



The Next Generation of Cryptanalytic Hardware

Microsoft Friday Security Talks, 8/19/05

David Hulton <dhulton@picocomputing.com>
Embedded Systems Engineer, Pico Computing, Inc.
Chairman, ToorCon Information Security Conference

Greg Edverson <gedverson@picocomputing.com>
Embedded Systems Engineer, Pico Computing, Inc.

Goals



- Introduction
 - What is an FPGA?
 - Gate Logic / Verilog
- ImpulseC
 - Introduction
 - Demo
- Cracking \w Hardware
 - History
- Chipper
 - Lanman/NTLM
 - Demo
 - Performance

Introduction to FPGAs


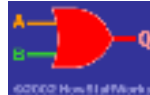
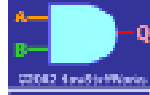


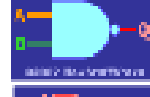



- Field Programmable Gate Array
 - Lets you prototype IC's
 - Code translates directly into circuit logic

What is Gate Logic?



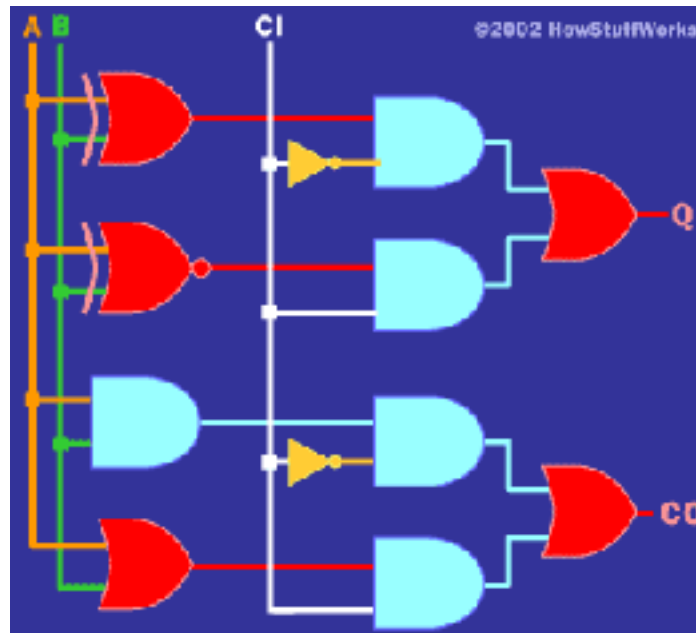
- The basic building blocks of any computing system

not	$\sim a$	not	
or	$a b$	or	
and	$a \& b$	and	
xor	$a \wedge b$	xor	
nor	$\sim(a b)$	nor	
nand	$\sim(a \& b)$	nand	
xnor	$\sim(a \wedge b)$	xnor	

What is Gate Logic?



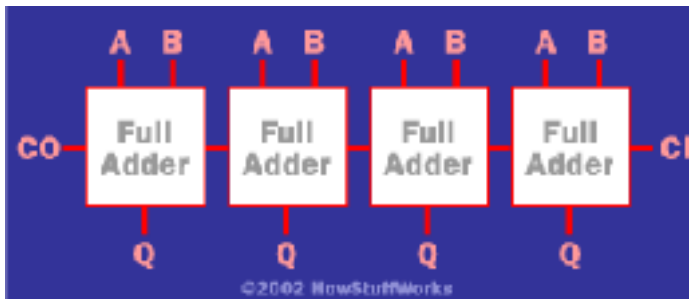
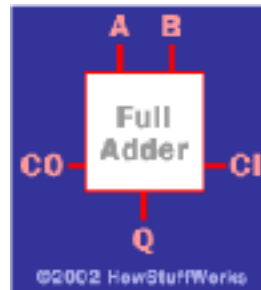
- Build other types of logic, such as adders:



What is Gate Logic?



- Which can be chained together:



What is Gate Logic?

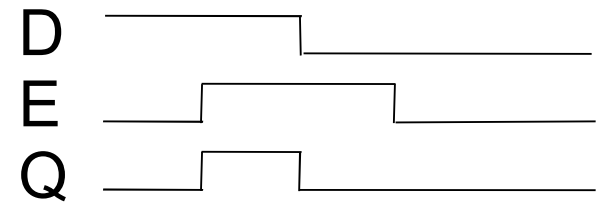
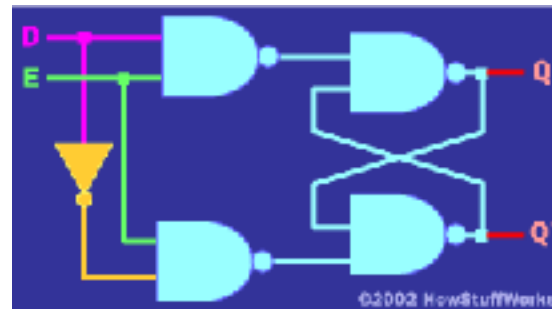


- And can be used for storing values:

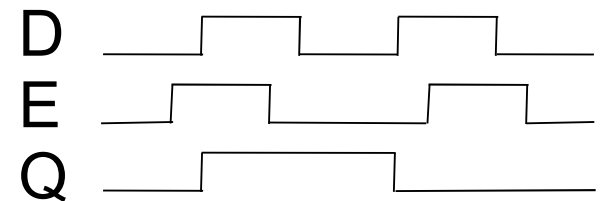
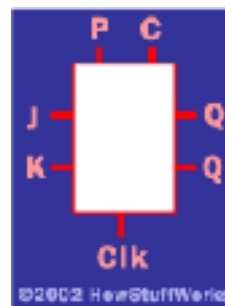
- Feedback



- Flip-Flop / Latch



- JK Flip-Flop

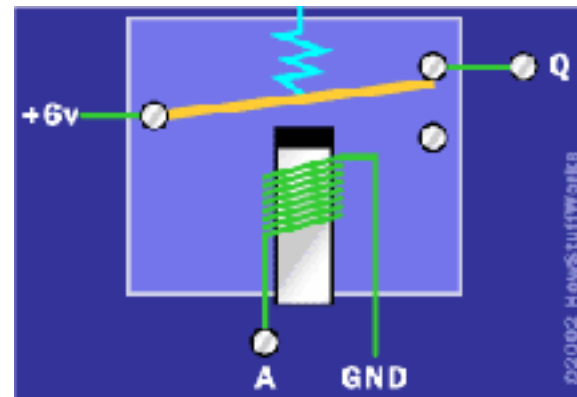


What is Gate Logic?

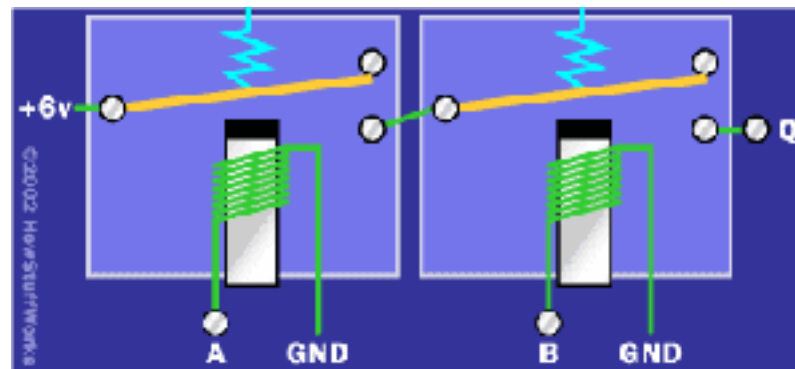


- This can be implemented with electronics:

- NOT



- AND



What is an FPGA?

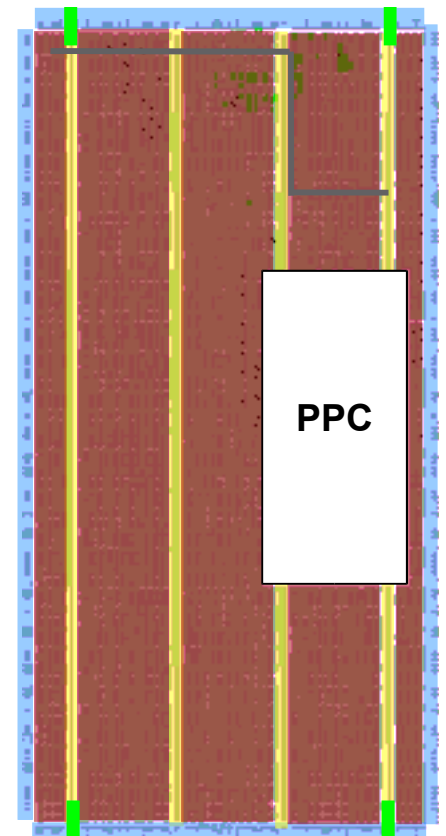


- An FPGA is an array of configurable gates
 - Gates can be connected together arbitrarily
 - States can be configured
 - Common components are provided
 - Any type of logic can be created

What is an FPGA?



- **Configurable Logic Blocks (CLBs)**
 - Registers (flip flops) for fast data storage
 - Logic Routing
- **Input/Output Blocks (IOBs)**
 - Basic pin logic (flip flops, muxs, etc)
- **Block Ram**
 - Internal memory for data storage
- **Digital Clock Managers (DCMs)**
 - Clock distribution
- **Programmable Routing Matrix**
 - Intelligently connects all components together



FPGA Pros / Cons



- Pros
 - Common Hardware Benefits
 - Massively parallel
 - Pipelineable
 - Reprogrammable
 - Self-reconfiguration
- Cons
 - Size constraints / limitations
 - More difficult to code & debug

Introduction to FPGAs



- Common Applications
 - Encryption / decryption
 - AI / Neural networks
 - Digital signal processing (DSP)
 - Software radio
 - Image processing
 - Communications protocol decoding
 - Matlab / Simulink code acceleration
 - Etc.

Introduction to FPGAs



- Common Applications
 - Encryption / decryption
 - AI / Neural networks
 - Digital signal processing (DSP)
 - Software radio
 - Image processing
 - Communications protocol decoding
 - Matlab / Simulink code acceleration
 - Etc.

Types of FPGAs



- Antifuse
 - Programmable only once
- Flash
 - Programmable many times
- SRAM
 - Programmable dynamically
 - Most common technology
 - Requires a loader (doesn't keep state after power-off)

Types of FPGAs



- Xilinx
 - Virtex-4
 - Optional PowerPC Processor
- Altera
 - Stratix-II

Verilog



- Hardware Description Language
- Simple C-like Syntax
- Like Go - Easy to learn, difficult to master

Verilog



- One bit AND

- C

```
u_char and(u_char a, u_char b) {  
    return((a & 1) & (b & 1));  
}
```

- Verilog

```
module and(a, b, c);  
input a, b;  
output c;  
  
assign c = a & b;  
endmodule
```

- Gate



Verilog



- 8 bit AND

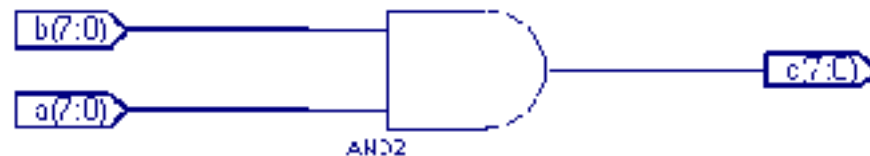
- C

```
u_char or(u_char a, u_char b) {  
    return(a & b);  
}
```

- Verilog

```
module or(a, b, c);  
input [7:0] a, b;  
output [7:0] c;  
  
assign c = a & b;  
endmodule
```

- Gate



Verilog



- 8 bit Flip-Flop

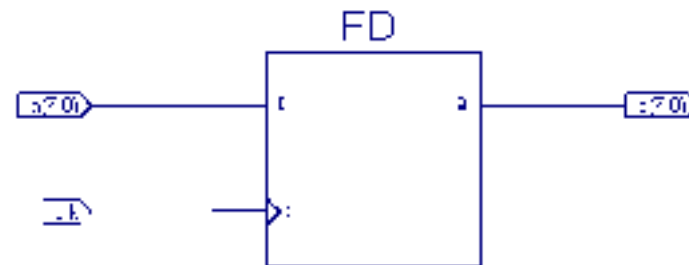
- C

```
u_char ff(u_char a) {  
    static u_char t = a;  
    return(t);  
}
```

- Verilog

```
module ff(clk, a, c);  
input clk;  
input [7:0] a;  
output [7:0] c;  
reg [7:0] c;  
  
always @(posedge clk) c <= a;  
endmodule
```

- Gate



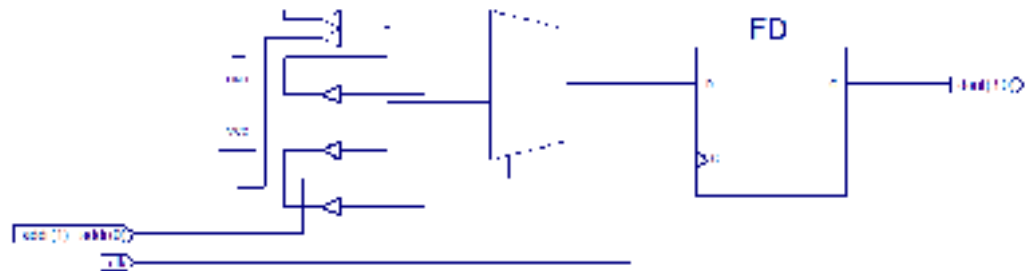
Verilog



- ROM
 - C
- Verilog
- Gate

```
u_char rom[] = {2, 3, 5, 7};
```

```
module rom(clk, addr, dout);  
input clk;  
input [1:0] addr;  
output [3:0] dout;  
reg [3:0] dout;  
  
always @(posedge clk)  
    case(addr[1:0])  
        0: dout <= 2;  
        1: dout <= 3;  
        2: dout <= 5;  
        3: dout <= 7;  
    endcase  
  
endmodule
```



CoDeveloper™ with Impulse C™

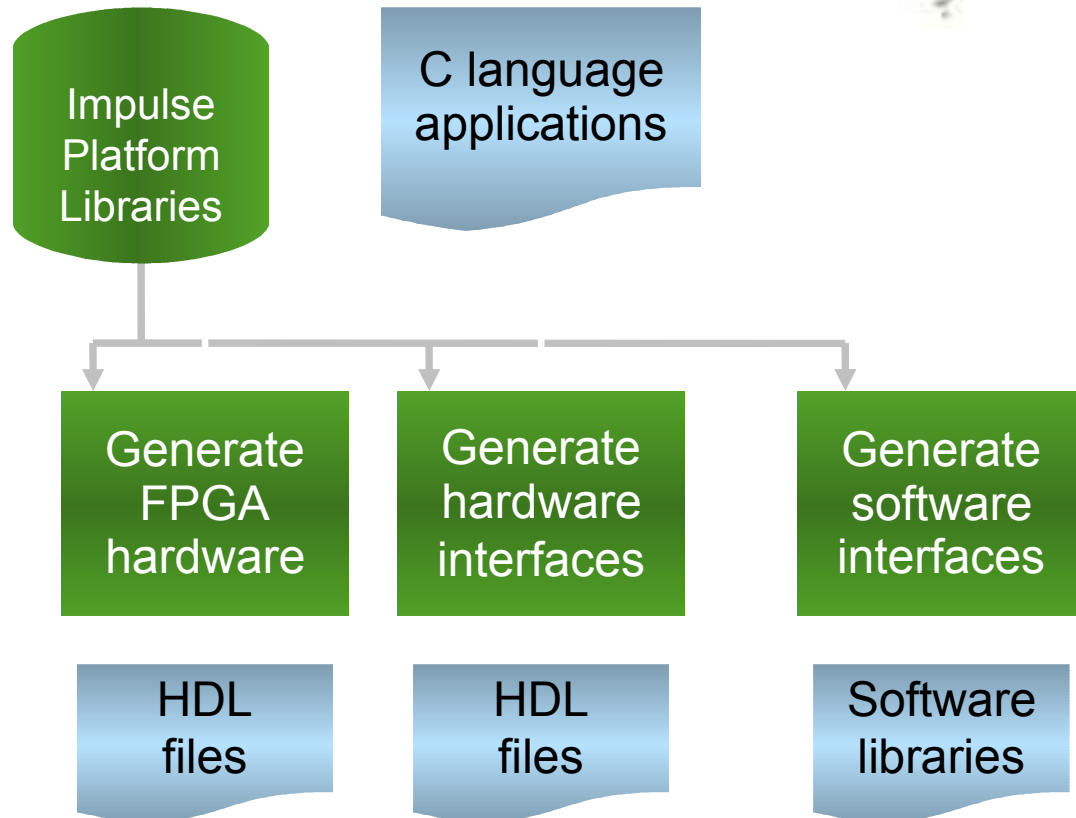


Develop and debug the application using standard C tools.

Use Impulse C functions to partition the application into hardware and software processes.

Use CoDeveloper to compile hardware processes to HDL and generate hardware stream and memory interfaces.

Export hardware and software blocks to selected synthesis, simulation and FPGA platform generation tools.



CoDeveloper™ with Impulse C™



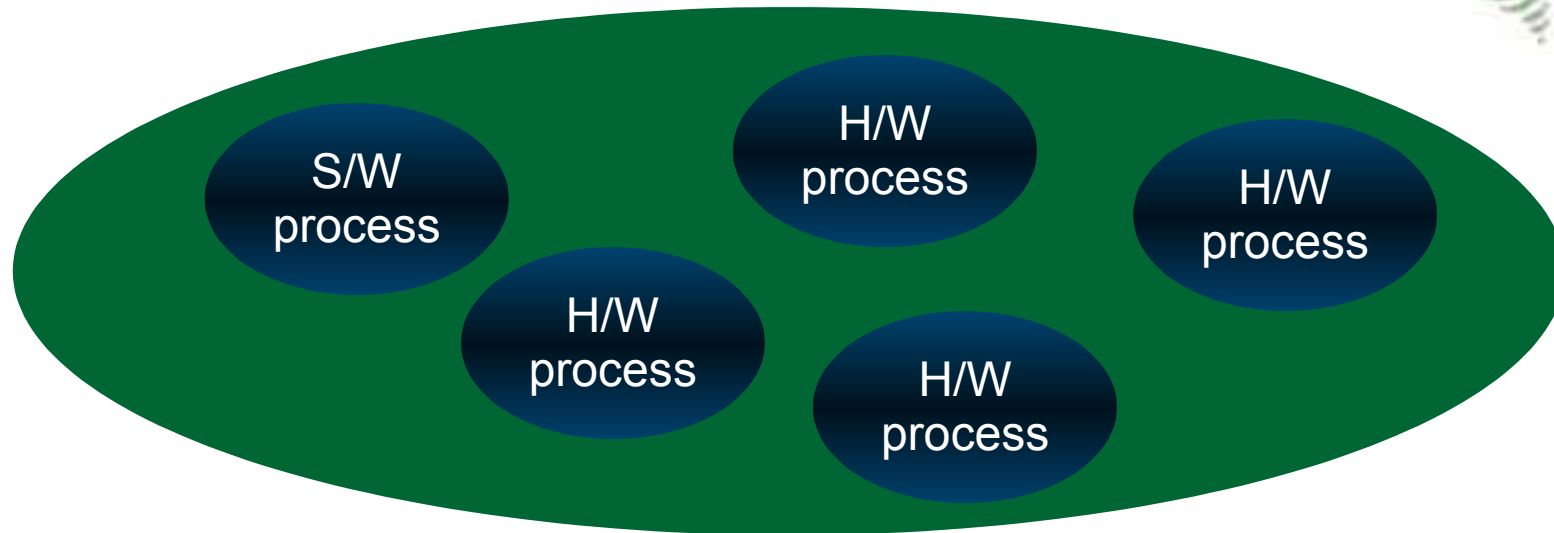
- Providing automatic conversion of C-language routines to FPGA hardware
 - Automated extraction of parallel hardware from untimed C
 - Proprietary optimization features
 - Instruction and resource scheduling, loop unrolling and pipelining
- Providing a programming model and tools for system-level parallel programming
 - Allows multiple software and hardware processes to be created, connected and synchronized, all in C-language
 - Programming model designed for software programmers
 - Based on Streams-C, from Los Alamos National Labs

Application Domains



- Applications requiring repetitive computations at very high speed
 - Application can be modularized (partitioned) between hardware and software
 - Dataflow-oriented, high degrees of parallelism
 - Pipelined algorithms (e.g. filters)
- For processing streams of data in real time
 - Imaging and audio
 - Communications
 - Digital Signal Processing (DSP)
 - Encryption and decryption
 - Bioinformatics and supercomputing

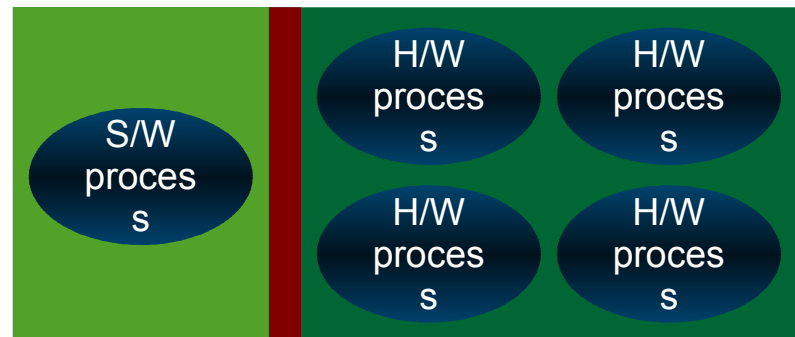
From C Application to FPGA Platform



Platform libraries support existing FPGA synthesis and embedded compiler environments

Automatic generation of hardware/software interfaces is optimized for target platforms

FPGA hardware is automatically created from C language processes



*The result?
Accelerated software
with minimal need for
hardware or FPGA
design knowledge*

Example: FIR Filter



- Data and coefficients passed into filter via data stream
 - Could also use shared memory
- Algorithm written using untimed, hardware-independent C code
 - Using coding styles familiar to C programmers
- Software test bench written in C to test functionality
 - In software simulation
 - In actual hardware



```
void fir(co_stream filter_in, co_stream filter_out) {  
    int32 coef[TAPS]; int32 firbuffer[TAPS];  
    int32 nSample, nFiltered, accum, tap;
```

Declare stream interfaces

Open the streams

Read in the coefficients

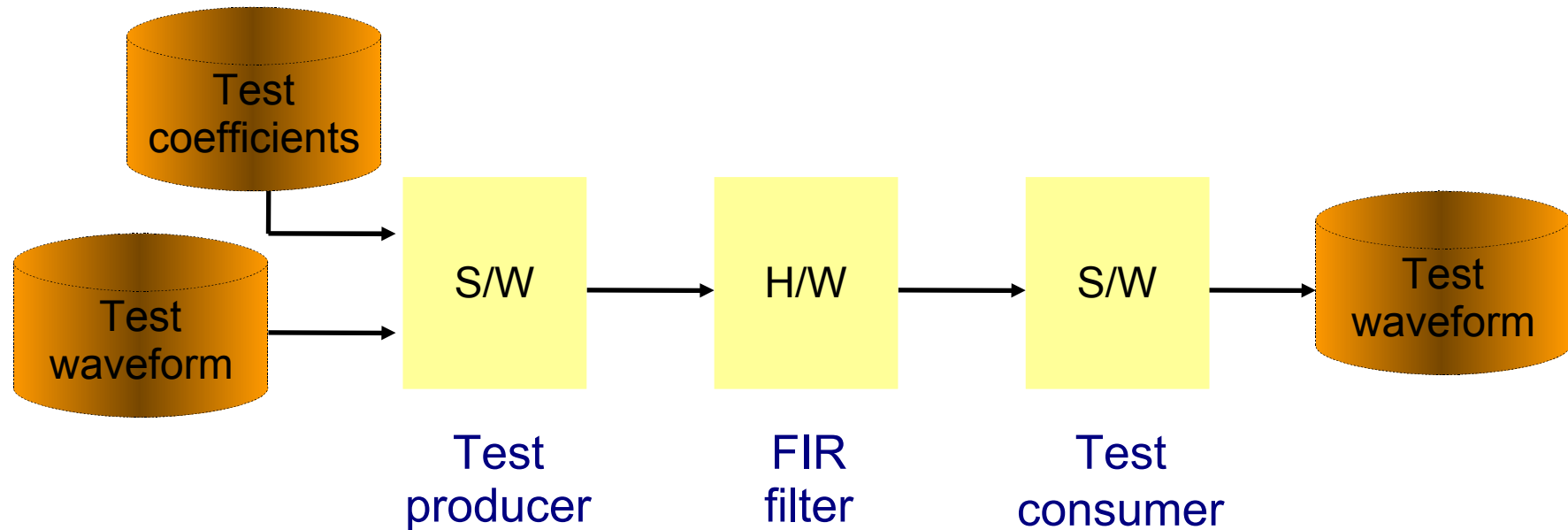
Read in the first *n* values

Process the incoming stream
and perform the filter operation
to generate outputs

When done, close the streams

```
    co_stream_open(filter_in, O_RDONLY, INT_TYPE(32));  
    co_stream_open(filter_out, O_WRONLY, INT_TYPE(32));  
    // First fill the coef array with the coefficients...  
    for (tap = 0; tap < TAPS; tap++) {  
        co_stream_read(filter_in, &nSample, sizeof(int32));  
        coef[tap] = nSample;  
    }  
    // Now fill the firbuffer array with the first n values...  
    for (tap = 1; tap < TAPS; tap++) {  
        co_stream_read(filter_in, &nSample, sizeof(int32));  
        firbuffer[tap-1] = nSample;  
    }  
    // Now we have an almost full buffer and can start processing waveform samples...  
    while ( co_stream_read(filter_in, &nSample, sizeof(int32)) == co_err_none ) {  
        firbuffer[TAPS-1] = nSample;  
        for (accum = 0; tap = 0; tap < TAPS; tap++) {  
            accum += firbuffer[tap] * coef[tap];  
        }  
        nFiltered = accum >> 2;  
        co_stream_write(filter_out, &nFiltered, sizeof(int32));  
        for (tap = 1; tap < TAPS; tap++) {  
            firbuffer[tap-1] = firbuffer[tap];  
        }  
    }  
    co_stream_close(filter_in);  
    co_stream_close(filter_out);  
}
```

FIR Filter Functional Test



This test can be performed in desktop simulation (using Visual Studio or some other C environment) or can be performed using an embedded processor for the producer/consumer modules.

Impulse C Configuration Function



```
void config_fir(void *arg)
{
    co_stream waveform_raw;
    co_stream waveform_filtered;

    co_process producer_process;
    co_process fir_process;
    co_process consumer_process;

    waveform_raw = co_stream_create("waveform_raw", INT_TYPE(32), BUFSIZE);
    waveform_filtered = co_stream_create("waveform_filtered", INT_TYPE(32), BUFSIZE);

    producer_process = co_process_create("producer_process", (co_function)test_producer,
                                         1, waveform_raw);

    fir_process = co_process_create("filter_process", (co_function)fir,
                                    2, waveform_raw, waveform_filtered);

    consumer_process = co_process_create("consumer_process", (co_function)test_consumer,
                                         1, waveform_filtered);

    // Assign processes to hardware elements
    co_process_config(fir_process, co_loc, "PE0");
}
```

stream declarations

process declarations

stream creation

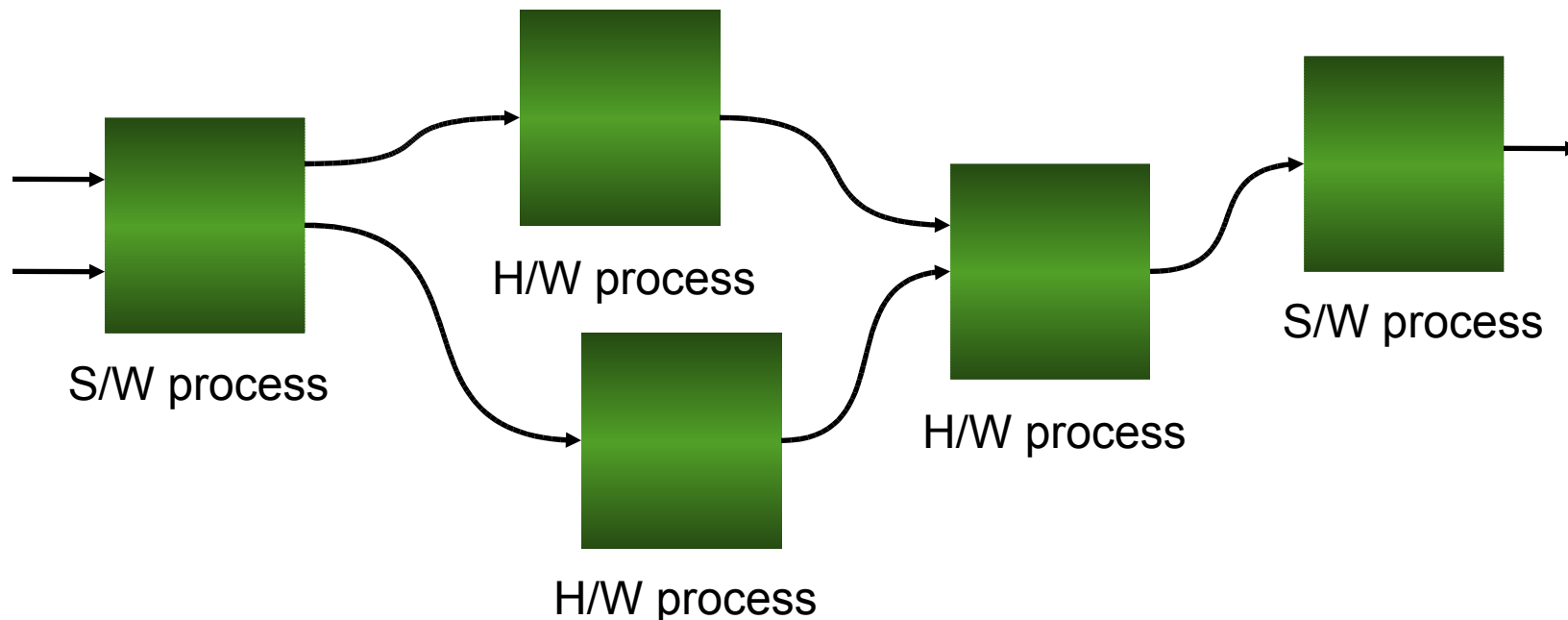
process creation

process configuration (hardware)

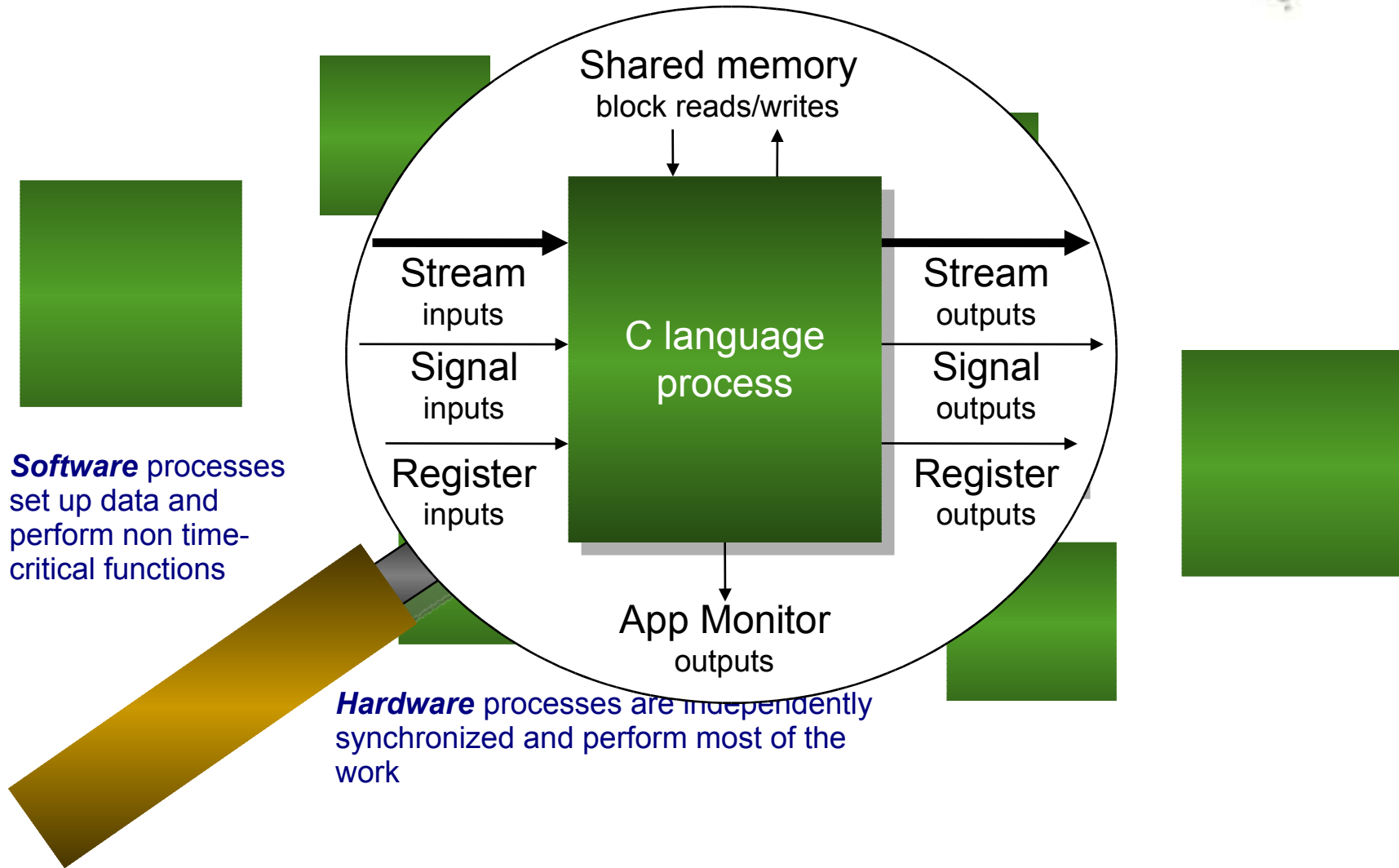
Parallel Programming Model



- **Communicating Process Model**
 - Buffered communication channels (FIFOs) to implement streams
 - Supports dataflow and message-based communications between functional units and local or shared memories
 - Supports parallelism at the application level and at the level of individual processes (via automated scheduling/pipelining)



An Impulse C™ Process



Software processes set up data and perform non time-critical functions

Hardware processes are independently synchronized and perform most of the work

Impulse CTM Demo



Demonstration

History of FPGAs and Cryptography



- Minimal Key Lengths for Symmetric Ciphers
 - Ronald L. Rivest (R in RSA)
 - Bruce Schneier (Blowfish, Twofish, etc)
 - Tsutomu Shimomura (Mitnick)
 - A bunch of other ad hoc cypherpunks

History of FPGAs and Cryptography



Budget	Tool	40-bits	56-bits	Recom
Pedestrian Hacker				
Tiny	Computers	1 week	infeasible	45
\$400	FPGA	5 hours	38 years	50
Small Company				
\$10K	FPGA	12 min	556 days	55
Corporate Department				
\$300K	FPGA	24 sec	19 days	60
	ASIC	0.18 sec	3 hrs	
Big Company				
\$10M	FPGA	0.7 sec	13 hrs	70
	ASIC	0.005 sec	6 min	
\$300M	ASIC	0.0002 sec	12 sec	75

History of FPGAs and Cryptography



- 40-bit SSL is crackable by almost anyone
- 56-bit DES is crackable by companies
- Scared yet?

This paper was published in 1996

History of FPGAs and Cryptography



- 1998
 - The Electronic Frontier Foundation (EFF)
 - Cracked DES in < 3 days
 - Searched ~9,000,000,000 keys/second
 - Cost < \$250,000

History of FPGAs and Cryptography



- 2001
 - Richard Clayton & Mike Bond (University of Cambridge)
 - Cracked DES on IBM ATMs
 - Able to export all the DES and 3DES keys in ~ 20 minutes
 - Cost < \$1,000 using an FPGA evaluation board

History of FPGAs and Cryptography



- 2002
 - Rouvroy Gael, Standaert Francois-Xavier and others from the UCL Crypto Group
 - Implemented a linear cryptanalysis attack on DES
 - Used FPGAs to generate dictionary tables
 - Chosen-plaintext attack can recover key in 10 seconds with 72% success rate

History of FPGAs and Cryptography



- 2004
 - Philip Leong, Chinese University of Hong Kong
 - IDEA
 - 50Mb/sec on a P4 vs. 5,247Mb/sec on Pilchard
 - RC4
 - Cracked RC4 keys 58x faster than a P4
 - Parallelized 96 times on a FPGA
 - Cracks 40-bit keys in 50 hours
 - Cost < \$1,000 using a RAM FPGA (Pilchard)

Chipper



- Currently Supports
 - Unix DES
 - Windows Lanman
 - Windows NTLM (full-support coming soon)
 - Multiple Cards/FPGAs ;-)

Lanman Hashes



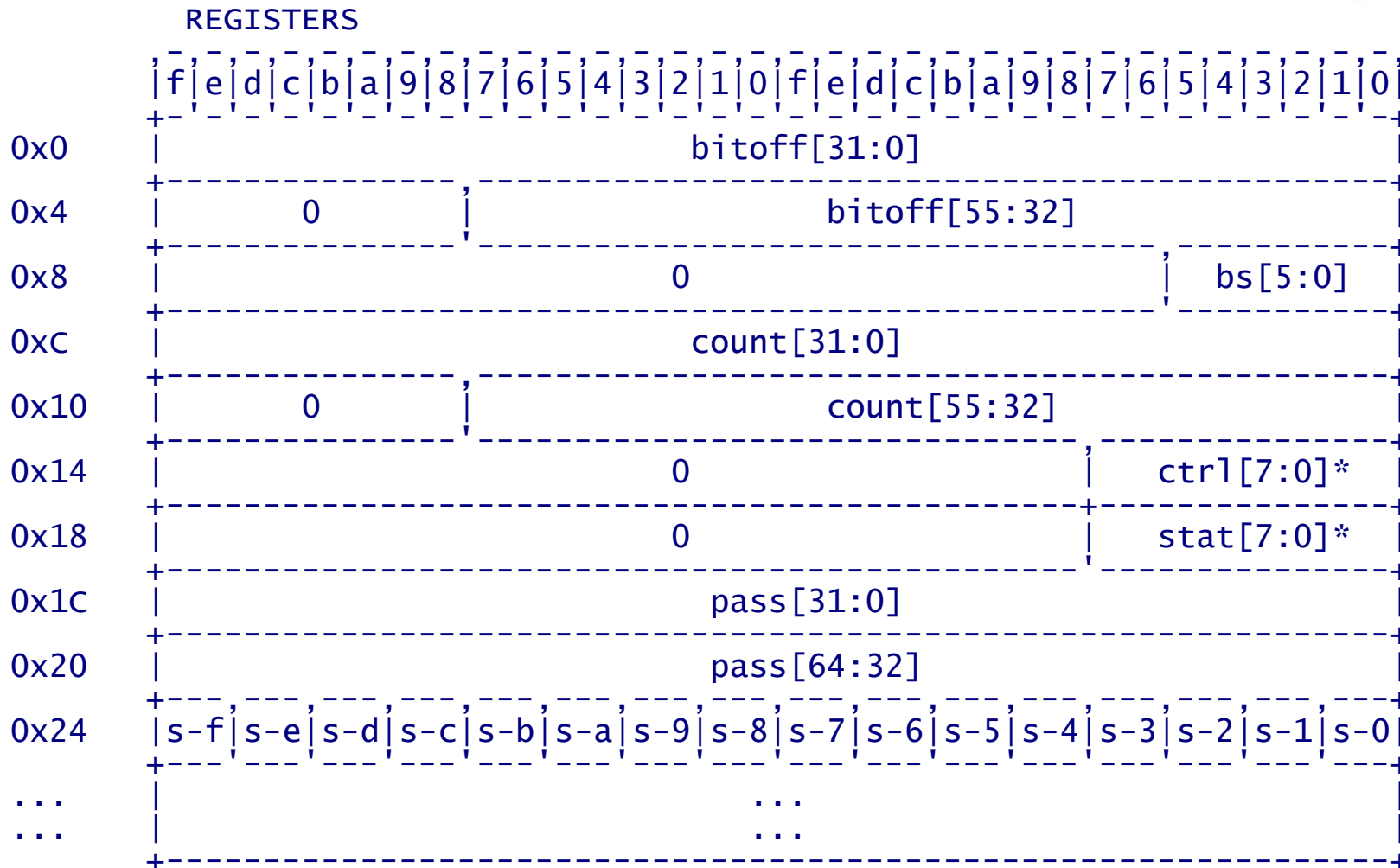
- Lanman
 - 14-Character Passwords
 - Case insensitive (converted to upper case)
 - Split into 2 7-byte keys
 - Used as key to encrypt static values with DES

Chipper



- Hardware Design
 - Pipeline design
 - Internal cracking engine
 - passwords = `lmcrack(hashes, options);`
 - Interface over PCMCIA/CompactFlash
 - Can specify cracking options
 - Bits to search
 - e.g. Search 55-bits (instead of 56)
 - Offset to start search
 - e.g. First card gets offset 0, second card gets offset 2^{55}
 - Typeable/printable characters
 - Alpha-numeric
 - Allows for basic distributed cracking & resume functionality

Interface Layout

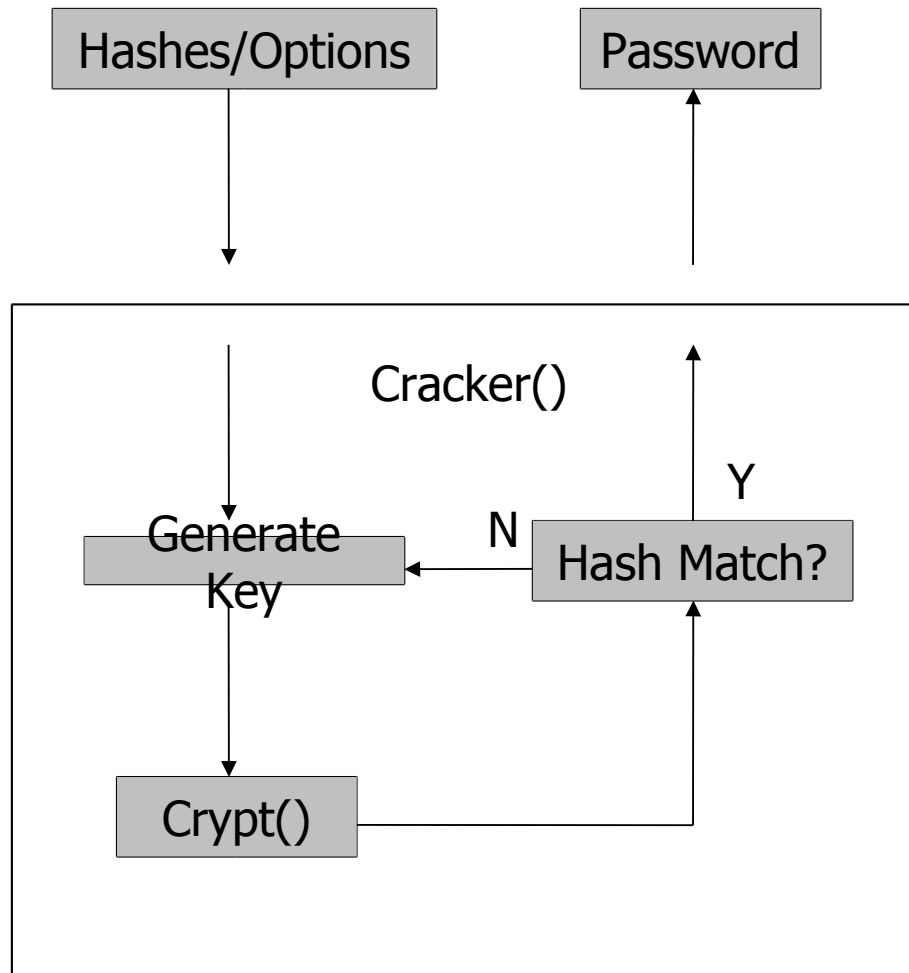


Interface Layout

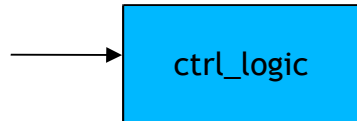


HASHES/PASSWORDS	
	f e d c b a 9 8 7 6 5 4 3 2 1 0 f e d c b a 9 8 7 6 5 4 3 2 1 0
0x0	hash/pass[0][31:0]
0x4	hash/pass[0][63:32]
0x8	hash/pass[0][95:64]
0xC	hash/pass[0][127:96]
0x10	hash/pass[1][31:0]
0x14	hash/pass[1][63:32]
0x18	hash/pass[1][95:64]
0x1C	hash/pass[1][127:96]
0x20	...
...	...

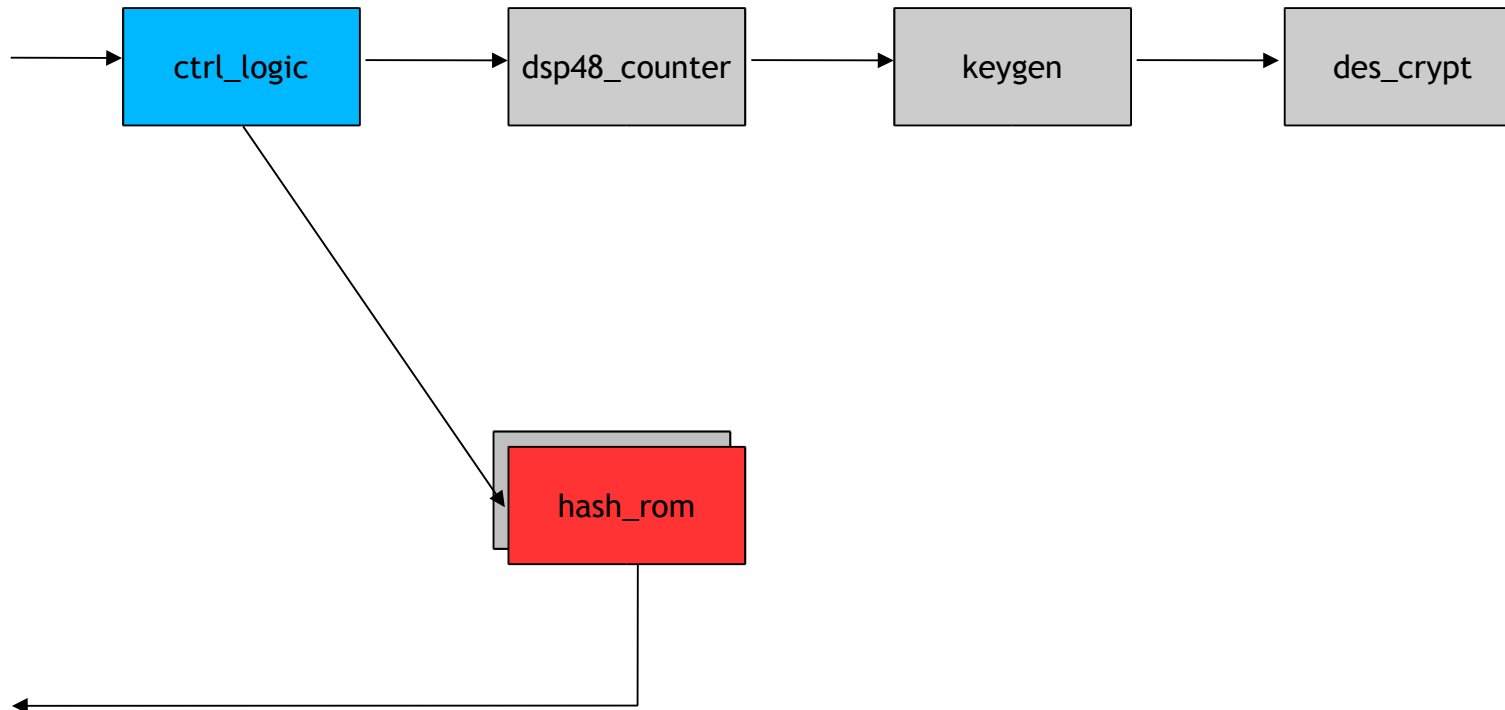
Password File Cracker



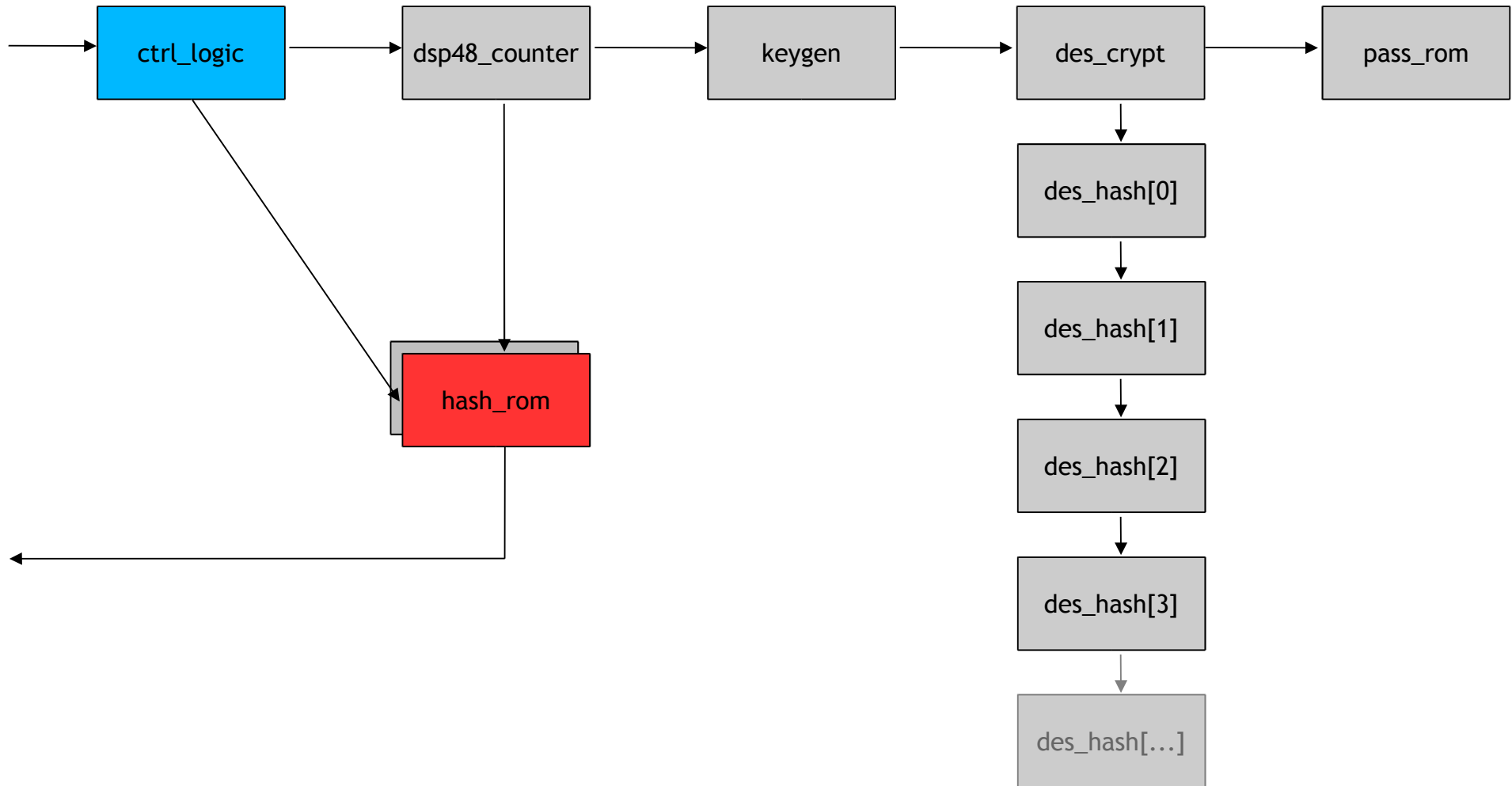
Interface Layout



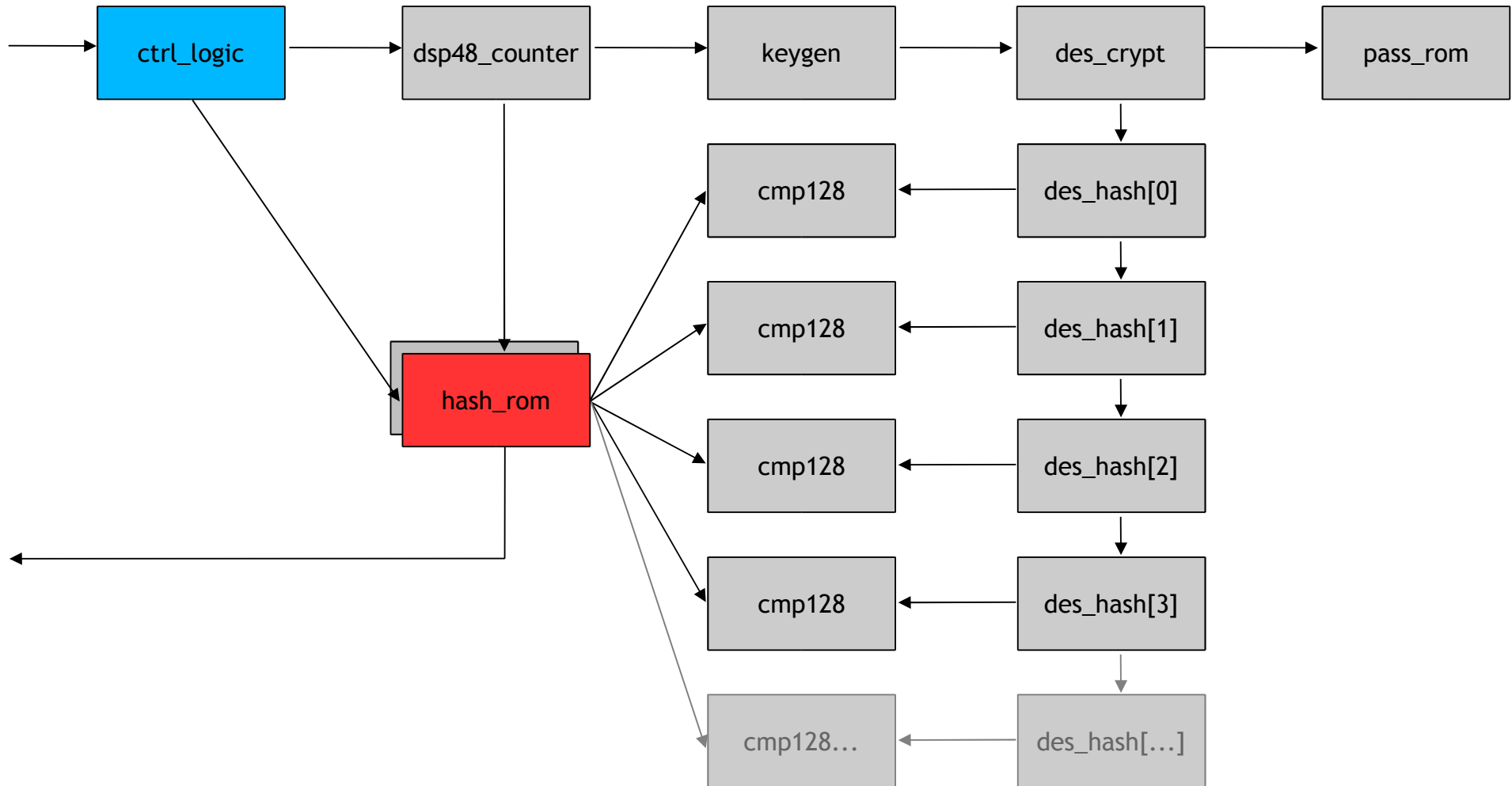
Interface Layout



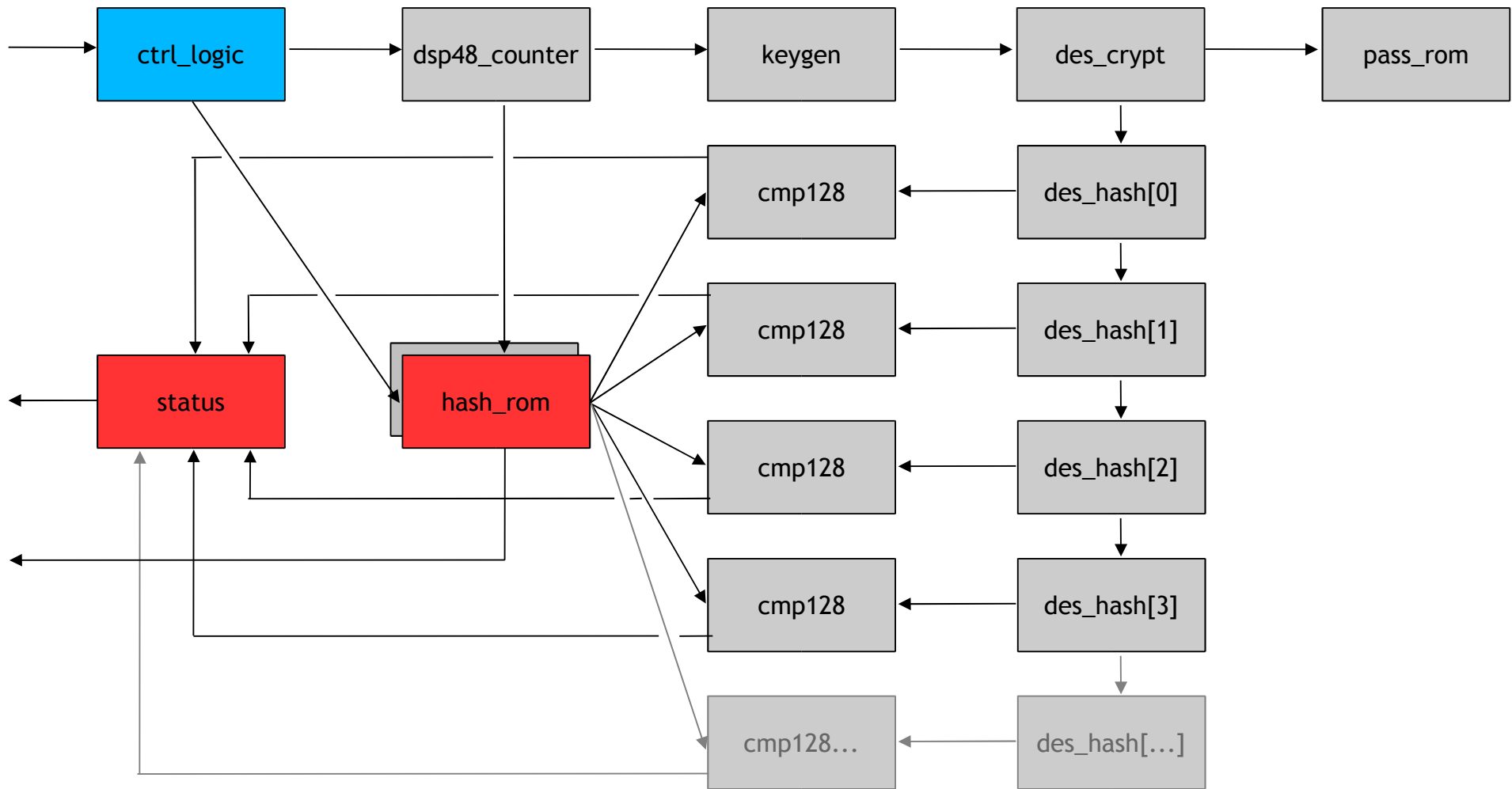
Interface Layout



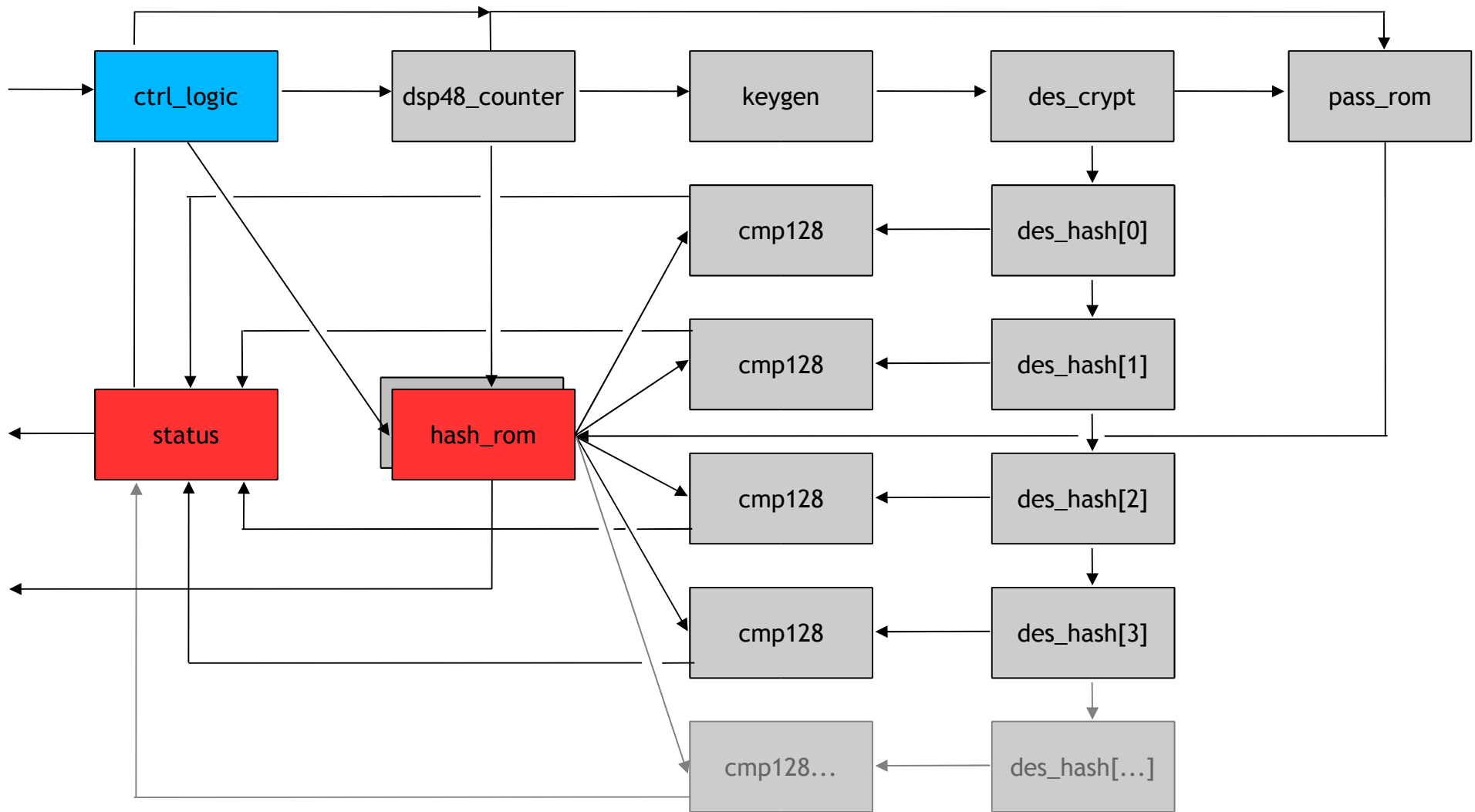
Interface Layout



Interface Layout



Interface Layout



Chipper



- Software Design
 - GUI and Console Interfaces
 - WxWidgets
 - Windows
 - Linux
 - MacOS X (coming soon)
 - Supports cracking 128 keys in parallel on each card
 - Supports 4x fast mode for just one hash pair
 - Can automatically load required FPGA image
 - Supports multiple card clusters

Lanman Cracking



- PC (3.0Ghz P4 \w rainbowcrack)
 - ~ 2,000,000 c/s
- Hardware (Low end FPGA \w Chipper)
 - 125Mhz = 125,000,000 c/s per core
 - 500Mhz = 500,000,000 c/s for fast mode!

Type	P4	E-12	8 E-12
64-characters	25 D	2 H	18 M
48-characters	3.4 D	20 M	1.5 M
32-characters	4.7 H	1 M	9 S

Pico E-12



- Pico E-12
 - Compact Flash Type-II Form Factor
 - Virtex-4 (LX25 or FX12)
 - 1 Million Gates (~25,000 CLBs)
 - Optional 450 MHz PowerPC Processor
 - 128 MB PC-133 RAM
 - 64 MB Flash ROM
 - Gigabit Ethernet
 - JTAG Debugging Port



PicoCrack Demonstration



Demonstration



- Sourceforge project
 - Chipper
 - Lanman & NTLM cracking cores
 - Modular Exponentiation
 - A5/2 (for some GSM research)

Technology Trends



- Technology Trends
 - Embedded platforms are either cheap and slow or expensive and fast
 - There will always be a cost factor with regards to crypto
 - This has plagued smart cards, speedpasses, mobile devices, etc.
 - The future is definitely implementing more advanced cryptanalysis attacks
 - As cheap chips get faster, the workload for brute-force increases exponentially with the keysize

Hardware Trends



- **FPGAs are increasing according to Moore's Law**
 - Density - Increasing
 - Clock Speed - Increasing
 - Components - Created and expanded to fit markets
 - Cost - Dropping
- **Slowly starting to compete with ASICs**
- **Starting to become cheap enough for consumers**
 - FPGA software acceleration will start to become mainstream
 - Fast parallel computing will increase in popularity
- **Future Applications**
 - Neural Networks & Self-reconfiguration
 - Attacks on WEP/WPA/GSM
 - Analysis and Correlation

Conclusions / Shameless Plugs



- ToorCon 7
 - September 16th-18th, 2005
 - Convention Center, San Diego, CA
 - <http://www.toorcon.org>
- ShmooCon 2
 - January, 2006
 - Washington DC
 - <http://www.shmoocon.org>

Questions ? Suggestions ?



- **OpenCiphers**
 - <http://www.openciphers.org>
- **OpenCores**
 - <http://www.opencores.org>
- **Xilinx**
 - ISE Foundation (Free 60-day trial)
 - <http://www.xilinx.com>
- **Pico Computing, Inc.**
 - <http://www.picocomputing.com>