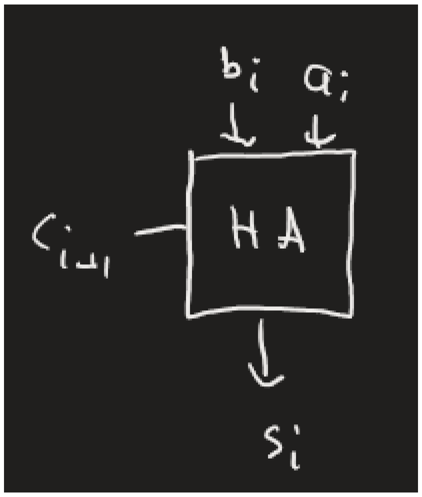
# CSE 331 HW3 Report

To design an multiplier which's work approach is given I need to design some circuits. At first I think about what do I require to design the given multiplier so I find that I need:

- 32 Bit Adder
- 64 Bit Shifter

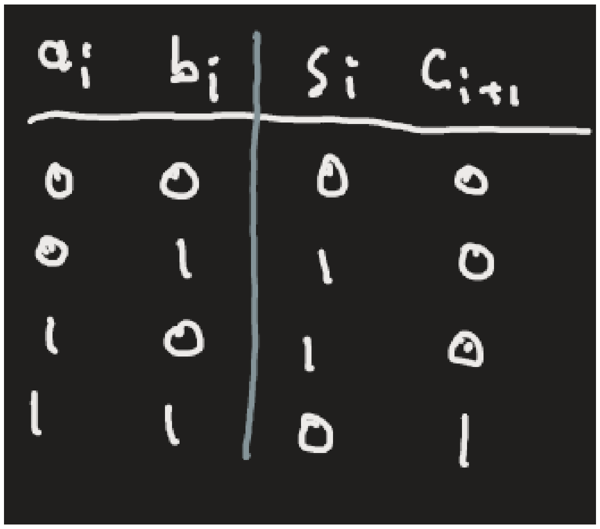
To design a combinational 32 bit adder I need 1 half adder and 31 full adder.

To design a half adder, I draw input and outputs of the half adder.

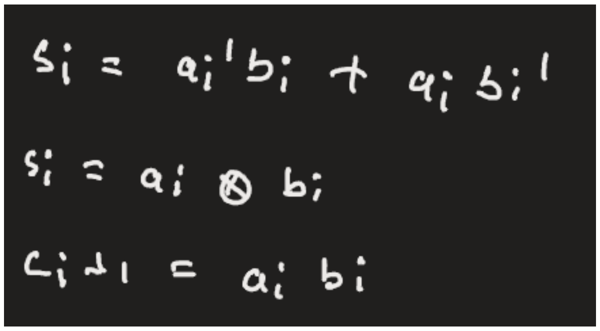


- *b_i* and *a_i* are input bits of the adder
- *c_i+1* is the carry out bit of addition
- *s_i* is the result bit of the addition

Then I write down the truth table of half adder which is written as writing desired outputs for given inputs. Then using truth table we need to derive boolean equations for the output signals.



| $a_i$ | $b_i$ | $s_i$ | $c_{i+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

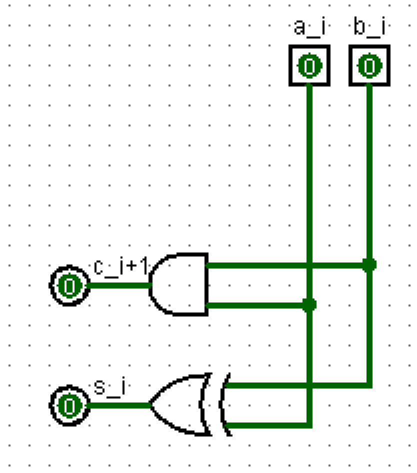


$$s_i = a_i' b_i + a_i b_i'$$

$$s_i = a_i \otimes b_i$$

$$c_{i+1} = a_i b_i$$

After all of this I designed my half adder in logisim which is

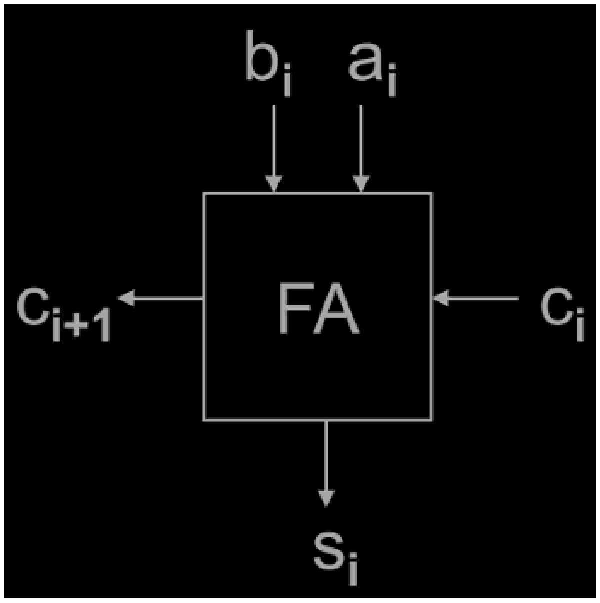


To design full adder I used the design which is given in the book. The input and output is demonstrated as

*b_i* and *a_i* are input bits of this adder

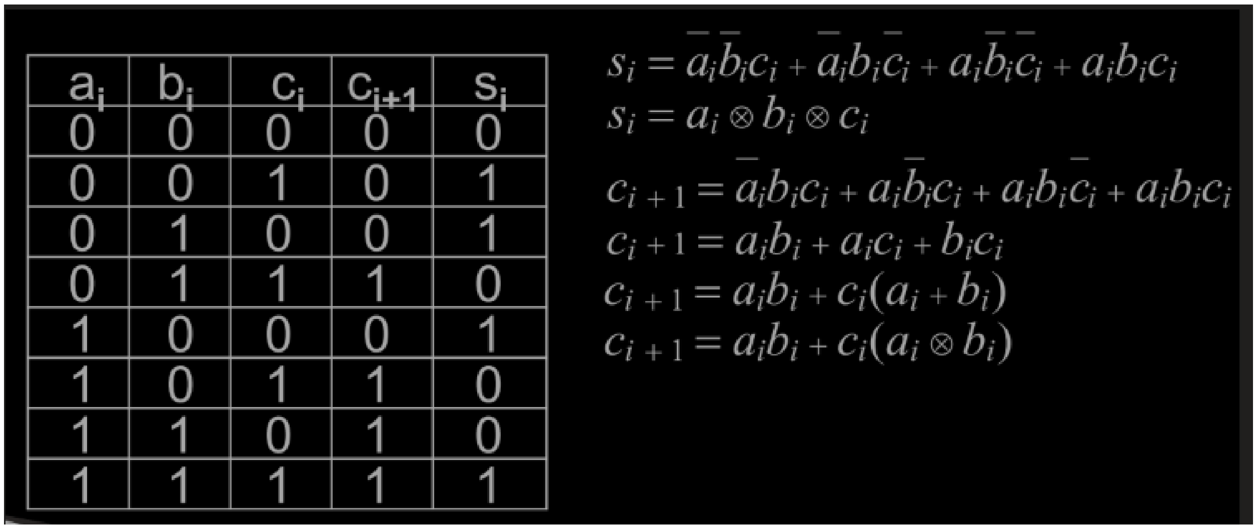
*c_i* is the carry in of this addition

*c_i+1* is the carry out of this addition



*s_i* is the result of this addition

The truth table and boolean equations of the outputs are given as:

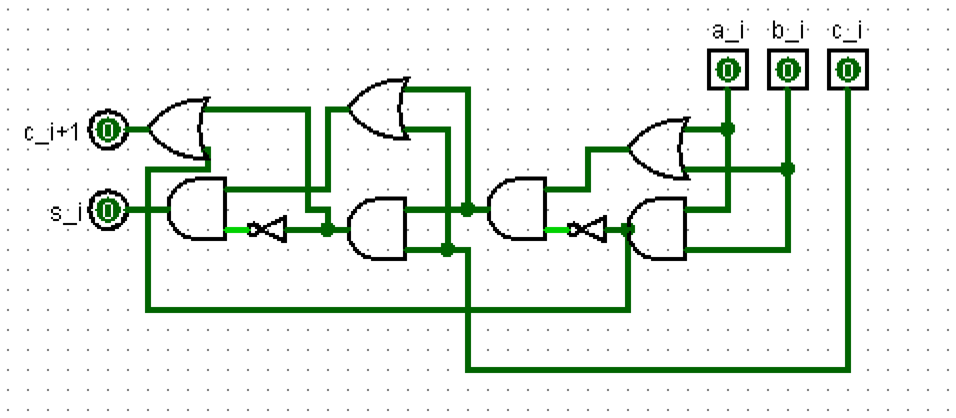


| $a_i$ | $b_i$ | $c_i$ | $c_{i+1}$ | $s_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

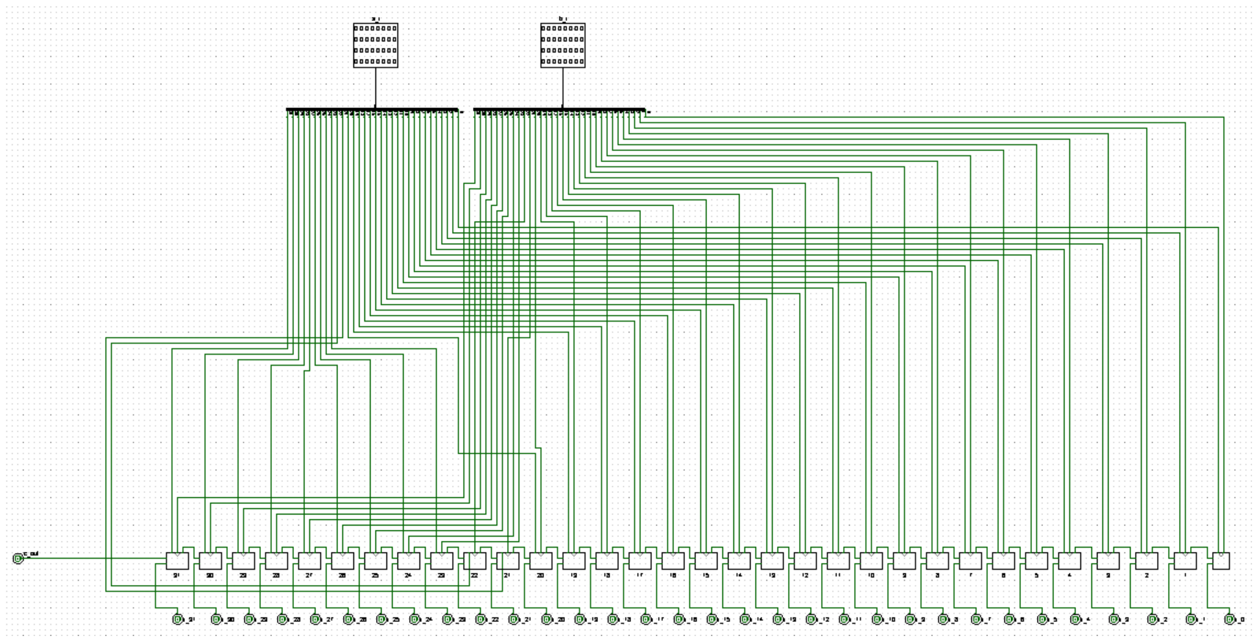
$$s_i = \overline{a_i}\,\overline{b_i}c_i + \overline{a_i}b_i\overline{c_i} + a_i\overline{b_i}\,\overline{c_i} + a_ib_ic_i$$
$$s_i = a_i \otimes b_i \otimes c_i$$

$$c_{i+1} = \overline{a_i}b_ic_i + a_i\overline{b_i}c_i + a_ib_i\overline{c_i} + a_ib_ic_i$$
$$c_i + 1 = a_ib_i + a_ic_i + b_ic_i$$
$$c_{i+1} = a_ib_i + c_i(a_i + b_i)$$
$$c_{i+1} = a_ib_i + c_i(a_i \otimes b_i)$$

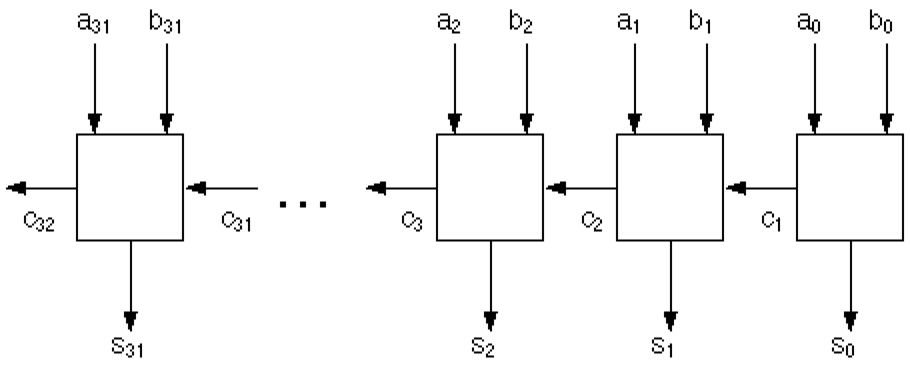
As a result the design of the full adder in logisim:



With using these two I can design 32 bit adder the design of the 32bit adder in logisim:

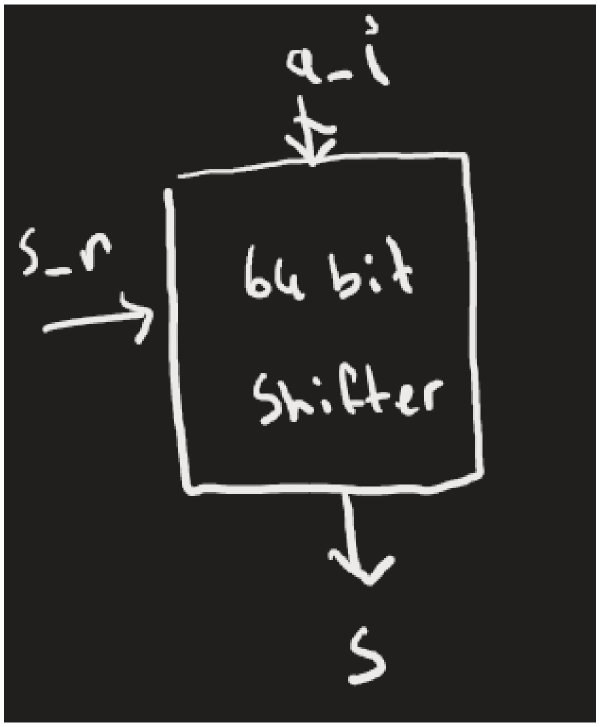


Since this design is easy to design but hard to observe I add a simple representation of the design approach



Since we designed the 32 bit adder now we need to design 64 shifter.

As you know shifting is a relative concept. To say something is shifted we need to assume some state as base and a thing in our concept can be shift to the right or left. In the multiplier example we need only a shifter which shifts the given  64 bit number to right by 1 bit.  We can show the input and outputs as:

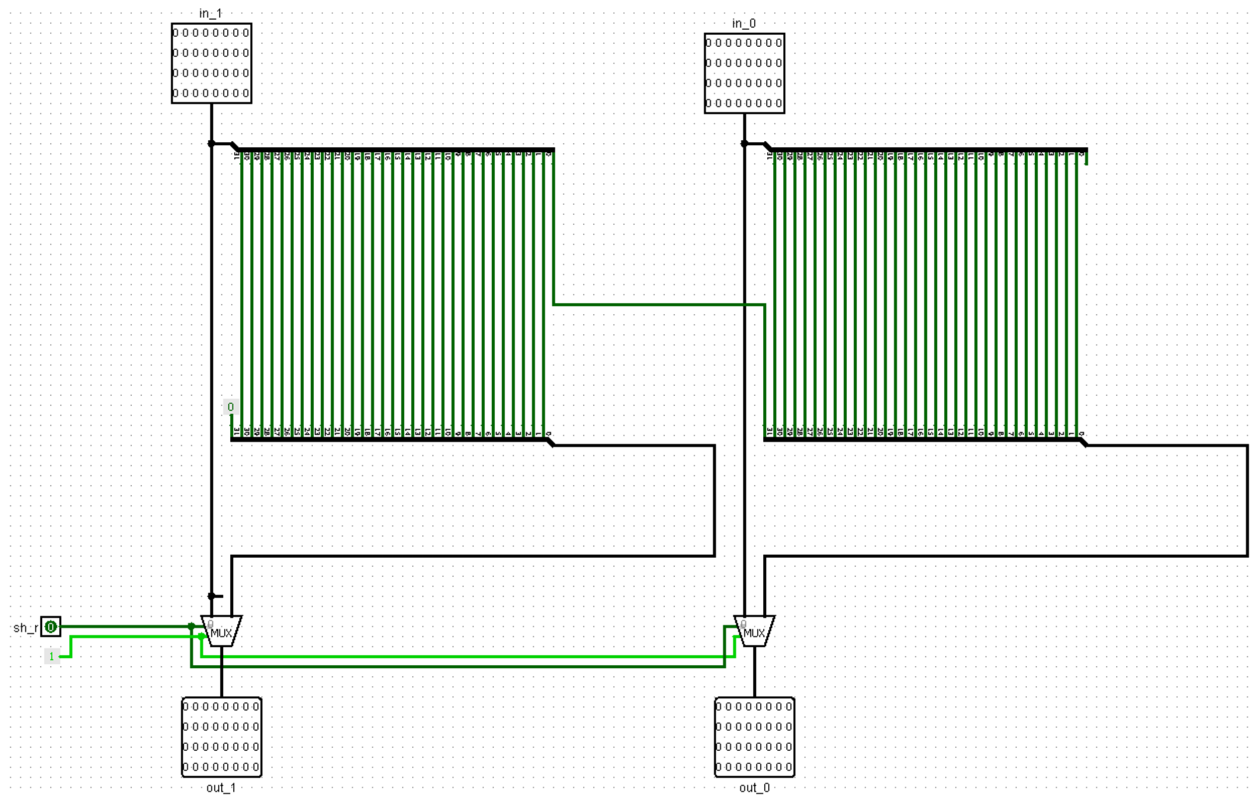


*a_i* is the 64 bit input for the 64 bit shifter

*s_r* is the signal which means shift the a_i if it is 1 and do not shift a_i if it is 0

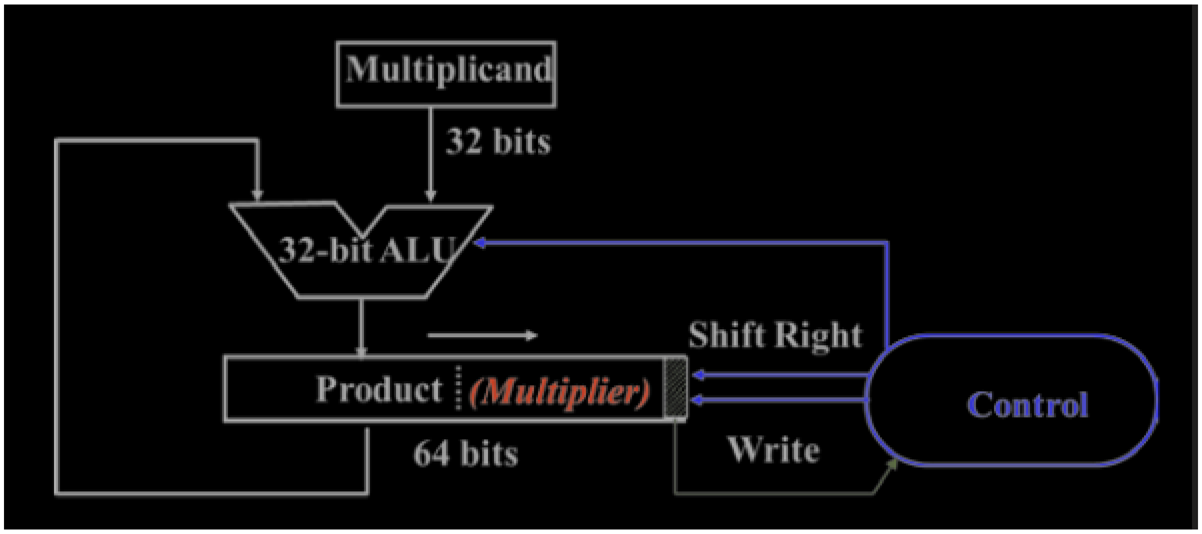
*s* is the result of the shifter

Design of the 64 bit 1 bit right shifter in logisim as:

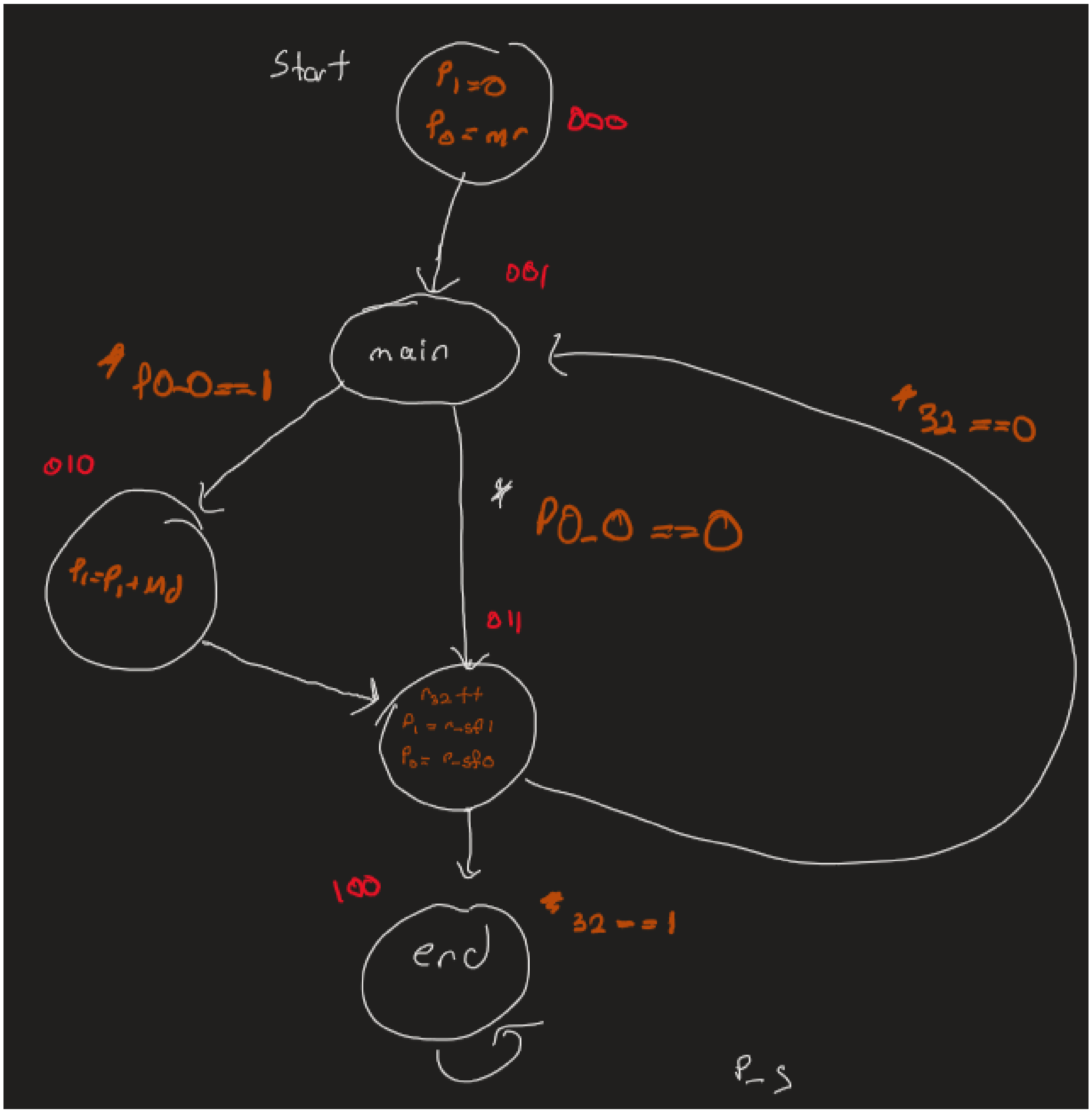


Since there is no 64 bit input or out in logisim I used two 32 bit inputs instead of 1 64 bit. to make right shifting by 1 for the result output we need to make decision between two cases do shifting or not. We solve this type of issues with multiplexers. I have 2 muxes and they have 32 bit data and 1 bit select. These muxes return the original number if the sh_r is 0 and returns shifted one if the sh_r is 1.

Now since we designed 32 bit adder and 64 bit shifter we can design 32 bit multiplier. We know that this will be sequential design. So we need to draw an FSM which is also given in book:



Since this FSM is only created for illustrative purposes, I created an FSM for myself using this one as base:



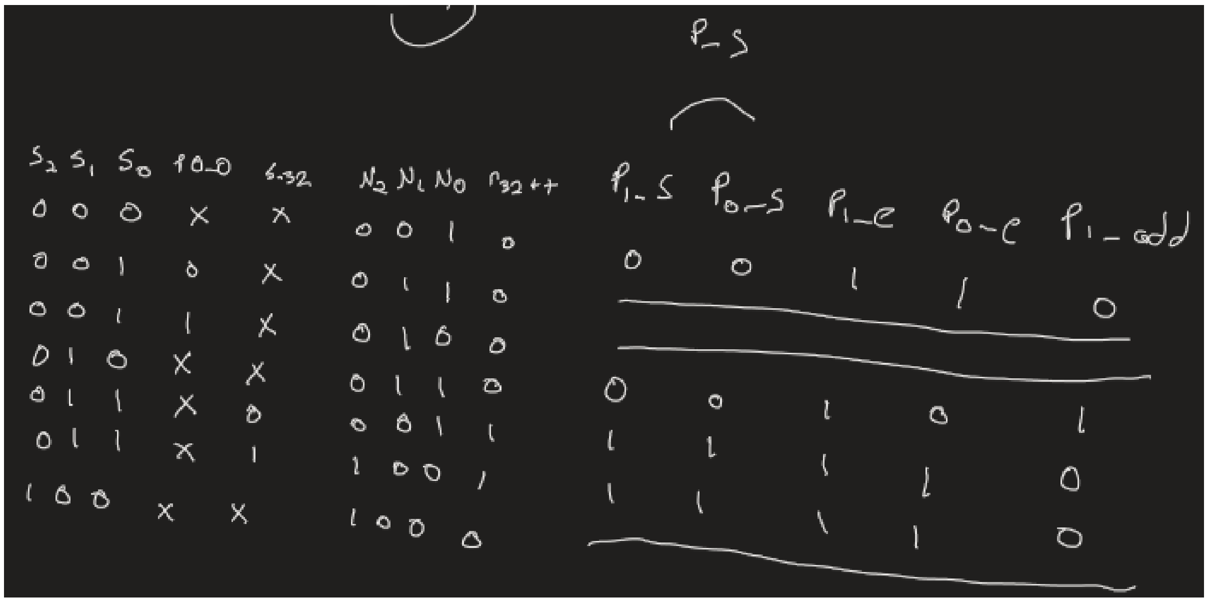
P1:  This represent the MSB 32 bit of the product

P0: This represents the LSB 32 bit of the product

P0_0: This represents the LSB of the product which is also the unused LSB of the multiplier.

*32: This represents that multiplier has right shifted 32 times

Using this FSM I created an Truth Table:



P_S

| $S_2$ | $S_1$ | $S_0$ | P0_0 | c=32 | $N_2$ | $N_1$ | $N_0$ | c32++ | P1_S | P0_S | P1_e | P0_e | P1_add |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | X | X | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | X | 0 | 1 | 1 | 0 | | | | | |
| 0 | 0 | 1 | 1 | X | 0 | 1 | 0 | 0 | | | | | |
| 0 | 1 | 0 | X | X | 0 | 1 | 1 | 0 | | | | | |
| 0 | 1 | 1 | X | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | X | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | X | X | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

P1_S: This signal determines the select bit of mux which determines the MSB 32  bit of the product. If it is 0 it assigns 0 and if it is 1 it assigns the result of the right shift by 1

P0_S:This signal determines the select bit of mux which determines the MSB 32  bit of the product. If it is 0 it assigns multiplier and if it is 1 it assigns the result of the right shift by 1

P0_E: Enable bit for register which holds P0

P1_E: Enable bit for register which holds P1

P1_add:  This signal determines the select bit of mux which determines the MSB 32  bit of the product. If it is 0 it assigns P1_S's result and if it is 1 it assigns the P1+Multiplicand

Derived boolean equations from truth table:

$$N_2 = s_2' s_1 s_0 s\_32 + s_2 s_1' s_0'$$

$$N_1 = s_2' s_1' s_0 \underbrace{(PO\_0' + PO\_0)}_{1} + s_2' s_1 s_0' = s_2' (\overbrace{s_1' s_0 + s_1 s_0'}^{s_1 \oplus s_0})$$

$$N_0 = \underbrace{s_2' s_1' s_0'} + s_2' s_1' s_0 PO\_0' + \underbrace{s_2' s_1 s_0'} + s_2' s_1 s_0 s\_32'$$

$$s_2' s_0' \underbrace{(s_1' + s_1)}_{1} + s_2' s_0 (s_1' PO\_0' + s_1 s\_32')$$

$$s_2' s_0' + s_2' s_0 (s_1' PO\_0' + s_1 s\_32')$$

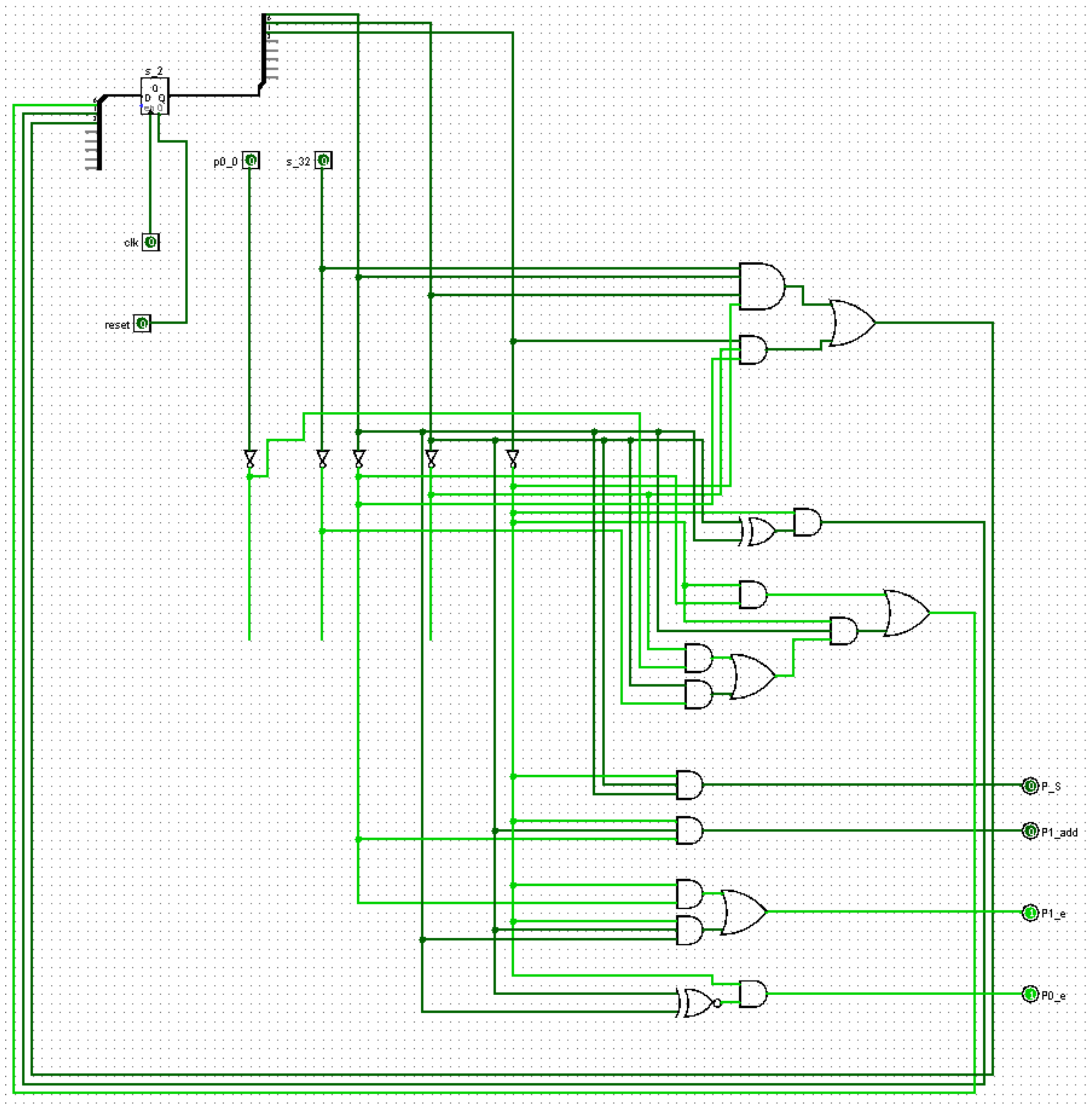
$$P\_S = s_2' s_1 s_0$$

$$P_1\_e = s_2' s_1' s_0' + s_2' s_1 s_0' + s_2' s_1 s_0 = s_2' s_0' + s_2' s_1 s_0$$

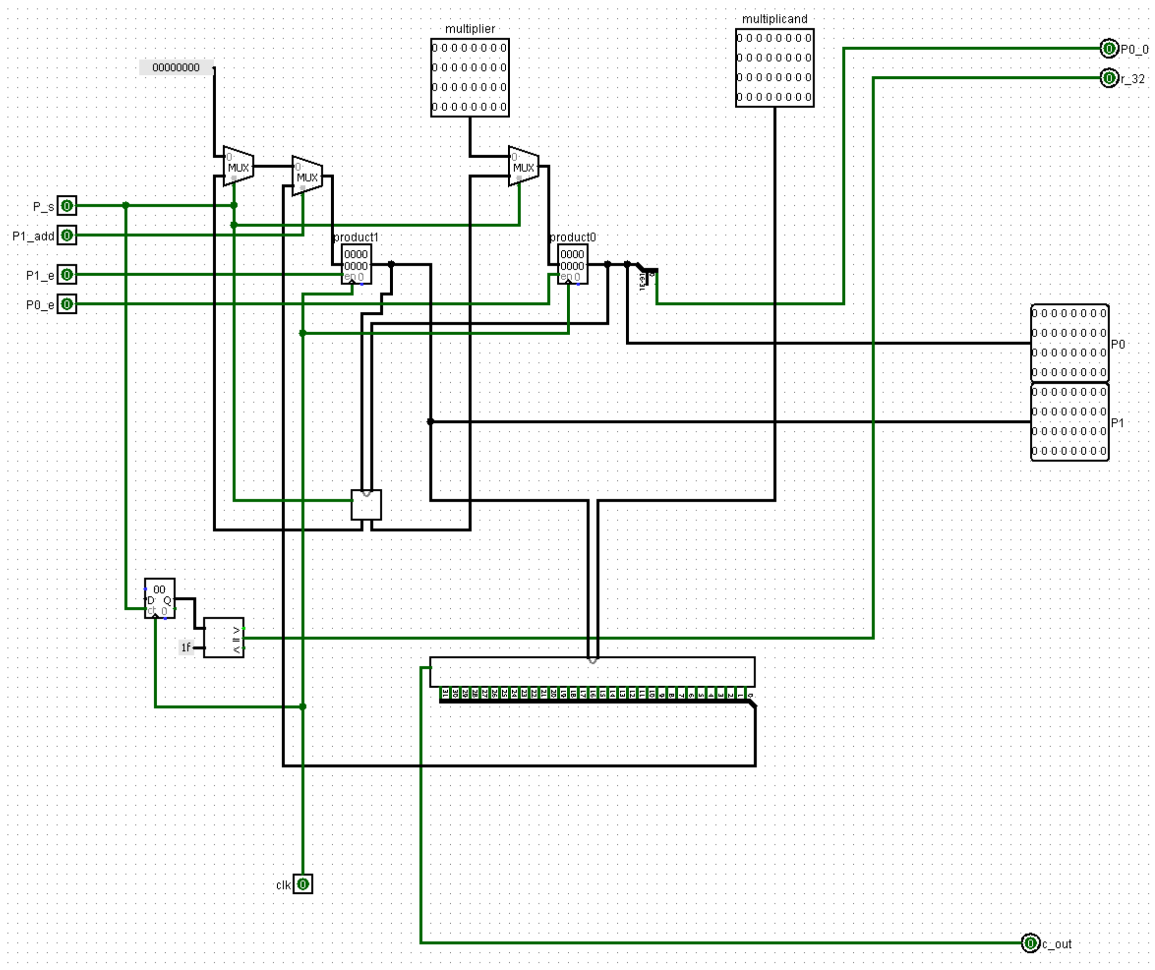
$$P_0\_e = s_2' s_1' s_0' + s_2' s_1 s_0 = s_2' (\underbrace{s_1' s_0' + s_1 s_0}_{s_1 \text{ XNOR } s_0})$$

$$P_1 \text{odd} = s_2' s_1 s_0'$$

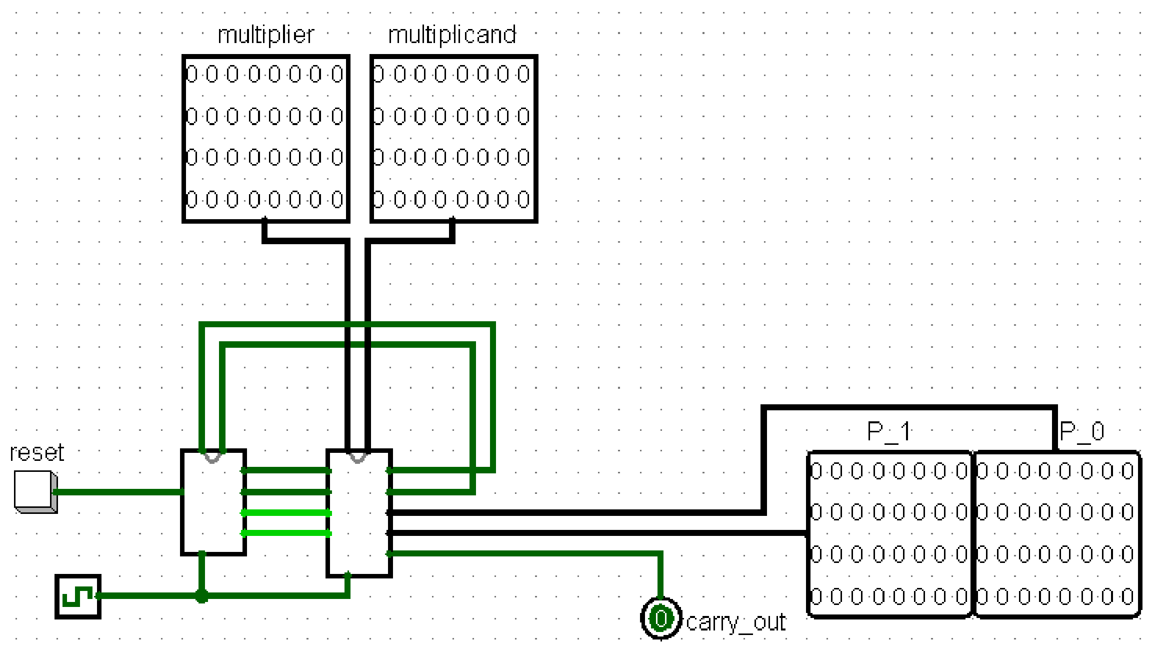
Using this signals I designed the control unit of the multiplier:



Then I designed datapath:



With combining control unit and datapath I can design 32 bit multiplier:


multiplier
multiplicand
reset
P_1
P_0
carry_out