2.2 RF Spy on USB and FPGA Bitstream Programming

Final Presentation

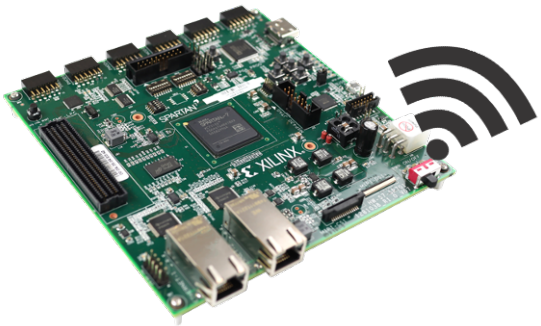
Austin Zhu & Yu Jun Shen

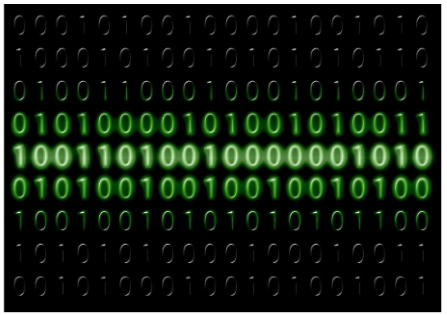
EENG 428 / ENAS 968 · Spring 2022 · Yale University

May 4, 2022

Research question

Can radio-frequency (RF) emanations from the connections of a FPGA give away its bitstream, and thus the programming design?





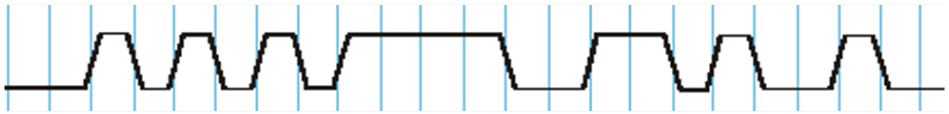
A security issue from physical side-channel attacks in nonlocal FPGAs

Contents

- Theory behind RF emanations of FPGA
- Methodology
- Preliminary data
- Signal analysis
- Conclusion and security implications

Source of RF emanations

- Digital signals have rising and falling edges which cause sharp electromagnetic emanations by Faraday's Law.
- If accurately captured and profiled, the bitstream may be reconstructed.
- Similar RF-based attacks have been mounted against computer keyboards, and power-line analysis of USB ports.



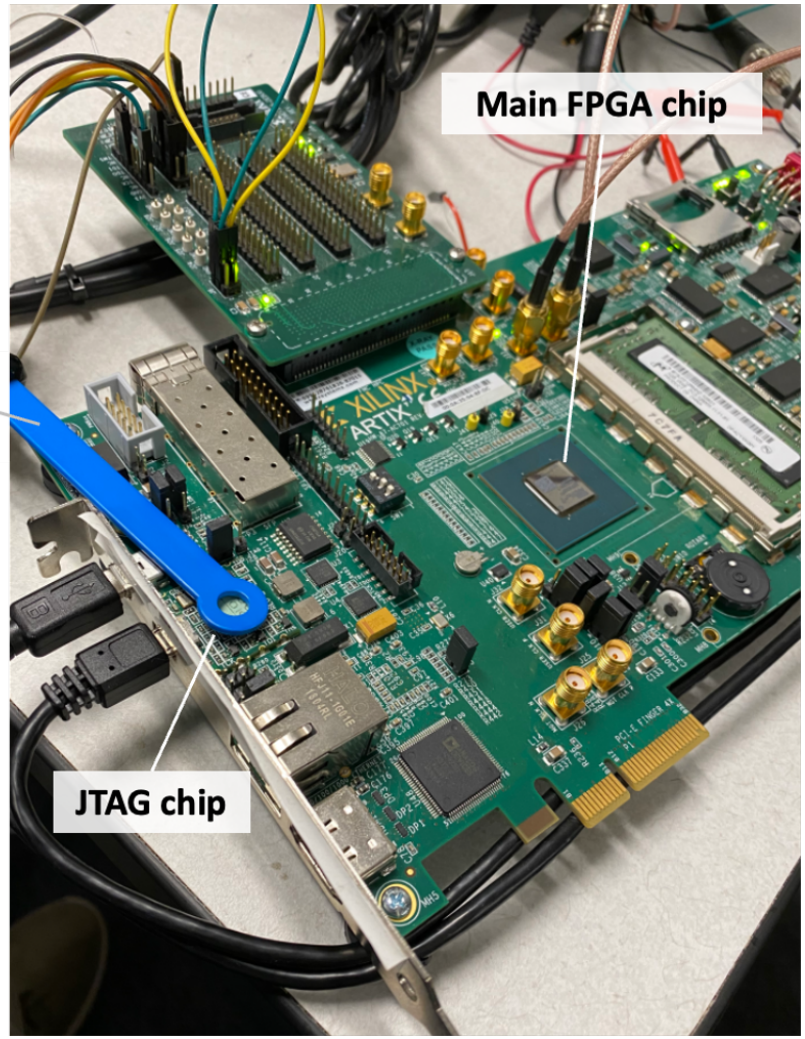
Theory - JTAG and USB

- USB cable - shielded, difficult to get usable RF signals
- JTAG operation: Typical frequencies from 6 MHz clock pulse, so harmonics possible
- OpenOCD process to change the rates of operation
- Ring oscillator percentage

Bitstream in theory has unique formats per vendor, may even be used as a digital fingerprint.

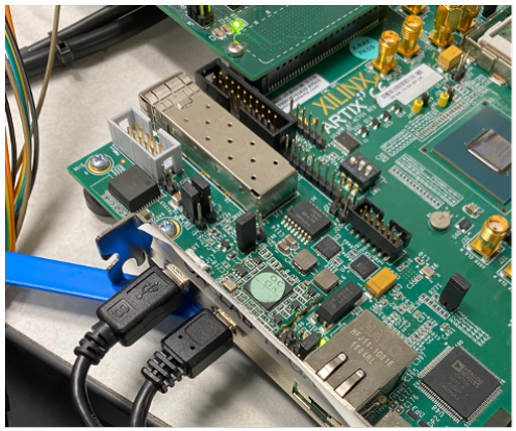
Methodology: lab setup

RF probe placed above AC701-02 FPGA JTAG chip, connected to oscilloscope

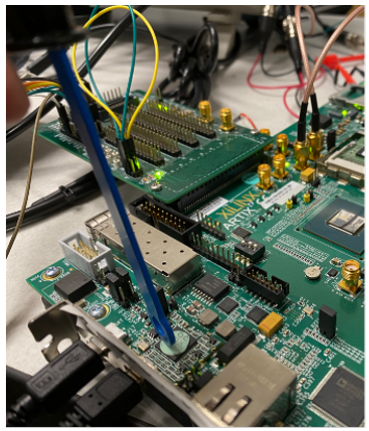


Methodology: lab setup (contd.)

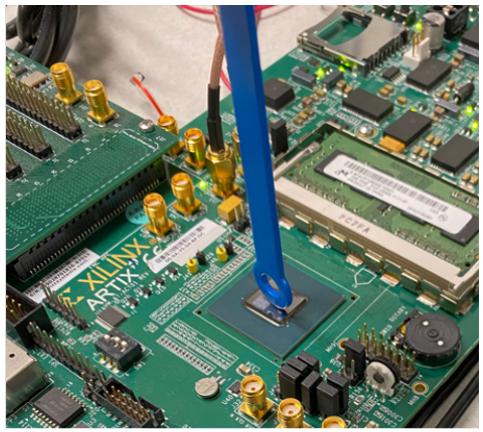
Alternative positions were experimented, but best results found with probe flat above the JTAG chip



Below the JTAG chip



Vertical orientation



On main FPGA chip

Also experimented with different RF probes in the lab, but no significant difference observed.

OpenOCD

Changing frequency of communication

- Used OpenOCD to directly load in bit files
- JTAG frequency can be modified from 6MHz to 30MHz
- Used number of ring oscillators programmed to change utilization percentage

Methodology: Quantifying signal energy

To quantify the measured frequency domain signature, we wrote a Python code to parse the many CSVs collected for a single energy per timestep reading.

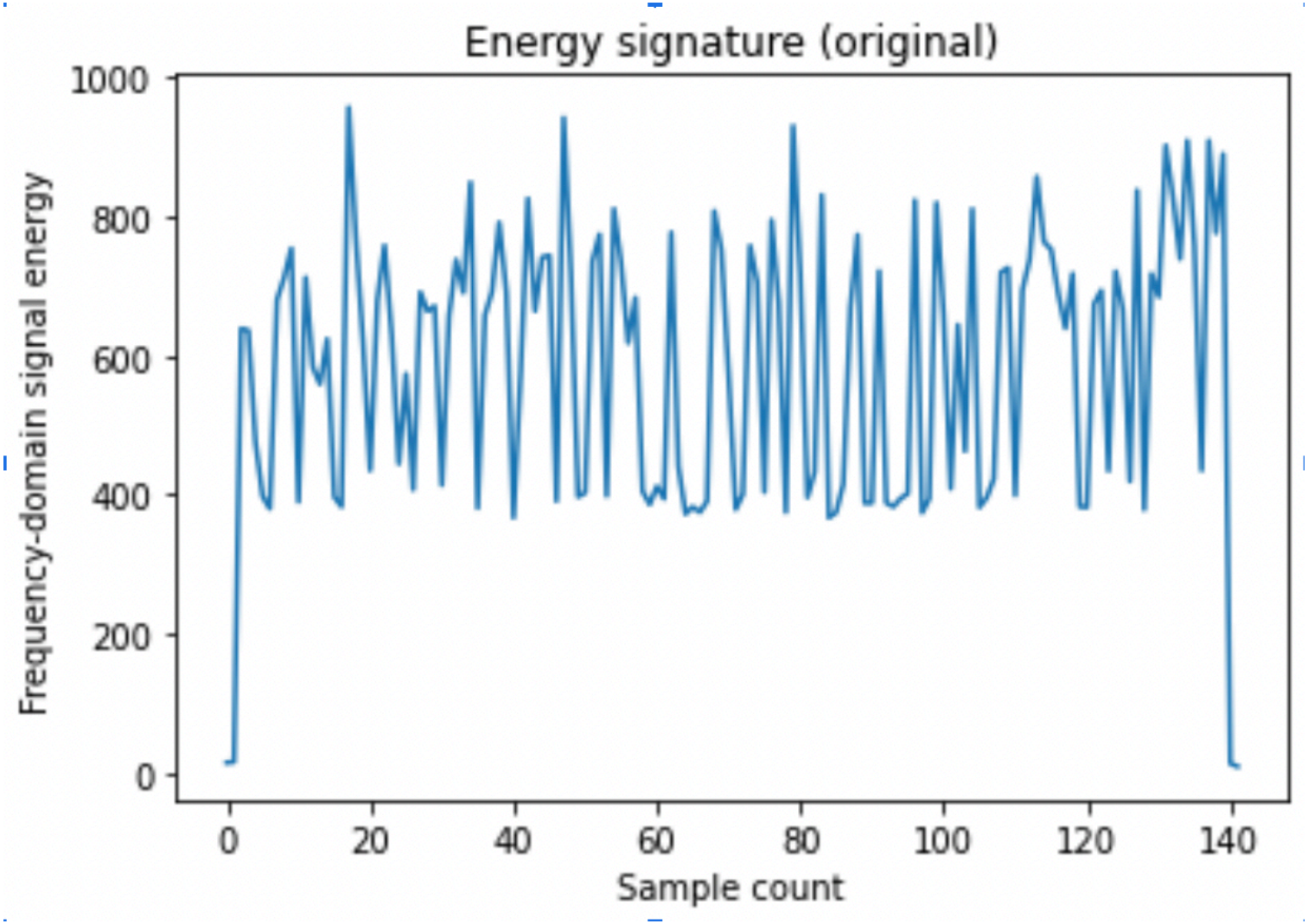
```python
def read_values(file_name, noise_file):
    """
    Reads in the frqeuency spectrum information of a csv with file_name
    Returns two lists - frequency, magnitude
    Assumes data starts at row 22
    Subtracts from noise_file
    """
    # get noise
    file_n = open(noise_file)
    data_n = file_n.readlines()
    noise_mag = []
    noise = data_n[-1].split(',')
    for row_count_n in range(len(data_n)):
        if row_count_n >= 22:
            f_n = data_n[row_count_n].split(',')
            if len(f_n) == 2:   # ignore /n row
                noise_mag.append(10**(float(f_n[1])/20))
                #print("Noise:", round(10**(float(f_n[1])/20), 4))
    file_n.close()

    file = open(file_name)
    data = file.readlines()

    freq = []
    mag = []
    for row_count in range(len(data)-1):
        if row_count >= 22:
            f = data[row_count].split(',')
            if len(f) == 2:   # ignore /n row
                freq.append(float(f[0]))
                mag.append(10**(float(f[1])/20))
                #if float(f[1]) > -63:
                #    mag.append(10**(float(f[1])/20))
                #else:
                #    mag.append(0)
    file.close()

    for idx in range(len(mag)):
        #print("Before:",mag[idx], " after: ",mag[idx] - noise_mag[idx])
        mag[idx] = mag[idx] - noise_mag[idx]    # subtract linearized version

    return freq, mag

def read_binary_file(fileName):
    """
    Read binary bitstream files
```

$$E(t) = \sum_{i=1}^{N} |A(f)|^2 \, (\Delta f)$$

- $E(t)$: Energy value
- $|A(f)|^2$: Amplitude from spectrum
- (Δf): Frequency spacing

Methodology: Isolating Data

Use energy data to isolate area of interest (81.45 Percent Utilization)



Bitstream File Analysis

How we parsed the bitstream file

findings

1) Re-sample the bitstream to match number of bins in the oscilloscope spectrum
2) Calculate the number of transitions (0->1 or 1->0) in each bin.
3) Transition <-> RF signature

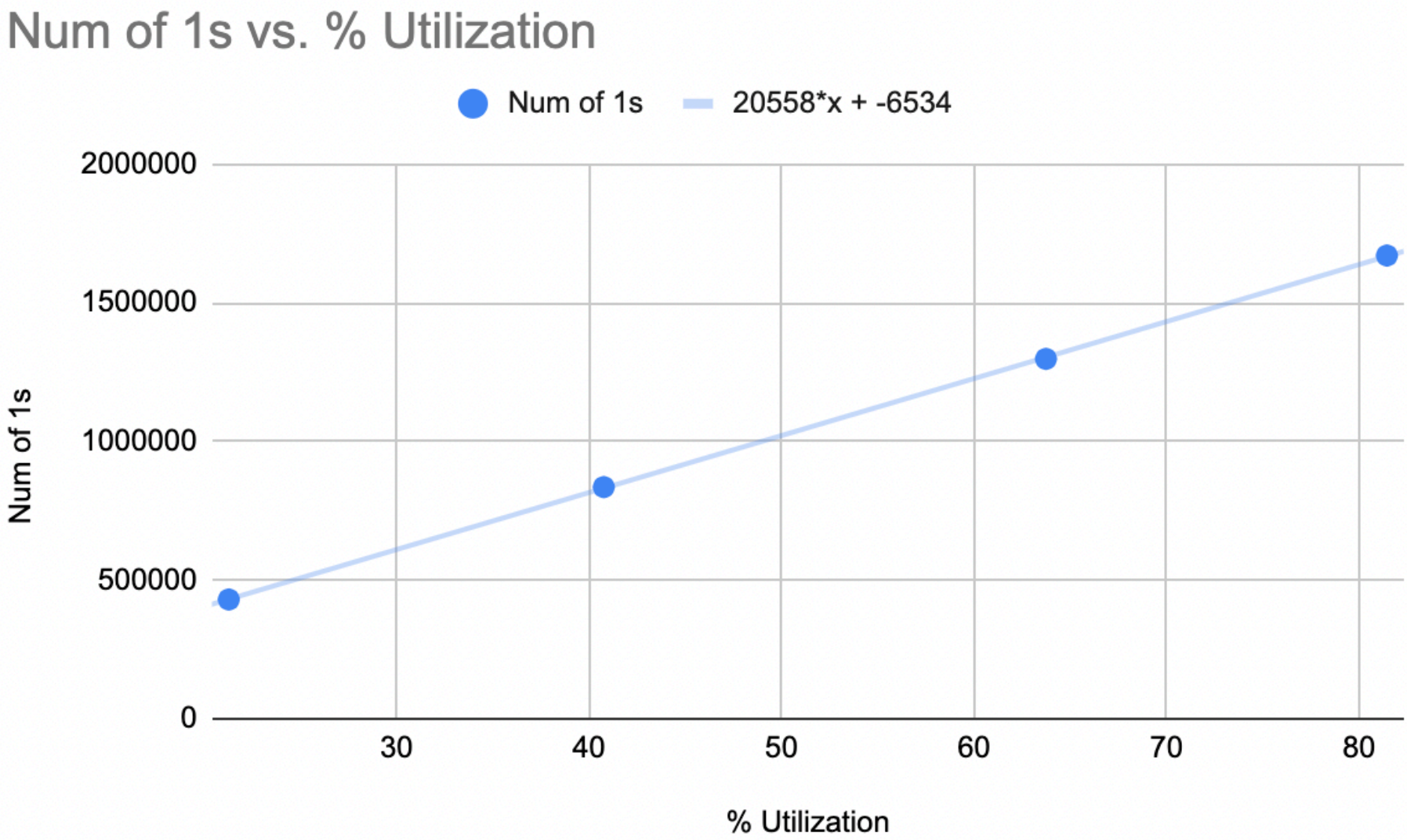
Bitstream Information

21.29 Percent Utilization has:

- Number of bytes: 3825888
- Number of zeros: 30176958
- Number of ones: 430146
- To achieve 139 bins, sample at 220194 size

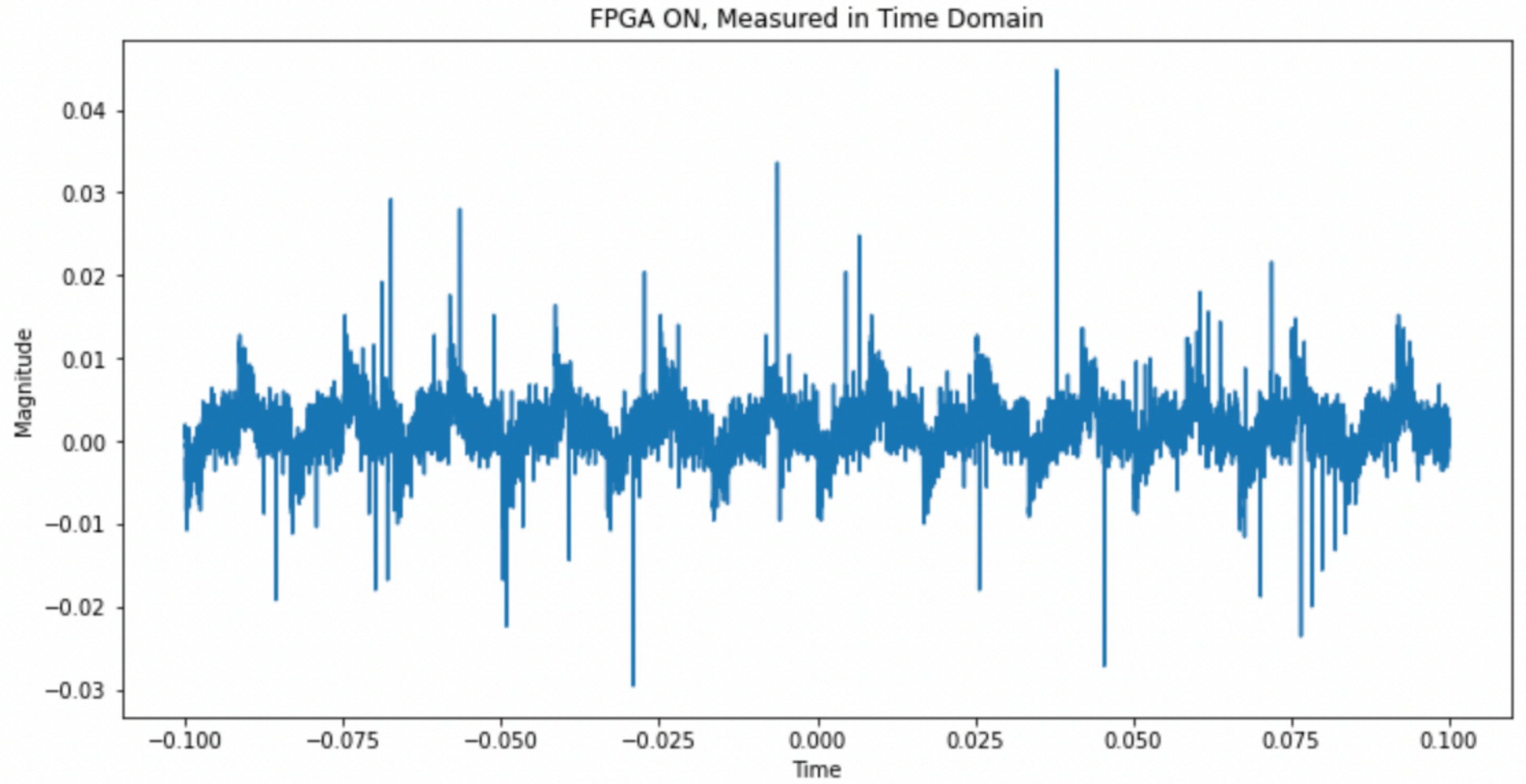
Bitstream Percent Utilization Feature

Direct linear relationship between number of 1s and FPGA utilization



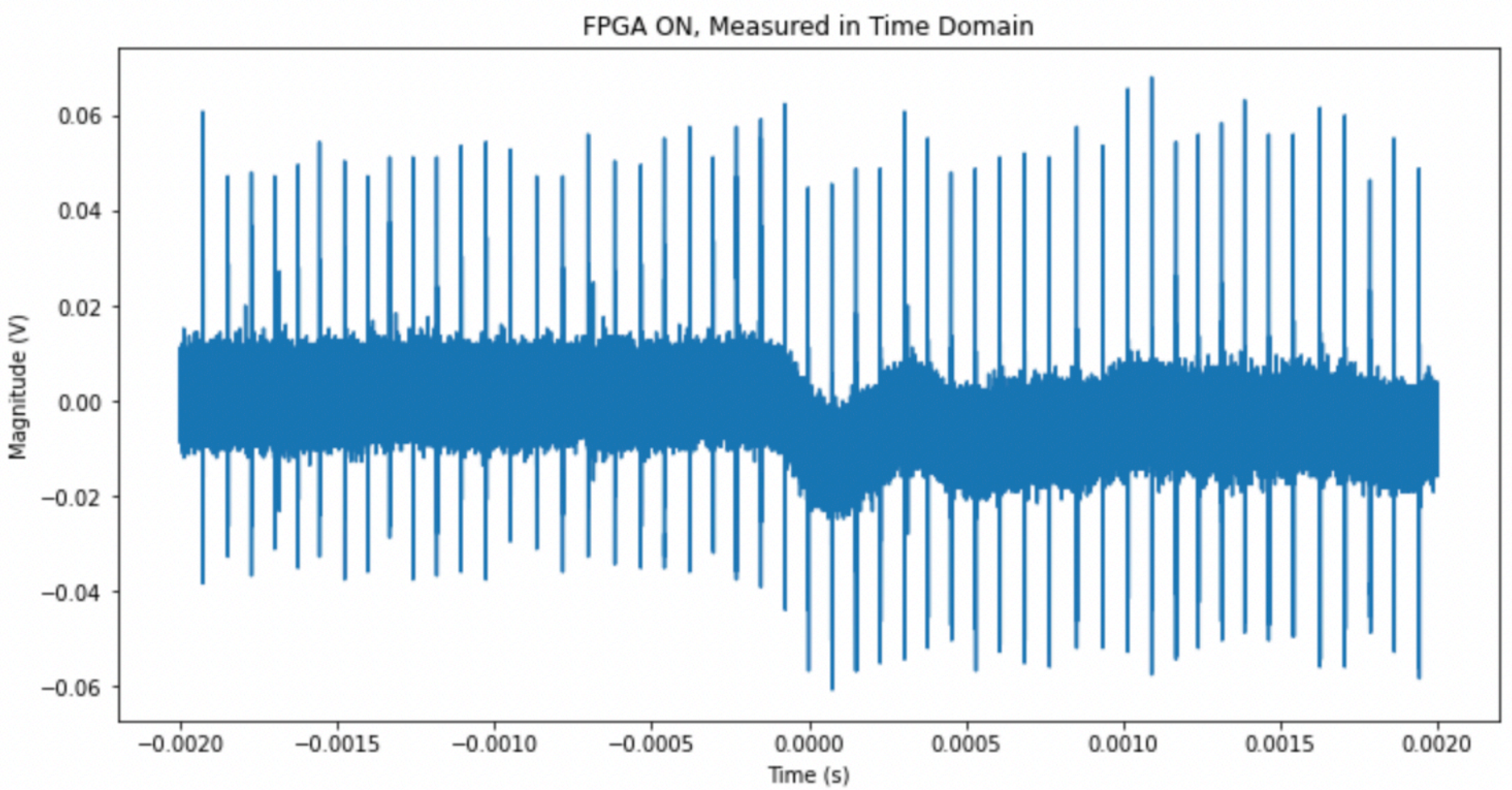
Preliminary observations: Time domain data, 50kHz

Wave pattern apparent (probe near JTAG), but remains even when probe moved further away - electronic noise?



Preliminary observations: Time domain data, 250MHz

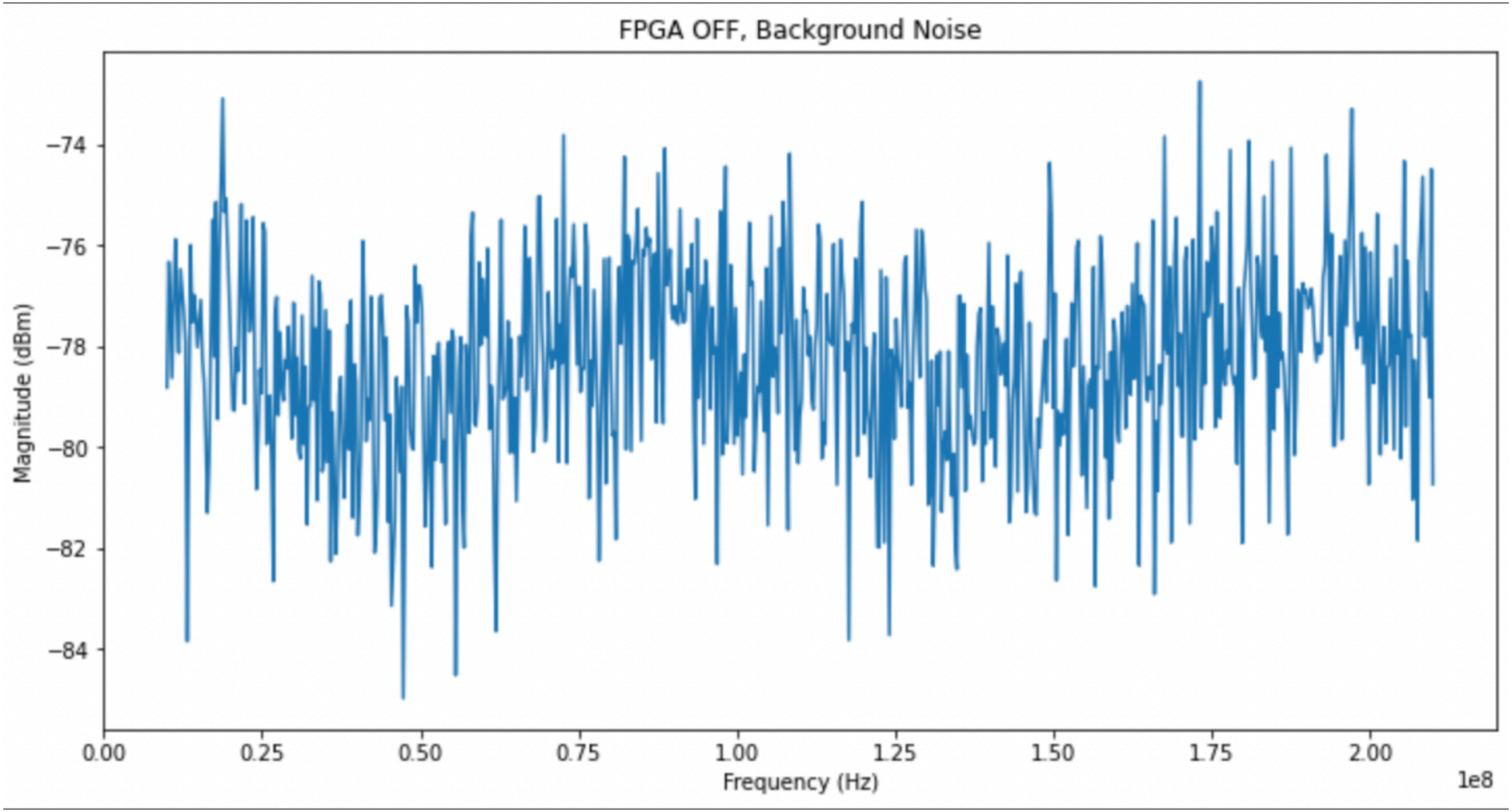
Difficult to capture individual 0->1 or 1->0 transition; need to consider in aggregate due to oscilloscope sampling during measurement.



Preliminary observations: Frequency domain, 10 to 210MHz

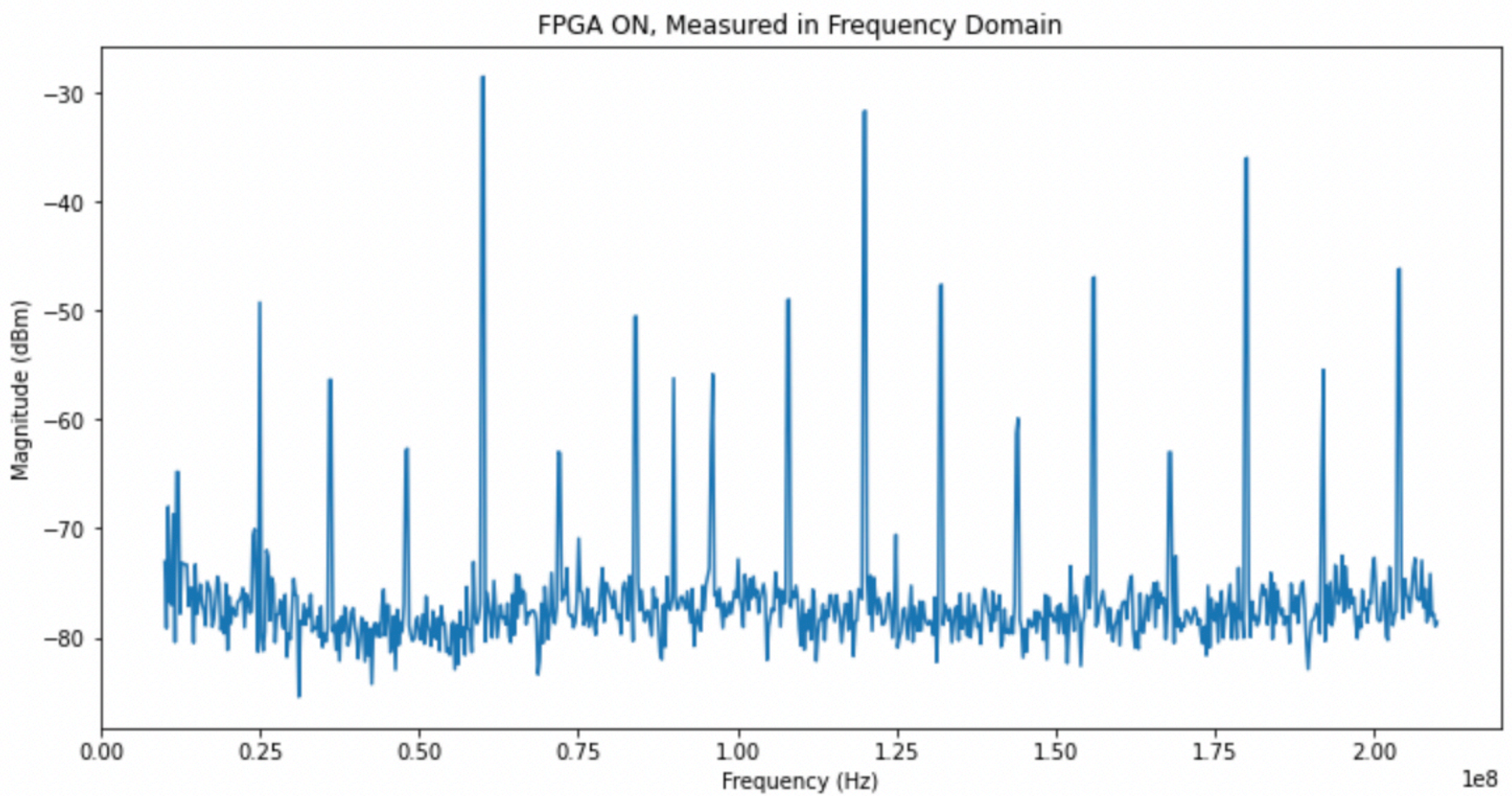
Change to frequency domain data

Characterize lab noise first with FPGA off



Preliminary observations: FPGA on

Large change once the FPGA is switched on. The largest peaks are at 60, 120 and 180 Mhz and generally 12 Mhz apart (harmonics).



Findings

From the above time and frequency domain results

findings

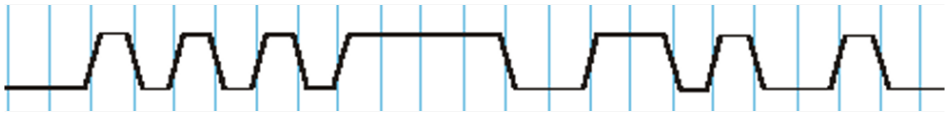
1) Significant noise in time domain data and instrument limitations –> move to frequency domain
2) Bitstream transitions <-> Energy signature
3) FPGA on/off/utilization effects clear

Information Exploitation Goals

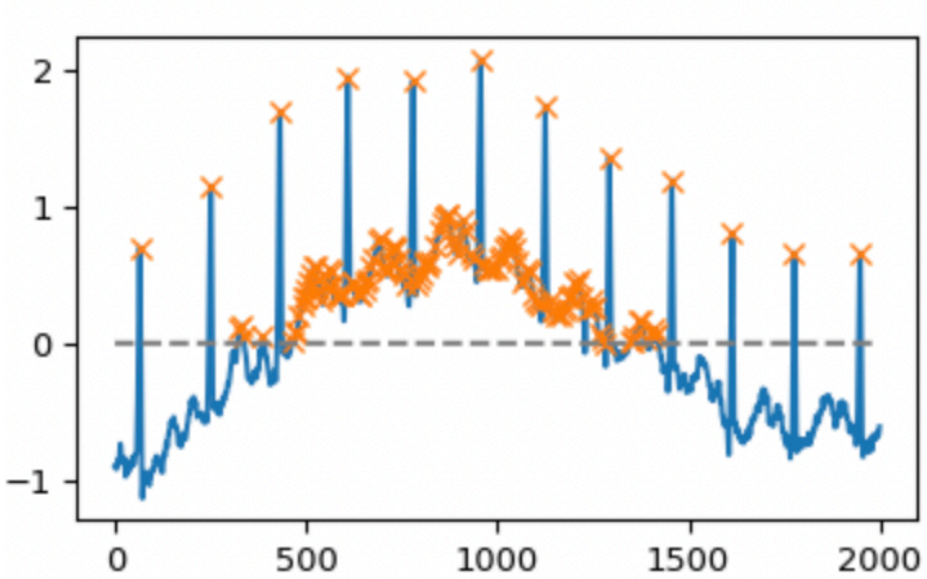
What we may be able to deduce from the RF signature, in increasing order of difficulty:

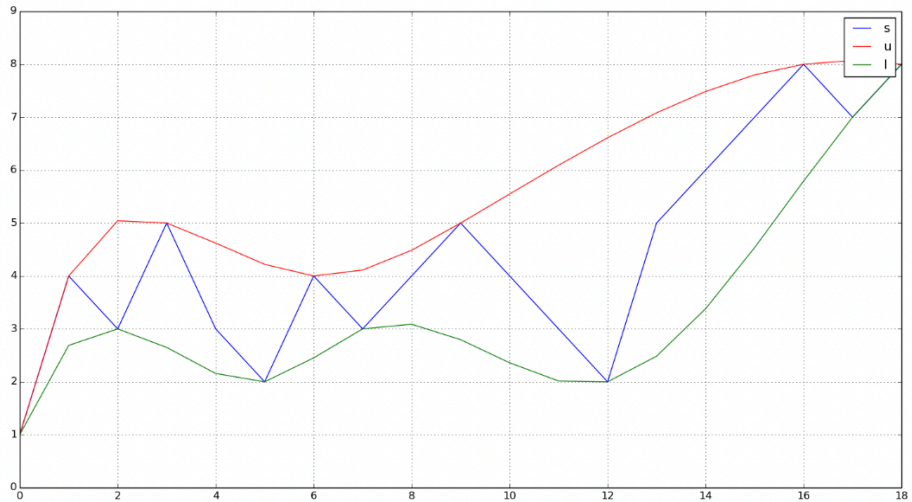
levels of information

0) Whether FPGA is switched on, or having a new bitstream uploaded
1) Observe the waveform of RF signature and compare with bitstream
2) Reverse-engineer JTAG frequency used
3) Bitstream feature relating to code e.g. percentage of Ring Oscillator use



Information Exploitation Methods



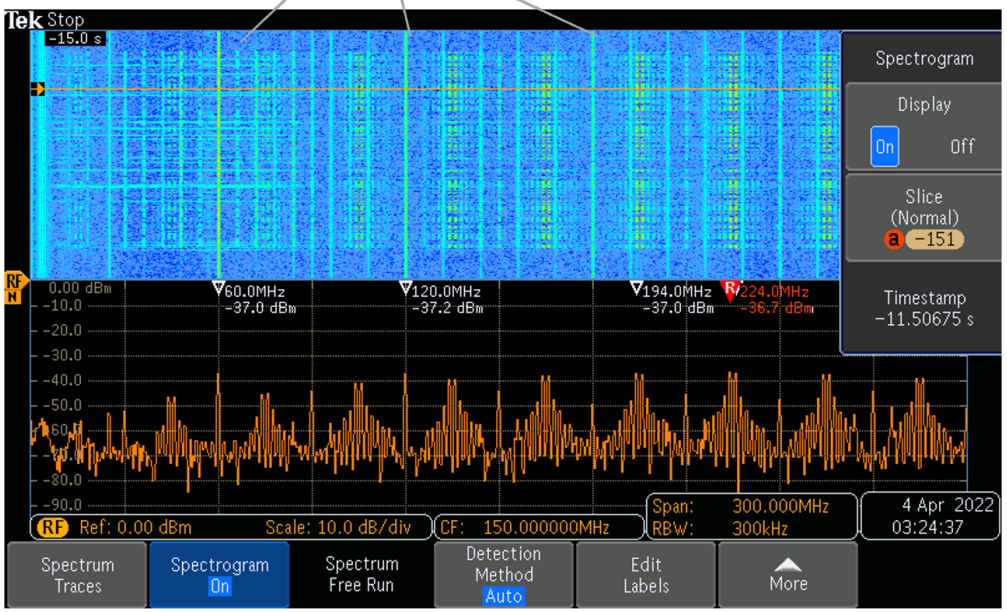


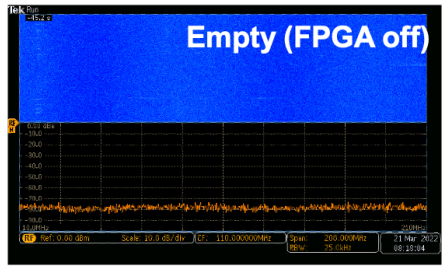
Level 0: Spectrogram reveals FPGA state

Record frequency domain information across time (each ”slice” is color-coded to intensity of frequency component)

Default FPGA **on** has vertical lines only (harmonic)

Signal expands **during** bitstream upload



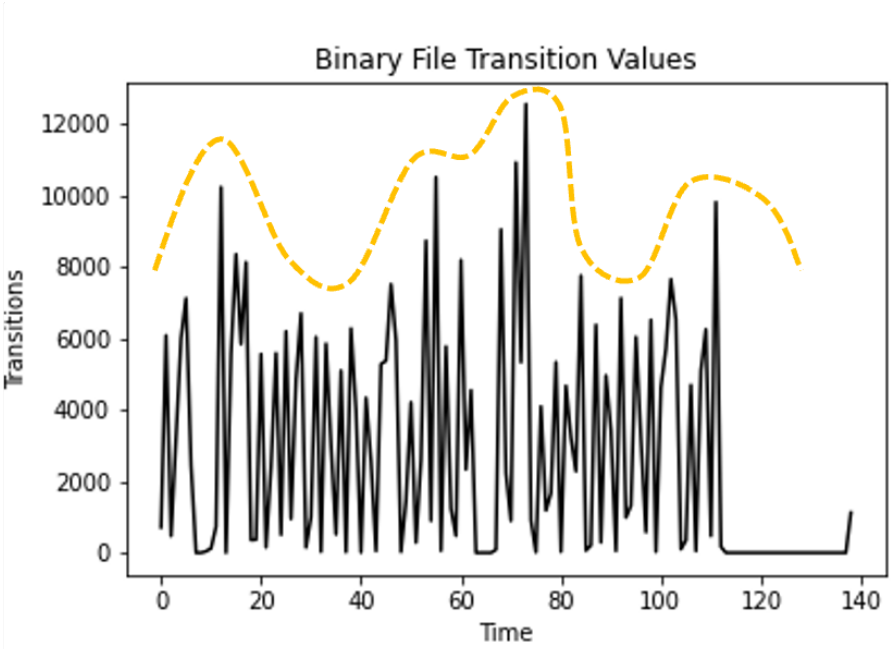


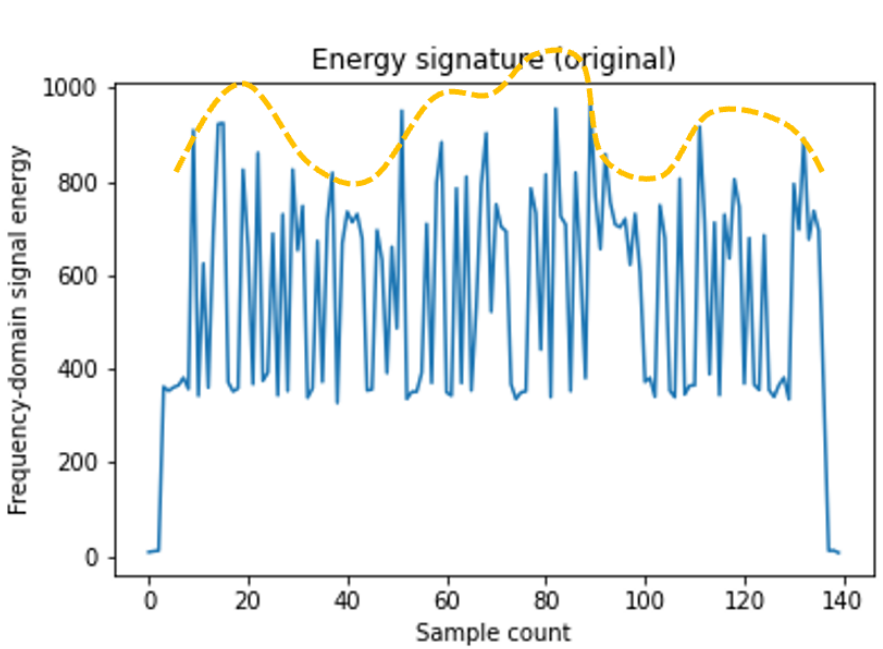
Thus, we can tell immediately if FPGA is Off, On (idle), or being uploaded a new bitstream.

Level 1: Data for RO 21 percent

Compare between the bitstream transitions (0–>1 and 1–>0) and electromagnetic waveform - any correlation?

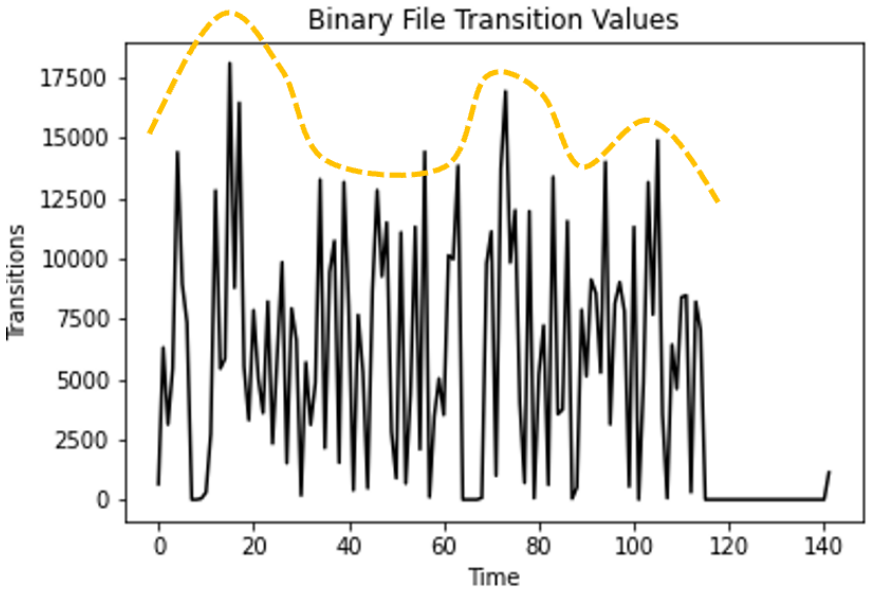
Ring oscillator codes generated for different levels of FPGA usage, starting with 21

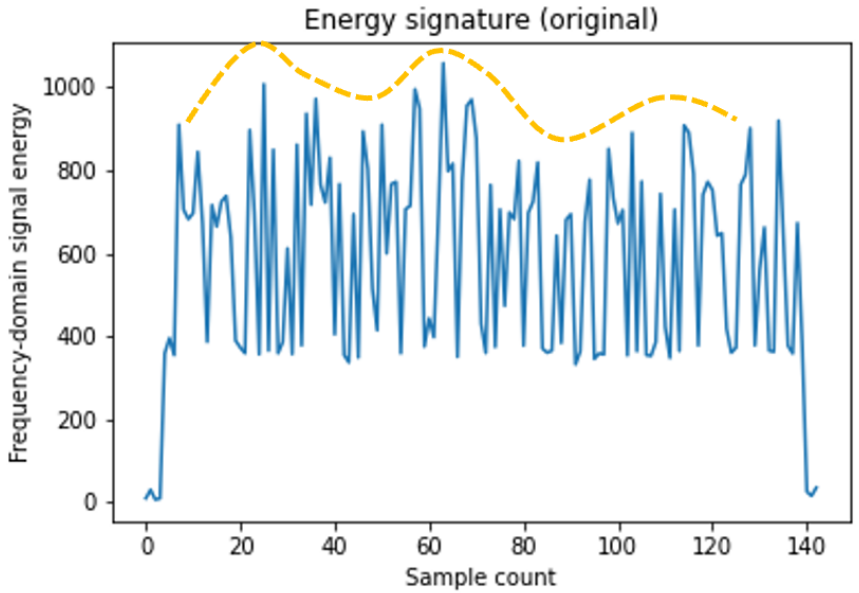




Level 1: Data for RO 41 percent

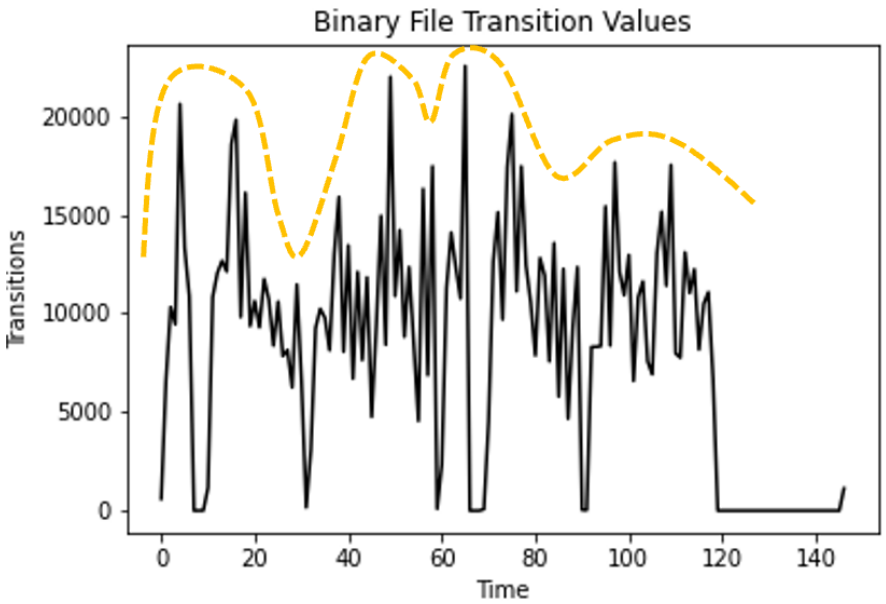
Bitstream analysis and electromagnetic waveform

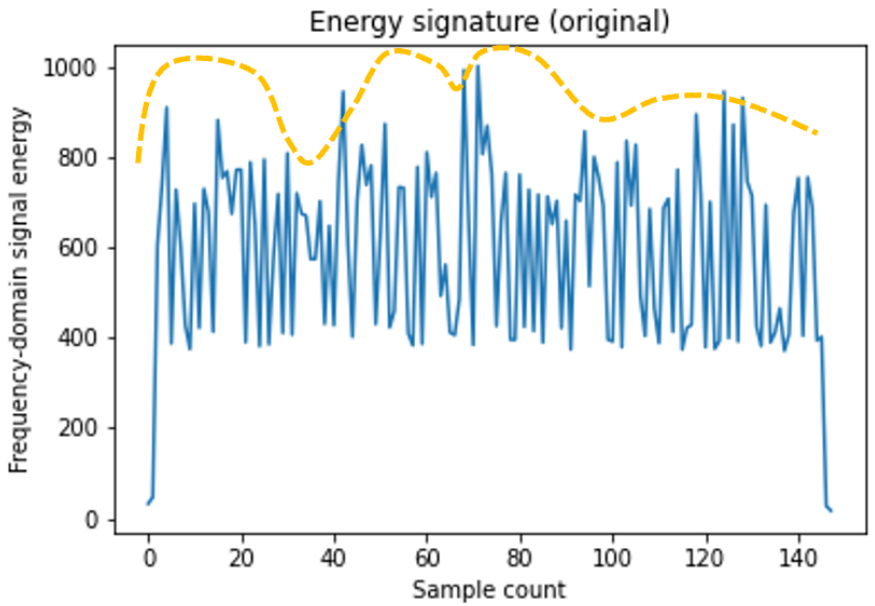




Level 1: Data for RO 64 percent

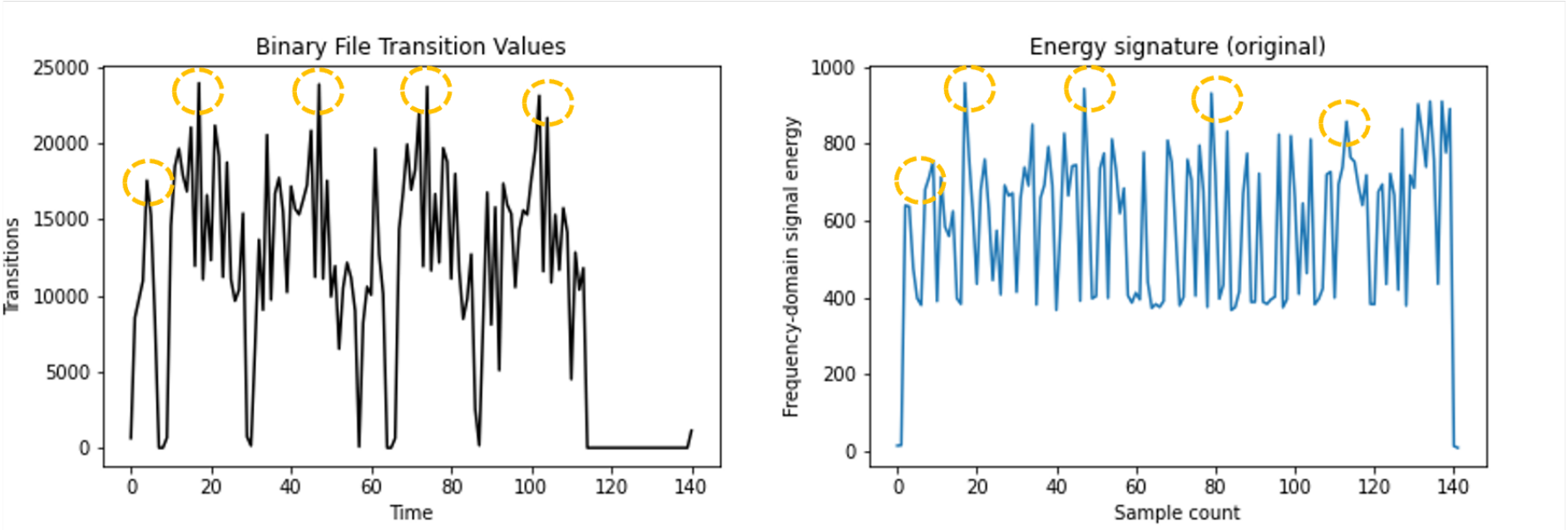
Bitstream analysis and electromagnetic waveform





Level 1: Data for RO 81 percent

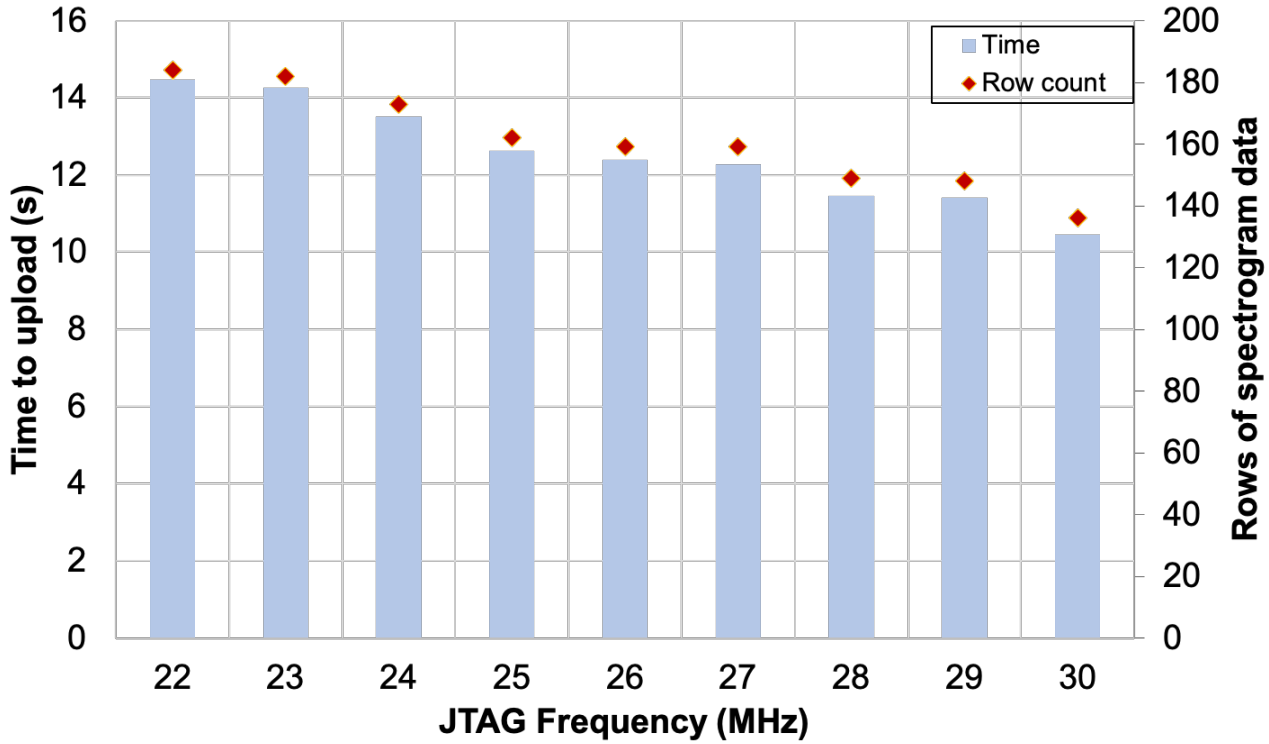
Bitstream analysis and electromagnetic waveform captured show peaks align



Since there is some correlation between the energy signature and actual bitstream transitions, a physical side-channel attack may reveal general patterns about the bitstream.

Level 2: Obtaining JTAG frequency from RF data

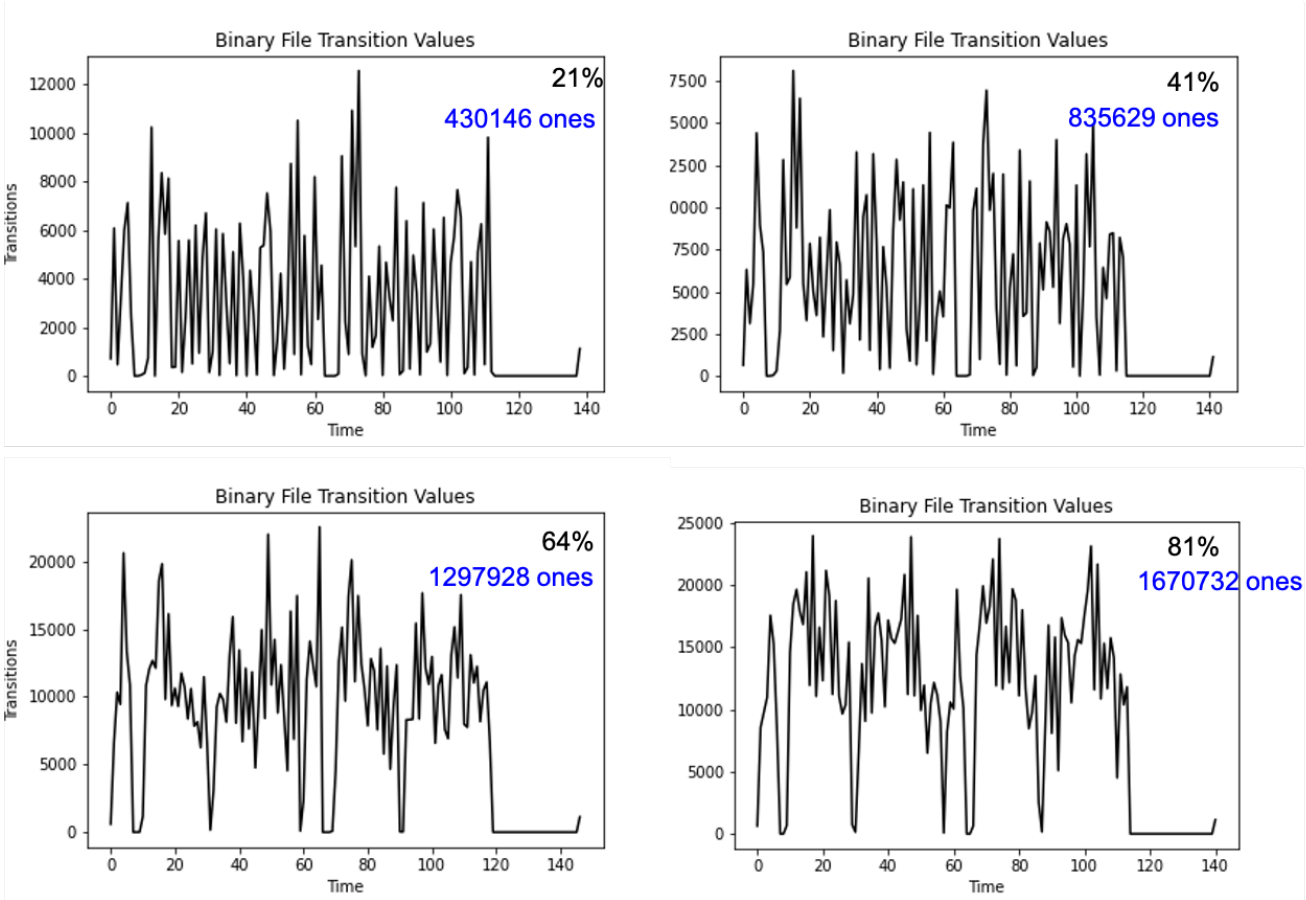
Record the time taken to upload the bitstream and number of rows of oscilloscope data present



From the linear trend, we can clearly deduce the JTAG frequency based on the size of observed oscilloscope data.

Level 3: Identifying bitstream to code

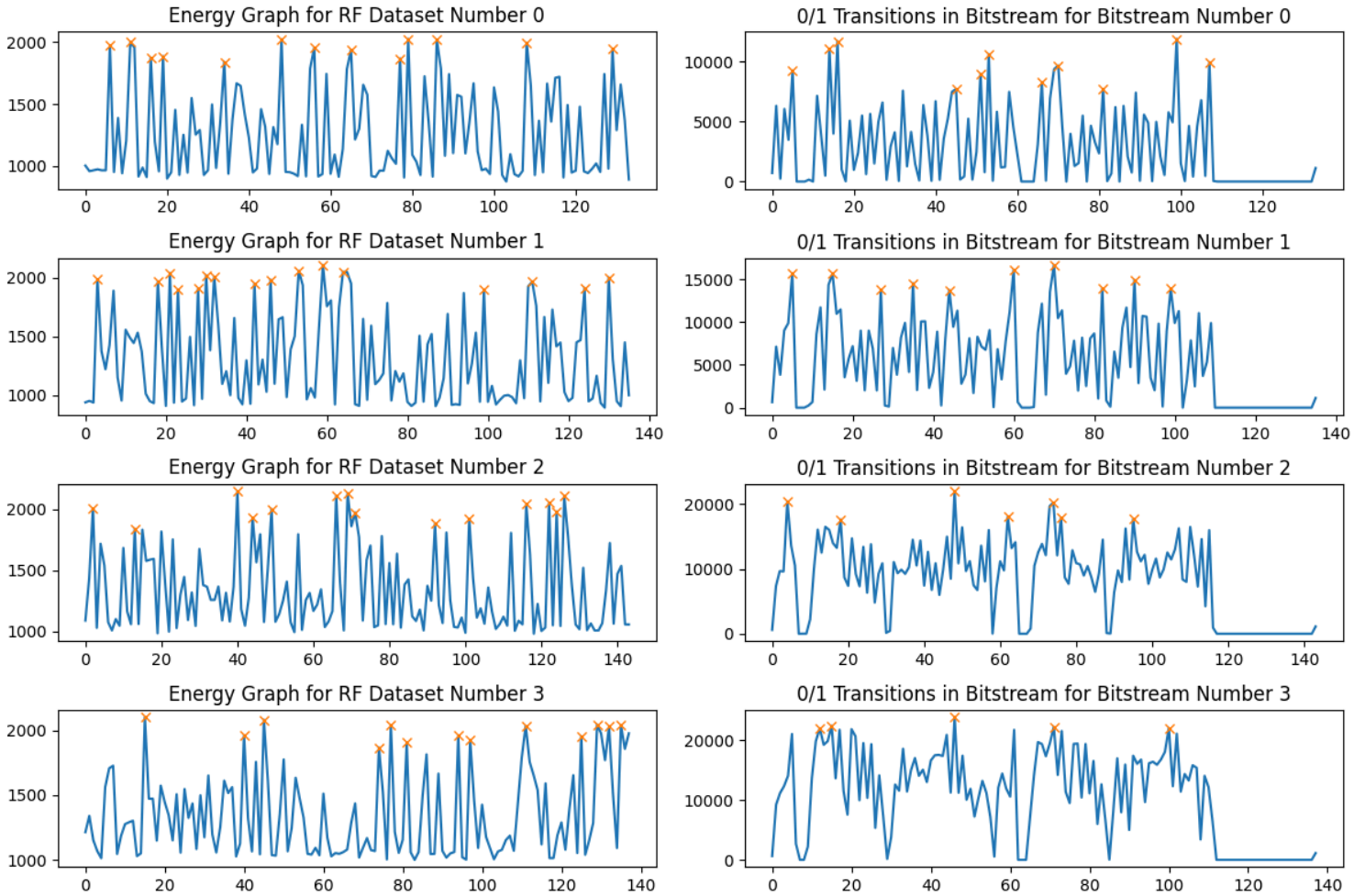
Each of the bitstreams has 30607104 bits, with majority zeros. But the number of ones is proportional to ring oscillator usage percentage!



Thus we can even deduce whether a design is ”light” or ”heavy” in FPGA usage.

Level 3: Quantitative analysis

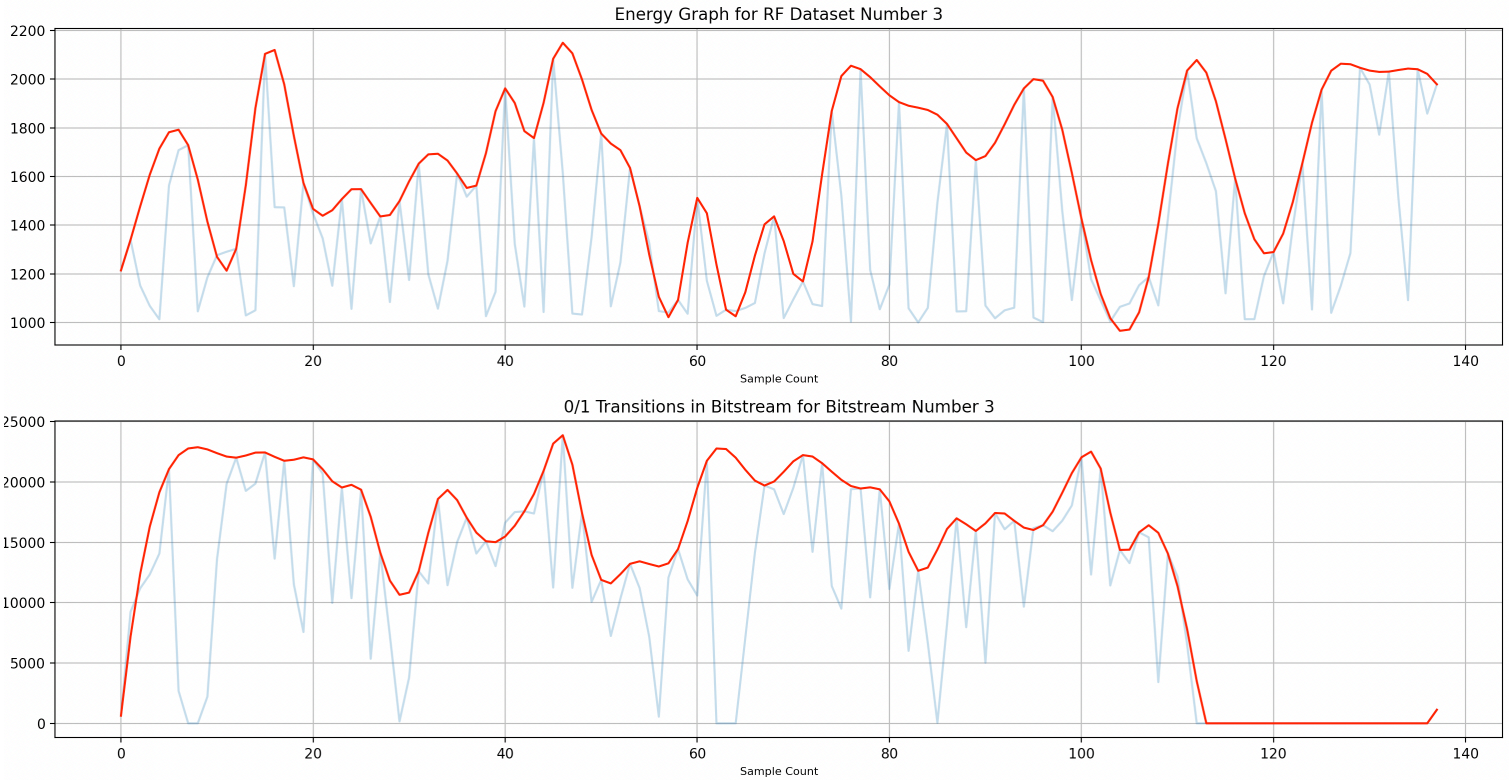
Pattern match between bitstreams and RF data with 1.5 s.d. threshold



Can improve with a moving average threshold

Level 3: 81 percent utilization fit example

Automatic identification of peaks



Conclusions

Analysis of emanations reveals sensitive information about FPGAs:

levels of information

0) Whether FPGA is switched on, or having a new bitstream uploaded
1) Observe the waveform of RF signature and compare with bitstream
2) Determine JTAG frequency used
3) Bitstream feature relating to code e.g. percentage of Ring Oscillator use

Developed scripts to automate energy signature calculation and re-sample bitstream to imitate real-life activity

Main difficulty was getting OpenOCD to work on lab FPGA

Security implications

Our results suggest physical side channel attacks can reveal private information about the FPGA status and bitstream (size, shape, frequency)

Possible defences are:

- Instead of lumping all bit signals in a small area of time, spread out over the bitstream
- Secure FPGA cases from probe insertion, and run many devices at the same time to generate a cover of noise

Future steps

Future steps for research

- Understand what parts of the bitstream correspond to which code segments, beyond the 0/1 digit layer (fingerprint header)
- Improve spectrograph collection resolution (our lab oscilloscope has a fixed window of 15 seconds, limiting to around 150 time bins max)
- Finetune quantitative waveform fitting between measured and bitstream data (standard deviation, moving averages)

Project milestones

23 Feb	Project introduction
2 April	Test bitstream from different Verilog designs and capture respective RF signals
5 April	Started OpenOCD implementation to modify FPGA parameters
10 April	Developed Python script to parse CSV and bitstream files
26 April	Collected and analyzed signals
30 April	Identify bitstream features (e.g. size, ”0”/”1”)
4 May	Project final presentation

References

Martin Vuagnoux, Sylvain Pasini. (2009) Compromising Electromagnetic Emanations of Wired and Wireless Keyboards.

Litao Wang, Bin Yu. (2011) Analysis and Measurement on the Electromagnetic Compromising Emanations of Computer Keyboards.

Yang Su, Daniel Genkin, Damith Ranasinghe, Yuval Yarom. (2017) USB Snooping Made Easy: Crosstalk Leakage Attacks on USB Hubs.

Thank You!

Austin Zhu & Yu Jun Shen

austin.zhu@yale.edu, yujun.shen@yale.edu