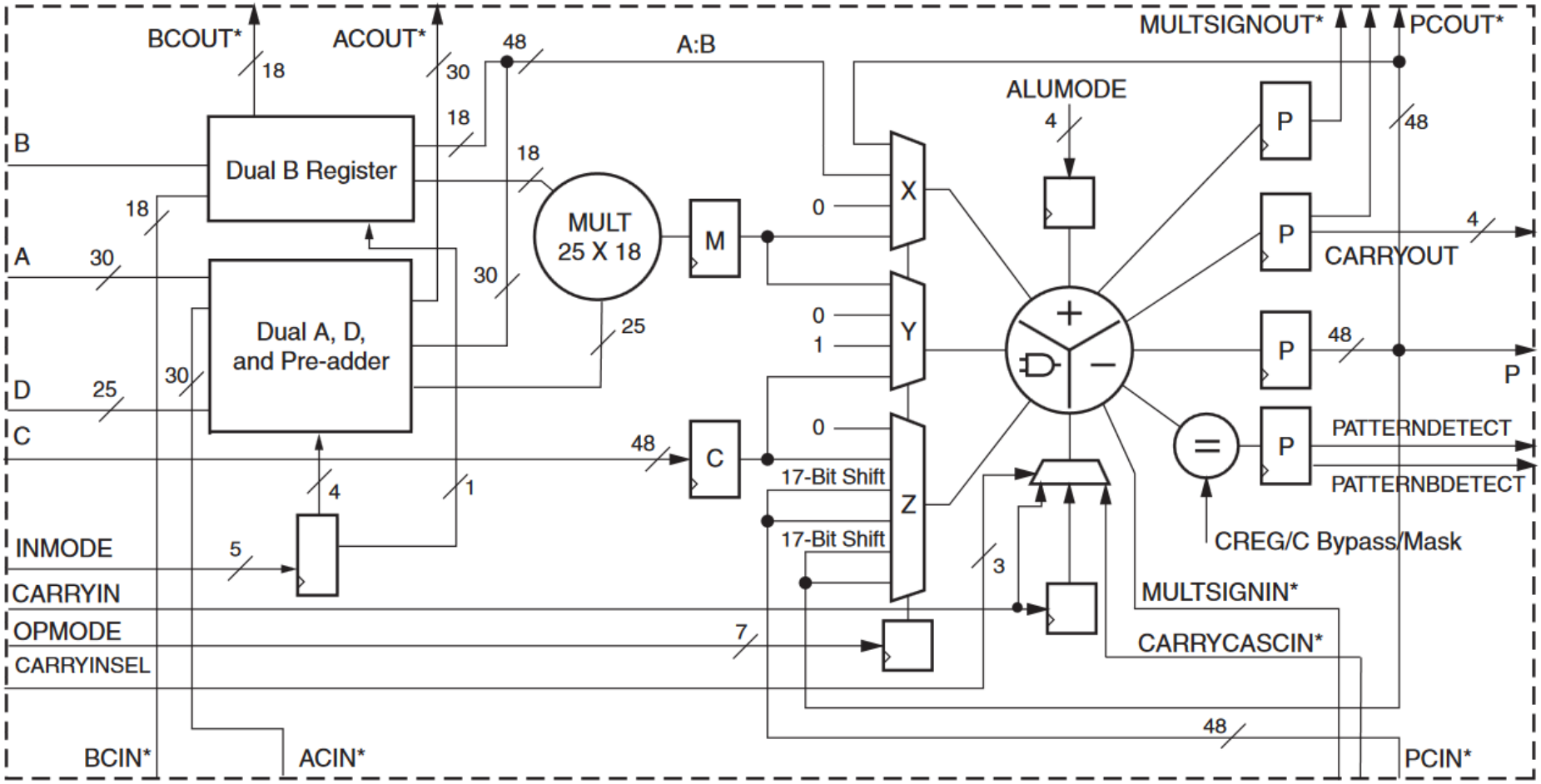
Xilinx 7 Series DSP Slice



Work on RSA signing speed

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

How fast can we sign?

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

How fast can we sign?

```
# one iteration of "Montgomery Ladder"

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

X1, X2 = T1, T2
```

How fast can we sign?

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

How fast can we sign?

```
# MMM
for ai in A:
     T  +=  ai * B  # multiplication

     mi :=  t0 * n'

     T  +=  mi * N  # reduction
     T  >>= 16

if T >= N: T -= N
```

How fast can we sign?

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

How fast can we sign?

- Total number of operations per ModExp:

  $S = L^3 / 512$

- Fmax is 460 MHz (-1 speed grade)

Maximum Frequency								
F_{MAX}	With all registers used	628.93	550.66	464.25	464.25	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

How fast can we sign?

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

	N = 4	N = 8	N = 16	N = 32	N = 64	N = 128	N = 256	N = 512
1024-bit key	219	438	877	1754	3509	7019	7019*	7019*
2048-bit key	27	54	109	219	438	877	1754	1754*
4096-bit key	3	6	13	27	54	109	219	438

How fast can we sign?

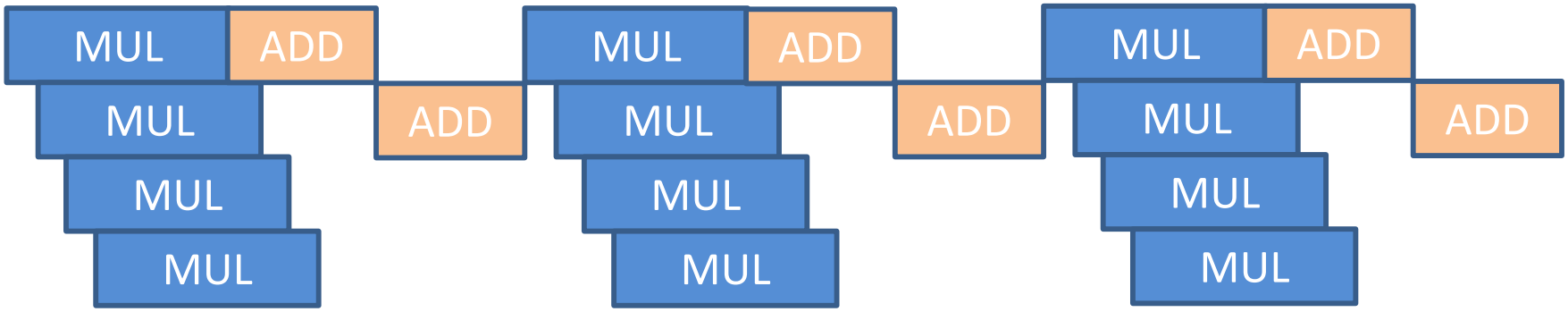
- I/O overhead not taken into account
- Difficult to completely avoid pipeline bubbles
- Operand broadcasting wastes time
- Last part of CRT (“Garner’s formula”) must be done after ModExp

How to improve performance?

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

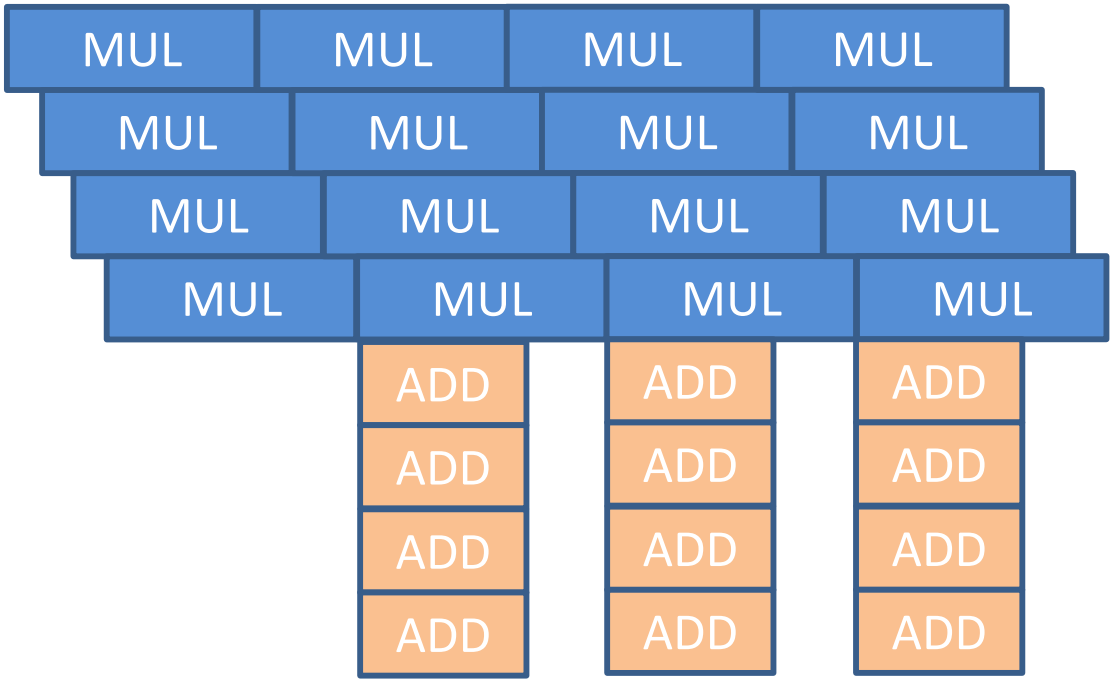
How to improve performance?

- Better scheduling of operations (before)



How to improve performance?

- Better scheduling of operations (after)

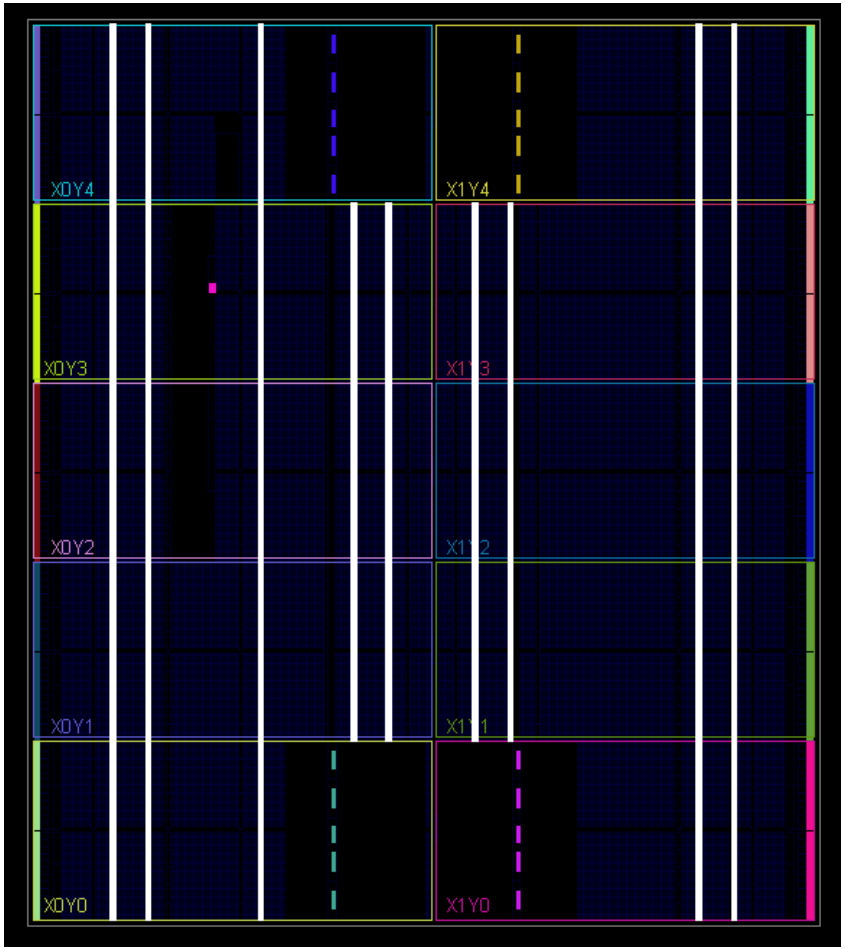


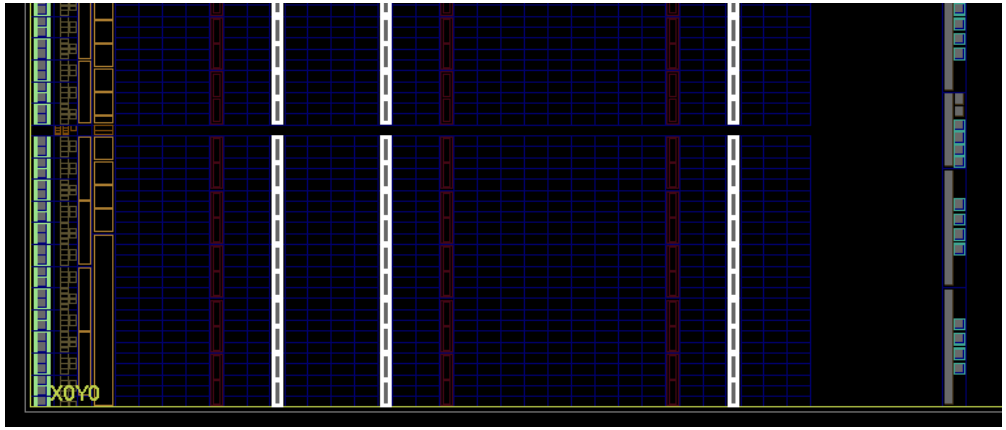
How to improve performance?

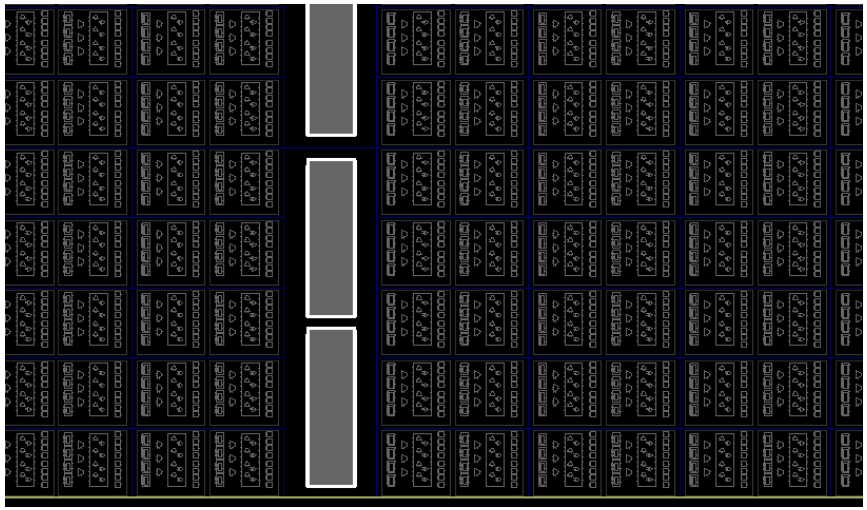
- Reasons for current core slowness:
  - Systolic architecture not suitable for field-programmable devices (multipliers arranged in narrow long columns, not rectangular array), increasing array size limits clock speed

How to improve performance?

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



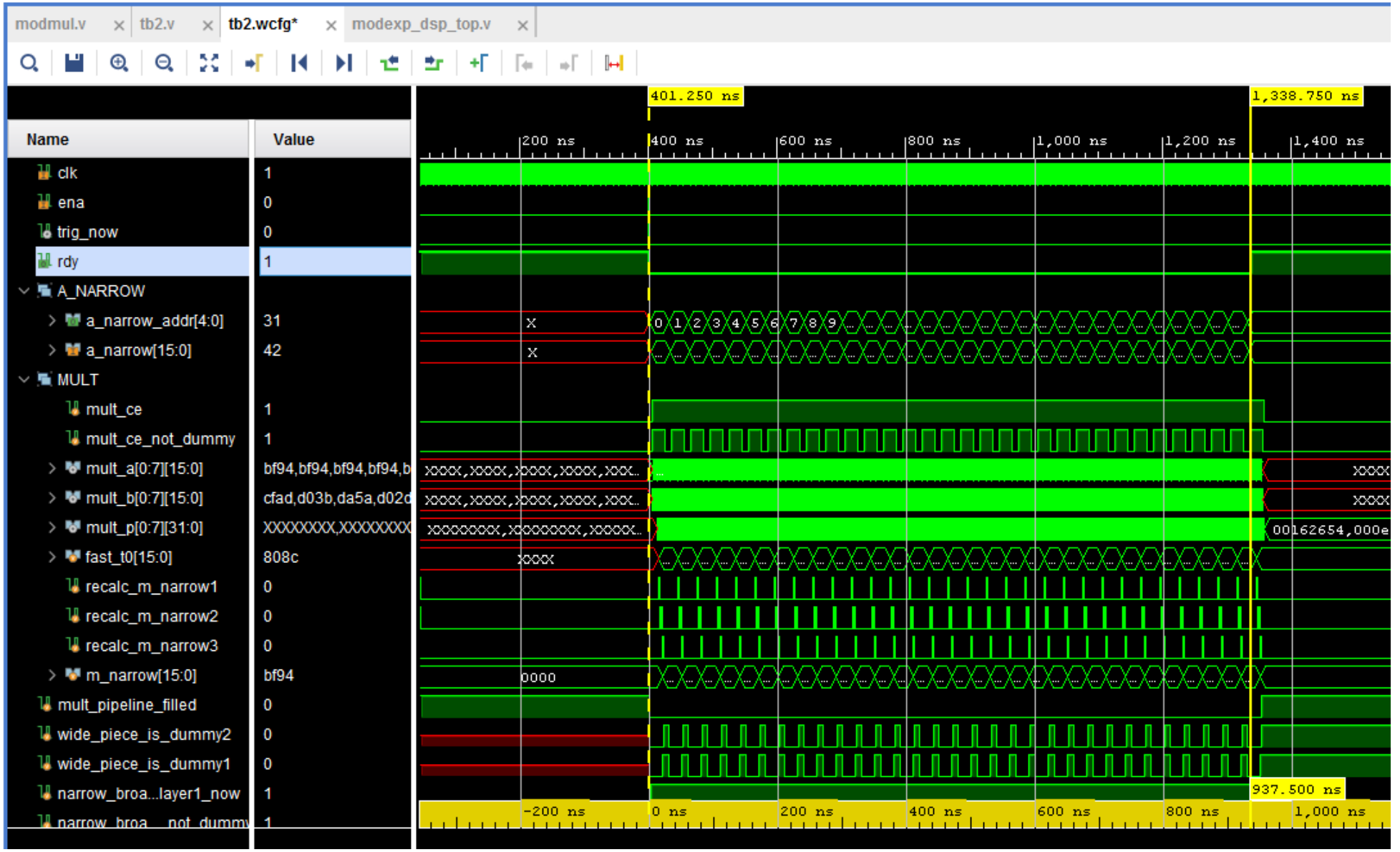




Progress so far

- 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



Progress so far

- PoC Verilog modular multiplier ~1 us cycle for 1024-bit modulus (in CRT mode)
- Expected speed is ~2000 signatures/second for 1024-bit keys
- 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?