Discussion Session 4

CS/ECE 552

Ramkumar Ravi

20 Feb 2012

IMPORTANT

- HW2 is due on 02/22
  - Electronic submissions due by 12:30 PM (02/22)
  - Manual copies due in class (02/22)
- HW2 clearly specifies what needs to be submitted electronically and what needs to be handed manually -> PLEASE FOLLOW
  - Ensure your top level modules are named as per specifications (not just filenames)
  - Points might be deducted from this HW onwards
- Why so many RULES !!! ☹
  - This is how industry works as well -> You are required to follow conventions
  - Paves way for uniformity; more efficient and reliable grading
  - Better marks ☺
- What happens on a late submission ?
  - Professor Wood will take the final call on this

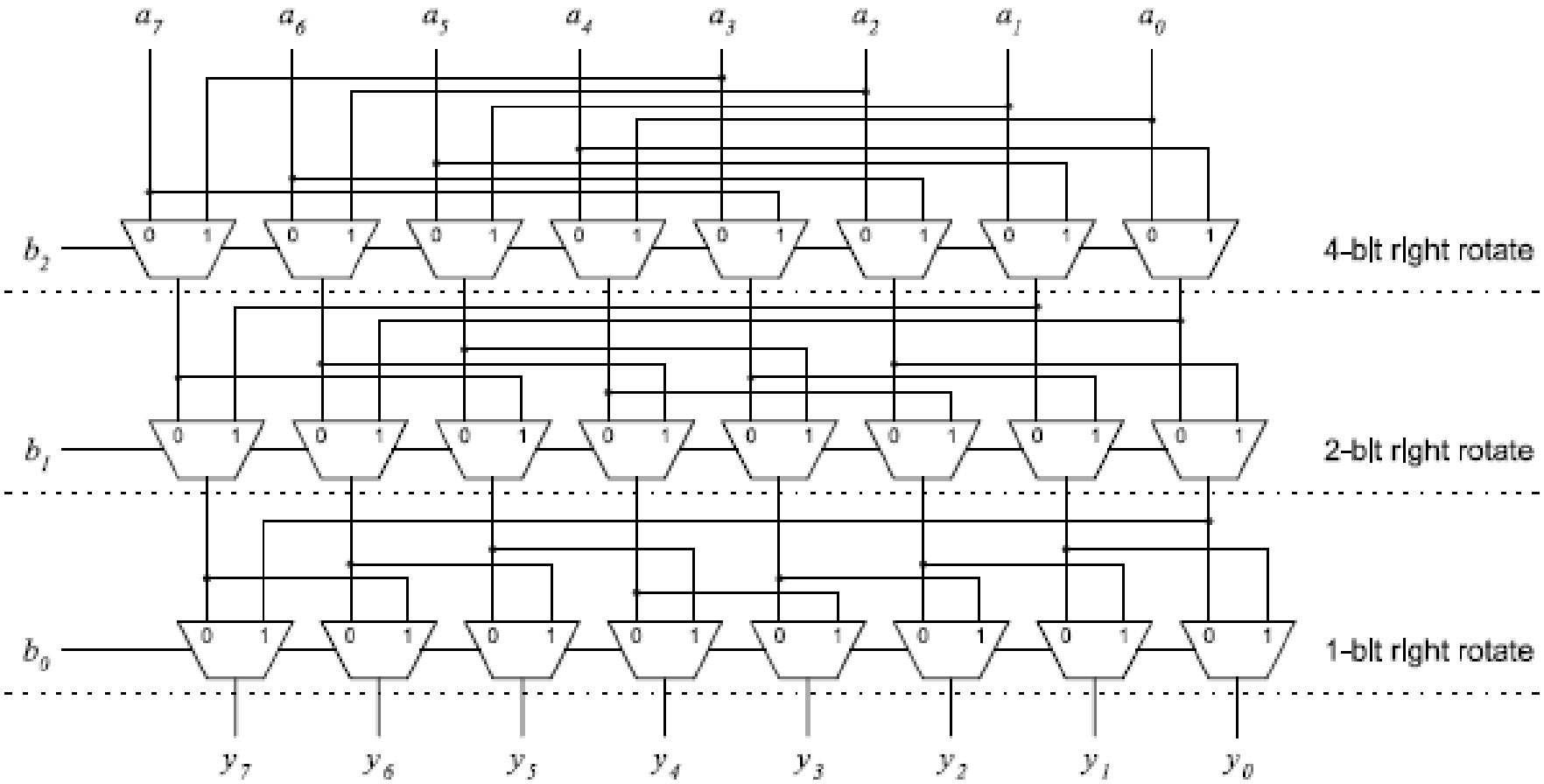
Problem 1 – Barrel shifter

- Barrel shifters are often utilized by embedded digital signal processors and general-purpose processors to manipulate data

- In this table, the bit vector for A is denoted as $a_7a_6a_5a_4a_3a_2a_1a_0$ and the shift/rotate amount is 3 bits. As illustrated in this table

Operation	Y
3-bit shift right logical	0 0 0 a_7 a_6 a_5 a_4 a_3
3-bit shift right arithmetic	a_7 a_7 a_7 a_7 a_6 a_5 a_4 a_3
3-bit rotate right	a_2 a_1 a_0 a_7 a_6 a_5 a_4 a_3
3-bit shift left logical	a_4 a_3 a_2 a_1 a_0 0 0 0
3-bit shift left arithmetic	$a_7$$a_3$ a_2 a_1 a_0 0 0 0
3-bit rotate left	a_4 a_3 a_2 a_1 a_0 a_7 a_6 a_5

8-bit Right Rotate example implementation



Points to note

- EXPLORE – Design can be simplified by using a combination of 2:1 MUX and the 4:1 MUX you designed in the last HW (<= 50 lines of code)

- Hint: What do you think about this code below ?

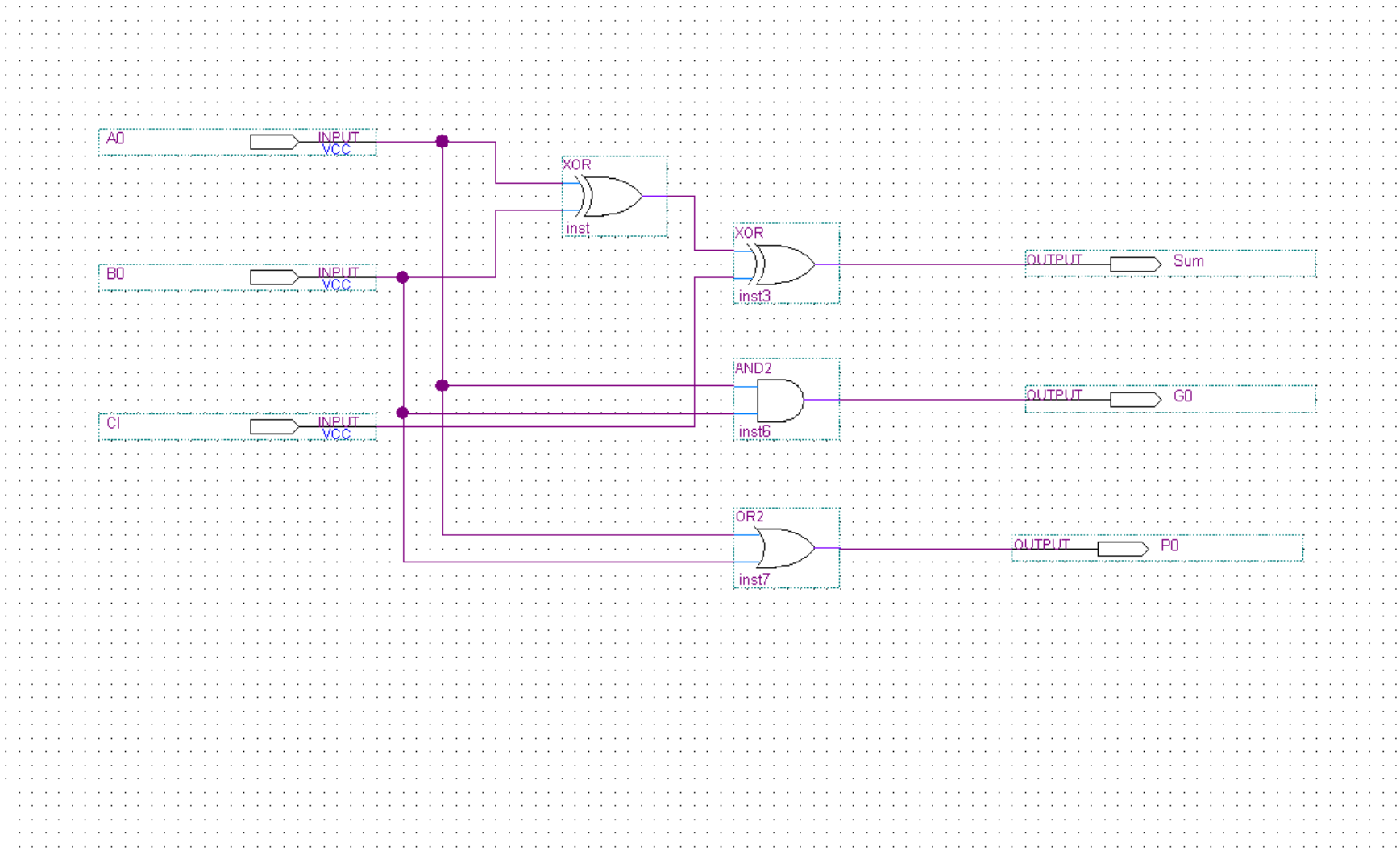
```
wire [15:0] S0;
wire [15:0] L0;
// level 1
mux2_1 inst [13:0] (.InA(In[13:0]), .InB(In[15:2]), .Out(lev0[14:1]), .S(Op[1]));
mux4_1 inst1 (.out(lev0[0]), .InA(In[15]), .InB(1'd0), .InC(In[1]), .InD(In[1]), .S(Op));
mux4_1 inst2 (.out(lev0[15]), .InA(In[14]), .InB(In[14]), .InC(In[0]), .InD(In[15]), .S(Op));
mux2_1 inst [15:0] (.InA(In), .InB(L0), .S(cnt[0]), .out(out));

/levels 2, 4 and 8
??
```

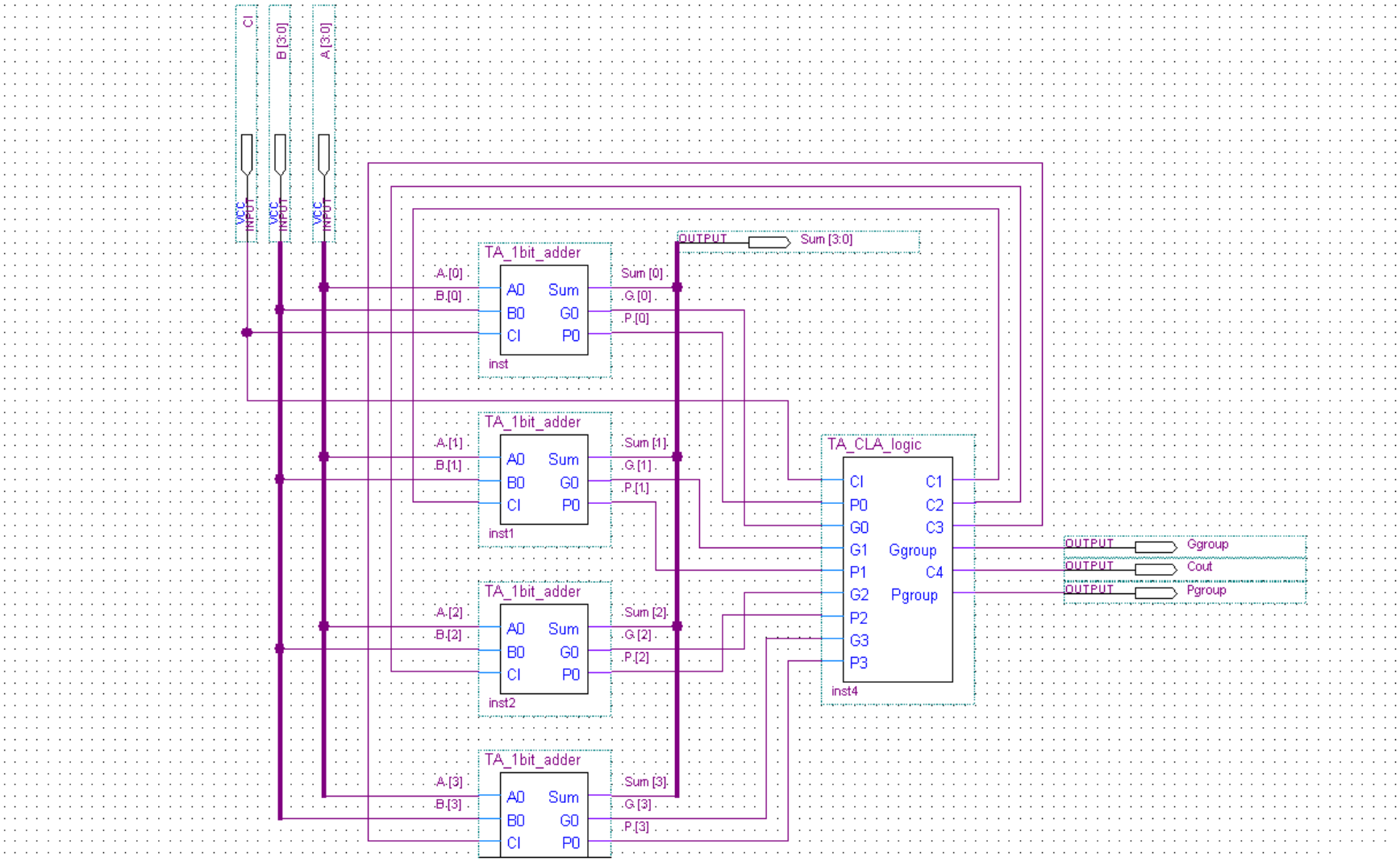
From previous slide – All 1 bit operations

- For rotate left (Op = 00)
  - lev0 [14:1] = In [13:0]
  - lev0 [0] = In [15]
  - lev0 [15] = In [14]
- For shift left (Op = 01)
  - lev0 [14:1] = In [13:0]
  - lev0 [0] = 0
  - lev0 [15] = In [14]
- For rotate right (Op = 10)
  - lev0 [14:1] = In [15:2]
  - lev0 [0] = In [1]
  - lev0 [15] = In [0]
- For shift right arithmetic (Op = 11)
  - lev0 [14:1] = In [15:2]
  - lev0 [0] = In [1]
  - lev0 [15] = In [15]

Start with a 1-bit Adder



Design CLA logic and build 4-bit CLA



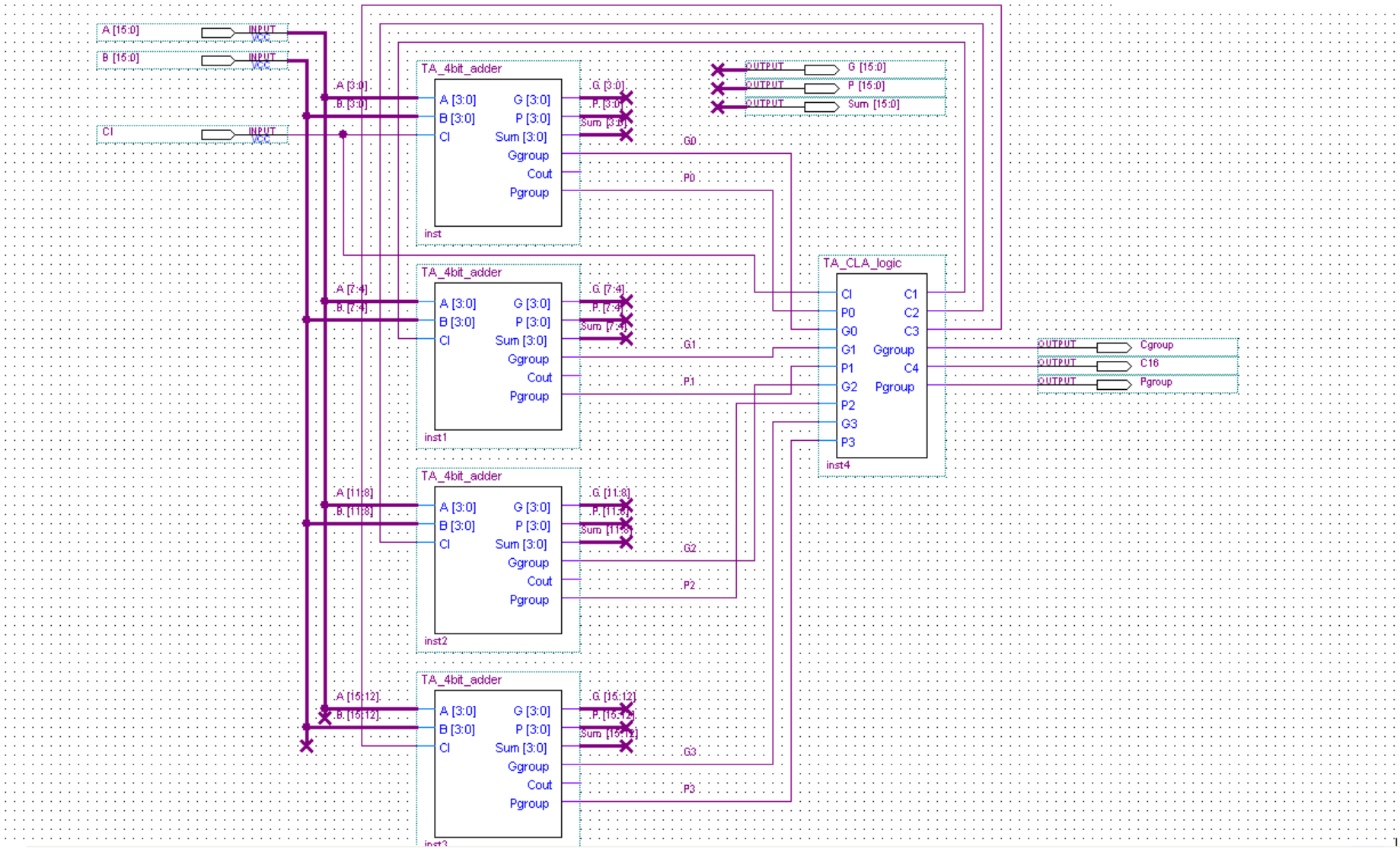
CLA Logic

- C1 = G0 + P0.C0
- C2 = G1 + P1.C1 = G1 + P1.G0 + P1.P0.C0
- C3 = G2 + P2.C2 = G2 + P2.G1 + P2.P1.G0 + P2.P1.P0.C0
- C4 = G3 + P3.C3 = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.G0 + P3.P2.P1.P0.C0

- Ggroup = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.G0
- Pgroup = P3.P2.P1.P0

- C4 = Ggroup + (Pgroup.C0)
- Cout = C4

Extend to 16-bit CLA



Other things to note

- Use shifter designed in problem 1
- Shift amount is represented by lower 4 bits of input B
  Input to be shifted is A

  a1 = invA ? (~InA) : InA;

  b1 = invB ? (~InB) : InB;
- Take care of OFL -> Keep track of sign bit and ‘cout’ bit

OVERFLOW

- Overflow will occur for both signed and unsigned arithmetic
  - For unsigned arithmetic, OFL can be detected just by using Cout

    Sign == 1'b0 and Cout == 1'b1
  - For signed arithmetic however, you will have to check for the following cases:
    - the sum of two positive numbers is negative;
    - the sum of two negative numbers is non-negative;
    - subtracting a positive number from a negative one yields a positive result; or
    - subtracting a negative number from a non-negative one yields a negative result.
- Example consider you are adding +17 and +19 in signed arithmetic and they are represented by 6 bits

  010001 + **0**10011 = **1**00100 -> However this is interpreted as -28 and not +36 as desired; overflow !!

  101111 + **1**01101 = **0**11100 -> However this is interpreted as +28 and not -36 as desired; overflow
- Check *sign* of the sum and compare it against the signs of the numbers added. Obviously, two positive numbers added together should give a positive result, and two negative numbers added together should give a negative result.

Problem 3 and 4

- Example problem

```
for (i=0; i<a; i++)
{
    a += b;
}
```

Assume a, b and i are in $s0, $s1 and $to respectively

```
    addi $t0, $0, 0              #$t0 = 0; i=0
    beq $0, $0, TEST             # branch to TEST
    LOOP: add $s0, $s0, $s1      # a=a+b
          addi $t0, $t0, 1       # i=i+1
    TEST: slti $t2, $t0, 10      # $t2 = 1 if $t0 < 10
          bne $t2, $0, LOOP      # if $t2 not equal to $0, go to LOOP
```

POINTS TO NOTE

- However in the problem , you will have to work with arrays
  - If a[i] is in location 0, a[i+1] will be in location 4 and so on

- Problem 3 is relatively simple
  - Load operands into registers
  - Perform operations
  - Store results into memory

Problem 5

- Total number of instructions is given
- Also mentioned is the % distribution of each instruction along with the number of cycles an instruction takes to execute
  - Overall CPI = (40*2 + .. + .. + ..) / total
  - Old number of multiplies = 8% of 200 = 16
  - 50% of the old number is replaced by shift-add that takes 3.5 cycles each
  - New number of multiplies = 16* 50% = 8
  - Additional ALU instructions = 16* 50% * 3.5 = X
  - Total number of instructions = 200 -16 +8 + X = Y
  - Calculate new total number of instructions and the new CPI

Problem 6

- Find the maximum IPC
  - IPS = IPC * clock speed
  - Ideal instruction sequence for P1 is once that is composed of instructions entirely from Class A -> because it takes the least amount of cycles to execute
  - Peak performance of P1 = 4GHz * IPC = XX MIPS
  - Similarly find for P2

- Average CPI

  = (2A+B+C+D+E)/(2+1+1+1+1)

  Find results for both P1 and P2

  Speedup (P2) / Speedup (P1) = Avg. time per instruction on P1 / Avg. time per instruction on P2

  For P1, average CPI / 4GHz and for P2, average CPI / 6GHz

- Find out that frequency where ratio above is 1

Problem 7

- CPI = (CPU time × clock rate)/No. instr.
  - (700 * 4 * 10^9) / (0.85 * old instr count)

- Clock rate ratio = New clock / old clock

  $(CPI)_{4\ GHz}$ / $(CPI)_{3\ GHz}$ for a and b

  Why do you think they are similar or dissimilar ?

- New execution time / old execution time

  CPU time reduction = [1 – (New exec time / old exec time)] * 100 = ?? %