

ModExp 4096

Definitions & Algorithms

Çetin Kaya Koç
koc@cs.ucsb.edu

$$c = m^e \pmod{n}$$

- These slides describe the architecture, input and output definitions, and algorithms, and analysis of a 4096-bit full ModExp code
- The 4096-bit numbers are organized as sw bits, where s is the number of words and w is the word length, such that $sw = 4096$
- We will take $s = 128$ and $w = 32$ bits in this implementation
- Additionally the exponent length k is available, as $k \leq 4096$

ModExp 4096 Inputs and Output

- The **primary** input parameters are the basis m and the exponent e , each of which are s -word (sw -bits: 4096 bits) integers
- The **secondary** input parameters are n , $n0'$, r and t , such that n is the s -word (sw -bit: 4096-bit) modulus, $n0'$ is a 1-word (w -bit) integer, while r and t are s -word (sw -bit: 4096-bit) integers
- The output c is a s -word (sw -bit: 4096-bit) integer

Primary Input Parameter: m

- The code assumes that m is a 4096-bit **signed** integer:

$$m = (m_{4095}m_{4094} \cdots m_2m_1m_0)$$

such that $m_{4095} = 0$ for $m > 0$ due to 2s-complement representation, and organized as a s -word by w -bit number: $sw = 4096$

- The LSB is on the top right and the MSB (the sign bit) is on the bottom left corner, for example, for $s = 4$ and $w = 6$, we have

LSB: m_0

m_5	m_4	m_3	m_2	m_1	m_0
m_{11}	m_{10}	m_9	m_8	m_7	m_6
m_{17}	m_{16}	m_{15}	m_{14}	m_{13}	m_{12}
m_{23}	m_{22}	m_{21}	m_{20}	m_{19}	m_{18}

MSB: $m_{23} = 0$ for $m > 0$

- If $m > 0$ then $m_{23} = 0$; LSB: m_0 and MSB: m_{23} (sign bit)

Primary Input Parameter: e

- The exponent e is k -bit **unsigned** binary number for $k \leq 4096$

$$e = (e_{k-1}e_{k-2} \cdots e_2e_1e_0)$$

- Many practical exponents are fewer than 4096 bits; we assume e is k bits for $k \leq 4096$
- The LSB is on the top right and the MSB (the sign bit) is on the bottom left corner, for example, for $s = 4$ and $w = 6$, we have

LSB: e_0

e_5	e_4	e_3	e_2	e_1	e_0
e_{11}	e_{10}	e_9	e_8	e_7	e_6
e_{17}	e_{16}	e_{15}	e_{14}	e_{13}	e_{12}
e_{23}	e_{22}	e_{21}	e_{20}	e_{19}	e_{18}

$$e_{23} = e_{22} = \dots = e_k = 0$$

- The MSBs $e_{23}, e_{22}, \dots, e_k$ are zero

Secondary Input Parameter: n

- The code assumes that n is a 4096-bit **signed** binary number:

$$n = (n_{4095} n_{4094} \cdots n_2 n_1 n_0)$$

such that $n_{4095} = 0$ for $n > 0$ due to 2s-complement representation

- We always have $n_{4095} = 0$ ($n > 0$) and $n_0 = 1$ (odd modulus)
- The LSB is on the top right and the MSB (the sign bit) is on the bottom left corner, for example, for $s = 4$ and $w = 6$, we have

LSB: $n_0 = 1$

n_5	n_4	n_3	n_2	n_1	n_0
n_{11}	n_{10}	n_9	n_8	n_7	n_6
n_{17}	n_{16}	n_{15}	n_{14}	n_{13}	n_{12}
n_{23}	n_{22}	n_{21}	n_{20}	n_{19}	n_{18}

MSB: $n_{23} = 0$

- Note that $n_{23} = 0$ and $n_0 = 1$

Secondary Input Parameters: $n0'$, r and t

- The parameters $n0'$, r and t are derived from the modulus n :

$$n0' = -n^{-1} \pmod{2^w}$$

$$r = 2^{sw} \pmod{n}$$

$$t = 2^{2sw} \pmod{n}$$

- Since it is often the case that the modulus is fixed, these parameters are computed at once and saved in the memory
- The ModExp code assumes that these parameters are available at the same time with the modulus n
- We will later give code for computing these parameters outside the ModExp code

Secondary Input Parameters: $n0'$, r and t

- $n0'$ is a 1-word (w -bit) unsigned integer: $n0' = (u_{w-1} \dots u_1 u_0)$
- r and t are s -word (sw -bit: 4096-bit) signed integers

r

r_5	r_4	r_3	r_2	r_1	r_0
r_{11}	r_{10}	r_9	r_8	r_7	r_6
r_{17}	r_{16}	r_{15}	r_{14}	r_{13}	r_{12}
r_{23}	r_{22}	r_{21}	r_{20}	r_{19}	r_{18}

t

t_5	t_4	t_3	t_2	t_1	t_0
t_{11}	t_{10}	t_9	t_8	t_7	t_6
t_{17}	t_{16}	t_{15}	t_{14}	t_{13}	t_{12}
t_{23}	t_{22}	t_{21}	t_{20}	t_{19}	t_{18}

$n0'$

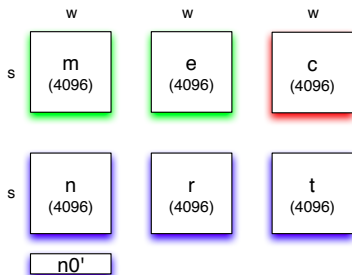
u_5	u_4	u_3	u_2	u_1	u_0
-------	-------	-------	-------	-------	-------

Target Device & Properties

- The target device is Altera Cyclone II EP2C50
- This device has 129 4k memory blocks
- Each 4k block can be addressed at different configurations: $4k \times 1$, $2k \times 2$, up to 128×32
- The memory supports byte writes with port data width of 1, 2, 4, 8, 16, 32, 36 bits
- The device has more than sufficient memory of the input and output data, as well as for temporary values
- This allow us to implement several different optimizations on the exponentiation level: 1-bit, 2-bit and 4-bit window sizes
- Some configuration decisions will be made after the coding of the MonPro block is completed

Input and Output

- The device assumes the primary inputs m and e are written into selected memory blocks (GREEN)
- The device also assumes the secondary input values n , $n0'$, r , and t are already written into selected memory blocks (BLUE)
- It is often the case that the primary values will change after each computation, while the secondary values will remain in place
- The device computes c and writes into a memory block (RED)



The ModExp Algorithm

- The algorithm of choice is the Montgomery-transformed exponentiation using s -word integers with a word size of w bits and the Montgomery constant as $r = 2^{sw}$
- We will describe the 1-bit (binary) and 2-bit (quaternary) exponentiation, however, we are experimenting with the 4-bit (hex) method, and make our final choice at the last phase of the project
- At the multiplication level, we are using the MonPro CIOS algorithm, whose details are given later on
- Primary Inputs: m and e
- Secondary Inputs: n , $n0'$, r , and t
- Output: c
- Functions: MonPro

The 1-bit ModExp Algorithm

Find the index $k < 4095$ of the leftmost 1 in e

$\bar{m} = \text{MonPro}(m, t)$

$\bar{c} = r$

for $i = k - 1$ down to 0

$\bar{c} = \text{MonPro}(\bar{c}, \bar{c})$

 if $e_i = 1$ then $\bar{c} = \text{MonPro}(\bar{c}, \bar{m})$

$c = \text{MonPro}(\bar{c}, 1)$

The 2-bit ModExp Algorithm

Express e in 2-bit blocks $e = (E_{2047} \cdots E_1 E_0)$ with $E_j \in \{0, 1, 2, 3\}$

Find the index $k < 2047$ of the leftmost nonzero in e

$\bar{m}1 = \text{MonPro}(m, t)$

$\bar{m}2 = \text{MonPro}(\bar{m}1, \bar{m}1)$

$\bar{m}3 = \text{MonPro}(\bar{m}2, \bar{m}1)$

$\bar{c} = \text{MonPro}(1, t)$

for $i = k$ down to 0

$\bar{c} = \text{MonPro}(\bar{c}, \bar{c})$

$\bar{c} = \text{MonPro}(\bar{c}, \bar{c})$

 if $E_i = 1$ then $\bar{c} = \text{MonPro}(\bar{c}, \bar{m}1)$

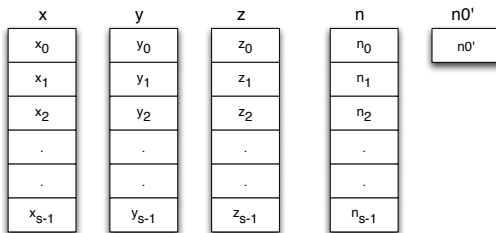
 else if $E_i = 2$ then $\bar{c} = \text{MonPro}(\bar{c}, \bar{m}2)$

 else if $E_i = 3$ then $\bar{c} = \text{MonPro}(\bar{c}, \bar{m}3)$

$c = \text{MonPro}(\bar{c}, 1)$

The MonPro Algorithm

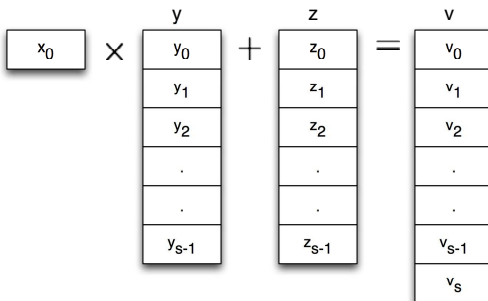
- The MonPro algorithm takes two inputs: x and y , and computes the output z such that each variable holds an s -word (sw -bit: 4096-bit) signed integer: $x = (x_{s-1}x_{s-2} \cdots x_1x_0)$, $y = (y_{s-1}y_{s-2} \cdots y_1y_0)$, and $z = (z_{s-1}z_{s-2} \cdots z_1z_0)$ with x_i, y_i, z_i as w -bit for $s-1 \geq i \geq 0$, and the initial value of z is zero
- The secondary inputs n , $n0'$, r and t are also available; we particularly need $n0'$ and n , which are 1-word and s -word integers



The MonPro Algorithm

- 1: We take the LSW of x , namely x_0 and multiply by the s -word y , and add it to the s -word partial product z (which is now all zero) to obtain the $(s + 1)$ -word temporary result v as

$$v = x_0 \cdot \sum_{i=0}^{s-1} y_i 2^{wi} + \sum_{i=0}^{s-1} z_i 2^{wi}$$



The MonPro Algorithm

- 1a: The computation in Step 1 is accomplished using a Multiply-Add block that multiplies two 1-word numbers (x_0 and y_0), adds the previous higher word (C_0), and adds another 1-word number (z_0), producing a 2-word number (C_1, S_0); the lower word (S_0) is assigned to (v_0), while the higher word (C_1) is kept for the next Multiply-Add step, as follows:

$$(C_1, S_0) = x_0 \cdot y_0 + C_0 + z_0$$

$$v_0 = S_0$$

$$(C_2, S_1) = x_0 \cdot y_1 + C_1 + z_1$$

$$v_1 = S_1$$

$$(C_3, S_2) = x_0 \cdot y_2 + C_2 + z_2$$

$$v_2 = S_2$$

...

such that the initial value $C_0 = 0$

The MonPro Algorithm

- 2: Then, we take the LSW of v , namely v_0 and multiply by the 1-word n_0' modulo 2^w and obtain the 1-word integer m as

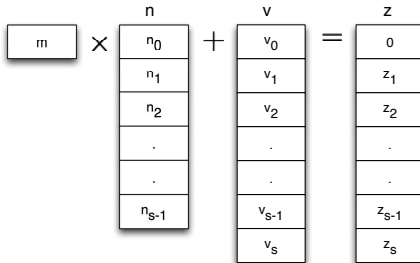
$$m = n_0' \cdot v_0 \pmod{2^w}$$

$$\boxed{v_0} \times \boxed{n_0'} = \boxed{m} \pmod{2^w}$$

The MonPro Algorithm

- 3: Then, we take the 1-word m and multiply by the s -word n , and add it to the $(s + 1)$ -word temporary value v to obtain the new partial product which is $(s + 1)$ -word

$$z = m \cdot \sum_{i=0}^{s-1} n_i 2^{wi} + \sum_{i=0}^s v_i 2^{wi}$$



The MonPro Algorithm

3a: The computation in Step 3 is accomplished using a Multiply-Add block that multiplies two 1-word numbers (m and n_0), adds the previous higher word (C_0), adds another 1-word number (v_0), producing a 2-word number (C_1, S_0); assigning S_0 to z_0 , while keeping the higher word (C_1) for the next Multiply-Add step, as follows:

$$(C_1, S_0) = m \cdot n_0 + C_0 + v_0$$

$$z_0 = S_0$$

$$(C_2, S_1) = m \cdot n_1 + C_1 + v_1$$

$$z_1 = S_1$$

$$(C_3, S_2) = m \cdot n_2 + C_2 + v_2$$

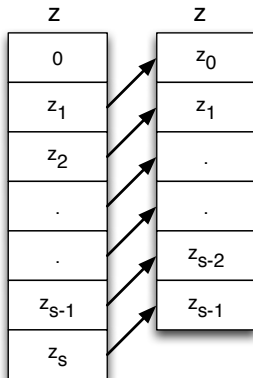
$$z_2 = S_2$$

...

such that the initial value $C_0 = 0$

The MonPro Algorithm

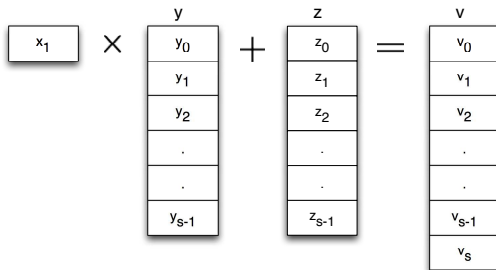
- 4: The resulting partial product z has its LSW as zero, due to the Montgomery property, and therefore, we shift up z to obtain the new s -word partial product



The MonPro Algorithm

- 5: In the next step the next word of x , namely x_1 , is taken and multiplied with the s -word y , and added to the partial product z to obtain the new temporary result v as

$$v = x_1 \cdot \sum_{i=0}^{s-1} y_i 2^{wi} + \sum_{i=0}^{s-1} z_i 2^{wi}$$



The MonPro Algorithm

6: This is followed up by computing the new m value

$$m = n0' \cdot v_0 \pmod{2^w}$$

and then the multiplication of m by n , and then the addition of the result to v to obtain the new $(s + 1)$ -word partial product z

$$z = m \cdot \sum_{i=0}^{s-1} n_i 2^{wi} + \sum_{i=0}^s v_i 2^{wi}$$

and finally shifting up z by one word (since $z_0 = 0$) to obtain the new s -word z value

7: Proceeding in this way, we multiply all x_i values by the multiplicand y and reduce it modulo n , for $i = 0, 1, 2, \dots, s - 1$

Resources and Components

- The ModExp requires implementation of several blocks
- A finite state machine to scan the bits of the exponent, 1-bit, 2-bit and 4-bit at a time
- A w -by- w bit integer multiplier producing $2w$ -bit products
- A w -by- w bit integer adder producing $(w + 1)$ -bit sums
- A finite state machine to control the flow of the MonPro algorithm from $i = 0$ to $i = s - 1$
- Several registers for the primary and secondary inputs, for the output, and for the temporary values

- Our analysis shows that $w = 32$ and $s = 128$ values will produce an implementation that will satisfy the specified requirements
- The target device Altera Cyclone II EP2C50 has 129 4k memory blocks which can be configured as 128×32
- At this configuration these memory blocks are not accessible in the true dual-port mode, however, we will not need that
- It is therefore assumed that the initiating processor uploads the 128-word primary inputs m and e , and 128-word secondary inputs n , r , and t , and 1-word secondary input $n0'$ to selected memory blocks
- The ModExp code will run and produce the 128-word output c in a selected memory block

- The ModExp code will not destroy the input values and all of them can be reused for subsequent executions
- It is generally the case the primary inputs m and e will be refreshed by the initiating processor, leaving the secondary input values n , r , and t , and $n0'$ intact for subsequent execution(s)