گزارش تمرین ۸

فهرست

بخش ۱

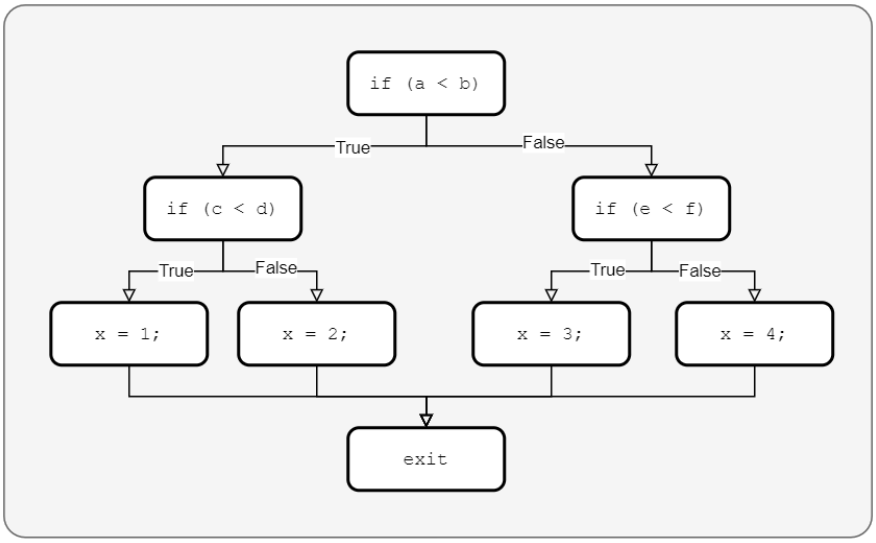
پیچیدگی سایکلوماتیک به دو روش محاسبه می‌شود.

$$CC = E - N + 2P$$

که E تعداد یال‌ها، N تعداد گره‌ها و P تعداد اجزاء گراف CDFG است.

$$CC = branch_{count} + 1$$

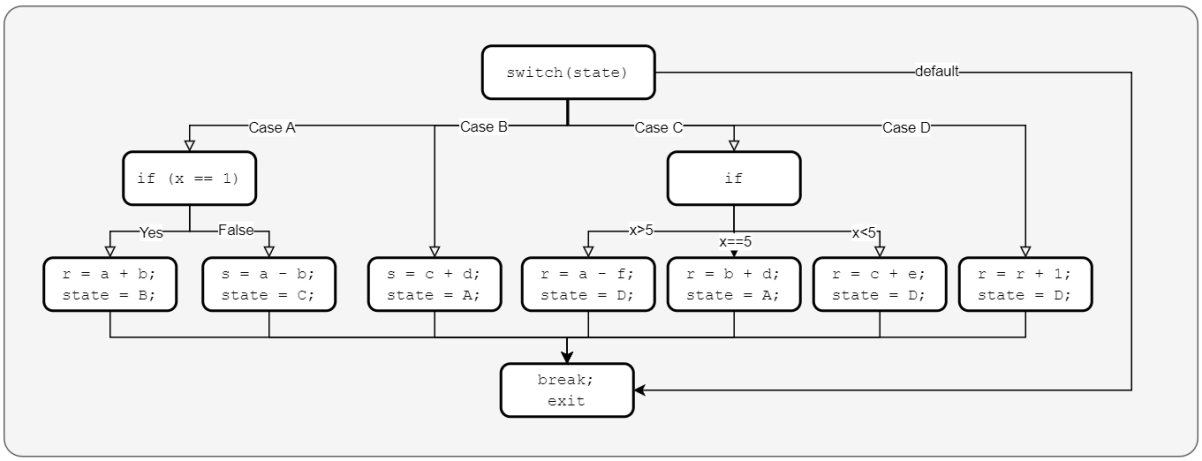
```
if (a < b) {
    if (c < d)
        x = 1;
    else
        x = 2;
} else {
    if (e < f)
        x = 3;
    else
        x = 4;
}
```



$$CC = 10 - 8 + 2 = 4$$

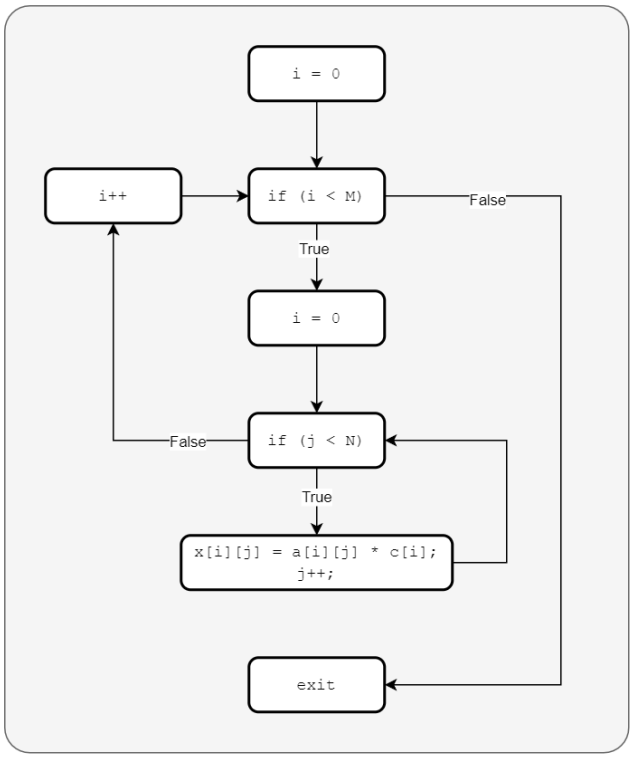
```
switch (state) {
    case A:
```

```
        if (x = 1) { r = a + b; state = B; }
        else { s = a e b; state = C; }
        break;
    case B:
        s = c + d;
        state = A;
        break;
    case C:
        if (x < 5) { r = a e f; state = D; }
        else if (x == 5) { r = b + d; state = A; }
        else { r = c + e; state = D; }
        break;
    case D:
        r = r + 1;
        state = D;
        break;
}
```



$$CC = 17 - 11 + 2 = 8$$

```
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        x[i][j] = a[i][j] * c[i];
```



$$CC = 8 - 7 + 2 = 3$$

بخش ۲

```
if (a < b || ptr1 == NULL) proc1();
else proc2();
```

Test Case	a < b	ptr1 == NULL	Outcome
1	True	True	proc1();
2	True	False	proc1();
3	False	True	proc1();
4	False	False	proc2();

```
switch (x) {
    case 0: proc1(); break;
    case 1: proc2(); break;
    case 2: proc3(); break;
    case 3: proc4(); break;
    default: dproc(); break;
}
```

Test Case	x	Outcome
1	0	proc0();
2	1	proc1();
3	2	proc2();
4	3	Proc3();
5	4	dproc();

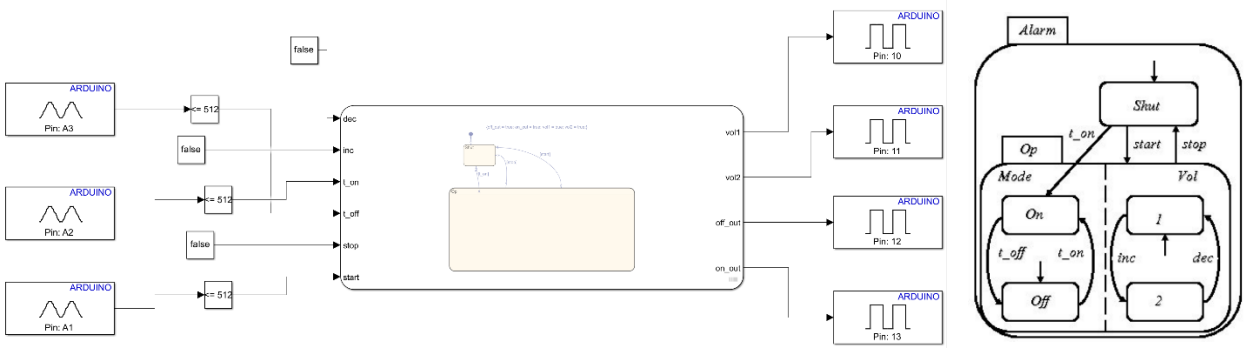
```
if (a < 5 && b > 7) proc1();
else if (a < 5) proc2();
else if (b > 7) proc3();
else proc4();
```

Test Case	a < 5	b > 7	Outcome
1	True	True	proc1();
2	True	False	proc2();
3	False	True	proc3();
4	False	False	Proc4();

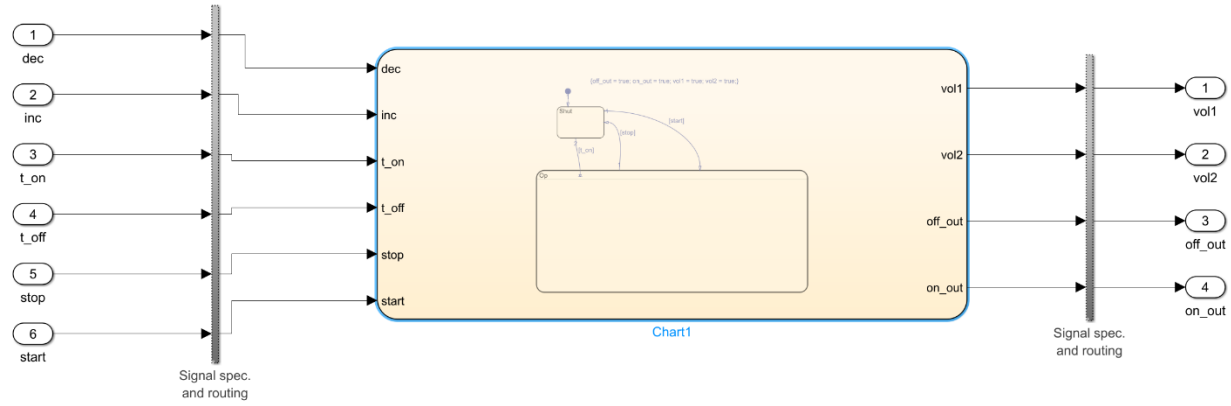
بخش ۳

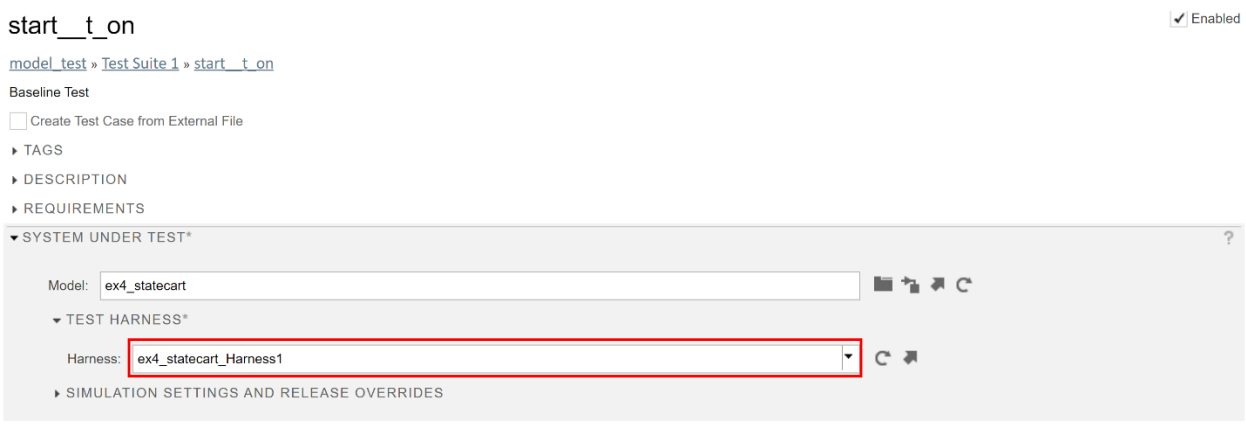
گزارش پوشش آزمون

مدل طراحی شده برای تمرین شماره ۴

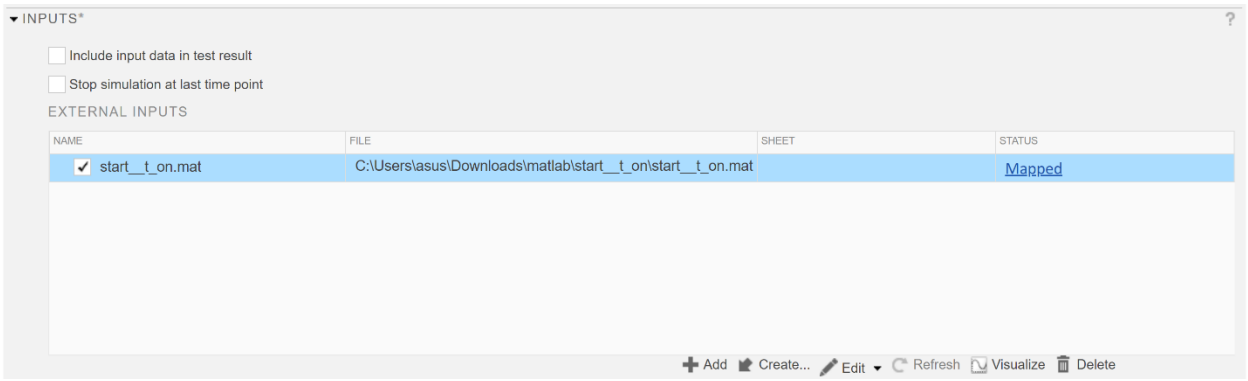


برای statechart یک مدل Harness ایجاد می‌کنیم. مدل Harness این کمک را به ما می‌کند تا بدون تغییر سیستم مورد نظر، ورودی‌ها و خروجی‌ها، یک مدل جداگانه مخصوص تست ایجاد کنیم.

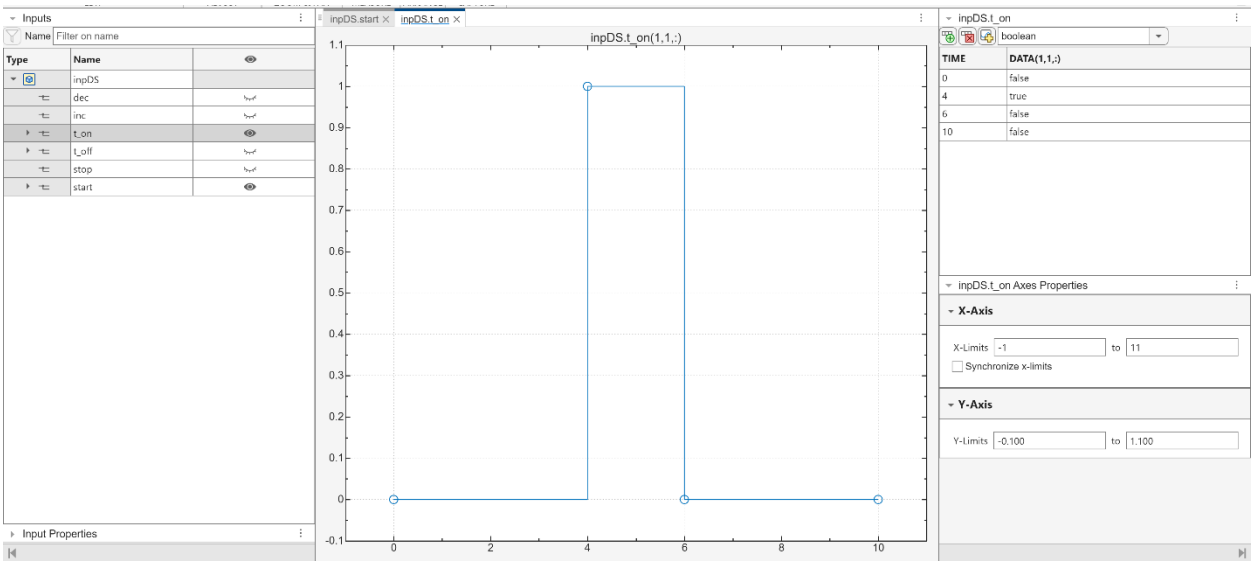


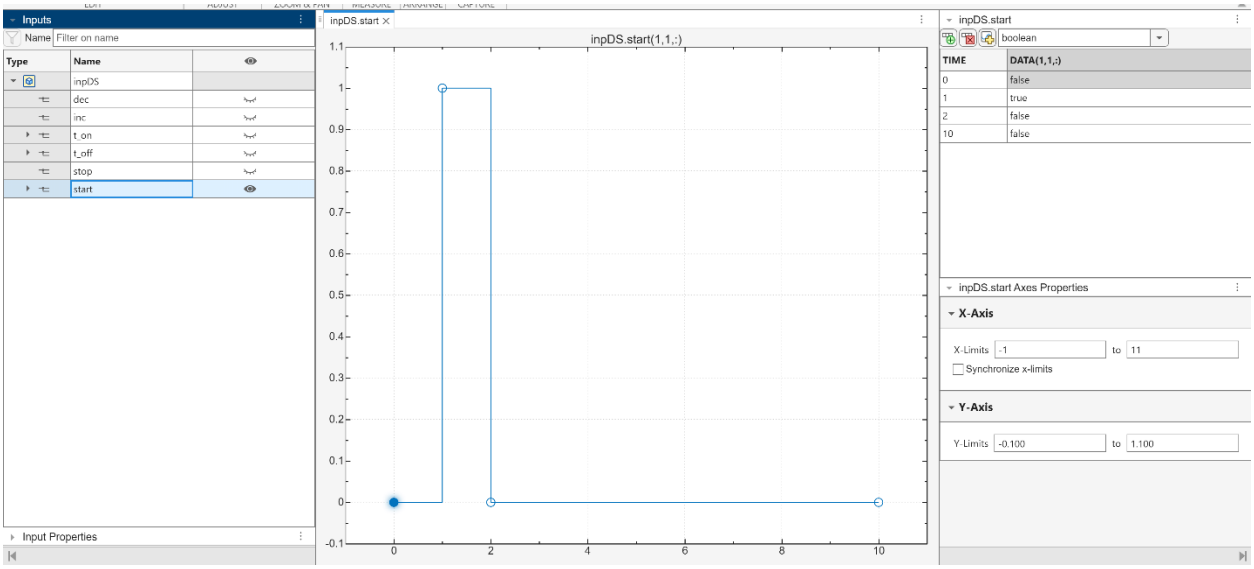


سپس در بخش inputs سیگنال‌های ورودی برای شبیه‌سازی یک سناریوی تست را ایجاد می‌کنیم.

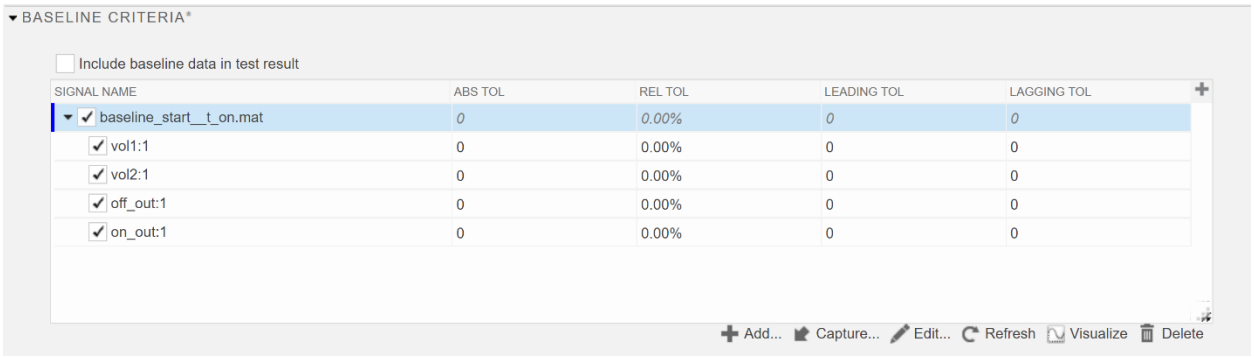


در این سناریو، ابتدا سیگنال start یک می‌شود و سیستم در وضعیت off قرار می‌گیرد. سپس با آمدن سیگنال t_on سیستم باید در وضعیت on قرار گیرد.

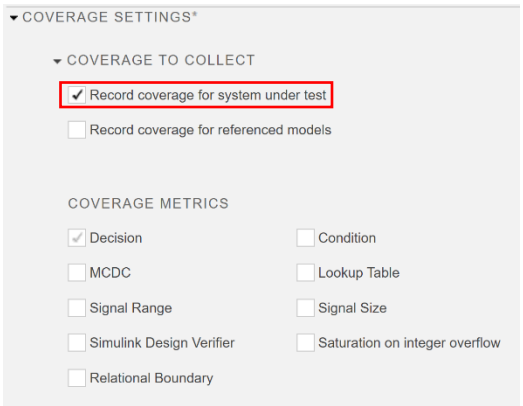




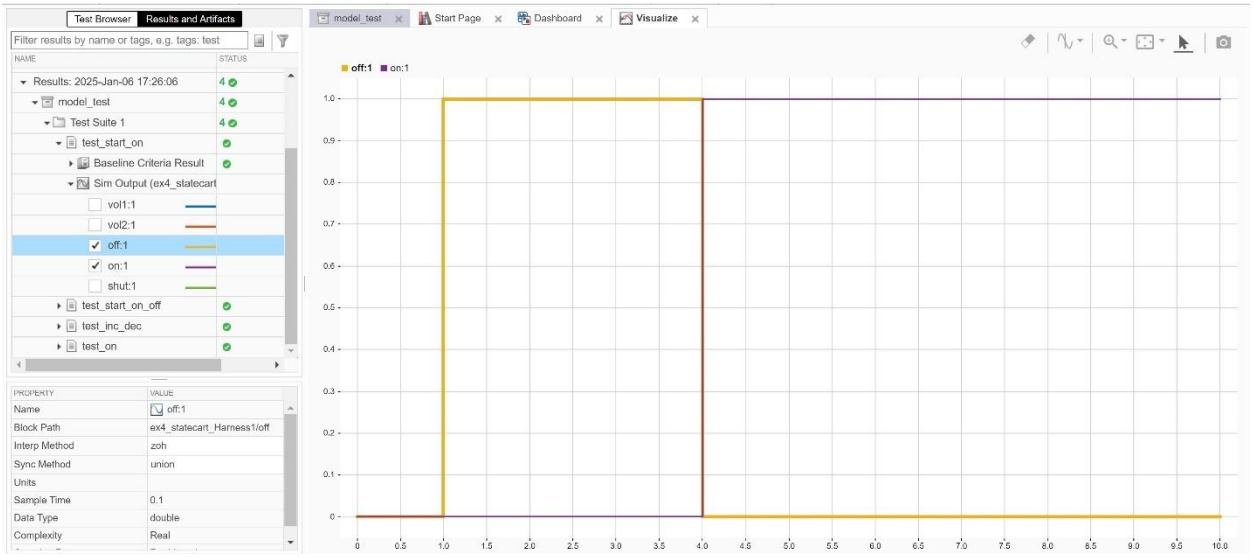
سپس خروجی مورد انتظار (Baseline) را تولید می‌کنیم.



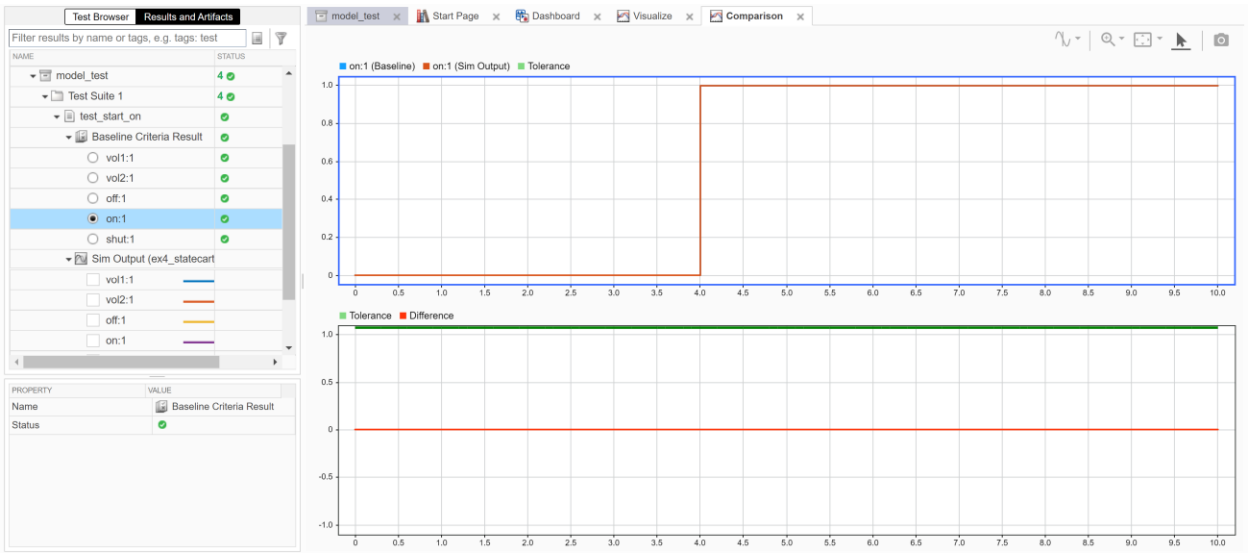
در مرحله پایانی پیکربندی تست، coverage را فعال می‌کنیم.



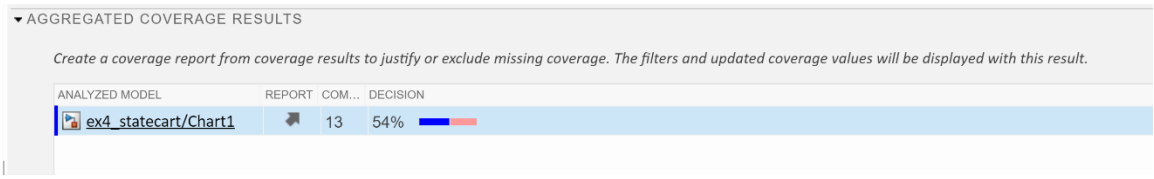
با اجرای تست‌ها، سیگنال‌های خروجی مورد انتظار حاصل می‌شود و تست پاس می‌شود. با آمدن سیگنال start خروجی off یک می‌شود و با آمدن سیگنال t_on خروجی off صفر و on یک می‌شود:



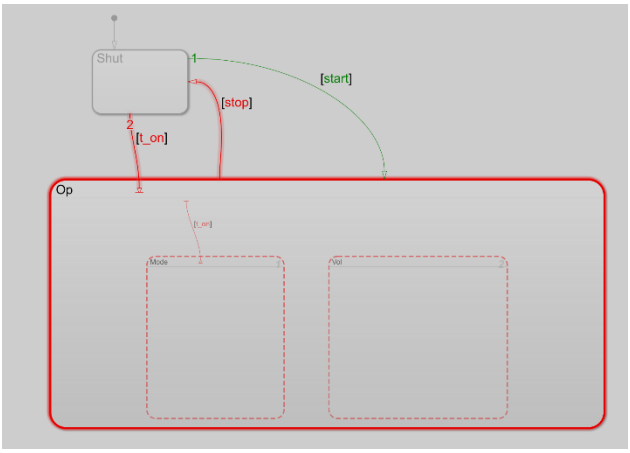
در نمودار حاصل انتظار می‌رفت در زمان یک شدن سیگنال t_on خروجی on یک شود که همین مورد در نمودار بالایی مشاهده می‌شود. در نمودار پایینی مشاهده می‌شود که Difference سیگنال تست شده با سیگنال مورد انتظار برابر بوده و تفاوتی نداشته‌اند.

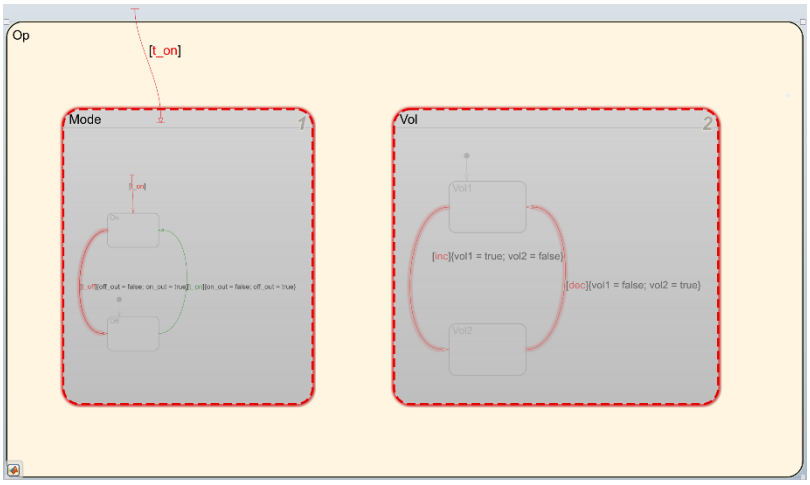


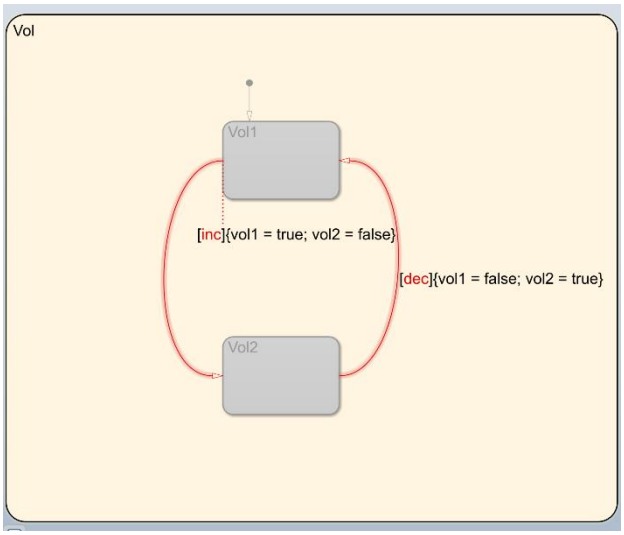
پوشش ۵۴ درصدی حاصل شده توسط تست:

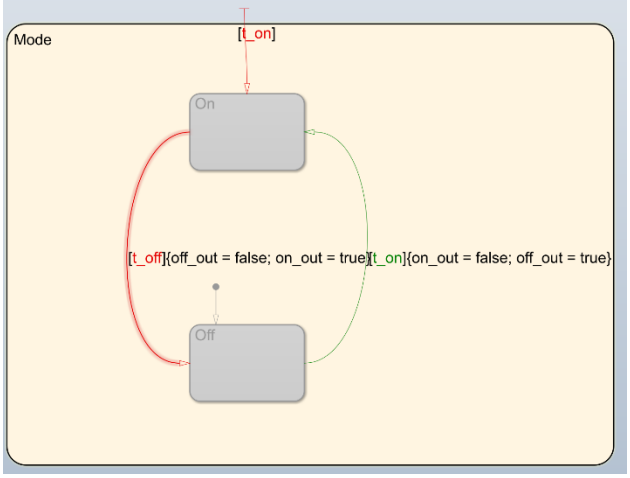


نمایش میزان پوشش با رنگ‌آمیزی statechart

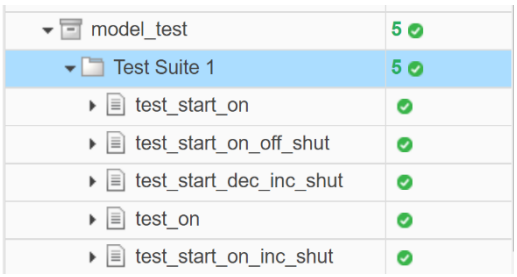






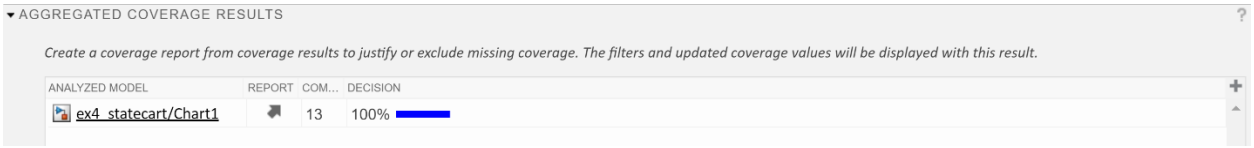


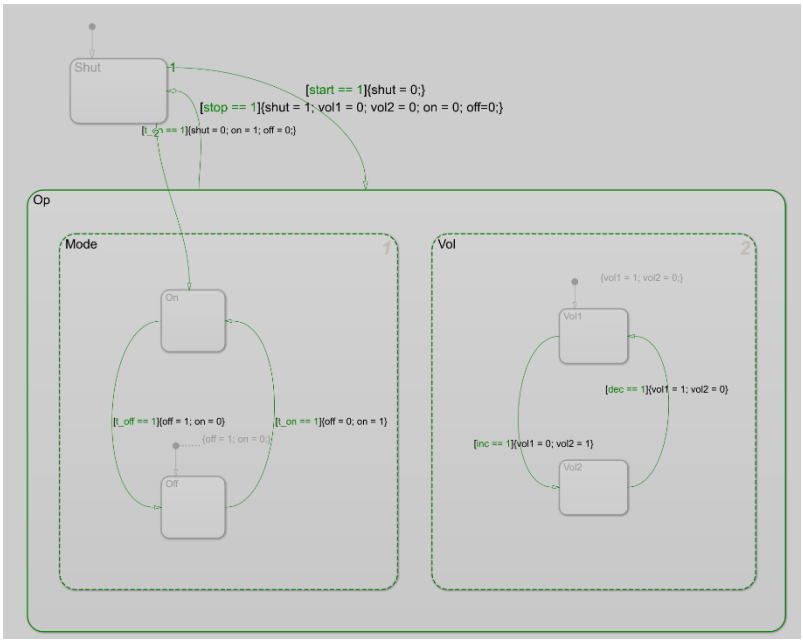
همانطور که از میزان پوشش یال‌ها مشخص است باید سناریوهای مختلفی برای تست اضافه شوند تا حداکثر پوشش حاصل شود. ۴ تست دیگر اضافه می‌شود:



تست	سناریو
test_on	این تست برای پوشش یال متصل کننده از حالت Shut به حالت On است.
test_start_on_off_shut	این تست برای پوشش یال‌های متصل کننده از حالت On به حالت Off و برعکس است. در نهایت زمانی که در حالت off قرار دارد با آمدن سیگنال shut به حالت Shut می‌رود.
test_start_dec_inc_shut	این تست برای پوشش حالت‌های Vol1 و Vol2 اضافه شده است. در نهایت زمانی که در حالت Vol1 قرار دارد با آمدن سیگنال shut به حالت Shut می‌رود.
test_start_on_inc_shut	حالتی که همزمان در حالت Vol2 و On باشد و با آمدن سیگنال shut به حالت Shut برود.

با اضافه شدن تست‌های جدید میزان پوشش به ۱۰۰ درصد می‌رسد:





آزمون بخش

ابتدا environment جدیدی با پلتفرم native می‌سازیم تا بر روی دستگاه خودمون انجام شود.

```
[env:native]
platform = native
lib_deps = throwtheswitch/Unity@^2.6.0
```

برای نوشتن تست‌ها از کتابخانه unity استفاده می‌کنیم.

در ویندوز توابع setUp و tearDown باید ابتدا هرچند خالی تعریف شوند. تابع setUp قبل از شروع تست‌ها و تابع tearDown پس از اجرای تست‌ها فراخوانی می‌شوند.

```
#include <unity.h>
#include <ex4.c>

void setUp(void) {
    // set stuff up here
    ex4_statecart_initialize();
    ex4_statecart_step();

}

void tearDown(void) {
    // clean stuff up here
}
```

ابتدا در تست اول مقادیر خروجی که بیان‌گر وضعیت‌های فعال است را بررسی می‌کنیم.

```
void test_initial(void)
{
    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.shut);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.off);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.on);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.vol1);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.vol2);
}
```

پس از فعال شدن start باید از وضعیت shut به وضعیت های vol1 و off برویم.

```
void test_start(void)
{
    ex4_statecart_U.start = 1;
    ex4_statecart_step();
    ex4_statecart_U.start = 0;


    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.shut);
```

```
    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.off);
    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.vol1);
}
```

پس از فعال شدن t_on وضعیت off به on تغییر می‌کند.

```
void test_t_on(void)
{
    ex4_statecart_U.t_on = 1;
    ex4_statecart_step();
    ex4_statecart_U.t_on = 0;

    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.off);
    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.on);
}
```

پس از فعال شدن inc وضعیت از vol1 به vol2 تغییر می‌کند.

```
void test_inc(void)
{
    ex4_statecart_U.inc = 1;
    ex4_statecart_step();
    ex4_statecart_U.inc = 0;

    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.vol1);
    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.vol2);
}
```

پس از فعال شدن dec وضعیت از vol2 به vol1 تغییر می‌کند.

```
void test_dec(void)
{
    ex4_statecart_U.dec = 1;
    ex4_statecart_step();
    ex4_statecart_U.dec = 0;

    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.vol1);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.vol2);
}
```

پس از فعال شدن stop تنها به وضعیت shut برمی‌گردیم.

```
void test_stop(void)
{
    ex4_statecart_U.stop = 1;
    ex4_statecart_step();
    ex4_statecart_U.stop = 0;

    TEST_ASSERT_EQUAL(1, ex4_statecart_Y.shut);
```

```
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.off);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.on);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.vol1);
    TEST_ASSERT_EQUAL(0, ex4_statecart_Y.vol2);
}
```

در انتها تست‌ها را در تابع main به صورت زیر اجرا می‌کنیم.

```
int main(int argc, char **argv)
{
    UNITY_BEGIN();

    RUN_TEST(test_initial);
    RUN_TEST(test_start);
    RUN_TEST(test_t_on);
    RUN_TEST(test_inc);
    RUN_TEST(test_dec);
    RUN_TEST(test_stop);

    UNITY_END();

    return 0;
}
```

نتایج تست به صورت زیر نشان داده می‌شوند.

```
Processing test_native in native environment
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
Building...
Testing...
test\test_native\test_main.c:84: test_initial   [PASSED]
test\test_native\test_main.c:85: test_start     [PASSED]
test\test_native\test_main.c:86: test_t_on      [PASSED]
test\test_native\test_main.c:87: test_inc       [PASSED]
test\test_native\test_main.c:88: test_dec       [PASSED]
test\test_native\test_main.c:89: test_stop      [PASSED]
----------------------------------------------------------- native:test_native [PASSED] Took 5.78 seconds -----------------------------------------------------------

============================================================================== SUMMARY ==============================================================================
Environment    Test         Status    Duration
-------------  -----------  --------  ------------
native         test_native  PASSED    00:00:05.778
================================================================ 6 test cases: 6 succeeded in 00:00:05.778 ================================================================
```

تحلیل ضریب اسپاگتی

ضریب اسپاگتی به کمک فرمول زیر محاسبه می‌شود:

$$SF = SCC + (Globals * 5) + (SLOC/20)$$

تحلیل انجام شده روی تابع ex4_statecart_step:

متغیرهای گلوبال:

```
/* Block states (default storage) */
DW_ex4_statecart_T ex4_statecart_DW;
/* External inputs (root inport signals with default storage) */
ExtU_ex4_statecart_T ex4_statecart_U;
/* External outputs (root outports fed by signals with default storage) */
ExtY_ex4_statecart_T ex4_statecart_Y;
```

در مجموع ۴۸ بار برای عملیات‌های خواندن و نوشتن از این متغیرهای گلوبال در کد استفاده شده است. در همین قسمت مشخص می‌شود که $48 \times 5 = 240$ بیشتر از ۱۰۰ است در نتیجه این کد در وضعیت **Nightmare** قرار دارد.

SCC برابر تعداد شرط‌ها (if) = ۱۱

تعداد خط‌های کد = ۵۷

$$SF = SCC + (Globals * 5) + (SLOC/20) \approx 11 + 240 + 2 = 253$$

بخش ۴ / امتیازی

Table 7.2 Memory characteristics

Memory	Size in bytes	Energy per access (nJ)
Scratchpad	4096 (4k)	1.3
Main memory	262,144 (256 k)	31

Also, let us assume that we are accessing variables as shown in the Table 7.3.

Table 7.3 Variable characteristics

Variable	Size in bytes	Number of accesses
a	1024	16
b	2048	1024
c	512	2048
d	256	512
e	128	256
f	1024	512
g	512	64
h	256	512

مسئله ما قرار دادن متغیرها در حافظه به گونه‌ای است که انرژی کل دسترسی‌ها کمینه شود.

هر کدام از متغیرهای a, b, c, d, e, f, g, h در معادله پایین وضعیت قرار گرفتن متغیر در اسکرچپد را مشخص می‌کند. ۱ به معنای قرارگیری در اسکرچپد و ۰ به معنای قرارگیری در حافظه اصلی است.

$$Energy_a = 16\big(1.3a + 31(1-a)\big)$$

$$Energy_b = 1024\big(1.3b + 31(1-b)\big)$$

$$Energy_c = 2048\big(1.3c + 31(1-c)\big)$$

$$Energy_d = 512\big(1.3c + 31(1-d)\big)$$

$$Energy_e = 256\big(1.3e + 31(1-e)\big)$$

$$Energy_f = 512\big(1.3f + 31(1-f)\big)$$

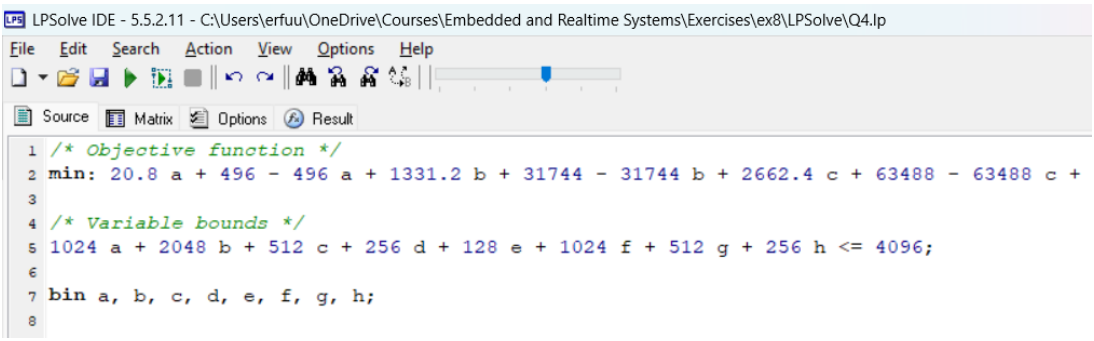
$$Energy_g = 64\big(1.3g + 31(1-g)\big)$$

$$Energy_h = 512\big(1.3h + 31(1-h)\big)$$

$$Energy = Energy_a + Energy_b + Energy_c + Energy_d + Energy_e + Energy_f + Energy_g + Energy_h$$

معادله بالا با رعایت شرط پایین باید کمینه شود.

$$1024a + 2048b + 512c + 256d + 128e + 1024f + 512g + 256h < 4096$$



Objective | Constraints | Sensitivity

Variables	MILP Feasible	result
	16406.4	16406.4
a	0	0
b	1	1
c	1	1
d	1	1
e	0	0
f	1	1
g	0	0
h	1	1

Objective | Constraints | Sensitivity

Constraints	MILP Feas...	result
	16406.4	16406.4
R1	4096	4096

$$Energy_{min} = 16406.4, \quad Scratchpad_{filled\ size} = 4096, \quad MainMemmory_{filled\ size} = 1664$$