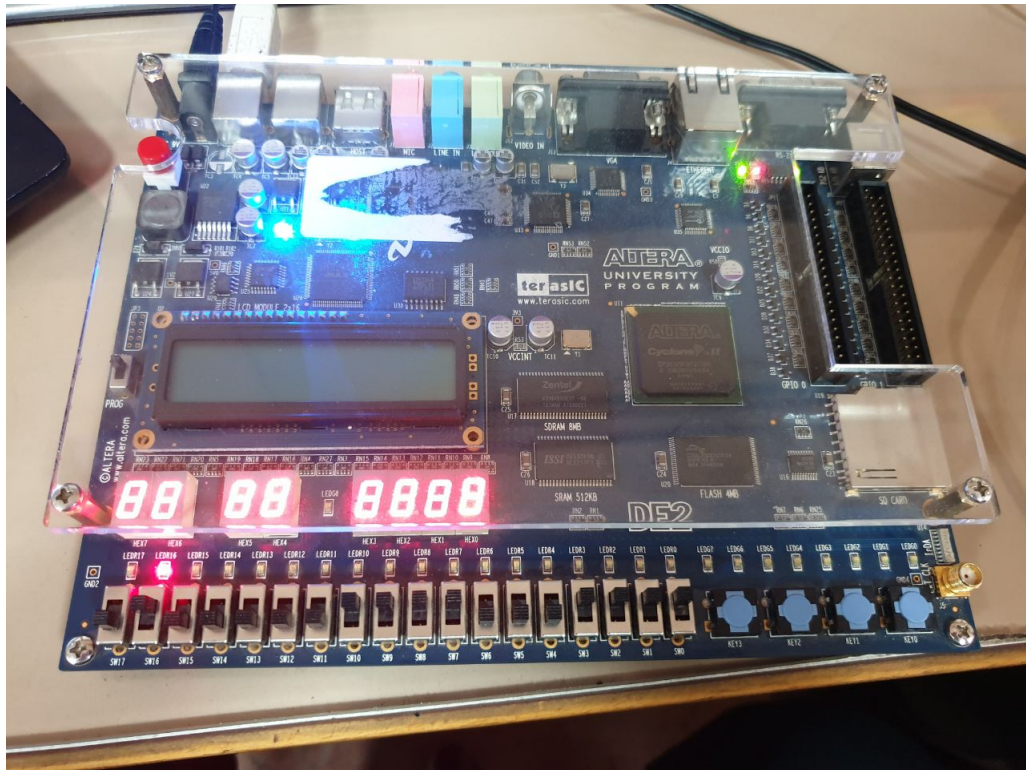
	Node Name	Direction	Reserved	Location
5	x[3]	Input		PIN_N1
6	x[2]	Input		PIN_A13
7	x[1]	Input		PIN_B13
8	x[0]	Input		PIN_C13
9	y[3]	Input		PIN_AE14
10	y[2]	Input		PIN_P25
11	y[1]	Input		PIN_N26
12	y[0]	Input		PIN_N25
13	<<new node>>			

حال به تست کردن FPGA می‌پردازیم. مطابق پین‌های بالا، مطابق FPGA، سویچ‌های SW_۹ و SW_{۱۰} و SW_۷ و SW_۸ به ترتیب ۴ بیت x از پر ارزش تا کم ارزش است. سویچ‌های SW_۳ و SW_۲ و SW_۱ و SW_۰ به ترتیب ۴ بیت y از پر ارزش تا کم ارزش هستند.

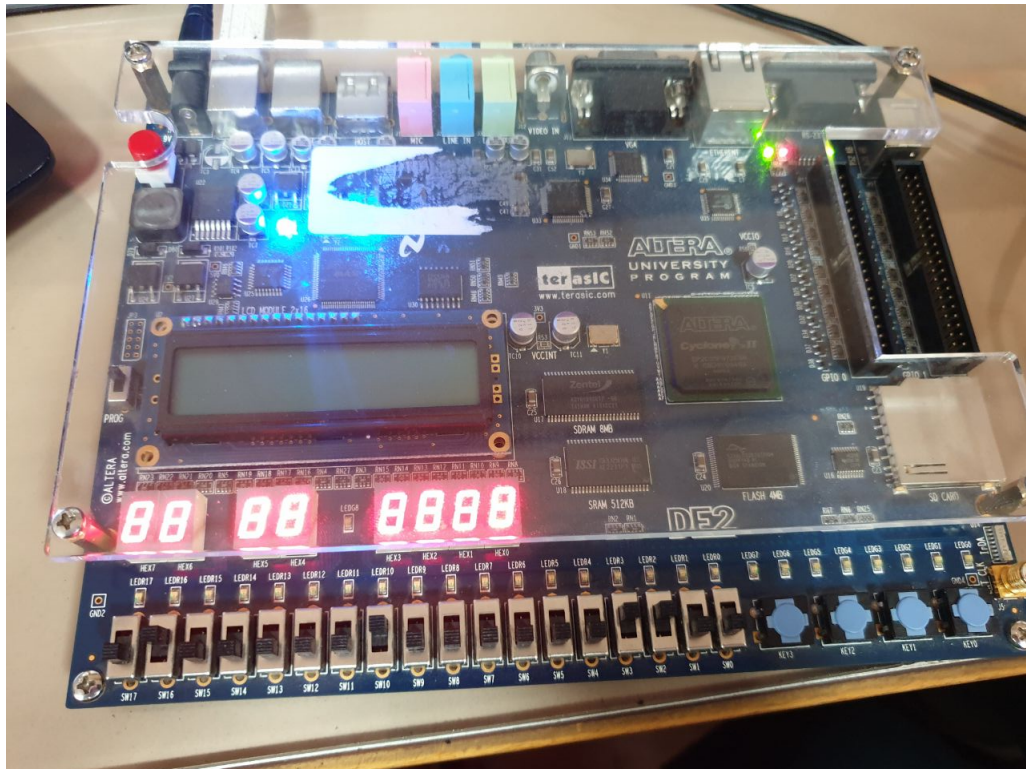
همچنین LED ۱۷ نشان گر بزرگ بودن عدد x و LED ۱۶ نشان‌دهنده‌ی برابری دوعدد است و همچنین اگر هر ۲ چراغ خاموش باشند نشانگر آن است که y بزرگ‌تر از x است.

۳.۲ تست FPGA

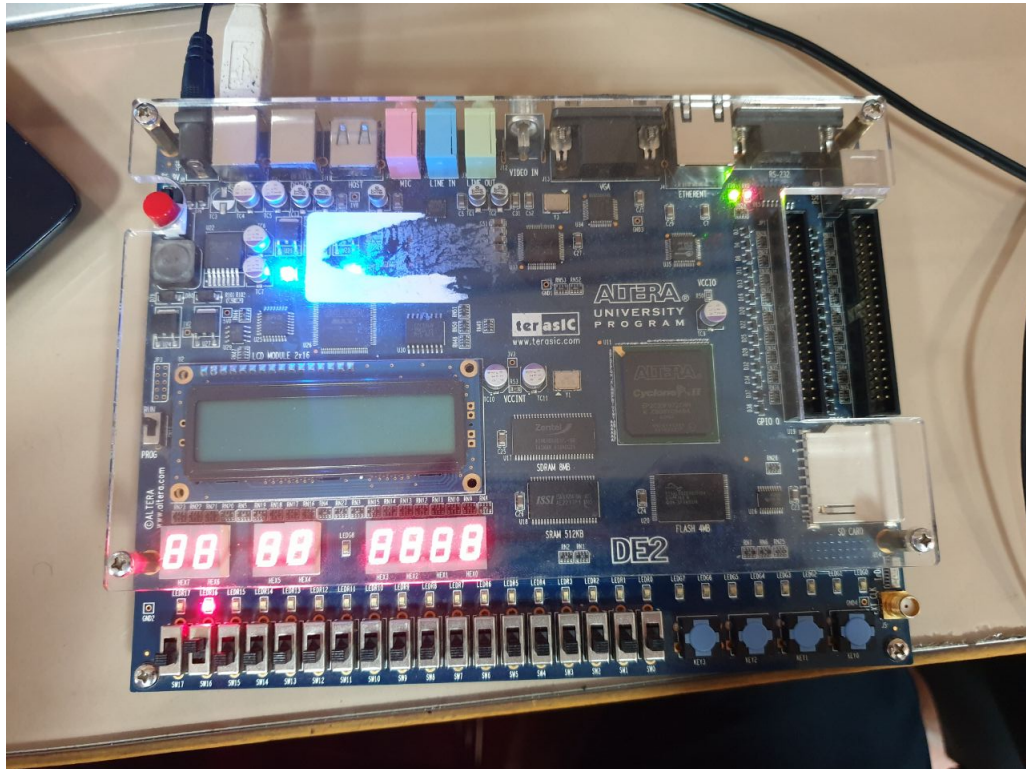
در عکس زیر تست FPGA است که $x=1111$ و $y=1111$ است و ۱۶ LED نیز روشن است که به معنی برابر بودن دو عدد می‌باشد.



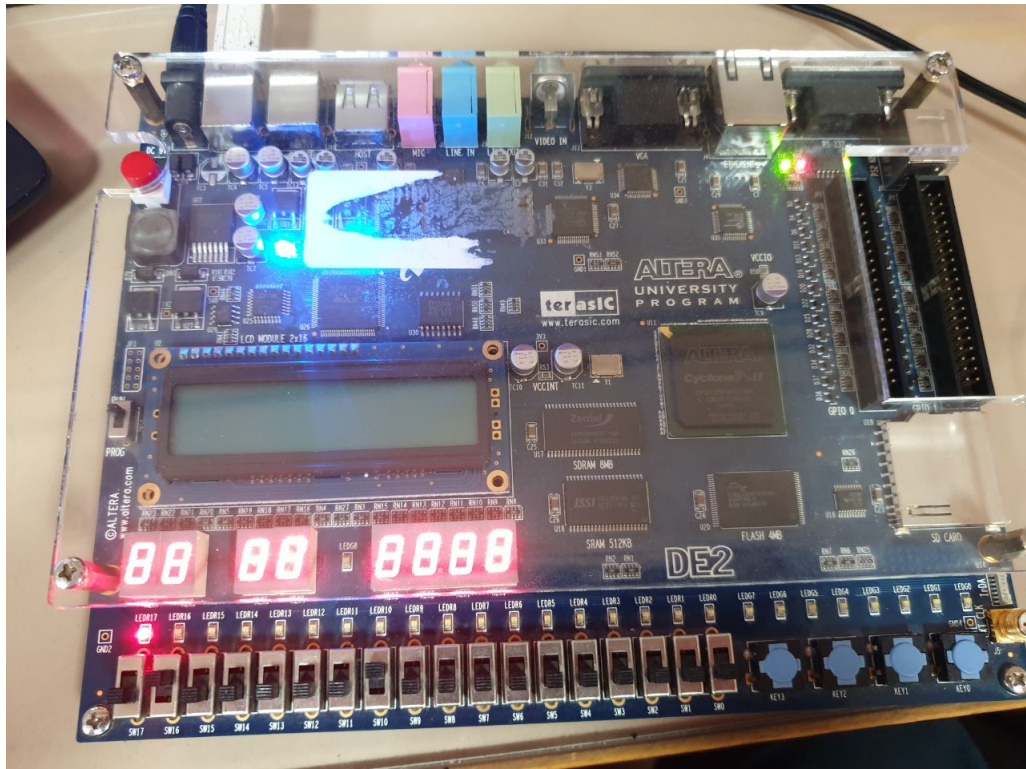
در عکس زیر تست FPGA است که $x = 1000$ و $y = 1100$ است و هر دو چراغ خاموش است که به معنی بزرگ‌تر بودن عدد y است.



در عکس زیر تست FPGA است که $x = 0000$ و $y = 0000$ است و چراغ 16 LED نیز روشن است که به معنی برابر بودن دو عدد است.



در عکس زیر تست FPGA است که $x = 000$ و $y = 0000$ است و چراغ LED 17 نیز روشن است که به معنی بزرگتر بودن عدد x است.



۳ پارت دوم

در این قسمت از مدار می‌خواهیم کد وریلگ یک مدار ترتیبی را بنویسیم به طوری که به صورت سریالی به آن ورودی دهیم و با هر دو بیت ورودی که می‌دهیم یک کلاک می‌زنیم. در این مدار ریست هم نیز داریم. برای این کار ما نیاز داریم تا بدانیم نتیجه‌ی مقایسه‌ی دو بیت قبلی داده شده به ماژول مان چی بوده است. آیا برابر بوده‌اند یا x بزرگ‌تر بوده یا y بزرگ‌تر بوده‌است.

به همین دلیل ما نیاز داریم که رجیستر یا به عبارتی flip flop داشته باشیم تا با توجه به ترتیبی بودن مدارمان بتوانیم تشخیص دهیم که عددمان بزرگ‌تر است یا کوچک‌تر.

در عکس زیر کد این ماژول را مشاهده می‌کنید.

```

1 module sequential_comparator (
2     input x,
3     input y,
4     input reset,
5     input clk,
6     output o_gt,
7     output o_lt
8 );
9     wire o_gt_not, o_lt_not, i_gt, i_lt;
10    // Create the flip flops
11    assign o_gt = ~(o_gt_not & ~(i_gt & clk));
12    assign o_gt_not = ~(o_gt & ~(i_gt & clk));
13    assign o_lt = ~(o_lt_not & ~(i_lt & clk));
14    assign o_lt_not = ~(o_lt & ~(i_lt & clk));
15    // Assign the inputs of flop flops
16    assign i_gt = (~reset) & (o_gt | ((~i_lt) & (x > y)));
17    assign i_lt = (~reset) & (o_lt | ((~i_gt) & (x < y)));
18 endmodule

```

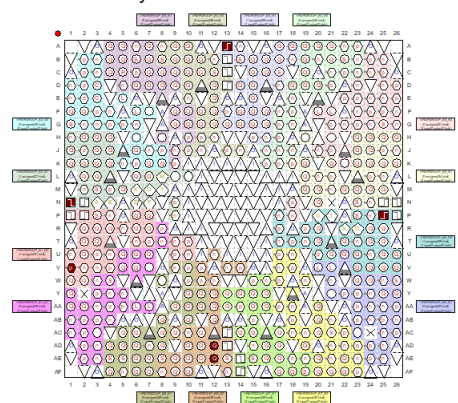
مطابق کد، زمانی که می‌خواهیم فلیپ‌فلاپ‌ها را مقداردهی کنیم مانند همان قسمت پارت اول است، با این تفاوت که در این جا چک می‌کنیم که ریست ما فعال نشده باشد و اگر فعال شده باشد فلیپ‌فلاپ‌ها را نیز ریست می‌کنیم. این کد نیز به صورت dataflow زده شده است.

در نهایت با توجه به مقدار فلیپ‌فلاپ‌ها خروجی نهایی یا همان o_gt و o_lt را مقداردهی می‌کنیم. همچنین مقدار کلاک را نیز در مقداردهی خروجی در نظر می‌گیریم.

۱.۳ FPGA

حال به ست کردن FPGA و ست کردن پین‌های ورودی و خروجی‌ها می‌پردازیم.

Top View - Wire Bond
Cyclone II - EP2C35F672C6



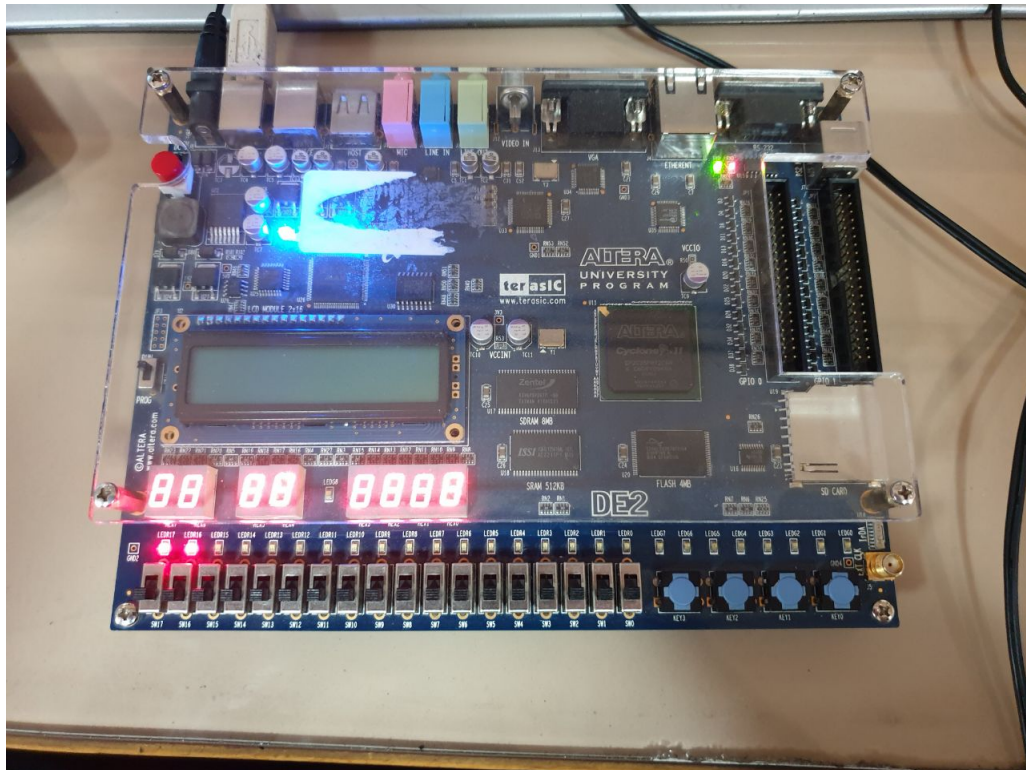
Node Name	Direction	Reserved	Location
clk	Input		PIN_P25
o_gt	Output		PIN_AD12
o_lt	Output		PIN_AE12
reset	Input		PIN_V1
x	Input		PIN_N1
y	Input		PIN_A13
<<new node>>			

	Node Name	Direction	Reserved	Location
1	clk	Input		PIN_P25
2	o_gt	Output		PIN_AD12
3	o_lt	Output		PIN_AE12
4	reset	Input		PIN_V1
5	x	Input		PIN_N1
6	y	Input		PIN_A13
7	<<new node>>			

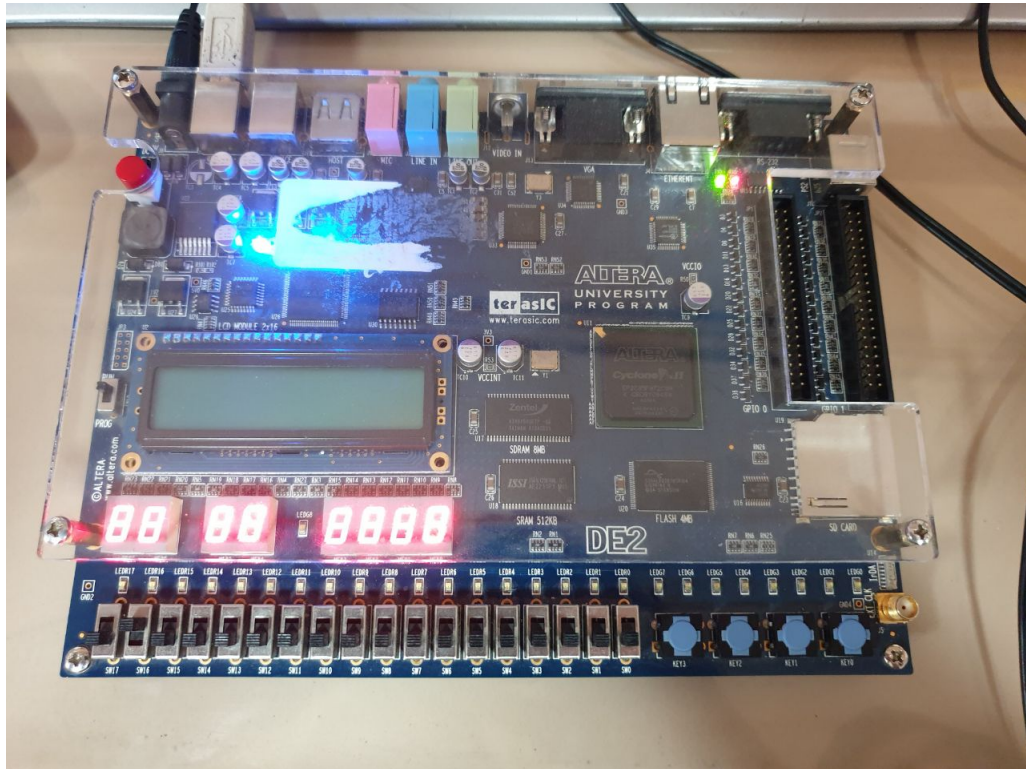
در این مدار، SW_۲ پین کلاک ما است و سویچ‌های SW_{۱۰} و SW_۹ به ترتیب بیت‌های x و y می‌باشند. SW_{۱۶} نیز سویچ ریست ما است. چراغ LED 17 به معنی بزرگ تر بودن و چراغ LED 16 به معنی کوچک بودن است.

۲.۳ تست FPGA

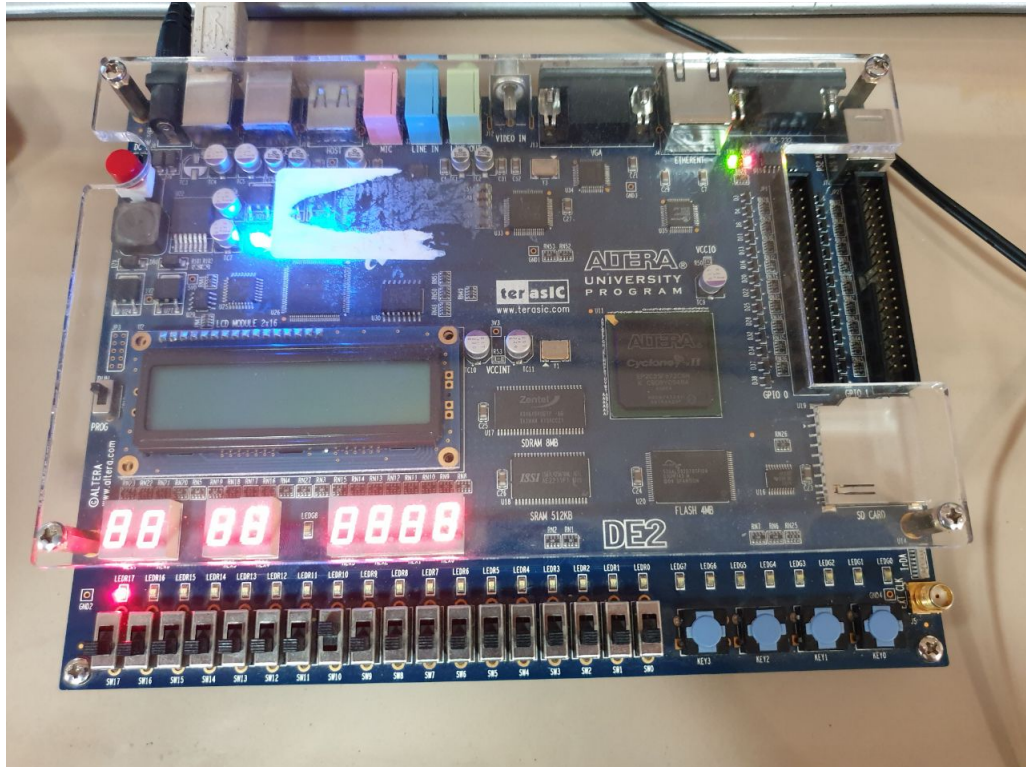
در عکس زیر لحظه‌ی ابتدایی FPGA را مشاهده می‌کنید که هر دو چراغ روشن است.



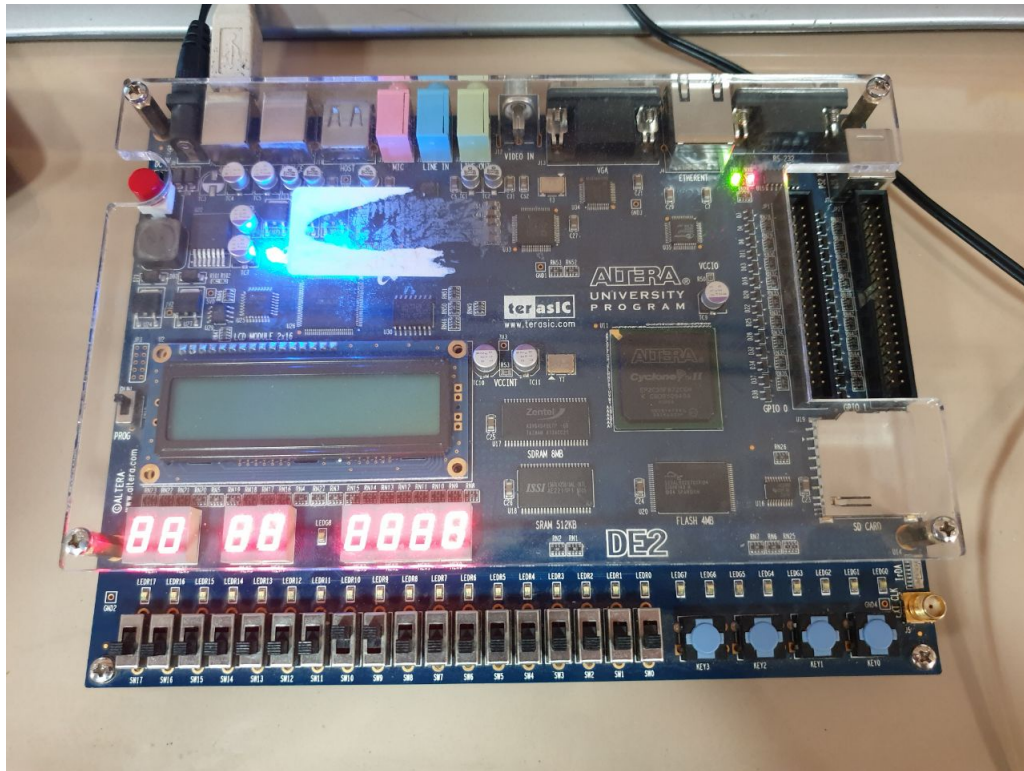
در عکس زیر زمانی که ریست را میزنیم و چراغ‌ها خاموش می‌شوند و یا به عبارتی خروجی‌ها ریست می‌شوند و صفر می‌شوند را مشاهده می‌کنید.



در عکس زیر تست FPGA زمانی که $x = 1$ و $y = 0$ است را مشاهده می‌کنید. همانطور که مشاهده می‌کنید چراغ LED 17 روشن است که به این معنی است که x بزرگ‌تر از y است.



در عکس زیر تست FPGA زمانی که $x = 1$ و $y = 1$ است را مشاهده می‌کنید. همانطور که مشاهده می‌کنید هر دو چراغ خاموش هستند که به این معنی است که دو عدد برابر هستند. (این تست مدار با بالایی متفاوت است و ما بعد از تست بالایی مدار را ریست کردیم.)



در عکس زیر تست FPGA زمانی که $x = 0$ و $y = 1$ است را مشاهده می‌کنید. همانطور که مشاهده می‌کنید هر چراغ LED 16 روشن است که به این معنی است که y از x بزرگ‌تر است. (این تست در ادامه‌ی تست بالا است و چون در قسمت بالا دو عدد برابر بودند و در اینجا y بزرگ‌تر است پس چراغ o_lt که یعنی x کوچک‌تر است روشن می‌شود.)

