A Slack-based Approach to Efficiently Deploy Radix 8 Booth Multipliers

Alberto A. Del Barrio and Román Hermida
Architecture and Technology of Computing Systems, Universidad Complutense de Madrid (UCM), Spain
{abarriog,rhermida}@ucm.es

***Abstract*—[1]In 1951 A. Booth published his algorithm to efficiently multiply signed numbers. Since the appearance of such algorithm, it has been widely accepted that radix 4-based Booth multipliers are the most efficient. They allow the height of the multiplier to be halved, at the expense of a simple recoding that consists of just shifts and negations. Theoretically, higher radix should produce even larger reductions, especially in terms of area and power, but the recoding process is much more complex. Notably, in the case of radix 8 it is necessary to compute $3X$, X being the multiplicand. In order to avoid the penalty due to this calculation, we propose decoupling it from the product and considering $3X$ as an extra operation within the application's Dataflow Graph (DFG). Experiments show that typically there is enough slack in the DFGs to do this without degrading the performance of the circuit, which permits the efficient deployment of radix 8 multipliers that do not calculate the $3X$ multiple. Results show that our approach is 10% and 17% faster than radix 4 and radix 8 Booth based implementations, respectively, and 12% and 10% more energy efficient in terms of Energy Delay Product.**

***Index Terms*—Multiplier, Booth, radix 8, slack, modulo scheduling**

I. Introduction

Signal processing, multimedia applications and even fixed point scientific calculations are often dominated by integer addition and multiplication [1]–[4]. Hence, it is essential to improve the features of adders, multipliers as well as those structures that are based on them. Moreover, this must be done without incurring significant area or power overhead.

The fastest adders, like the Kogge-Stone prefix adder [5], exhibit a noticeable area and power penalty [6]. Nevertheless, datapaths are still dominated by multipliers, as their delay, area and power grow faster than in the case of adders [7]–[9]. Multipliers typically consist of a partial product matrix (PPM), which accumulates the partial products and reduces them to just two operands, and a last stage Carry Propagate Adder (CPA), which adds these two operands and calculates the final result. Given an mxn multiplier, the PPM is composed of mxn 1-bit partial products $p_{i,j}$, defined by Equation 1.

$$p_{i,j} = x_i y_j \ , \forall i,j, 0 \leq i < m, \ 0 \leq j < n. \quad (1)$$

where $X = x_{m-1}x_{m-2}...x_1x_0$ and $Y = y_{n-1}y_{n-2}...y_1y_0$ represent the multiplicand and the multiplier, respectively. Hence, in order to reduce the complexity of a multiplier either

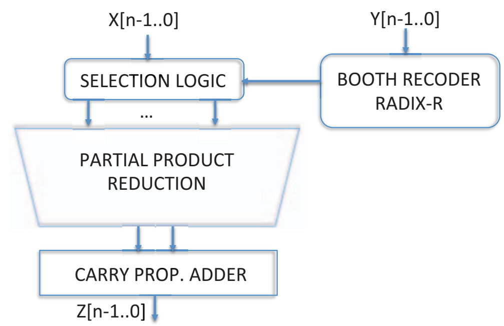


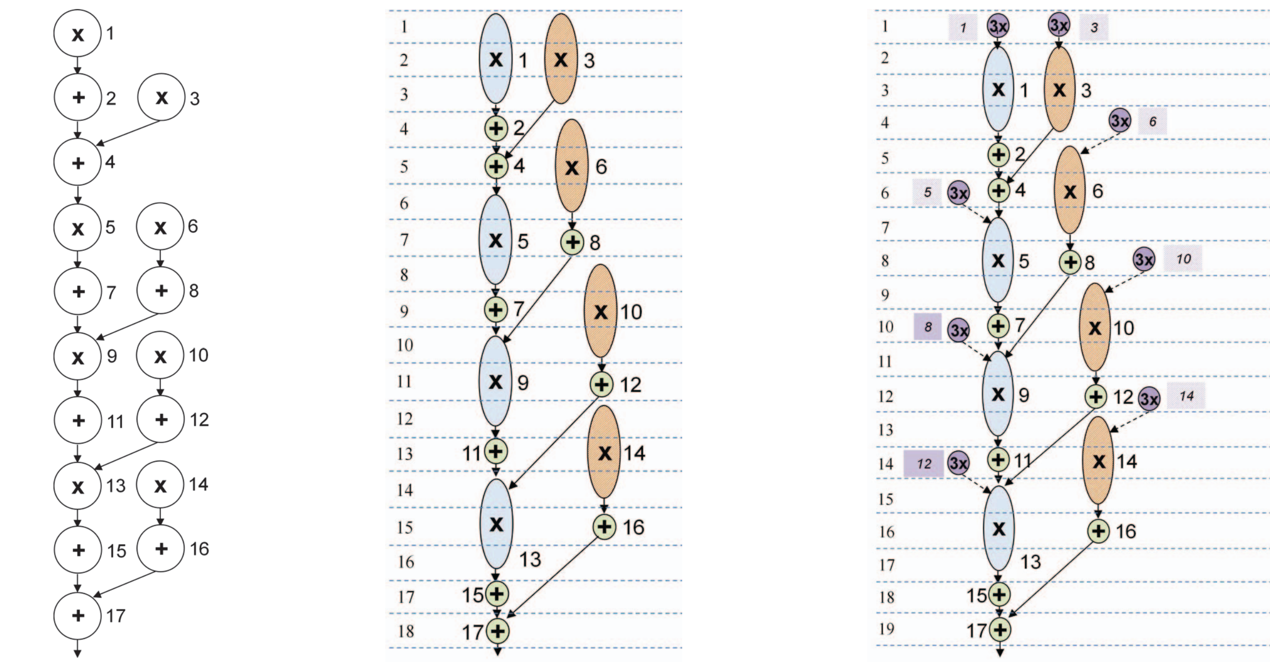
Fig. 1: Radix R Booth Multiplier

the width m or the height n must be diminished. Narrowing m leads to truncated multipliers [10]–[12], which is not the purpose of this work. On the other hand, the height of the PPM is usually reduced by applying a Booth recoding [8], [9], [13] in radix $R = 2^\beta, \beta > 0$, which maintains the accuracy of the multiplier. Thus, the height of an mxn multiplier is reduced from n to $\lceil (n+1)/\beta \rceil$. Intuitively, the larger the β the better. Nevertheless, applying this recoding technique some extra logic will be necessary to calculate the *Booth multiples* or subproducts. For this reason, typically $\beta = 2$ [14], because given a product X*Y, only ± 0X, ± 1X and ± 2X multiples need to be generated [13] and all of them can be easily calculated via shift and negation operations. For $\beta > 2$, there appear *hard multiples*, i.e., those that are not composable with only shifts and negations, and the penalty due to their calculation exceeds the gains from reduction of the PPM height. The generic structure of a Radix R Booth Multiplier is depicted in Figure 1 [8], [9], [13].

Concretely, the case of the radix-8 Booth recoding obliges to compute the following multiples: $\pm 0X$, $\pm 1X$, $\pm 2X$, $\pm 3X$ and $\pm 4X$, where $\pm 3X$ multiples are not straightforward to compute. Typically $3X$ is computed as the addition of $2X+X$ and $-3X$ as its negation, increasing the critical path of the multiplier. In this paper we leverage the existence of slack cycles in the datapath to compute these $3X$ multiples. In this way we can take advantage of the larger height reduction produced by the radix-8 recoding instead of the radix 4-based one, without negatively impacting the radix 8 Booth multiplier critical path. Experiments show that our approach outperforms both the radix 4 and radix 8 Booth based implementations in terms of execution time (10% and 17%) and Energy Delay

[1]This work has been supported by the EU (FEDER) and the Spanish MINECO, under grant TIN 2015-65277-R

978-3-9815370-8-6/17/$31.00 ©2017 IEEE



(a) DWT DFG

(b) DWT scheduled and bound

(c) DWT scheduled and bound after the inclusion of the $3X$ multiples precalculation

Fig. 2: DWT example

Product (12% and 10%).

The rest of the paper is organized as follows: Section II examines several Booth-based proposals, Section III presents an example to motivate this work and Section IV describes our flow to efficiently include the $3X$ calculations. Finally, Sections V and VI discuss our experimental results and give our concluding remarks about this work.

II. Related Work

Table I depicts the truth table for the radix 8 Booth encoding. As can be observed, every four bits $t_p = y_{3*p+2}y_{3*p+1}y_{3*p}y_{3*p-1}$ produce two outputs, namely: the selection bits, labelled as $SelectEnc$, and the sign bit, labelled as $Sign$. In the following, the t_p bit groups shall be referred as *Booth tuples*, being $0 \leq p < \lceil (n+1)/3 \rceil$. As can be observed, there are tuples producing $\pm 3X$ hard multiples, which causes an additional delay when computing $3X$ typically as $2X + X$.

Authors in [15] propose radix 8 to implement efficient $2^n \pm 1$ multipliers. Another type of proposal is described in [16], where a Redundant Signed-Digit Booth encoding technique is introduced to remove the hard multiples, at the expense of duplicating the PPM. With the Redundant Signed-Digit system, every bit x_i is represented by two bits x_i^+ and x_i^-, such that $x_i = x_i^+ - x_i^-$. The main advantage is that addition becomes carry free, but in the multiplier a PPM is necessary for accumulating the multiples due to the x_i^+ bits, and another one for the multiples generated by x_i^-. This problem also appears in the Floating-Point unit proposed in [17], [18].

The proposals presented in [19], [20] describe a hybrid architecture combining both radix 4 and 8. The first partial products are generated using radix 4 recoding, while the $3X$ computation is taking place, and the later ones are generated using radix 8 recoding. On the other hand, in the work presented in [21], authors try to optimize a radix 8 Booth multiplier thanks to a prior knowledge of the inputs. In order to reduce the aforementioned $3X$ delay, some works have proposed a partial-carry save addition [22]–[24] for computing this hard multiple [25]. In partial carry-save an operand $X = U + A$, but unlike total carry-save only certain positions may contain a '1', i.e. every k-bits [22], [24], [25]. This allows diminishing the $3X$ computation time, but at the expense of increasing the number of bits added in the PPM [22], [25] or propagating the carries through the whole datapath [24].

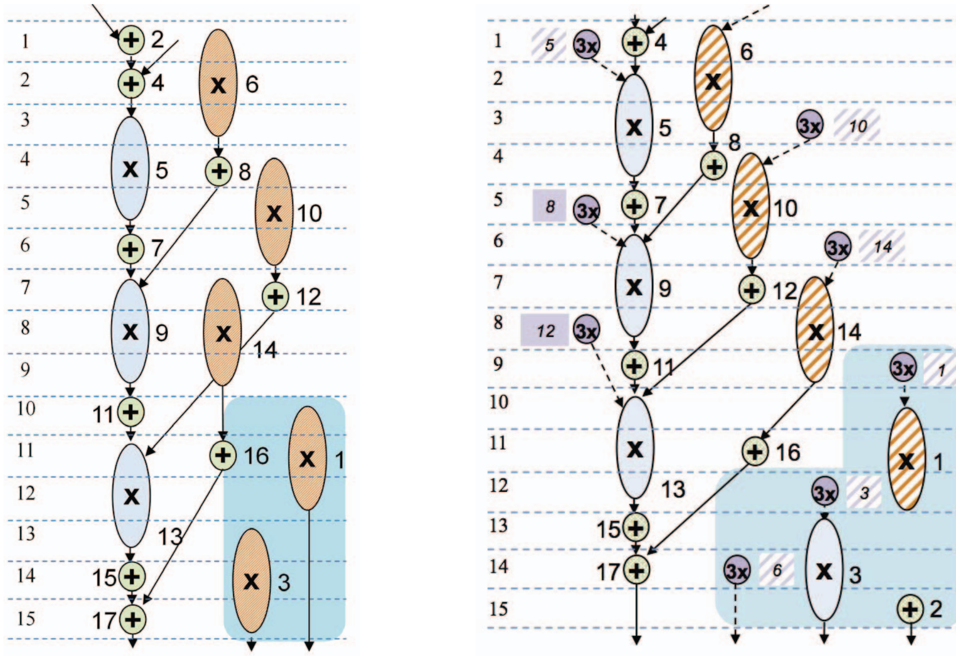
Overall, this type of approaches [19], [20], [25] get a performance similar to a conventional radix 4 Booth multiplier, but with area and power ranging between the radix 4 and radix 8 Booth implementations. Leveraging the slack that often appears when scheduling a DFG, in this paper we propose precalculating the $3X$ hard multiples to deploy highly efficient radix 8 Booth multipliers, achieving execution time savings with respect to both radix 4 and radix 8 Booth implementation styles, while getting an area and an energy lower than the radix 4 implementation and close to the radix 8 one.

III. Motivational example

In this section, an example to show the benefits from our proposal will be depicted. The first issue to be noticed is the trivial PPM reduction that happens when recoding with radix 8 Booth technique, instead of radix 4. It is clear that diminishing the PPM from $\lceil (n+1)/2 \rceil$ to $\lceil (n+1)/3 \rceil$ notably improves delay, area and power consumption.

TABLE I: Radix 8 Booth Recoder

y_{3*p+2}	y_{3*p+1}	y_{3*p}	y_{3*p-1}	Select	Select Enc	Sign	y_{3*p+2}	y_{3*p+1}	y_{3*p}	y_{3*p-1}	Select	Select Enc	Sign
0	0	0	0	+0X	000	0	1	0	0	0	-4X	100	1
0	0	0	1	+1X	001	0	1	0	0	1	-3X	011	1
0	0	1	0	+1X	001	0	1	0	1	0	-3X	011	1
0	0	1	1	+2X	010	0	1	0	1	1	-2X	010	1
0	1	0	0	+2X	010	0	1	1	0	0	-2X	010	1
0	1	0	1	+3X	011	0	1	1	0	1	-1X	001	1
0	1	1	0	+3X	011	0	1	1	1	0	-1X	001	1
0	1	1	1	+4X	100	0	1	1	1	1	-0X	000	1



(a) DWT scheduled and bound with RC-modulo scheduling

(b) DWT scheduled and bound after the inclusion of the $3X$ multiples precalculation, and using RC-modulo scheduling

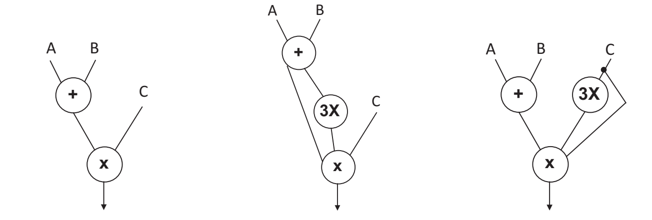
Fig. 3: DWT example with Resource Constrained Modulo Scheduling (RCMS) targeting λ=15 cycles, and with 2 multipliers, 1 adder and 1 tripler as resource set

The second issue to illustrate is the slack. For this purpose, Figure 2 shows an example based on the Discrete Wavelet Transform (DWT) [1], [2]. Figure 2a and 2b depict the DFG as well as a possible scheduling and binding, considering 2 multipliers and 1 adder. Multipliers and adders possess a latency of 3 and 1 cycles, respectively. On the other hand, Figure 2c contains a scheduling and binding including the $3X$ multiple calculations. As these computations can be performed as $2X + X$, they have been modelled as an addition, i.e. with a latency of 1 cycle. The $3X$ calculations are shown in purple color, with an adjacent box indicating the operation that they are tripling. If this operation is a primary input, there is a light purple adjacent box, and a darker one otherwise. For instance, let us consider *Operation 9*, whose predecessors are *Operations 7* and *8*. It is important to notice that *Operation 8* has been selected as the input to triple, as it provides the largest slack. On the contrary, selecting *Operation 7* would increase the critical path, i.e. the latency. As can be observed, an extra cstep will be necessary for precomputing the $3X$ values for *Operations 1* and *3*, which impacts over the total latency of the circuit. It is worth mentioning that in order to avoid the extra cstep due to the $3X$ computation for *Operations 1* and *3*, these calculations could be generated in a prior iteration, in a *modulo scheduling* fashion [26], [27]. This can be observed in Figures 3a and 3b, where the same latency is obtained using the same resource set as in Figure 2. Note that including the $3X$ nodes in Figures 2c and 3b implies employing another adder, aka tripler.

Therefore, as illustrated in this motivational example, and will be shown in the experiments, slack cycles use to appear during the scheduling phase. Our proposal leverages this slack to strategically introduce the $3X$ computations in such a way that the circuit latency is not affected or, if affected, the increase is minimized.

IV. Including the $3X$ calculations

In this section our methodology to introduce extra nodes in the DFG to compute the $3X$ operations will be described in detail. But prior to describing it, the formulation of our problem is as follows:



(a) A DFG example (b) Wrong inclusion of the 3X node (c) Correct inclusion of the 3X node

Fig. 4: An example to illustrate the 3X inclusion

Given: (1) a DFG $G(V, E)$ that represents the operations and dependencies in the circuit.

Goal: (1) build a DFG $G'(V', E')$ containing the $3X$ nodes.

In order to solve this problem algorithm 1 has been devised. It introduces a $3X$ addition node per product, selecting the farthest predecessor in terms of DFG height, i.e. X, of the multiplication. In this way, the probabilities to provide enough slack to compute $3X$ are maximized. Moreover, it is important to avoid increasing the DFG height, which may impact the latency of the circuit. This situation is illustrated with the DFG depicted in Figure 4a. Considering the same FU latencies as in Section III, it is clear that in Figure 4c the circuit latency will be 4 cycles, while in Figure 4b it will become 5 cycles.

It must be noted that a more accurate solution would comprise of introducing the $3X$ nodes at the scheduling level, where the latencies of the FUs are known, and select the predecessor with the shortest path in terms of cycles. Nevertheless, we decouple this from the scheduling phase to reduce the complexity of the flow. On the one hand, if there are no FUs with large latencies, the number of nodes is a good estimator to find the shortest path. On the other hand, some scheduling algorithms, as Resource Constrained Modulo Scheduling (RCMS) [26], [27], are based on Integer Linear Programming (ILP), their execution being very slow.

Figure 5 depicts the whole design flow of our approach.

Algorithm 1 introduce3XAdditions

Input:G(V,E)
Output:G'(V',E')
$G' \leftarrow G$
for all $v \in V$ **do**
  if v *is a product* **then**
    $c \leftarrow selectCandidate(v, G')$
    if $c = null$ **then** ▷ v is a leaf node
      $x \leftarrow leftInput(v)$
      $t \leftarrow 2x + x$
      $addPredec(t, v, G')$ ▷ t is predecessor of v
    else ▷ v is not primary input
      $t \leftarrow 2c + c$
      $addPredec(t, v, G')$ ▷ t is predecessor of v
      $addPredec(c, t, G')$ ▷ c is predecessor of t
    end if
  end if
end for

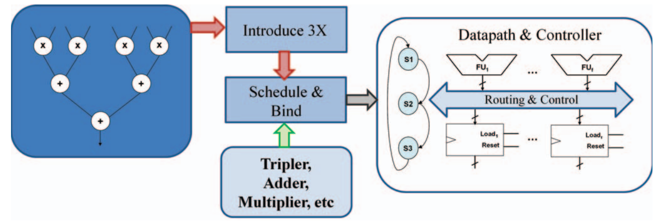


Fig. 5: Overall design flow

TABLE II: Radix 8 Booth multipliers synthesis results normalized with respect to radix 4 Booth multipliers

	Delay		Area		Power		Energy	
N	w 3x	w/o 3x	w 3x	w/o 3x	w 3x	w/o 3x	w 3x	w/o 3x
8	0.85	0.85	0.66	0.60	0.84	0.72	0.71	0.61
16	0.91	0.87	0.73	0.69	0.86	0.75	0.79	0.65
32	1.13	0.91	0.72	0.70	0.89	0.79	1.00	0.72
64	1.13	1.01	0.75	0.74	0.89	0.81	1.01	0.81
128	1.26	0.96	0.76	0.75	0.91	0.82	1.14	0.79

After introducing the $3X$ computations, the resulting graph G' will be scheduled, bound and finally synthesized, considering the FUs available in the library as well as the *Tripler* FU. This unit is responsible for computing the $3X$ calculations, which are computed as $2X + X$. Hence, given an $n - bit$ input, the Tripler is basically an $(n+1) - bit$ adder whose output will be stored and later driven to the input of the multiplier. In this way, it is possible to deploy radix 8 Booth multipliers avoiding the penalty of computing the $3X$ value.

V. Experiments

In this section our results are presented. Several multipliers as well as complete datapaths have been synthesized using *Synopsys Design Compiler* with a 65 nm library.

A. Radix 8 Booth multipliers

Several multipliers have been synthesized considering no constraints. We have measured the delay, area, power and energy of both radix 4 and radix 8 Booth multipliers. Results shown in Table II correspond with the radix 8 Booth multipliers measurements normalized with respect to the radix 4 ones. Besides, it must be noticed that for every measured magnitude two values are depicted, namely: those considering that the $3X$ multiple is being calculated within the radix 8 Booth multiplier, labelled as *w 3X*, and those considering that the $3X$ multiple has been precalculated in the datapath, labelled as *w/o 3X*.

As can be observed, in terms of delay radix 8 implementations are slightly faster when the size is small, i.e. 8 and 16-bits. However, for 32-bits and larger sizes, radix 4 implementations are better. As it is shown in column *w/o 3X*, the delay can be balanced and even shortened by precalculating the $3X$ multiple for radix 8 implementations. Regarding the area and power, it is clear that diminishing the height of the PPM from $\lceil (n+1)/2 \rceil$ to $\lceil (n+1)/3 \rceil$ achieves a noticeable reduction. As pointed by Table II, area decrease ranges from 34% to 24% for complete radix 8 Booth multipliers, and from 40% to 25% when precalculating the $3X$ multiple. In

TABLE III: Resource Constrained Modulo Scheduling (RCMS) results. The latency (λ) is given in cycles and the runtime in ms

		Booth8			Rand+1T			Rand+2T			Alg+1T			Alg+2T		
	λ	#Vars	#Eqs	Runtime	#Vars	#Eqs	Runtime	#Vars	#Eqs	Runtime	#Vars	#Eqs	Time	#Vars	#Eqs	Runtime
DES	9	198	37	50	306	60	135	306	60	110	306	59	124	306	59	78
AR	24	1344	106	128259	2112	170	-	2112	170	-	2112	162	-	2112	162	-
FFT	20	1080	99	-	1560	155	-	1560	155	-	1560	143	-	1560	143	-
FIR	24	1488	109	23397	2256	165	-	2256	165	-	2256	165	-	2256	165	-
DWT	13	442	59	25171	650	91	1125917	650	91	484798	650	91	26080	650	91	7495
DCT	19	1520	131	-	1938	183	-	1938	183	-	1938	183	-	1938	183	-
IDCT	20	1600	135	-	2040	184	-	2040	184	-	2040	184	-	2040	184	-
LMS	18	612	69	441	936	110	10740702	936	110	733172	936	107	338015	936	107	2315
LAT	12	312	52	31	432	76	189	432	76	152	432	76	192	432	76	172

the case of power, the reduction ranges from 16% to 9% for complete radix 8 Booth multipliers, and from 28% to 18% when precalculating the $3X$ multiple. In terms of energy, radix 8 implementations are more efficient just for small bitwidths. Nevertheless, this fact can be also lessened, actually getting energy reductions, by precalculating the $3X$ multiple.

Hence, it is clear that precalculating the $3X$ multiple can produce worthy improvements when dealing with radix 8 Booth multipliers. However, precalculating has a drawback: some extra hardware must be included. This will be considered when synthesizing whole benchmarks in Section V-D.

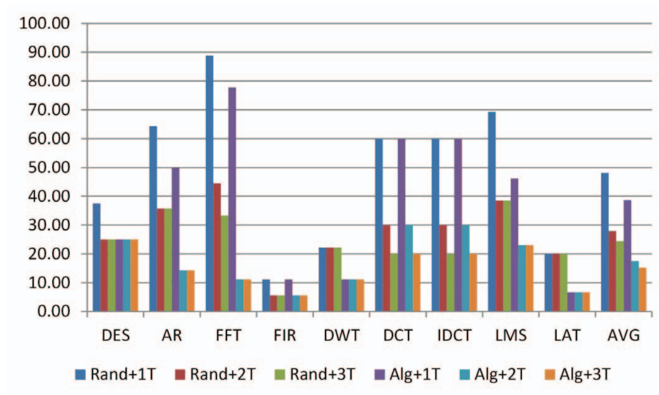
B. *Tripler effect over the latency*

In this subsection the effects of introducing the $3X$ nodes in the DFG are evaluated. Figures 6a and 6b show the percentage latency increase when introducing these extra calculations in an unconstrained and resource constrained scenario. In both cases a list-based scheduling has been employed. Two candidate selection algorithms have been utilized, namely: random (labelled as *Rand*) and Algorithm 1 (labelled as *Alg*). The suffix $+NT$ refers to the number of Tripler units that has been considered. For example, *Alg+2T* means that our algorithm is being used to select the $3X$ candidate, and that 2 Triplers will be employed when running the resource constrained scheduling algorithm. The last columns set in both figures illustrates the average results. As can be observed Algorithm 1 behaves better, always minimizing the latency penalty with respect to the random selection.

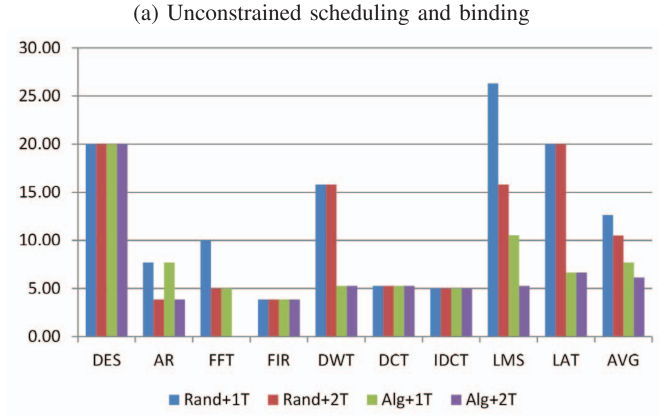
C. *Reducing latency penalty*

As has been mentioned in Section III, a possible way of mitigating the latency increase due to the introduction of the $3X$ calculations is to employ Modulo Scheduling. Concretely the time-indexed formulation presented in [26] has been utilized. The RCMS ILP algorithm has been run over an Intel i5 Dual Core at 2.4GHz, with 8 GB of RAM memory. Table III depicts the results for the radix 8 based implementation, and the 4 types of implementations shown in Figure 6b utilizing the same resource set. The target latency is shown in the second column. Next columns contain the number of generated variables (*#Vars*) and equations (*#Eqs*) by the RCMS model as well as the runtime for each kind of implementation.

As can be observed in Table III, thanks to the use of the RCMS it is possible to get the same latencies as in the case of the radix 8 Booth based implementation, regardless of the employment of Algorithm 1. Nevertheless, for many benchmarks the solution is infeasible due to the tremendous computational cost of RCMS. And for those feasible solutions, it should be noted that in general the application of our algorithm reduces the runtime. On the one hand, tripling the primary inputs instead of other operations reduces the number of equations, and on the other hand, providing more slack for the $3X$ calculations the dependency equations in RCMS are easier to satisfy.



(a) Unconstrained scheduling and binding



(b) Resource constrained scheduling and binding (2*,2+)

Fig. 6: Latency increase for Random and Algorithm 1 $3X$ nodes insertion methods, considering $\lambda_* = 3$ and $\lambda_+ = 1$ cycles

TABLE IV: Radix 8 Booth datapath synthesis results normalized with respect to the radix 4 Booth-based implementation

	Ex. Time		Area		Energy		EDP	
	w 3x	w/o 3x	w 3x	w/o 3x	w 3x	w/o 3x	w 3x	w/o 3x
DES	1.09	1.07	0.76	0.84	0.90	1.11	0.99	1.19
ARF	1.06	0.92	0.77	0.86	0.91	0.94	0.97	0.86
FFT	1.05	0.86	0.79	0.85	0.92	0.93	0.97	0.80
FIR	1.10	0.94	0.75	0.83	0.90	0.92	0.99	0.87
DWT	1.05	0.90	0.76	0.83	0.90	0.91	0.95	0.83
DCT	1.04	0.94	0.80	0.88	0.89	1.03	0.93	0.97
IDCT	1.04	0.81	0.80	0.87	0.92	1.11	0.96	0.90
LMS	1.15	0.89	0.76	0.84	0.91	0.92	1.04	0.82
LAT	1.11	0.79	0.75	0.79	0.93	0.90	1.04	0.72
AVG	1.08	0.90	0.77	0.84	0.91	0.98	0.98	0.88

D. Datapath synthesis

In this experiment several benchmarks with 32-bit precision have been synthesized to compare three types of implementations, namely: based on radix 4 Booth multipliers (*Booth4*) and based on conventional radix 8 Booth multipliers (*w 3x*) and decoupling the $3X$ calculation (*w/o 3x*). Execution time (Ex. Time), area, energy and Energy Delay Product (EDP) are shown in Table IV. In this test the list-based scheduling has been employed. Results corresponding to both radix 8 Booth implementations are shown in this table, normalized with respect to the radix 4 Booth implementation. Two multipliers, two adders and two triplers have been employed as the resource set. The adders, as well as the triplers, are based on a Kogge-Stone-like implementation.

As can be observed in Table IV, our approach reduces 10% average execution time (21% best case) with respect to the baseline, while a conventional radix 8 Booth implementation produces an increase ranging from 4% to 15% (8% on average). In terms of area and energy, the conventional radix 8 implementation obtains the best results. However, our approach is not far from those results, achieving 16% area reduction with respect to the baseline. In terms of energy, 6 out of 9 benchmarks get an energy reduction ranging from 6% to 10% (being 9% the average cut for *w 3x*). Finally, the EDP proves our approach is more efficient, with an average 12% reduction (28% best case), while the conventional radix 8 Booth implementation gets 2% EDP reduction (7% best case).

VI. Conclusions

In this paper a flow to introduce power-efficient radix 8 Booth multipliers has been proposed. In order to overcome the delay limitation imposed by the $3X$ calculation, we first decouple this computation and introduce it as an independent operation in the DFG. Our algorithm selects the farthest predecessor in terms of DFG height to provide as much slack as possible. A list-based and an ILP-based scheduling algorithms have been employed to prove the efficiency of our approach. The proposed flow achieves faster datapaths than both the radix 4 and radix 8 Booth implementations, with an energy consumption lower than the radix 4 one, and close in general to the radix 8 implementation. Overall, the tradeoff offered by our flow outperforms the aforementioned types of implementations. In the future, partial carry-save units will be incorporated to the flow to improve these results.

References

[1] S. Gupta, A. Nicolau, N. D. Dutt, and R. K. Gupta, *SPARK : a parallelizing approach to the high-level synthesis of digital circuits*. Kluwer Academic Publishers, 2004.

[2] P. Coussy and A. Morawiec, *High-Level Synthesis: From Algorithm to Digital Circuit*, 1st ed. Springer Publishing Company, 2008.

[3] A. A. D. Barrio, R. Hermida, and S. O. Memik, "Exploring the energy efficiency of multispeculative adders," in *ICCD*, 2013, pp. 309–315.

[4] A. A. D. Barrio, R. Hermida, S. O. Memik, J. M. Mendias, and M. C. Molina, "Improving circuit performance with multispeculative additive trees in high-level synthesis," *Microelectronics Journal*, vol. 45, no. 11, pp. 1470–1479, Nov. 2014.

[5] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, Aug 1973.

[6] J. L. et al., "An algorithmic approach for generic parallel adders," in *ICCAD*, 2003, pp. 734–740.

[7] P.-M. Seidel, L. McFearin, and D. Matula, "Binary multiplication radix-32 and radix-256," in *ARITH*, 2001, pp. 23–32.

[8] M. Ercegovac and T. Lang, *Digital Arithmetic*, 1st ed. Morgan Kauffman, 2003.

[9] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. AK Peters, 2002.

[10] E. J. King and E. E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *Signals, Systems amp; Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, vol. 2, Nov 1997, pp. 1178–1182 vol.2.

[11] H. J. Ko and S. F. Hsiao, "Design and application of faithfully rounded and truncated multipliers with combined deletion, reduction, truncation, and rounding," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 5, pp. 304–308, May 2011.

[12] T. A. Drane, T. M. Rose, and G. A. Constantinides, "On the systematic creation of faithfully rounded truncated multipliers and arrays," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2513–2525, Oct 2014.

[13] A. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanical Applied Mathematics*, vol. 4, pp. 236–240, mar 1951.

[14] H. H. Saleh, B. S. Mohammad, and E. E. Swartzlander, "The optimum booth radix for low power integer multipliers," in *Design and Test Symposium (IDT), 2013 8th International*, Dec 2013, pp. 1–4.

[15] R. Muralidharan and C. H. Chang, "Area-power efficient modulo $2^n - 1$ and modulo $2^n + 1$ multipliers for $2^n - 1, 2^n, 2^n + 1$ based rns," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2263–2274, Oct 2012.

[16] N. Besli and R. G. Deshmukh, "A novel redundant binary signed-digit (rbsd) booth's encoding," in *SoutheastCon, 2002. Proceedings IEEE*, 2002, pp. 426–431.

[17] A. Fahmy, A. Liddicoat, and M. Flynn, "Improving the effectiveness of floating point arithmetic," in *Signals, Systems and Computers, Asilomar Conference on*, 2001, pp. 875–879.

[18] A. Fahmy and M. Flynn, "The case for a redundant format in floating point arithmetic," in *Computer Arithmetic, IEEE Symposium on*, 2003, pp. 95–102.

[19] B. S. Cherkauer and E. G. Friedman, "A hybrid radix-4/madix-8 low power signed multiplier architecture," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 8, pp. 656–659, Aug 1997.

[20] J. C. et al., "A 600-mhz superscalar floating-point processor," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 7, pp. 1026–1029, Jul 1999.

[21] J. H. et al., "A radix-8 multiplier unit design for specific purpose," *XIII Conference of Design of Circuits and Integrated Systems*, vol. 10, pp. 1535–1546, 1998.

[22] M. Ferguson and M. Ercegovac, "A multiplier with redundant operands," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, 1999, pp. 1322–1326.

[23] S. Belloeil-Dupuis, R. Chotin-Avot, and M. H, "Exploring redundant arithmetics in computer-aided design of arithmetic datapaths," *Integr. VLSI J.*, vol. 46, pp. 104–118, Mar. 2013.

[24] A. A. D. Barrio, R. Hermida, and S. O. Memik, "A partial carry-save on-the-fly correction multispeculative multiplier," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3251–3264, Nov 2016.

[25] G. Bewick, "Fast multiplication: Algorithms and implementation," Ph.D. dissertation, UC at Stanford, 1994.

[26] B. D. de Dinechin, "From machine scheduling to vliw instruction scheduling," *ST Journal of Research*, vol. 1, no. 2, 2004.

[27] M. Ayala, A. Benabid, C. Artigues, and C. Hanen, "The resource-constrained modulo scheduling problem: An experimental study," *Comput. Optim. Appl.*, vol. 54, no. 3, pp. 645–673, Apr. 2013.