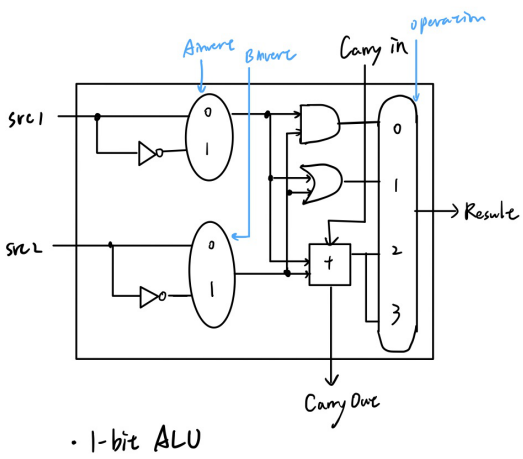
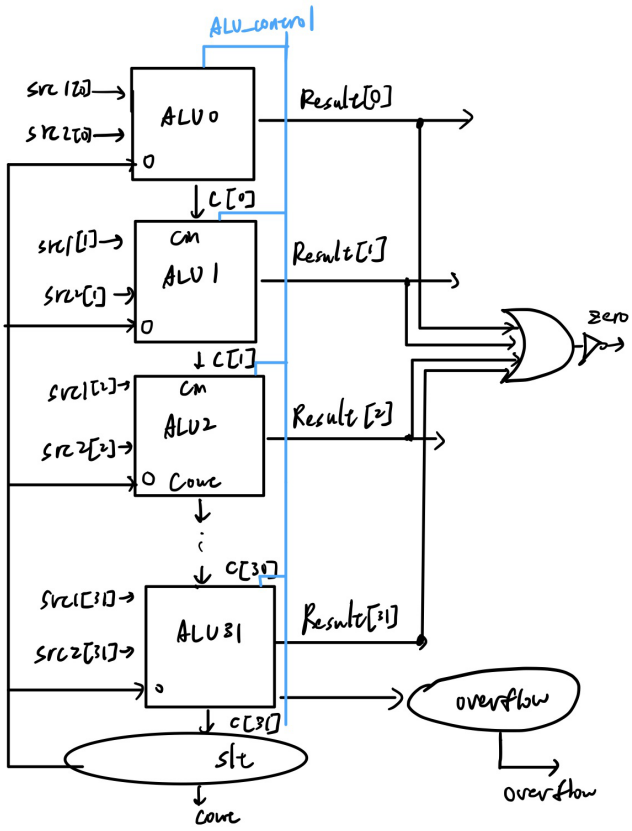
計組Lab2 report

Architecture diagram





Detailed description of the implementation

- 1-bit ALU

```
always @(*) begin
  if(Ainvert==1'b1) begin
    temp1=(~ src1);
  end
  else begin
    temp1=src1;
  end
  if(Binvert==1'b1) begin
    temp2=(~ src2);
```

```
  end
  else begin
    temp2=src2;
  end
  case(operation)
    2'b00:begin
      result=temp1 & temp2;   //and nand
      cout=Cin;
    end

    2'b01:begin
      result=temp1 | temp2;   //or nor
      cout=Cin;
    end

    2'b10,2'b11:begin
      result=temp1 + temp2 + Cin;   //add sub
      if(Cin==1)begin
          if(temp1 | temp2)
          cout=1;
          else
          cout=0;
      end
      else if(Cin==0) begin
          if(temp1 & temp2)
          cout=1;
          else
          cout=0;
      end
    end
  endcase
end
```

1. 首先我判斷Ainvert，Binvert需不需要取補數。
2. 再來判斷operation分三個part，因為已經取補數所以and nand，or nor，add sub可以一起做。
3. slt跟add sub一起做，等32bit的時候再另外拉出來做。
4. add sub時另外處理carry in和carry out，否則cout=Cin。

- 32-bit ALU

```
always @(negedge rst_n) begin
  if(~rst_n)begin
    result<=0;
    zero<=0;
    cout<=0;
    overflow<=0;

  end
end
```

```
wire  [32-1:0]c;
wire  [32-1:0]temp;
ALU_1bit alu_32(
    .src1(src1[0]),
    .src2(src2[0]),
    .Ainvert(ALU_control[3]),
    .Binvert(ALU_control[2]),
    .Cin(ALU_control[3]^ALU_control[2]),
    .operation(ALU_control[1:0]),
    .result(temp[0]),
    .cout(c[0])
  );
genvar id;
generate
  for(id=1;id<32;id=id+1)begin
    ALU_1bit alu_32(
      .src1(src1[id]),
      .src2(src2[id]),
      .Ainvert(ALU_control[3]),
      .Binvert(ALU_control[2]),
      .Cin(c[id-1]),
      .operation(ALU_control[1:0]),
      .result(temp[id]),
      .cout(c[id])
    );
  end
endgenerate

reg signed  [32-1:0]a,b,r;
always @(*) begin
  a=src1;
  b=src2;
  r=temp;
  case(ALU_control[1:0])
    2'b11:begin
      result[31:1]=0;
      result[0]=temp[31];
      cout=0;
      zero=~(|result);
    end
    default:begin
    result=temp;
    cout=c[31];
    zero=~(|result);
    end
endcase
end
always @(*) begin
  if(ALU_control[2:0]==3'b010)begin
    //add
    if(a>0 && b>0 && r<0)
      overflow=1;
    else if(a<0 && b<0 && r>0)
      overflow=1;
    else
      overflow=0;
  end
```

```
  else if(ALU_control[2:0]==3'b110)begin
    //sub
    if(a>0 && b<0 && r<0)
      overflow=1;
    else if(a<0 && b>0 && r>0)
      overflow=1;
    else
      overflow=0;
  end
  else begin
    overflow=0;
  end
end
```

1. 當rst_n=0時，初始化把output設為0，ALU[0]先在always外面做一次。
2. 接著用generate for把剩下的31個ALU串起來。
3. 另外開一個always(*)計算最後的carry out和zero，順便處理slt的指令。
4. 最後再開always處理overflow的計算。

Implementation results

- 1-bit ALU測資

```
            a = 1;
            b = 1;
            Ainvert = 0;
            Binvert = 0;
            Cin = 1;
            operation = 2'b00;
            #1
      $display("sum %d", sum);
      $display("carry %d", carry);
      $display("===============");
            #period; // wait for period

            a = 0;
            b = 0;
            Ainvert = 0;
            Binvert = 0;
            Cin = 0;
            operation = 2'b01;
            #1
      $display("sum %d", sum);
      $display("carry %d", carry);
      $display("===============");
            #period;

            a = 0;
            b = 1;
```

```
            Ainvert = 1;
            Binvert = 0;
            Cin = 1;
            operation = 2'b11;
            #1
      $display("sum %d", sum);
      $display("carry %d", carry);
      $display("===============");
            #period;
```

- 1-bit ALU result

```
C:\Users\Suyin\Desktop\ALU>vvp t1.vvp
VCD info: dumpfile alu.vcd opened for output.
sum 1
carry 1
===============
sum 0
carry 0
===============
sum 1
carry 1
===============
```

- 32-bit ALU 測資

```
//-助教提供的testbench。
```

- 32-bit ALU result

```
C:\Users\Suyin\Desktop\ALU>vvp t1.vvp
****************************************************
*             PATTERN RESULT TABLE                 *
****************************************************
* PATTERN *              Result             * ZCV *
****************************************************
*      Congratulation! All data are correct!       *
****************************************************
Correct Count: 30
testbench.v:95: $finish called at 415000 (1ps)
```

Problems encountered and solutions

- 一開始1-bit的時候不知道怎麼處理slt，卡了很久，後來搞清楚slt的作用後才改到32-bit處理。
- 處理carry out的時候用了不同的邏輯匣判斷不同情況，想了很久，但其實也可以用{ }把bit接起來。
- 有時候程式碼會忘記用begin end包起來導致錯誤，分好block真的很重要，即使只有一行最好也有begin end。
- 寫32-bit時一開始以為要寫32次ALU_1bit，後來同學建議可以用generate for，省了不少工夫。
- 處理ZCY時不太清楚應該要怎麼一次判斷32個bit，一開始用迴圈把reg的32個bit跑過一遍，但是有error，上網查才知道可以直接對一個register做邏輯運算。
- overflow的部分也是花了很久時間研究。
- 計算carry out時發現很奇怪怎麼改都不對，回去看1-bit才發現有地方寫錯。

Comment

上次寫verilog已經是一年前的事了，而且當時學數位電路時也沒有很精進verilog，現在必須花兩倍的時間回憶語法以及習慣硬體邏輯思維，期間也一直不斷上網查資源還有請教同學，深感自己對硬體還需要多努力。寫verilog時掌握always block的概念並活用對於寫作業很有幫助，很多東西需要分好幾個block來完成，但對語法的不熟悉也拉長了寫作業的時間。雖然過程很辛苦，但最後看到終於全部correct的時候真的很有成就感。