# Pipeline

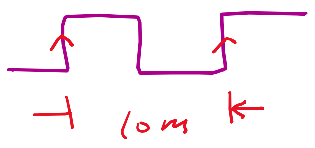
# Pipeline

- ***Pipeline is an important technique used in (DSP) systems, microprocessors, etc.***

$f_{clk} = 100\,MHz$

- ***Pipeline results in speed enhancement for the critical path.***

$T_{clk} = 10\,(ns)$



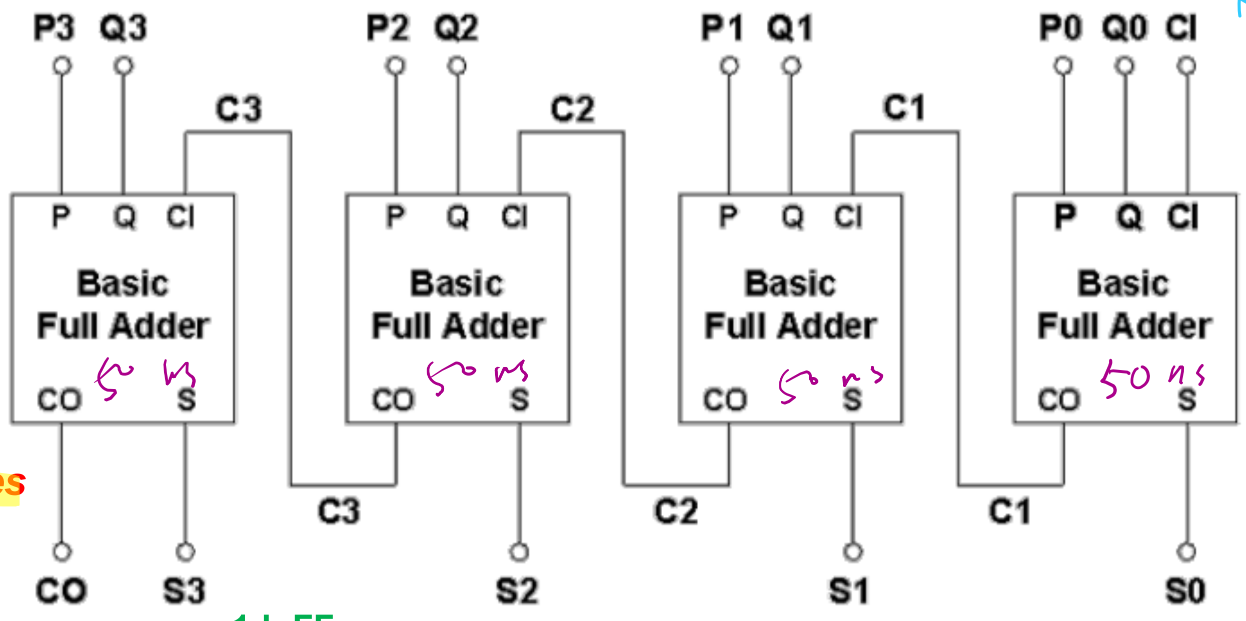
- ***It can either increase the clock speed or reduce the power consumption at the same speed in a DSP system.***

- ***It increases the throughput of the system when processing a stream of tasks.***

## Pipeline

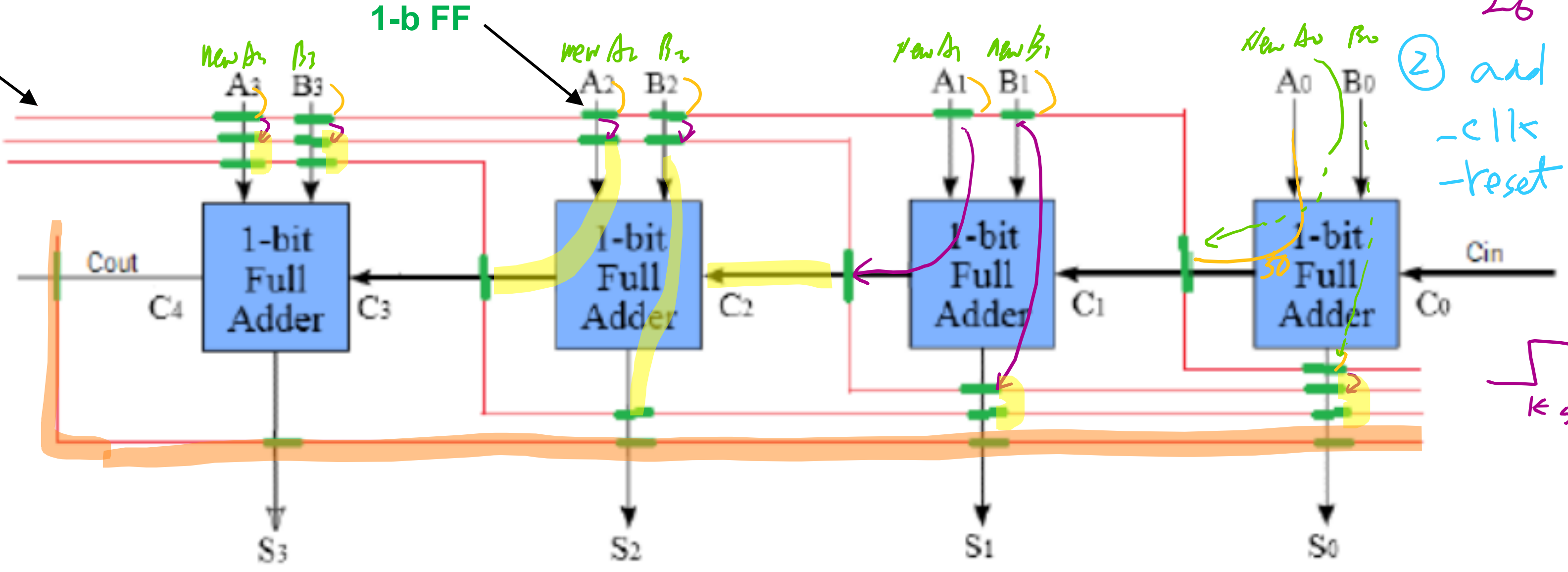
| | |
|---|---|
| 10 ns | 100 MHz |
| 100 | 10 MHz |
| 200 | 5 MHz |
| 50 | 20 MHz |

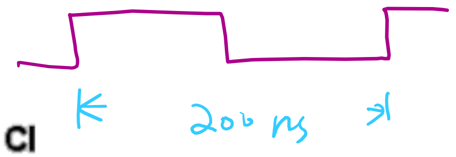
Non-pipeline Adder vs Pipeline Adder



- *Insert 4 pipelines*
- *Insert ~~24~~ 26 FF's*

Pipeline



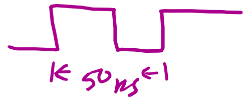


Pipeline hardware overhead

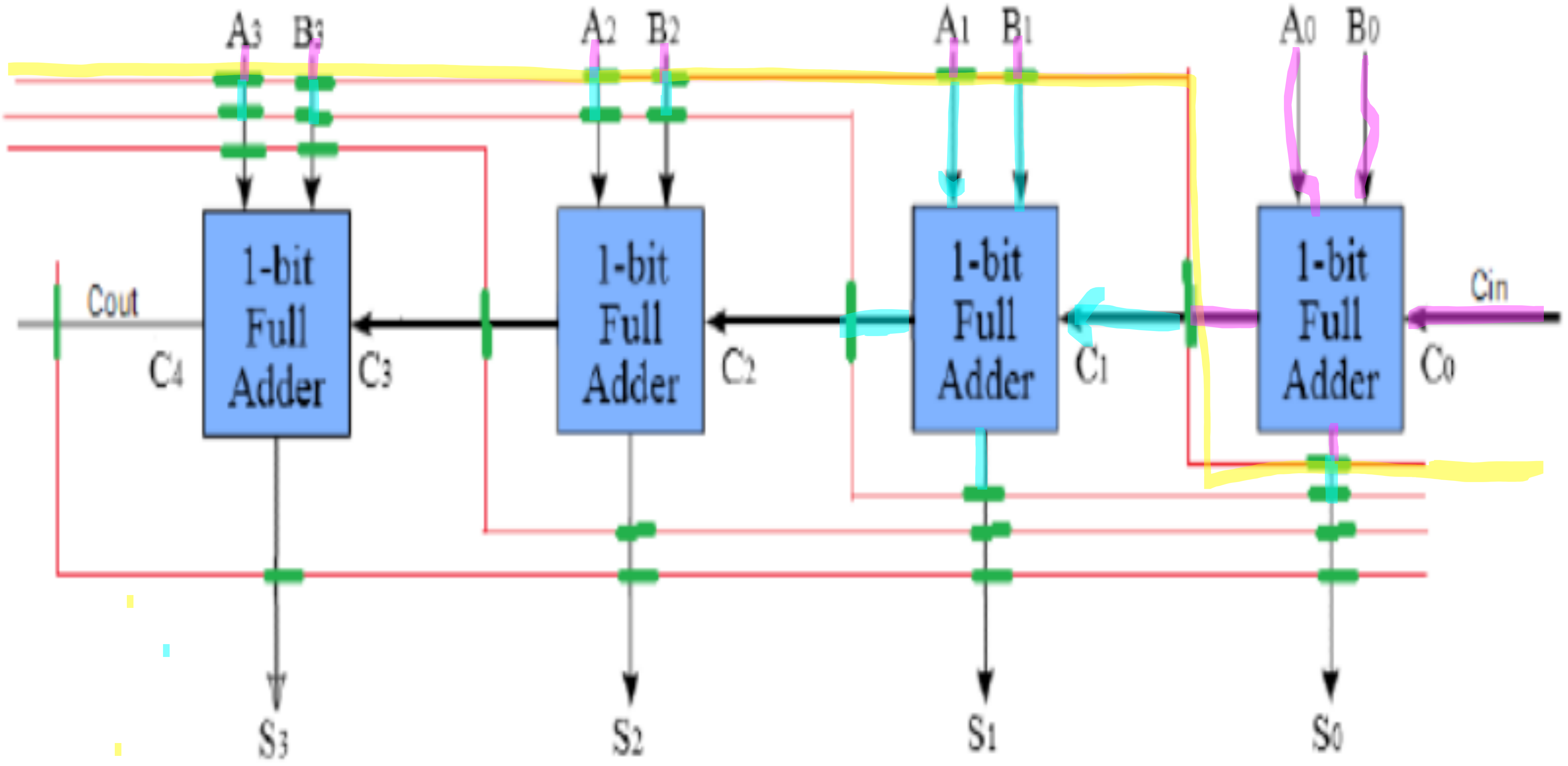
① 8
7
6
+ 5

26 D-FF's

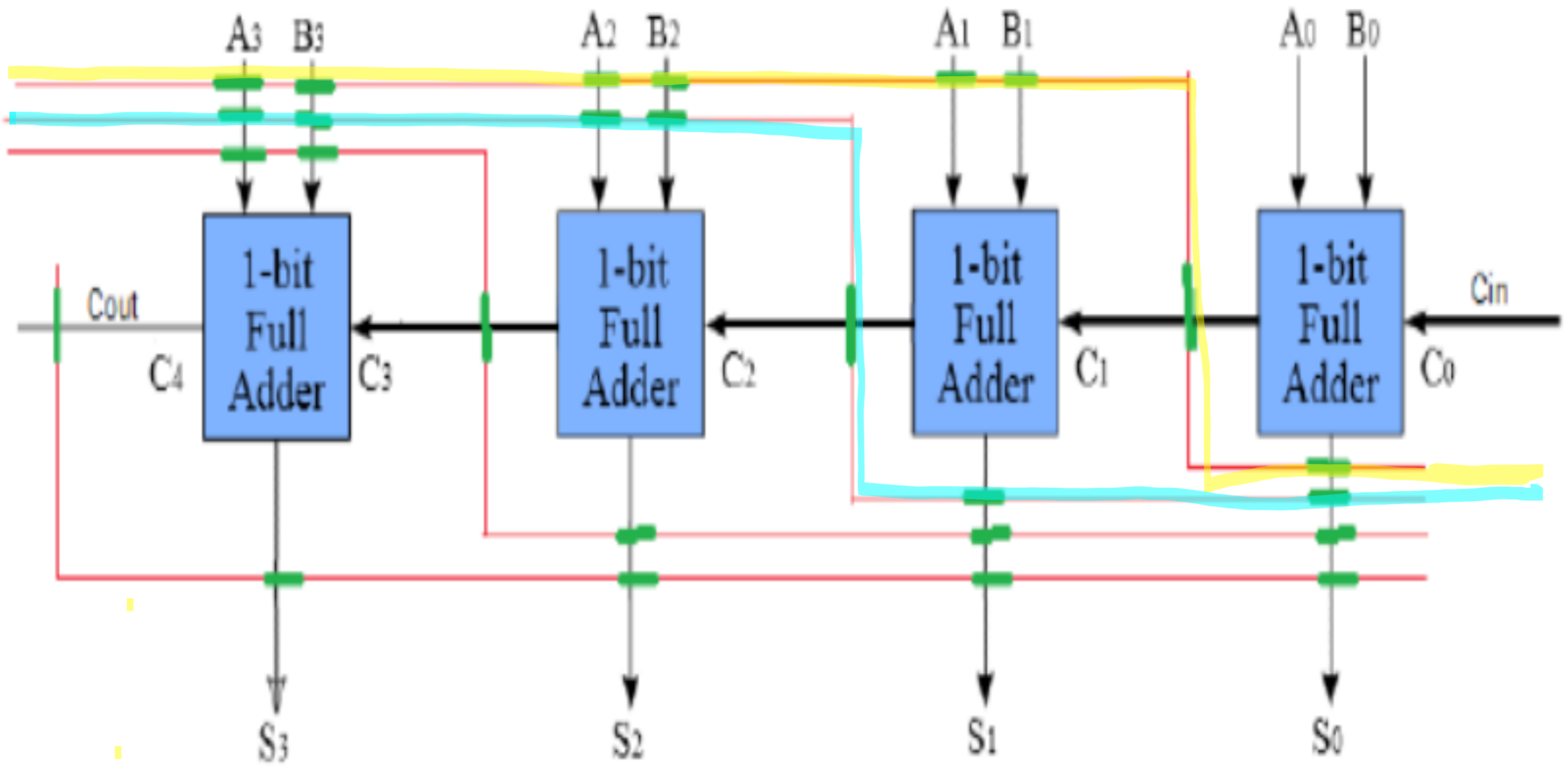
② and
– clk
– reset or resetn



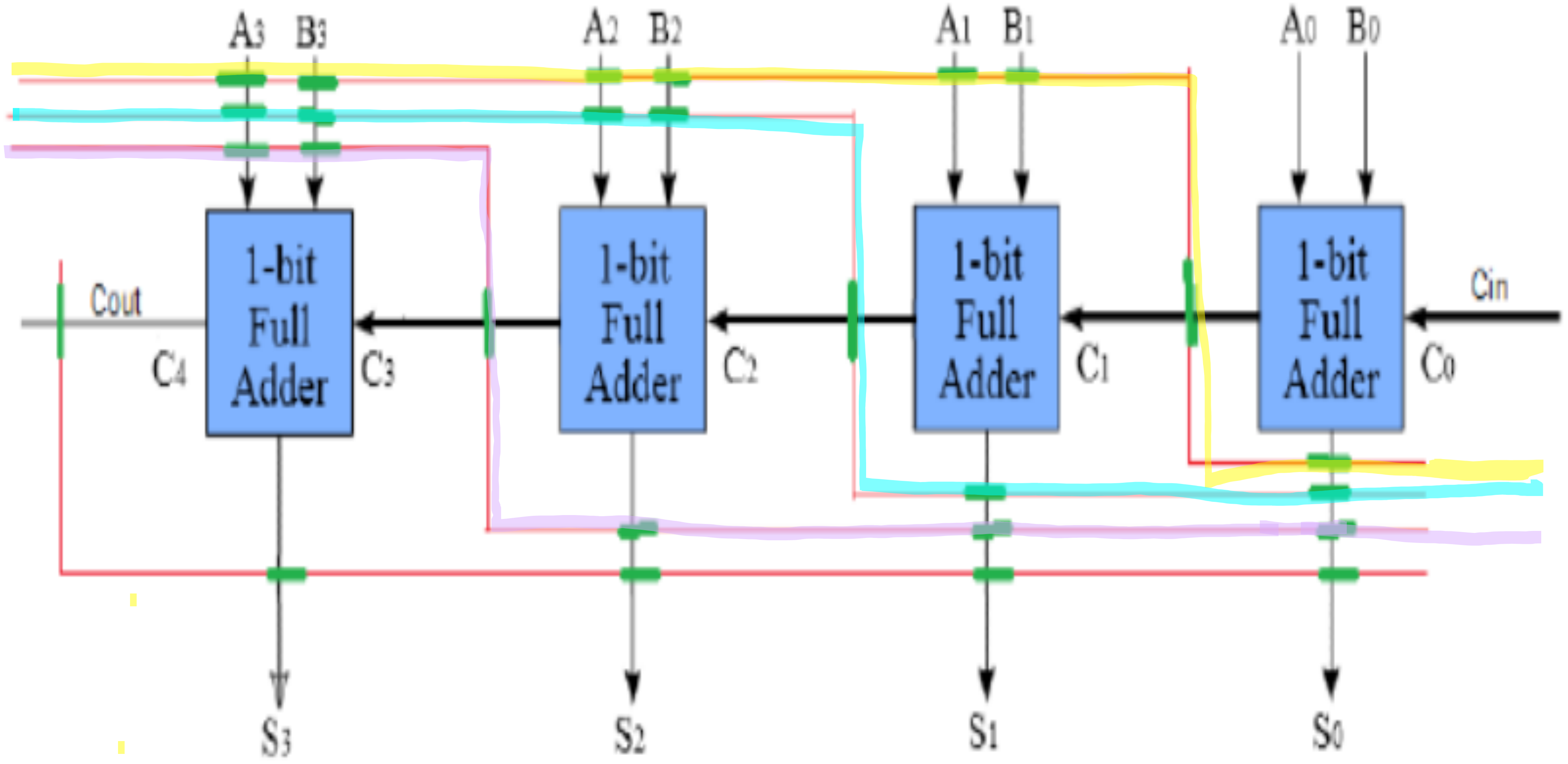
*1st clock*

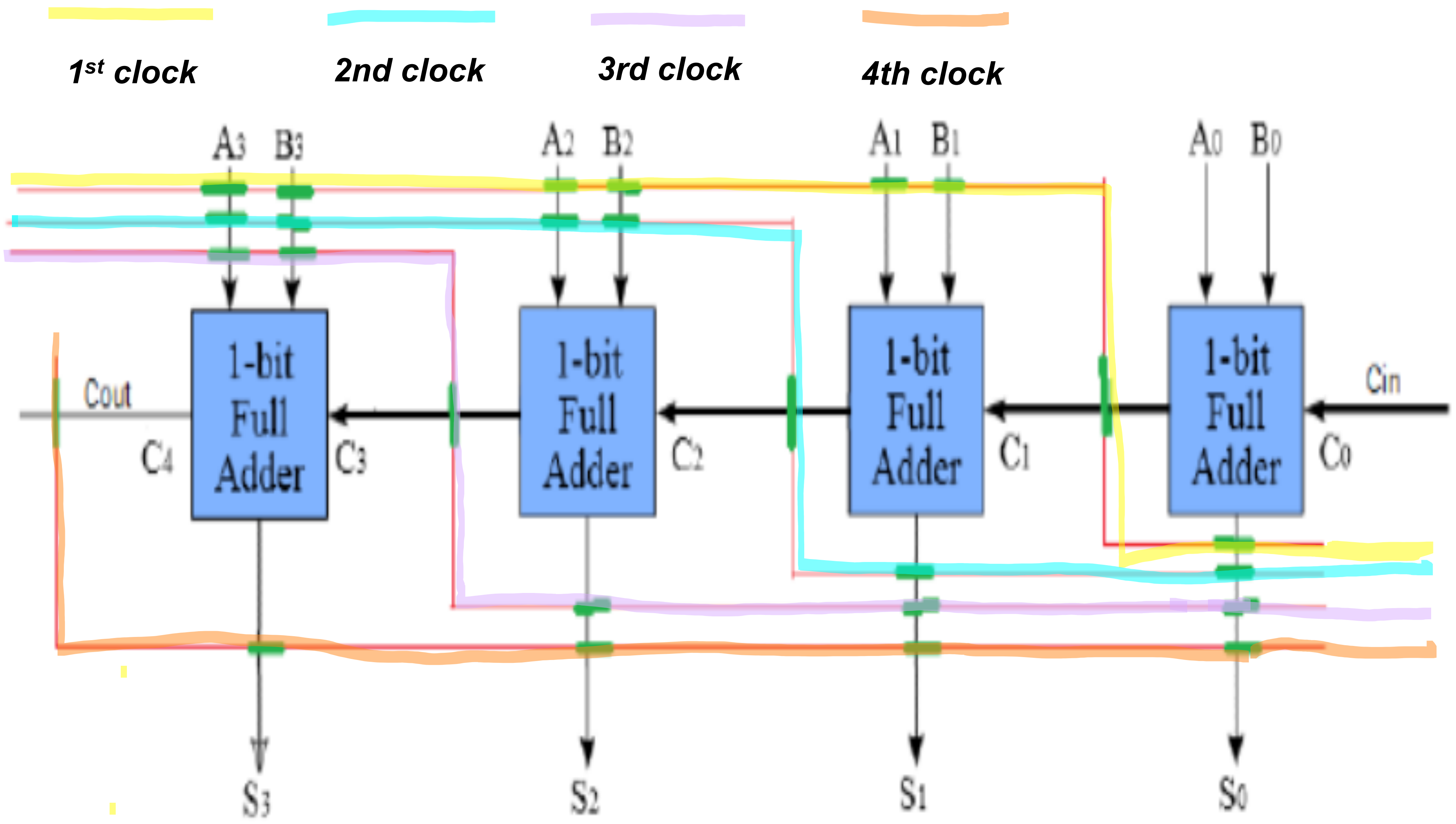


A3
B3
A2
B2
A1
B1
A0
B0
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
Cout
C4
C3
C2
C1
C0
Cin
S3
S2
S1
S0


1st clock
2nd clock
A3
B3
A2
B2
A1
B1
A0
B0
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
Cout
C4
C3
C2
C1
C0
Cin
S3
S2
S1
S0

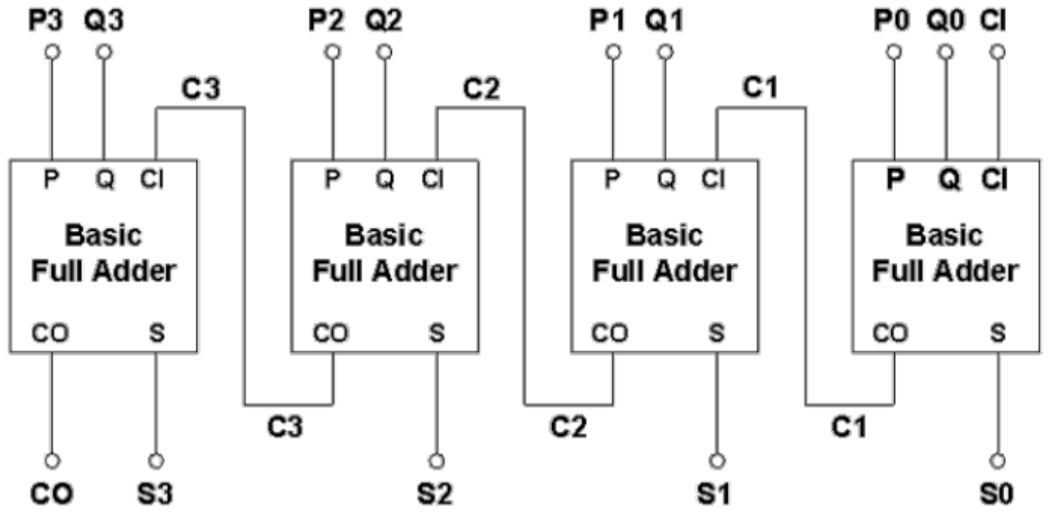
🟨 *1st clock* 🟦 *2nd clock* 🟪 *3rd clock*

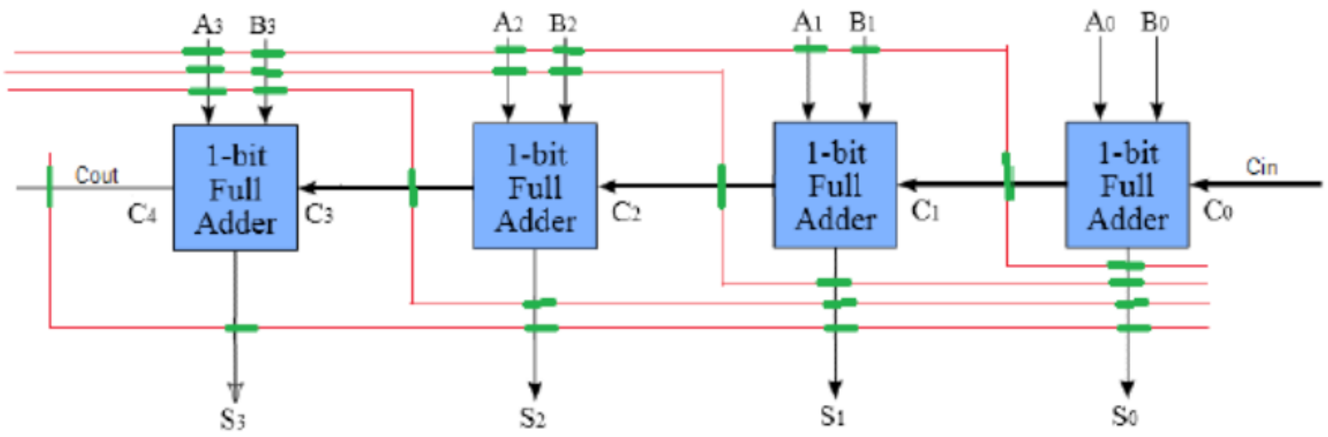



1st clock
2nd clock
3rd clock
4th clock
A3 B3
A2 B2
A1 B1
A0 B0
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
Cout
C4
C3
C2
C1
C0
Cin
S3
S2
S1
S0

# Pipeline

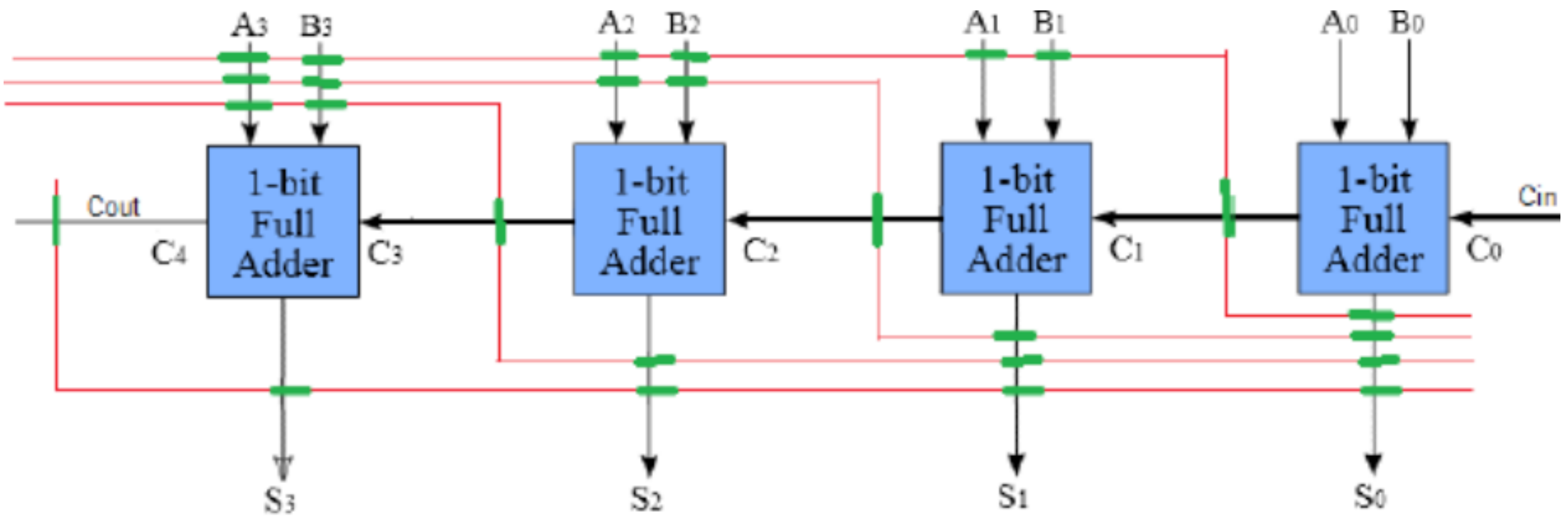
Non-pipeline Adder vs Pipeline Adder





Assume 1-b FA delay is 10 ns.

- Non-pipeline: It takes 40 ns to complete 4-b addition => f = 1/T = 1/40ns = 25 MHz.
- Pipeline: It takes the first 4 clock cycles to generate the first 4-b addition result; thereafter, It generates 4-b addition result every clock cycle. => f = 1/T = 1/10ns = 100 MHz.



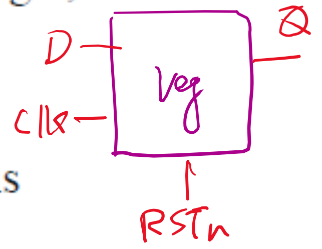
```
entity FA is
PORT (Cin, A, B: IN STD_LOGIC ;
 Cout, S: OUT STD_LOGIC ) ;
END entity;

ARCHITECTURE behaviour OF FA is
begin
   S <= A XOR B XOR Cin ;
   Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;
end architecture;
```

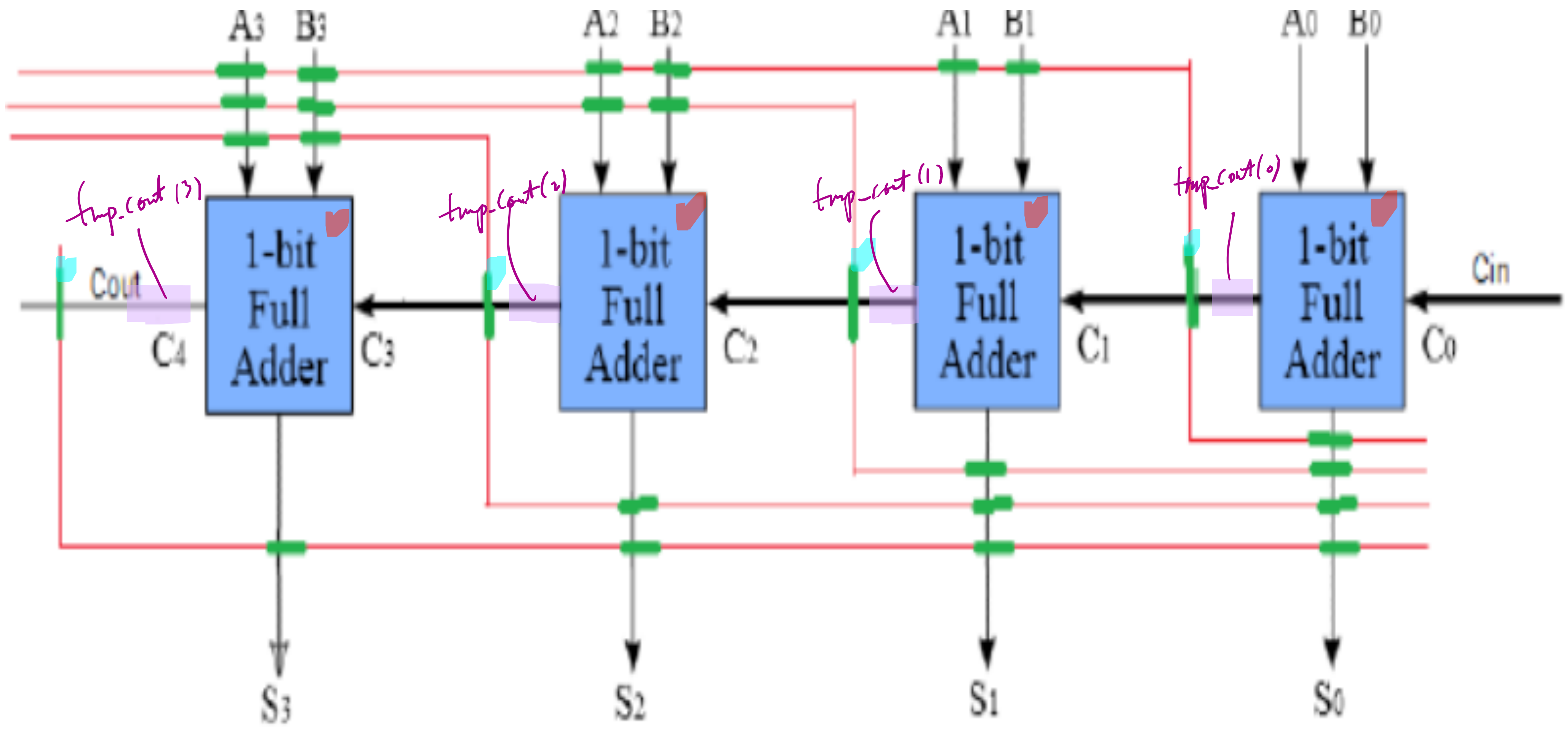
```
entity reg is
 port(D, CLK, RSTn: in std_logic;
 Q: out std_logic);
end entity;

architecture behaviour of reg is
begin
    process(CLK, RSTn)
    begin
        if RSTn = '0' then
          Q <= '0';

        elsif rising_edge(clk) then
          Q <= D;
        end if;
    end process;
end architecture;
```



clk'EVENT AND clk='1'



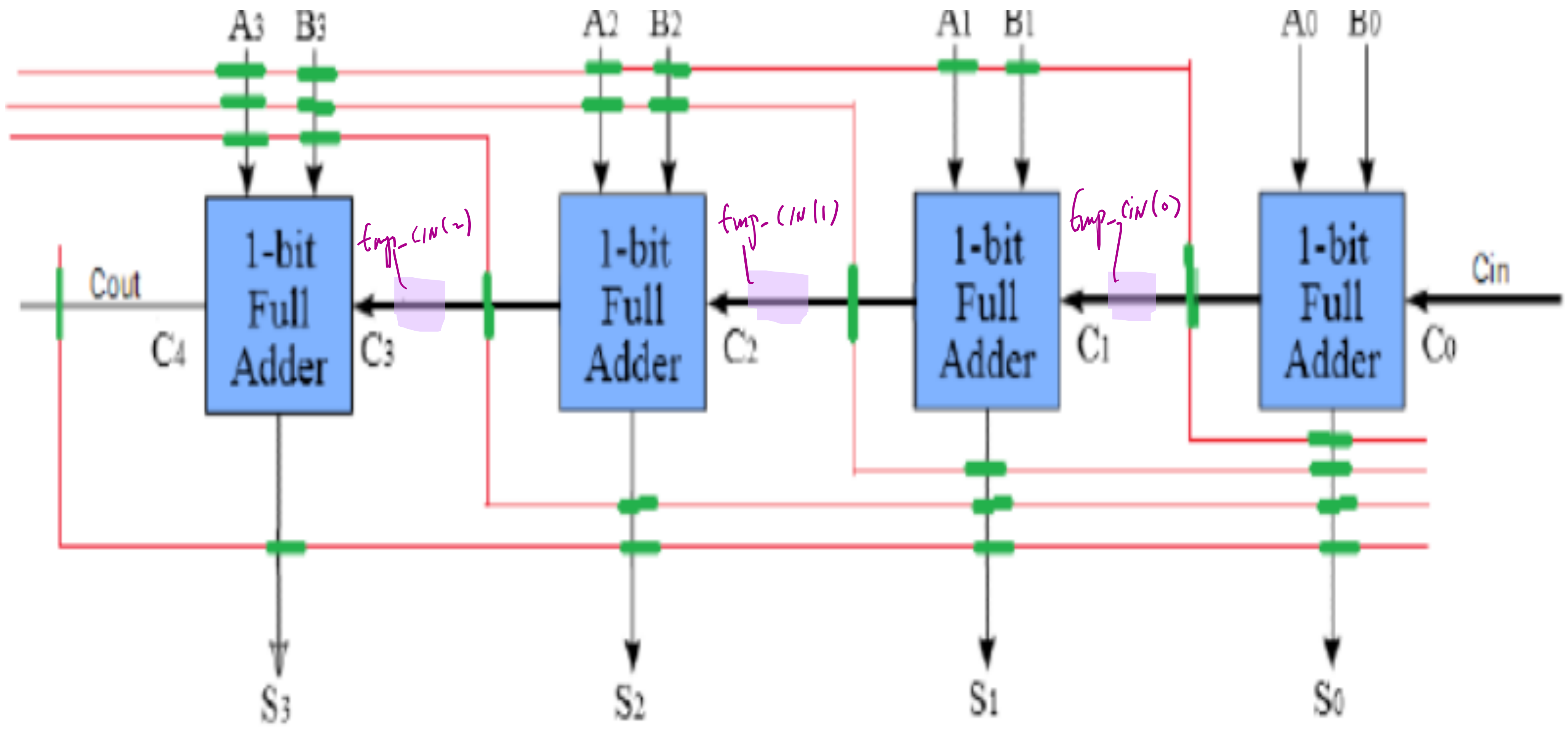
-- **Declaration of signals used to interconnect gates**

```
signal tmp_cout : std_logic_vector(3 downto 0);
signal tmp_cin : std_logic_vector(2 downto 0);
signal tmp_S0 : std_logic_vector(3 downto 0);
signal tmp_S1 : std_logic_vector(2 downto 0);
signal tmp_S2 : std_logic_vector(1 downto 0);
signal tmp_S3 : std_logic;
signal tmp_A1 : std_logic;
signal tmp_B1 : std_logic;
```

```
reg_cout0: reg port map(tmp_cout(0), CLK, RSTn, tmp_cin(0));
reg_cout1: reg port map(tmp_cout(1), CLK, RSTn, tmp_cin(1));
reg_cout2: reg port map(tmp_cout(2), CLK, RSTn, tmp_cin(2));
reg_cout3: reg port map(tmp_cout(3), CLK, RSTn, Cout);

FA_0: FA port map(Cin=>Cin, A=>A(0), B=>B(0), S=>tmp_S0(0),
Cout=>tmp_cout(0));
FA_1: FA port map(Cin=>tmp_cin(0), A=>tmp_A1, B=>tmp_B1, S=>tmp_S1(0),
Cout=>tmp_cout(1));
FA_2: FA port map(Cin=>tmp_cin(1), A=>tmp_A2(1), B=>tmp_B2(1),
S=>tmp_S2(0), Cout=>tmp_cout(2));
FA_3: FA port map(Cin=>tmp_cin(2), A=>tmp_A3(2), B=>tmp_B3(2), S=>tmp_S3,
Cout=>tmp_cout(3));
```

⟸ port map by "NAME"



**-- Declaration of signals used to interconnect gates**

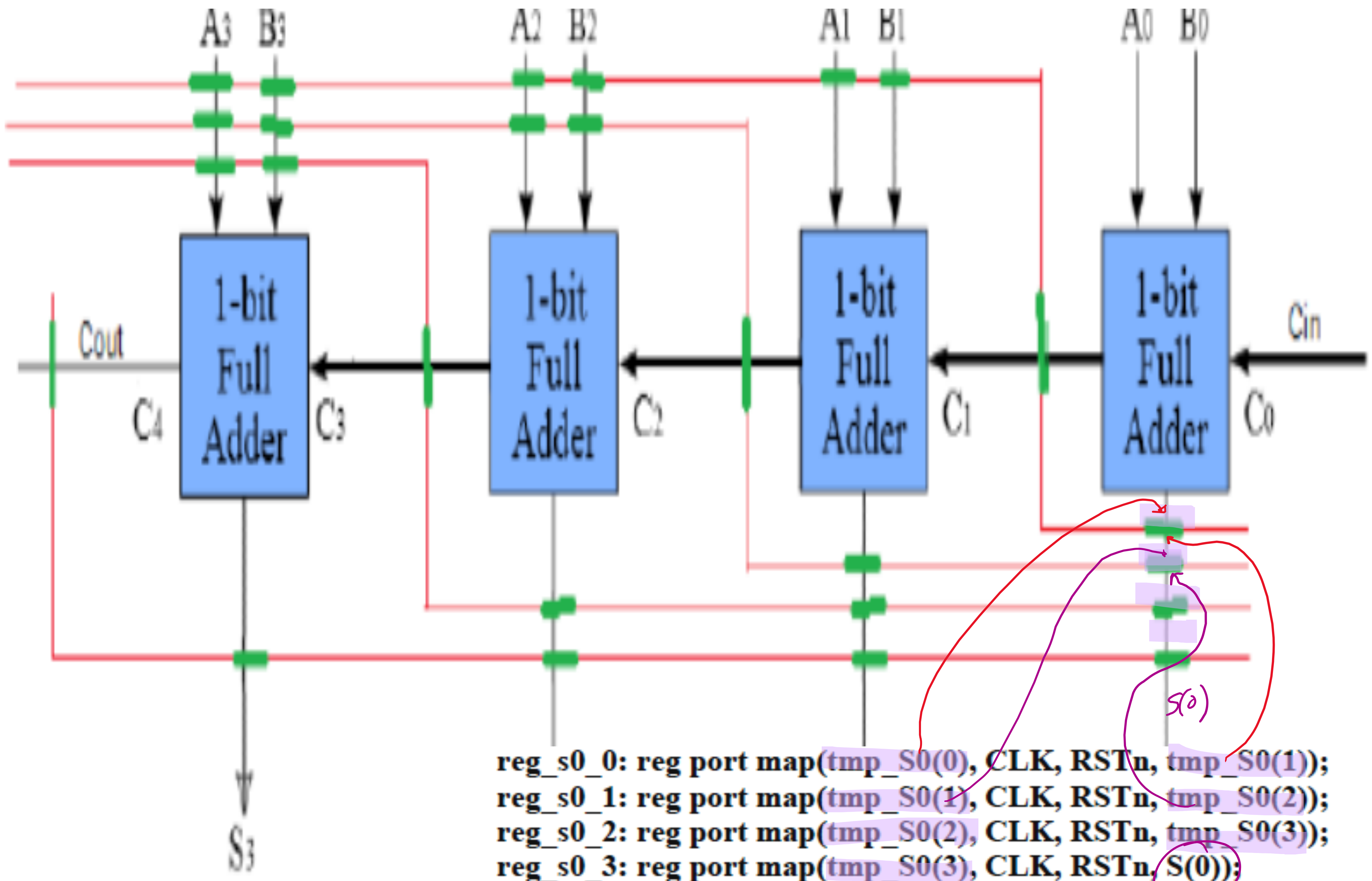
```
signal tmp_cout : std_logic_vector(3 downto 0);
signal tmp_cin : std_logic_vector(2 downto 0);
signal tmp_S0 : std_logic_vector(3 downto 0);
signal tmp_S1 : std_logic_vector(2 downto 0);
signal tmp_S2 : std_logic_vector(1 downto 0);
signal tmp_S3 : std_logic;
signal tmp_A1 : std_logic;
signal tmp_B1 : std_logic;
```

3

```
FA_1: FA port map(Cin=>tmp_cin(0), A=>tmp_A1, B=>tmp_B1, S=>tmp_S1(0),
Cout=>tmp_cout(1));
FA_2: FA port map(Cin=>tmp_cin(1), A=>tmp_A2(1), B=>tmp_B2(1),
S=>tmp_S2(0), Cout=>tmp_cout(2));
FA_3: FA port map(Cin=>tmp_cin(2), A=>tmp_A3(2), B=>tmp_B3(2), S=>tmp_S3,
Cout=>tmp_cout(3));
```

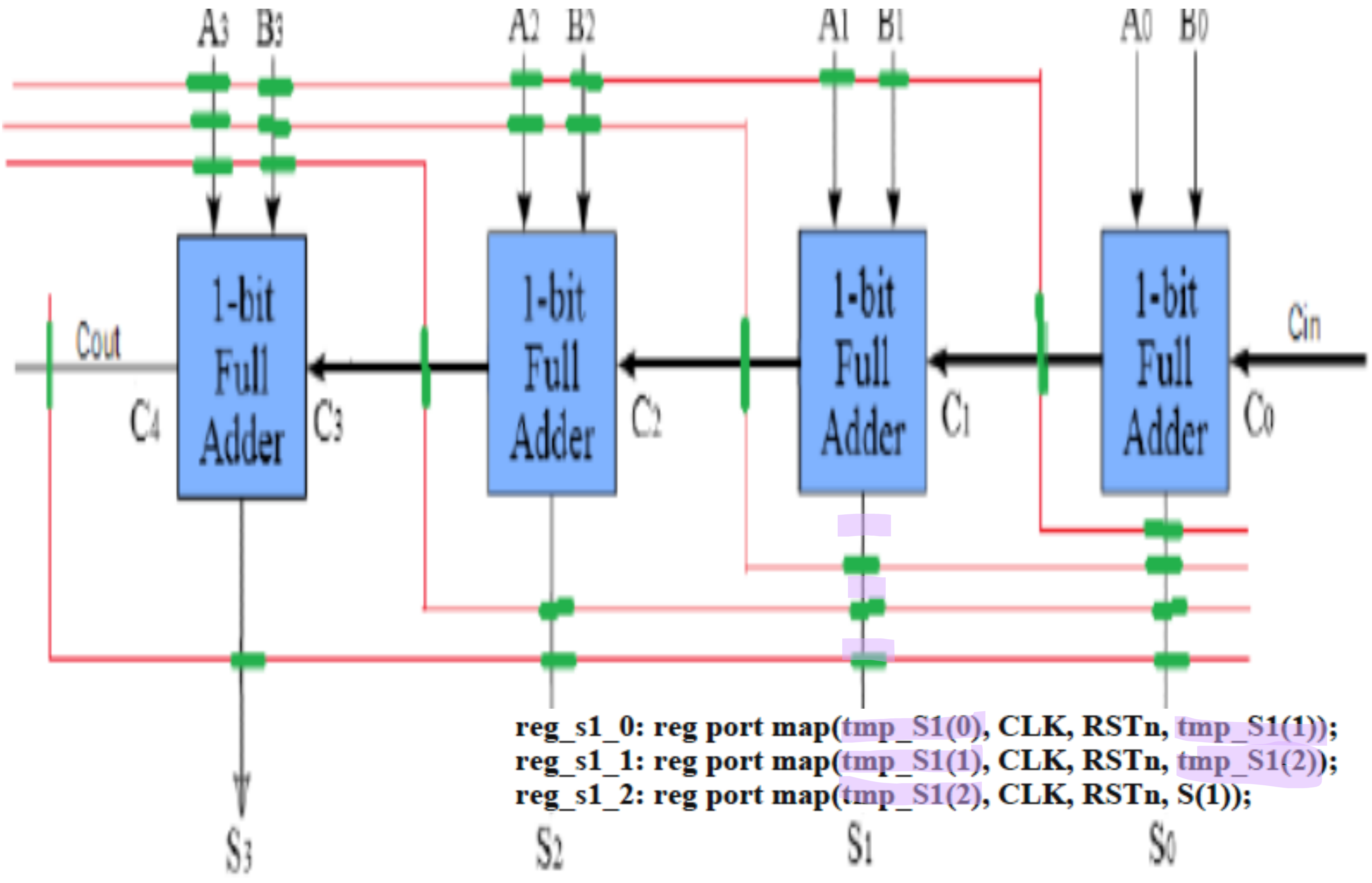
3

```
reg_cout0: reg port map(tmp_cout(0), CLK, RSTn, tmp_cin(0));
reg_cout1: reg port map(tmp_cout(1), CLK, RSTn, tmp_cin(1));
reg_cout2: reg port map(tmp_cout(2), CLK, RSTn, tmp_cin(2));
```



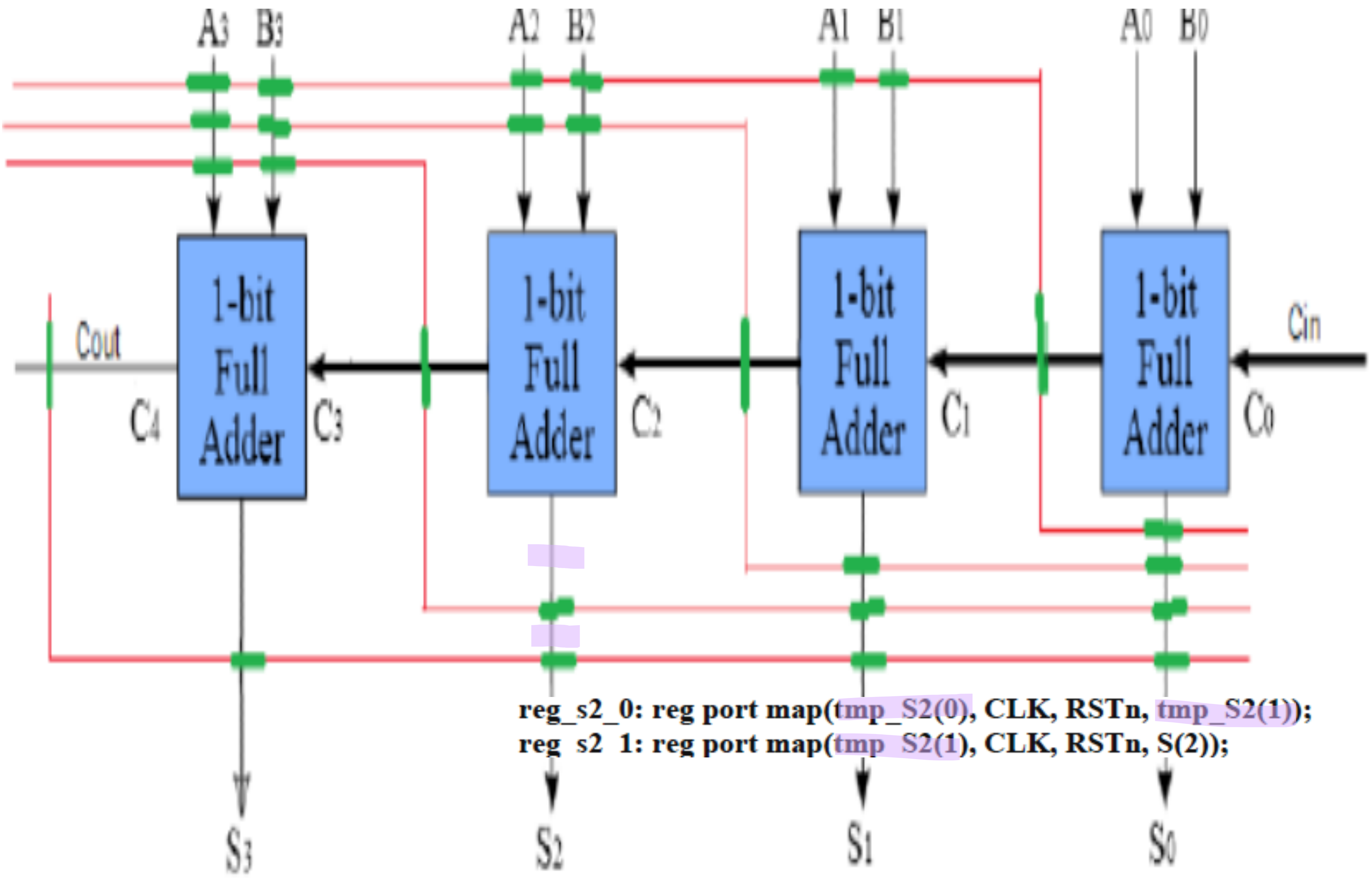
```
reg_s0_0: reg port map(tmp_S0(0), CLK, RSTn, tmp_S0(1));
reg_s0_1: reg port map(tmp_S0(1), CLK, RSTn, tmp_S0(2));
reg_s0_2: reg port map(tmp_S0(2), CLK, RSTn, tmp_S0(3));
reg_s0_3: reg port map(tmp_S0(3), CLK, RSTn, S(0));
```

```
signal tmp_S0 : std_logic_vector(3 downto 0);
```



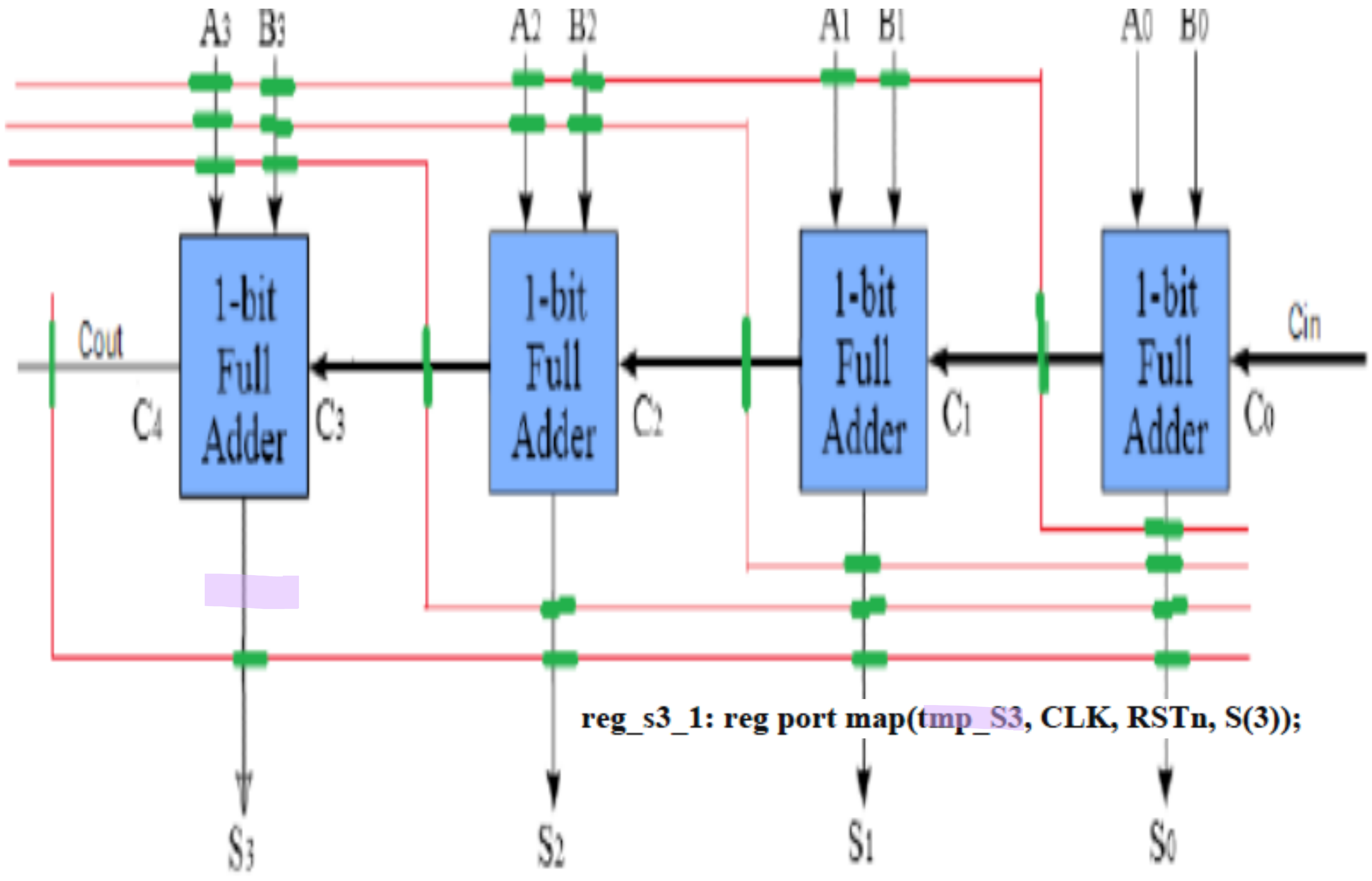
```
reg_s1_0: reg port map(tmp_S1(0), CLK, RSTn, tmp_S1(1));
reg_s1_1: reg port map(tmp_S1(1), CLK, RSTn, tmp_S1(2));
reg_s1_2: reg port map(tmp_S1(2), CLK, RSTn, S(1));
```

```
signal tmp_S1 : std_logic_vector(2 downto 0);
```



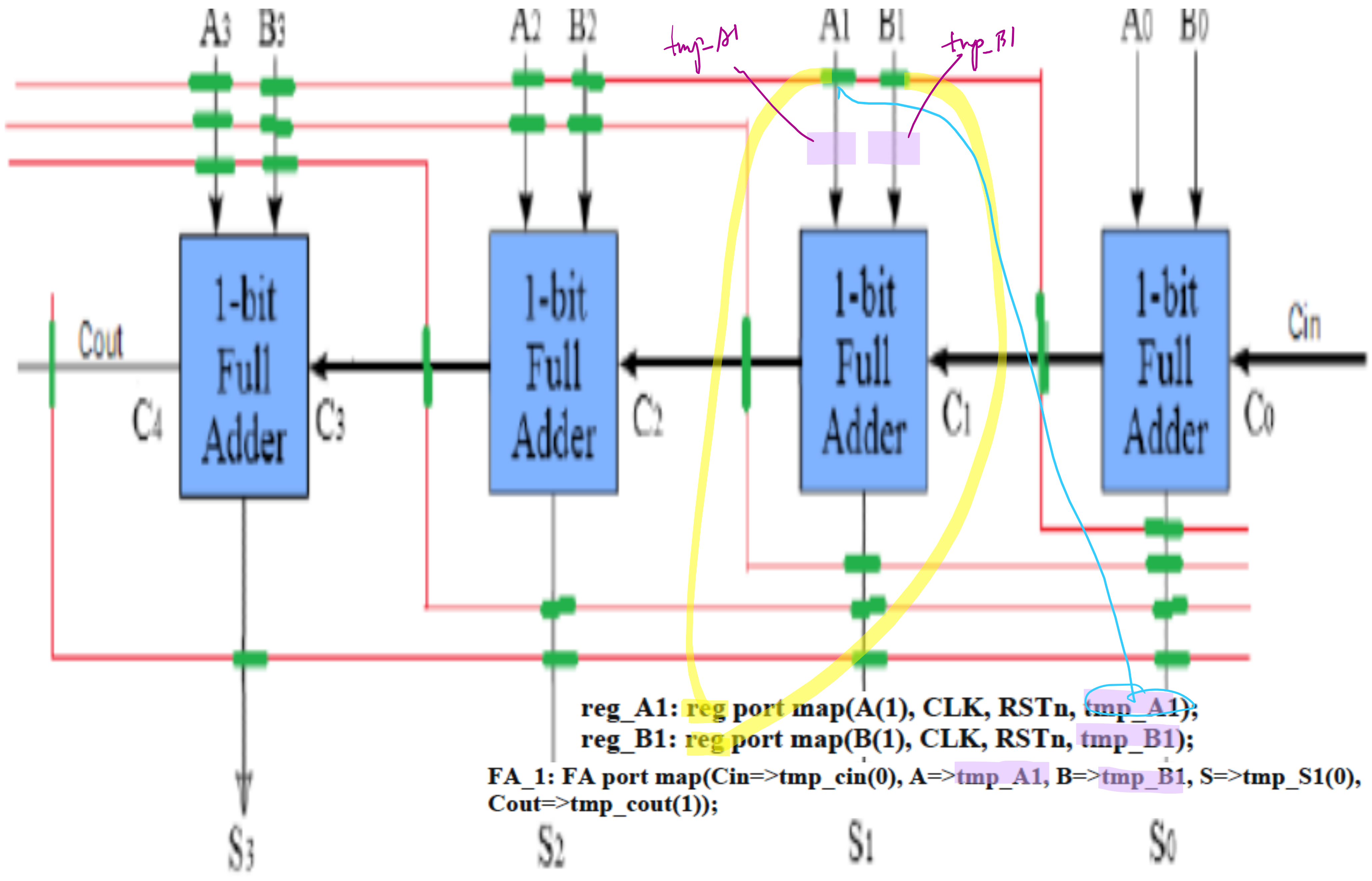
```
reg_s2_0: reg port map(tmp_S2(0), CLK, RSTn, tmp_S2(1));
reg_s2_1: reg port map(tmp_S2(1), CLK, RSTn, S(2));
```

```
signal tmp_S2 : std_logic_vector(1 downto 0);
```



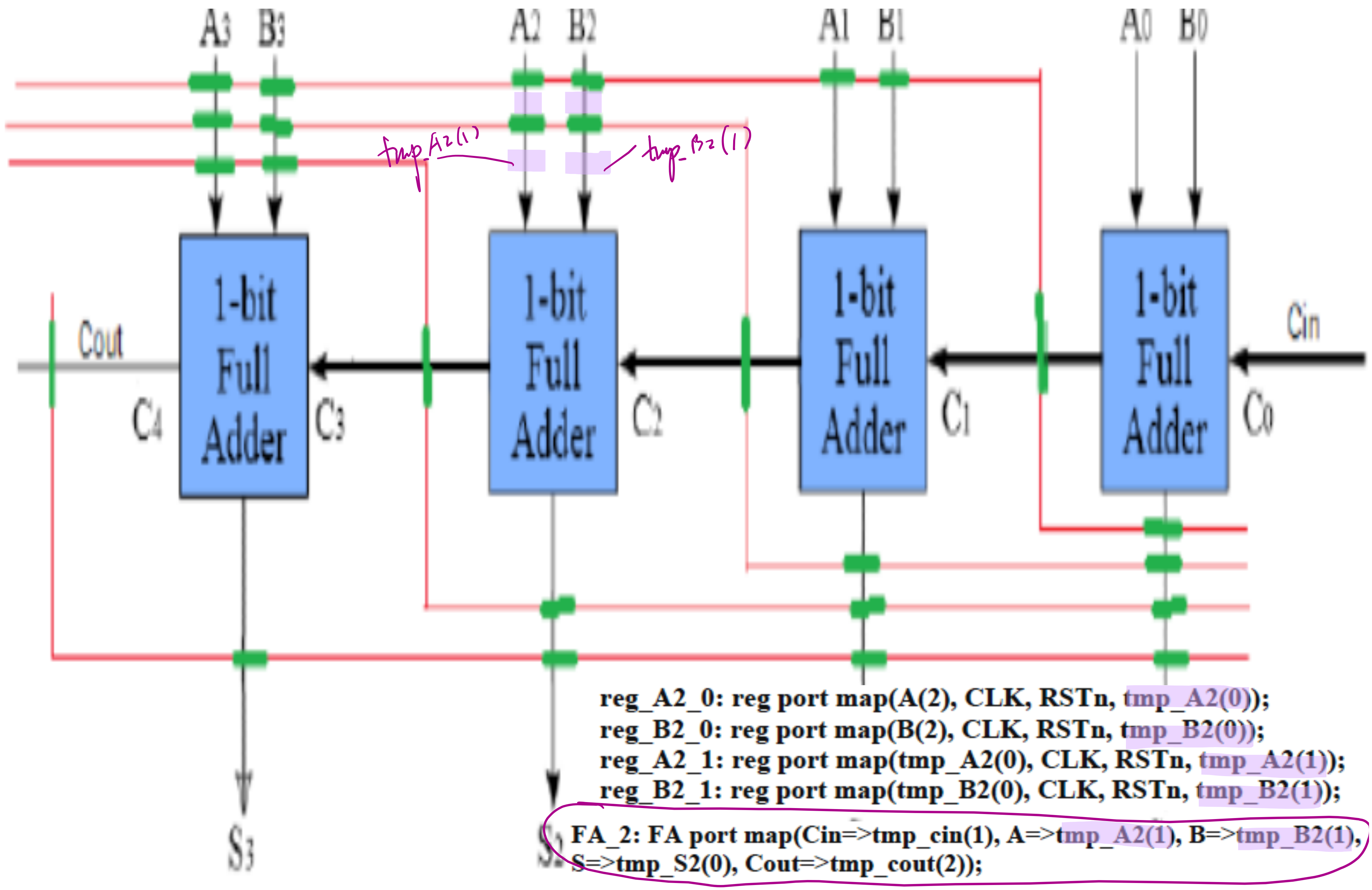
```
reg_s3_1: reg port map(tmp_S3, CLK, RSTn, S(3));
```

```
signal tmp_S3 : std_logic;
```



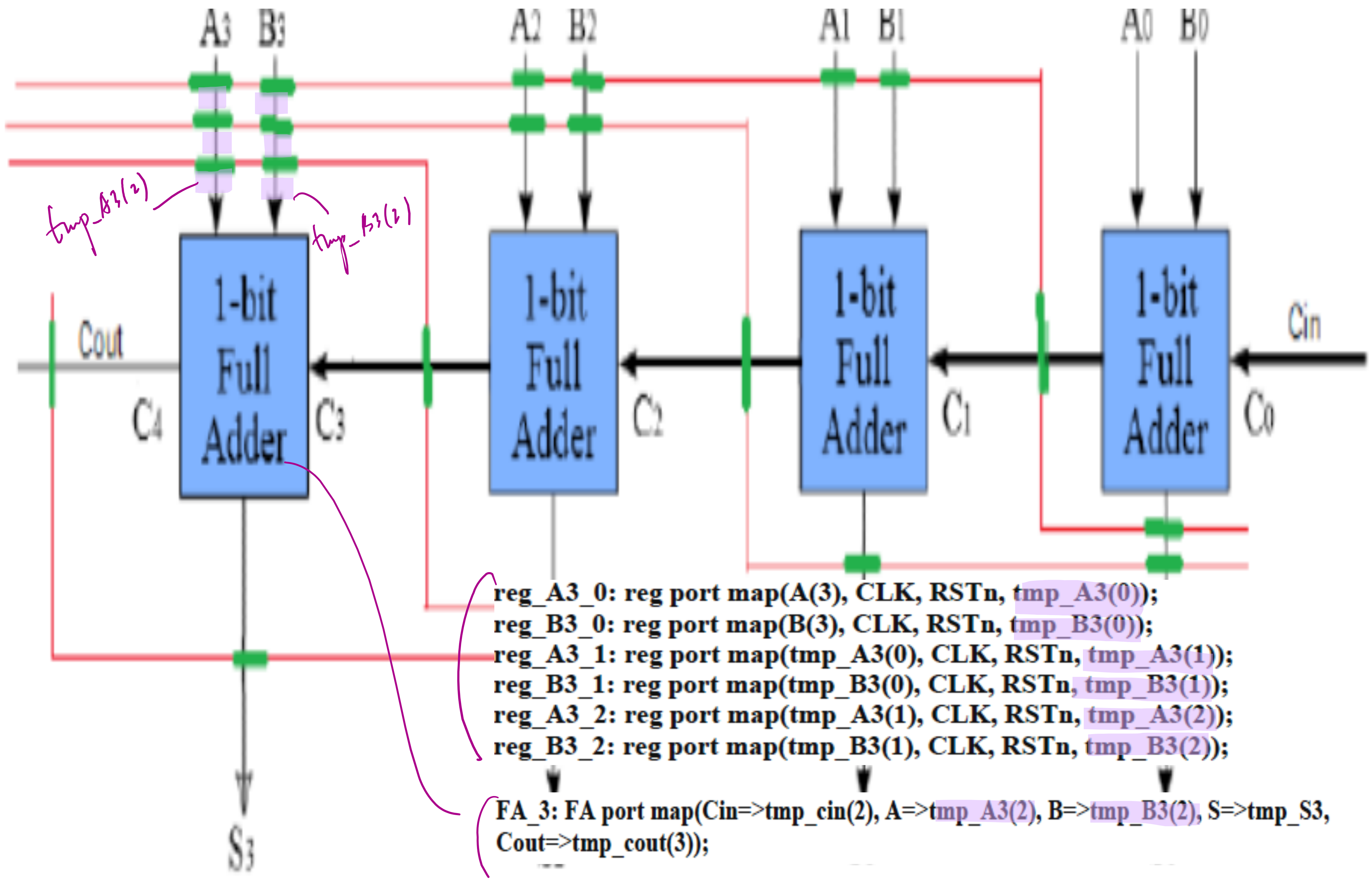
```
reg_A1: reg port map(A(1), CLK, RSTn, tmp_A1);
reg_B1: reg port map(B(1), CLK, RSTn, tmp_B1);
FA_1: FA port map(Cin=>tmp_cin(0), A=>tmp_A1, B=>tmp_B1, S=>tmp_S1(0),
Cout=>tmp_cout(1));
```

```
signal tmp_A1 : std_logic;
signal tmp_B1 : std_logic;
```



```
reg_A2_0: reg port map(A(2), CLK, RSTn, tmp_A2(0));
reg_B2_0: reg port map(B(2), CLK, RSTn, tmp_B2(0));
reg_A2_1: reg port map(tmp_A2(0), CLK, RSTn, tmp_A2(1));
reg_B2_1: reg port map(tmp_B2(0), CLK, RSTn, tmp_B2(1));
FA_2: FA port map(Cin=>tmp_cin(1), A=>tmp_A2(1), B=>tmp_B2(1),
S=>tmp_S2(0), Cout=>tmp_cout(2));
```

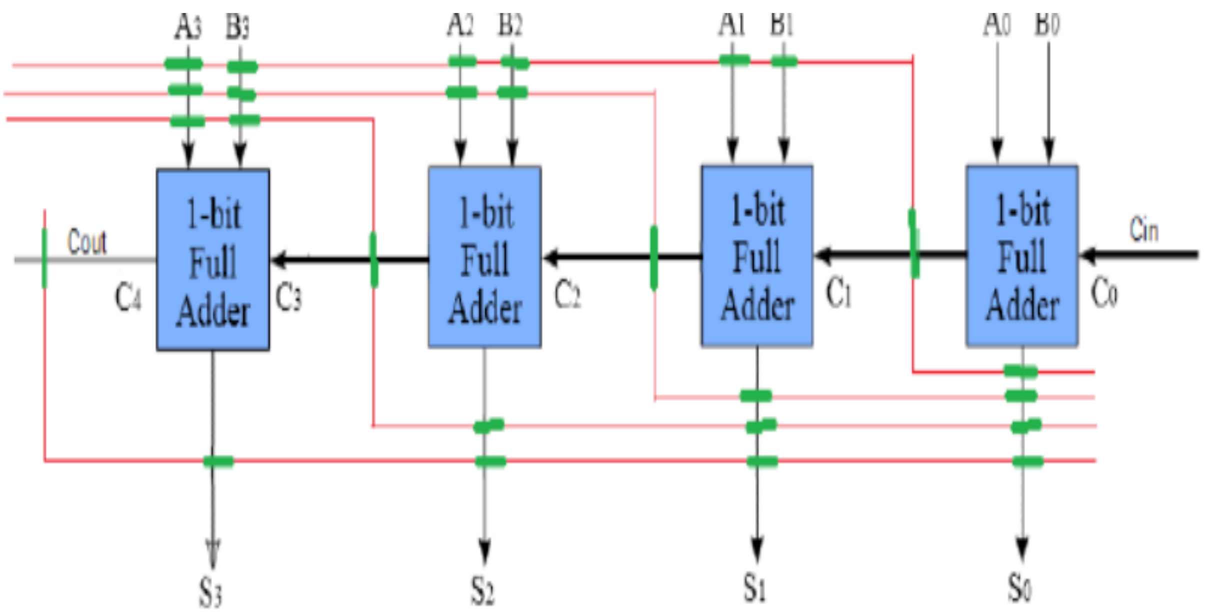
```
signal tmp_A2 : std_logic_vector(1 downto 0);
signal tmp_B2 : std_logic_vector(1 downto 0);
```



```
reg_A3_0: reg port map(A(3), CLK, RSTn, tmp_A3(0));
reg_B3_0: reg port map(B(3), CLK, RSTn, tmp_B3(0));
reg_A3_1: reg port map(tmp_A3(0), CLK, RSTn, tmp_A3(1));
reg_B3_1: reg port map(tmp_B3(0), CLK, RSTn, tmp_B3(1));
reg_A3_2: reg port map(tmp_A3(1), CLK, RSTn, tmp_A3(2));
reg_B3_2: reg port map(tmp_B3(1), CLK, RSTn, tmp_B3(2));

FA_3: FA port map(Cin=>tmp_cin(2), A=>tmp_A3(2), B=>tmp_B3(2), S=>tmp_S3,
Cout=>tmp_cout(3));
```

```
signal tmp_A3 : std_logic_vector(2 downto 0);
signal tmp_B3 : std_logic_vector(2 downto 0);
```



```
architecture behaviour of reg is
begin
   process(CLK, RSTn)
   begin
      if RSTn = '0' then
         Q <= '0';
      elsif rising_edge(clk) then
         Q <= D;
      end if;
   end process;
end architecture;
```

```
ARCHITECTURE behaviour OF FA is
begin
   S <= A XOR B XOR Cin ;
   Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;
end architecture;
```

```
reg_s0_0: reg port map(tmp_S0(0), CLK, RSTn, tmp_S0(1));
reg_s0_1: reg port map(tmp_S0(1), CLK, RSTn, tmp_S0(2));
reg_s0_2: reg port map(tmp_S0(2), CLK, RSTn, tmp_S0(3));
reg_s0_3: reg port map(tmp_S0(3), CLK, RSTn, S(0));
reg_s1_0: reg port map(tmp_S1(0), CLK, RSTn, tmp_S1(1));
reg_s1_1: reg port map(tmp_S1(1), CLK, RSTn, tmp_S1(2));
reg_s1_2: reg port map(tmp_S1(2), CLK, RSTn, S(1));
reg_s2_0: reg port map(tmp_S2(0), CLK, RSTn, tmp_S2(1));
reg_s2_1: reg port map(tmp_S2(1), CLK, RSTn, S(2));
reg_s3_1: reg port map(tmp_S3, CLK, RSTn, S(3));
reg_cout0: reg port map(tmp_cout(0), CLK, RSTn, tmp_cin(0));
reg_cout1: reg port map(tmp_cout(1), CLK, RSTn, tmp_cin(1));
reg_cout2: reg port map(tmp_cout(2), CLK, RSTn, tmp_cin(2));
reg_cout3: reg port map(tmp_cout(3), CLK, RSTn, Cout);
reg_A1: reg port map(A(1), CLK, RSTn, tmp_A1);
reg_B1: reg port map(B(1), CLK, RSTn, tmp_B1);
reg_A2_0: reg port map(A(2), CLK, RSTn, tmp_A2(0));
reg_B2_0: reg port map(B(2), CLK, RSTn, tmp_B2(0));
reg_A2_1: reg port map(tmp_A2(0), CLK, RSTn, tmp_A2(1));
reg_B2_1: reg port map(tmp_B2(0), CLK, RSTn, tmp_B2(1));
reg_A3_0: reg port map(A(3), CLK, RSTn, tmp_A3(0));
reg_B3_0: reg port map(B(3), CLK, RSTn, tmp_B3(0));
reg_A3_1: reg port map(tmp_A3(0), CLK, RSTn, tmp_A3(1));
reg_B3_1: reg port map(tmp_B3(0), CLK, RSTn, tmp_B3(1));
reg_A3_2: reg port map(tmp_A3(1), CLK, RSTn, tmp_A3(2));
reg_B3_2: reg port map(tmp_B3(1), CLK, RSTn, tmp_B3(2));
```

```
-- Component declarations
component FA PORT (Cin, A, B: IN STD_LOGIC ;  Cout, S: OUT STD_LOGIC ) ;
end component;

component reg port(D, CLK, RSTn: in std_logic;  Q: out std_logic);
end component;
```

```
FA_0: FA port map(Cin=>Cin, A=>A(0), B=>B(0), S=>tmp_S0(0),
Cout=>tmp_cout(0));
FA_1: FA port map(Cin=>tmp_cin(0), A=>tmp_A1, B=>tmp_B1, S=>tmp_S1(0),
Cout=>tmp_cout(1));
FA_2: FA port map(Cin=>tmp_cin(1), A=>tmp_A2(1), B=>tmp_B2(1),
S=>tmp_S2(0), Cout=>tmp_cout(2));
FA_3: FA port map(Cin=>tmp_cin(2), A=>tmp_A3(2), B=>tmp_B3(2), S=>tmp_S3,
Cout=>tmp_cout(3));
```