Computer Architecture LAB Report

امیرحسین عالیان
4021120017

آزمایش چهارم، پنجم و ششم

تاریخ نوشتن توصیف آزمایش‌ها: ۲۴ آبان ۱۴۰۴

تاریخ نوشتن تست بنچ آزمایش‌ها: ۱ آذر ۱۴۰۴

تاریخ تحویل گزارش: ۱۳ آذر ۱۴۰۴

فهرست مطالب

۱ آزمایش چهار

۱.۱ هدف آزمایش

در این آزمایش هدف آشنایی با دستورات VHDL و همچنین نحوه پیاده سازی مدار به روش DataFlow بود.
در این آزمایش وظیفه ما پیاده سازی یک تابع با سه ورودی و دو خروجی بود که جدول درستی آن در زیر آمده است:

A	B	C	F_1	F_2
0	0	0	1	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

Table 1: جدول درستی تابع مطلوب

ابتدا از طریق جدول کارنو یا روش دلخواه دیگر تابع بولین F_1 و F_2 بررسی کردیم تا در صورت امکان ساده سازی انجام داده و ساده ترین فرم آن را بدست آوریم.

$$\mathbf{F_1 = \overline{A}.\overline{C} + B\overline{C} + \overline{A}B}$$

$$\mathbf{F_2 = \overline{A}.\overline{B} + AC}$$

سپس شروع به نوشتن توصیف سخت افزاری آن می‌کنیم.

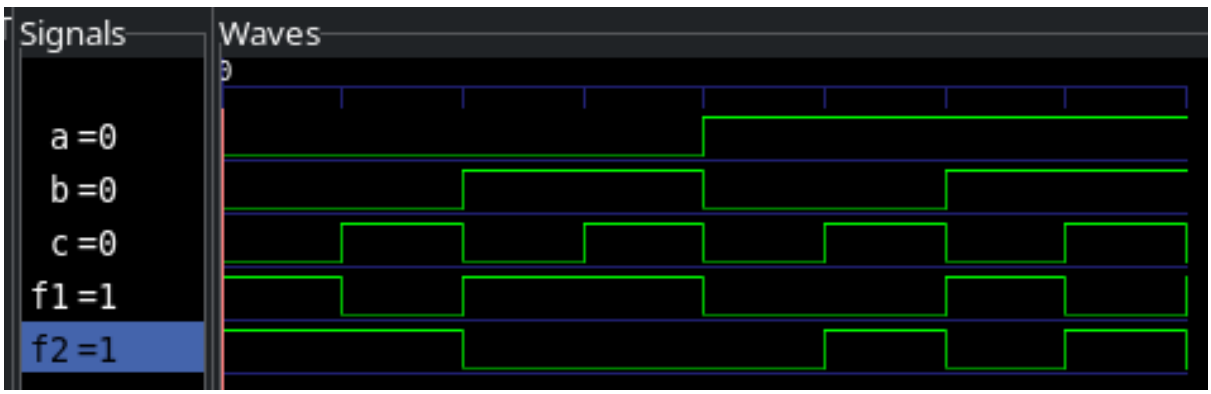
۲.۱ توصیف سخت افزار

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FUNCTION_3_TO_2 IS
    PORT(
        A, B, C : IN STD_LOGIC;
        F1, F2  : OUT STD_LOGIC
    );
END FUNCTION_3_TO_2;

ARCHITECTURE DataFlow OF FUNCTION_3_TO_2 IS
BEGIN
    F1 <= (NOT A AND NOT C) OR (B AND NOT C) OR (NOT A AND B);
    F2 <= (NOT A AND NOT B) OR (A AND C);
END DataFlow;
```

۳.۱ تست بنچ

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tb_FUNCTION_3_TO_2 IS
END tb_FUNCTION_3_TO_2;

ARCHITECTURE TestBench OF tb_FUNCTION_3_TO_2 IS
    SIGNAL a, b, c, f1, f2 : STD_LOGIC;
BEGIN
    UUT: ENTITY work.FUNCTION_3_TO_2 PORT MAP (a, b, c, f1, f2);
    a <= '0',
         '1' AFTER 4 ns;

    b <= '0',
         '1' AFTER 2 ns,
         '0' AFTER 4 ns,
         '1' AFTER 6 ns;

    c <= '0',
         '1' AFTER 1 ns,
         '0' AFTER 2 ns,
         '1' AFTER 3 ns,
         '0' AFTER 4 ns,
         '1' AFTER 5 ns,
         '0' AFTER 6 ns,
         '1' AFTER 7 ns,
         '0' AFTER 8 ns;
END TestBench;
```

۴.۱ نتیجه تست بنچ



شکل ۱: در این تست بنچ هر 8 حالت ترکیب باینری ممکن برای 3 بیت ورودی تولید شده است که شکل موج های خروجی یعنی F_1 و F_2 مطابق با جدول درستی داده شده است.

۲ آزمایش پنج

۱.۲ هدف آزمایش

در این آزمایش نیز هدف آشنایی با دستورات VHDL و همچنین نحوه پیاده سازی مدار به روش DataFlow بود.
در این آزمایش ما یک مالتی‌پلکسر ۲ به ۱ (MUX 2x1) پیاده سازی کردیم که شامل دو ورودی، یک خط کنترلی برای انتخاب و یک خط خروجی بود که جدول درستی آن در زیر آمده است:

S	I_1	I_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Table 2: جدول درستی مالتی‌پلکسر

به سادگی و حتی بدون نیاز به جدول کارنو می‌توان تابع بولین Y را بدست آورد که $\mathbf{Y = I_0\overline{S} + I_1S}$ است.
حال شروع به نوشتن توصیف سخت افزاری آن می‌کنیم.

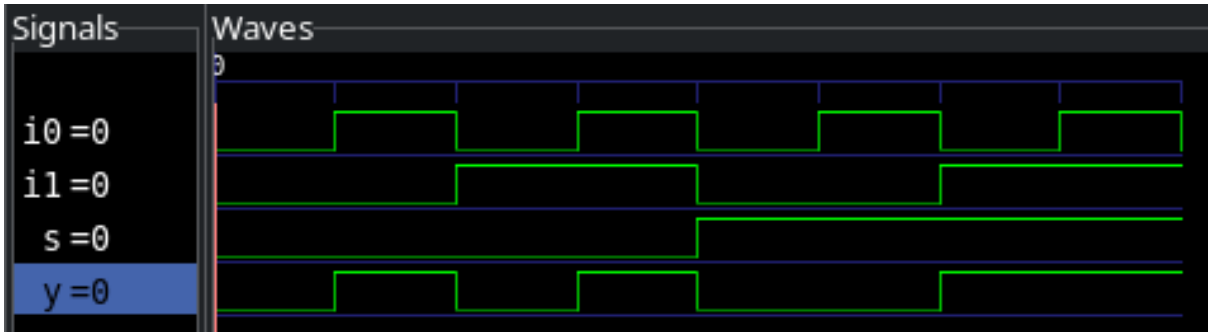
۲.۲ توصیف سخت افزار

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX_2_TO_1 IS
    PORT(
        I0 : IN STD_LOGIC;
        I1 : IN STD_LOGIC;
        S  : IN STD_LOGIC;
        Y  : OUT STD_LOGIC
    );
END MUX_2_TO_1;

ARCHITECTURE DataFlow OF MUX_2_TO_1 IS
BEGIN
    Y <= (NOT S AND I0) OR (S AND I1);
END DataFlow;
```

۳.۲ تست بنچ

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tb_MUX_2_TO_1 IS
END tb_MUX_2_TO_1;

ARCHITECTURE TestBench OF tb_MUX_2_TO_1 IS
    SIGNAL i0, i1, s, y : STD_LOGIC;
BEGIN
    UUT: ENTITY work.MUX_2_TO_1 PORT MAP (i0, i1, s, y);
    i0 <= '0',
          '1' AFTER 1 ns,
          '0' AFTER 2 ns,
          '1' AFTER 3 ns,
          '0' AFTER 4 ns,
          '1' AFTER 5 ns,
          '0' AFTER 6 ns,
          '1' AFTER 7 ns,
          '0' AFTER 8 ns;

    i1 <= '0',
          '1' AFTER 2 ns,
          '0' AFTER 4 ns,
          '1' AFTER 6 ns;

    s  <= '0',
          '1' AFTER 4 ns;
END TestBench;
```

۴.۲ نتیجه تست بنچ



شکل ۲: در این تست بنچ هر 8 حالت ترکیب باینری ممکن برای 3 بیت ورودی تولید شده است که شکل موج خروجی یعنی Y کاملا مطابق با جدول درستی داده شده است

۳ آزمایش شش

۱.۳ هدف آزمایش

در این آزمایش نیز هدف آشنایی با دستورات VHDL و همچنین نحوه پیاده سازی عناصر ترتیبی با کمک ساختار Process بود.
در این آزمایش ما یک فلیپ فلاپ (D Flip-flop) پیاده سازی کردیم که با لبه بالا رونده کلاک ورودی را ذخیره می‌کرد، همچنین شامل یک خط ورودی داده D یک خط ورودی آسنکرون $\overline{\text{reset}}$ در منطق Active-Low و دو خط خروجی Q و $\overline{\text{Q}}$ بود.

۲.۳ توصیف سخت افزار

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY D_FF IS
    PORT(
        clk       : IN STD_LOGIC;
        reset_not : IN STD_LOGIC;
        D         : IN STD_LOGIC;
        Q         : OUT STD_LOGIC;
        Q_NOT     : OUT STD_LOGIC
    );
END D_FF;

ARCHITECTURE Behavioural OF D_FF IS
BEGIN
    PROCESS (clk, reset_not)
    BEGIN
        IF reset_not = '0' THEN
            Q <= '0';
            Q_NOT <= '1';
        ELSIF rising_edge(clk) THEN
            Q <= D;
            Q_NOT <= NOT D;
        END IF;
    END PROCESS;
END Behavioural;
```

نکته: با توجه به ترتیب نوشتن شرط IF، در حالتی که خطوط D و $\overline{\text{reset}}$ بخواهند مقادیر متفاوتی را به Q انتساب دهند، اولویت با خط $\overline{\text{reset}}$ خواهد بود و این خط مقدار Q را مشخص می‌کند.

این نحوه پیاده سازی با توجه به اینکه خط $\overline{\text{reset}}$ از نوع آسنکرون در نظر گرفته شده است یک پیاده سازی معقول است.

۳.۳ تست بنچ

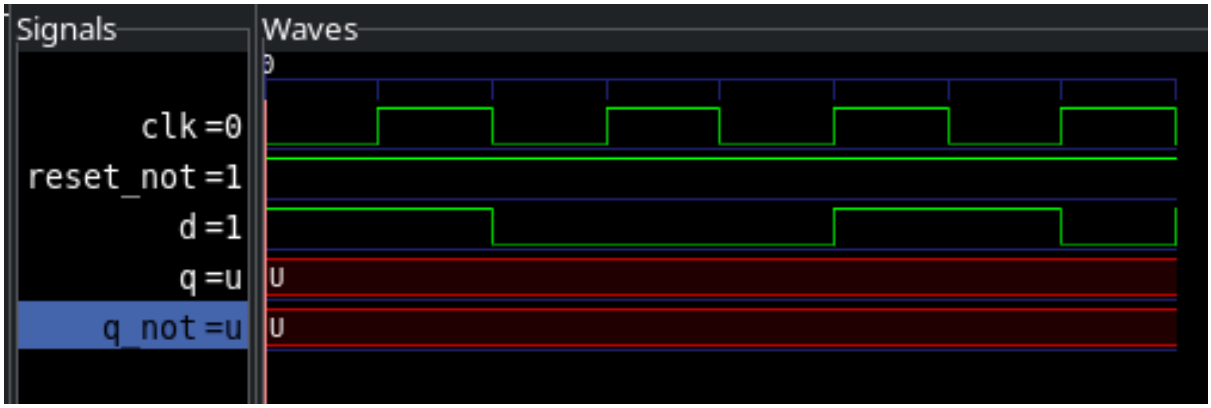
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tb_D_FF IS
END tb_D_FF;

ARCHITECTURE tb OF tb_D_FF IS
    SIGNAL clk, reset_not, D, Q, Q_NOT : STD_LOGIC;
BEGIN
    UUT: ENTITY work.D_FF PORT MAP (clk, reset_not, D, Q, Q_NOT);

    clk <= '0',
           '1' AFTER 1 ns,
           '0' AFTER 2 ns,
           '1' AFTER 3 ns,
           '0' AFTER 4 ns,
           '1' AFTER 5 ns,
           '0' AFTER 6 ns,
           '1' AFTER 7 ns,
           '0' AFTER 8 ns;

    reset_not <= '0',
                 '1' AFTER 1 ns;
    D <= '1',
         '0' AFTER 2 ns,
         '1' AFTER 5 ns,
         '0' AFTER 7 ns,
         '1' AFTER 8 ns;
END tb;
```

۴.۳ نتیجه تست بنچ

شکل ۳: در این تست تلاش شده چندین وضعیت خاص پوشش داده شوند برای مثال حالتی که دو خط D و $\overline{\text{reset}}$ تلاش کرده اند که دو مقدار متفاوت را به Q بدهند یا حالتی که مقدار D تغییر می‌کند اما بدلیل نبود لبه بالا رونده کلاک، این تغییر تا سیکل بعد اعمال نمی‌شود.

۵.۳ یک تجربه در شبیه سازی این آزمایش

در ابتدا کدی برای توصیف سخت افزار این آزمایش نوشته شد که عملکرد خط $\overline{reset}$ در یک Process و عملکرد خط D در یک Process دیگر قرار داشت به عبارت دیگر به ازای هر کدام یک Process جداگانه نوشته شده بود که منجر به شکل موج زیر می‌شد:



شکل ۴: نتیجه تست بنچ در حالتی که خطوط $\overline{reset}$ و D را در Process های جداگانه تعریف کرده بودیم.

پس از تحقیق و بررسی مشخص شد که توصیف عملکرد این دو خط در Process های جداکانه کار اشتباهی است زیرا هر دوی آنها به صورت همزمان در تلاش برای انتساب مقدار به خط Q بودند. در نتیجه توصیف آنها به یک بلوک Process کاهش یافت که همان توصیف نوشته شده در بخش ۲.۳ است.

۴ مراحل کامپایل و اجرای شبیه‌سازی در GHDL

ابتدا باید فایل های VHDL تحلیل شوند. این کار را با دستور زیر انجام می‌دهیم:

```
ghdl -a design.vhd       #Analysis
ghdl -a testbench.vhd    #Analysis
```

بعد از تحلیل باید Entity مربوط به تست‌بنچ را بسازیم:

```
ghdl -e testbench_entity_name    #Elaboration
```

حالا می‌توان شبیه سازی را اجرا کرد. دستور زیر یک فایل موج با نام wave.ghw ایجاد می‌کند.

```
ghdl -r testbench_entity_name --wave=wave.ghw    #Run
```

مشاهده waveform در نرم افزار gtkwave:

```
gtkwave wave.ghw
```

۵ نرم افزار های مورد استفاده و منابع

نرم افزار ها
توصیف کد این آزمایش‌ها در کد ادیتور **Vim** و در محیط سیستم عامل **GNU/Linux** و در توزیع **Debian 13** صورت گرفته است. همچنین کامپایل و شبیه سازی کد توصیف‌ها و تست بنچ‌ها با نرم افزار **GHDL Version 6** و مشاهده موج های شبیه سازی در نرم افزار **gtkwave** انجام شده است.

منابع
۱- پیاده سازی عناصر ترتیبی - allaboutcircuits
۲- راهنمای سریع کار با نرم افزار GHDL
۳- مقاله پیاده سازی سیگنال ریست به صورت آسنکرون و سنکرون - سایت فراد اندیش