Imprecise Arithmetic for Low Power Image Processing

Pietro Albicocco, Gian Carlo Cardarilli, Alberto Nannarelli*, Massimo Petricca and Marco Re
Department of Electrical Engineering, University of Rome ”Tor Vergata”, Rome, Italy
* Dept. of Informatics & Mathematical Modeling, Technical University, Denmark

***Abstract*—Sometimes reducing the precision of a numerical processor, by introducing errors, can lead to significant performance (delay, area and power dissipation) improvements without compromising the overall quality of the processing. In this work, we show how to perform the two basic operations, addition and multiplication, in an imprecise manner by simplifying the hardware implementation. With the proposed ”sloppy” operations, we obtain a reduction in delay, area and power dissipation, and the error introduced is still acceptable for applications such as image processing.**

I. Introduction

There are several fields of application of computer arithmetic that can tolerate some imprecision. For example, in audio and image processing or in wireless communication, it might be desirable to get better performance (faster, smaller, less power-hungry systems) at expenses of some quality degradation.

Recently, a few papers have addressed this issue of designing imprecise hardware to save power [1], [2], [3], [4].

In this work, we introduce a systematic way of having imprecise arithmetic operations for the two most common operations: addition and multiplication. We liked the term *”sloppy”* introduced in [5], and we will use this term in the paper to refer to imprecise arithmetic operations.

II. Sloppy addition

Ignoring the least significant bits of an addition, by implementing a truncated adder saves area and it is faster at expenses of a truncation error. Instead of completely ignoring the least significant bits, in the ”sloppy” approach we do not propagate the carry in those bits.

Assuming that we are operating on positive integers, and defining position k as the bit of weight 2^k in a n-bit word, we can ignore the carry up to position k when implementing the addition.

The bit-level algorithm to implement this sloppy adder is the following:

```
$c = 0$ // carry
if $(i < k)$ then
    $s_i = a_i$ XOR $b_i$
else
    $s_i = a_i$ XOR $b_i$ XOR $c$
    $c = (a_i$ AND $b_i)$ OR $(a_i$ AND $c)$ OR $(b_i$ AND $c)$
end if
```

For example, the addition $103 + 70$ ($n = 8$, $k = 4$) is

```
          sloppy              precise
  A :   0110 0111 +         0110 0111 +
  B :   0100 0110 +         0100 0110 +
  c :   100- ---- =         0100 110- =
      -------------       -------------
  S :   1010 0001           1010 1101
```

That is, the sloppy adder computes 161 (exact value is 173) introducing an error $\epsilon = 12$.

By looking at the bits of weight $< 2^k$, we notice that the XOR of two ones produces a zero sum bit ($1 \oplus 1 = 0$). Because the carry is not computed (or propagated), in position k an error 2^{k+1} is generated. The error can be halved to 2^k by computing the OR of the two bits in place of the XOR. For the example above we have:

```
        sloppy (OR-ing)
  A :   0110 0111 +
  B :   0100 0110 +
  c :   100- ---- =
      -------------
  S :   1010 0111
```

and the error is reduced from $\epsilon = 12$ to $\epsilon = 6$ (halved).

By simulating all possible combinations of the operands for the 8-bit addition ($k = 4$), we found that by obtaining the sum by OR-ing the k least-significant bits the average error is $\epsilon_{mean} = 3.75$, while by XOR-ing, it is $\epsilon_{mean} = 7.5$.

We show in Fig. 1 the comparison of the hardware implementation of the sloppy adder used in the above example ($n = 8$, $k = 4$) and an error-free 8-bit carry-propagate adder (CPA). The data[1] on delay, area and power dissipation are reported in Table I.

In a rough evaluation, we considered lowering the supply voltage V_{DD} in the sloppy adder to match the delay of the error-free adder ($1.0\ ns$). In our library, when V_{DD} is lowered from $1.0\ V$ to $0.7\ V$ the delay doubles. In the expression for the power dissipated by a circuit containing N gates

$$P_{1.0V} = V_{DD}^2 f \cdot \sum^{N} a_i C_i \quad \Rightarrow \quad 20\ \mu W = (1.0\ V)^2 \cdot \mathcal{K}$$

we assume that the switching capacitance $a_i C_i$ does not change when scaling V_{DD}. Therefore, $\mathcal{K} = 20$ is constant:

$$P_{0.7V} = (0.7)^2 \cdot 20 \simeq 10\ \mu W$$

[1]The adders are synthesized with radix-4 carry-look-ahead iterative carry network [6].

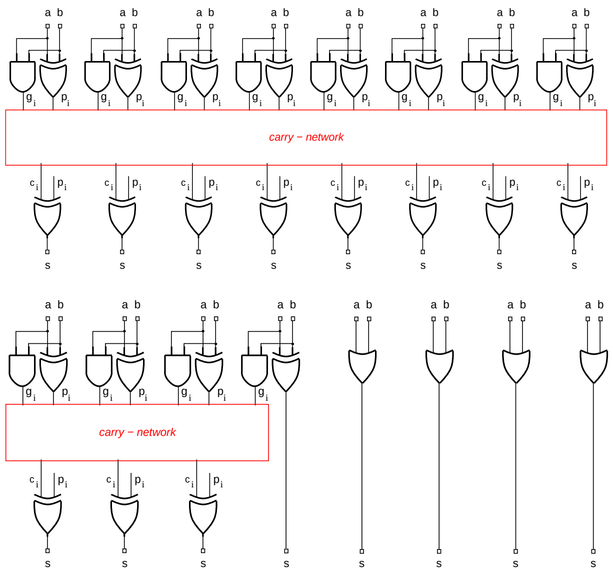


Fig. 1. Implementation of 8-bit error-free (top) and sloppy $k = 4$ (bottom) adders.

	CPA 8-bit	sloppy	ratio
max. delay $[ps]$	999	495	2.00
Area $[\mu m^2]$	191	112	1.70
Power $[\mu W]$	42	20	2.10

TABLE I
SYNTHESIS DATA OF ADDERS IN FIG. 1.

That is, with the sloppy adder the power is reduced to 1/4 at same adder speed.
The natural competitor of the sloppy adder is the truncated adder. We performed a comparison between our imprecise adder and the truncated adder by implementing a 16-bit adder with a Carry Look-Ahead (CLA) network to propagate the carry for different sloppy/truncated configurations. The output bits affected by errors are shown as $\circ$ in Fig. 2 for truncation $t = 8$ and sloppy bits $k = 8$.
Gate-level netlists are generated, by a C program, for each unit under test. The netlists are synthesized (unconstrained) to optimize buffering and cells' drive strength according to the actual fan-out.
In the comparison, we are interested in relating the introduced error to the power dissipation.
Fig. 3 shows the error introduced by the imprecise adders as a function of the number of imprecise/truncated bits (4, 8 and 12 bits). In Fig. 4 we show the power dissipation of each imprecise adder as function of the error. The sloppy adder turned out to dissipate lower power than the truncated adder for the same error level.

III. SLOPPY MULTIPLICATION

Parallel multiplication $p = x \cdot y$ can be divided into three steps:

1) generation of Partial Products (PPs);
2) carry-free reduction from n PPs to 2 operands;
3) carry-propagate two operands addition.

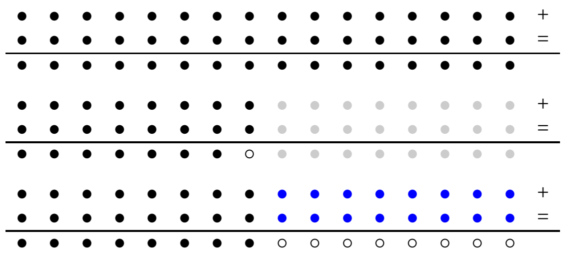


Fig. 2. Bit array and errors $\circ$ for adders: precise 16-bit adder (top), truncated $t = 8$ 16-bit adder (middle), sloppy $k = 8$ 16-bit adder (bottom).

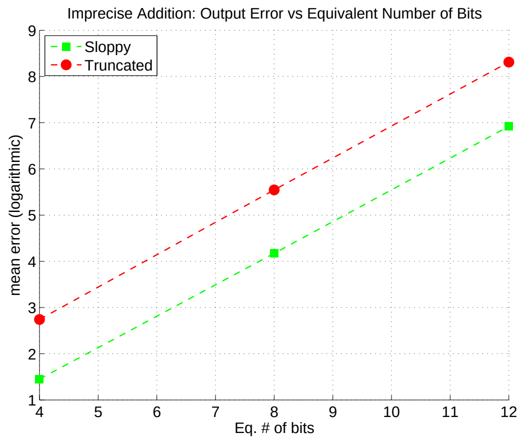


Fig. 3. Error of sloppy and truncated adders implementations for different number of imprecise bits.

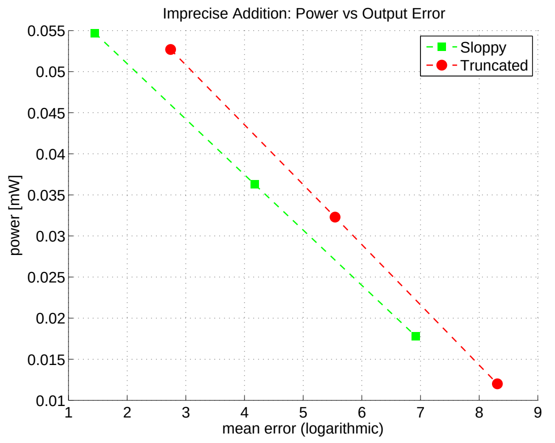


Fig. 4. Power dissipation of sloppy and truncated adders as function of the introduced error.

We use a sloppy approach for step 1 only, as step 2 is quite delay-efficient (no carry propagation) and step 3 has been addressed in the previous section.
We consider radix-4 multiplication because for $n \times n$ bit operands the unit is smaller: only $\frac{n}{2}$ PPs are generated. In radix-4 multiplication, the radix-4 digits of the multiplier y are recoded into signed-digit representation to avoid multiples of 3 and carry propagation as explained in [6]. The resulting architecture (for one digit) recoder plus PP generation (rec+PPgen)

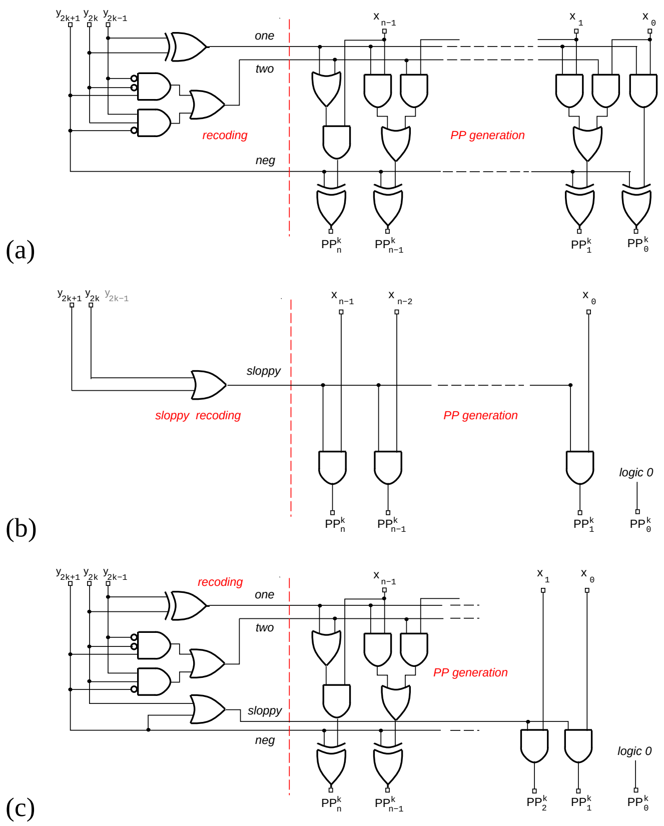
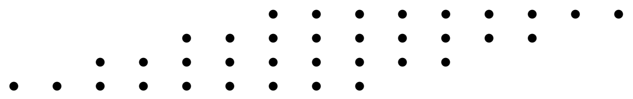
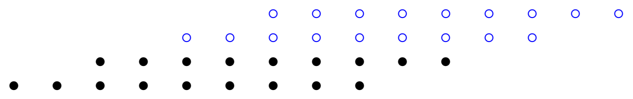

(a)
y_{2k+1} y_{2k} y_{2k-1}
x_{n-1} x_1 x_0
one
two
recoding
PP generation
neg
PP^k_n PP^k_{n-1} PP^k_1 PP^k_0
(b)
y_{2k+1} y_{2k} y_{2k-1}
x_{n-1} x_{n-2} x_0
sloppy
sloppy recoding
PP generation
logic 0
PP^k_n PP^k_{n-1} PP^k_1 PP^k_0
(c)
y_{2k+1} y_{2k} y_{2k-1}
recoding
x_{n-1} x_1 x_0
one
two
PP generation
sloppy
neg
logic 0
PP^k_n PP^k_{n-1} PP^k_2 PP^k_1 PP^k_0

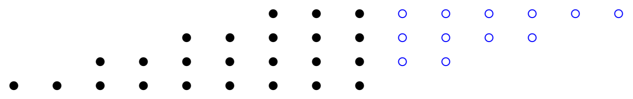
Fig. 5. Implementation of radix-4 rec+PPgen: (a) error-free, (b) whole row sloppy, and (c) sloppy columns in rows.



(a) radix-4 bit array (error-free)



(b) radix-4 bit array with 2 sloppy rows



(c) radix-4 bit array with 6 sloppy columns

Sign extension and/or correction omitted.

Fig. 6. Bit array and sloppy bits ○ for radix-4 8×8 multipliers.

Example: $(0111)_4 \times (0231)_4 = (0032301)_4$

			0	2	2	2	*sloppy*
		0	2	2	2		*sloppy*
	0	2	2	2			*regular*
0	0	0	0				*regular*
0	0	3	**1**	3	0	**2**	

Errors (boldface) are in radix-4 columns 0 and 3.

TABLE III
EXAMPLE OF ERROR COMPENSATION IN INTERNAL COLUMNS.

is sketched in Fig. 5(a).

Similarly to what was done for the addition, we have a sloppy rec+PPgen for the least-significant digits of y. The recoding is performed as shown in Table II.

The resulting hardware implementation is greatly simplified as shown in Fig. 5(b). Fig. 6(b) shows how the sloppy bits $\circ$ are arranged in the array. As the average error for sloppy recoding is zero (Table II), for patterns in which two adjacent digits in y are $1 = (01)_2$ and $3 = (11)_2$ the errors on two different rows compensate in the internal columns of the array. This is shown in the example of Table III for $(0111)_4 \times (0231)_4$.

From Fig. 6(b) it is clear that the error due to sloppy rows can propagate well into the most-significant digits of the product. To limit this propagation, we opt for a hybrid row in which only the least-significant digits of the row are computed sloppy as shown in Fig. 5(c). With this scheme, called in the following *sloppy-columns*, the propagation of the error can be limited to

radix-4 digit		PP$_k$		error
y_{2k+1}	y_{2k}	standard	sloppy	ϵ_k
0	0	0	0	0
0	1	$x \cdot 4^k$	$2x \cdot 4^k$	$x \cdot 4^k$
1	0	$2x \cdot 4^k$	$2x \cdot 4^k$	0
1	1	$3x \cdot 4^k$	$2x \cdot 4^k$	$-x \cdot 4^k$

TABLE II
SLOPPY RADIX-4 RECODING.

a given column Fig. 6(c).

Again, a competitor of the sloppy scheme is the truncated one. To compare performance and error introduced, we implemented a 12×12-bit multiplier (two's complement) in the following schemes:

1) **r2-mult** a radix-2 standard multiplier;
2) **r4-mult** a radix-4 standard multiplier (with PPs generation as in Fig. 5(a));
3) **r2-trunc** a r2-mult with t truncated bits;
4) **r4-trunc** a r4-mult with t truncated bits;
5) **sloppy-rows** a radix-4 multiplier with PPs generation as in Fig. 5(b) for k multiplier radix-4 digits (rows).
6) **sloppy-cols** a radix-4 multiplier with PPs generation as in Fig. 5(c) for t radix-2 columns (bits).

As done for the adder, we report the mean error as function of the imprecise digits/bits in Fig. 7 and the power dissipation as function of the error in Fig. 8. The power dissipation of the precise multipliers is $P_{r2} = 0.53\ mW$ for the radix-2 and $P_{r4} = 0.47\ mW$ for the radix-4 multiplier. The power figures do not include the final carry-propagate adder.

Fig. 8 shows that among the truncated schemes, radix-4 is by far more power efficient than radix-2 because of the reduced number of PPs. Moreover, from Fig. 8 we derive that the sloppy row schemes with $k = 1, 2, 3$ have very similar characteristics (error and power) to those of radix-4 truncated

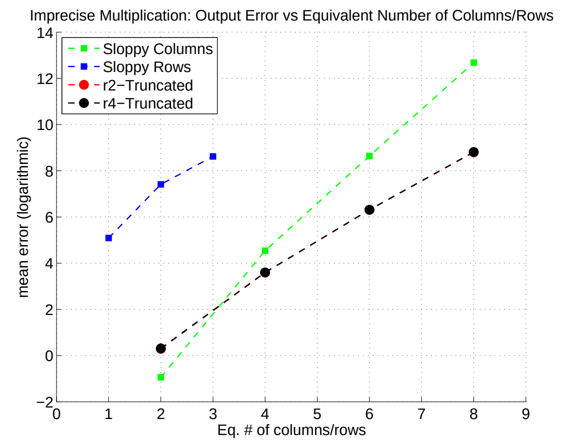


Fig. 7. Error of multiplier implementations for different imprecise schemes. Error curves for radix-2 and radix-4 truncated schemes overlap.

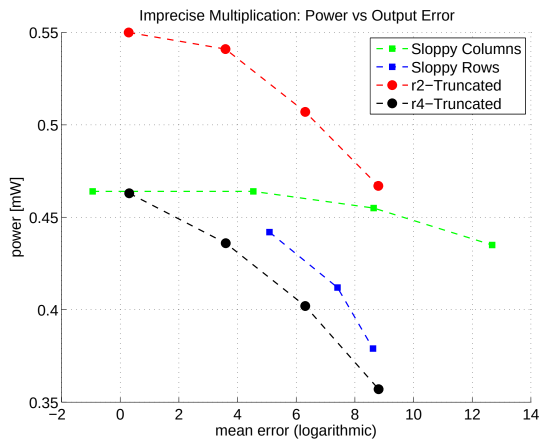


Fig. 8. Power dissipation for imprecise multipliers as function of the error.

to $t = 4, 6, 8$ bits, respectively.

IV. Applications in Image Processing

To verify the figures found in the stand-alone characterization of the imprecise operators, we implement some common image processing algorithms in imprecise hardware and evaluate the performance.
As sample pictures, we used the two grayscale (each pixel is an unsigned 8-bit integer) images of Fig. 9 (upper part).

A. Image Filtering

We use the sloppy adder defined in Sec. II with $k = 4$ sloppy bits to process two 256×256 grayscale images of Fig. 9 (top) for the following bidimensional filters:

1) an averaging (low-pass) filter;
2) a sharpening filter;
3) an edge-detection unit.

These filters can be implemented in the the spatial domain by addition and shift operations.
The error is evaluated by taking the absolute value of the difference between the precise I^{ef} and sloppy I^{sl} value of

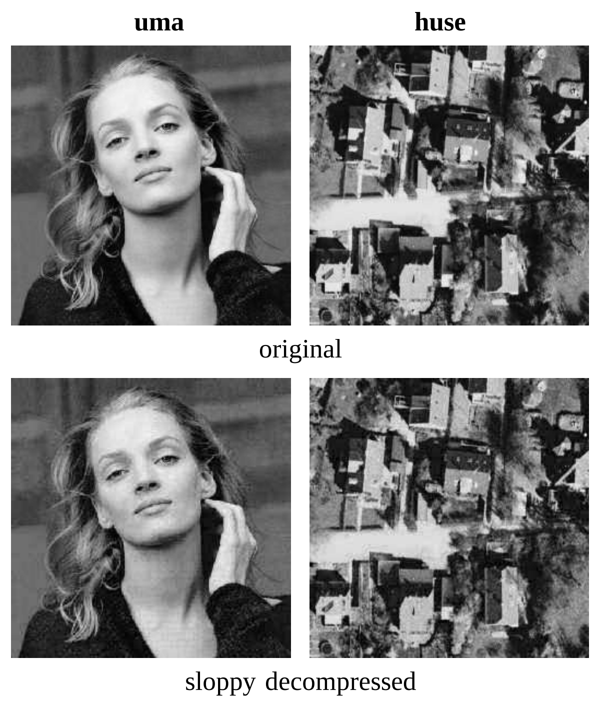


Fig. 9. Original pictures (top) and after decoding by **sloppy-row-2** IDCT (bottom).

	smoothing		*sharpening*		*edge det.*	
	ϵ_{max}	$\overline{\epsilon}$	ϵ_{max}	$\overline{\epsilon}$	ϵ_{max}	$\overline{\epsilon}$
uma	26	7.2	60	18.9	64	9.0
huse	28	7.8	59	17.5	68	9.2

TABLE IV
Error in 2D-filtered images.

intensity (luminosity) per pixel (i, j):

$$\epsilon_{i,j} = |I^{ef}_{i,j} - I^{sl}_{i,j}|$$

The maximum error ϵ_{max} and the average error $\overline{\epsilon} = \frac{\sum \epsilon_{i,j}}{N^2}$ are reported in Table IV for the different types of filtering. The results show that the degradation is independent of the image (**uma** is a portrait, while **huse** has greater detail). Depending on the filter mask, we can change the design of the sloppy adder to obtain larger savings. For example, for edge-detection, a sloppy adder with $k = 6$ has an average error $\overline{\epsilon} = 28$, but visually, the degradation is not noticeable.

B. Inverse Discrete Cosine Transformation (IDCT)

Now we combine the imprecise multiplier schemes with an error-free adder in a multiply-add (and accumulate) unit (Fig. 10) which can be used for the trivial implementation of the Inverse Discrete Cosine Transform (IDCT), which is part of the JPEG decompression algorithm.
For the unit of Fig. 10 we opted for carry-save (error-free) accumulation to keep separate the imprecision due to the multiplier and to the adder. Based on the results of software simulations, we decided not to use a sloppy adder as the extra

Unit	delay	area	**uma**			**huse**			power
MULT	$[ps]$	$[\mu m^2]$	P_{ave} $[\mu W]$	$\overline{\epsilon}$	ϵ_{max}	P_{ave} $[\mu W]$	$\overline{\epsilon}$	ϵ_{max}	ratio
r4-mult	1398	7702	208	3.7	9	284	3.8	10	1.00
r4-trunc-6	1254	5778	163	5.1	22	224	8.1	24	0.78
r4-trunc-8	1244	5197	143	24.2	115	194	42.9	129	0.68
sloppy-row-2	1286	7003	189	4.2	40	255	5.1	47	0.90
sloppy-row-3	1286	6839	180	11.3	157	239	14.7	189	0.85

TABLE V
SUMMARY OF RESULT FOR IDCT IMPLEMENTATION.

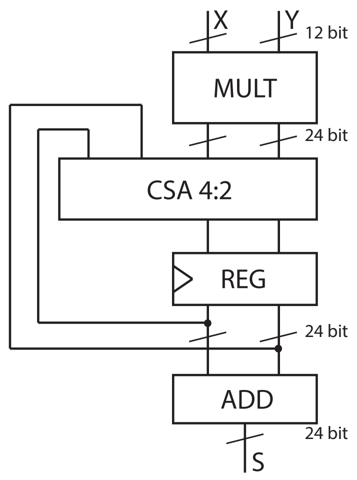


Fig. 10. Scheme of multiply-accumulate used for IDCT.

error introduced was negligible[2].

We implemented the multiply unit of Fig. 10 with several variants of imprecise multipliers. Based on Fig. 8, we excluded from the IDCT evaluation radix-2 truncated multipliers (more power hungry than all others) and the sloppy-columns schemes (power dissipation savings are marginal when the error increases). In summary, we implemented the following multiply-accumulate units:

1) **r4-mult**: radix-4 12×12-bit multiplier and 24-bit adder;
2) **r4-trunc-6**: r4-mult with $t = 6$ truncated bits and 18-bit CSA 4:2, registers and adder;
3) **r4-trunc-8**: r4-mult with $t = 8$ truncated bits and 16-bit CSA 4:2, registers and adder;
4) **sloppy-row-2**: radix-4 multiplier with $k = 2$ sloppy rows and 24-bit adder;
5) **sloppy-row-3**: radix-4 multiplier with $k = 2$ sloppy rows and 24-bit adder;

The results in Table V are obtained by implementation in a 90 nm standard cells library (clock rate is 100 MHz). The errors are computed with respect to a floating-point software implementation (quantization error for **r4-mult**).

The results show that the larger reduction in power is obtained for radix-4 truncated multipliers. This is in large part justified by the smaller area required by the accumulate circuitry (accumulate-path: CSA 4:2, two registers and final adder) that for the truncated schemes are reduced up to 33% (16 vs. 24 bit accumulate-path). For the multiplier itself, as shown in Fig. 8, the smaller sloppy rows in the sloppy scheme compensate for the larger tree when compared to the truncated multipliers.

The visual results obtained by **sloppy-row-2** IDCT computation are shown in Fig. 9 (bottom). For the **sloppy-row-3** and the **r4-trunc-8** schemes the image degradation is such that for the IDCT these multipliers are probably not acceptable. The complete visual results of the IDCT test are reported in an electronic appendix [7].

[2]In the IDCT trivial algorithm the carry-propagate addition (Fig. 10) is executed every 8×8=64 imprecise multiplications.

V. CONCLUSIONS AND FUTURE WORK

We have presented simple ways of performing addition and multiplication in an imprecise manner with the aim to get better performance (delay, area and power) at expenses of an increased error which can be tolerated in some applications. Different combinations of precise/truncated/sloppy operators can be used depending on the specific implementation of the algorithm.

In future work, we plan to characterize in term of error and performance (delay, area, power dissipation) these imprecise operators and find a systematic way of combining them to meet the desired error/performance constraint for a given application.

REFERENCES

[1] K. He, A. Gerstlauer, and M. Orshansky, "Controlled Timing-Error Acceptance for Low Energy IDCT Design," *Proc. of 2011 Design, Automation and Test in Europe Conference (DATE)*, Mar. 2011.

[2] A. Lingamneni, J.-L. N. C. Enz, K. Palem, and C. Piguet, "Energy Parsimonious Circuit Design through Probabilistic Pruning," *Proc. of 2011 Design, Automation and Test in Europe Conference (DATE)*, Mar. 2011.

[3] P. Krause and I. Polian, "Adaptive Voltage Over-Scaling for Resilient Applications," *Proc. of 2011 Design, Automation and Test in Europe Conference (DATE)*, Mar. 2011.

[4] D. Mohapatra, V. Chippa, A. Raghunathan, and K. Roy, "Design of Voltage-Scalable Meta Functions for Approximate Computing," *Proc. of 2011 Design, Automation and Test in Europe Conference (DATE)*, Mar. 2011.

[5] L. Hardesty. "The surprising usefulness of sloppy arithmetic". MIT News Office. [Online]. Available: http://web.mit.edu/newsoffice/2010/fuzzy-logic-0103.html

[6] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[7] Electronic Appendix to. "Imprecise Arithmetic for Low Power Image Processing". Nov. 2012. [Online]. Available: http://www.imm.dtu.dk/∼alna/projects/sloppy/