Project 3 – RSA Encoder – Design for Performance.

Objective

One utilization of an FPGA is for hardware acceleration. Most applications that can be performed on an FPGA can be done using a micro controller for a fraction of the cost, but there are applications that require the programmable processing power of a local CPU, as well as the computational power of hardware. There are even groups experimenting with using FPGAs for on the fly programmable hardware acceleration blocks[i] called reconfigurable hardware acceleration. Project three is used to illustrate an application of hardware acceleration for internet security.

Summary

This project starts with a simple hardware solution to an encryption problem that uses 16 bits to simulate that is easy to debug. Your boss's work is done, and he passes the project off to you. The problem is that the project will becomes unwieldy to implement as you scale up the size of the encryption engine. This means that coding styles, architecture, and careful planning are required to complete this project. You will need to add a number of components to complete all the required attributes of this project. There will need to be at least two clock domains. Having multiple clock domains allows one to optimize the clock for different portions of the circuit. There will also need to be at least one math function used from the IP Cores library. You will find that these are also useful.

As you progress from the starting point of a 16 bit encryption system, to 32 bits, and finally to 64 bits, you will find that different tools are useful for different levels of circuit complexity. Finally, you will add a Picoblaze software interface that will allow you communicate with the FGPA from a USB serial port and enter messages to be encrypted or decrypted from a remote computer. It will be useful to work way through the different levels to experiment with the different scales of complexity, before moving on to the larger size circuits, unless that is, you are looking forward to watching your computer repeatedly place and route your circuit for long periods of time. You will also want to simulate your state machines using behavior models before implementing the structural models.

Your deliverable is a working 64bit encryption processor that interfaces with your computer due in two weeks. Extra credit will be given for a working solution that uses a minimum area. Extra credit will also be given to a solution that can correctly decrypt a single line or file from the terminal the fastest. You are not bound by the initial architecture given, or the original code that is given, but this is a good place to start because it is so simple. Be careful about going big, or adding too much complexity initially. Finally, extra credit will be given for higher bit combinations.

Encryption Primer

The problem of communicating securely between two individuals has been an issue as long as man has been communicating. Generally, secure communications have been done through secrecy and obfuscation. More recently messages have been encrypted using secret keys.

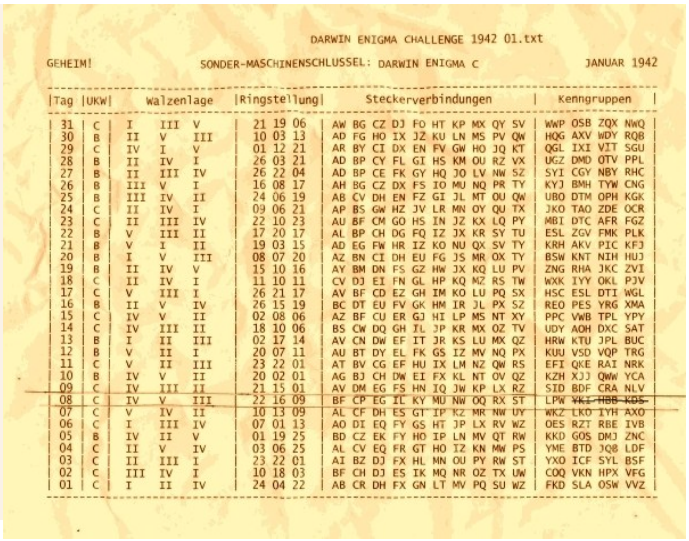
Table 1. Simple Encryption Example Encryption Key: November 5th, 2012

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6	7	8	9
5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4

For a simple example, say you and a friend want to pass secret notes to each other. You come up with a system where the secret key is date that a letter is written. The day of the month is used as an index to shift the letter of the alphabet over (see Table 1). This works well enough until you tell someone what the code is, or someone knows that it is a simple replacement system. An analysis of single letters in a simple replacement system, will clearly identify I's and A's, two letter words all need vowels, so the remainder of the vowels can easily be identified, and it becomes pretty easy to crack the code. You could relatively easy write a program to crack this code.

It is also a drag to have to write letters this way, encoding one letter at a time. The Enigma machine was meant to solve the simplicity issue by a complex set of mechanical, electrical, and secret key code books. This machine was developed in the thirties and was used heavily by the Germans in WWII through the end of the war to encrypt all their wartime transmissions. Good thing for the Allies that the Polish Intelligence had broken the German code five years before the outbreak of WWII, and five weeks before they were invaded by the Germans, they handed the keys to their decipher technology to the British, and formed their government exile in Britain.

Illustration 1: Enigma Machine



DARWIN ENIGMA CHALLENGE 1942 01.txt

GEHEIM! SONDER-MASCHINENSCHLUSSEL: DARWIN ENIGMA C JANUAR 1942

Tag	UKW	Walzenlage	Ringstellung	Steckerverbindungen	Kenngruppen
31	C	I III V	21 19 06	AW BG CZ DJ FO HT KP MX QY SV	WWP OSB ZQX NWQ
30	B	II V III	10 03 13	AD FG HO IX JZ KU LN MS PV QW	HQG AXV WDY RQB
29	C	IV I V	01 12 21	AR BY CI DX EN FV GW HO JQ KT	QGL IXI YJT SGU
28	B	II IV I	26 03 21	AD BP CY FL GI HS KM OU RZ VX	UGZ DMD OTV PPL
27	B	II III IV	26 22 04	AD BP CE FK GY HQ JO LV NW SZ	SYI CGY NBY RHC
26	B	III V I	16 08 17	AH BG CZ DX FS IO MU NQ PR TY	KYJ BMH TYW CNG
25	B	III IV II	24 06 19	AB CV DH EN FZ GI JL MT OU QW	UBQ DTM OPH KGK
24	C	II IV I	09 06 21	AP BS GW HZ JV LR MN OY QU TX	JKO TAO ZDE OCR
23	C	II III IV	22 10 23	AU BF CM GO HS IN JZ KX LQ PY	MBI DTC AFR FGZ
22	B	V III II	17 20 17	AL BP CH DG FQ IZ JX KR SY TU	ESL ZGV FMK PLK
21	B	V I II	19 03 15	AD EG FW HR IZ KO NU QX SV TY	KRH AKV PIC KFJ
20	B	I V III	08 07 20	AZ BN CI DH EU FG JS MR OX TY	BSW KNT NIH HUJ
19	B	II IV V	15 10 16	AY BM DN FS GZ HW JX KQ LU PV	ZNG RHA JKC ZVI
18	C	II IV I	11 10 11	CV DJ EI FN GL HP KQ MZ RS TW	WXK IYY OKL PJV
17	C	V III I	26 21 17	AV BF CD EZ GH IM KO LU PQ SX	HSC ESL DTI WGL
16	B	II V IV	26 15 19	BC DT EU FV GK HM IR JL PX SZ	REO PES YRG XMA
15	C	IV V II	02 08 06	AZ BF CU ER GJ HI LP MS NT XY	PPC VWB TPL YPY
14	C	IV III II	18 10 06	BS CW DQ GH JL JP KR MX OZ TV	UDY AOH DXC SAT
13	B	I II III	02 17 14	AV CN DW EF IT JR KS LU MX QZ	HRW KTU JPL BUC
12	B	V II I	20 07 11	AU BT DY EL FK GS IZ MV NQ PX	KUU VSD VQP TRG
11	C	V II III	23 22 01	AT BV CG EF HU IX LM NZ QW RS	EFI QKE RAI NRK
10	B	IV V II	20 02 01	AG BJ CH DW EI FX KL NT OV QZ	KZH XJJ QWW YCA
09	C	IV III II	21 15 01	AV DM EG FS HN IQ JW KP LX RZ	STD BDF CRA NLV
08	C	IV V III	22 16 09	BF CP EG IL KY MU NW OQ RX ST	LPW ~~XKJ HBB XDS~~
07	C	V IV II	10 13 09	AL CF DH ES GT IP KZ MR NW UY	WKZ LKO IYH AXO
06	C	I III IV	07 01 13	AO DI EQ FY GS HT JP LX RV WZ	OES RZT RBE IVB
05	B	IV II V	01 19 25	BD CZ EK FY HO IP LN MV QT RW	KKD GOS DMJ ZNC
04	C	II V IV	03 06 25	AL CV EQ FR GT HO IZ KN MW PS	YME BTD JQB LDF
03	C	II III I	23 22 01	AI BZ DJ FX HL MN OU PY RW ST	YXO ICF SYL BSF
02	C	III IV I	10 18 03	BF CH DJ ES IK MQ NR OZ TX UW	COQ VKN HPX VFG
01	C	I II IV	24 04 22	AB CR DH FX GN LT MV PQ SU WZ	FKD SLA OSW VVZ

Illustration 2: Enigma Code book page

The idea of using a secret key for encryption is still a concept that is used by the US Government today using ISA encryption. This encryption methodology is similar to the Enigma Machine in that each portion of the code goes through a process of rearranging the incoming message in 128 bit blocks. It would be extremely difficult to break, but there are always innovations, and like other secret key systems, you need to share your secret key with somebody else.

It is difficult to share a secret key on the internet – millions of online transactions occur on a daily basis between people that have no ability to exchange a secret code, but they still need to communicate in privacy. The solution to this problem is with the use of public and private keys in RSA encoding. A person can give the world the public key to encrypt a message, but without the private key, there is no way to decode the message. This has lead to the field of public key cryptography.

RSA Encryption Methodology

RSA encryption is algorithmically simple; it relies on the computational complexity of factoring a large number in order to prevent decryption. It works like this, when two people want to exchange a plain text message (M) , the receiver of the private message gives two numbers to the sender of the message. The two numbers of the public key are the modulus (n) of the message and the encryption exponent (e). The receiver is the only person that knows how to unscramble the message by using the private key that is the secret decoding exponent (d) and the message modulus. Before encryption, the plain text message must be padded. This padded message (m) breaks the message into message blocks that are each smaller than the modulus and ensured message complexity.

Table 2. RSA Encryption Transaction

Receiver	**Transaction**	**Sender**
Requests encrypted information	Public-key (n , e) ==>	
		Pads M => (m_1, m_2,... m_i) (e.g. n = 32bits, then M_1 = 24bits (three ASCII characters), then $m_1 = M_1 + 32h'1000\ 0000$)
		Encrypts Message (c) $c_i = m_i^e \bmod n$
	<== Sender transmits to Receiver the Encrypted Message (c)	
Decrypts Message (m) $m_i = c_i^d \bmod n$		
Removes message padding (e.g. $M_1 = m_1 - 32h'1000\ 0000$)		
Reassembles M		

Table 2 shows an example of such a transaction. While the padding shown is simplified, it is representative of how it works The encryption function only operates on chunks of the message that are smaller than the modulus, but it can't work on chunks that are too small, because this will make easy to crack the code. Padding handles this by enforcing rules that the message chunks are strictly of smaller bit count than the modulus, and then add a number of similar size to the modulus to ensure the size of the of the padded message will be sufficiently large to require operation of the mod function in the equation during encryption.

This algorithm was derived by number theory and works because of the modulus function, which as you will see, is a perfect match with digital hardware. The modulus function defines a limited number line that mathematics can be performed on, so operations that exceed the scale of the number line cause the numbers to wrap around to the other side. Analog clocks are a good example of the modulus function. If it is 1pm, and someone asks you to meet them in eighteen hours, the simple method of calculating the proper time is to add 18 + 1 = 19 hours, and then subtract 12 hours, increment the date field in you mind, and reset AM/PM toggle. The clock would now read 7, and mentally, you see 7am the next morning. The mathematical representation is 1 + 18 mod 12.

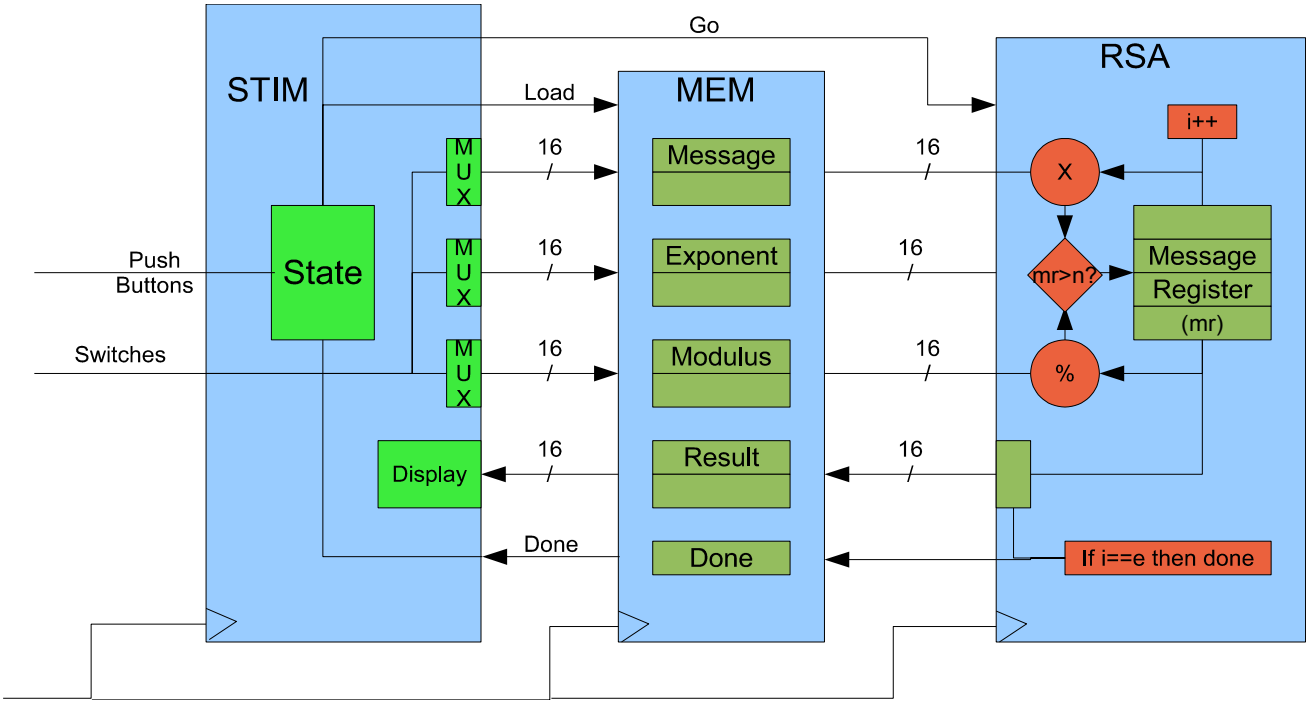
Time is based 12 hour increments because of its divisibility, 12 is divided by 2 3 4 6. 24 hours is divisible by 2 3 4 6 8 12. This means that you can break up a day into multiple smaller increments without the use of a calculator. Schedules are easier (three eight hour shifts a day), time cards are easier, and project management is easier. Say a task takes 13 eight hour shifts, and the task is to determine how long a task will take, then the answer is mathematically solved 13*8 mod 24, or 4 and 1/3 days. This can be simplified mentally by finding the least common denominators of the numbers, or 13*1 mod 3. Another example is the unit circle that is broken into 360 degrees, divisible by 2 3 4 5 6 8 9 10 12 15 1618 ect... calculation can be made using the integer number line and rational fractions.

RSA encryption goes in the other direction, by making it as difficult as possible to find shortcuts for doing a calculation. Two very large prime numbers are chosen, and multiplied together to generate the modulus. Finding the divisors of large numbers is a task that is difficult only because it is computationally expensive. It is estimated that it would take a billion years for a single computer to decompose a 1024 bit number, and the number could be made large enough that it would be impossible to decompose using all the computers in existence. The problem would be getting a single computer to operate on such a large modulus.

RSA encryption is increasing in bit length as time goes by. Yesterday's 128 bit encryption is easily broken by 64 bit machines, while at the same time, the number of discovered prime numbers is always increasing in addition to decompositions. Today's minimum encryption length is 512 bits, and while this is not even an order of magnitude larger bit count than 128 bit, the number of new prime numbers added is staggering. It will be obsolete at some time in the future, but right now, 512 bit is a mouthful for a 64 bit processor. Recall the encryption and description equation: $m_i = c_i^d \bmod n$. The exponential results in the multiplication of two 512 bit numbers with a 1024 bit number as the output. Every bit is important so a CPU will need to break the operation into 16 64 bit multiplies, followed by bit shifting the 16 128 partial multiplications and summing into a memory location 1024 bits long.

Dedicated hardware should yield a significant improvement in the computational performance of the CPU if operations are handled in parallel, and this is where we begin Project 3.

Project 3 Starting Point



Drawing 1: Function Diagram of Demo Drawing

The initial Verilog module demo in the project package contains a stimulus module, a memory buffer, an RSA encryption engine, and the familiar debouce and display modules. Drawing 1 gives a block diagram description of the circuit blocks. The initial module has been written for simplicity sake, and will simulate with the included test bench. The project will synthesis on ISE, but it will not operate on the board due to timing violations. You will first need to alter the clock rate to get this project operational using a DCM.

Functional Description:

Push Buttons:

Center	Reset System
Lower	Load Memory
Right	Go Start RSA function
Upper	Stop RSA function (clears RSA)
Left	Not Used

LED Description:

LEDs will be lit for a valid switch combination, with the asserted switches lit.

Switch Description:

0	$\overline{\text{Transmit}}$ / Receive
1	16 bit Plain text in hex
2	16 bit encrypted text in hex
3	32 bit Plain text in hex
4	32 bit encrypted text in hex
5	64 bit Plain text in hex
6	64 bit encrypted text in hex

Project Objectives

Stage 1: Simulate the demo project running at 16 bits, and then get the project running on the hardware get a feel for the hardware controls. You will need to identify the longest path in the hardware, and place a DCM in your circuit to get the circuit running. Indicate in a diagram what you did to get the circuit to work.
Stage 2: Alter the demo project to get the project running at 32 bits. The transmit algorithm should produce a result, but the receive algorithm may not work. Why is this the case? Identify the longest path and replace the synthesized blocks with IP Core logic and build a state machine to control the synthesized logic blocks. Verify your project works on the hardware.
Stage 3: Merge the Picoblaze into your design that will allow you to control the FPGA with a laptop. Partition your design with multiple clock domains to improve the performance of your circuit. Make to synthesis design objective files, one for speed, and one for area. What are the tradeoffs. Estimate the number of clock cycles that your design will take to process each message element, and then estimate the computation time of your circuit. Document your longest path on a schematic diagram. Also document you resource utilization. Discuss the important parts of the resource utilization.
Stage 4: Boost your circuit performance to 64 bits. It is likely that the architecture of the project will change to work at this bit depth. Read through the papers[ii] for some ideas on a faster[iii] architecture. Choose a direction that you want to move in (speed vs area) and implement an architecture that will work for 64 bits. Do not fear writing your own mathematical functions.
Stage 5: How many bits can you get you design to work on[iv]? Once you are writing your own modules, you can make these modules globally definable.

Deliverable

Demonstrate your project using the Picoblaze interface with a working solution. A 64 bit solution is required for full credit in this lab but you can turn 32 bits for a B, and 16 bits for a C. Getting a working system at greater than 64 bits will result in bonus points awarded. The fastest 64 bit system in computational time will be awarded extra credit, as well as the minimum area with a working function.

A paper is required that documents the actions taken in the 5 stages answering the questions that are posed there. A final section will detail the operation of your final circuit, including a description of the area that was used, the performance of the architecture, block diagrams, description of the slowest path, number of clock cycles per computation, computation time, a description of you two optimize design files, for speed and size. Let me know how you final circuit works.

Include your final design verilog files, the final .syr, and .twr files. Make sure that you reference all sources of code.

i See paper Hardware_Acceleration.pdf "Hardware Technologies for High-Performance Data-Intensive Computing", *Maya Gokhale, Jonathan Cohen, Andy Yoo, and W. Marcus Miller,* Lawrence Livermore National Laboratory, *Arpith Jacob,* Washington University in St. Louis, *Craig Ulmer,* Sandia National Laboratories, *Roger Pearce,* Texas A&M University

ii See paper Altera_FPGA.pdf "RSA & Public Key Cryptography in FPGAs", John Fry, Martin Langhammer, Altera Corporation

iii See paper FPGA_multiply.pdf "Efficient Hardware Realization of Truncated Multipliers using FPGA", Muhammad H. Rais, *member IEEE.*

iv See paper FPGA_RSA.pdf "FPGA Implementation of RSA Encryption Engine with Flexible Key Size", Muhammad I. Ibrahimy, Mamun B.I. Reaz, Khandaker Asaduzzaman and Sazzad Hussain.