🚜 Vmake ASM Chart Compiler

Like Cmake but for V

درس طراحی سیستم‌های دیجیتال
دکتر امین فصحتی

> *Alireza Habibzadeh* `99109393`
> *Amirreza Ghadyani* `99109206`

> *July 2022*

معرفی

کامپایلر Vmake ASM Chart یک کامپایلر ساده است که با پایتون پیاده شده و با دریافت یک ASM chart به شکل یک گراف، یک module وریلاگ پیاده می‌کند که آن ASM chart را شبیه‌سازی می‌کند.

منطق کلی کامپایلر بر پایه‌ی یک ماشین `n_state` ،`p_state` است که با هر تغییر ورودی یا `p_state`، ماشین، `n_state` یعنی حالت بعدی خود را تعیین (پیدا) می‌کند. کد زیر که توسط برنامه تولید شده این منطق را نشان می‌دهد:

```
module main(
        input clock,
        input reset
);
reg [0:0] p_state, n_state;

always @(p_state) begin
case (p_state)
0: begin
n_state = 0;
end
endcase
always @(posedge clock) begin
        if (reset) p_state = 0;
        else p_state = n_state;
end
endmodule
```

برای آشنایی با نحوه‌ی ورودی دادن و کار با برنامه می‌توانید فایل `tests.py` که در پروژه قرار دارد را ببینید. در این فایل، نمونه‌هایی که در انتهای این گزارش می‌بینیم پیاده شده‌اند.

امکانات

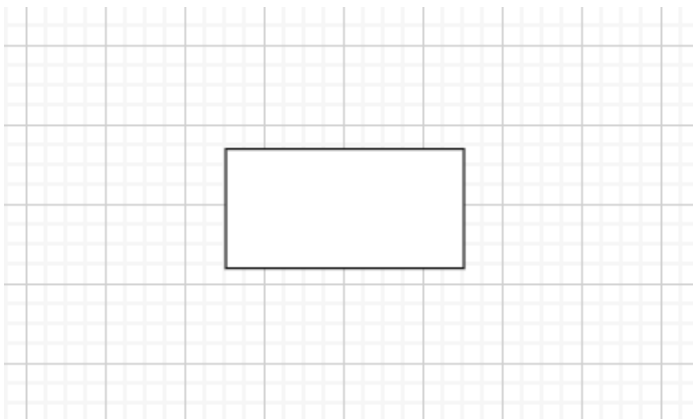
- امکان صدا زده شدن با پارامتری برای negedge بودن `clock`
- امکان صدا زده شدن با پارامتری برای negedge بودن `reset`
- امکان صدا زده شدن با پارامتری برای asynchronous بودن `reset`
- تشخیص کاراکترهای غیرمجاز مانند “,” و “;” در نام ورودی/خروجی‌ها و اعلام آن
- تشخیص اسامی غیرمجاز مانند begin و end و if در نام ورودی/خروجی‌ها و اعلام آن
- امکان استفاده از inout
- از آن‌جایی که parse شدن chart و پیاده‌سازی آن به صورت بازگشتی پیاده شده، امکان استفاده از شرط‌های تو در تو و به طور خلاصه هر نوع ترکیبی از و Decision boxها، Condition boxها و State boxها وجود دارد. تا وقتی که chartما از نظر قوانین طراحی ASM chart مجاز باشد شرط‌ها و نتایج آن‌ها به صورت تودرتو پیاده می‌شوند.

تست‌ها و نمونه‌کار

برای کامپایلر در ابتدا چند تست ساده تعریف شده و در انتها به عنوان نمونه **ASM chart تمرین شماره 5 درس** توسط برنامه به کد وریلاگ تبدیل شده است. تمامی کدهای خروجی برنامه عینا و بدون تغییر آورده شده‌اند.

Do Nothing Test

تست خالی برای چک کردن اجرایی بودن ساختار کلی برنامه و آشنا شدن با آن.

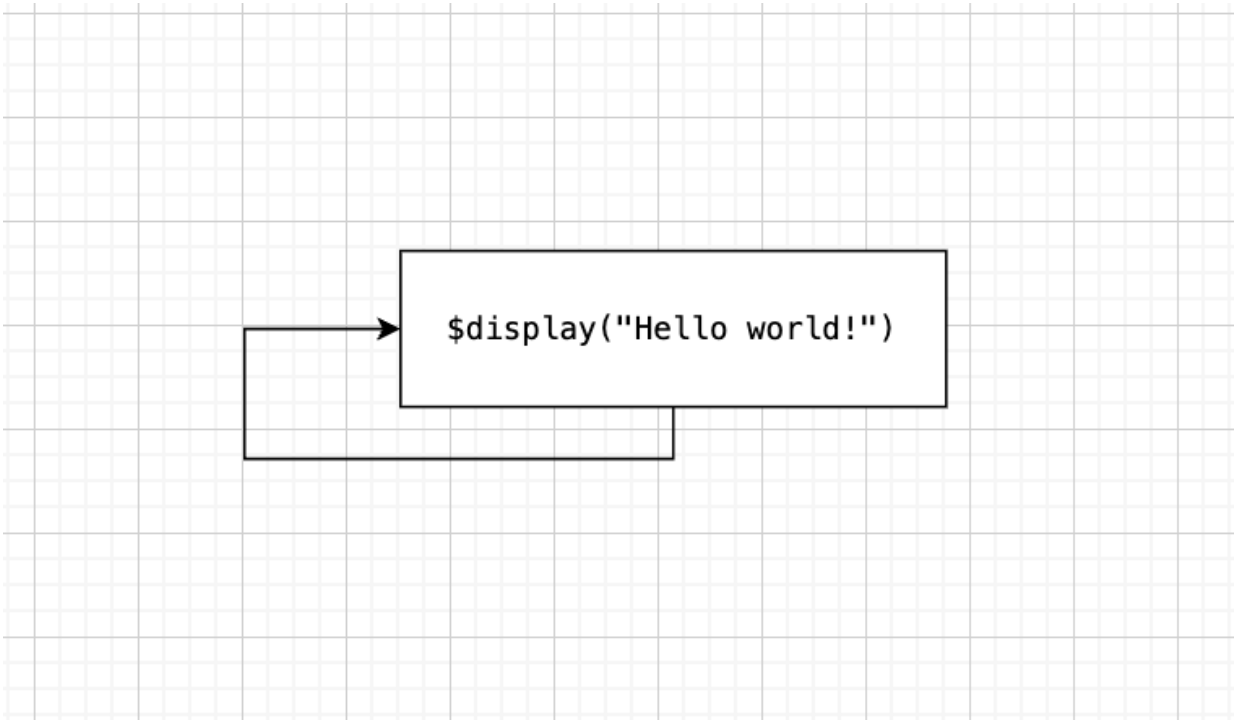


```
module main(
        input clock,
        input reset
);
```

```
reg [0:0] p_state, n_state;

always @(p_state) begin
case (p_state)
0: begin
n_state = 0;
end
endcase
always @(posedge clock) begin
        if (reset) p_state = 0;
        else p_state = n_state;
end
endmodule
```

Hello world!




```
module main(
        input clock,
        input reset
);
reg [0:0] p_state, n_state;
```

```
6
7  always @(p_state) begin
8  case (p_state)
9  0: begin
10 $display("Hello world!");
11 n_state = 0;
12 end
13 endcase
14 always @(posedge clock) begin
15         if (reset) p_state = 0;
16         else p_state = n_state;
17 end
18 endmodule
```

Hello world! negedge async reset

همان چارت قبلی اما این بار کامپایلر با تنظیمات `async_reset=True, negedge_reset=True,` صدا زده شده. `negedge_clock=True`

```
1  module main(
2          input clock,
3          input reset
4  );
5  reg [0:0] p_state, n_state;
6
7  always @(p_state) begin
8  case (p_state)
9  0: begin
10 $display("Hello negedge-low and asynchronous world!");
11 n_state = 0;
12 end
13 endcase
14 always @(negedge clockor negedge) begin
15         if (~reset) p_state = 0;
16         else p_state = n_state;
17 end
18 endmodule
```

Illegal name 1

تست اسم غیرمجاز

```
vmake(root, input_list=['p_state'])
```

```
ValueError: p_state is a forbidden name
```

Illegal name 2

تست کاراکتر غیرمجاز در اسم

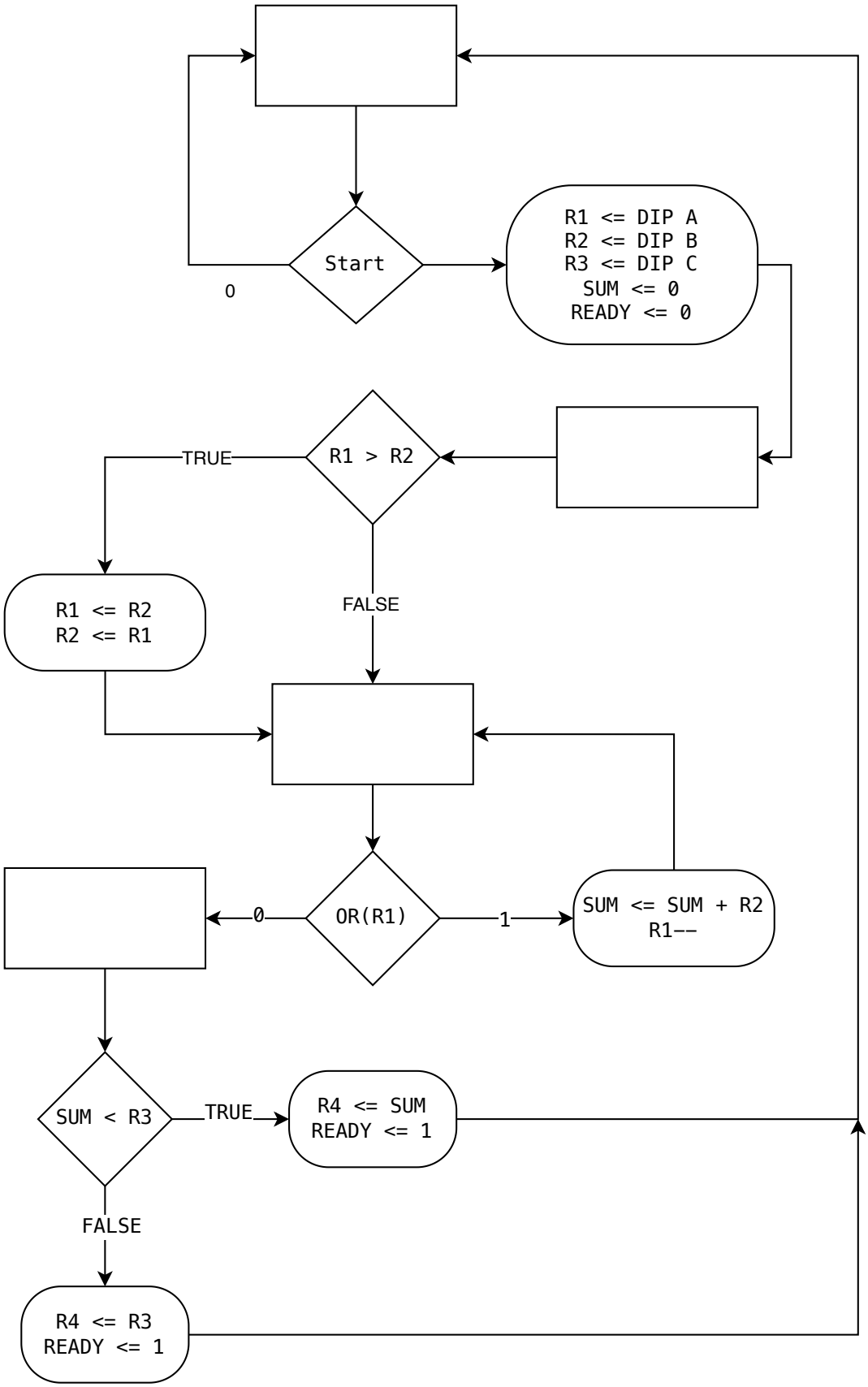
```
vmake(root, input_list=['my;input'])
```

```
ValueError: my;input contains a forbidden character ";"
```

Homework 5

چارت تمرین 5 درس که سپس توسط برنامه پیاده شده.

DSD – HW5
Alireza Habibzadeh 99109393



```
module main(
        output reg [3:0] R1,
        output reg [3:0] R2,
        output reg [3:0] R3,
        output reg [3:0] SUM,
        output reg READY,
        input [3:0] DIP_A,
        input [3:0] DIP_B,
        input [3:0] DIP_C,
        input clock,
        input reset
);
reg [2:0] p_state, n_state;

always @(p_state or DIP_A or DIP_B or DIP_C) begin
case (p_state)
0: begin
if (START == 1) begin
R1 <= DIP_A;
R2 <= DIP_B;
R3 <= DIP_C;
SUM <= 0;
READY <= 0;
n_state = 1;
end else begin
n_state = 0;
end
end
1: begin
if (R1 > R2) begin
R1 <= R2;
R2 <= R1;
n_state = 2;
end else begin
n_state = 2;
end
```

```
end
2: begin
if (OR(R1)) begin
SUM <= SUM + R2;
R1 <= R1 - 1;
n_state = 2;
end else begin
n_state = 3;
end
end
3: begin
if (SUM < R3) begin
R4 <= SUM;
READY <= 1;
n_state = 0;
end else begin
R4 <= R3;
READY <= 1;
n_state = 0;
end
end
endcase
always @(posedge clock) begin
        if (reset) p_state = 0;
        else p_state = n_state;
end
endmodule
```

Roadmap

در آینده می‌توان موارد زیر را در پروژه بهبود بخشید یا اضافه کرد:

رابطه‌ی گرافیکی (GUI)
طبعا بهتر است در آینده یک رابط گرافیکی درست کنیم تا با کامپایلر ما ارتباط برقرار کند و کاربر مستقیما به شکل گرافیکی ASM chart را در نرم‌افزار بکشد. البته یک راه ساده‌تر و اتفاقا بهتر این است که برنامه خروجی‌های استاندارد ASM chart را بتواند به عنوان ورودی دریافت و parse کند. برای مثال کاربر در سایت https://app.diagrams.net یا نرم‌افزار VIsio چارت را می‌کشد و سپس خروجی آن را در برنامه‌ی ما import می‌کند.

تورفتگی‌ها یا Indentation درست

هرچند در وریلاگ intentation از نظر کارکرد اهمیتی ندارد، اما بهتر است در آینده برای خوانایی بهتر کد خروجی توسط انسان، کامپایلر ما indentationها را در همه‌ی بخش‌ها به درستی رعایت کند. اکنون هم در بعضی جاها رعایت می‌کند و در بعضی جاها خیر.

تشخیص خودکار رجیسترها

اکنون کامپایلر آنقدر هوشمند نیست که بتواند خودش رجیسترهایی که در ASM chart استفاده شده‌اند را تشخیص داده و تعریف کند. این تعریف باید توسط کاربر در هنگام صدا زدن کامپایلر انجام شود. در آینده می‌توان این ویژگی را اضافه کرد که متغییرهایی که در ASM chart به آن‌ها assignment صورت می‌گیرند به طور خودکار به عنوان یک رجیستر داخلی تعریف شوند.