Multiplier Design Comparative Analysis (Pipelined v. Combinational)

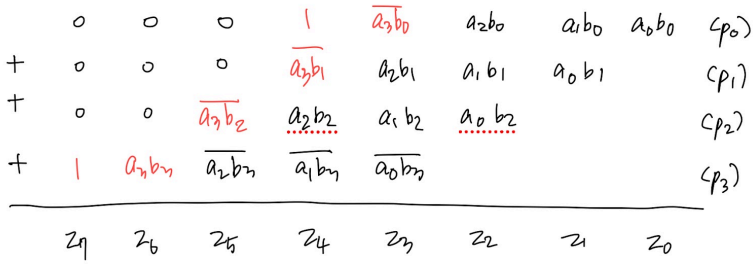
Summer 2025

Designing a systolic array and TinyNPU taught me the importance of multiply-accumulate (MAC) processing elements in machine learning hardware. Thus, I designed a pipelined multiplier for signed INT8 using the Baugh-Wooley algorithm and compared it to a combinational ($*$) multiplier in terms of power, performance, area, and other metrics.

Background

Baugh-Wooley Multiplication

The Baugh-Wooley method is useful for multiplying signed (i.e., two's complement) integers. Below is an example of multiplying two signed 4-bit integers.



Essentially, the Baugh-Wooley method allows multiplying two signed integers without requiring excess hardware for sign-extended partial products. This is made possible through several steps of boolean algebra that simplifies to (1) inverting the MSB of each line of partial products, (2) adding constants to the first and last line of partial products while inverting the latter, and (3) accumulating all the partial products.

INT8

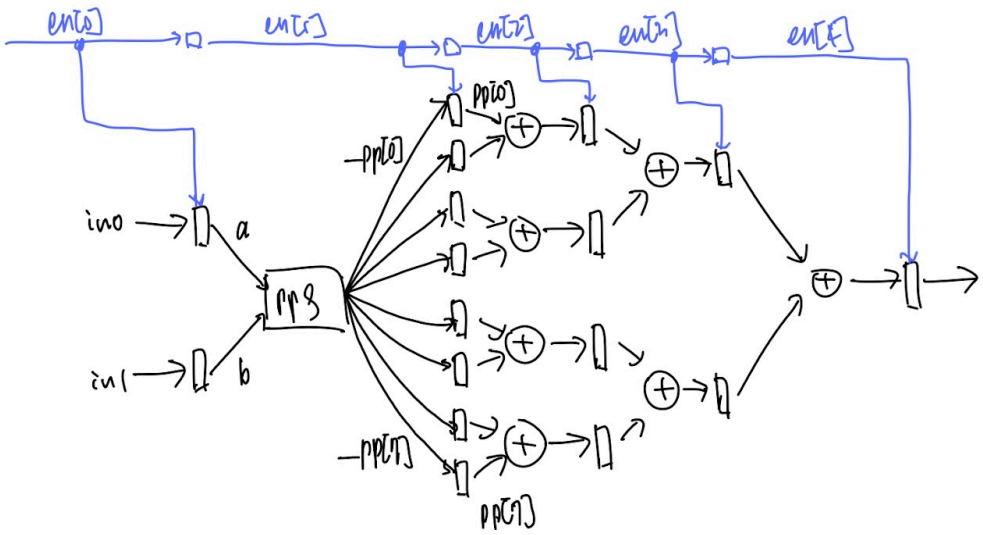
INT8 stands for 8-bit signed integers that are often used for quantizing deep learning model parameters. INT8 is often used for inference computations because it is both time- and space-efficient than large bitwidths like 32-bit floating point numbers (FP32). Thus, it is critical to have efficient hardware designs for multiplying INT8 numbers involved in MAC operations.

Note that multiplying two INT8 numbers yield an INT16 number. When performing intermediate MAC operations, 32-bit accumulators are often used to add those INT16 numbers to preserve precision until saturating to INT8 as needed.

RTL

Pipelined Multiplier

The pipelined multiplier for signed INT8 has five stages in the following order: input stage, partial product generation (PPG) stage, and three accumulation reduction tree stages. Below is a diagram of the pipelined multiplier.



Note that all logic between each pipeline stage is combinational (i.e., no stalling). Also, the pipelined multiplier has an enable signal going through the pipeline along with the two inputs. This assumes that the pipelined multiplier is a building block included in a larger system with an overarching control logic for correctly enabling the multiplier as needed.

Combinational Multiplier

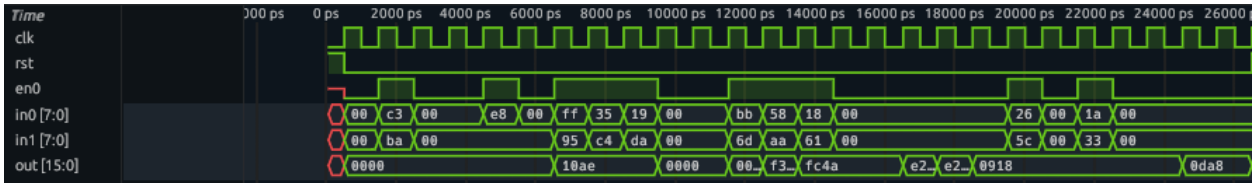
The combinational multiplier simply uses the `*` operator triggered by a rising clock edge.

Design Verification

The **pipelined multiplier** was simulated using Cocotb (Synopsys VCS) and the following test cases. Note that various number distributions (e.g., full domain, full positive domain, full negative domain, small/large) positive/negative domain) were used 10,000 times each for generating INT8 input pairs.

- Simple directed multiplications (e.g., alternating `0x0=0` and `1x1=1` transactions)
- Directed input stream with constant enable and random INT8 input pairs
- Random input stream with random enable and random INT8 input pairs

The waveform shows a random input stream test case.



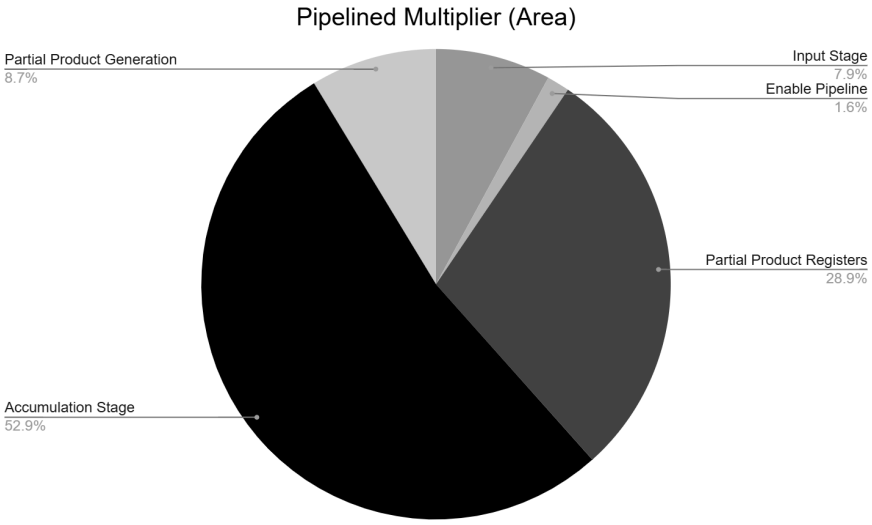
Physical Design

Technical details beyond what is provided below cannot be shared under the non-disclosure agreement between Cornell Custom Silicon Systems, MPW (Multi-Project-Wafer) University Service (MUSE), and Taiwan Semiconductor Manufacturing Company (TSMC).

The MUSE TSMC 180 nm process node was used to collect power, performance, and area metrics for the pipelined and combinational multipliers.

Area

The **pipelined multiplier** requires approximately 16,590 μm^2 of space where much of this area is allocated for the accumulation reduction tree and the partial product generator and registers.



The **combinational multiplier** requires approximately 6,900 μm^2 of space.

Performance

The **pipelined multiplier** can operate at a clock period of 3 ns (333 MHz) where the critical path goes through the combinational adder of the last accumulation stage with a slack of 0.394 ns.

```
Path 1: MET Setup Check with Pin g_s2_reg_0__acc_reg/q_reg_15_/CP
Endpoint:   g_s2_reg_0__acc_reg/q_reg_15_/D (v) checked with  leading edge of
'Multiplier_INT8_clk'
```

```
Beginpoint: g_s1_reg_1__acc_reg/q_reg_4_/Q  (v) triggered by  leading edge of
'Multiplier_INT8_clk'
Path Groups: {Multiplier_INT8_clk}
Analysis View: analysis_typical
Other End Arrival Time          0.000
- Setup                         0.054
+ Phase Shift                   3.000
+ CPPR Adjustment               0.000
= Required Time                 2.946
- Arrival Time                  2.551
= Slack Time                    0.394
     Clock Rise Edge                 0.000
     + Clock Network Latency (Ideal) 0.000
     = Beginpoint Arrival Time       0.000
     +-----------------------------------------------------------------------------------------------+
     |              Instance               |     Arc      |     Cell     | Delay | Arrival | Required |
     |                                     |              |              |       |  Time   |   Time   |
     |-------------------------------------+--------------+--------------+-------+---------+----------|
     | g_s1_reg_1__acc_reg/q_reg_4_        | CP ^         |              |       |   0.000 |    0.394 |
     | g_s1_reg_1__acc_reg/q_reg_4_        | CP ^ -> Q v  | xxxxxxxxxx   | 0.270 |   0.270 |    0.664 |
     | g_s2_adder_0__acc_adder/U5          | A2 v -> Z v  | xxxxxxxxxx   | 0.120 |   0.390 |    0.784 |
     | g_s2_adder_0__acc_adder/intadd_1_U9 | CI v -> CO v | xxxxxxxxxx   | 0.192 |   0.582 |    0.976 |
     | g_s2_adder_0__acc_adder/intadd_1_U8 | CI v -> CO v | xxxxxxxxxx   | 0.200 |   0.782 |    1.176 |
     | g_s2_adder_0__acc_adder/intadd_1_U7 | CI v -> CO v | xxxxxxxxxx   | 0.211 |   0.993 |    1.388 |
     | g_s2_adder_0__acc_adder/intadd_1_U6 | CI v -> CO v | xxxxxxxxxx   | 0.198 |   1.192 |    1.586 |
     | g_s2_adder_0__acc_adder/intadd_1_U5 | CI v -> CO v | xxxxxxxxxx   | 0.193 |   1.385 |    1.779 |
     | g_s2_adder_0__acc_adder/intadd_1_U4 | CI v -> CO v | xxxxxxxxxx   | 0.192 |   1.577 |    1.971 |
     | g_s2_adder_0__acc_adder/intadd_1_U3 | CI v -> CO v | xxxxxxxxxx   | 0.213 |   1.790 |    2.184 |
     | g_s2_adder_0__acc_adder/intadd_1_U2 | CI v -> CO v | xxxxxxxxxx   | 0.175 |   1.964 |    2.359 |
     | g_s2_adder_0__acc_adder/U2          | A1 v -> Z v  | xxxxxxxxxx   | 0.117 |   2.081 |    2.476 |
     | g_s2_adder_0__acc_adder/U6          | A2 v -> ZN ^ | xxxxxxxxxx   | 0.063 |   2.144 |    2.539 |
     | g_s2_adder_0__acc_adder/U7          | A2 ^ -> ZN v | xxxxxxxxxx   | 0.186 |   2.330 |    2.724 |
     | g_s2_reg_0__acc_reg/U19             | B2 v -> Z v  | xxxxxxxxxx   | 0.221 |   2.551 |    2.945 |
     | g_s2_reg_0__acc_reg/q_reg_15_       | D v          | xxxxxxxxxx   | 0.000 |   2.551 |    2.946 |
     +-----------------------------------------------------------------------------------------------+
```

The **combinational multiplier** can operate at a clock period of 4 ns (250 MHz) with a slack of 0.214 ns.

```
Path 1: MET Setup Check with Pin out_reg_14_/CP
Endpoint:   out_reg_14_/D (^) checked with  leading edge of 'StarMultiplier_clk'
Beginpoint: in0[3]        (v) triggered by  leading edge of 'StarMultiplier_clk'
Path Groups: {StarMultiplier_clk}
Analysis View: analysis_typical
Other End Arrival Time          0.000
- Setup                         0.062
+ Phase Shift                   4.000
+ CPPR Adjustment               0.000
= Required Time                 3.938
- Arrival Time                  3.724
= Slack Time                    0.214
     Clock Rise Edge                     0.000
     + Input Delay                       0.005
     = Beginpoint Arrival Time           0.005
     +-------------------------------------------------------------------------+
     |    Instance    |     Arc      |      Cell      | Delay | Arrival | Required |
     |                |              |                |       |  Time   |   Time   |
     |----------------+--------------+----------------+-------+---------+----------|
     |                | in0[3] v     |                |       |   0.005 |    0.219 |
     | FE_DBTC1_in0_3 | I v -> ZN ^  | xxxxxxxxxx     | 0.038 |   0.044 |    0.257 |
     | U60            | A1 ^ -> Z ^  | xxxxxxxxxx     | 0.104 |   0.148 |    0.361 |
     | U42            | I ^ -> ZN v  | xxxxxxxxxx     | 0.036 |   0.184 |    0.397 |
     | U199           | A2 v -> ZN ^ | xxxxxxxxxx     | 0.126 |   0.309 |    0.523 |
     | U116           | B2 ^ -> Z ^  | xxxxxxxxxx     | 0.143 |   0.452 |    0.666 |
     | U175           | I ^ -> ZN v  | xxxxxxxxxx     | 0.040 |   0.492 |    0.706 |
     | U65            | A1 v -> Z v  | xxxxxxxxxx     | 0.121 |   0.613 |    0.827 |
     | U141           | I v -> ZN ^  | xxxxxxxxxx     | 0.041 |   0.655 |    0.868 |
     | U52            | B ^ -> Z ^   | xxxxxxxxxx     | 0.095 |   0.750 |    0.963 |
     | U138           | I ^ -> ZN v  | xxxxxxxxxx     | 0.050 |   0.800 |    1.014 |
```

```
| U56           | A v -> ZN ^  | xxxxxxxxxx     | 0.213 |   1.013 |    1.227 |
| U307          | A ^ -> ZN v  | xxxxxxxxxx     | 0.230 |   1.243 |    1.457 |
| U68           | I v -> ZN ^  | xxxxxxxxxx     | 0.039 |   1.282 |    1.496 |
| U133          | A2 ^ -> ZN v | xxxxxxxxxx     | 0.052 |   1.334 |    1.548 |
| U137          | A2 v -> Z v  | xxxxxxxxxx     | 0.108 |   1.442 |    1.656 |
| U292          | A2 v -> ZN ^ | xxxxxxxxxx     | 0.076 |   1.518 |    1.732 |
| U296          | C ^ -> ZN v  | xxxxxxxxxx     | 0.193 |   1.711 |    1.925 |
| U257          | A v -> ZN ^  | xxxxxxxxxx     | 0.253 |   1.964 |    2.178 |
| intadd_1_U2   | A ^ -> CO ^  | xxxxxxxxxx     | 0.271 |   2.234 |    2.448 |
| U55           | B ^ -> ZN v  | xxxxxxxxxx     | 0.093 |   2.327 |    2.541 |
| intadd_2_U2   | CI v -> CO v | xxxxxxxxxx     | 0.174 |   2.502 |    2.715 |
| U253          | B v -> ZN ^  | xxxxxxxxxx     | 0.278 |   2.780 |    2.993 |
| intadd_0_U2   | A ^ -> CO ^  | xxxxxxxxxx     | 0.268 |   3.048 |    3.262 |
| U153          | I ^ -> ZN v  | xxxxxxxxxx     | 0.034 |   3.082 |    3.296 |
| U300          | C v -> ZN ^  | xxxxxxxxxx     | 0.281 |   3.364 |    3.577 |
| U66           | A3 ^ -> Z v  | xxxxxxxxxx     | 0.220 |   3.583 |    3.797 |
| U54           | A2 v -> ZN ^ | xxxxxxxxxx     | 0.141 |   3.724 |    3.938 |
| out_reg_14_   | D ^          | xxxxxxxxxx     | 0.000 |   3.724 |    3.938 |
+-------------------------------------------------------------------------------+
```

Power

The **pipelined multiplier** consumes 10.3 mW of power.

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:        9.00352961         87.4074%
Total Switching Power:       1.29705423         12.5920%
Total Leakage Power:         0.00006185          0.0006%
Total Power:                10.30064570
-----------------------------------------------------------------------------------------

Group                            Internal   Switching     Leakage       Total  Percentage
                                 Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                          7.141      0.4968   2.811e-05       7.638       74.15
Macro                                   0           0           0           0           0
IO                                      0           0           0           0           0
Combinational                       1.863      0.8002   3.375e-05       2.663       25.85
Clock (Combinational)                   0           0           0           0           0
Clock (Sequential)                      0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                               9.004       1.297   6.185e-05        10.3         100
-----------------------------------------------------------------------------------------

Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                       1.8      9.004       1.297   6.185e-05        10.3         100

Hierarchy                        Internal   Switching     Leakage       Total  Percentage
                                 Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
g_en_reg_1__en_reg                 0.0475     0.02737   1.991e-07     0.07487      0.7269
g_en_reg_2__en_reg                0.04518     0.01815   2.503e-07     0.06333      0.6148
g_en_reg_3__en_reg                0.04273    0.001992   2.459e-07     0.04472      0.4342
g_en_reg_4__en_reg                0.04049   0.0006269   2.437e-07     0.04111      0.3991
a_reg                              0.4045      0.1057   2.449e-06      0.5102       4.953
b_reg                              0.4108     0.08573   2.506e-06      0.4965        4.82
ppg                                0.1694      0.2149   4.662e-06      0.3843       3.731
g_pp_reg_0__pp_reg                 0.4228      0.0433   2.346e-06      0.4661       4.525
g_pp_reg_1__pp_reg                 0.3629     0.02907   1.998e-06       0.392       3.805
g_pp_reg_2__pp_reg                 0.3624     0.03315    1.98e-06      0.3956        3.84
```

g_pp_reg_3__pp_reg	0.3727	0.03879	2.07e-06	0.4115	3.994
g_pp_reg_4__pp_reg	0.3629	0.0295	1.998e-06	0.3924	3.809
g_pp_reg_5__pp_reg	0.3727	0.04086	2.07e-06	0.4135	4.015
g_pp_reg_6__pp_reg	0.3723	0.0361	2.069e-06	0.4084	3.965
g_pp_reg_7__pp_reg	0.4566	0.05111	2.582e-06	0.5077	4.929
g_s0_adder_0__acc_adder	0.08522	0.009178	1.691e-06	0.0944	0.9165
g_s0_adder_2__acc_adder	0.08206	0.0114	1.711e-06	0.09346	0.9073
g_s0_adder_4__acc_adder	0.08165	0.009244	1.711e-06	0.0909	0.8824
g_s0_adder_6__acc_adder	0.1201	0.02798	1.678e-06	0.1481	1.438
g_s0_reg_0__acc_reg	0.4766	0.05031	2.643e-06	0.5269	5.115
g_s0_reg_1__acc_reg	0.4714	0.04964	2.389e-06	0.5211	5.058
g_s0_reg_2__acc_reg	0.4772	0.04935	2.662e-06	0.5265	5.111
g_s0_reg_3__acc_reg	0.4963	0.05446	2.631e-06	0.5507	5.346
g_s1_adder_0__acc_adder	0.1375	0.01878	2.115e-06	0.1563	1.517
g_s1_adder_2__acc_adder	0.1419	0.02139	1.883e-06	0.1632	1.585
g_s1_reg_0__acc_reg	0.6343	0.07393	3.387e-06	0.7082	6.876
g_s1_reg_1__acc_reg	0.595	0.07625	3.167e-06	0.6713	6.517
g_s2_adder_0__acc_adder	0.1688	0.024	2.365e-06	0.1928	1.872
g_s2_reg_0__acc_reg	0.7898	0.06484	4.149e-06	0.8547	8.297

The **combinational multiplier** consumes 2.5 mW of power.

Total Power

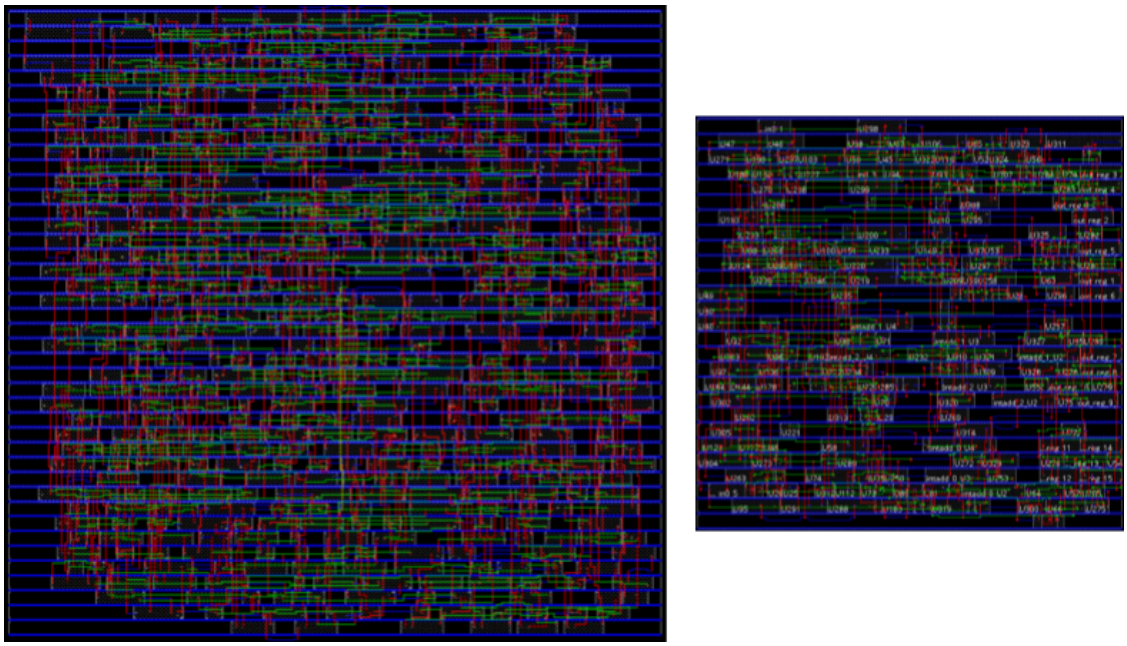
Total Internal Power:	1.91007316	77.4983%
Total Switching Power:	0.55456115	22.5005%
Total Leakage Power:	0.00003071	0.0012%
Total Power:	2.46466502	

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.5631	0.0213	2.476e-06	0.5844	23.71
Macro	0	0	0	0	0
IO	0	0	0	0	0
Combinational	1.347	0.5333	2.823e-05	1.88	76.29
Clock (Combinational)	0	0	0	0	0
Clock (Sequential)	0	0	0	0	0
Total	1.91	0.5546	3.071e-05	2.465	100

Rail	Voltage	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
VDD	1.8	1.91	0.5546	3.071e-05	2.465	100

Layout

Below is the final layout of the pipelined (left) and combinational (right) multipliers without filler cells (roughly in relative scale).



Comparative Analysis

Metric	Pipelined Multiplier	Combinational (*) Multiplier
Process Node	*TSMC 180 nm*	
I/O	*INT8 x INT8 = INT16*	
Area	*16,590 μm^2*	*6,900 μm^2*
Power	*10.3 mW*	*2.5 mW*
Performance	*333 MHz (3 ns)*	*250 MHz (4 ns)*
Implementation	*High complexity*	*Low complexity*
Physical Design	*Dependent on RTL*	*Dependent on synthesis tool*

The combinational multiplier consumes 146% and 312% less area and power, respectively, than the pipelined multiplier due to the numerous registers between each pipeline stage and repeated adders in the accumulation reduction tree. However, the pipelined multiplier's critical path only goes through a combinational adder, which intuitively expends less logic and time than the combinational multiplier to finish its compute per cycle. As a result, the pipelined multiplier's clock frequency can be 33% higher than that of the combinational multiplier (note that the $*$ is triggered by a rising clock edge).

The pipelined multiplier has higher implementation complexity than the combinational multiplier because the former requires various boolean and RTL considerations while the latter simply uses the $*$ operator. This means that the pipelined multiplier's final physical layout is highly

dependent on the architect's choices while the combinational multiplier is entirely dependent on the synthesis tool. Thus, designing a pipelined multiplier may provide more control and knowledge of the hardware (at the expense of increased production time) than designing a combinational multiplier.

Summary

- The combinational (`*`) multiplier is best for applications requiring area- and power-efficient single-cycle operations at the cost of lower throughput and losing control over physical synthesis.
- The pipelined multiplier is best for applications requiring high throughput and more control over physical synthesis at the cost of additional area, power, and implementation complexity.