Karatsuba with Rectangular Multipliers for FPGAs

Martin Kumm
Johannes Kappauf
Peter Zipf

Oscar Gustafsson

Florent de Dinechin

Introduction

Introduction

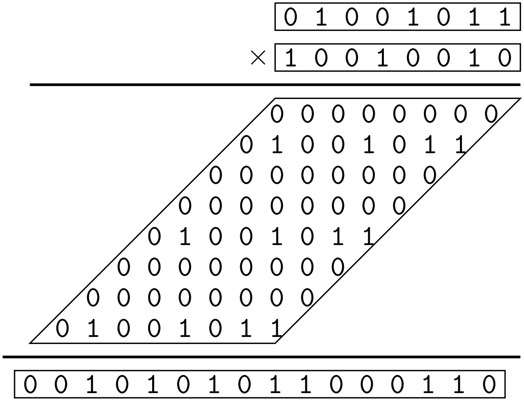
A matter of alignment

Some results

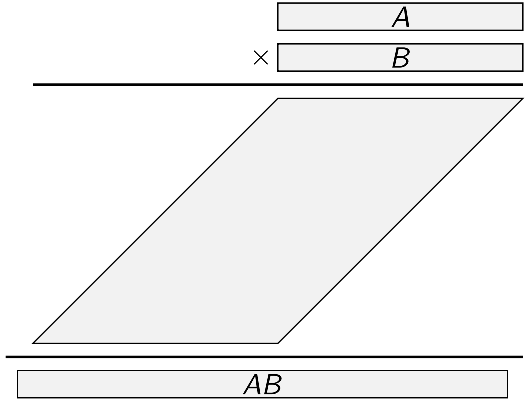
This paper is about multiplication

```
                  0 1 0 0 1 0 1 1
                × 1 0 0 1 0 0 1 0
-----------------------------------
                0 0 0 0 0 0 0 0
              0 1 0 0 1 0 1 1
            0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0
        0 1 0 0 1 0 1 1
      0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
  0 1 0 0 1 0 1 1
-----------------------------------
0 0 1 0 1 0 1 0 1 1 0 0 0 1 1 0
```

This paper is about multiplication

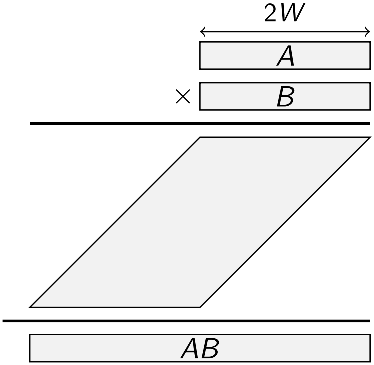


This paper is about multiplication



Classical Karatsuba

Multiplication of $A \times B$, each of size 2W bits

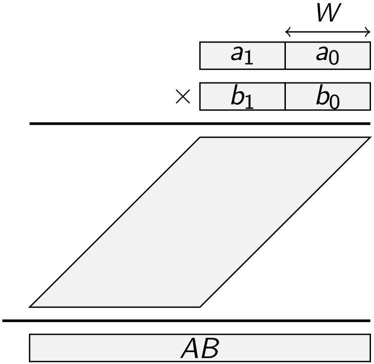


Classical Karatsuba

Multiplication of $A \times B$, each of size 2W bits

Split each input into W-bits words:

$$A \times B = (2^W a_1 + a_0) \times (2^W b_1 + b_0)$$



Classical Karatsuba

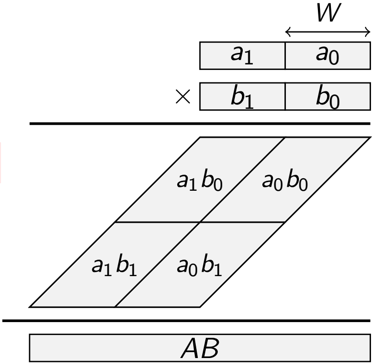
Multiplication of $A \times B$, each of size 2W bits

Split each input into W-bits words:

$$A \times B = (2^W a_1 + a_0) \times (2^W b_1 + b_0)$$

$$= 2^{2W}\, a_1 b_1 + 2^W\, a_1 b_0 + 2^W\, a_0 b_1 + a_0 b_0$$

(4 multiplications of W-bit inputs)



Classical Karatsuba

Multiplication of $A \times B$, each of size 2W bits

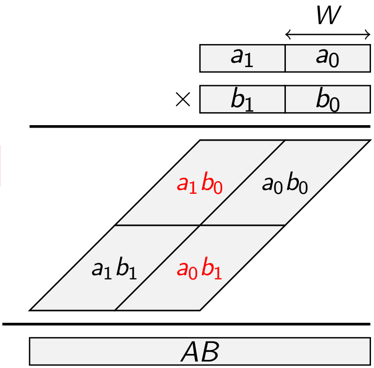
Split each input into W-bits words:

$$A \times B = (2^W a_1 + a_0) \times (2^W b_1 + b_0)$$

$$= 2^{2W} a_1 b_1 + 2^W a_1 b_0 + 2^W a_0 b_1 + a_0 b_0$$

(4 multiplications of W-bit inputs)

$$= 2^{2W} a_1 b_1 + 2^W (a_1 b_0 + a_0 b_1) + a_0 b_0$$



Classical Karatsuba

Multiplication of $A \times B$, each of size 2W bits

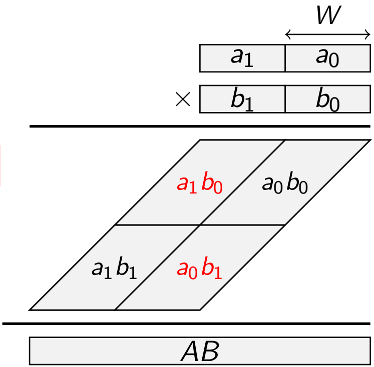
Split each input into W-bits words:

$$A \times B = (2^W a_1 + a_0) \times (2^W b_1 + b_0)$$

$$= 2^{2W}\, a_1 b_1 + 2^W\, a_1 b_0 + 2^W\, a_0 b_1 + a_0 b_0$$

(4 multiplications of W-bit inputs)

$$= 2^{2W}\, a_1 b_1 + 2^W (a_1 b_0 + a_0 b_1) + a_0 b_0$$



Karatsuba identity

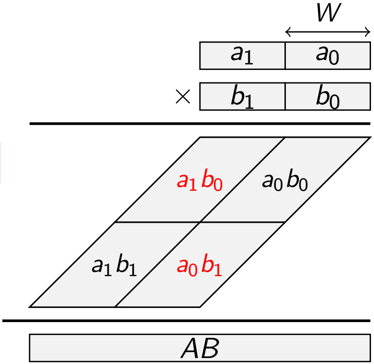
$$a_1 b_0 + a_0 b_1 = (a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$$

Classical Karatsuba

Multiplication of $A \times B$, each of size 2W bits

Split each input into W-bits words:

$$A \times B = (2^W a_1 + a_0) \times (2^W b_1 + b_0)$$



$$= 2^{2W}\, a_1 b_1 + 2^W\, a_1 b_0 + 2^W\, a_0 b_1 + a_0 b_0$$

(4 multiplications of W-bit inputs)

$$= 2^{2W}\, a_1 b_1 + 2^W (a_1 b_0 + a_0 b_1) + a_0 b_0$$

Karatsuba identity

$$a_1 b_0 + a_0 b_1 = (a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$$

3 multiplications only instead of 4

Classical Karatsuba

Multiplication of $A \times B$, each of size 2W bits

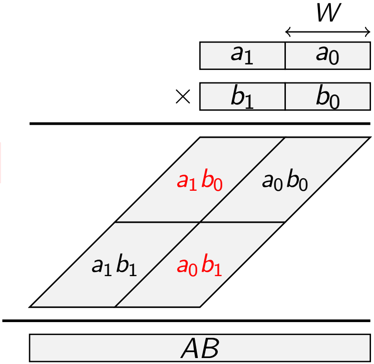
Split each input into W-bits words:

$$A \times B = (2^W a_1 + a_0) \times (2^W b_1 + b_0)$$

$$= 2^{2W} a_1 b_1 + 2^W a_1 b_0 + 2^W a_0 b_1 + a_0 b_0$$

(4 multiplications of W-bit inputs)

$$= 2^{2W} a_1 b_1 + 2^W (a_1 b_0 + a_0 b_1) + a_0 b_0$$

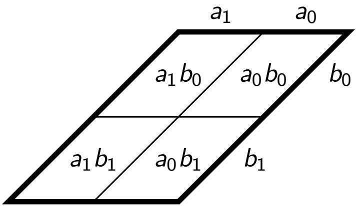


Karatsuba identity

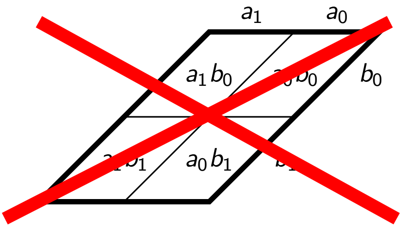
$$a_1 b_0 + a_0 b_1 = (a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$$

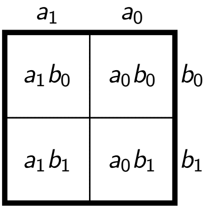
3 multiplications only instead of 4 at the cost of 4 extra additions

Graphical representation

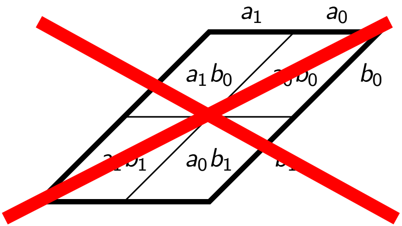


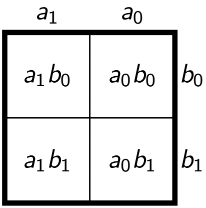
Graphical representation





Graphical representation

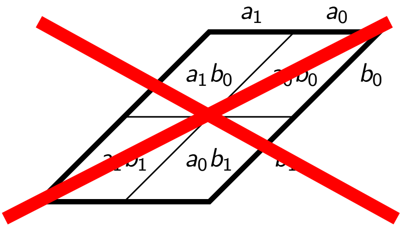


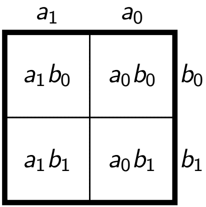


Karatsuba algorithm

- compute $a_0 b_0$ and $a_1 b_1$
- $a_1 b_0 + a_0 b_1$ computed as $(a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$

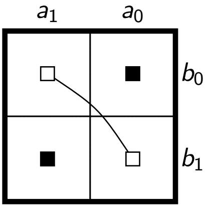
Graphical representation



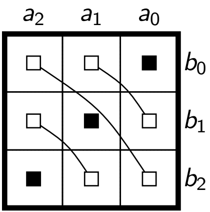


Karatsuba algorithm

- compute $a_0 b_0$ and $a_1 b_1$
- $a_1 b_0 + a_0 b_1$ computed as $(a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$

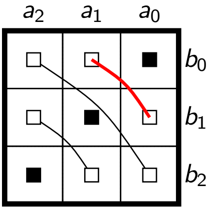


3-part Karatsuba



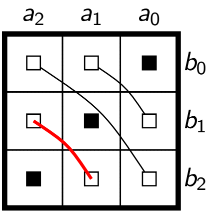
- compute $a_0 b_0$, $a_1 b_1$ and $a_2 b_2$

3-part Karatsuba



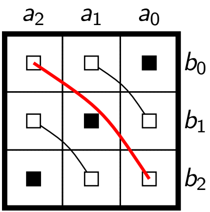
- compute $a_0 b_0$, $a_1 b_1$ and $a_2 b_2$
- $a_1 b_0 + a_0 b_1$ computed as
  $(a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$

3-part Karatsuba



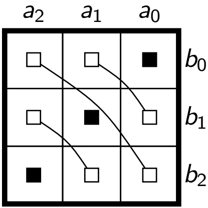
- compute $a_0 b_0$, $a_1 b_1$ and $a_2 b_2$
- $a_1 b_0 + a_0 b_1$ computed as
  $(a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$
- $a_2 b_1 + a_1 b_2$ computed as
  $(a_2 - a_1)(b_1 - b_2) + a_1 b_1 + a_2 b_2$

3-part Karatsuba



- compute $a_0 b_0$, $a_1 b_1$ and $a_2 b_2$
- $a_1 b_0 + a_0 b_1$ computed as $(a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$
- $a_2 b_1 + a_1 b_2$ computed as $(a_2 - a_1)(b_1 - b_2) + a_1 b_1 + a_2 b_2$
- $a_2 b_0 + a_0 b_2$ computed as $(a_2 - a_0)(b_0 - b_2) + a_0 b_0 + a_2 b_2$

3-part Karatsuba



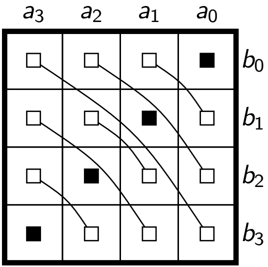
- compute $a_0 b_0$, $a_1 b_1$ and $a_2 b_2$
- $a_1 b_0 + a_0 b_1$ computed as
  $(a_1 - a_0)(b_0 - b_1) + a_0 b_0 + a_1 b_1$
- $a_2 b_1 + a_1 b_2$ computed as
  $(a_2 - a_1)(b_1 - b_2) + a_1 b_1 + a_2 b_2$
- $a_2 b_0 + a_0 b_2$ computed as
  $(a_2 - a_0)(b_0 - b_2) + a_0 b_0 + a_2 b_2$

6 multiplications instead of 9

4-part Karastuba

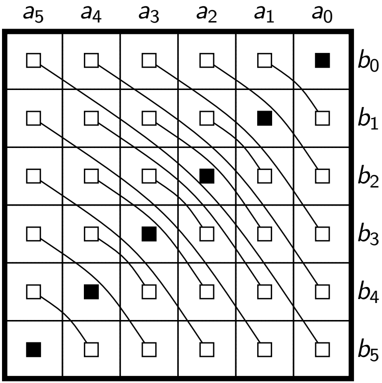
Two choices here:

- more of the same: 10 multipliers



- recursion on Karatsuba-2: 9 multipliers (but two levels of pre-adders).

More of the same



In general, Karatsuba-N uses $N(N+1)/2$ multipliers

This is nice for FPGAs

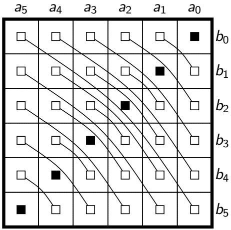
… which embed up to 2000 small multiplier ($W = 17$ bits)
Large multipliers that we could wish to build:

- 53×53 bits for double precision
- 113×113 for quad-precision
- 2560×2560 bits for fully homomorphic encryption

A word on sign management

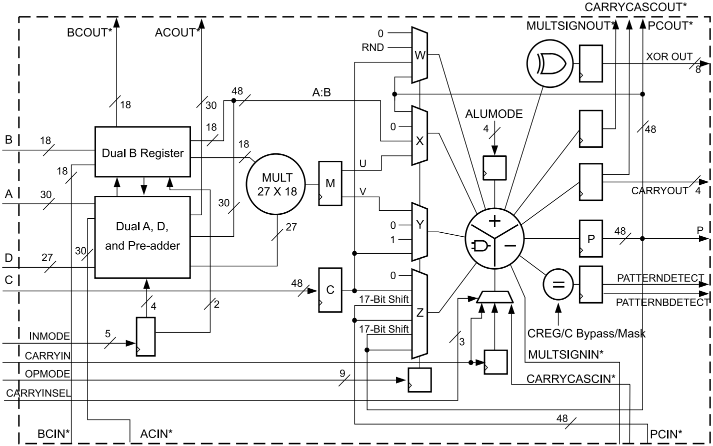
- We are interested in large *unsigned* multiplications
  - the a_i and b_j are unsigned, and so are the $a_i b_j$
- Our FPGA devices have signed multipliers, e.g. signed 18x18
  - can be used as unsigned 17x17
  - so the tile size should be 17x17
- Two variants of the Karatsuba formula
  - $a_0 b_1 + a_1 b_0 \;=\; (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$
  - $a_0 b_1 + a_1 b_0 \;=\; (a_1 - a_0)(b_0 + b_1) + a_0 b_0 + a_1 b_1$
  - In both cases the new multiplier is one bit larger

- The variant with presubtraction needs a signed 18x18: perfect match!
- … for the majority of subproducts.



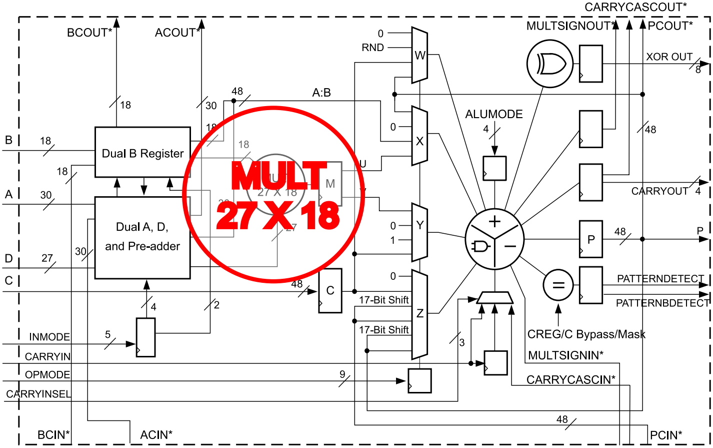
But multipliers in recent Xilinx devices are not square

(figure cut from Xilinx Ultrascale documentation)



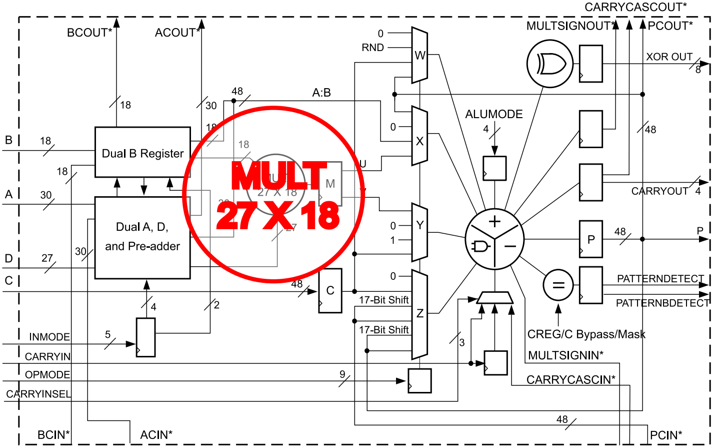
But multipliers in recent Xilinx devices are not square

(figure cut from Xilinx Ultrascale documentation)



But multipliers in recent Xilinx devices are not square

(figure cut from Xilinx Ultrascale documentation)



Can we use them to build Karatsuba multipliers?

Previous state of the art

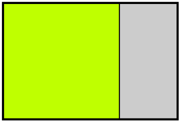
Yes you can !

Previous state of the art

Yes you can !

- either under-used

  as 18×18-bit (signed) multipliers
  or 17×17-bit (unsigned) ones

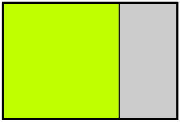


Previous state of the art

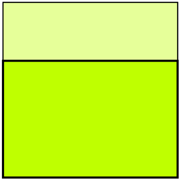
Yes you can !

- either under-used

  as 18×18-bit (signed) multipliers
  or 17×17-bit (unsigned) ones



- or, complemented to 27×27 bit (signed),
  using soft logic

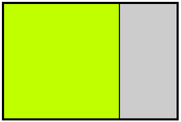


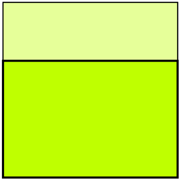
Previous state of the art

Yes you can !

- either under-used

  as 18×18-bit (signed) multipliers
  or 17×17-bit (unsigned) ones





- or, complemented to 27×27 bit (signed),
  using soft logic

The present work

A solution with less waste (for large multiplications)

A matter of alignment

Back to the Karatsuba formula

The Karatsuba formula

$$a_i b_j + a_k b_\ell \; = \; (a_i + a_k)(b_j + b_\ell) - a_i b_\ell - a_k b_j$$

can be used if

the products $a_i b_j$ and $a_k b_\ell$ have the same weight

i.e.

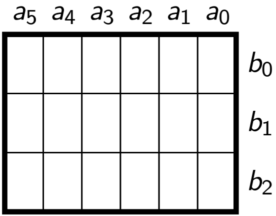
$$2^{Wi+Wj} \; = \; 2^{Wk+W\ell}$$

or

$$i + j \; = \; k + \ell$$

Tiling with rectangular multipliers

- split A in 17-bit chunks,
- split B in 26-bit ones,
- tile the large multiplication;
- a tile corresponds to a DSP.

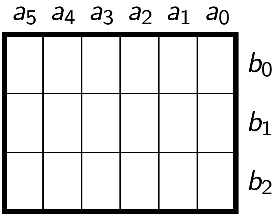


The Karatsuba formula can be used if

the products $a_i b_j$ and $a_k b_\ell$ have the same weight

Tiling with rectangular multipliers

- split A in 17-bit chunks,
- split B in 26-bit ones,
- tile the large multiplication;
- a tile corresponds to a DSP.



The Karatsuba formula can be used if

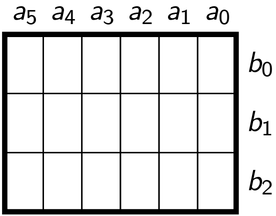
the products $a_i b_j$ and $a_k b_\ell$ have the same weight

$$2^{17i} 2^{26j} \quad = \quad 2^{17k} 2^{26\ell}$$

$$\text{or} \qquad 17i + 26j \; = \; 17k + 26\ell \qquad \text{or} \qquad 17(i-k) = 26(\ell - j)$$

Tiling with rectangular multipliers

- split A in 17-bit chunks,
- split B in 26-bit ones,
- tile the large multiplication;
- a tile corresponds to a DSP.



The Karatsuba formula can be used if

the products $a_i b_j$ and $a_k b_\ell$ have the same weight

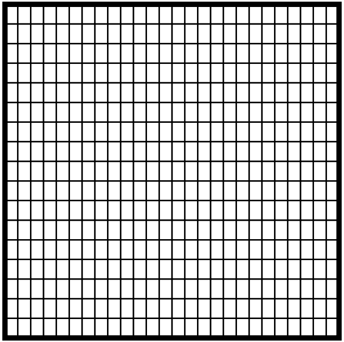
$$2^{17i}2^{26j} \quad = \quad 2^{17k}2^{26\ell}$$

$$\text{or} \qquad 17i + 26j \;=\; 17k + 26\ell \qquad \text{or} \qquad 17(i-k) = 26(\ell - j)$$

17 being prime with pretty much anything, including 26,
$i - k$ must be a multiple of 26 and $(\ell - j)$ must be a multiple of 17...

So far no good

... We could save one DSP block out of 486 in this 486-bit multiplier

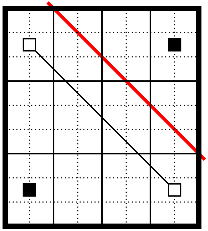


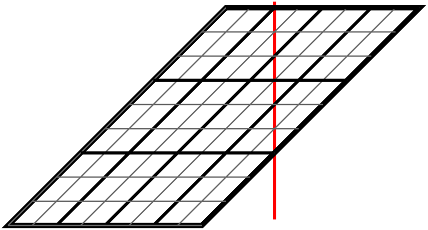
So 17 is not a good number

Now consider a tile whose dimensions have a common factor W
for example: **16x24** with $W = 8$

- $16 = 2 \cdot 8$ $\quad (W = 8, M = 2)$
- $24 = 3 \cdot 8$ $\quad (W = 8, N = 3)$

(a few other combinations of (W, M, N) studied in the paper)
Tiles split into 8x8 squares, now looking for $2(i - k) = 3(\ell - j)$
And we find two aligned tiles in the figure below:





… their sum can be computed using a single 17x25 signed product

Fighting the evil mischiefs of the fanatics of 17

… require primal(ity) sacrifices.

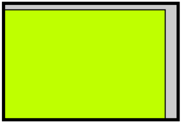
Fighting the evil mischiefs of the fanatics of 17

... require primal(ity) sacrifices.

Our DSPs shall be

- either ($W = 8$) under-used

as 16×24 bits (unsigned)
or 17×25 bit (signed)

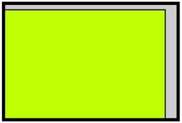


Fighting the evil mischiefs of the fanatics of 17

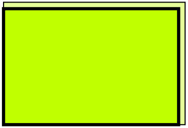
… require primal(ity) sacrifices.

Our DSPs shall be

- either ($W = 8$) under-used
  as 16×24 bits (unsigned)
  or 17×25 bit (signed)



- or ($W = 9$), complemented by soft logic
  to 18×27 bit (unsigned)
  or 19×28 bit (signed)

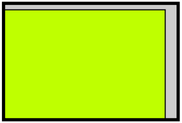


Fighting the evil mischiefs of the fanatics of 17

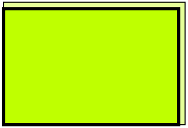
... require primal(ity) sacrifices.

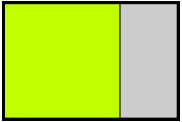
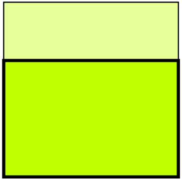
Our DSPs shall be

- either ($W = 8$) under-used
  as 16×24 bits (unsigned)
  or 17×25 bit (signed)

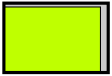


- or ($W = 9$), complemented by soft logic
  to 18×27 bit (unsigned)
  or 19×28 bit (signed)

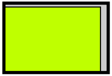
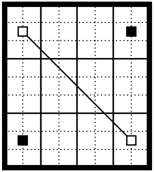
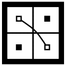


Less waste than  or 

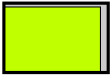
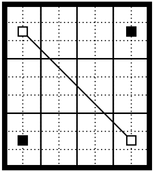
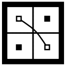
The trade-off now

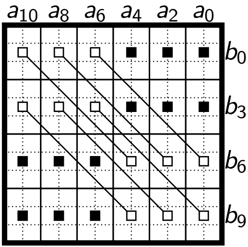
- Better use of the DSP resource:  versus 

The trade-off now

- Better use of the DSP resource:  versus 
- but fewer Karatsuba opportunities:  versus 

The trade-off now

- Better use of the DSP resource:  versus 
- but fewer Karatsuba opportunities:  versus 



(it gets better for larger multipliers, e.g. 96x96)

Polynomial interpretation

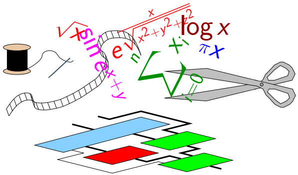
A large multiplier of size $W(N+1)M \times W(M+1)N$
corresponds to the polynomial multiplication
of an $N+1$-term M-sparse polynomial
by an $M+1$-term N-sparse polynomial,
both in $X = 2^W$:

$$\left(\sum_{j=0}^{N} a_{jM} X^{jM}\right)\left(\sum_{k=0}^{M} b_{kN} X^{kN}\right) \quad = \quad \sum_{i=0}^{2MN} c_i X^i. \qquad (1)$$

- Identifying coefficients on both sides gives Karatsuba opportunities
- ... and arguments of optimality.
- Patterns studied in the paper:
  - $W(N+1)M \times W(M+1)N$
  - $2WNM \times 2WMN$
  - $W(2N+1)M \times W(2M+1)N$

Some results

The big tables are in the paper



- implementation with comparable effort in FloPoCo
- comparison of
  - square Karatsuba,
  - rectangular tiling,
  - rectangular Karatsuba.
- Different soft spots, hence difficult to compare
- Executive summary: rectangular Karatsuba saves DSP and logic for multipliers large enough

Operation counts

Square				Rectangular				
	Karatsuba				Tiling	Karatsuba		
Size	Mult	Pre-add	Post-add	Size	Mult = Post-add	Mult	Pre-add	Post-add
51×51	6	6	6	48×48	6	6	0	6
68×68	10	12	22	64×72	12	11	2	13
102×102	21	30	51	96×96	24	18	5	30
119×119	28	42	70	112×120	35	27	7	43

Actual sythesis results on Virtex-6

Implementation in FloPoCo
(state-of-the-art compression techniques for the final summation)

	Size	DSPs	LUTs	Cycles	f_{max} [MHz]
Square Kara.	68 × 68	10	1405	11	215.1
Rect. Tiling	64 × 72	12	764	10	217.5
Rect. Kara.	64 × 72	11	867	10	247.0
Square Kara.	102 × 102	21	2524	13	192.0
Rect. Tiling	96 × 96	24	1586	13	215.1
Rect. Kara.	96 × 96	18	2032	14	195.1
Square Kara.	119 × 119	28	3438	15	192.6
Rect. Tiling	112 × 120	35	2293	16	218.9
Rect. Kara.	112 × 120	27	2292	14	190.1

Conclusion and future work

- Significant reduction in DSP and pre-adder counts,
- Translate predictably to reduction in resource consumption
- Effective only for very large multipliers (well beyond double precision)
- Paper was written for Virtex6 with 18x25, series 7 has 18x27
- Other challenges for homomorphic-grade multipliers...

Other tricks for smaller multipliers