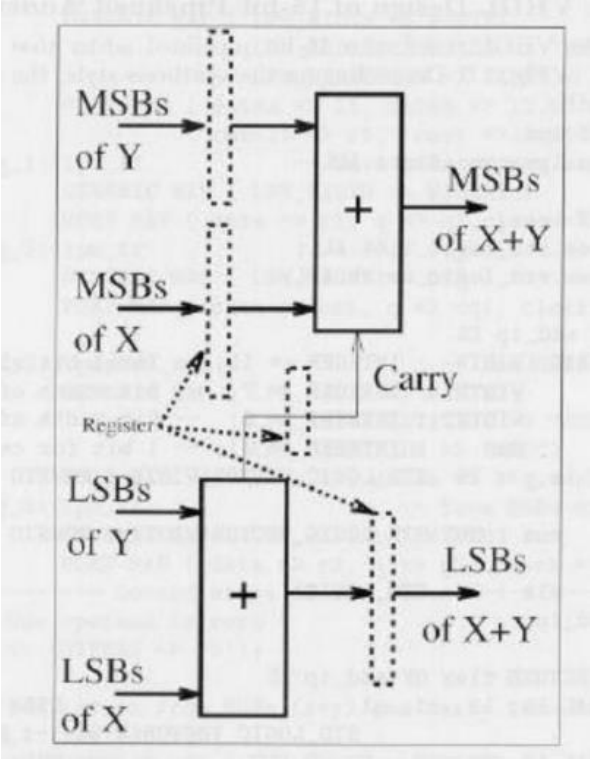
Abstract

Digital computer arithmetic is an aspect of logic design with the objective of developing appropriate algorithms in order to achieve an efficient utilization of the available hardware. Computer basic arithmetic operation is enhanced by pipelining the process. The paper focuses on the implementation of a basic but configurable ALU with capability of adding and subtracting with pipeline. The ALU also takes care of the carry bit generated while adding of LSBs in the current cycle and utilize it in the addition of MSBs in the next cycle. The ALU is currently programmed to add two 16-bit values and provide the 16-bit result. The testbench is generating random numbers for the inputs and compares the output with its own reference calculation.

Introduction



Features:

- Configurable width.
- Provides output in the next cycle.
- The width of each adder is configurable.
- Can run at high frequency, when compared to regular adder.
- Carry forward from LSB adder to MSB adder.

The diagram shows a basic ALU to add two n-bit numbers x and y and provide n-bit output in the next cycle. The significance of running and dividing the adder into 2 segments is a regular adder slows down when number of bits increase.

Body:

lpm_add_sub:

Ports	**Direction**
dataa	Input
datab	Input
cin	Input
cout	Output
result	Output

This module is configurable to take inputs and output of the data as WIDTH. The default value is 1. The parameter ALU decides the operation to be done. When ALU = ADD, result is the addition of dataa, datab, and cin. The carry generated by the addition is given on cout. On ALU = SUB, result is the subtraction of datab from dataa.

lpm_ff:

Ports	**Direction**
clk	Input
data	Input
q	Output

This module is a basic D-flipflop and registers the input. The register works on the positive edge of the clock. The width of data path is configurable.

add_1p:

Ports	Direction
clk	Input
x	Input
y	Input
sum	Output

This is the top module of the design. It takes x and y in the current cycle and returns the addition in the next cycle.

In the first cycle, the LSBs of both x and y adds to generate summation r1 and carry out cr1. The number of LSB bits is configurable. r1 and cr1 then gets registers to next cycle.

Meanwhile, the MSBs of x and y gets registers the moment they enter the module. The output of the register, l3 and l4 are then feeded to the adder along with the carry out bit of LSB adder, cr1.

The output sum is the concatenation of the output of MSB adder and r1.

adder_tb:

This module is the testbench for the design add_1p.

The clock of frequency feeding to the adder is 100MHz. The testbench generates random data values of x and y and compares the output of design with its reference calculation. It displays Adder PASSED if the test passes and displays Adder FAILED if the test fails.

Summary

```
Adder PASSED: A = 0000 :: B = 0000 :: SUM = 0000
Adder PASSED: A = 3524 :: B = 5e81 :: SUM = 93a5
Adder PASSED: A = d609 :: B = 5663 :: SUM = 2c6c
Adder PASSED: A = 7b0d :: B = 998d :: SUM = 149a
Adder PASSED: A = 8465 :: B = 5212 :: SUM = d677
Adder PASSED: A = e301 :: B = cd0d :: SUM = b00e
Adder PASSED: A = f176 :: B = cd3d :: SUM = beb3
Adder PASSED: A = 57ed :: B = f78c :: SUM = 4f79
Adder PASSED: A = e9f9 :: B = 24c6 :: SUM = 0ebf
Adder PASSED: A = 84c5 :: B = d2aa :: SUM = 576f
Adder PASSED: A = f7e5 :: B = 7277 :: SUM = 6a5c
Adder PASSED: A = d612 :: B = db8f :: SUM = b1a1
Adder PASSED: A = 69f2 :: B = 96ce :: SUM = 00c0
Adder PASSED: A = 7ae8 :: B = 4ec5 :: SUM = c9ad
Adder PASSED: A = 495c :: B = 28bd :: SUM = 7219
Adder PASSED: A = 582d :: B = 2665 :: SUM = 7e92
Adder PASSED: A = 6263 :: B = 870a :: SUM = e96d
Adder PASSED: A = 2280 :: B = 2120 :: SUM = 43a0
Adder PASSED: A = 45aa :: B = cc9d :: SUM = 1247
Adder PASSED: A = 3e96 :: B = b813 :: SUM = f6a9
Adder PASSED: A = 380d :: B = d653 :: SUM = 0e60
$finish called from file "adder_tb.v", line 31.
$finish at simulation time                216
          V C S   S i m u l a t i o n   R e p o r t
```

Code

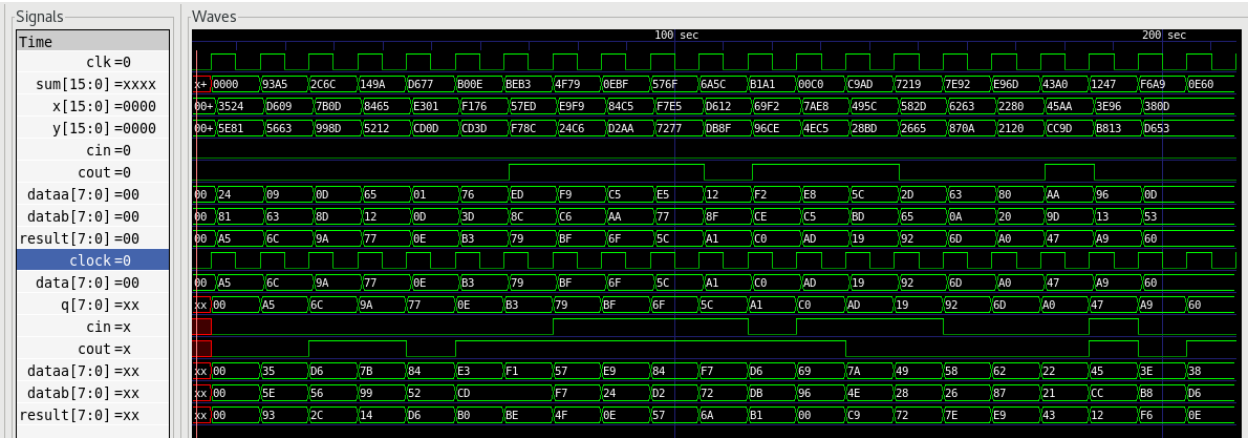
Adder design – adder.v

```
module add_1p (x, y, sum, clk);

parameter WIDTH  = 16,  // Total bit width
          LSB_WIDTH = 8,   // Bit width of LSBs
          MSB_WIDTH = 8;   // Bit width of MSBs

input [WIDTH-1:0] x,y; // Inputs
output [WIDTH-1:0] sum; // Result
input clk; // Clock

reg [LSB_WIDTH-1:0] l1, l2; // LSBs of inputs
wire [LSB_WIDTH-1:0] q1, r1; /// LSBs of inputs
reg [MSB_WIDTH-1:0] l3, l4; // MSBs of input
wire [MSB_WIDTH-1:0] r2, q2, u2; // MSBs of input
wire cr1,cq1; // LSBs carry signal

wire [MSB_WIDTH-1:0] h2; // Auxiliary MSBs of input

// Split in MSBs and LSBS and store in registers
always @(posedge clk) begin
// Split LSBs from input x,y
l3[MSB_WIDTH-1:0] <= x[MSB_WIDTH-1+LSB_WIDTH:LSB_WIDTH];
l4[MSB_WIDTH-1:0] <= y[MSB_WIDTH-1+LSB_WIDTH:LSB_WIDTH];
end

/************* First stage of the adder ***********/
lpm_add_sub #(LSB_WIDTH, "ADD") add_1 // Add LSBs of x and y
(.result(r1), .dataa(x[LSB_WIDTH-1:0]), .datab(y[LSB_WIDTH-1:0]), .cin(1'b0), .cout(cr1)); // Used ports

lpm_ff #(LSB_WIDTH) reg_1 // Save LSBs of x+y
(.data(r1), .q(q1), .clock(clk));

lpm_ff #(1) reg_2 // Save LSBs carry
(.data(cr1), .q(cq1), .clock(clk));

lpm_add_sub #(MSB_WIDTH, "ADD") add_2 //Add MSBs of x and y
(.dataa(l3), .datab(l4),.result(r2), .cin(cq1), .cout()); //Used ports

assign sum = {r2,q1};

endmodule

module lpm_ff #(parameter WIDTH = 1)
(input [WIDTH-1:0] data, output reg [WIDTH-1:0] q, input clock);
always @(posedge clock) begin
   q <= data;
end
endmodule

module lpm_add_sub #(parameter WIDTH = 1, parameter ALU = "ADD")
(input [WIDTH-1:0] dataa, input [WIDTH-1:0] datab, input cin, output cout, output [WIDTH-1:0] result);

reg [WIDTH:0] temp;

always @* begin
case(ALU)
   "ADD" : temp = dataa + datab + cin;
   "SUB" : temp = dataa - datab;
   default: temp = {WIDTH{1'b1}};
endcase
end
assign cout = temp[WIDTH];
assign result = temp[WIDTH-1:0];

//assign {cout,result} = dataa + datab + cin;
endmodule

```

Testbench – adder_tb.v

```
module adder_tb();

parameter WIDTH = 16,
          LSB_WIDTH = 8,
          MSB_WIDTH = 8;

reg  clk;
reg  [WIDTH-1:0] x,y;
wire [WIDTH-1:0] sum;

reg  [WIDTH-1:0] x_r, y_r;

add_1p #(WIDTH, LSB_WIDTH, MSB_WIDTH) u_adder_1p
(x, y, sum, clk);

initial begin
   clk=0;
   forever #5 clk=~clk;
end

initial begin
   x=0;
   y=0;
   repeat(20) begin
      @(posedge clk);
      #1;
      x=$random;
      y=$random;
   end
   #20;
   $finish;
end

always @(posedge clk) begin
   x_r <= x;
   y_r <= y;
end

always @(posedge clk) begin
   if(sum === x_r + y_r)
      $display("Adder PASSED: A = %h :: B = %h :: SUM = %h",x_r, y_r, sum);
   else
      $display("Adder FAILED: A = %h :: B = %h :: SUM = %h",x_r, y_r, sum);
end

initial begin
   $dumpfile("adder.vcd");
   $dumpvars(0);
end

endmodule
```

Waveform



References

https://books.google.com/books?id=Uj7-m45kRtUC&pg=PA490&lpg=PA490&dq=lpm_add_sub+lpm_ff&source=bl&ots=DNzI7z_ii0&sig=ACfU3U0FZNd_he1WIOQcmCKDUJqml3b5gA&hl=en&sa=X&ved=2ahUKEwjMiszZ24ToAhWbKDQIHSccCCsQ6AEwBHoECAoQAQ#v=onepage&q=lpm_add_sub%20lpm_ff&f=false