Assign_3_Grp27

Assignment 3 (Verilog)
Group No 27
Question 1
Semester 5
Yash Sirvi (21CS10083)
Sanskar Mittal (21CS10057)

Question 1

Add module

- **Input:** Two 8 bit numbers `n1` and `n2`
- **Output:** One 8 bit number sùm
- **Extra wires:** 8 bit carry `c`, propagate `p` and generate `g`

Behaviour:

- Carry lookahead adder works by calculating propagate `p`i and generate `gi` for every bit `i` ahead of getting the carry_in `ci`
- `p = n1 xor n2`
- `g = n1 and n2`
- `ci = gi or (pi and ci)`
- We calculate `p` and `g` in one go
- We then iteratively calculate all the `ci`s
- The final sum is given by `sum = p xor c`

Subtract module

- **Input:** Two 8 bit numbers `n1` and `n2`
- **Output:** One 8 bit number `diff`
- **Extra wires:**
  - 8 bit complement of `n2` : `n2_comp`
  - 8 bit value one `00000001`
  - 8 bit intermediate `sum`

Behaviour

- We calculate the difference of two numbers using the above defined adder by passing the complement of the 2nd number as the 2nd input
- We calculate 2's complement of `n2` using following steps
  - `n2_comp = ~n2`

    - `n2_comp+=1`
- `sum` stores the value of `~n + 1`
- final difference is stored in `diff` which is `add(n1, n2_comp)`

Identity module

- **Input:** One 8 bit number `inp`
- **Output:** One 8 bit number `out`

Behaviour:

- We pass the input directly to output
- `f(x) = x`

Left Shift module

- **Input:** One 8 bit number `inp`
- **Output:** One 8 bit number òut

Behaviour:

- Shift the bits of `inp` by 1 towards the left side
- This effectively multiplies the input by 2
- eg `left_shift(2) = 4`

Right Shift module

- **Input:** One 8 bit number `inp`
- **Output:** One 8 bit number òut

Behaviour:

- Shift the bits of `inp` by 1 towards the right side, Logical Right Shift
- This effectively divides the input by 2 and takes the floor of it
- `out = floor(x/2)`
- eg `right_shift(128) = 64`

AND module

- **Input:** Two 8 bit numbers `n1` and `n2`
- **Output:** One 8 bit number `out`
  Behaviour:
- Calculates bitwise `AND` of two numbers
- `out = n1 AND n2`
- eg: `and_gate(23, 62) = 22`

NOT module

- **Input:** One 8 bit number `inp`
- **Output:** One 8 bit number `out`

Behaviour:

- Outputs a number with all the bits flipped
- `out = ~inp`
- The output for a 8 bit number will be `2^8 - n - 1`
- eg: `not_gate(50) = 205 = 256 - 50 - 1`

OR module

- **Input:** Two 8 bit numbers `n1` and `n2`
- **Output:** One 8 bit number `out`

Behaviour:

- Calculates bitwise `OR` of two numbers
- `out = n1 OR n2`
- eg: `and_gate(23, 4) = 4`

Schematic