Distributed FPGA Number Crunching For The Masses

Felix Domke <tmbinc@elitedvb.net>

27C3

number crunching for the masses

Who still crunches numbers these days?
Breaking Keys?

Hasn’t everyone moved to 128 bit crypto yet?

Turns out: Plenty of stuff still unsolved!

number crunching for the masses

DES is the classic example for brute force

56 bits keyspace

Synthetic problems as well:

NQueens, OGR, ...

But let's focus on a real-world example.

SEGA

SERVICE MANUAL

TRIFORCE

SEGA CORPORATION

a number crunching for the masses

Triforce Arcade Console

Games have been dumped, emulated…

But still: one unsolved mystery!

number crunching for the masses

xxd -g1 < FIRMWARE.ASIC

```
0000000: ca f8 6d 9b c9 82 0a e8 42 39 0c c6 4a 14 82 7f
0000010: 0c d8 2d b5 13 a8 af 59 06 de 6f f3 56 23 73 59
0000020: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000030: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000040: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000050: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000060: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000070: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000080: 06 de 6f f3 56 23 73 59 06 de 6f f3 56 23 73 59
0000090: 5f e1 63 54 fe 5e 29 a1 06 de 6f f3 56 23 73 59
00000a0: 0f 1e e4 12 53 af d5 30 06 de 6f f3 56 23 73 59
00000b0: 65 4a 06 d6 59 5a b8 16 06 de 6f f3 56 23 73 59
00000c0: a8 54 24 87 96 2a 51 f7 06 de 6f f3 56 23 73 59
00000d0: 1c fd b0 54 6f 0f 30 f5 06 de 6f f3 56 23 73 59
00000e0: 38 d4 f5 2a 76 7e 38 5d 06 de 6f f3 56 23 73 59
00000f0: 10 9f 99 d5 e1 4b 4a b1 06 de 6f f3 56 23 73 59
0000100: 93 1c f9 0f 47 48 cc cb 06 de 6f f3 56 23 73 59
...
```

number crunching for the masses

Hey, repeating 8-byte blocks!

Could indicate a non-chaining 8b block cipher.

We know that games use DES encryption.

So firmware might use the same?

number crunching for the masses

Histogram of 8 byte blocks:

```
...
b37b715f960d0969 6
132c1f06dba18bcd 7
f7d005d50bfd9843 10
bb7e6b52bc53bb46 10
eeaa44290bf2aebb 11
9bfdf2b8e963d9f5 11
0d3d6820f74fd4d5 180
06de6ff356237359 191
```

number crunching for the masses

Most occuring plaintexts are usually all-zero or all-FF. We don’t know which one is which, and call them C1 and C2.

Triforce games encryption bytereverses ciphertexts - so these might be reversed as well.

DES “complementation property”:

$$\mathrm{E}_k(P)=\mathrm{C} \Leftrightarrow \mathrm{E}_{\sim k}(\sim P)=\sim \mathrm{C}$$

“Inverted plaintext and key yields inverted ciphertext”

If we search for ~C(~0), we might find ~k earlier than k.

Saves us one bit (half the time)!

number crunching for the masses

To formulate the goal:

We're looking for a key that encrypts
00 00 00 00 00 00 00 00 into either of:

```
0d3d6820f74fd4d5 C1
06de6ff356237359 C2
d5d44ff720683d0d C1[::-1]
59732356f36fde06 C2[::-1]
f2c297df08b02b2a ~C1
f921900ca9dc8ca6 ~C2
2a2bb008df97c2f2 ~C1[::-1]
a68cdca90c9021f9 ~C2[::-1]
```

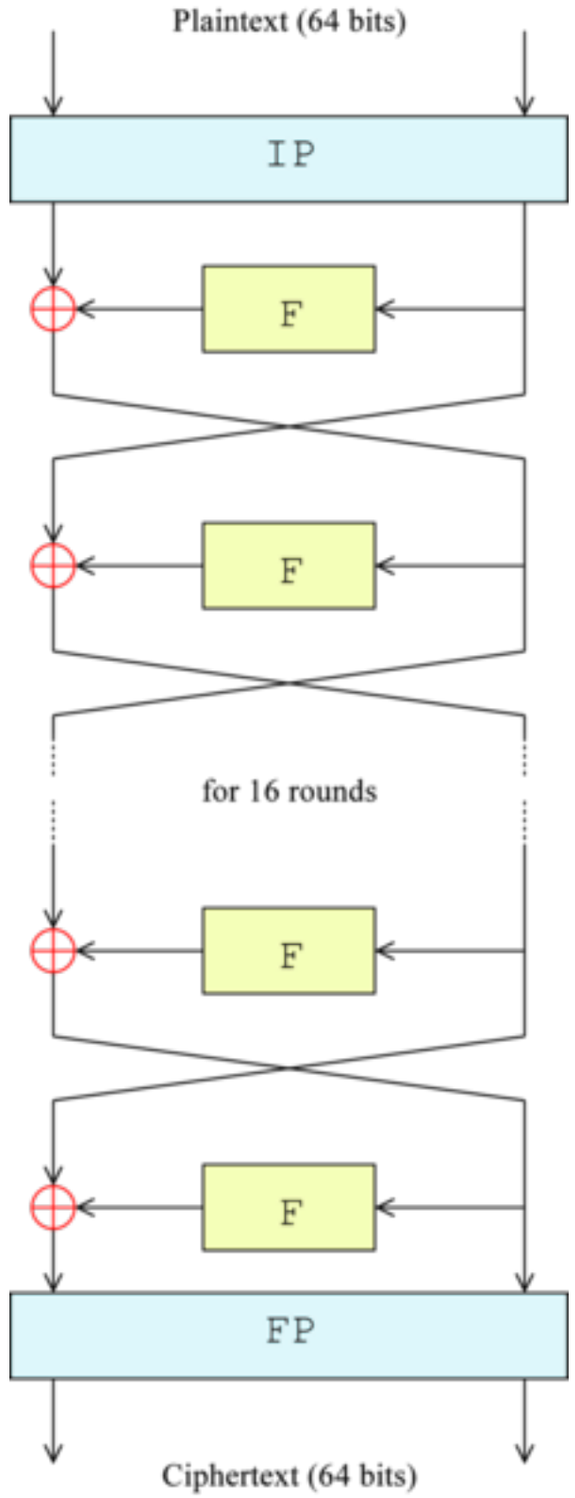
number crunching for the masses

DES Blockcipher

Block Cipher based on Feistel Scheme

16 rounds

Each round uses 8 6-bit lookups (“S-Boxes”)



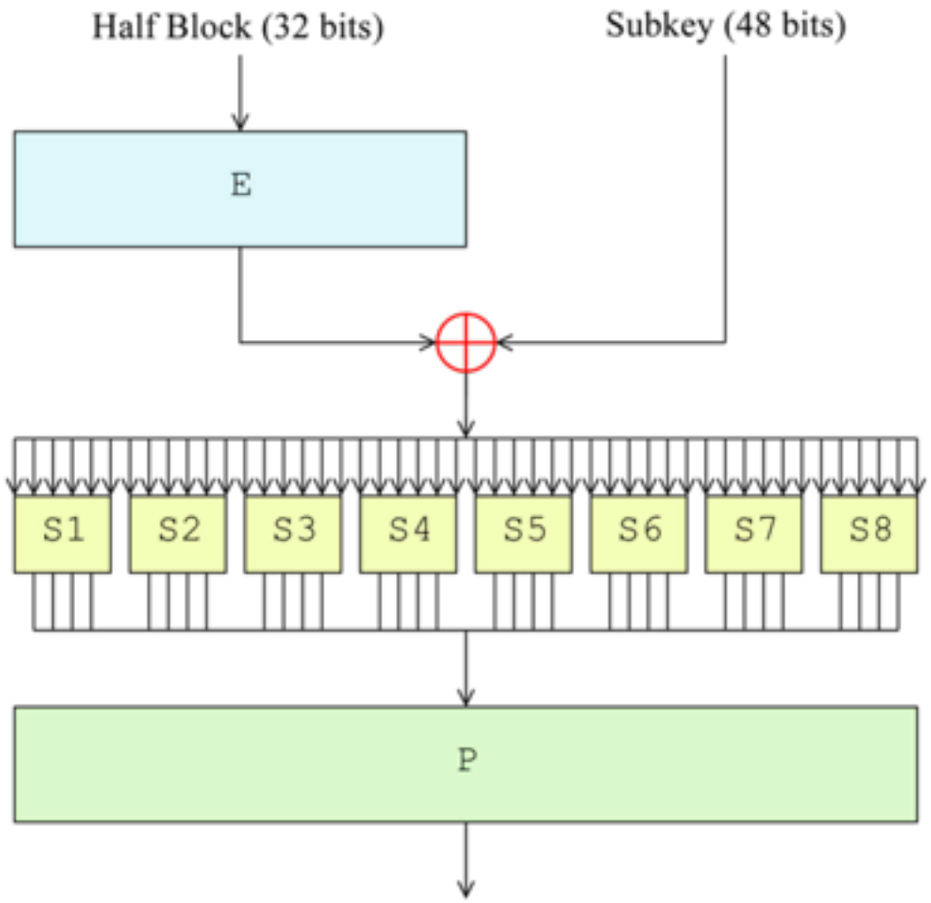
number crunching for the masses

DES Blockcipher

Block Cipher based on Feistel Scheme

16 rounds

Each round uses 8 6-bit lookups (“S-Boxes”)



number crunching for the masses

Main work are S-Boxes

For ALU-based architectures, bitslicing is the way to go

On a traditional Intel CPU: ~64MKey/s/core, or 384 MKeys/s per high-end machine

(based on Intel X5460 3.16GHz “john” crypt(3) benchmark result from http://openwall.info/wiki/john/benchmarks)

number crunching for the masses

384 MKeys/s are ~28.5 bits in 1s.

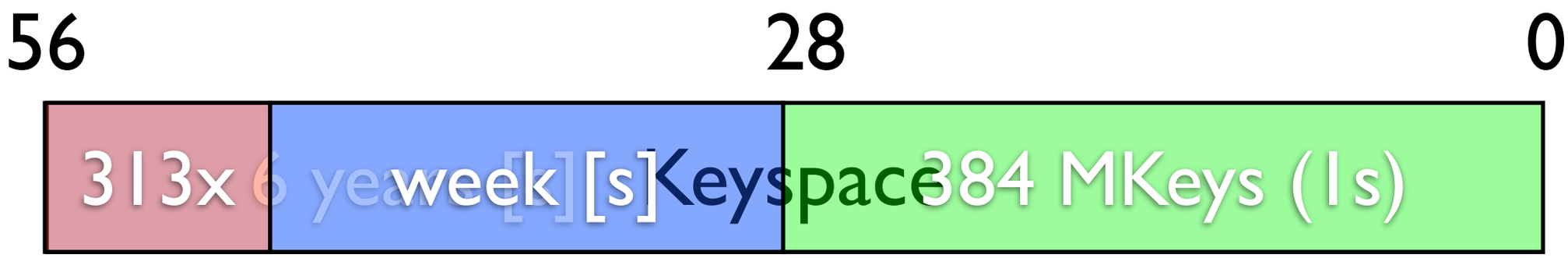
A day has ~2^16.4 seconds.

That's 28.5+16.4=44.9 bits for a day.

That's 2^11.1 days, or 6 years. Uh.

I want my key in a week.

So I need 313 machines.



number crunching for the masses

On Cell's SPU, an S-Box can be evaluated in 40 instructions on average for 128 bit in parallel.

That's equivalent to ~64 cycles per key guess, or 50 MKeys/s @ 3.2GHz.

That's 300 MKeys/s per PS3 (~28.1 bits in 1s).

number crunching for the masses

GPU are FAAAST!

But nobody uses them for DES cracking, so no good benchmarks

Estimated to be ~10x as a single CPU core

That'd be 640MKeys/s, or 29.2 bits in 1s.

number crunching for the masses

Device	bits/ (s*node)	$/node	nodes*weeks	$/Key/Week
PC	28.5	500	313	~150k
PS3	28.1	300	310	~93k (+Hack)
GPU	29.2	300	150	~45k
FPGA				

Two choices: Pipelined (unrolled) design vs. space-optimized design.

Pipelined design gives a result each cycle, vs. every 16 cycles, but is about 16 times as big.

Actually, non-pipelined design is less efficient.

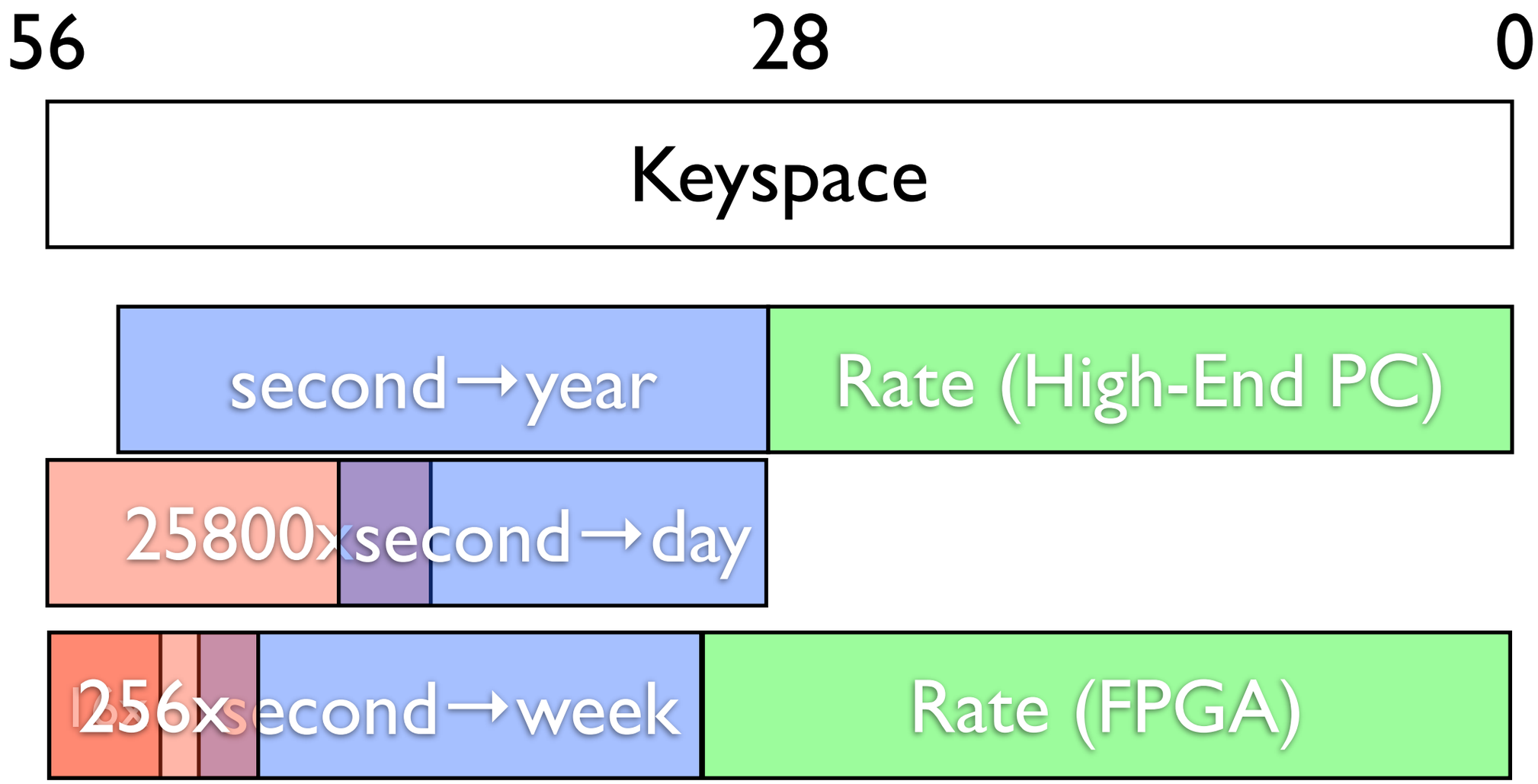
We can fit about 1-10 DES pipelined designs in an FPGA (depending on FPGA size)

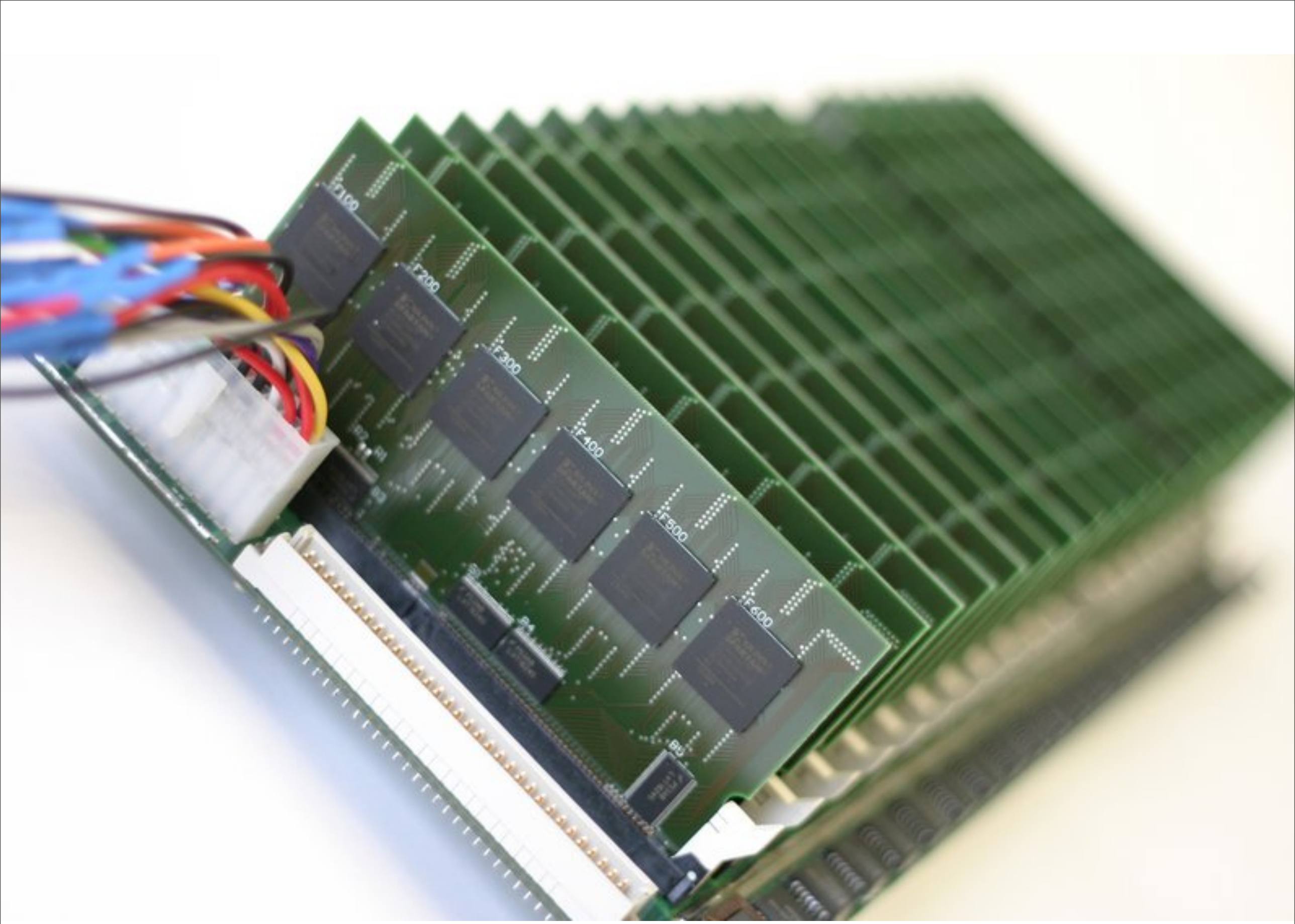
We can run them at about 100-200 MHz

That gives 100-2000 MKeys/s, or 26.5-30.9 bits in 1s.

One week with 60 of those big ones, or 1200 of the tiny ones.

d fpga number crunching for the masses





d fpga number crunching for the masses

Copacobana: Cost-Optimized Parallel Code Breaker

12.8 days to walk keyspace

120 Xilinx Spartan3-1000

$35 @120 pcs.: $4200 in FPGAs alone!

That’s cool, but…

d fpga number crunching for the masses

$4200+ for the key?

My goal was $1000 max.

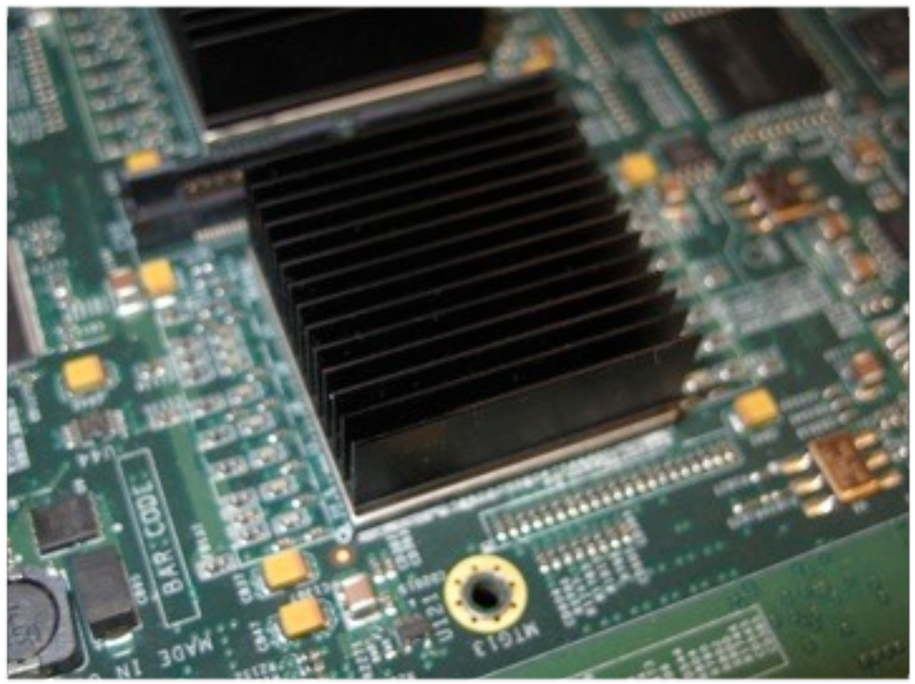
eBay FTW!

Search for “FPGA” and “for chip recovery”

Usually no documentation or decent pictures

Kind of hit-or-miss.

eBay FTW!

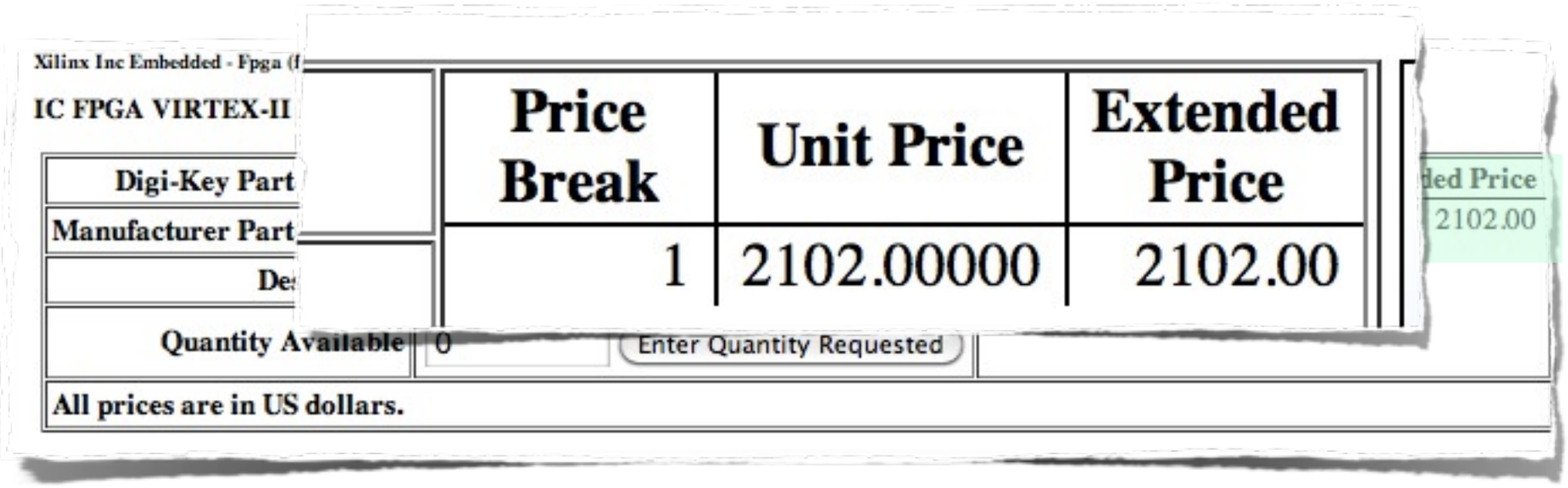


Artikelnummer	Artikelbezeichnung	Menge	Preis	Betrag
350327678358	3 Xilinx Virtex-II Pro XC2VP50 On PC Board	3	$55.00	$165.00

Zwischensumme:	US $165,00
Verpackung und Versand per Standard Int'l Flat Rate Shipping:	US $40,00
Gesamt	**US $205,00**

Xilinx Virtex 2 Pro

XC2VP50 List Price:


Xilinx Inc Embedded - Fpga (

IC FPGA VIRTEX-II

Digi-Key Part			
Manufacturer Part			
De			
Quantity Available	0	Enter Quantity Requested	

All prices are in US dollars.

Price Break	Unit Price	Extended Price
1	2102.00000	2102.00

ded Price
2102.00

We paid $50 for 3.

Ok, it’s an EOL’ed part.

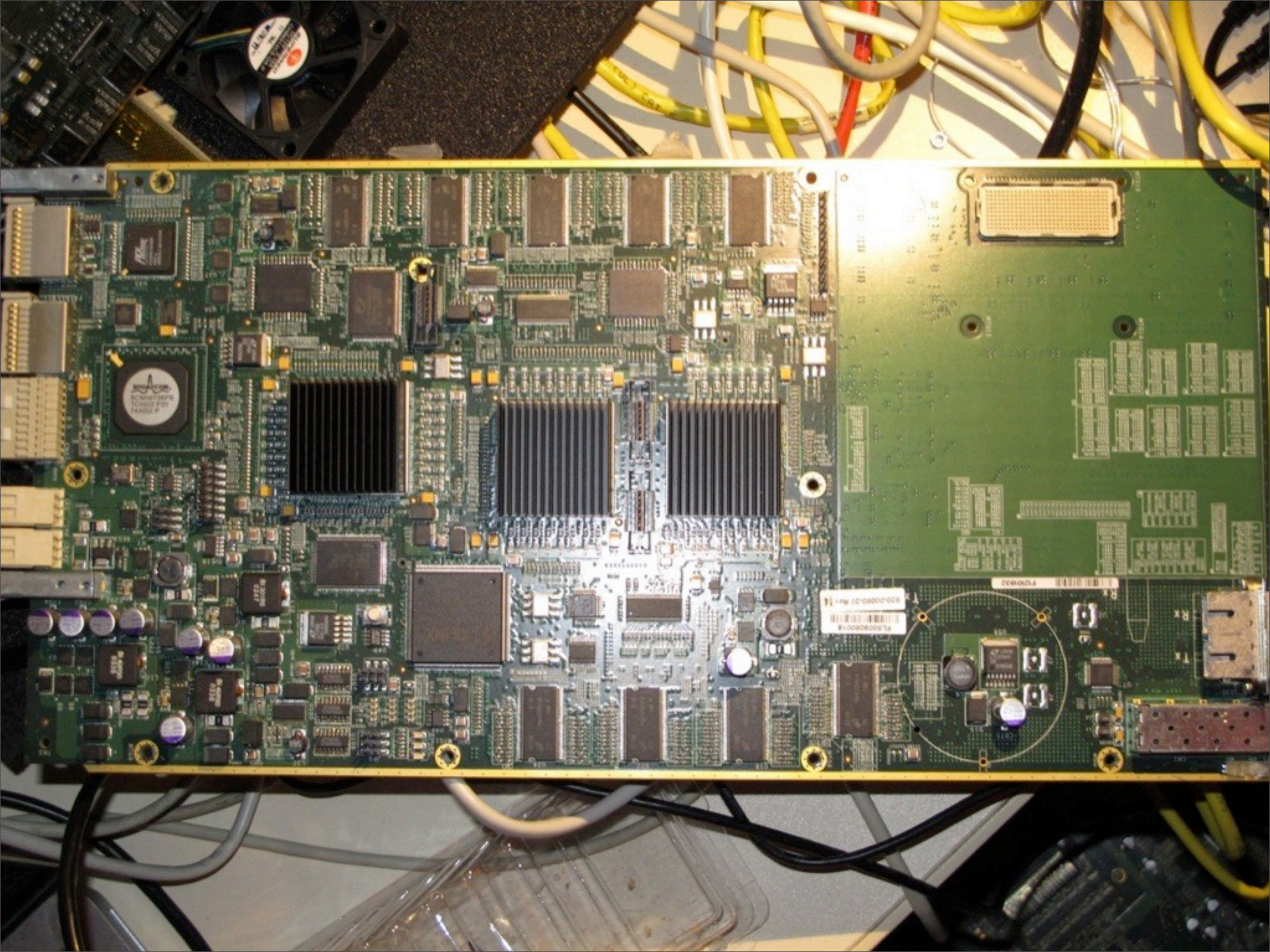
d fpga number crunching for the masses

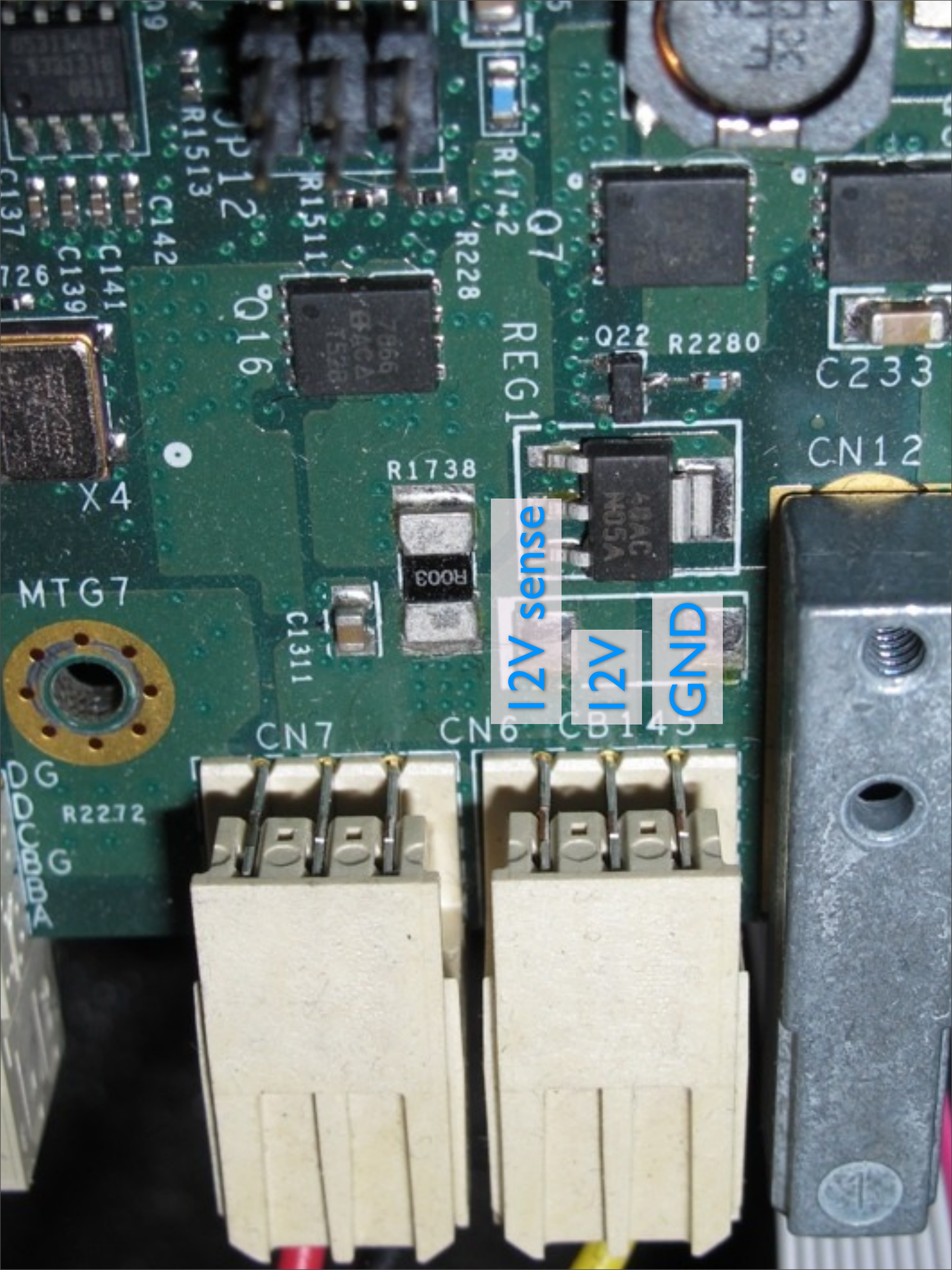
Remaining stock (~50) of these boards are in “good hands”

Plan is to bring them up, make people use them.

d fpga number crunching for the masses

1. Find Power.
2. Find JTAG.
3. Find clocks.
4. Profit.




JP12
R1513
R1511
C137
C142
C139
C141
726
R1742
R228
Q7
REG1
Q16
Q22
R2280
C233
CN12
R1738
X4
MTG7
C1311
R003
CN7
CN6
CB145
DG
R2272
G
12V sense
12V
GND

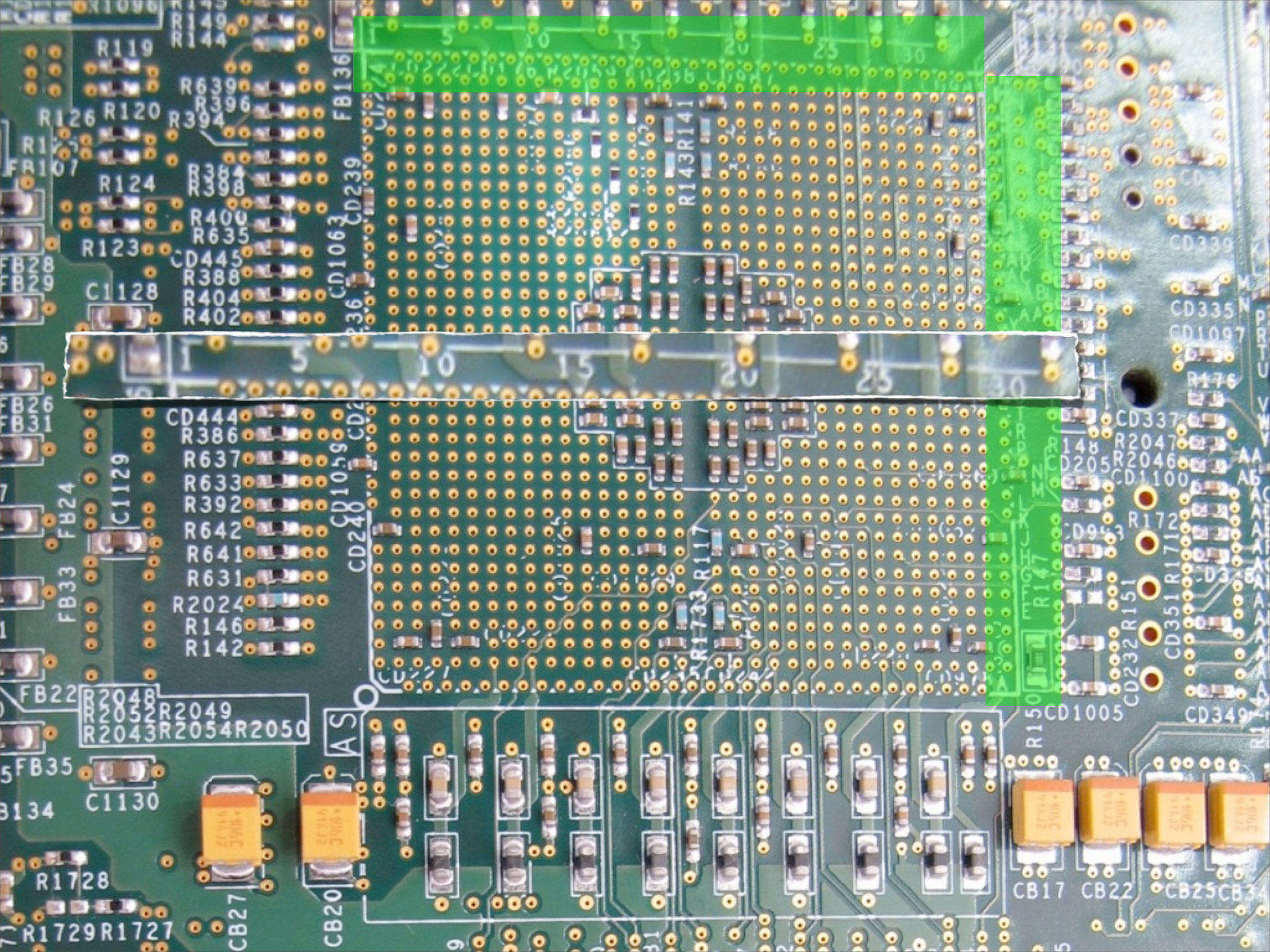
d fpga number crunching for the masses

1. Find Power. ✓

2. Find JTAG.

3. Find clocks.

4. Profit.


R119
R144
R149
R639
R396
R120
R126
R394
FB136
R124
R384
R398
R400
R635
R123
CD445
R388
C1128
R404
R402
CD1063
CD239
FB28
FB29
CD444
R386
R637
R633
R392
R642
R641
R631
R2024
R146
R142
C1129
FB24
FB33
CD240
R143R141
R117
R1733
FB22
R2048
R2052R2049
R2043R2054R2050
FB35
C1130
R1728
R1729R1727
CB27
CB20
CD335
CD1097
R176
CD337
R2047
R2046
CD1100
R172
R171
CD351
R151
CD232
CD1005
CD349
R150
CB17
CB22
CB25

d fpga number crunching for the masses

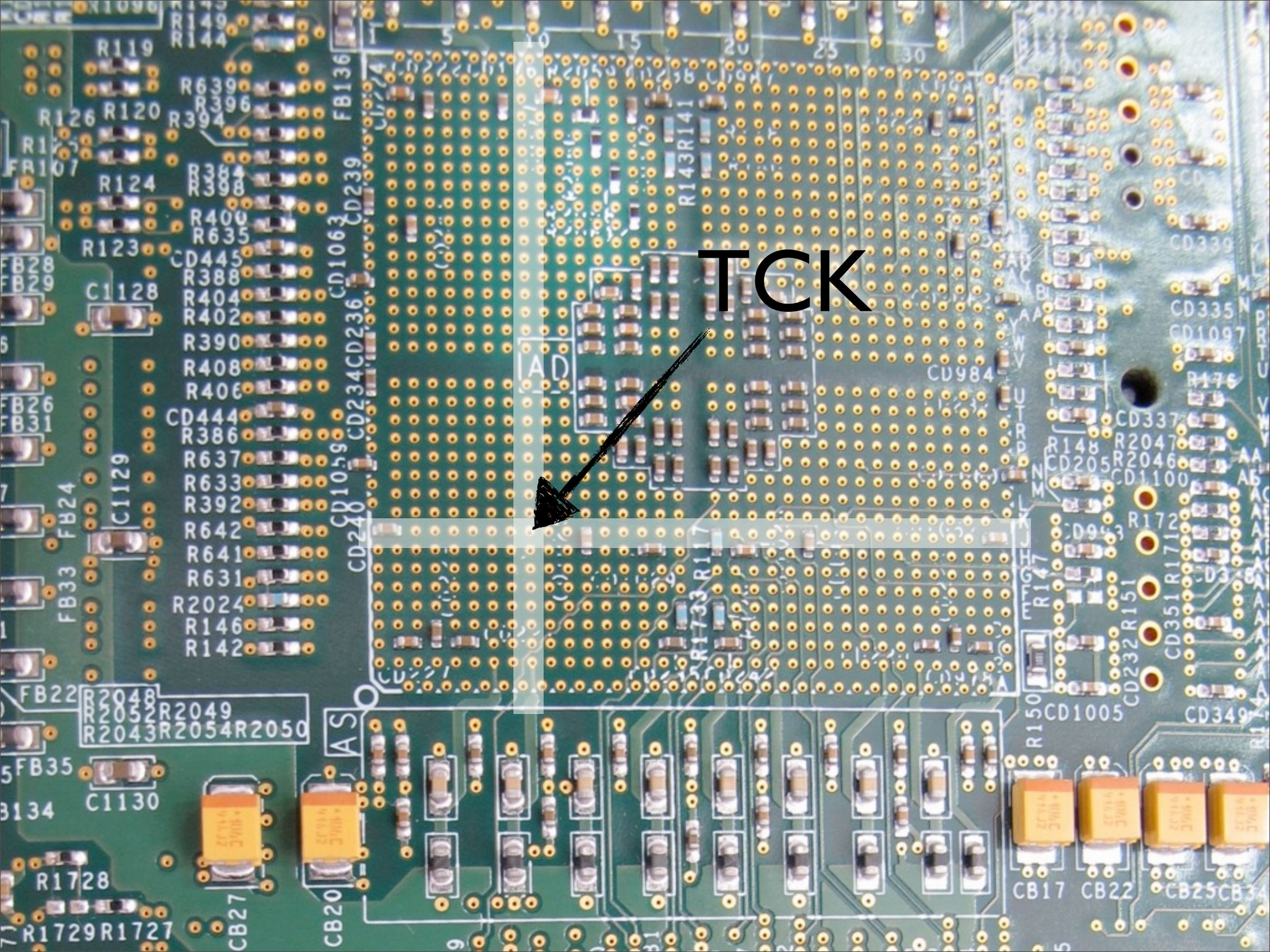
Just look it up in the datasheet:

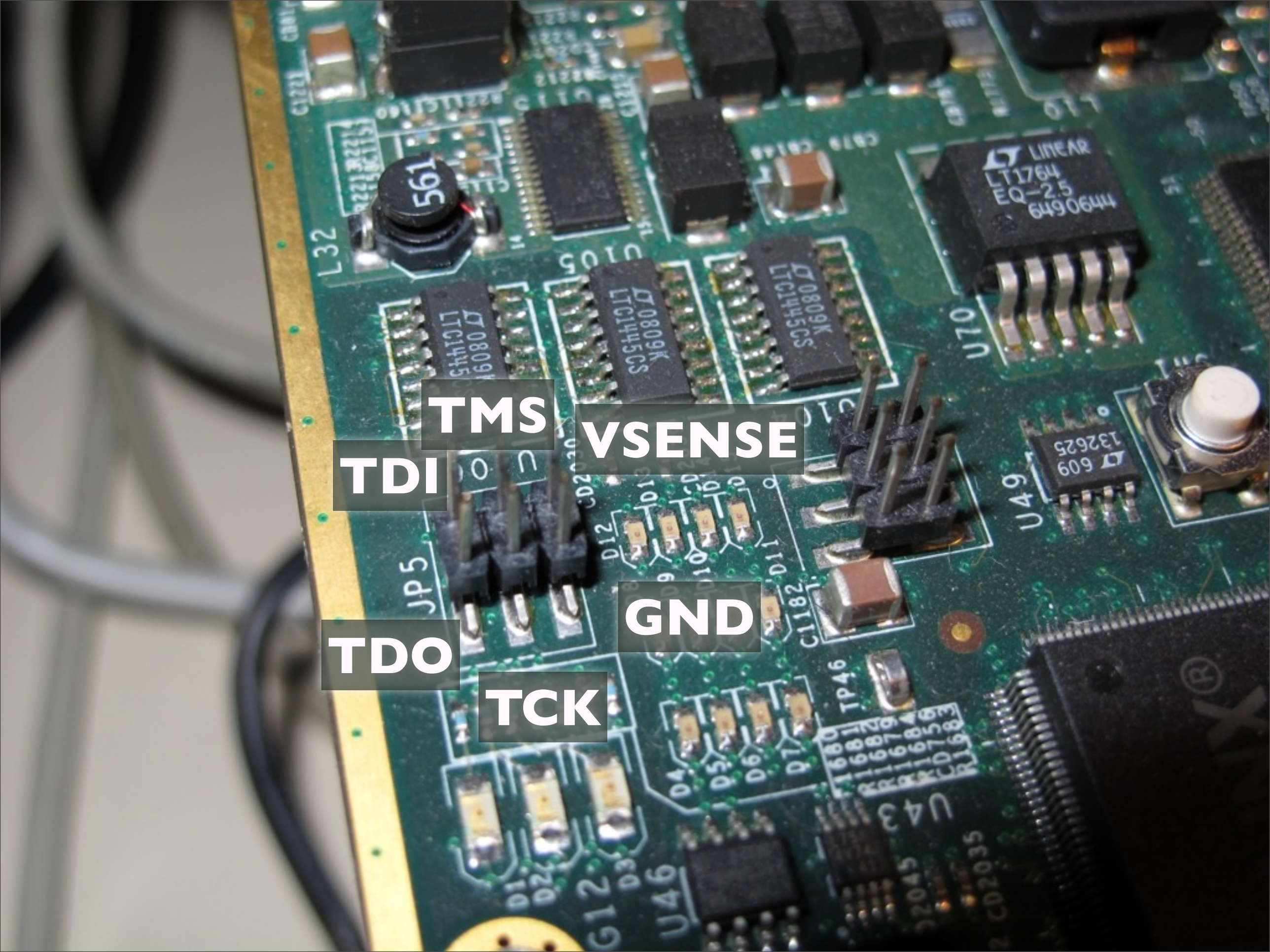
Table 10: **FF1152 — XC2VP20, XC2VP30, XC2VP40, and XC2VP50**

Bank	Pin Description	Pin Number	No Connects			
			XC2VP20	XC2VP30	XC2VP40	XC2VP50
7	VCCO_7	T23				
7	VCCO_7	U23				
N/A	CCLK	AE9				
N/A	PROG_B	J26				
N/A	DONE	AE10				
N/A	M0	AF26				
N/A	M1	AE26				
N/A	M2	AE25				
N/A	TCK	J9				
N/A	TDI	H28				
N/A	TDO	H7				
N/A	TMS	K10				
N/A	PWRDWN_B	AF9				

Locate the via

Use some tinfoil to trace the signal!

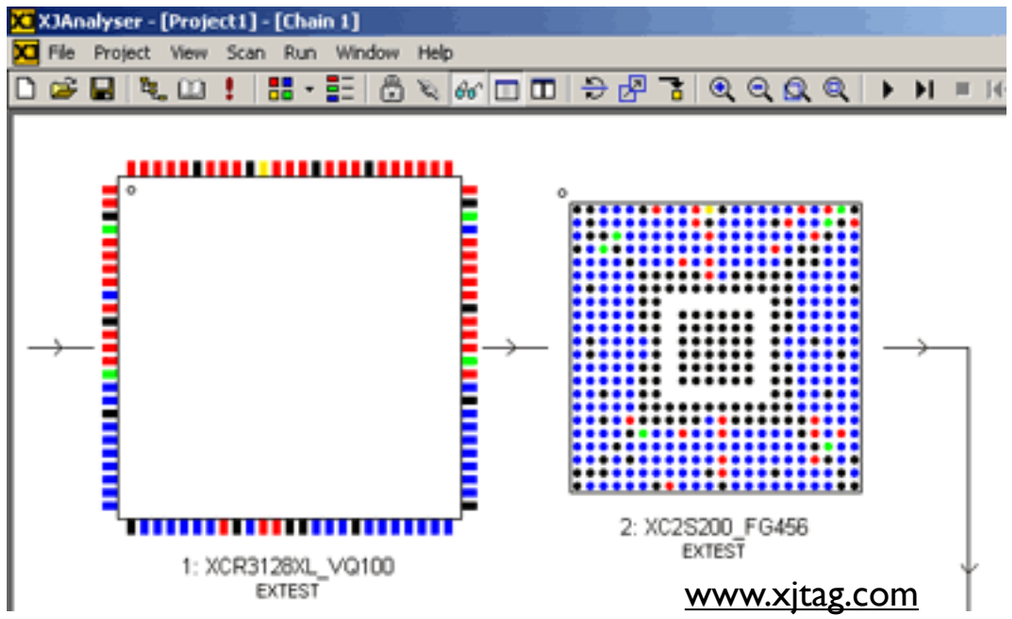

TCK


TMS
VSENSE
TDI
GND
TDO
TCK

d fpga number crunching for the masses

1. Find Power. ✓
2. Find JTAG. ✓
3. Find clocks.
4. Profit.

Method #1: Boundary Scan



Method #2: FPGAs usually have dedicated clock pins - check them!

d fpga number crunching for the masses

1. Find Power. ✓
2. Find JTAG. ✓
3. Find clocks. ✓
4. Profit.

d fpga number crunching for the masses

~20 Boards required.

20 * $50 = $1k - coincidence?

d fpga number crunching for the masses

Device	bits/ (s*node)	$/node	nodes*weeks	$/Key/Week
PC	28.5	500	313	~150k
PS3	28.1	300	310	~93k (+Hack)
GPU	29.2	300	150	~45k
FPGA	30.9	50/3	20	1k

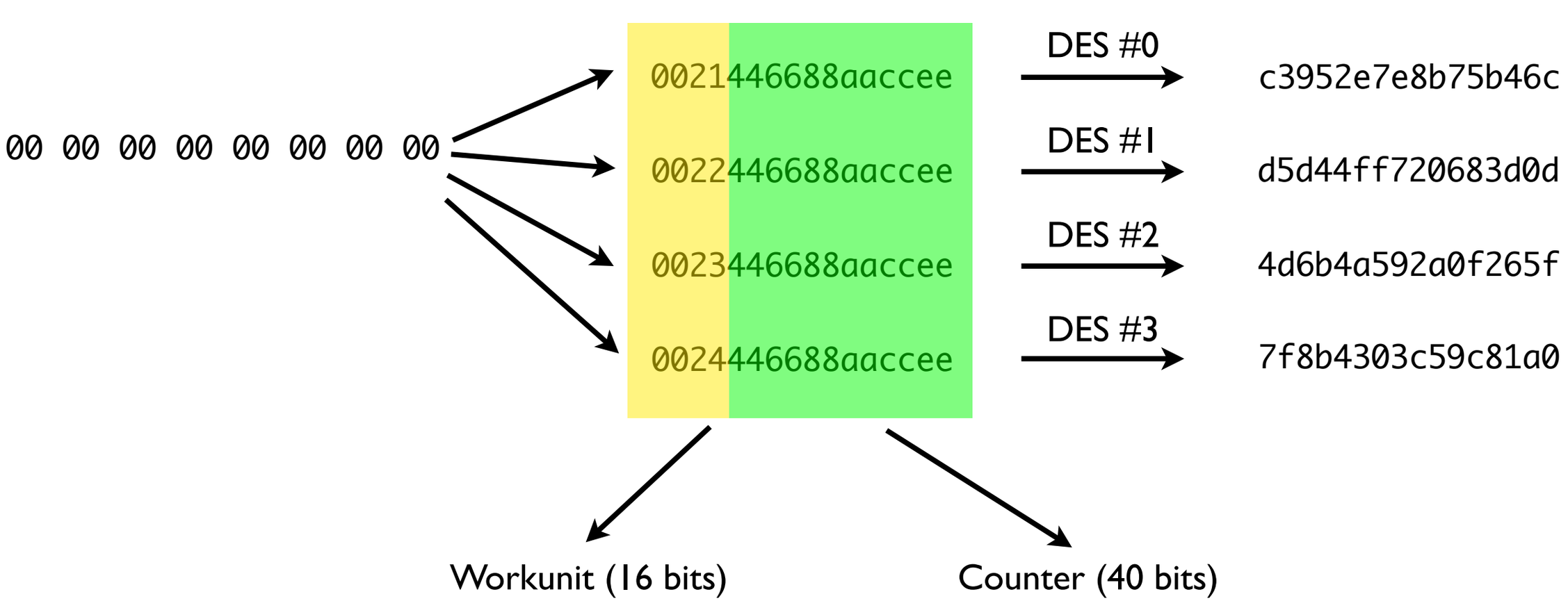
Multiple pipelined DES cores all encrypting the same plaintext.

Matched against multiple Patterns.

Each core searches a keyspace segment, lower bits will be identical.

d fpga number crunching for the masses

Plaintext Key Ciphertext



d fpga number crunching for the masses

	c3952e7e8b75b46c		4d6b4a592a0f265f	
		d5d44ff720683d0d		7f8b4303c59c81a0
0d3d6820f74fd4d5	✘	✘	✘	✘
06de6ff356237359	✘	✘	✘	✘
d5d44ff720683d0d	✘	✓	✘	✘
59732356f36fde06	✘	✘	✘	✘
f2c297df08b02b2a	✘	✘	✘	✘
f921900ca9dc8ca6	✘	✘	✘	✘
2a2bb008df97c2f2	✘	✘	✘	✘
a68cdca90c9021f9	✘	✘	✘	✘

d fpga number crunching for the masses

To get a sense of the numbers:

8 cores@200 Mhz w/ 16 patterns:

12.8 GByte/s encrypted data matched against 16 Ciphertexts.

That’s 204.8 GByte/s data to compare!

Naive approach:

16 patterns à 8 byte registers.

8 x 16 compare units.

Already 1/4th of the resources.

"Content-Addressable Memory" (CAM) is what we want - supply a ciphertext, get a match mask.

In Xilinx BRAMs, implemented as bitmasks, for example 256x64 bit.

Looks like a waste on first sight, but...

Using bitmasks allow for masked searching, or searching ASCII.

BRAMs are “free”. There are a lot of them. Just use them.

Frees up a lot of resources.

d fpga number crunching for the masses

Communication with the FPGA doesn’t need to be high-speed for brute-force.

Focus on reusability, (host-)cross-platform, simplicity.

Xilinx offer JTAG “USER” chains.

JTAG needed for programming, so you get communication for free.

xc3sprog (http://sourceforge.net/projects/xc3sprog/) supports a few different cables and a lot of Xilinx FPGAs.

Python wrapper added to provide two features to python:

- “Program Bitstream”
- “Scan USER chain”

Host programs FPGAs, configures matcher, set DES key prefixes

Matcher stops counting up on match, sets status bit.

Host polls matcher, if match is found, restart at key+1

Or actually at key+1-latency.

d fpga number crunching for the masses

Server to keep track of workunits.

Client requests a workunit, configures FPGA, delivers result.

BOINC?

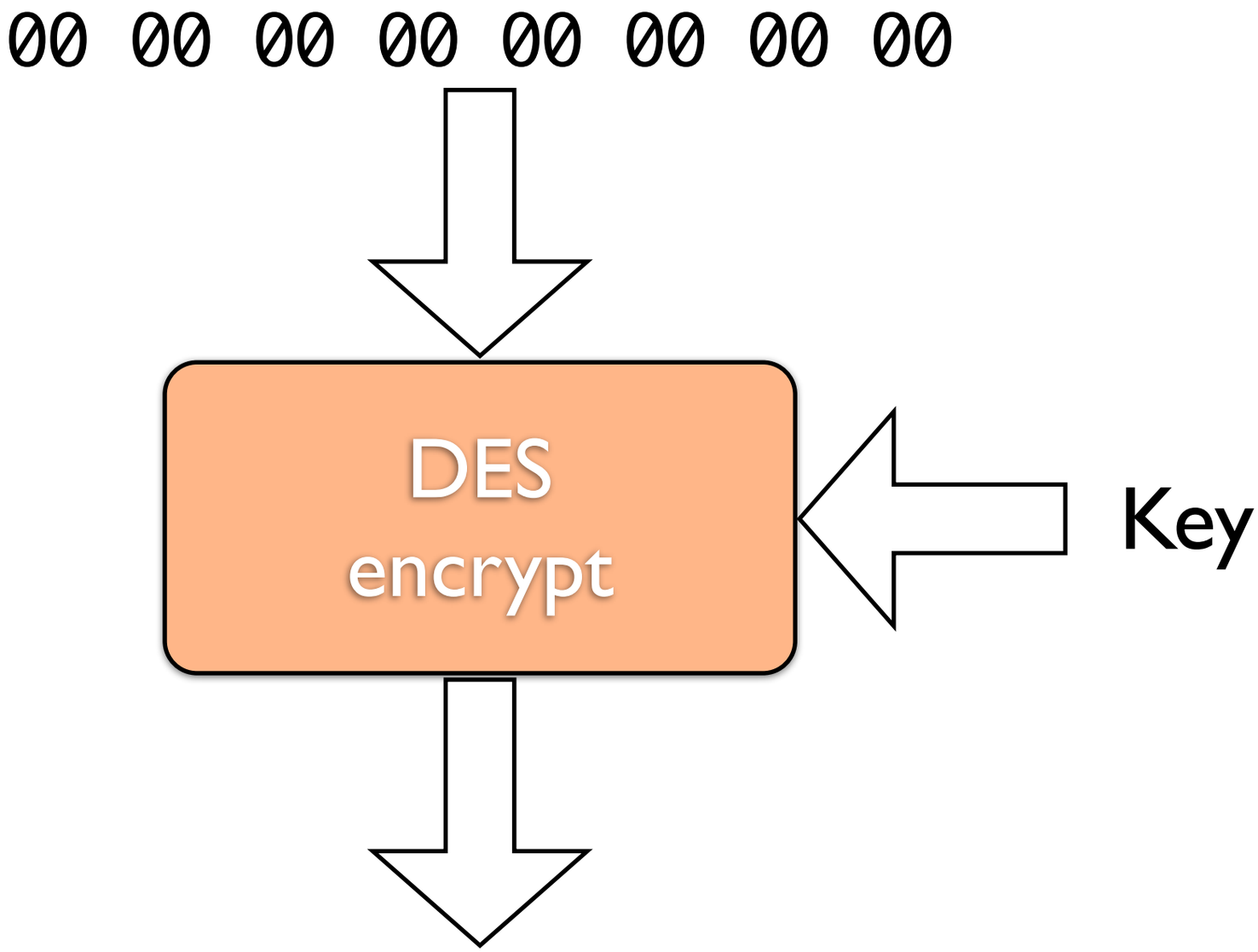
distributed fpga number crunching for the mass

Let's do a live demo!

~~... and wait a week.~~

... and skip some bits.

distributed fpga number crunching for the mass



distributed fpga number crunching for the mass

Roughly 1 hit per 40 bits of keyspace, i.e. 1/65536 of a real search.

Still takes about 1 hour on a high-end machine!

Let’s do it here, live.

distributed fpga number crunching for the mass

Prototype implementation on $150
Spartan-3A Starter Kit

Works, but still takes a few hours.

Let’s add some more FPGAs.

distributed fpga number crunching for the mass

DEMO: Dance2048 * 3

distributed fpga number crunching for the mass

"Crunchy" - A lightweight framework for easy development of distributed FPGA number crunching tasks.

https://github.com/tmbinc/crunchy

for the masses

The code scaled up with a single command line:

```
python build.py S3AStarter NR_CORES=1
```

```
python build.py Dance2048 NR_CORES=6
```

for the masses

Board: S3AStarter, ...

Project: descrack, ...

for building the bitstream (HDL, Meta)

server (workunit distribution, Python)

for client (communication with FPGA)

Parameters: NR_CORES, WORKUNIT_BITS, ...

for the masses

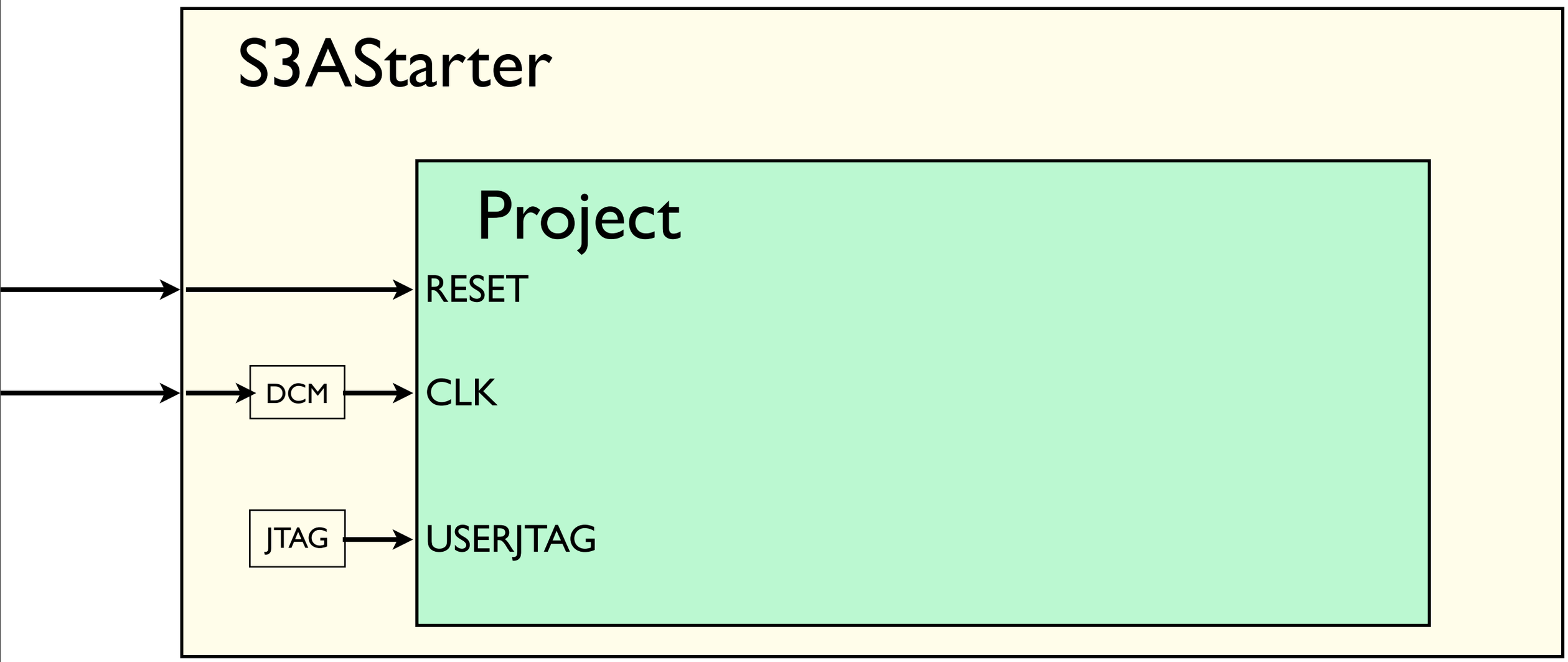
Board support provides:

Top-level HDL and constraints.

A number of defined interfaces to the “project” that the project may or may not use, like “CLOCK”, “LEDS”, “UserJTAG”.

Usually <100 lines of code.

for the masses



for the masses

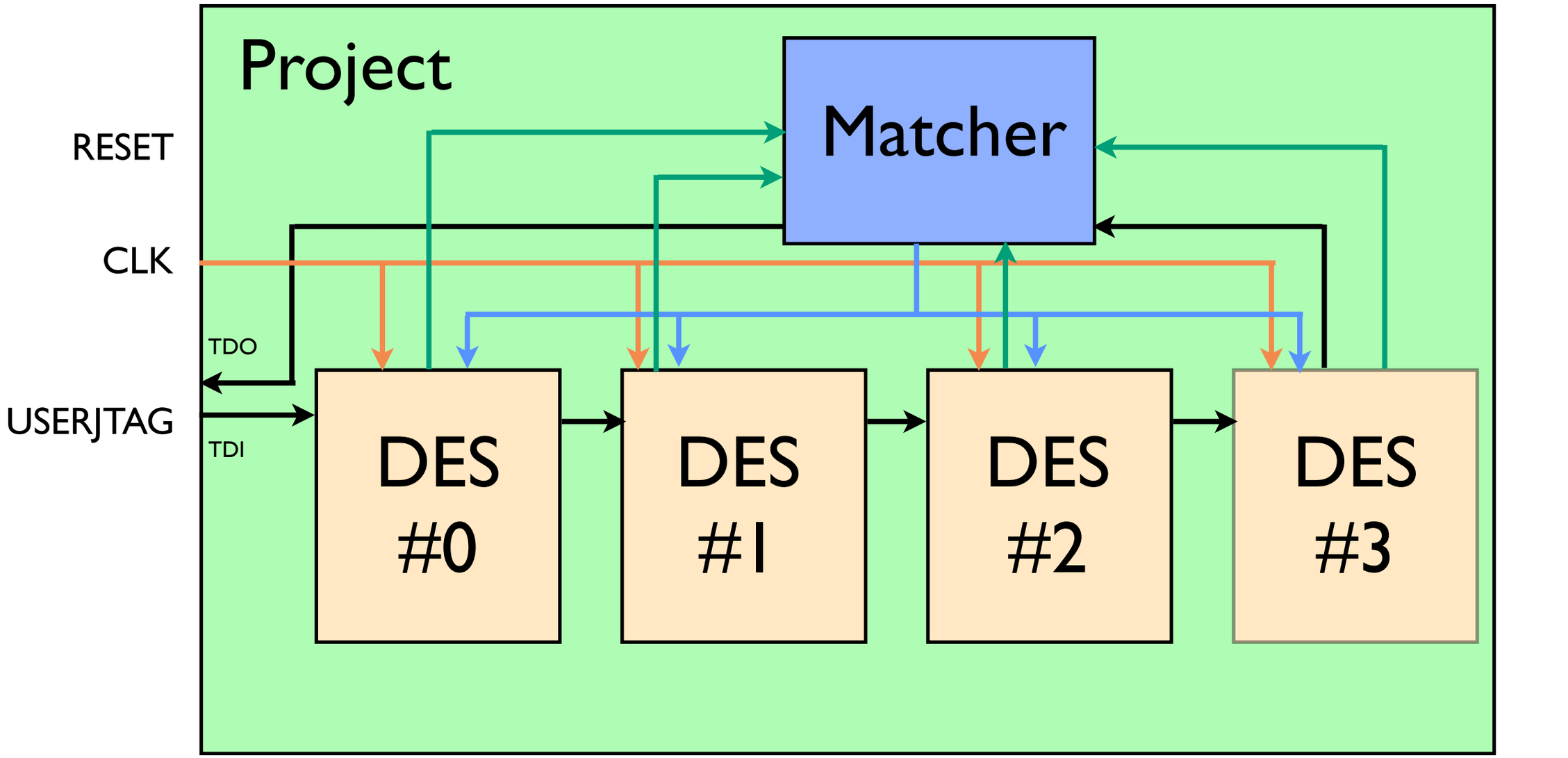
Project provides:

Parameterizable HDL using defined interfaces to plug into the board.

Client code to use abstracted board interface to talk to the FPGA.

Server code to provide workunits parameters.

for the masses



NR_CORES=4
NR_PATTERNS=16
WORKUNIT_BITS=38

Results
Counter
Clock
JTAG

for the masses

(board, project, parameters) define a bitstream

Server project definition includes list of supported boards with their parameters, distributes bitstreams

Client project definition can talk to any project instance.

for the masses

Prototype your brute-forcer core on a small FPGA!

Run it on a big cluster.

Or run it on a lot of devices!

for the masses

Some consumer hardware have FPGAs.

Some of them can be pretty big:

3G Femto Cells

Dreambox DM8000 Satellite Receiver

for the masses

Dreambox DM8000:

Embedded Linux DVB Receiver

Uses Xilinx Spartan 3E-500 for muxing between tuners and controlling CIs.

JTAG available as GPIO.

for the masses

1 Core per Device at max. ~100Mhz.

But: a lot of devices out there!

Users can donate their “free” FPGA power, for example in standby.

for the masses

DEMO DM8000

for the masses

FPGA improvements:

copacobana has a MUCH better DES core

More communication methods

Non-Xilinx devices

for the masses

Client/Server improvements:

Subscribe to project feeds, automatically take part in interesting projects.

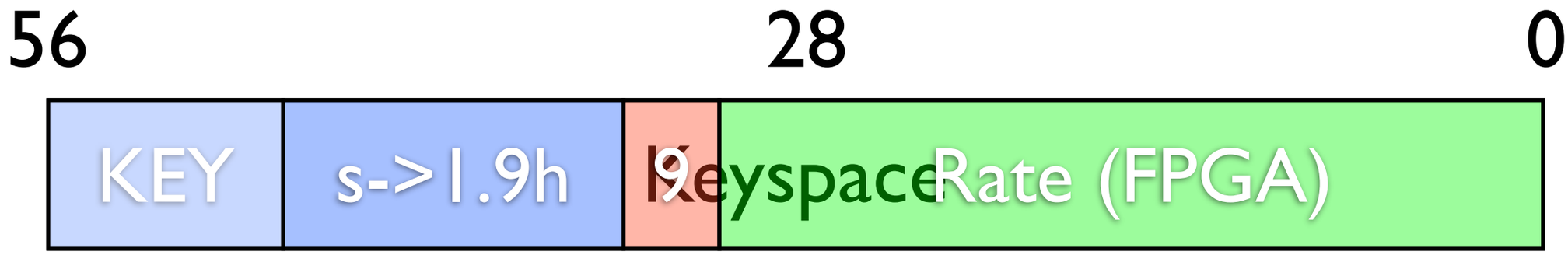
Security (Anti-Cheating and bitstream authentication)

for the masses

Results

for the masses

```
PROGRESS:
255133+57+6954 / 262144 (2.65 %)
PENDING:
[...]
RESULTS:
[...]
          276 0022446688aaccee
```



for the masses

Ciphertext (Pattern)	Plaintext with k
0d3d6820f74fd4d5	1010642f4858bbd1
06de6ff356237359	2b83b6f8295bddc9
d5d44ff720683d0d	0000000000000000
59732356f36fde06	ffffffffffffffff
f2c297df08b02b2a	764281af54fcc725
f921900ca9dc8ca6	0e5d79eb501a8e0c
2a2bb008df97c2f2	34c7a1849994274b
a68cdca90c9021f9	0a3cf4931bc484a2

We found the non-negated key!

C1 was all-zero and byte-reversed.

for the masses

```
...
0000760: 69 3c 3f 6a 30 65 66 33 33 66 65 30 6a 3f 3c 69  i<?j0ef33fe0j?<i
0000770: 03 56 55 00 5a 0f 0c 59 59 0c 0f 5a 00 55 56 03  .VU.Z..YY..Z.UV.
0000780: 66 33 30 65 3f 6a 69 3c 3c 69 6a 3f 65 30 33 66  f30e?ji<<ij?e03f
0000790: 65 30 33 66 3c 69 6a 3f 3f 6a 69 3c 66 33 30 65  e03f<ij??ji<f30e
00007a0: 00 55 56 03 59 0c 0f 5a 5a 0f 0c 59 03 56 55 00  .UV.Y..ZZ..Y.VU.
00007b0: 21 49 44 43 6f 64 65 30 00 00 5c 49 44 43 6f 64  !IDCode0..\IDCod
00007c0: 65 31 00 00 23 49 44 43 6f 64 65 32 00 00 61 74  e1..#IDCode2..at
00007d0: 65 73 74 70 69 63 00 00 62 73 65 63 5f 76 65 72  estpic..bsec_ver
00007e0: 00 00 6b 61 69 6a 79 6f 21 3f 00 00 66 4e 61 6f  ..kaijyo!?..fNao
00007f0: 6d 69 47 44 00 00 41 4b 45 59 43 4f 44 45 00 00  miGD..AKEYCODE..
0000800: 42 6b 65 79 63 6f 64 65 00 00 43 31 73 74 72 64  Bkeycode..C1strd
0000810: 66 30 00 00 44 31 73 74 72 64 66 31 00 00 21 49  f0..D1strdf1..!I
0000820: 44 43 6f 64 65 30 00 00 5c 49 44 43 6f 64 65 31  DCode0..\IDCode1
0000830: 00 00 23 49 44 43 6f 64 65 32 00 00 61 74 65 73  ..#IDCode2..ates
0000840: 74 70 69 63 00 00 62 73 65 63 5f 76 65 72 00 00  tpic..bsec_ver..
0000850: 6b 61 7a 75 68 69 73 61 00 00 66 4f 6b 61 62 65  kazuhisa..fOkabe
...
```

for the masses

1. Find Power. ✓
2. Find JTAG. ✓
3. Find clocks. ✓
4. Profit. ✓

for the masses

THANKS, and... FIND MORE KEYS.

tmbinc@elitedvb.net

http://debugmo.de/

https://github.com/tmbinc/crunchy