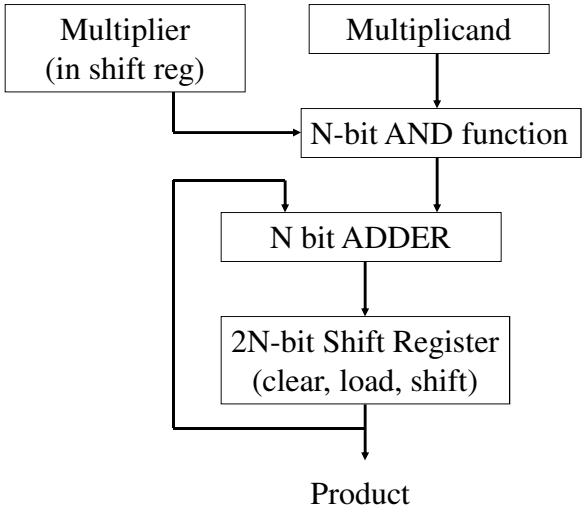
Working Up a High Speed Multiplier

Multiplication Algorithms

- Gradeschool Algorithm
- Modified Gradeschool Algorithm – A
- Modified Gradeschool Algorithm – B
- Booth's Algorithm
- High Speed Multiply Algorithm

Multiply: Gradeschool Algorithm

```
            10111001   (185)
          x 11010111   (215)
          ----------
            10111001
           10111001
          10111001
         00000000
        10111001
       00000000
      10111001
     10111001
    ----------------
    1001101101011111 (39775)
```



Start Condition

```
          10111001
        x 11010111
        ----------
          10111001
         10111001
        10111001
       00000000
      10111001
     00000000
    10111001
   10111001
----------------
0000000000000000
```

```
          10111001
        x 11010111
        ----------
          10111001



----------------
0000000000000000
```

```
  10111001
x 11010111
----------
```

```
  10111001
----------------
0000000000000000
```

```
  10111001
x 11010111
----------
```

```
  10111001
----------------
0000000010111001
```

```
  10111001
x 11010111
----------
```

```
  10111001
----------------
0000000010111001
```

```
  10111001
x 11010111
----------
```

```
  10111001
----------------
0000001000101011
```

```
  10111001
x 11010111
----------
```

```
      10111001
----------------
0000001000101011
```

```
  10111001
x 11010111
----------
```

```
      10111001
----------------
0000010100001111
```

```
  10111001
x 11010111
----------

     00000000
----------------
0000010100001111
```

```
  10111001
x 11010111
----------

    10111001
----------------
0000010100001111
```

```
  10111001
x 11010111
----------

   10111001
---------------
0001000010011111
```

```
  10111001
x 11010111
----------

   00000000
---------------
0001000010011111
```

```
  10111001
x 11010111
----------
```

```
  10111001
----------------
0011111011011111
```


```
  10111001
x 11010111
----------
```

```
  10111001
----------------
0011111011011111
```

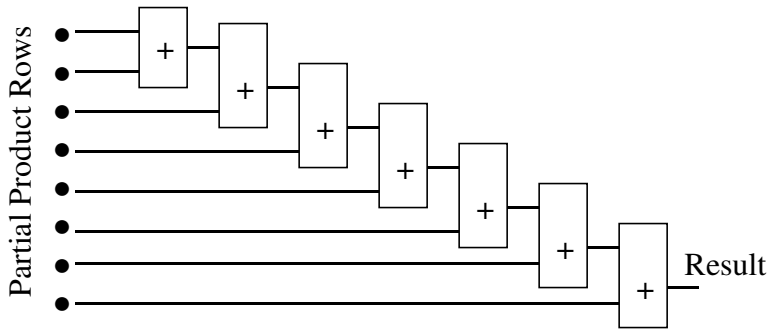
```
  10111001
x 11010111
----------
```

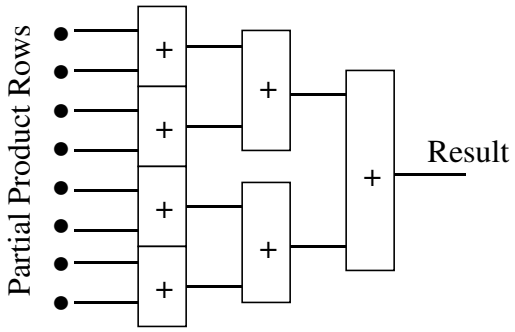
```
 10111001
----------------
0011111011011111
```

```
  10111001
x 11010111
----------
```

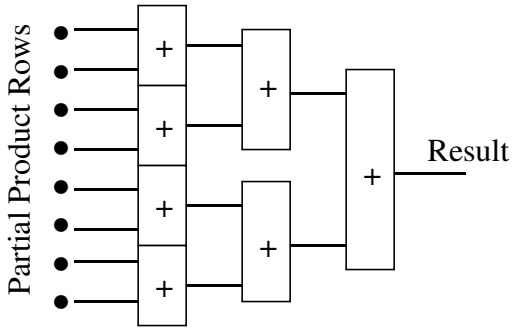
```
 10111001
----------------
1001101101011111
```

```
          10111001
        x 11010111
        ----------
          10111001→●
         10111001-→●
        10111001--→●
       00000000---→●
      10111001----→●
     00000000-----→●
    10111001------→●
   10111001-------→●
----------------
1001101101011111
```

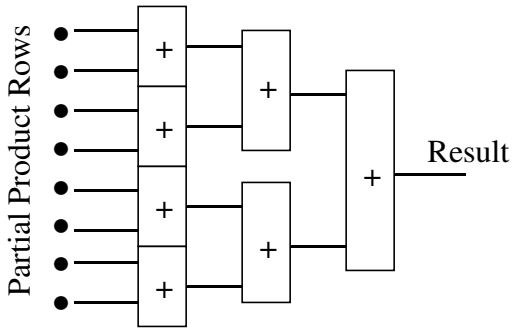




Let's do this one in VHDL

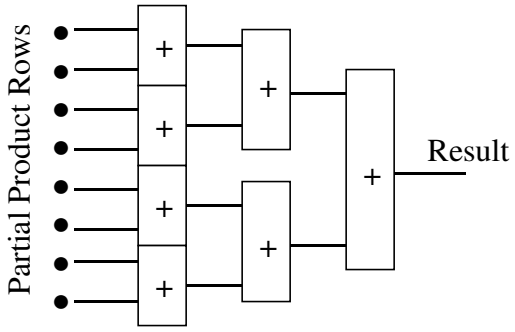


Let's do this one in VHDL



Step 1: Figure out data flow

Let's do this one in VHDL

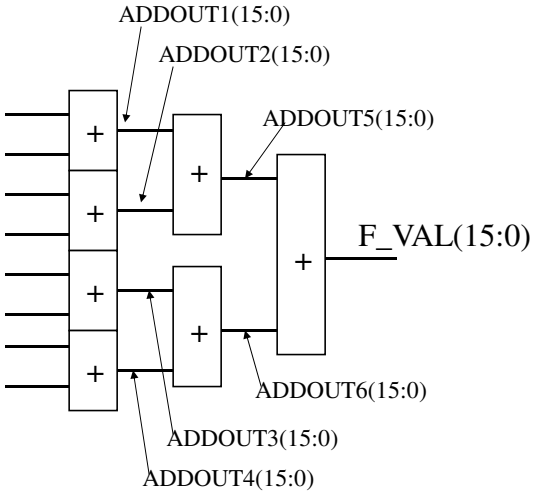


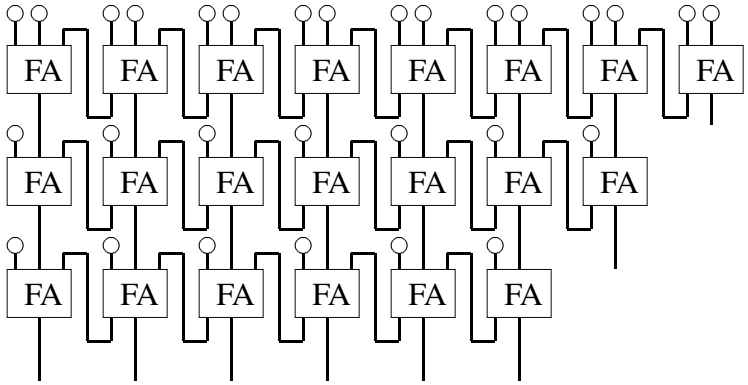
Step 1: Figure out data flow
Step 2: Name stuff….

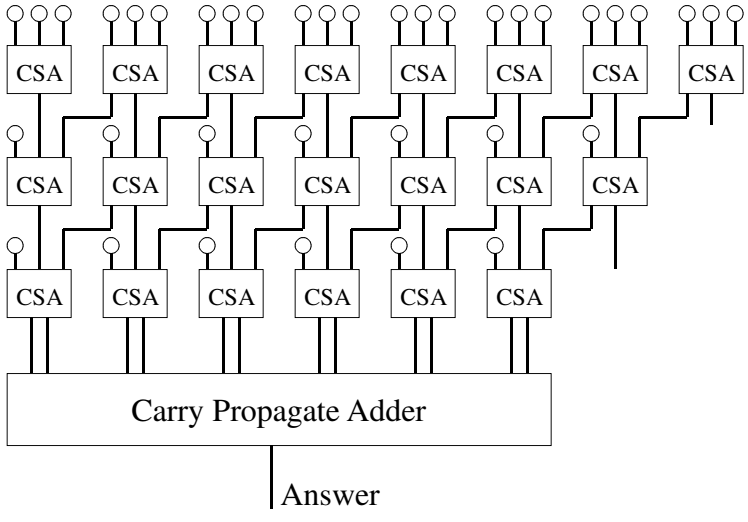
Start with the Partial Product Array

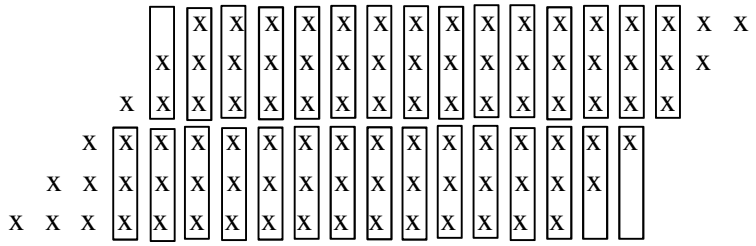
PPA0(15:0)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
PPA1(15:0)	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
PPA2(15:0)	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
PPA3(15:0)	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
PPA4(15:0)	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
PPA5(15:0)	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
PPA6(15:0)	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
PPA7(15:0)	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

More naming – the adder outputs

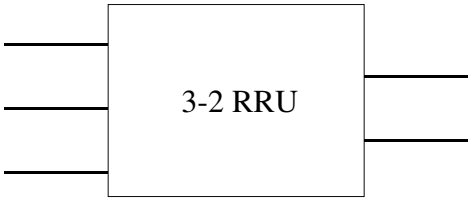



FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA
FA


CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
CSA
Carry Propagate Adder
Answer

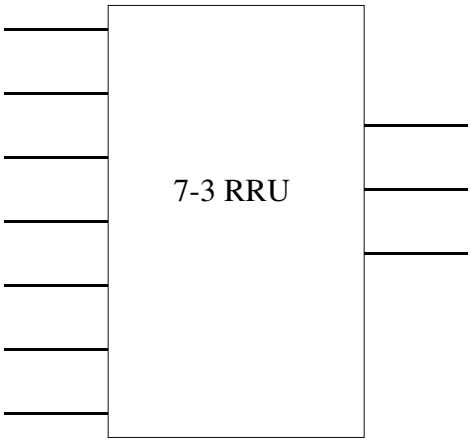


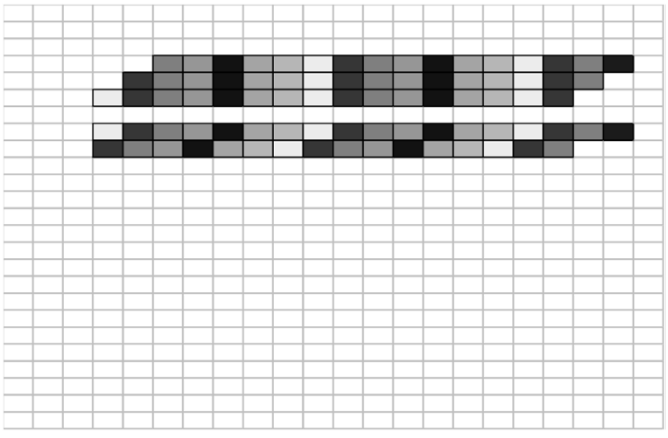
Carry-Save Adder:
Minimal Row Reduction Unit (RRU)
Input: 3 rows
Output: 2 rows

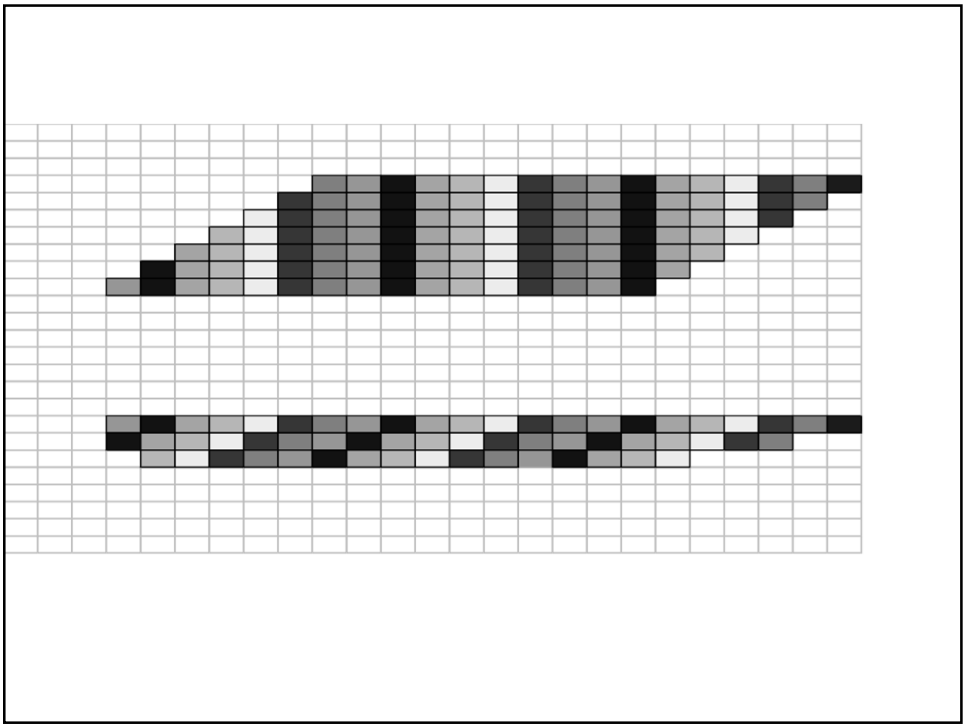


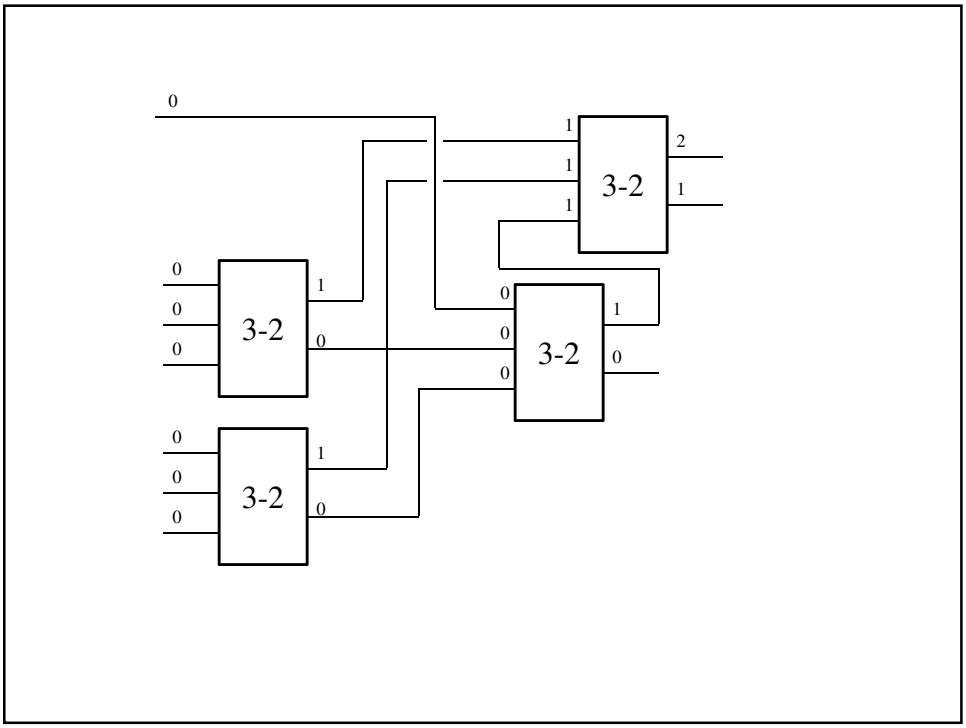
Note: Care must be taken to make sure that
the significance of the bits is handled properly

7-3 Row Reduction Unit
Input: 7 rows
Output: 3 rows








0
1
1
1
3-2
2
1
0
0
0
3-2
1
0
0
0
3-2
1
0
0
0
3-2
1
0
0
0
3-2
1
0

Row Reduction: any combination of 2^N-1 rows → N rows

2^N-1		N
3	→	2
7	→	3
15	→	4
31	→	5

Row Reduction System – 24 Bits