New Identities and Transformations for Hardware Power Operators

Romain Michard, Arnaud Tisserand, and Nicolas Veyrat-Charvillon

Abénaire Project, LIP (CNRS–ENS–INRIA–UCBL), École Normale Supérieure de Lyon, 46 allée d’Italie, F-69364 Lyon, France;
Arith Group, LIRMM (CNRS–UM2), 161 rue Ada, F-34392 Montpellier, France.

ABSTRACT

In this work we present some improvements on hardware operators dedicated to the computation of power operations with fixed integer exponent ($x^3, x^4, \ldots$) in unsigned radix-2 fixed-point or integer representations. The proposed method reduces the number of partial products using simplifications based on new identities and transformations. These simplifications are performed both at the logical and the arithmetic levels. The proposed method has been implemented in a VHDL generator that produces synthesizable descriptions of optimized operators. The results of our method have been demonstrated on FPGA circuits.

Keywords: Computer arithmetic, integer power operation, square, cube, hardware operator, VLSI, FPGA

1. INTRODUCTION

Square, cube or higher order power operations with fixed integer exponent are frequently used in digital signal processing or graphics applications.\textsuperscript{1,2} In digital integrated circuit implementations of such powering operations, one usually replaces standard multipliers by dedicated powering operators. These operators use logical identities in the partial products array (PPA). Due to the symmetries and other simplifications in the PPA, the dedicated operators are significantly smaller and faster than standard multipliers.

In this work we use simplifications both at the logical and the arithmetic levels. The state of the art simplifications are extended and generalized. We also introduce new identities that improve the reduction of the PPA. The application order of these identities is a major concern to minimize the circuit area. In the literature on this subject, this problem is not covered. In this paper, we propose a method that allows a fast reduction of the PPA. Our method works for $x^n$ with $n$ a fixed integer larger than or equal to 3.

This paper is organized as follows. The background and previous works are presented in Section 2. The proposed method is detailed in Section 3. Implementation results and comparisons are reported in Section 4. Section 5 concludes the paper.

2. BACKGROUND

2.1. Notations

Here we deal with $x^n$. The argument $x$ is a $w$-bit value in radix-2 unsigned fixed-point or integer format. The bits of $x$ are noted $x_{w-1}, x_{w-2}, \ldots, x_0$. The exponent $n$ is a fixed (i.e., constant in time) integer and $n \geq 3$.

Logical formulas are to be read the following way:

- $\overline{a}$ stands for “not $a$”;

Further author information:
Romain Michard: E-mail: Romain.Michard@ens-lyon.fr
Arnaud Tisserand: E-mail: Arnaud.Tisserand@lirmm.fr
Nicolas Veyrat-Charvillon: E-mail: Nicolas.Veyrat-Charvillon@ens-lyon.fr
Figure 1. Partial products array (PPA) for the square of a 5-bit unsigned integer.

- \( a \land b \), also noted \( ab \), stands for “a and b”;
- \( a \lor b \) stands for “a or b”;
- \( a \Rightarrow b \) stands for “a implies b”.

The partial products array (PPA) is graphically represented as depicted in Figure 1 (case of the square of a 5-bit unsigned integer). All the partial products (PP) in the column of rank \( r \) have a weight equal to \( 2^r \).

We need to perform arithmetic operations on the PPs after transformations. For the “conversion” of logical values to binary values, we use the \( [s] \) notation \( [3, \text{p. 23}] \). If \( s \) is any Boolean statement (true or false), then:

\[
[s] = \begin{cases} 
1, & \text{if } s \text{ is true}, \\
0, & \text{if } s \text{ is false}.
\end{cases}
\]

So the “arithmetic” value of \([x_i x_j \lor x_k]\) in the rank-\( r \) column of the PPA is \( 2^r \) or 0 depending on the truth of \( x_i x_j \lor x_k \). Using this notation it is now clear what the value of \([x_i x_j \lor x_k] + [x_l x_m]\) is (no possible confusion between the arithmetic and the logical operations).

A reduction rule is the arithmetic interpretation of a logical identity. Reduction rules use the \( [s] \) notation. For instance the commutativity of the logical and leads to the reduction rule \([x_i x_j] + [x_j x_i] \rightarrow 2 \cdot [x_i x_j] \).

A transformation is a sequence of reduction rules. We show in Section 3.3 that the application order of the reduction rules in transformations is an important parameter. The order of reduction rules inside a transformation is denoted by \( T_{x \rightarrow y \rightarrow z} \) where rule \( R_x \) is applied first, then rule \( R_y \) and finally \( R_z \).

2.2. Previous Works

2.2.1. Square

We present here some useful results on dedicated square operators. The presented solutions may be used in higher order powering operators. We have to recall that our method, proposed in Section 3, does not improve square operators.

A folded table-based square method is presented in \( [4] \). The table of squares is repeatedly folded using mathematical properties of the squaring. Implementation results in CMOS technology show that this approach provides a large improvement compared to a conventional ROM implementation.

Symmetries of PPA in squaring operation can also be used in bit-serial architectures \( [5] \).

Implementations of multiplication and squaring on FPGAs with specialized \( 18 \times 18 \)-bit multiplier blocks are proposed in \( [6] \). Even for this heterogeneous architecture, a dedicated squarer is about half the size of a multiplier.

Dedicated square operators can be designed for signed values. A combined unsigned and two’s complement squarer is presented in \( [7] \).
Implementation of squarers for ASIC targets using Wallace-tree and carry-select adder for the PPA addition is presented in.8

A specialized squarer is about half the size of a direct multiplier, and is faster. It is also smaller than a booth-encoded multiplier, which requires a non-negligible amount of hardware for the operand recoding.

An example of the symmetries in the PPA for the square is presented in Figure 1 in the case of a 5-bit unsigned integer argument. The standard logical identities usually used in square operators are presented in textbooks on computer arithmetic such as [9, p. 221] and [10, p. 201]. The application of these reduction rules transforms the PPA that will give the same output, but with a lower hardware cost and/or a higher speed.

The first two standard identities rely on the properties of the logical and operation:

- Idempotency: $x_i \land x_i = x_i$. The corresponding reduction rule removes an and gate.
- Commutativity: $x_i \land x_j = x_j \land x_i$. The addition of two such PPs is replaced by a multiplication by two:

$$ [x_i x_j] + [x_j x_i] = 2 \cdot [x_i x_j] $$

Then, the term $2 \cdot [x_i x_j]$ at rank $r$ is replaced by $[x_i x_j]$ at rank $r + 1$.

Those rules are used as much as possible, since they strictly reduce the hardware area.

A third reduction rule can be used:

$$ [x_i x_j] + [x_i] = 2 \cdot [x_i x_j] + [x_i x_j] $$

This last rule trades one and gate and a half adder for two and gates and an inverter. This rule can be used only once in the transformation of the PPA on the highest column of the PPA, in order to reduce its overall height [9, p. 221]. It can also be applied on several columns in order to diminish the length of the final carry-propagate addition [10, p. 201].

2.2.2. Cube

The cube can be computed using a $n \times n$ multiplication followed by a $2n \times n$ multiplication, but some significant simplifications can be done in the PPA of the cube operators.

The case of an optimized cubing operator is presented in.11 The PPA reductions rely on an extension of the first two identities used for the square:

- Idempotency of the and operation: $x_i \land x_i \land x_i = x_i$;
- Associativity and Commutativity: $x_i \land x_j \land x_k = x_i \land x_k \land x_j = x_j \land x_i \land x_k = \ldots$

After using the corresponding reduction rules, the PPs in the reduced PPA are divided into two groups: PPs of the form $x_i x_i x_i = x_i$, that occur with a multiplicity of 1, and those of the forms $x_i x_i x_j = x_i x_j$ and $x_i x_j x_k$. These occur respectively 3 and 6 times each. The authors of11 compute the cube by adding the PPs of the first group with the second group multiplied by three. Their results show that the resulting dedicated cubing operator is faster than a multiplier-based implementation, but has a higher hardware cost.

2.2.3. Other Powering Methods

A modification of a piecewise linear approximation is presented in.12 It computes a power $x^p$ of an operand $x$. $p$ is in the form $\pm 2^k$ or $\pm 2^{k_1} \pm 2^{k_2}$, where $k, k_1$ are integers and $k_2$ is a positive integer. It can be used for accuracies up to 24 bits but no practical results are reported.

In case of a floating-point argument some polynomial approximations can be used. In,13 a degree-2 minimax polynomial approximation is used. But in this kind of solution there is a complex trade-off between the accuracy (faithful rounding for instance) and the performances.
3. NEW IDENTITIES AND TRANSFORMATIONS

In Section 3.1 we generalize the previous reduction rules. We introduce some new identities in Section 3.2. Some ideas on the application order of reduction rules inside transformations are presented in Section 3.3. Section 3.4 illustrates the transformations results on several reduced PPA examples.

3.1. Generalization of Previous Reduction Rules

The first reduction rule deals with the idempotency of multiple and identical inputs for an and gate:

\[ x_i \land x_i \cdots \land x_i = [x_i]. \quad (R0) \]

The second reduction rule relies on the associativity and commutativity of the and operation. The output of an and gate remains the same when permutations and groupings are applied to its inputs, \( x_1 \land x_2 \land x_3 \land x_4 = (x_2 \land x_4) \land (x_1 \land x_3) \) for instance. This leads to a general reduction rule:

\[ [x_i x_j \cdots x_l] + [x_{\sigma(i)} x_{\sigma(j)} \cdots x_{\sigma(l)}] = 2 \cdot [x_i x_j \cdots x_l], \quad (R1) \]

where \( \sigma \) is a permutation of the indexes \( \{i, j, \ldots, l\} \).

The third reduction Eq. (1) used for square operators can be extended straightforwardly the following way:

\[ [P \land x_i] + [P] = 2 \cdot [P \land x_i] + [P \land \overline{x_i}], \]

where \( P \) is a logical formula. This allows some hardware reduction, but a more general form can be derived. The idea behind this reduction is to add two PPs, \( P_1 \) and \( P_2 \), where \( P_1 \) is always satisfied when \( P_2 \) is (i.e., \( P_2 \Rightarrow P_1 \)). The carry bit of their addition is \( P_2 \). The sum bit is true whenever \( P_1 \) is satisfied and \( P_2 \) is not (the inverse cannot be true). So, we deduce a more general reduction rule:

\[ \text{if } P_2 \Rightarrow P_1, \text{ then } [P_1] + [P_2] = 2 \cdot [P_2] + [P_1 \land \overline{P_2}] \quad (R2) \]

For instance \([x_1 x_2 x_3] + [x_1] = 2 \cdot [x_1 x_2 x_3] + [x_1 \land (\overline{x_2} \lor \overline{x_3})]\). This reduction rule alone does not seem to reduce the number of PPs. But it replaces the addition of two PPs in the same column \( ([x_1 x_2 x_3] + [x_1]) \) in the previous example) by two PPs in consecutive columns \( (2 \cdot [x_1 x_2 x_3] + [x_1 \land (\overline{x_2} \lor \overline{x_3})]) \) in the previous example). An example of transformation using this reduction rule for the cube is illustrated in Figure 2.

![Figure 2. Example of reduction sequence in a cube operator.](image)

3.2. New Identities and Reduction Rules

The transformations based on R2 introduce PPs with complemented bits. In some cases the sum of PPs cannot generate a carry. For instance \([x_1 x_2] + [\overline{x_3}] < 2\). Then a half-adder (HA) can be replaced by a 2-input or gate. This simplification leads to a new reduction rule for two logical formulas \( P_1 \) and \( P_2 \):

\[ \text{if } P_1 \land \overline{P_2}, \text{ then } [P_1] + [P_2] = [P_1 \lor P_2]. \quad (R3) \]

The condition \( P_1 \land \overline{P_2} \) implies that \( P_1 \) and \( P_2 \) cannot be true in the same time. Then there is no possible carry and the xor gate of the HA can be replaced by a simple or gate.

Another reduction rule can be used, it is based on the law of middle excluded:

\[ [Q \land x_i] + [Q \land \overline{x_i}] = [Q], \quad (R4) \]

where \( Q \) is a logical formula. Applying this reduction before rule R3, additional reductions are possible.
3.3. Transformations Optimization

As seen in figure 2, the PPs generated using one reduction rule may be used in following reductions. Then, the application order of the reduction rules in transformations is a major concern to minimize the circuit area. This problem seems to not be covered in the literature. It is not computationally feasible to choose the optimal sequence of reductions in the PPA. We study here some possible orders in the transformations.

Figure 3 presents an example of such a difference for the cube of 3-bit values. The upper part of the figure is the initial PPA. The result using the two transformations $T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$ and $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$ are illustrated on the bottom part of the figure. The result obtained using transformation $T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$ requires an adder from rank 3 up to the most significant column. The transformation $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$ leads to a solution without addition but more complex logical cells.

### Ranks:

<table>
<thead>
<tr>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

| Initial PPA: |
| $x_2$ | $x_1x_2$ | $x_0x_2$ | $x_0x_1x_2$ | $x_0x_2$ | $x_0x_1$ | $x_0$ |
| $x_1x_2$ | $x_1x_2$ | $x_0x_2x_1$ | $x_0x_1$ | $x_1x_0$ |
| $x_2x_1$ | $x_2x_1$ | $x_1x_0x_2$ | $x_2x_0$ | $x_1x_0$ |
| $x_0x_2$ | $x_2x_1$ | $x_1x_0$ | $x_1x_0$ | $x_0$ |
| $x_2x_0$ | $x_1x_2$ | $x_1x_0$ | $x_0$ | $x_0$ |
| $x_2x_0$ | $x_2x_0$ | $x_2x_0$ | $x_2x_0$ | $x_0$ |

Result using transformation $T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$:

| $x_0x_1x_2$ | $x_1x_2x_0$ | $x_2(x_1 \lor x_0)$ | $x_0x_2$ | $x_0x_1$ | $x_0x_2x_0$ | $x_1x_0$ | $x_0$ |

Result using transformation $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$:

| $x_1x_0x_2$ | $x_2x_1x_0$ | $x_2x_0 \lor x_1x_0 \lor x_1x_2$ | $x_0x_1x_2 \lor x_0x_2x_0 \lor x_0x_2x_1$ | $x_0x_2$ | $x_0x_1$ | $x_0$ |

**Figure 3.** Example of initial and reduced PPA for the cube and $w = 3$ using different orders ($T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$ and $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$).

Figures 4 and 5 illustrates the reduction of column at rank 3 from example Figure 3 using the $T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$ and $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$ transformations respectively.

**Figure 4.** Reduction of rank-3 column in example Figure 3 using transformation $T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$. 
Every rule transforms two PPs from rank \( r \) into at most one at rank-\( r \) and one at rank \( r + 1 \). This means that a good way of applying rules is to process the PPA from the least significant toward the most significant columns. Then, in a column, the number of PPs that can be reduced is maximized.

Each time a couple of PPs is reduced, a new PP may be introduced in the column. The column has to be tested for other reductions. By keeping track of the PPs that have already been tested, the computation time is significantly reduced.

Different orders have been tested. An obvious one is an extension of the textbook method, were our new rules are applied after R1 and R2. This seems a reasonable choice, since the rule R1 is the most hardware efficient, and the new rules R4 and R3 rely on the complemented bits generated by rule R2. This gives transformation \( T_{0\rightarrow1\rightarrow2\rightarrow4\rightarrow3} \).

The most efficient order found is \( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} \). By first applying R2, more complemented bits are introduced in the new PPs, which are then used for R4 and R3. The remaining identical PPs are finally simplified using R1. This order results in less PPs in the reduced PPA. These PPs are usually more complex than those produced by the first order, but this has little impact on LUT-based FPGA implementations.

Table 1 reports the number of PPs obtained using several orders in the case of cube for an operand width of 16 and 24 bits. This shows that the transformation \( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} \) seems to be the best one among all the tested transformations.

<table>
<thead>
<tr>
<th>transformation</th>
<th>( w = 16 )</th>
<th>( w = 24 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} )</td>
<td>997 100%</td>
<td>3723 100%</td>
</tr>
<tr>
<td>( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} )</td>
<td>1090 109%</td>
<td>3916 105%</td>
</tr>
<tr>
<td>( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} )</td>
<td>1127 113%</td>
<td>4071 109%</td>
</tr>
<tr>
<td>( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} )</td>
<td>1134 114%</td>
<td>4062 109%</td>
</tr>
<tr>
<td>( T_{0\rightarrow2\rightarrow4\rightarrow3\rightarrow1} )</td>
<td>1465 147%</td>
<td>4892 131%</td>
</tr>
</tbody>
</table>

Table 1. Number of PPs using several transformation orders for the cube.

### 3.4. Reduction Examples

Example of a reduced PPA for the cube and \( w = 4 \):

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2x_1x_3 )</td>
<td>( x_3(x_1x_2 \vee x_0x_1) )</td>
<td>( x_3(x_0x_1 \vee x_2x_1 \vee x_2x_0) )</td>
<td>( x_3(x_1x_0x_3 \vee x_1x_0x_5 \vee x_1x_0x_7) )</td>
<td>( x_3(x_1x_1x_0 \vee x_2x_1) )</td>
</tr>
</tbody>
</table>
Some operators have been generated using the proposed method for $x^3$ and $w \in \{4, 8, 12, 16, 20, 24\}$ bits. Implementations have been done on Spartan 3 (XC3S1500-4) FPGAs using ISE8.1.03i tools both from Xilinx. Synthesis was area-oriented with a normal effort and place-and-route was processed with a standard effort.
Table 2. Implementation results for state of the art methods (9, p. 221 and 10, p. 201).

Table 3 reports the implementation results using our method for two different orders. The left part presents the results for the transformation $T_{0\rightarrow 1\rightarrow 2\rightarrow 4\rightarrow 3}$ and the right part those for $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$. Transformation $T_{0\rightarrow 2\rightarrow 4\rightarrow 3\rightarrow 1}$ seems to be the best one among all the tested transformations even after implementation.

Table 3. Implementation results for our method.

Figure 6 summarizes the implementation results for the cube operators and $w \in \{4, 8, 12, 16, 20, 24\}$ bits. The results are not so good for the larger argument widths. But we think this is due to the FPGA structure, better relative improvement would be obtained on ASIC implementations.

5. CONCLUSION

Some improvements on hardware operators dedicated to the power operation with fixed integer exponent (i.e., $x^3, x^4, \ldots$) have been proposed. The proposed method reduces the number of partial products. It is based on transformations at the logical and the arithmetic levels using new identities and some extension of the state of the art ones. This paper also deals with the application order of the reduction rules in the transformations. The proposed method has been implemented in a VHDL generator that produces synthesizable and optimized operators.

For moderate argument width the proposed method leads to 10–30% area improvement. The area reduction is small for larger argument width. Only a small speed improvement is reported.

The generated operators currently compute the exact value of the result. In the future, we plan to develop truncated version of these operators. We will work on signed representations for this operation. Another interesting future prospect would be the generation of ASIC and low-power consumption versions of these operators. We also plan to apply our results to function approximation methods.
Figure 6. Relative area and speed improvement of our method compared to the state of the art methods.

REFERENCES


