برای این بخش مانند بخش اول که تحلیل asm chart بود، باید فایل json را با همان قواعد گفته شده بسازید و سپس در کنار فایل این script گذاشته و اسم آن را در script مشخص کنید و فایل را اجرا کنید، خروجی script یک فایل با نام output.v هست که تبدیل شده ی asm chart به کد وریلاگ را درون خودش دارد و از ایده اصلی روش هافمن استفاده شده است در تولید کد وریلاگ. به عنوان نمونه در سوال اول، من فایل json مربوط به ضرب اعداد با روش جمع متوالی و همچنین مشخص کردن عدد اول و غیر اول را گذاشتم که خروجی هر کدام به ترتیب به شکل زیر است:

```
module MainModule(
  input wire [31:0] R1,
  output reg [31:0] R2,
  output reg [31:0] result,
  input wire clk,
  input wire reset
);
  reg [3:0] current_state, next_state;
  always @(*) begin
      case (current_state)
      0: begin
          if (R2 > 0) begin
              R1 = 10;
              R2 = 4;
              result = 0;
              next_state = 2;
          end else begin
              next_state = 0;
          end
      end
      1: begin
          result = result + R1;
          R2 = R2 - 1;
          next_state = 1;
      end
      2: begin
          next_state = 2;
      end
      endcase
  end
  always @(posedge clk) begin
      if (reset) begin
```

```
        current_state <= 0;
        R2 <= 0;
        result <= 0;
    end else begin
        current_state <= next_state;
    end
  end
endmodule
```

برای مشخص کردن اعداد اول به شکل زیر است:

```
module MainModule(
  input wire [31:0] R1,
  output reg [31:0] result,
  output reg [31:0] counter,
  input wire clk,
  input wire reset
);
  reg [2:0] current_state, next_state;
  output reg [31:0] rem;
  always @(*) begin
    case (current_state)
    0: begin
        if (counter < R1) begin
            R1 = 15;
            result = 1;
            counter = 2;
            rem = R1 % counter;
            next_state = 1;
        end else begin
            next_state = 0;
        end
    end
    1: begin
        if (rem == 0) begin
            result = 0;
            counter = counter + 1;
```

```
            next_state = 1;
        end else begin
            next_state = 1;
        end
    end
    endcase
  end
  always @(posedge clk) begin
    if (reset) begin
        current_state <= 0;
        result <= 0;
        counter <= 0;
        rem <= 0;
    end else begin
        current_state <= next_state;
    end
  end
endmodule
```