

# Xilinx 7 Series DSP Slice

# Work on RSA signing speed

- 1. How fast can we sign?
- 2 How to increase performance?
- 3. Progress so far
- 4. Further steps

- ModExp is the cornerstone of RSA
- Outer loop does "Montgomery Ladder":
- L iterations per L-bit exponent
- 2x area requirement

# one iteration of "Montgomery Ladder"

else: if D & (1 << bit): (T1, T2) = (MMM(X1, X2), MMM(X2, X2))(T1, T2) = (MMM(X1, X1), MMM(X2, X1))

X1, X2 = T1, T2

- Inner loop is Montgomery Modular **Multiplication:**
- 2·(L/16)<sup>2</sup> sub-products must be computed for L-bit operands
- $2 \cdot (L/16)^2$  word additions must be interleaved with multiplications

#### # MMM if T >= N: T for ai in A: ۲Щ. + V V II + - -|| How fast can we sign? 16 LW ם. ב tO | || || 岕 岕 岕 n, Ζ Β # reduction # multiplication

- CRT improves speed by 8x
- modulus and exponent 2x shorter
- two "easier" exponentiations, that can be done at the same time

- Total number of operations per ModExp:
- $S = L^3 / 512$
- Fmax is 460 MHz (-1 speed grade)

1	FMAX	Maximum Frequency	
· · · · ·	With all registers used		
	628.93		
	550.66		
	464.25		
	550.66 464.25 464.25 464.25 363.77 MHz		
***	464.25		
· · · · · · · · · · · · · · · · · · ·	363.77		
	MHz		

- Total of 740 multipliers per device (A-7 200T)
- Multiplier count must be power of 2, up to
- 512 useable per core

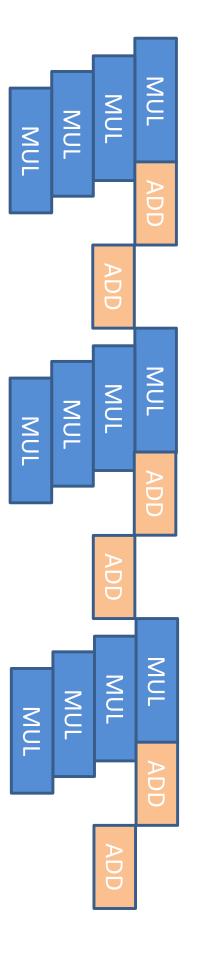
Fmax and no pipeline bubbles, N = number of Theoretical signatures per second (assumes multipliers):

2048- bit key	1024- bit key	
27	219	N = 4
54	438	N = 8
109	877	N = 16
219	1754	N = 16 N = 32 N = 64
438	3509	
877	7019	N = 128
1754	7019*	N = 256
1754*	7019*	N = 512
	27 54 109 219 438 877 1754	2194388771754350970197019*27541092194388771754

- I/O overhead not taken into account
- Difficult to completely avoid pipeline bubbles
- Operand broadcasting wastes time
- done after ModExp Last part of CRT ("Garner's formula") must be

- Reasons for current core slowness:
- Interleaved multiplication and reduction in MMM coefficient in advance) (can be done in parallel by computing "magic"
- Carry propagation during addition kills speed since multipliers) the end of the inner loop) => 2x space carry-save adders, carries are propagated once at requirement (needs as many adders as there are multiplier pipeline is stalled (can be done using

Better scheduling of operations (before)

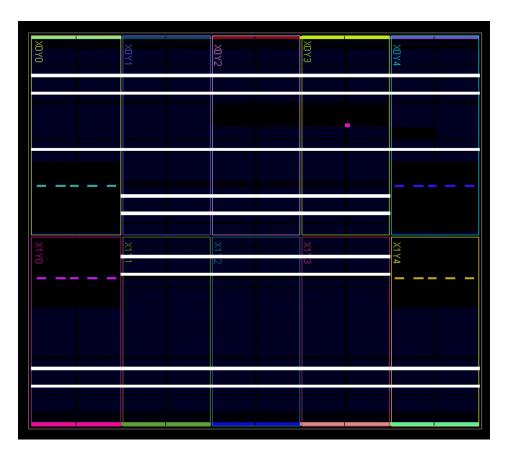


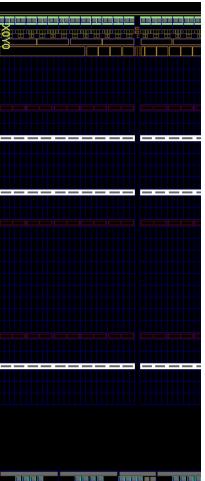
Better scheduling of operations (after)

				MUL	MUL	MUL	MUL
				_			Σ
ADD	ADD	ADD	ADD	MUL	MUL	MUL	MUL
							2
ADD	ADD	ADD	ADD	MUL	MUL	MUL	MUL
							7
ADD	ADD	ADD	ADD	MUL	MUL	MUL	MUL

- Reasons for current core slowness:
- Systolic architecture not suitable for fieldincreasing array size limits clock speed narrow long columns, not rectangular array), programmable devices (multipliers arranged in

# DSP slice layout (5 cols of 100, 4 cols of 60)





0000000		6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6		( <sup>4</sup> <sup>2</sup> 6 <sup>2</sup>	4 <sup>9</sup> 4 <sup>9</sup> 4 <sup>9</sup> 4 <sup>9</sup> 4 <sup>9</sup>	
	8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		88 \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$			
			8888 *** ****		8888 *** ****	₽₽₽₽ ∀∀∀ <b>{}</b> {} <b>?</b>
	88 88 88 88 88 88 88 88 88 88 88 88 88		8888 222 \$***		88888	
	00000 \$\$\$\$ \$\$\$\$ \$	0000000	00000 \$\$\$\$	₽₽₽₽₽ ♥♥♥ ₽ <sup>₽</sup> ₽ <sup>₽</sup> ₽ <sup>₽</sup> ₽	00000	₩₩₩₩ ₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩

#### Progress so far

- Colin D. Walter "Hardware Aspects of Montgomery Modular Multiplication"
- Many references from above
- Complete python model that mimics exactly how FPGA does ModExp using DSP slices
- Verilog for modular multiplier (360+ MHz, 32 multipliers and 32 adders)

#### Progress so far

<sup>™</sup> narrow broa not dumm	🕌 narrow_broalayer1_now	"⊌ wide_piece_is_dummy1	14 wide_piece_is_dummy2	🕌 mult_pipeline_filled	> 책 m_narrow[15:0]	🕌 recalc_m_narrow3	🕌 recalc_m_narrow2	🕌 recalc_m_narrow1	> 😻 fast_t0[15:0]	> 😽 mult_p[0:7][31:0]	> 😻 mult_b[0:7][15:0]	> 😼 mult_a[0:7][15:0]	¼ mult_ce_not_dummy	🕌 mult_ce	MULT	> 👹 a_narrow[15:0]	> 👹 a_narrow_addr[4:0]	A_NARROW	🕌 rdy	ا trig_now	🕌 ena	🕌 dk	Name		Q ■ @ X	modmul.v × tb2.v × tb2
<b>_</b>	1	0	0	0	bf94	0	0	0	808c	X000000X,X000000X	cfad,d03b,da5a,d02d	bf94,bf94,bf94,bf94,b		1		42	31		1	0	0	4	Value		±, ⊼ ∑ f	tb2.wcfg* X modexp_dsp_top.v
-200 ns					0000				X00X	20000000,2000000,200000	,2000,2000,2000,2000,2000	,000,000,000,000,000,000				×	×						200 ns		<b>₽</b> <b>+</b> <b>-</b> <b>-</b> <b>-</b>	dsp_top.v x
0 ns 200																	01234547						400 ns  600	401.250 ns		
200 ns 400 ns									<u> </u>														00 ns  800 ns			
600 ns 800																	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX						1,000 ns 1,20			
ns 1,000 ns	937.500 ns									00162654,000e	X00X												1,200 ns  1,400 ns	1,338.750 ns		

#### Progress so far

- PoC Verilog modular multiplier ~1 us cycle for 1024-bit modulus (in CRT mode)
- for 1024-bit keys Expected speed is ~2000 signatures/second
- Extrapolated speed (twice larger modulus => 8x longer operation):
- ~250 sigs/sec for 2048-bit keys
- ~30 sigs/sec for 4096-bit keys

#### Further steps

- Finish outer ModExp loop
- Implement "Garner's formula" part of CRT in hardware
- Blinding in hardware?