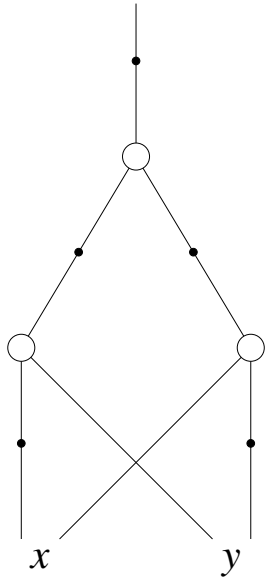
And-Inverter-Graph (AIG)

- only AND and NEGATION operators
- compact gate-level data-structure (bitstuffing)
- pioneered by IBM (equivalence checking)
  [KühlmannGanaiParuthi-DAC'01]
- simplifies algorithms (synthesis & verification)
- ABC from Berkeley uses AIGs heavily
  - FPGA synthesis
  - DAG aware rewriting
    [MishchenkoChatterjeeBrayton-DAC'06]
- standardized file format AIGER
  [Biere'07] [BiereWieringaHeljanko'11]
  - structural SAT tracks
  - Hardware Model Checking Competition
  - variant in synthesis competition

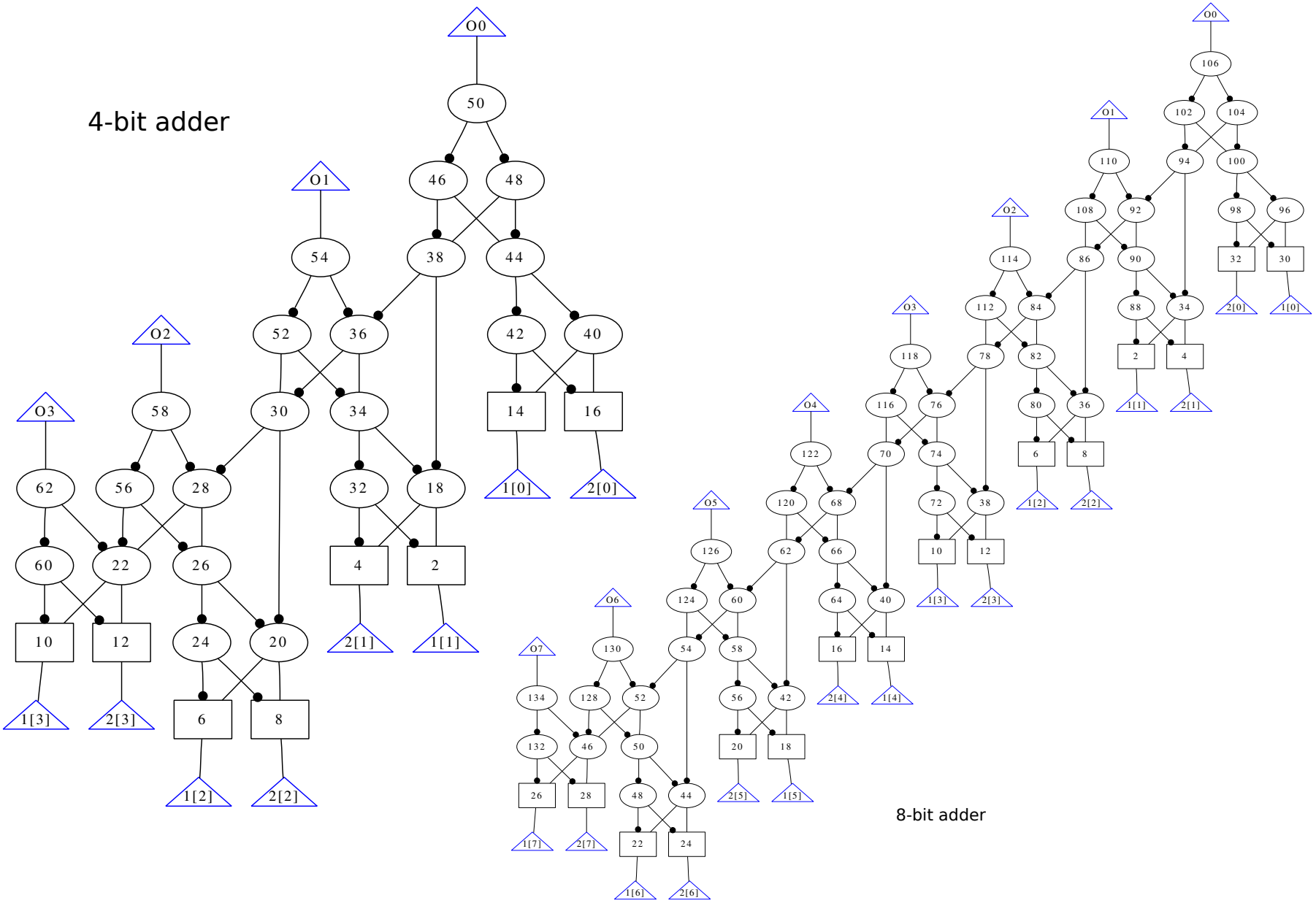
XOR as AIG

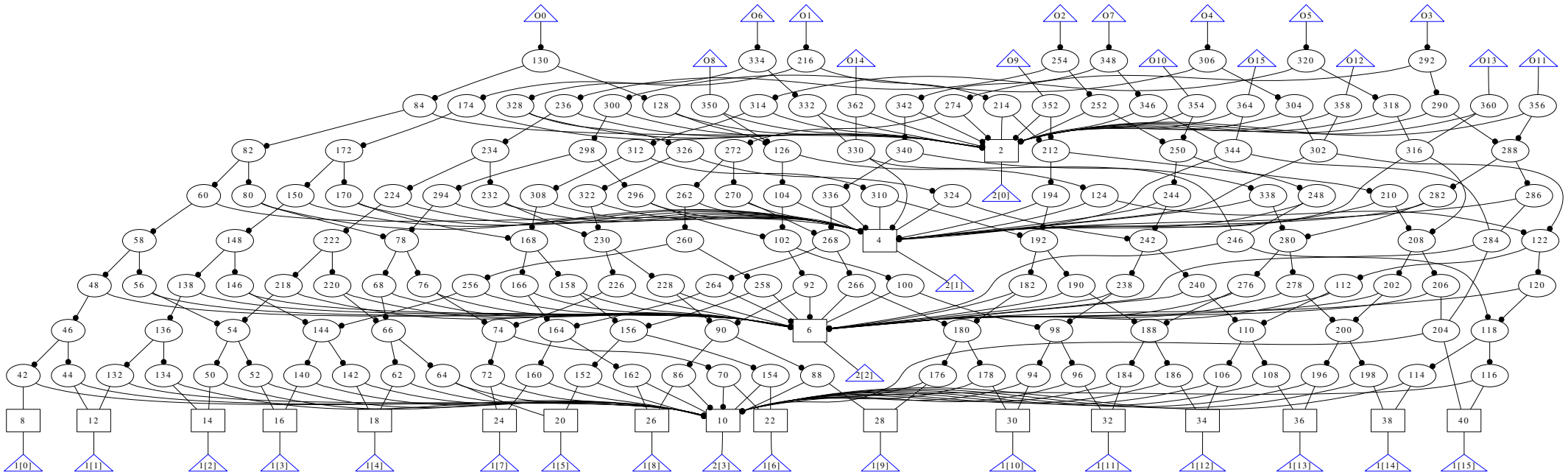


negation/sign are edge attributes

not part of node

$$x \oplus y \equiv (\bar{x} \wedge y) \vee (x \wedge \bar{y}) \equiv \overline{\overline{(\bar{x} \wedge y)} \wedge \overline{(x \wedge \bar{y})}}$$


4-bit adder
O0
50
O1
46
48
54
38
44
O2
52
36
42
40
O3
58
30
34
14
16
62
56
28
32
18
1[0]
2[0]
60
22
26
4
2
10
12
24
20
2[1]
1[1]
1[3]
2[3]
6
8
1[2]
2[2]
8-bit adder
O0
106
O1
102
104
110
94
100
O2
108
92
98
96
114
86
90
32
30
O3
112
84
88
34
2[0]
1[0]
118
78
82
2
4
O4
116
76
80
36
1[1]
2[1]
122
70
74
6
8
O5
120
68
72
38
1[2]
2[2]
126
62
66
10
12
O6
124
60
64
40
1[3]
2[3]
O7
130
54
58
16
14
134
128
52
56
42
2[4]
1[4]
132
46
50
20
18
26
28
48
44
2[5]
1[5]
1[7]
2[7]
22
24
1[6]
2[6]



bit-vector of length 16 shifted by bit-vector of length 4

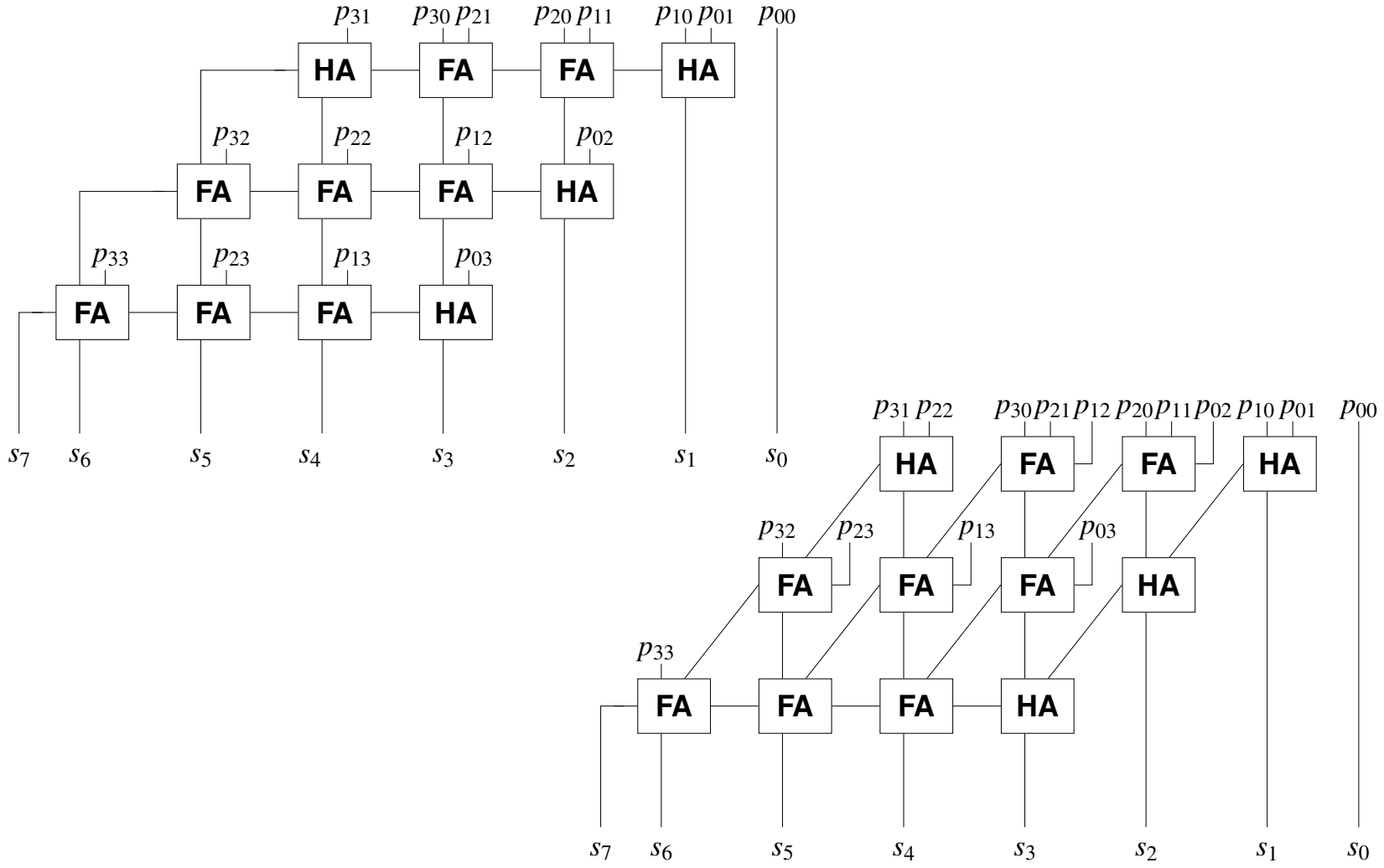
“barrel shifter”

Binary Multiplication

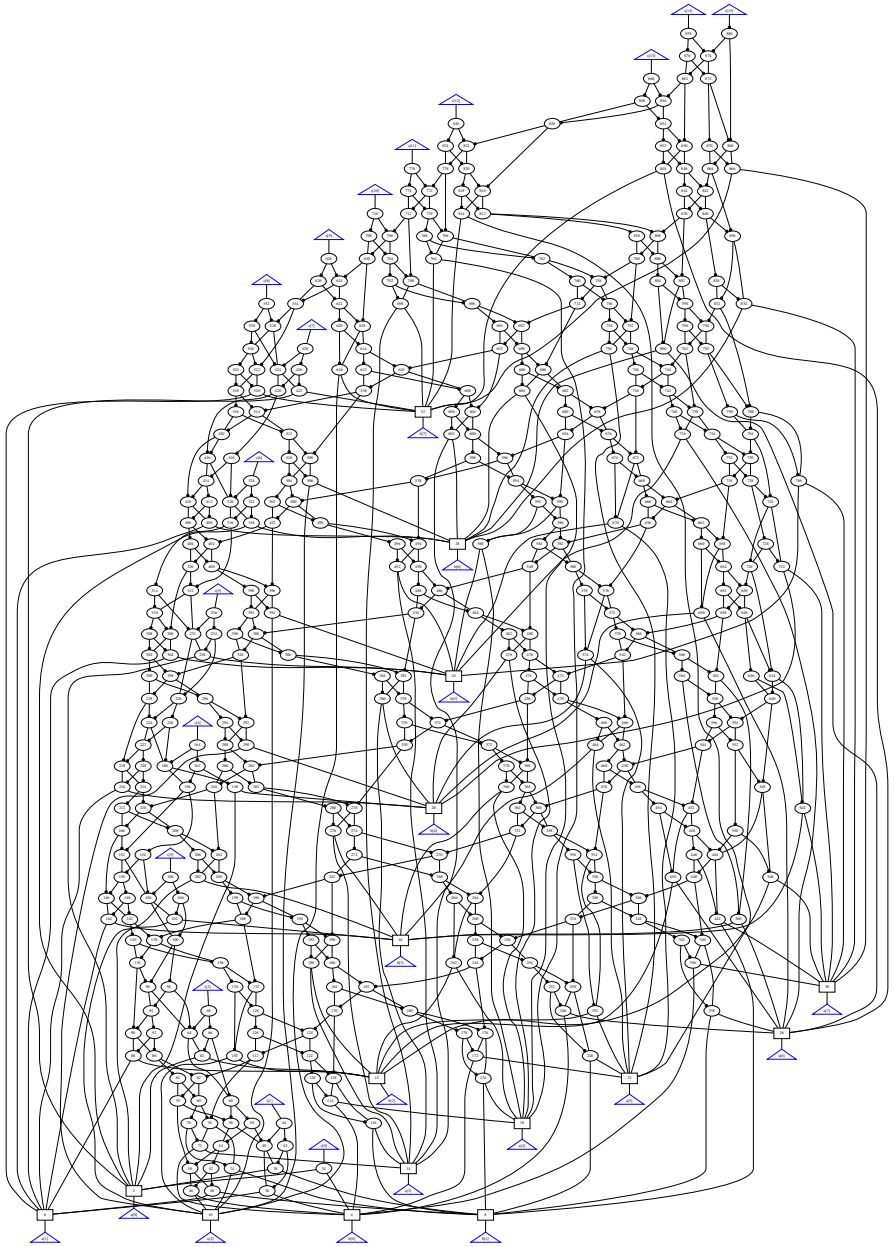
$$
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & \cdot & 1 & 1 & 0 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 & \\
 & & & & 1 & 1 & 0 & 1 & & \\
 & & & 1 & 1 & 0 & 1 & & & \\
 & & 1_{2} & 1_{2} & 0_{2} & 1_{1} & 0 & 0 & & \\
\hline
 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 &
\end{array}
$$

$15 \cdot 13 = 195$

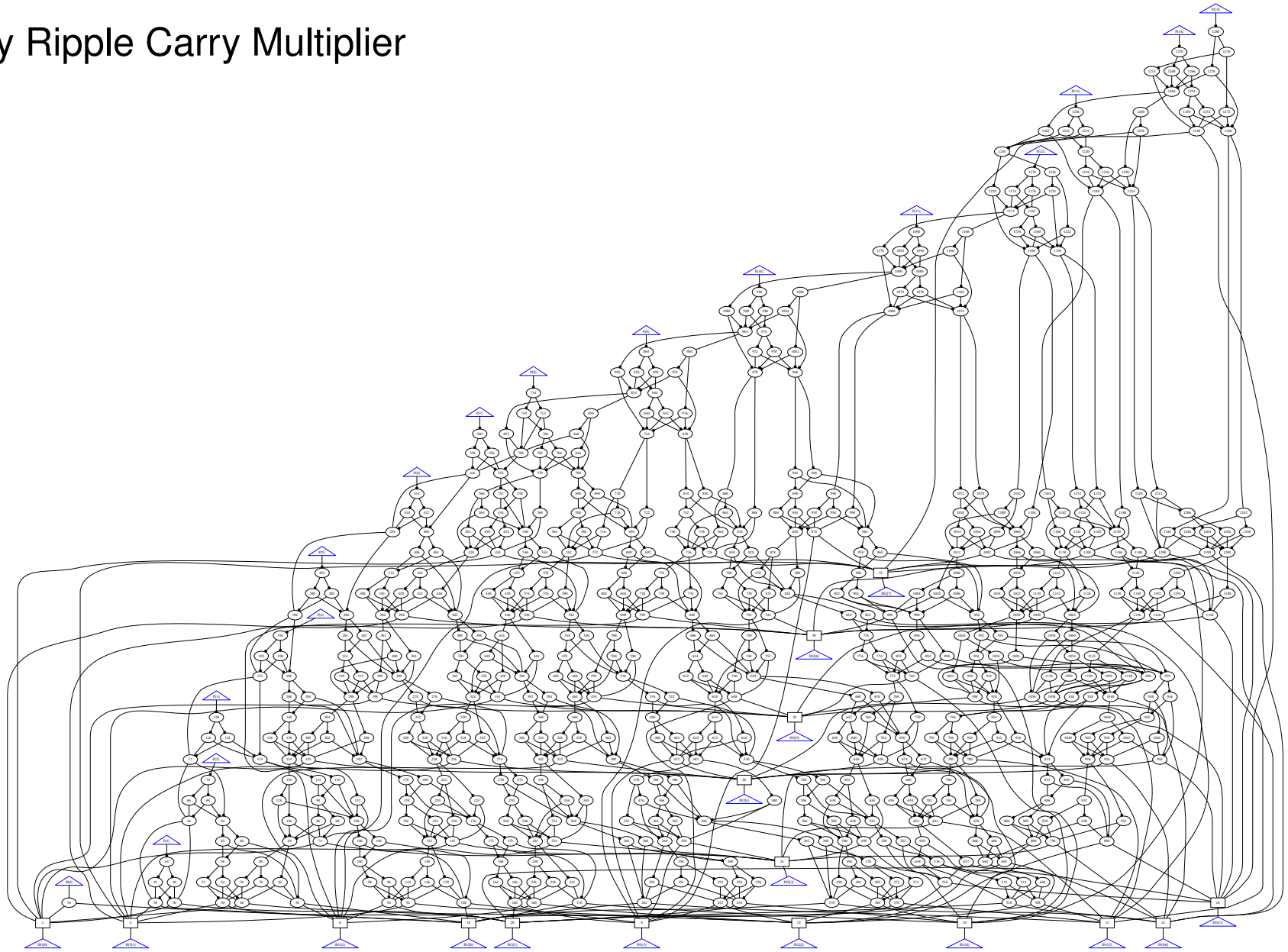
Multipliers



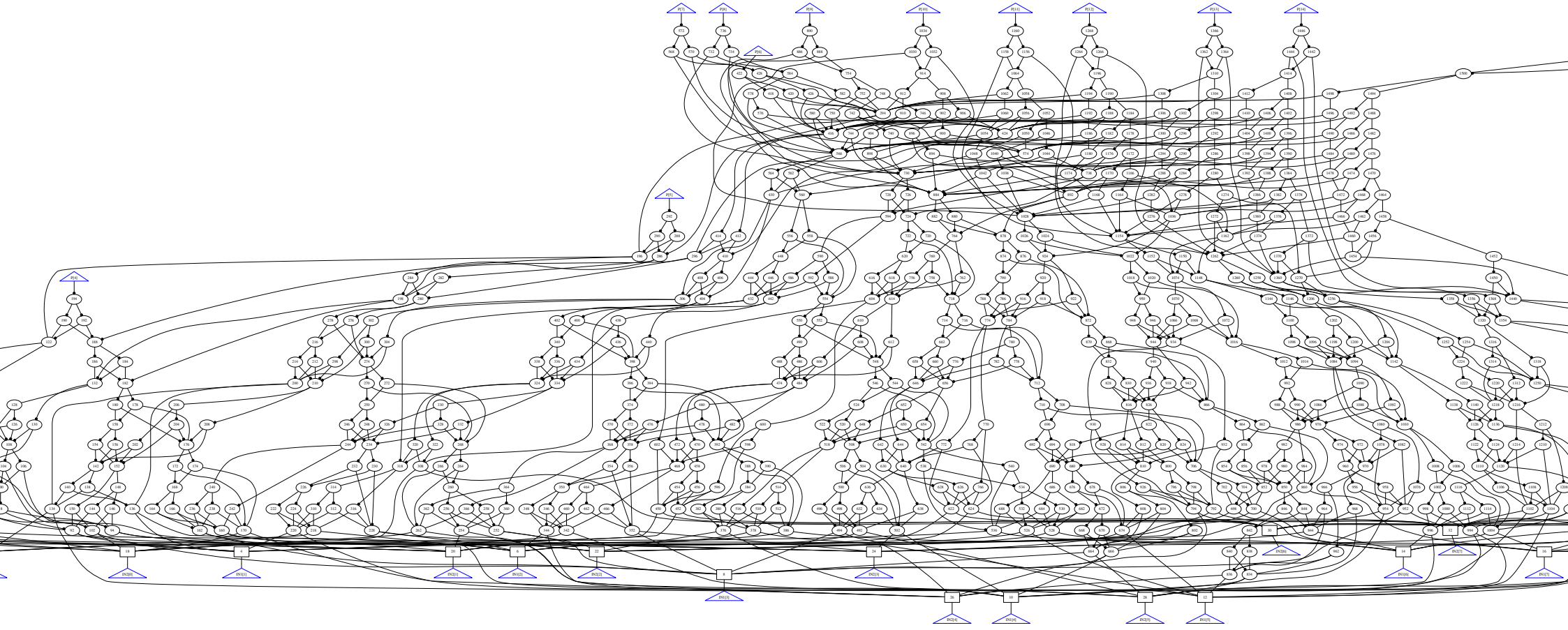
8-Bit Multiplier of Boolector



8-Bit Array Ripple Carry Multiplier



8-Bit Wallace-Tree Carry-Lookahead Multiplier



Fast Propagate and Generate Adders

$$
\begin{aligned}
p &= x \vee y && \text{propagate} \\
g &= x \wedge y && \text{generate} \\
c_{out} &= g \vee p \wedge c_{in} && \text{carry}
\end{aligned}
$$

$$
\begin{aligned}
p_i &= x_i \vee y_i && \text{propagate} \\
g_i &= x_i \wedge y_i && \text{generate} \\
c_{i+1} &= g_i \vee p_i \wedge c_i && \text{carry}
\end{aligned}
$$

$$
\begin{aligned}
c_0 &= 0 \\
c_1 &= g_0 \\
c_2 &= g_1 \vee p_1 \wedge g_0 \\
c_3 &= g_2 \vee p_2 \wedge g_1 \vee p_2 \wedge p_1 \wedge g_0 \\
c_4 &= g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0
\end{aligned}
$$

Now apply parallel prefix scan algorithm from parallel computing!