Group 24

Gorantla Thoyajakshi - 21CS10026

Ashwin Prasanth - 21CS300009

Computer Organisation and Architecture Lab

Verilog Assignment 5

Previous Approach

- We used an approach like this:
  - Consider output `Y` to be made of 4 bytes `Y3`, `Y2`, `Y1` and `Y0` , and similarly input `X` to be made of 4 bytes `X3`, `X2`, `X1`, `X0`, with `X3` and `Y3` being the most significant bytes
  - Then we can say:

$$\begin{aligned} Y &= Y3\ \ Y2\ \ Y1\ \ Y0 \\ 256Y &= Y2\ Y1\ \ Y0\ \ Z \\ X &= X3\ \ X2\ \ X1\ \ X0 \end{aligned} \tag{1}$$

  - Where `Z` is 8 bits of all zeroes
  - Then, on subtracting `X` from `256Y`, we can obtain `Y3`, `Y2`, `Y1` and `Y0`:

$$\begin{aligned} Y0 &= -X0 \\ Y1 &= Y0 - X1 \\ Y2 &= Y1 - X2 \\ Y3 &= Y2 - X3 \end{aligned} \tag{2}$$

  - Borrow outs will have to be considered in each byte-wise subtraction too
  - Finally, they can be concatenated to get `Y`
- We defined an external `subtractor8bit` module which takes `A` , `B` and `Bin` (borrow in) as input and gives `Bout` (borrow out) and `out` (`A`-`B`) as output
- 4 such modules were instantiated, for computation of each `Yi`
- The 32 bit number `X` is taken 16 at a time, through input `x`
- Similarly, the 32 bit output `X` is displayed 16 at a time, through output `y`
- We made 4 flag variables `flg1, flg2, flg3, flg4`: 2 for controlling input, 2 for controlling output
- In the always block, we defined 10 states for each computation:
  - `S0`: initial state, from here, when `flg1` is activated, it moves to `S1`
  - `S1`: in this state, the input `x` gets read into the 16 MSBs of `X`. After that, on activating `flg2` , it moves to `S2`
  - `S2`: in this state, the new input `x` gets read into the 16 LSBs of `X`. After that, it moves to `S3`, where the computation of `Y` begins
  - `S3`: here, all `Xi`s are found from `X`, and `Yi`s are initialised to zero, and it moves to `S4`

- `S4`: here `Y0` was obtained from the corresponding subtractor's output, then it moves to `S5`
- Similarly, in `S5`, `S6` and `S7`: `Y1`, `Y2` and `Y3` are obtained sequentially, and finally it moves to `S8`
- In `S8`, it prints the 16 MSBs of output, then moves to `S9` when `flg3` is activated
- In `S9`, it prints the 16 LSBs of output, then moves back to `S0` if `flg1` is actvated

- **This method, however, does not work for numbers that are not perfect multiples of 255, so we moved to our final approach, which however includes 32 bit addder**

Final Approach

- This method uses the following logic:

$$\begin{aligned} y &= 256y - x \\ (y >> 8) &= y - (x >> 8) \\ (y >> 16) &= (y >> 8) - (x >> 16) \\ (y >> 24) &= (y >> 16) - (x >> 24) \end{aligned} \tag{3}$$

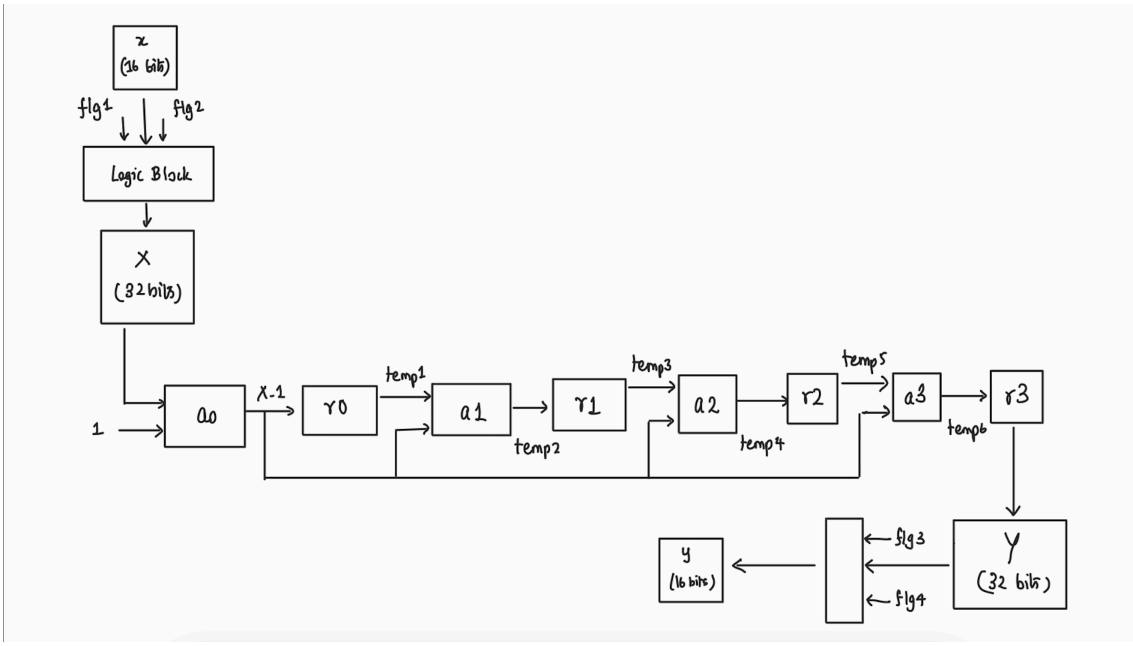
  - At each step, we are right shifting by 8
  - In the final step, we get $y >> 24$, and we know y could be a maximum of 24 bits long, so $y >> 24$ is equal to zero
  - Adding all steps and rearranging, we get:

$$y = (x >> 8) + (x >> 16) + (x >> 24) \tag{4}$$

  - This sum was found by right-shifting x by 8 one-by-one, and adding one-by-one, until the final result is obtained, the pseudocode is as follows:

```
X_1 = X + 1
temp1 = X_1 >> 8
temp2 = temp1 + X_1
temp3 = temp2 >> 8
temp4 = X_1 + temp3
temp5 = temp4 >> 8
temp6 = X_1 + temp5
Y = temp6 >> 8
```

- We defined two external modules: `rshift8bit` , to right shift by 8 bits, and `adder`, for adding 32 bit numbers
- Those modules were instantiated as required for calculating `Y`
- In the always block, we defined 5 states for each computation:
  - `S0`: initial state, from here, when `flg1` is activated, it moves to `S1`
  - `S1`: in this state, the input `x` gets read into the 16 MSBs of `X`. After that, on activating `flg2` , it moves to `S2`
  - `S2`: in this state, the new input `x` gets read into the 16 LSBs of `X`. After that, it moves to `S3`, where `Y` is first outputted
  - `S3`: here, it prints the 16 MSBs of output, then moves to `S4` when `flg3` is activated
  - `S4`: here, it prints the 16 LSBs of output, then moves back to `S0` if `flg4` is actvated (reset to initial state)
- Here is a simple schematic of our program:


x
(16 bits)
flg1
flg2
Logic Block
X
(32 bits)
1
a0
X-1
r0
temp1
a1
r1
temp2
temp3
a2
r2
temp4
temp5
a3
r3
temp6
Y
(32 bits)
flg3
flg4
y
(16 bits)