

سوال ۶. واحد محاسبات سریال

طراحی سیستم‌های دیجیتال
آزمون میانترم بخش Take-home

Alireza Habibzadeh 99109393

May 28, 2022

طراحی ASM Chart

هرچند ASM Chart برنامه ساده است و می‌توان آن را در ذهن یا مستقیماً روی کد پیاده کرد اما برای دقت بیشتر نموداری که بتواند به درستی ورودی‌های مدار را به ترتیب دریافت کند و در همان حال خروجی‌های حالت قبلی را بدهد پیاده می‌کنیم.

در این پیاده‌سازی چهار state زیر در نظر گرفته شده‌اند:

حالت status1

در این حالت کد status اول که همان بیت parity است خروجی داده می‌شود. به طور هم‌زمان بیت `[opcode[1]` هم برای محاسبه‌ی بعدی ورودی گرفته می‌شود.

حالت status2

در این حالت کد status دوم که بیت چک صفر بودن خروجی است داده می‌شود. به طور هم‌زمان بیت `[opcode[0]` هم برای محاسبه‌ی بعدی ورودی گرفته می‌شود.

حالت data1

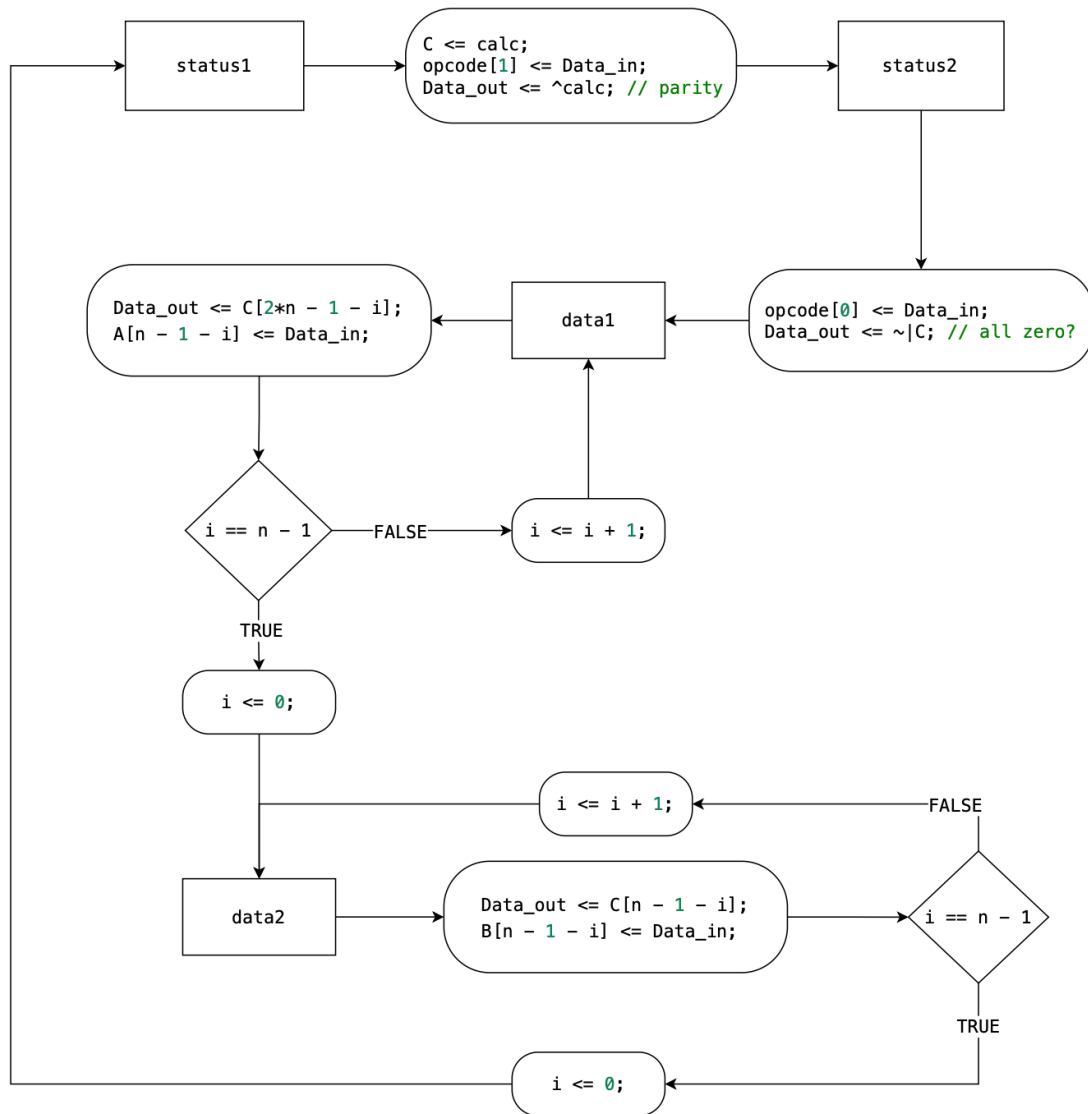
در این حالت که n بار اتفاق می‌افتد هر بار یک بیت از `A` (یکی از عملوندها) را ورودی می‌گیریم. همچنین هر بار هم‌زمان یک بیت از `C` که مربوط به محاسبه‌ی قبلی است خروجی می‌دهیم.

حالت data2

در این حالت که باز n بار اتفاق می‌افتد هر بار یک بیت از `B` (عملوند دیگر) را ورودی می‌گیریم. همچنین هر بار هم‌زمان یک بیت از `C` را خروجی می‌دهیم که با این n بیت کل $2n$ بیت `C` تمام می‌شود.

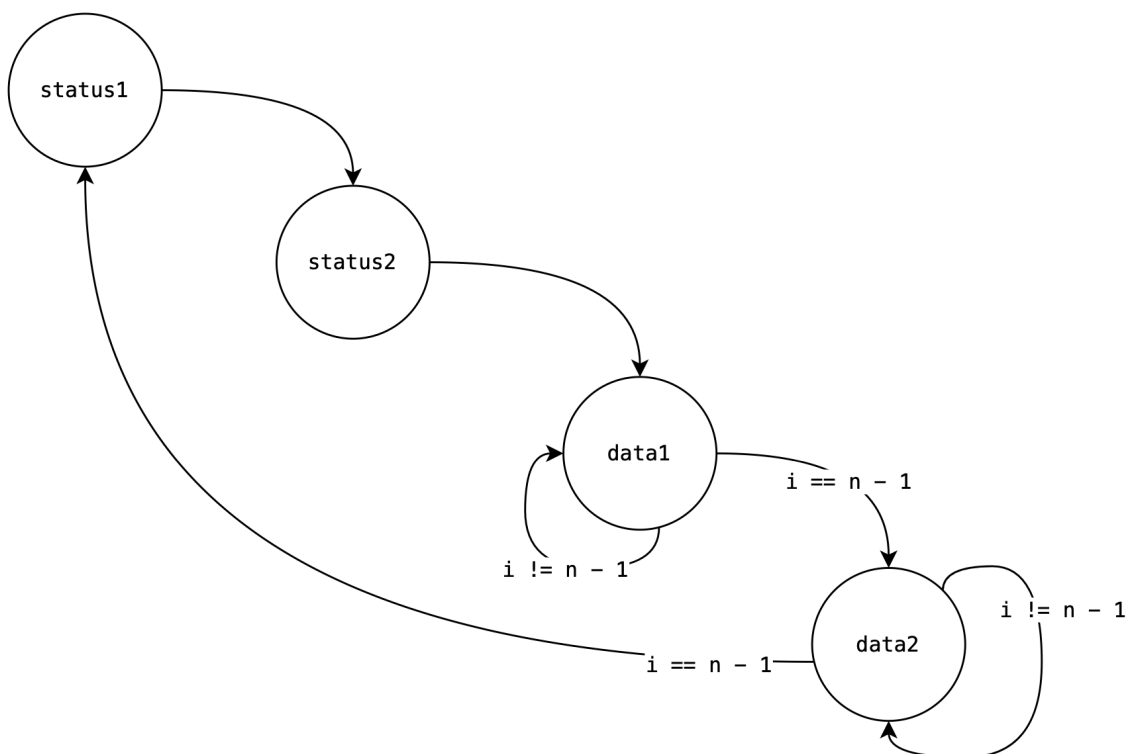
در پیاده‌سازی باید به این نکته توجه کنیم که اعداد باینری در این قطعه‌ی سریال از MBS به LSB داده می‌شوند.

طراحی نهایی ASM Chart در انتهای این فایل و همچنین به صورت ضمیمه در فایل زیپ پیوست شده است.



نمودار حالات

از روی ASM Chart قسمت قبل می‌توانیم یک نمودار حالت ساده (State Diagram) طراحی کنیم تا stateهای مختلف طراحی را بهتر درک کنیم. این نمودار هم در انتهای این فایل و همچنین به صورت ضمیمه در فایل زیپ آمده است.



پیاده‌سازی

پارامتر n

برای n که یک پارامتر مدار است از کد زیر استفاده می‌کنیم تا بتوان به سادگی آن را تغییر داد و n در همه‌جای کد تغییر کند:

```
parameter n = 2;
```

پارامترهای local

برای سادگی کد و همچنین خوانایی آن از پارامترهای زیر استفاده شده است. یکی از سری پارامترها برای opcode و دیگری برای state‌های مدار است.

```

1 localparam [1:0] sum_op = 2'b10, sub_op = 2'b01, mul_op = 2'b11, div_op = 2'b00;
2 localparam [1:0] status1 = 2'b00, status2 = 2'b01, data1 = 2'b10, data2 = 2'b11;
  
```

قسمت جریان داده (dataflow)

برای توصیف محاسبات مدار از یک بخش به صورت dataflow modeling استفاده شده است:


```

19                                     opcode == mul_op ? A * B : A /
B; // implicit assignment
20     always @(negedge Clock or posedge Reset) begin
21         if (Reset) begin
22             A <= 0;
23             B <= 0;
24             C <= 0;
25             i <= 0;
26             opcode <= 0;
27             Data_out <= 0;
28             state <= status1;
29         end else begin
30             case (state)
31                 status1: begin
32                     C <= calc;
33                     opcode[1] <= Data_in;
34                     Data_out <= ^calc; // parity
35                     state <= status2;
36                 end
37                 status2: begin
38                     opcode[0] <= Data_in;
39                     Data_out <= ~|C; // all zero?
40                     state <= data1;
41                 end
42                 data1: begin
43                     Data_out <= C[2*n - 1 - i];
44                     A[n - 1 - i] <= Data_in;
45                     if (i == n - 1) begin
46                         state <= data2;
47                         i <= 0;
48                     end else i <= i + 1;
49                 end

```

```

50         data2: begin
51             Data_out <= C[n - 1 - i];
52             B[n - 1 - i] <= Data_in;
53             if (i == n - 1) begin
54                 state <= status1;
55                 i <= 0;
56             end else i <= i + 1;
57         end
58     endcase
59 end
60 end
61 endmodule

```

تست مدار

برای تست کردن مدار از یک testbench مشابه همان ورودی‌های صورت سوال استفاده شده است. کد تست در زیر آمده است. در پایان تست از دستور \$finish استفاده شده است تا شبیه‌سازی متوقف شود.

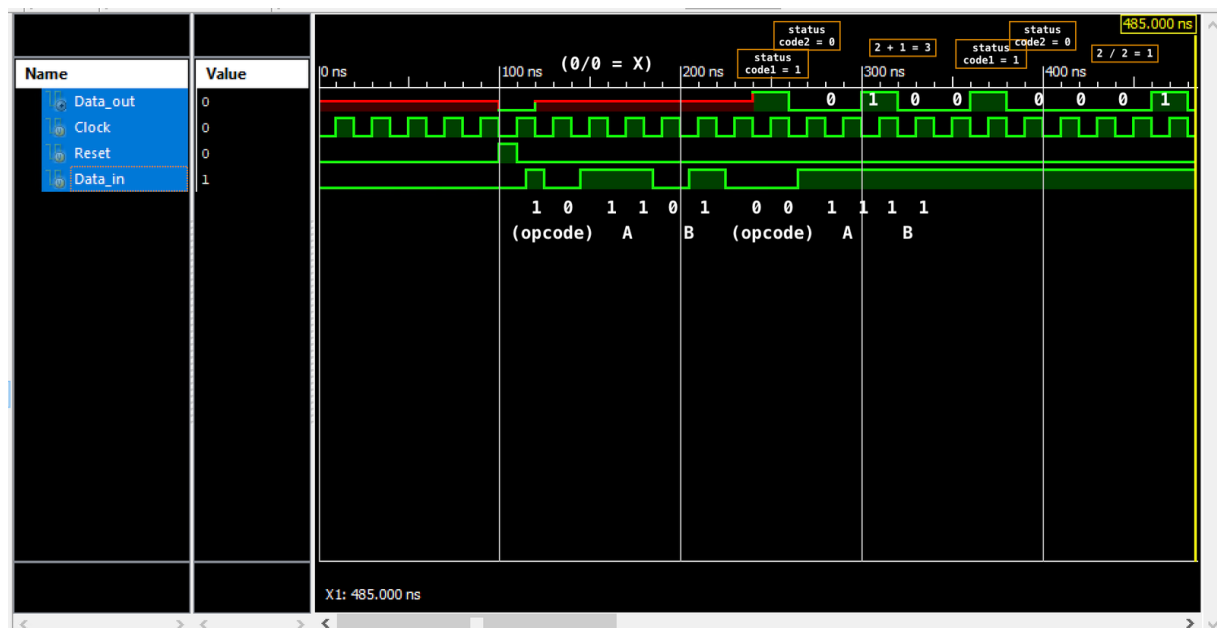
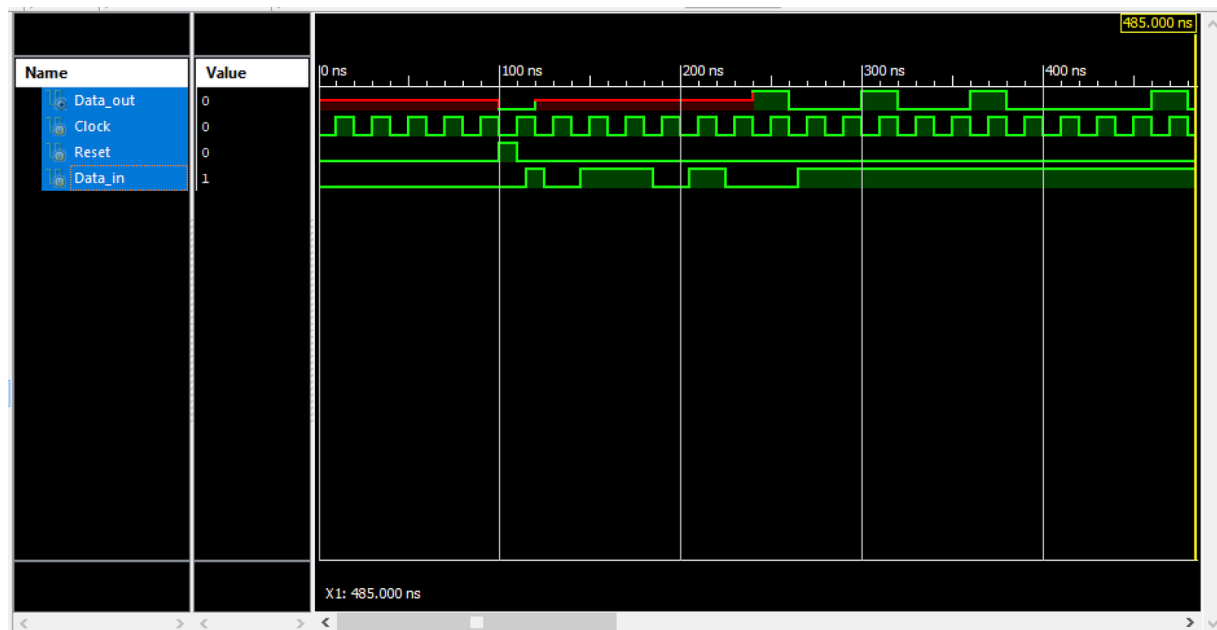
```

1 module alu_tb;
2     // Inputs
3     reg Clock;
4     reg Reset;
5     reg Data_in;
6     // Outputs
7     wire Data_out;
8     // Instantiate the Unit Under Test (UUT)
9     alu uut (
10         .Clock(Clock),
11         .Reset(Reset),
12         .Data_in(Data_in),
13         .Data_out(Data_out)
14     );

```

```
15
16     always #10 Clock = ~Clock;
17     initial begin
18         // Initialize Inputs
19         Clock = 0;
20         Reset = 0;
21         Data_in = 0;
22         // Wait 100 ns for global reset to finish
23         #100;
24
25         // Add stimulus here
26         Reset = 1;
27         #10 Reset = 0;
28         #5 Data_in = 1;
29         #10 Data_in = 0;
30         #20 Data_in = 1;
31         #40 Data_in = 0;
32         #20 Data_in = 1;
33         #20 Data_in = 0;
34         #40 Data_in = 1;
35         #220;
36         $finish;
37     end
38 endmodule
```

نتیجہی تست



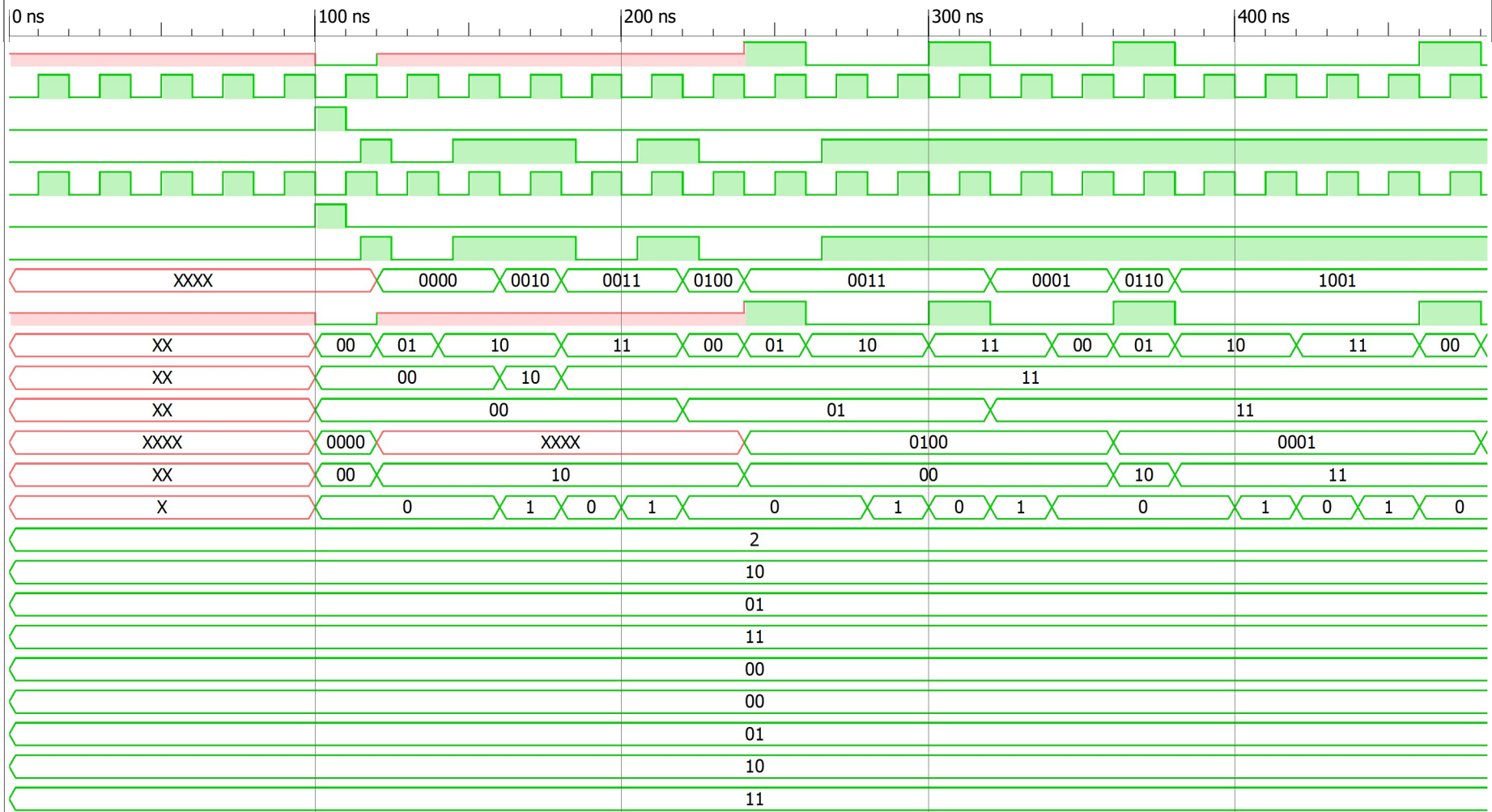
موج خروجی با توضیحات این که هر قسمت مربوط به چیست


اولین جایی که **Data_out** سبز می شود مربوط به بعد از ریست است. پس از reset مدار، opcode برابر با 00 (تقسیم) و هر دو متغیر برابر با 0 هستند. بنابراین مقدار خروجی به علت تقسیم بر 0 برابر با X است.

موج خروجی در انتهای این فایل و همچنین در فایل زیپ پیوست شده است.

در حین شبیه سازی می توان موج هایی از قطعات دیگر هم به قطعه ی اصلی master اضافه کرد. با اضافه کردن خود قطعه به موج ها می توانیم تغییر متغیرهای داخلی قطعه را بینیم که بسیار جالب است. این موج ها هم در انتهای این فایل و به صورت پیوست آمده.

- Data_out
- Clock
- Reset
- Data_in
- Clock
- Reset
- Data_in
- calc[3:0]
- Data_out
- state[1:0]
- A[1:0]
- B[1:0]
- C[3:0]
- opcode[1:0]
- i[31:0]
- n[31:0]
- sum_op[1:0]
- sub_op[1:0]
- mul_op[1:0]
- div_op[1:0]
- status1[1:0]
- status2[1:0]
- data1[1:0]
- data2[1:0]



-  Data_out
-  Clock
-  Reset
-  Data_in

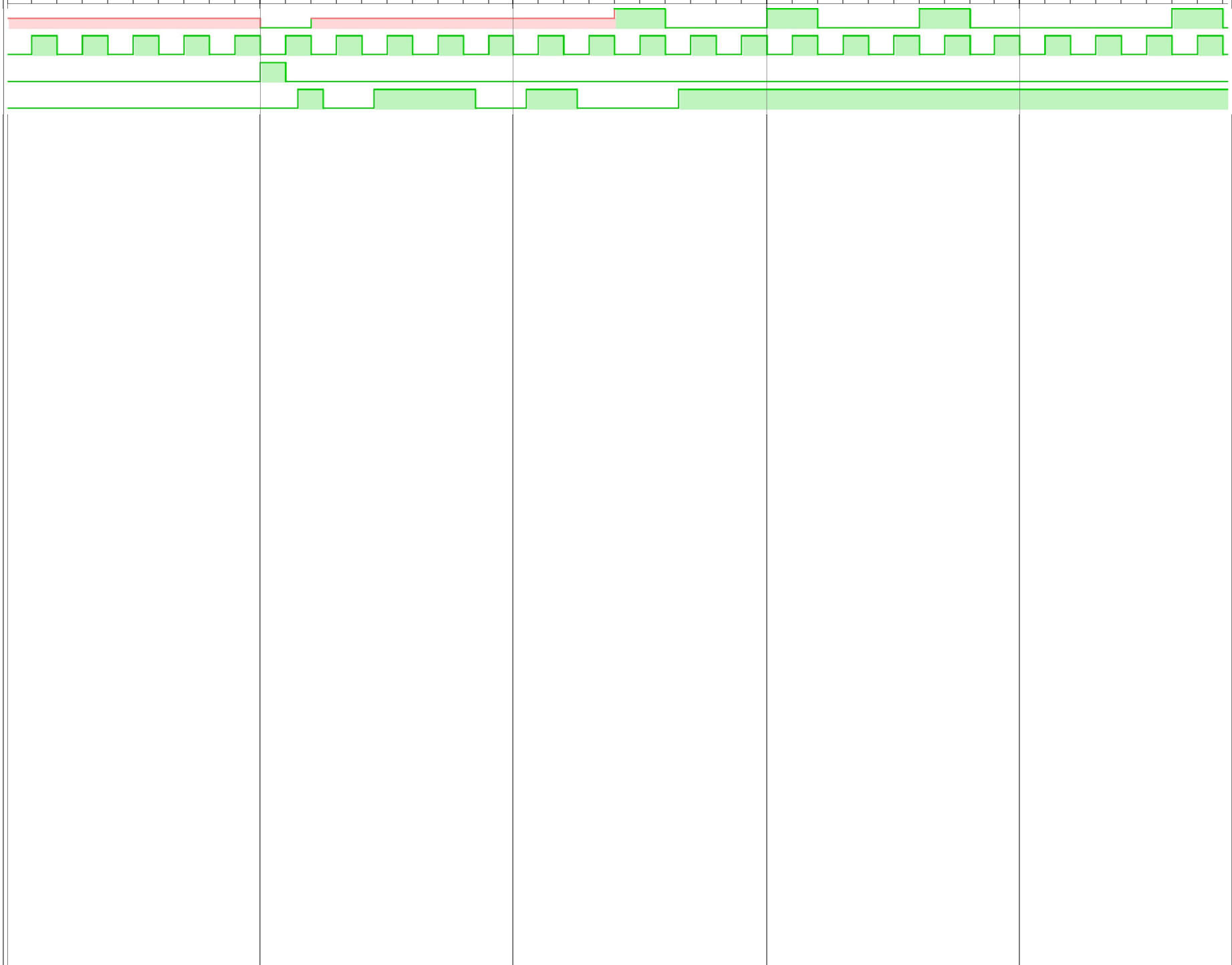
0 ns

100 ns

200 ns

300 ns

400 ns



```

wire [2*n-1:0] calc =
  opcode == sum_op ? A + B :
  opcode == sub_op ? A - B :
  opcode == mul_op ? A * B : A / B; // implicit assignment

```

