

# Modular Inverse for Integers using Fast Constant Time GCD Algorithm and its Applications



Sanjay Deshpande\*, Santos Merino del Pozo<sup>†</sup>, Victor  
Mateu<sup>†</sup>, Marc Manzano<sup>†§</sup>, Najwa Aaraj<sup>†</sup>, and Jakub Szefer\*

\*Computer Architecture and Security Laboratory, Department of Electrical Engineering, Yale University, New Haven, CT, USA

<sup>†</sup>Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE

<sup>§</sup>Now with Sandbox@Alphabet, Mountain View, CA, USA

# Outline

- Motivation, Background, and Introduction
- Hardware Design – Fast GCD Inverse
- Applications
- Comparison with Related Work

# Motivation, Background, and Introduction

# Motivation for Efficient, Constant-time Modular Inverse

- Used in several public key cryptography schemes
- One of the most computationally intensive operations becoming the performance bottleneck

<b>Applications</b>	<b>Field Width (bits)</b>
Elliptic Curve Cryptography (ECC)	$\geq 255$
Supersingular Isogeny Key Encapsulation (SIKE)	434, 503, 610, 751
ElGamal Cryptosystem	$\geq 1024$
Digital Signature Algorithm (DSA)	$\geq 2048$
RSA	$\geq 2048$

# Modular Multiplicative Inverse for Integers

Definition: Consider ' $g$ ' is an integer in the ring of integers modulo ' $f$ ' and ' $g^{-1} \pmod{f}$ ' represents modular multiplicative inverse then

$$g \cdot g^{-1} \equiv 1 \pmod{f}$$

- Well known techniques to calculate modular multiplicative inverse
  - Binary Extended Euclidian Algorithm (BEEA) or also known as Greatest Common Divisor (GCD) Method
    - Based on multi-precision division steps
    - Fast and efficient
    - non-constant time
  - Fermat's Little Theorem
    - Based on modular exponentiation
    - Constant time
    - Widely used in cryptographic implementations both in Software and Hardware

# Modular Inverse using GCD Method – Non-Constant Time

- This simple Sage code is an adaptation of modular inverse calculation using GCD method from [KNU]
- Non-constant time
- Vulnerable to timing side-channel attacks


```
def mod_inv_gcd(g,f):  
    g1 = 1  
    g3 = g  
    f1 = 0  
    f3 = f  
    iter = 1  
  
    while(f3!=0):  
        q = int(g3 / f3)  
        t3 = int(g3 % f3)  
        t1 = int(g1 + q * f1)  
        g1 = f1; f1 = t1; g3 = f3; f3 = t3;  
        iter = -iter  
  
    if (g3 != 1):  
        return 0  
    if (iter <0):  
        inv = f-u1  
    else:  
        inv = g1  
  
    return inv
```

# Modular Inverse using GCD Method – Constant time (Fast GCD)

- Bernstein and Yang gave an efficient, constant-time software version of modular inverse using GCD method for integers and for polynomials in [DJY]
- This Sage code is derived from the algorithm provided in [DJY]

```
def iterations(d):  
    if d < 46:  
        rounds = (49*d+80)//17  /// does floor division  
    else:  
        rounds = (49*d+57)//17  
    return rounds
```

```
def divsteps2(n,delta,f,g):  
    v,r = 0,1  
    m = n  
    for n in range(m):  
        mask1 = (delta > 0) and (g&1 ==1)  
        mask0 = (delta <= 0) or (g&1 ==0)  
        delta,f,g,v,r = (mask0*delta + mask1*(-delta)),  
            (mask0*f + mask1*g), (mask0*g + mask1*(-f)),  
            (mask0*v + mask1*r), (mask0*r + mask1*(-v))  
        g0 = g&1  
        delta,g,r = 1+delta, (g+g0*f)/2, (r+g0*v)/2  
        g = ZZ(g) #zz() means Integer Ring  
    v_out = sign(f)*ZZ(v*2^(m-1))  
    return v_out
```

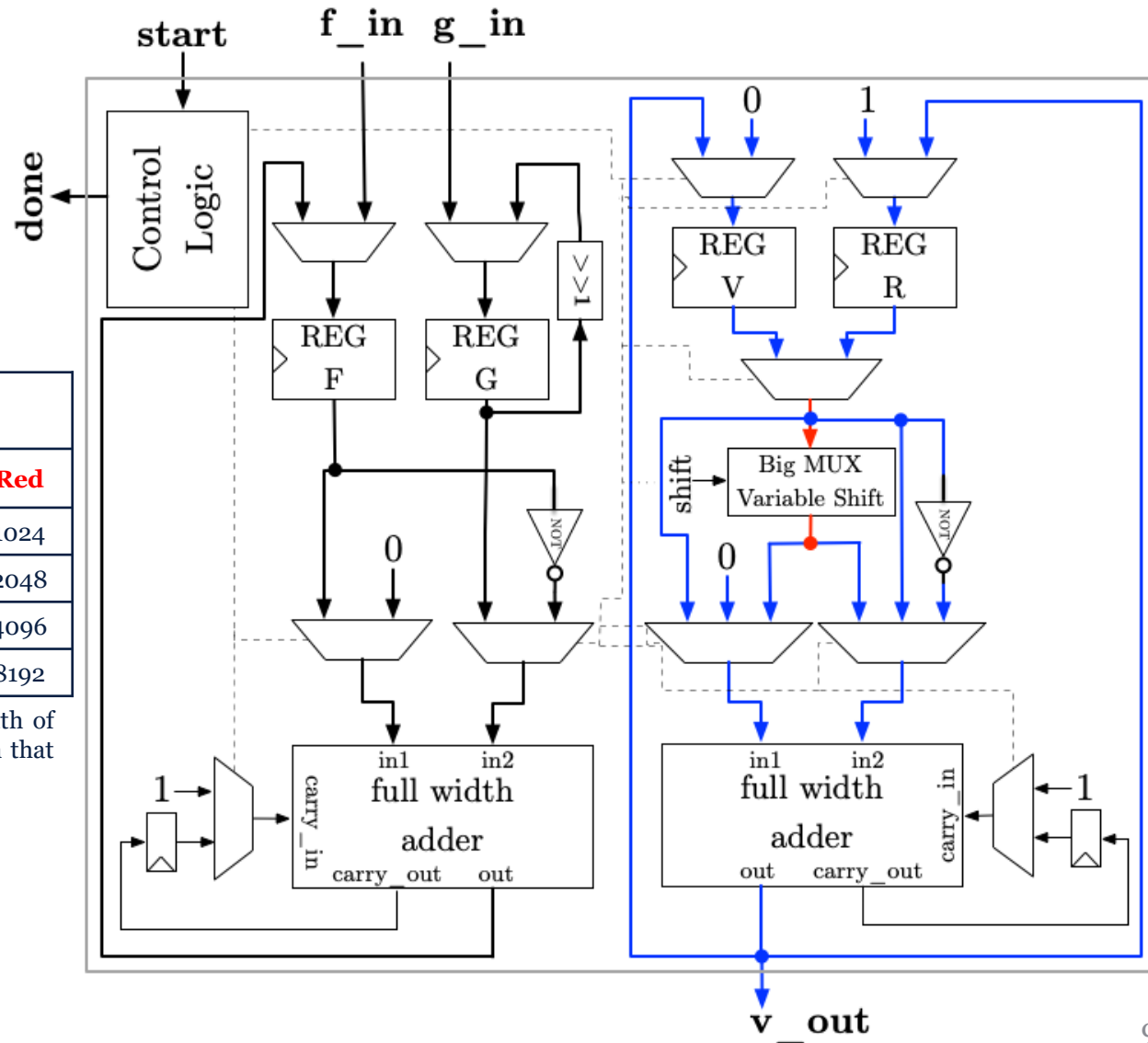


```
def fast_gcd(f,g):  
    d = max(f.nbits(),g.nbits())  
    m = iterations(d)  
    precomp = Integers(f)((f+1)/2)^(m-1)  
    v_out = divsteps2(m,1,f,g)  
    inverse = ZZ(v_out*precomp)  
    return inverse
```

# Fast GCD Inverse Hardware Design



# Full-Width Hardware Design



Application	Input/ Field Width (bits)	Bus Widths (bits)		
		Black	Blue	Red
ECC	255	256	739	1024
SIKE	434	435	1255	2048
ElGamal	1279	1280	3690	4096
DSA/RSA	2048	2049	5907	8192

The Black bus shown in the table refers to the width of registers REG F, REG G and all the related buses in that datapath.

# Full-Width Hardware Design: Results

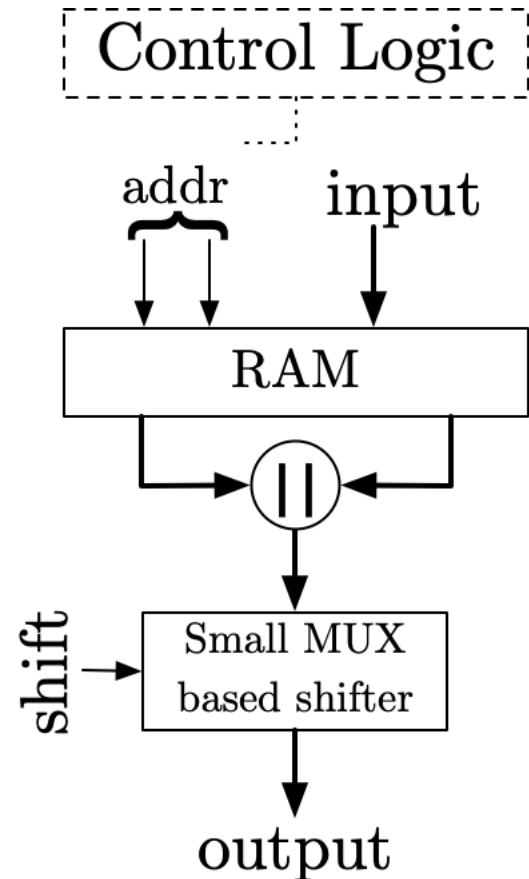
Target Device: Xilinx Zynq UltraScale+ XCZU7EG

Input Width	Freq (MHz)	Time (us)	Area			T.A . 10 <sup>3</sup>
			Slices	FFs	% usage	
255	207	40.9	1,847	6,704	6.41%	76
434	202	93.1	3,495	8,856	12.14%	326
1279	-	-	-	-	-	-
2048	-	-	-	-	-	-

- For the designs with Input widths 255 and 434, the full-width design is efficient and can be used in the respective applications
- For Input widths 1279 and 2048, the tool couldn't complete the implementation process due to big variable shifters and big registers
- **To address the issues due to large input width, a sequential design is implemented**

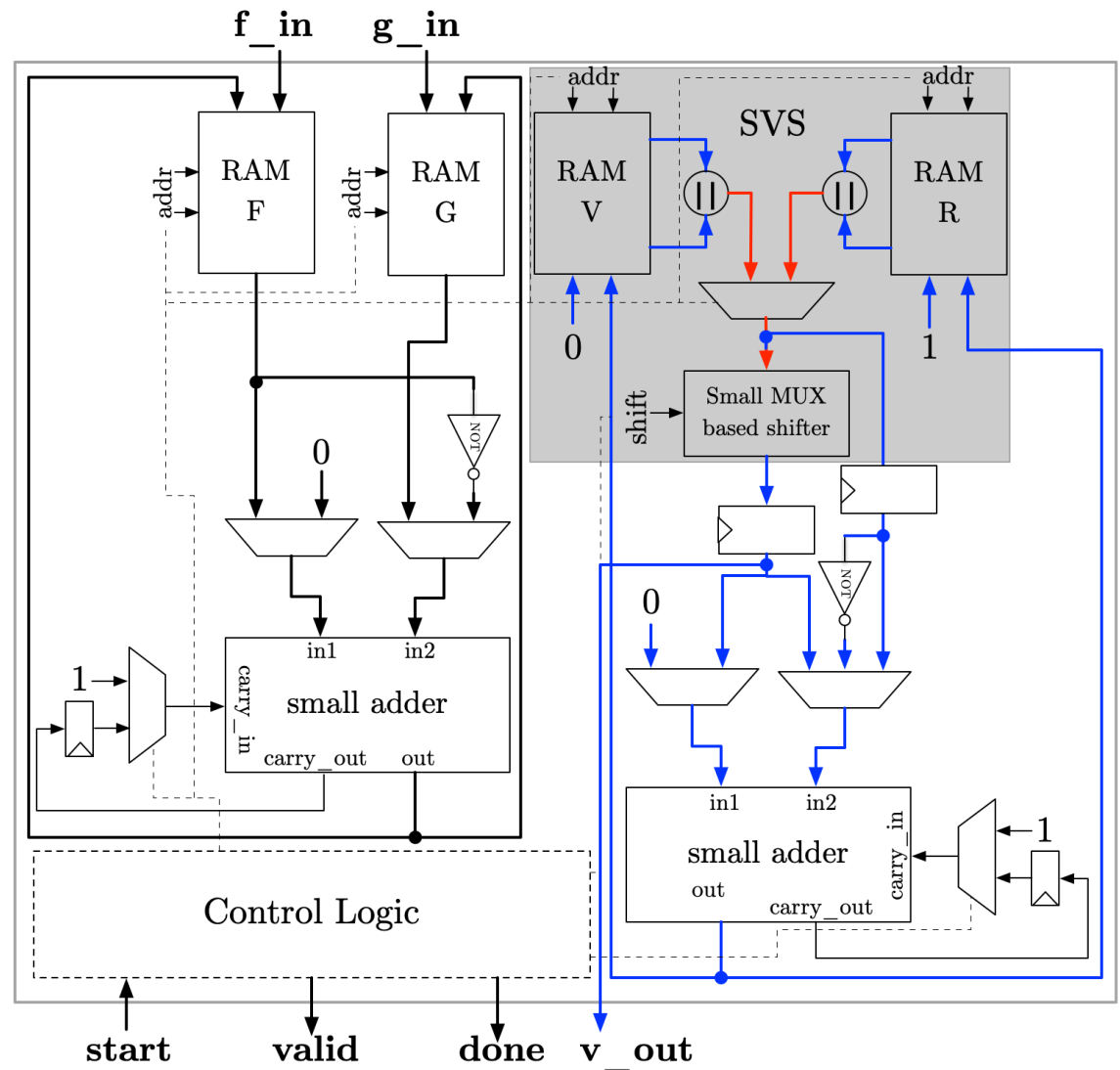
# Sequential Variable Shifter (SVS)

- SVS is derived from the Xilinx document from [XIL]
- Originally, SVS supported only rotate left operation for input divided into four parts
- The capability of SVS was extended to work for rotate left and rotate right for input divided into user defined number of blocks



# Sequential Hardware Design

Application	Input/ Field Width (bits)	Bus Widths (bits)		
		Black	Blue	Red
256-bit Architecture				
ECC	255	64	128	256
SIKE	434	73	128	256
ElGamal	1279	80	128	256
DSA, RSA	2048	86	128	256
128-bit Architecture				
ECC	255	32	64	128
SIKE	434	44	64	128
ElGamal	1279	43	64	128
DSA, RSA	2048	43	64	128
64-bit Architecture				
ECC	255	22	32	64
SIKE	434	22	32	64
ElGamal	1279	24	32	64
DSA, RSA	2048	22	32	64
32-bit Architecture				
ECC	255	11	16	32
SIKE	434	11	16	32
ElGamal	1279	11	16	32
DSA, RSA	2048	11	16	32
16-bit Architecture				
ECC	255	6	8	16
SIKE	434	6	8	16
ElGamal	1279	6	8	16
DSA, RSA	2048	6	8	16



The Black bus shown in the table refers to the width of each location of RAM F, RAM G and all the related buses in that datapath.

# Sequential Hardware Design - Results

Target Device: Xilinx Zynq UltraScale+ XCZU7EG

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
256-bit architecture						
255	190	46.6	915	803	22.50	43
434	189	106.4	920	851	22.50	98
1279	188	709.5	922	625	22.50	654
2048	182	1,680.0	933	932	22.50	1,567

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
128-bit architecture						
255	250	59.2	442	722	4.00	26
434	250	120.6	482	638	4.00	58
1279	238	992.1	531	650	4.00	527
2048	232	2,539.8	554	564	12.00	1,407

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
64-bit architecture						
255	303	68.4	234	434	2.00	16
434	294	187.9	256	453	2.00	48
1279	294	1,505.9	282	412	2.00	425
2048	263	4,399.2	291	390	6.00	1,280

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
32-bit architecture						
255	333	115.4	163	307	1.00	19
434	333	316.5	166	318	1.00	53
1279	333	2,613.2	235	320	1.00	614
2048	312	7,335.4	220	370	1.50	1,614

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
16-bit architecture						
255	346	213.8	115	241	0.50	25
434	339	607.6	120	265	0.50	73
1279	336	5,147.6	128	254	1.50	659
2048	336	13,591.7	136	275	1.50	1,848

# Sequential Hardware Design - Results

Target Device: Xilinx Zynq UltraScale+ XCZU7EG

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
256-bit architecture						
255	190	46.6	915	803	22.50	43
434	189	106.4	920	851	22.50	98
1279	188	709.5	922	625	22.50	654
2048	182	1,680.0	933	932	22.50	1,567

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
128-bit architecture						
255	250	59.2	442	722	4.00	26
434	250	120.6	482	638	4.00	58
1279	238	992.1	531	650	4.00	527
2048	232	2,539.8	554	564	12.00	1,407

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
64-bit architecture						
255	303	68.4	234	434	2.00	16
434	294	187.9	256	453	2.00	48
1279	294	1,505.9	282	412	2.00	425
2048	263	4,399.2	291	390	6.00	1,280

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
32-bit architecture						
255	333	115.4	163	307	1.00	19
434	333	316.5	166	318	1.00	53
1279	333	2,613.2	235	320	1.00	614
2048	312	7,335.4	220	370	1.50	1,614

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
16-bit architecture						
255	346	213.8	115	241	0.50	25
434	339	607.6	120	265	0.50	73
1279	336	5,147.6	128	254	1.50	659
2048	336	13,591.7	136	275	1.50	1,848

# Sequential Hardware Design - Results

Target Device: Xilinx Zynq UltraScale+ XCZU7EG

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
256-bit architecture						
255	190	46.6	915	803	22.50	43
434	189	106.4	920	851	22.50	98
1279	188	709.5	922	625	22.50	654
2048	182	1,680.0	933	932	22.50	1,567

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
128-bit architecture						
255	250	59.2	442	722	4.00	26
434	250	120.6	482	638	4.00	58
1279	238	992.1	531	650	4.00	527
2048	232	2,539.8	554	564	12.00	1,407

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
64-bit architecture						
255	303	68.4	234	434	2.00	16
434	294	187.9	256	453	2.00	48
1279	294	1,505.9	282	412	2.00	425
2048	263	4,399.2	291	390	6.00	1,280

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
32-bit architecture						
255	333	115.4	163	307	1.00	19
434	333	316.5	166	318	1.00	53
1279	333	2,613.2	235	320	1.00	614
2048	312	7,335.4	220	370	1.50	1,614

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
16-bit architecture						
255	346	213.8	115	241	0.50	25
434	339	607.6	120	265	0.50	73
1279	336	5,147.6	128	254	1.50	659
2048	336	13,591.7	136	275	1.50	1,848

# Sequential Hardware Design - Results

Target Device: Xilinx Zynq UltraScale+ XCZU7EG

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
256-bit architecture						
255	190	46.6	915	803	22.50	43
434	189	106.4	920	851	22.50	98
1279	188	709.5	922	625	22.50	654
2048	182	1,680.0	933	932	22.50	1,567

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
128-bit architecture						
255	250	59.2	442	722	4.00	26
434	250	120.6	482	638	4.00	58
1279	238	992.1	531	650	4.00	527
2048	232	2,539.8	554	564	12.00	1,407

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
64-bit architecture						
255	303	68.4	234	434	2.00	16
434	294	187.9	256	453	2.00	48
1279	294	1,505.9	282	412	2.00	425
2048	263	4,399.2	291	390	6.00	1,280

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
32-bit architecture						
255	333	115.4	163	307	1.00	19
434	333	316.5	166	318	1.00	53
1279	333	2,613.2	235	320	1.00	614
2048	312	7,335.4	220	370	1.50	1,614

Input Width	Freq. (MHz)	Time (T) (us)	Area (A)			T.A .10 <sup>3</sup>
			Slices	FFs	BRAM	
16-bit architecture						
255	346	213.8	115	241	0.50	25
434	339	607.6	120	265	0.50	73
1279	336	5,147.6	128	254	1.50	659
2048	336	13,591.7	136	275	1.50	1,848



# Comparison with Related Work – Modular Inverse

Design	Device	Frequency (MHz)	Clock Cycles ( $\cdot 10^3$ )	Time (ms)	Speed-up
<b>Input Width = 255-bits</b>					
Our Work	Zynq 7000	100	9	0.09	5.0
[MRR]	Zynq 7000	200	90	0.45	1.0
<b>Input Width = 434-bits</b>					
Our Work	Virtex 7	112	20	0.17	4.2
[PPJ]	Virtex 7	154	111	0.72	1.0
Our Work	Virtex 7	112	20	0.17	1.9
[PPJ]	Virtex 7	141	47	0.33	1.0
<b>Input Width = 2048-bits</b>					
Our Work	Zynq Ultrascale+	182	307	1.68	339.0
[JFR]	Stratix	200	113,880	569.40	1.0

\*Our Results from Sequential design with 256-bit architecture have been used for this comparison

[MRR] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast, small, and area-time efficient architectures for key-exchange on curve25519. In 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), pages 72–79, 2020.

[PPJ] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. Cryptology ePrint Archive, Report 2020/040, 2020.

[JFR] John Fry. RSA & public key cryptography in FPGAs. Technical report, Altera Corp., 2005.

# Comparison with Related Work – Modular Inverse

Design	Device	Frequency (MHz)	Clock Cycles ( $\cdot 10^3$ )	Time (ms)	Speed-up
<b>Input Width = 255-bits</b>					
Our Work	Zynq 7000	100	9	0.09	5.0
[MRR]	Zynq 7000	200	90	0.45	1.0
<b>Input Width = 434-bits</b>					
Our Work	Virtex 7	112	20	0.17	4.2
[PPJ]	Virtex 7	154	111	0.72	1.0
Our Work	Virtex 7	112	20	0.17	1.9
[PPJ]	Virtex 7	141	47	0.33	1.0
<b>Input Width = 2048-bits</b>					
Our Work	Zynq Ultrascale+	182	307	1.68	339.0
[JFR]	Stratix	200	113,880	569.40	1.0

\*Our Results from Sequential design with 256-bit architecture have been used for this comparison

[MRR] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast, small, and area-time efficient architectures for key-exchange on curve25519. In 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), pages 72–79, 2020.

[PPJ] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. Cryptology ePrint Archive, Report 2020/040, 2020.

[JFR] John Fry. RSA & public key cryptography in FPGAs. Technical report, Altera Corp., 2005.

# Comparison with Related Work – Modular Inverse

Design	Device	Frequency (MHz)	Clock Cycles ( $\cdot 10^3$ )	Time (ms)	Speed-up
<b>Input Width = 255-bits</b>					
Our Work	Zynq 7000	100	9	0.09	5.0
[MRR]	Zynq 7000	200	90	0.45	1.0
<b>Input Width = 434-bits</b>					
Our Work	Virtex 7	112	20	0.17	4.2
[PPJ]	Virtex 7	154	111	0.72	1.0
Our Work	Virtex 7	112	20	0.17	1.9
[PPJ]	Virtex 7	141	47	0.33	1.0
<b>Input Width = 2048-bits</b>					
Our Work	Zynq Ultrascale+	182	307	1.68	339.0
[JFR]	Stratix	200	113,880	569.40	1.0

\*Our Results from Sequential design with 256-bit architecture have been used for this comparison

[MRR] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast, small, and area-time efficient architectures for key-exchange on curve25519. In 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), pages 72–79, 2020.

[PPJ] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. Cryptology ePrint Archive, Report 2020/040, 2020.

[JFR] John Fry. RSA & public key cryptography in FPGAs. Technical report, Altera Corp., 2005.

# Comparison with Related Work – Modular Inverse

Design	Device	Frequency (MHz)	Clock Cycles ( $\cdot 10^3$ )	Time (ms)	Speed-up
<b>Input Width = 255-bits</b>					
Our Work	Zynq 7000	100	9	0.09	5.0
[MRR]	Zynq 7000	200	90	0.45	1.0
<b>Input Width = 434-bits</b>					
Our Work	Virtex 7	112	20	0.17	4.2
[PPJ]	Virtex 7	154	111	0.72	1.0
Our Work	Virtex 7	112	20	0.17	1.9
[PPJ]	Virtex 7	141	47	0.33	1.0
<b>Input Width = 2048-bits</b>					
Our Work	Zynq Ultrascale+	182	307	1.68	339.0
[JFR]	Stratix	200	113,880	569.40	1.0

\*Our Results from Sequential design with 256-bit architecture have been used for this comparison

[MRR] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast, small, and area-time efficient architectures for key-exchange on curve25519. In 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), pages 72–79, 2020.

[PPJ] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. Cryptology ePrint Archive, Report 2020/040, 2020.

[JFR] John Fry. RSA & public key cryptography in FPGAs. Technical report, Altera Corp., 2005.

# Applications

# Profiling the Applications

Algorithm	Operation where inverse is used	Ref. Design	Field Width	% Improv. in time Inverse	Expected % Improv. in time Overall
ECC	Coordinate Conversion	[MRR]	255	58%	5%
SIKE	Key Generation	[PPJ]	434	82%	4%
	Decapsulation				3%
ElGamal	Decryption	[LSS]	1279	NA	49%
DSA	Signing	[BNH]	2048	90%	49%
	Signature Verification				33%
RSA	Key Generation	[MKE]	2048	90%	3%

[BNH] B. Hanindhito, N. Ahmadi, H. Hogantara, A. I. Arrahmah, and T. Adiono. FPGA implementation of modified serial montgomery modular multiplication for 2048-bit RSA cryptosystems. In 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA), pages 113–118, 2015.

[MRR] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast, small, and area-time efficient architectures for key-exchange on curve25519. In 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), pages 72–79, 2020.

[PPJ] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. Cryptology ePrint Archive, Report 2020/040, 2020.

[MKE] Matthew Kenney. Implementing an RSA key generation and encryption/decryption functionalities on the Basys3 FPGA board. <https://github.com/kenneym/cs56/tree/master/design-source>, 2019.

[LSS] Loai A. Tawalbeh and Saadeh Sweidan. Hardware design and implementation of ElGamal public-key cryptography algorithm. Information Security Journal: A Global Perspective, 19(5):243–252, 2010.

# Profiling the Applications

Algorithm	Operation where inverse is used	Ref. Design	Field Width	% Improv. in time Inverse	Expected % Improv. in time Overall
ECC	Coordinate Conversion	[MRR]	255	58%	5%
SIKE	Key Generation	[PPJ]	434	82%	4%
	Decapsulation				3%
ElGamal	Decryption	[LSS]	1279	NA	49%
DSA	Signing	[BNH]	2048	90%	49%
	Signature Verification				33%
RSA	Key Generation	[MKE]	2048	90%	3%

[BNH] B. Hanindhito, N. Ahmadi, H. Hogantara, A. I. Arrahmah, and T. Adiono. FPGA implementation of modified serial montgomery modular multiplication for 2048-bit RSA cryptosystems. In 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA), pages 113–118, 2015.

[MRR] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast, small, and area-time efficient architectures for key-exchange on curve25519. In 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), pages 72–79, 2020.

[PPJ] Pedro Maat C. Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. Cryptology ePrint Archive, Report 2020/040, 2020.

[MKE] Matthew Kenney. Implementing an RSA key generation and encryption/decryption functionalities on the Basys3 FPGA board. <https://github.com/kenneym/cs56/tree/master/design-source>, 2019.

[LSS] Loai A. Tawalbeh and Saadeh Sweidan. Hardware design and implementation of ElGamal public-key cryptography algorithm. Information Security Journal: A Global Perspective, 19(5):243–252, 2010.

# ElGamal Decryption

- Designed two different light-weight hardware modules
  - D1 –Uses Fermat’s Little Theorem for inverse
  - D2 –Uses our Fast GCD Inverse
- Mersenne prime  $2^{1279}-1$  is used to define the integer ring
- For modular exponentiation, constant-time left to right, square and multiply always algorithm described in [CNT]
- Modular multiplication is designed using a combination of schoolbook multiplication and specific modular reduction

Algorithm – ElGamal Decryption

**Input:** Ciphertext 1 ( $c_1$ ), Ciphertext 2 ( $c_2$ ),  
SecretKey ( $x$ )

**Output:** Message ( $m$ )

Step 1: Compute  $s := c_1^x$

Step 2: Compute  $\ell := s^{-1}$

Step 3: Compute  $m := c_2 \cdot \ell$

	<b>Our Method D1</b>	<b>Our Method D2</b>
Step 1	ME	ME
Step 2	ME	FGCD
Step 3	MM	MM

ME: Modular Exponentiation

MM: Modular Multiplication

FGCD: Fast GCD Inverse



# ElGamal Decryption

- Designed two different light-weight hardware modules
  - D1 –Uses Fermat’s Little Theorem for inverse
  - D2 –Uses our Fast GCD Inverse
- Mersenne prime  $2^{1279}-1$  is used to define the integer ring
- For modular exponentiation, constant-time left to right, square and multiply always algorithm described in [CNT]
- Modular multiplication is designed using a combination of schoolbook multiplication and specific modular reduction

Algorithm – ElGamal Decryption

**Input:** Ciphertext 1 ( $c_1$ ), Ciphertext 2 ( $c_2$ ),  
SecretKey ( $x$ )

**Output:** Message ( $m$ )

Step 1: Compute  $s := c_1^x$

**Step 2: Compute  $\ell := s^{-1}$**

Step 3: Compute  $m := c_2 \cdot \ell$

	<b>Our Method D1</b>	<b>Our Method D2</b>
Step 1	ME	ME
<b>Step 2</b>	<b>ME</b>	<b>FGCD</b>
Step 3	MM	MM

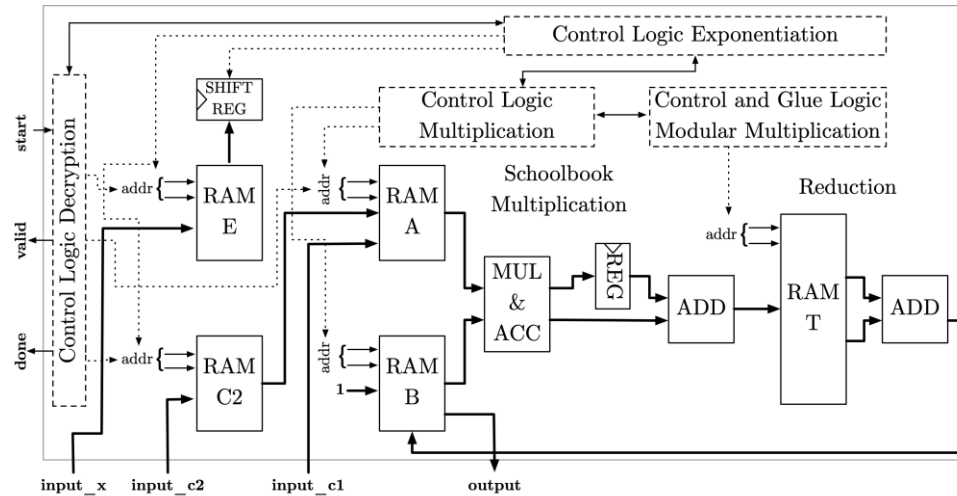
ME: Modular Exponentiation

MM: Modular Multiplication

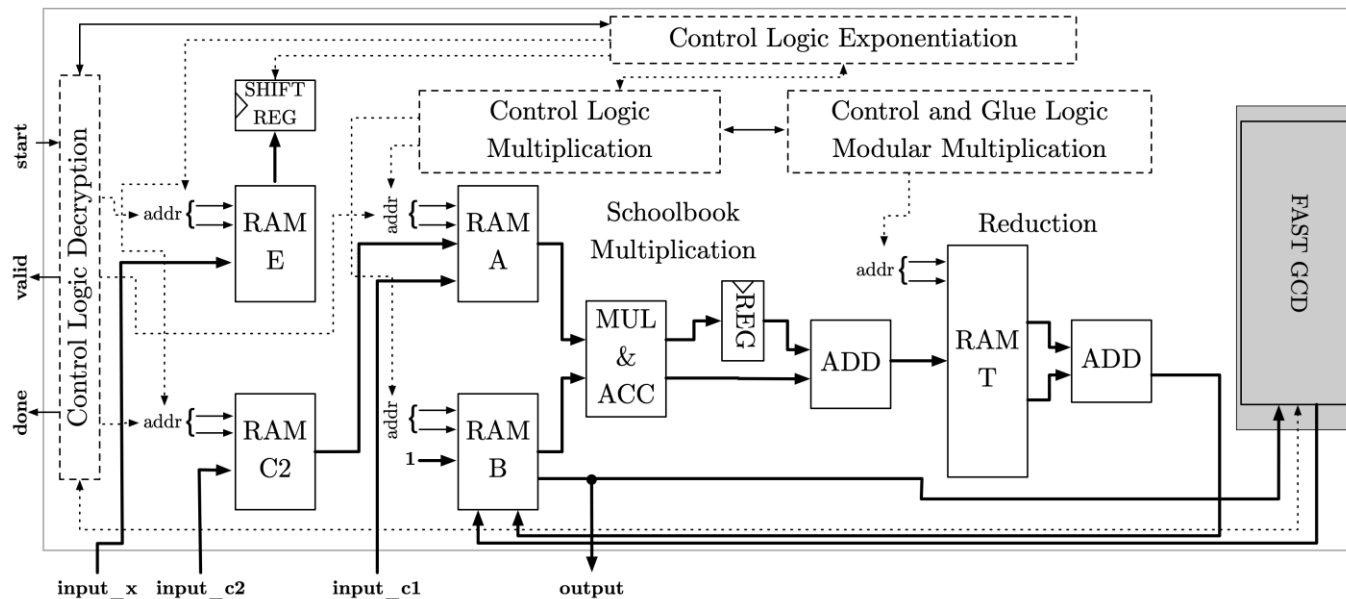
FGCD: Fast GCD Inverse

# ElGamal Decryption Hardware Design

D1



D2



# Comparison with Related Work – ElGamal Decryption

Design	Device	Input Width (bits)	Logic	Freq (MHz)	Time (T) (s)	T.A	
Our Work D1	Zynq UltraScale+	1279	Slices (A)- 84	300.00	0.11	9	
			DSP – 3				
			Total usage - 0.29%				
Our Work D2			Slices (A) – 229	300.00	0.06	14	
							DSP – 5
							Total usage - 0.79%
[LSS] V2	Virtex E	512	Slices (A) – 83,621	28.39	35.21	2,944,295	
[LSS] V4			Slices (A) – 83,442	27.58	36.24	3,023,938	

- For computing module inverse, our Fast GCD inverse in D2 is **10x** faster than the modular inverse in D1
- For decrypting a cipherttext, our overall D2 design is **1.84x** faster than our D1 design

# Future Work

- The compact Fast GCD inverse hardware module could be embedded as a hardware accelerator in a RISC V processor such as [PQS] and modular inverse functionality could be added to its instruction set architecture
- The ElGamal decryption implementation can be extended to a complete ElGamal cryptosystem and ElGamal signature scheme
- Faster implementation of ElGamal Cryptosystem can be then used with other applications such as electronic voting system, and homomorphic encryption operations.

Thank  You!

Title: Modular Inverse for Integers using Fast Constant Time GCD Algorithm and its Applications,

Authors: Sanjay Deshpande, Santos Merino del Pozo, Victor Mateu, Marc Manzano, Najwa Aaraj, and Jakub Szefer

Source code: <https://caslab.csl.yale.edu/code/fast-constant-time-gcd>