Glitching For n00bs
A Journey to Coax Out Chips' Inner Secrets

exide
REcon 2014, Montreal
Introduction
Background
Platforms
Example
Q & A
Introduction

- About Me
  - IT Monkey (Consultant) by day
  - Hardware Hacker by night

- Likes
  - Designing & reversing embedded systems
  - IC security & failure analysis
  - Arcade platforms
  - Automotive stuff

- Contact
  - Email: exide31337@yahoo.com
Background
What is Glitching?

- *Glitch* is a transient which can induce alteration in device operation
- *Electrical* glitching for purposes of this talk
  - Clock glitching
  - Voltage glitching
- Other glitching variants
  - Laser
  - Thermal
  - Radioactive
  - Etc.
A form of *non-invasive* attack on a device
- Doesn’t alter device package
- Doesn’t *permanently* alter operation
- Repeatable
- Surreptitious (no signs of tamper)
- Usually cheap
  - Don’t need expensive lab
  - Don’t need specialized microscopes
- Any background details can be helpful
  - To help narrow scope & strategy
Types of Attacks

- **Non-invasive examples**
  - Fault injection
    - Clock glitching
    - Voltage glitching
    - Thermal
    - Radioactive
  - Side channels
    - Power analysis
    - Timing attacks
    - Data remanence
  - Software
    - Code vulnerabilities
    - Brute-forcing a secret
    - Backdoors (undocumented instructions, debug interfaces)
Types of Attacks

- **Semi-invasive attack**
  - Device package altered
    - Decapsulation/milling vs. die alteration
  - Doesn’t *permanently* alter operation
  - Usually repeatable
    - Unless you leave the laser on too long
  - More expensive
    - Lasers, microscopes, chemicals, mill
    - May be beyond a single person’s budget
  - Provides background details
    - To help narrow scope & strategy
Types of Attacks

- *Semi-invasive* examples
  - Glitching
    - Laser
    - Flash
    - Thermal
  - Laser scanning
    - Unpowered vs. powered device
  - Optical imaging
    - Frontside / backside
    - Visible / infrared
    - Floorplan of structures & features
Types of Attacks

- **Invasive attack**
  - Device package altered
    - Decapsulation/milling & die alteration
  - Can render device non-functional
    - If careful, chip can still run
  - Some techniques are one-time
    - FIB workstation can create & undo edits
  - Can be costly
    - Decapping & readouts reasonable
    - Circuit edits prohibitive
  - Provides complete background details
    - Initial efforts can be used to find easier vulnerabilities
Types of Attacks

**Invasive examples**
- Decapsulation & delayering
- Memory (i.e., ROM) readout
  - Need to get to bottom metal layer
- Circuit edits
  - Etching
  - Deposition
  - Wire bonding
  - Purposely destroy traces or transistors
- Microprobing
  - Listen to busses
  - Inject signals on busses
Glitch Generation

- Methods
  - Clock divider
  - PLL
  - Poly-PWM
  - Polyphase
  - Etc.
Glitch Generation

- Clock divider
  - Use D flip-flops to divide-by-2 as needed
  - Feed MUX w/ nominal clock & faster glitch clock

Diagram:
- System CLK (48 MHz) flows through two D FFs.
- The clock is divided to 24 MHz and 12 MHz.
- The output is fed into a MUX.
- The glitch select signal is connected to the MUX.
- The DUT Clock is the output.
Clock divider

1x Freq. Clock

2x Freq. Clock

Glitch Clock

Glitch Select Line

Glitch Pulses
Glitch Generation

- **PLL**
  - Multipliers/dividers to generate arbitrary clocks
  - Fed from upstream clock (i.e., system clock)
  - Provides more clock choices
Step 1: Use multiple (i.e., 3) PWM blocks to generate clock signals with successively longer duty cycles.

Step 2: When XOR'd together, duty cycles allow creation of arbitrary start offset and pulse duration.
Glitch Generation

- Poly-PWM
  - Frequency is the same
  - Phase is fixed

- Duty Cycle 50%
- Duty Cycle 60%
- Duty Cycle 70%
- After 3-XOR

Glitch Pulse Glitch Pulse
Glitch Generation

- **Polyphase**
  - Generate multiple (i.e., 3) waveforms, each one phase shifted from the previous waveform
  - Frequency of waveforms is the same
  - Duty cycle is fixed

![Diagram showing the process of generating glitch pulses](image-url)
Polyphase

- Similar to Poly-PWM, but leading and trailing edges will combine to form twice the glitch pulses.

0° Phase Shift

45° Phase Shift

90° Phase Shift

After 3-XOR

More Glitch Pulses
PLL Dynamic Phase Shift

Implementing PLL Dynamic Phase Shifting in the Quartus II Software

The dynamic phase-shifting feature allows the output phases of individual PLL outputs to be dynamically adjusted relative to each other and to the reference clock without having to load the scan chain of the PLL. The phase is shifted by 1/8th of the period of the voltage-controlled oscillator (VCO) at a time. The output clocks are active during this dynamic phase-shift process.

To perform one dynamic phase-shift, follow these steps:

1. Set PHASEUPDOWN and PHASECOUNTERSELECT as required.

2. Assert PHASESTEP for at least two SCANCLK cycles. Each PHASESTEP pulse allows one phase shift.

3. Deassert PHASESTEP after PHASEDONE goes low.

4. Wait for PHASEDONE to go high.

5. Repeat steps 1 through 4 as many times as required to perform multiple phase-shifts.
PLL Dynamic Phase Shift

Figure 6. Timing Diagram for Dynamic Phase Shift

- SCANCLK
- PHASESTEP
- PHASEUPDOWN
- PHASECOUNTERSELECT
- PHASEDONE

PHASEDONE goes low synchronous with SCANCLK
Clock Glitching

- Momentary burst in frequency
- Timing-critical
  - Value of Program Counter
  - Offset of glitch within cycle
  - Duration of glitch
- Register/Flip-flop latches invalid data
  - Signals still propagating through combinatorial logic
  - Destination flip-flop suddenly clocked ahead of schedule
Clock Glitching

- Instructions replaced w/ mutated opcode
  - i.e., turn a JSR into an ADD
  - Like patching a software binary
  - Instruction is NOT skipped
    - Program Counter doesn’t just jump ahead 2 locations
- If security fuse logic is slower, fuses can latch advantageous values
Clock Glitching

- Setup & hold-time of flip-flop out of spec
Voltage Glitching

- Momentary reduction in supply voltage
- Drop supply to/below transistor switching threshold
- Increases propagation delay
  - Decrease in $V_{CC}$, which decreases $V_{GS}$ and $I_{DS}$
  - Lower drive strength causes slower rise times & more delay
- Timing-critical
  - Value of Program Counter
  - Offset of glitch within cycle
  - Duration of glitch
Voltage Glitching

- Alter values at memory sense-amplifiers during read operation
  - i.e., Flash, EEPROM, RAM, etc.
  - Corrupt data latched onto address or data bus
- Security fuse logic can latch corrupt values
  - Due to operation at/below switching threshold
NOT throwing random voltage sags/surges at IC and “seeing what sticks”
- Respect *Absolute Max VCC & VCC\textsubscript{\text{io}}* ratings
- Some 74-series can handle insane swings (+/- 12V)
  - Not common, and always w/ current-limited condition

NOT randomly jarring clock frequency to wild extents

NOT skipping instructions
- Replacing/mutating them
Misconceptions

- **Timing-critical**
  - Target a cycle at specific point in program
  - Start/offset of glitch pulse within cycle
  - Duration of pulse

- Unless chip stuck in a loop, random glitching usually counterproductive
  - Instruction search space smaller
  - Popping loop more likely
Outcomes

- Make CPU replace impeding instruction(s)
- Truncate cryptographic operation / key
- Linear code extraction
  - I/O channel to TX commands / data & RX data / clues
  - Von Neumann dumps can be exhaustive
  - Provides clarity on device internal operation
  - Secrets revealed (crypto keys, serial #, known S-box, etc.)
  - Scour for software vulns
- Bypass bootloader-enforced check(s)
  - Stop MMU, page tables, etc. from initializing
- Prevent lockout counters from rolling
- Erase security fuses / lock bits
  - But keep Flash/EEPROM intact
  - Just read-out device w/ parallel/serial programmer
Targets of Interest

GENERAL-PURPOSE

- CPUs
- Microcontrollers
- Memories
- DSPs

CUSTOM

- FPGAs
- ASICs

SECURITY-ENHANCED

- SIM cards
- Smart meters
- Military devices
- Banking / “Chip & PIN” cards
- Pay TV
- Transit/metro passes
- Automotive sector
  - Keyless entry
  - Immobilizer
  - V2V & V2I
Countermeasures

- CPUs which halt/trap on invalid instruction
  - Mutated instruction may still be valid
- Erase volatile memory on startup / reset
  - Like HeartBleed didn’t, minimize # of copies of important secrets
  - Wipe between iterations of routine (if possible)
- Clocking
  - Run off internal oscillator
  - Use asynchronous logic
  - Use aperiodic / random clock period generator
Countermeasures

- Supply voltage
  - Glitch / brownout detection
  - Low-pass filter
  - Reset / halt / wipe device
- Many general-purpose devices have little or no designed-in protections
- AVR, PIC, MSP, etc. have memory protections
- Modern smartcards have extensive protections
  - Glitch detectors
  - Random / asynchronous internal clock w/ dummy cycles
  - Dual lockstep cores sanity-checking one another
Platforms
Arrow LPRP + Breadboard

Altera Cyclone III FPGA
MIPS32 CPU
3-PWM Clock
3-Phase Clock
16550 UART
SRAM/Flash Control
Output MUXes

Level Shifting Signal Conditioning
Solderless Breadboard

20 x 2 I/O Ribbon Cable

74LV125 +3.3V

74LV125 +5V

DUT

Pull-Up Pot

+5V

+3.3V
Arduino

ATmega 328P

USB

FT232

DUT

74LV125

+5V

Analog MUX
Photo-Etched PCB
Photo-Etched PCB
Photo-Etched PCB
Professional PCB

- Analog MUX
- Op-Amp Buffer
- Inverter
- AND
- AVR
- D Flip-Flop
Cheap & Dirty Logic Analyzer

- Altera SignalTap II
  - Can select almost any internal signal, net, bus
  - External I/O pins
  - Can increase sample depth by using more LEs
  - Plenty of trigger options
    - Simple – low, high, edge, etc
    - Advanced – chained events, segmented capture, etc
  - Export data as plaintext, image, other formats
  - Equivalent to Xilinx ChipScope
# Cheap & Dirty Logic Analyzer

<table>
<thead>
<tr>
<th>Instance</th>
<th>Status</th>
<th>LEs: 983</th>
<th>Memory: 126976</th>
<th>Small: 0/0</th>
<th>Medium: 64/66</th>
<th>Large: 0/0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>auto_signals</td>
<td>983 cells</td>
<td>126976 bits</td>
<td>0 blocks</td>
<td>16 blocks</td>
<td>0 blocks</td>
</tr>
</tbody>
</table>

## Log Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_phase:inst1:phasestep</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>has:inst1:phasetupdown</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m_dyn_phase:inst1:reset</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9_phasecounterselect</td>
<td>1h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dut_pll:inst9:phasedone</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dut_pll:inst9:phasestep</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dut_pll:inst9:phasetupdown</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dut_pll:inst9:scanclk</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Data Log

- **signal_set**: 2012/12/26 01:55:54 #0
- **trigger**: 2012/12/26 01:55:54 #1

- **log**: 2012/12/26 01:55:54 #2 - 9 cycles @ 48MHz between glitch request and Vcc actually dropping

- **signal_set**: 2014/05/27 06:03:12 #0
- **trigger**: 2014/05/27 06:03:12 #1

- **log**: 2014/05/27 06:03:12 #2 - dut_pll phase shift programming via state machine

- **log**: 2014/05/27 06:48:52 #0 - as above, made PhaseStep wait High one more cycle
Example
Example

- Victim IC
  - Secure microcontroller
    - Not sure what architecture
  - Pairs with partner device
  - Accepts data, encrypts/decrypts it with key(s), returns data to partner
  - Starting from blackbox
    - Not sure what datasheet(s) to look for
      - Even if device known, datasheet(s) may not be public
Example

- Start probing device pads
  - Initial sweep w/ multimeter
  - Revisit interesting pads w/ oscilloscope
- One pad appears to speak slow-ish serial protocol
  - Capture & transcribe beginning of waveform from scope
  - One pad, thus half-duplex conversation
Example

- Rig up sniffer board to MITM the victim-to-partner conversation
  - Level shifting
  - Buffering
- Use SignalTap to digitize conversation
  - Export waveforms as plaintext
  - Parse into binary data
- ISO 7816 APDU header matched!
Example

- Bolt UART to FPGA / soft-CPU
  - Allows for HW framing of TX & RX data w/ victim
  - Don’t need to screw around bit-banging
- Use unrelated Altera JTAG UART to talk w/ soft-CPU
  - Otherwise, separate programming vs. data cables
  - PC can talk to victim via soft-CPU
- Have PC speak ISO 7816 w/ victim via FPGA
Example

- ISO 7816 header has *length* field
  - Hunch that victim compares *length* to max it’ll allow as buffer input
    - When storing command to RAM
- Issue too-long ISO 7816 commands to victim
  - Too long, but computed to be otherwise valid
  - Observe error response
- *Get ready to glitch!*
Sucker Punch!
One-Two Punch!
Example

- Start glitching!
  - In this case, clock glitching
  - Glitch during suspected victim command handler
  - Try different pulse offsets & durations
- Milestone reached when victim responds to too-long command correctly
  - Length check bypassed
- Make best guess at victim architecture
  - Motorola 6805-based
  - Intel 8051-based
  - Etc.
Example

- Pad more and more bogus data at end of command
  - Until victim crashes or does something weird
    - Stack smashed (return address overwritten)
    - Might be hard to notice if watchdog present
  - Distance to stack pointer now known
- Using guess at victim architecture
  - Write minimal code that tries to write to low-addressed special registers
    - PORTx, PINx, DDRx, etc.
  - Keep trying candidate return addresses
Victim Memory Layout

- Milestone reached when victim output pin(s) change
  - Code execution confirmed
  - Architecture guess confirmed
  - Probably Von Neumann or Modified Harvard
Example

- Write code that loads dummy ASCII byte to desired register / memory, then sweeps jumps into address space
  - Could be unwieldy if large address space
- Milestone reached when ASCII byte pops out victim’s serial pin
  - Victim serial TX routine address found
Example

- Write code that loads data at each sequential address location into register, then jumps to serial TX routine address
  - Be prepared to empty the FPGA UART’s RX FIFO quickly & regularly
- Cause it’s gonna get clogged up with 9000 tons of **WINNING!!!**

(a.k.a. Code & Data Space)
Epilogue

- Try to figure out memory map
  - Analyze dump for mirroring of address space
  - Try poking values at different addresses
    - See if address is mutable or not

- Back in familiar territory
  - Disassemble
  - Search for secrets
  - Discover code vulnerabilities
Conclusions

- Electrical glitching can be a viable attack vector against a variety of ICs
  - Except some modern purpose-built security ICs
- Cheap to perform
- Don't need a big laboratory
- Non-destructive in nature
- Another tool in the reverser's arsenal
  - Can provide results where other approaches fail
Q & A