پژوهشگاه‌دانش‌های‌بنیادی (مرکز تحقیقات فیزیک نظری و ریاضیات)

IPM

INSTITUTE FOR RESEARCH IN FUNDAMENTAL SCIENCES

Cyber-physical systems society of Iran

RTEST 20

SOFTWARE-BASED MONITORING AND ANALYSIS OF A USB HOST CONTROLLER SUBJECT TO ELECTROSTATIC DISCHARGE

Presenter: Natasha Jarus

Authors: Natasha Jarus, Antonio Sabatini, Pratik Maheshwari, and Dr. Sahra Sedigh Sarvestani

Missouri S&T Electrical and Computer Engineering

June 10, 2020

INTRODUCTION

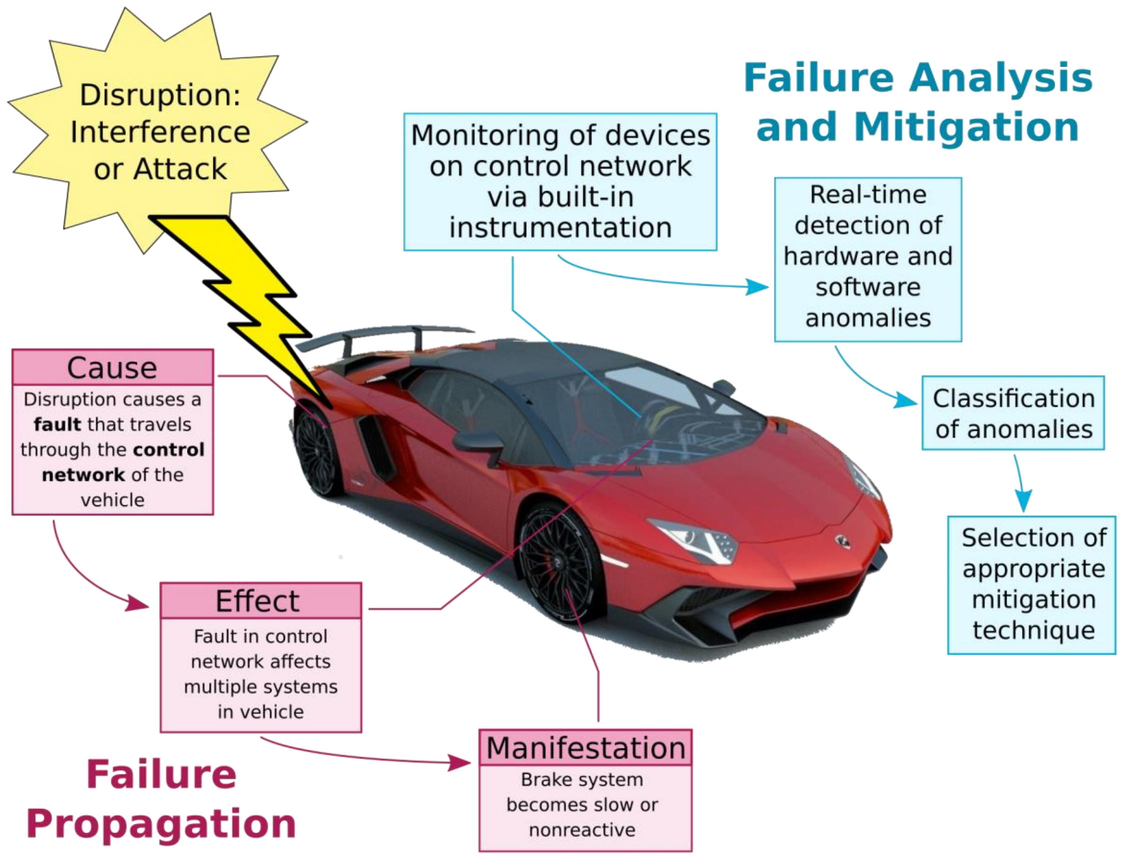
Natasha Jarus

I am a PhD candidate in computer engineering at the Missouri University of Science and Technology. My work focuses on modeling and metamodeling complex systems to understand and improve their dependability.

This research has been done in collaboration with Samsung, Ford, and the Missouri S&T Electromagnetic Compatibility Lab.

INTRODUCTION

- Static electricity discharge can cause:
  - Screen glitches
  - Program crashes
  - Erroneous software operation
  - System resets
  - Permanent hardware failures
- Dependable cyber-physical systems must be robust to the effects of these shocks
- The effects of these shocks on system hardware are much better understood than they are for software operation


Disruption: Interference or Attack
Failure Analysis and Mitigation
Monitoring of devices on control network via built-in instrumentation
Real-time detection of hardware and software anomalies
Classification of anomalies
Selection of appropriate mitigation technique
Cause
Disruption causes a **fault** that travels through the **control network** of the vehicle
Effect
Fault in control network affects multiple systems in vehicle
Manifestation
Brake system becomes slow or nonreactive
Failure Propagation

MONITORING ESD

- Hardware instrumentation
  - Can provide a precise understanding of how Electro-Static Discharge (ESD) entered and propagated through the system
  - Is difficult to scale up to instrumentation for the whole system
  - Is infeasible to implement for field tests on commercially available equipment
  - Tests are often implemented using custom low-level software rather than a typical system software load
- Software instrumentation
  - Is often focused on user-visible faults such as display flicker and program crashes
  - Investigates lower-level faults, such as bit errors in registers, and is usually done with low-level code that cannot coexist with other software
  - The software executing on a system can affect its immunity to ESD

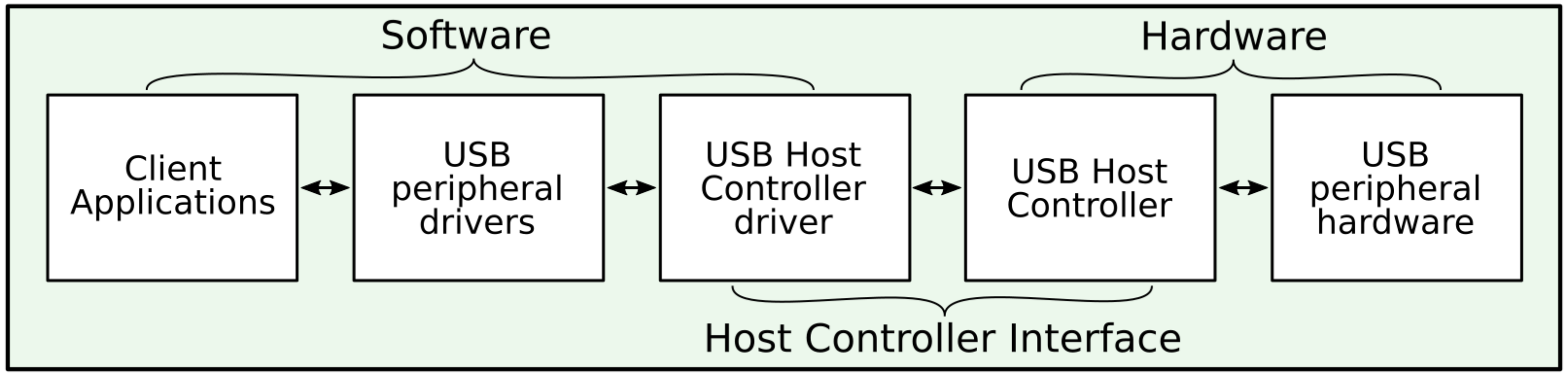
RESEARCH OBJECTIVES

- Improve software instrumentation for low-level faults
- Achieve software fault detection on consumer hardware in field use conditions
- Enable lightweight real-time monitoring and failure recovery

- Create a generic approach that applies to many system peripherals
- Validate and demonstrate by applying to USB devices

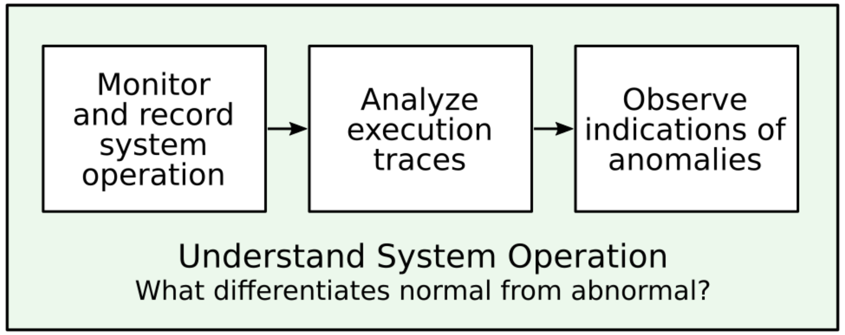
MONITORING APPROACH

- The USB Host Controller connects to the USB bus and performs low-level USB host device duties
- Responsibilities:
  - connecting and disconnecting devices
  - configuring power delivery
  - communicating control and data signals between the system's memory and the USB devices



MONITORING APPROACH

- The Host Controller exposes a set of control registers to the host CPU
- These registers conform to Open Host Controller Interface specifications
- We record snapshots of these register values to approximate the HC's internal operation, presuming that:
  - Certain sequences of values will be common during typical system operation
  - When exposed to ESD, we may observe anomalous values or sequences of values
- Our goal is to infer ESD exposure from anomalies in recorded traces of these snapshots

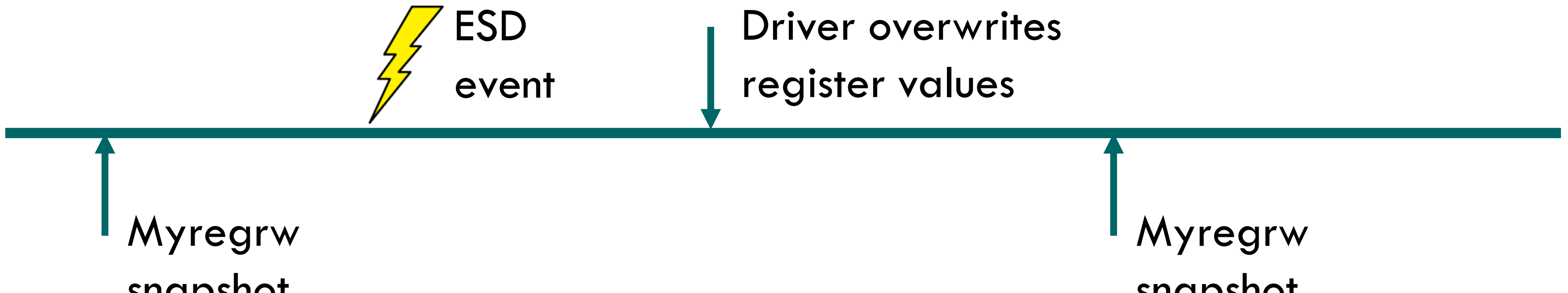

Monitor and record system operation
Analyze execution traces
Observe indications of anomalies
Understand System Operation
What differentiates normal from abnormal?

INITIAL INSTRUMENTATION APPROACH

- Directly read the memory-mapped Host Controller registers
- Modified an open-source tool, Myregrw, to suit our needs
  - System driver that reads memory addresses on command
  - User program that sends control signals to driver and records values
- Read values continuously while exposing the system to ESD
- Problem: register values remained mostly constant
- This approach failed to capture even typical Host Controller operation. Why?

INITIAL INSTRUMENTATION APPROACH

- We empirically determined the sampling rate of Myregrw on our system to be 342 Hz
- Assuming that, in the worst case, the register values change at 400 MHz, we have a 0.000856% chance of observing a given value
- Furthermore, Myregrw is racing the Host Controller driver to read the values written by the host controller before its driver overwrites them
- A new approach was needed to address both of these issues



IMPROVED INSTRUMENTATION APPROACH

- Instrument the USB Host Controller driver to record the register values at the start of each function
- Gives an exact picture of what the driver “sees” the host controller doing
- Requires minimal modifications to the driver code
- Values are recorded using the standard kernel logging framework
- Overhead is low: about 10% with a naive logging approach

```
function: ohci_irq
HcControl: 0x83
HcCommandStatus: 0x4
HcInterruptStatus: 0x24
HcInterruptEnable: 0x8000005e
HcInterruptDisable: 0x8000005e
HcHCCA: 0x338b1000
HcPeriodCurrentED: 0x0
HcControlHeadED: 0x339b2000
HcControlCurrentED: 0x0
HcBulkHeadED: 0x339b2080
HcBulkCurrentED: 0x0
HcDoneHead: 0x0
HcFmInterval: 0xa7782edf
HcFmRemaining: 0x80002760
HcFmNumber: 0x921d
HcPeriodicStart: 0x2a2f
HcLSThreshold: 0x628
HcRhDescriptorA: 0x2001202
HcRhDescriptorB: 0x0
HcRhStatus: 0x8000
HcRhDescriptorA: 0x2001202
HcRhPortStatus[0]: 0x103
HcRhPortStatus[1]: 0x100
Done.
```

ANALYSIS APPROACH

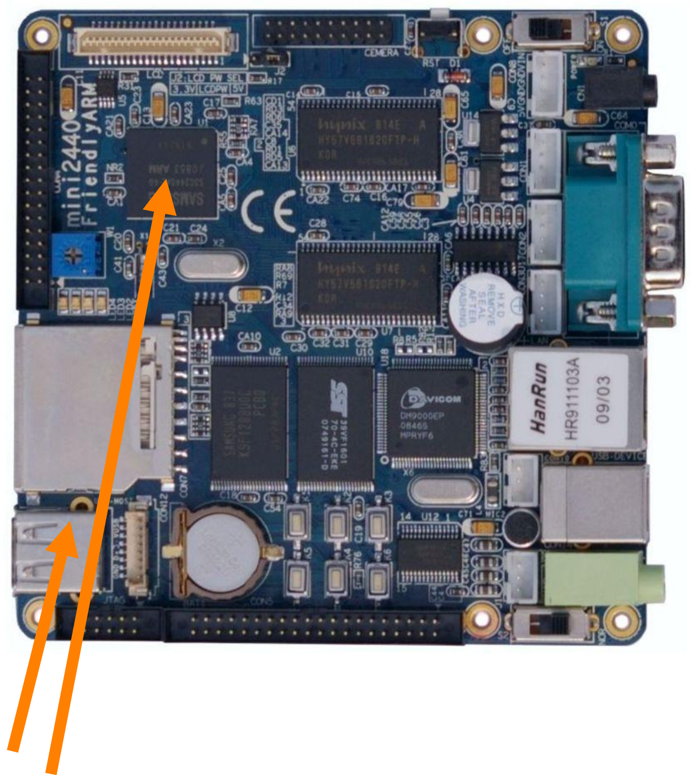
- Monitor system operation both without interference and while exposed to ESD
  - Baseline logs capture ‘normal’ system behavior without interference
  - ESD-exposed logs capture normal and abnormal system behavior
- These log files consist of sequences of snapshots of register values
- We refer to a log as an *execution trace*
- We refer to a snapshot of the registers’ values as a *state*

ANALYSIS APPROACH

- Identify and coalesce duplicate states in each trace to construct an *execution graph*
  - We also record the path through the graph taken by the execution trace
- Identify and coalesce duplicate states in each graph to construct a *global execution graph*
  - We ignore certain registers whose values are memory addresses set by the kernel memory allocator, as they do not reflect the operation of the Host Controller itself
  - Execution paths through the global graph are recorded for each trace
- We can then identify states and transitions unique to ESD-exposed execution traces

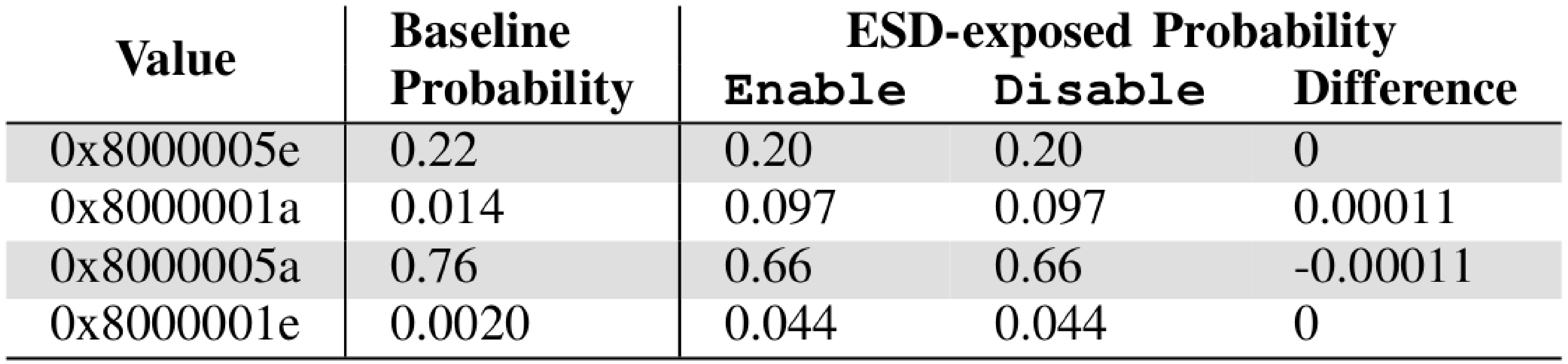
CASE STUDY

- System: FriendlyArm mini2440
  - 400 MHz ARM Samsung CPU
  - Running Linux as the operating system
- A flash drive is attached and a script runs to copy data to and from it
- Several ESD injections were performed:
  - Electric field coupling probe: ESD pulses between 500 V and 5.5 kV
  - Magnetic field coupling probe: ESD pulses between 500 V and 8 kV
  - The system proved to be more immune to magnetic field coupling, hence the higher pulse voltage
  - Probes were positioned over the USB port or the USB Host Controller IC



RESULTS: REGISTERS

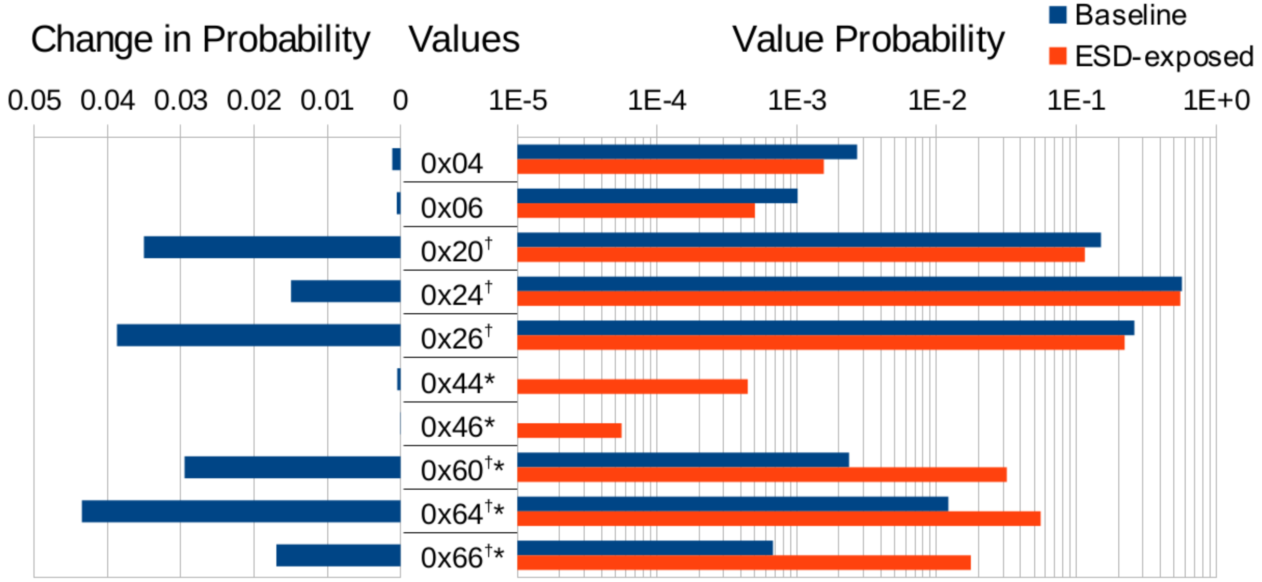
- The `HcInterruptEnable` and `HcInterruptDisable` registers control whether the various hardware interrupts on the Host Controller are enabled or disabled
- When read, these registers ought to be duplicates of each other
  - Baseline data confirms the host controller follows this specification
  - However, under ESD exposure, slight dissimilarities are observed: bit 7 sometimes disagrees



Value	Baseline Probability	ESD-exposed Probability		
		`Enable`	`Disable`	Difference
0x8000005e	0.22	0.20	0.20	0
0x8000001a	0.014	0.097	0.097	0.00011
0x8000005a	0.76	0.66	0.66	-0.00011
0x8000001e	0.0020	0.044	0.044	0

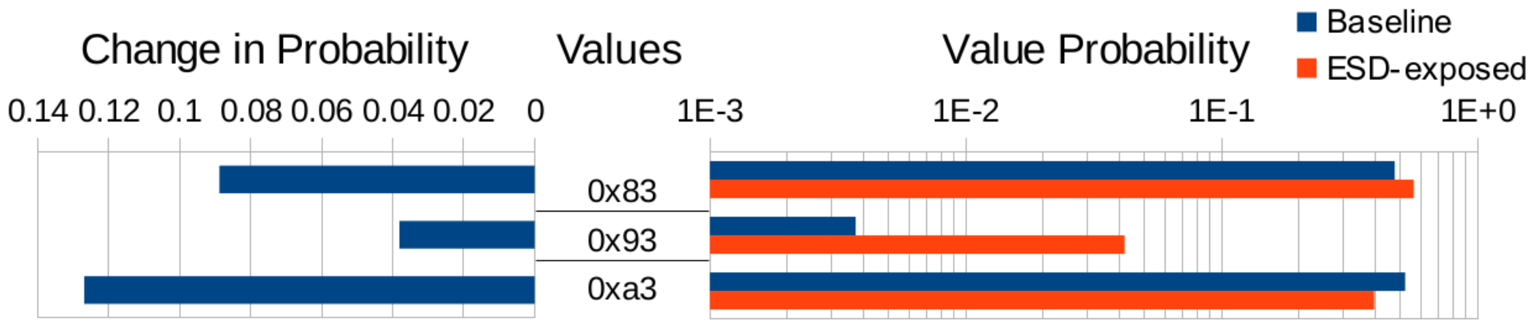
RESULTS: REGISTERS

- The `HcInterruptStatus` register records whether an interrupt has been triggered
- When exposed to ESD, we observe certain interrupts are more likely to be triggered:
  - Frame number counter overflows (marked †)
  - Hub status change events (marked *)



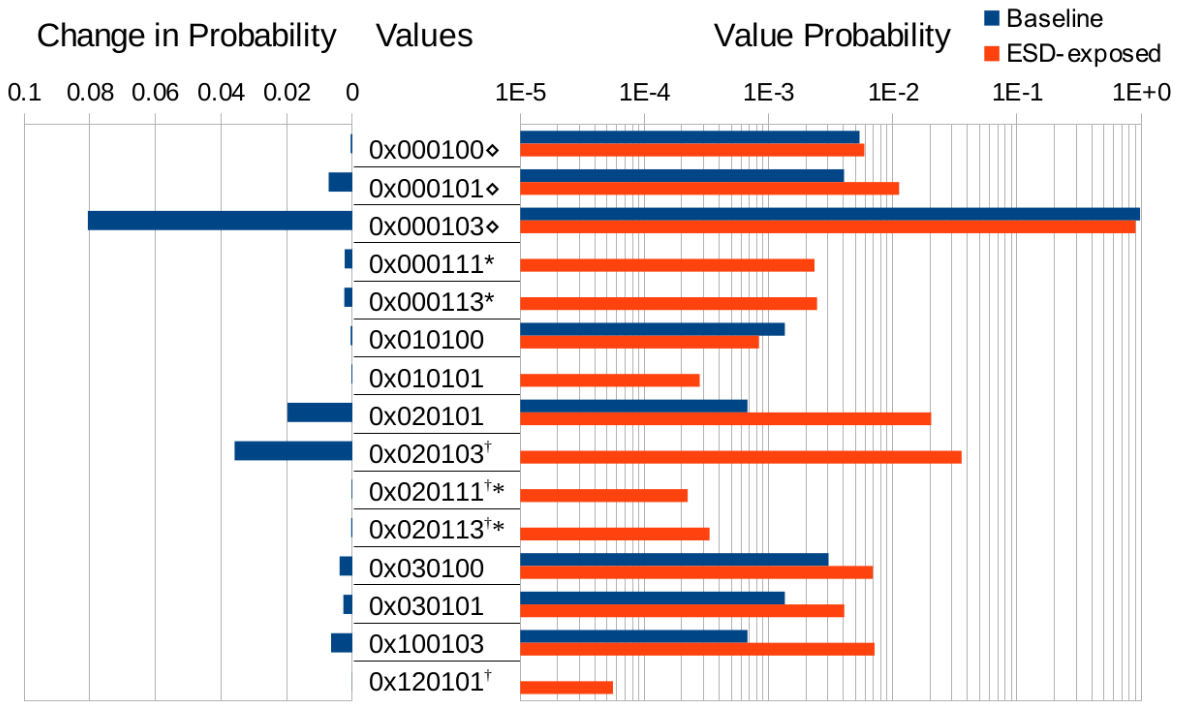
RESULTS: REGISTERS

- The `HcControl` register allows the Host Controller driver to control what the Host Controller processes next
- Under ESD exposure, we observe
  - Increased control frame processing (`0x93`) and decreased bulk data processing (`0xa3`)
  - Increased number of new control and data frames (`0x83`)
  - It is possible that ESD is disrupting the bus's operation, resulting in more status change frames and more retransmissions of corrupted data frames



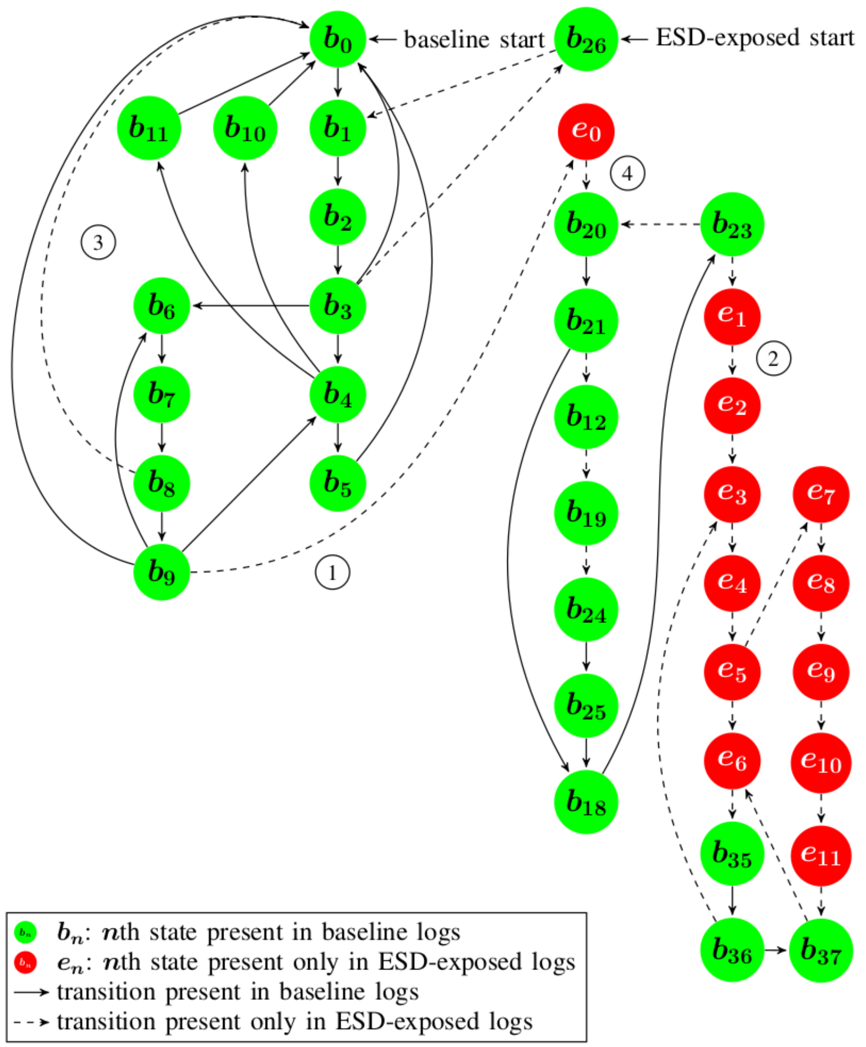
RESULTS: REGISTERS

- The `HcRhPortStatus0` register reports the status of the port we plugged the USB device into during tests
- Values where the port status remains unchanged (◊) occur less often and port enable/disable events (†) occur more often
- Furthermore, port resets (*) are only observed in ESD-exposed traces



RESULTS: EXECUTION GRAPHS

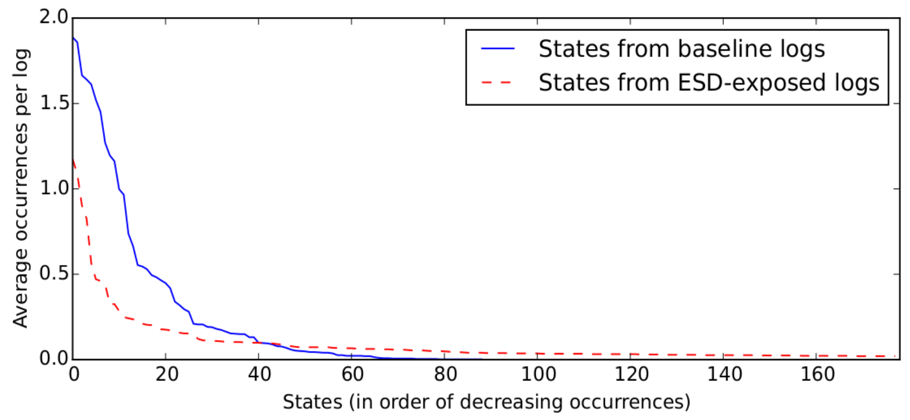
- The unified execution graph of two traces, one baseline and one ESD-exposed, is shown to the right
- Green states appear in baseline traces; red states appear only in ESD-exposed traces
- We see several potential effects of ESD:
  1. Transitions from baseline states to non-baseline states
  2. Transitions between non-baseline states
  3. Transitions between baseline states not present in baseline traces
  4. Transitions from non-baseline states to baseline states



b_0 ← baseline start
b_{26} ← ESD-exposed start
b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11}
b_{20}, b_{21}, b_{12}, b_{19}, b_{24}, b_{25}, b_{18}, b_{23}
e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}
b_{35}, b_{36}, b_{37}
1
2
3
4
b_n: nth state present in baseline logs
e_n: nth state present only in ESD-exposed logs
→ transition present in baseline logs
⇢ transition present only in ESD-exposed logs

RESULTS: EXECUTION GRAPHS

- We also observe differences in the distribution of state occurrences between baseline and ESD-exposed traces
- During normal operation, we expect to see:
  - A large group of states revisited frequently — for example, the main control loop
  - A short tail of 'exceptional' states that handle unusual events
- If operation were disturbed by ESD, we would expect:
  - Fewer common states
  - A much longer tail of unusual or anomalous states
- These assumptions are confirmed by our data



CONCLUSIONS

- We have developed a methodology for studying the effects of ESD on system peripherals using software instrumentation
- Our software probe records traces of a peripheral's control registers
- This approach approximates the unobservable internal behavior of the peripheral based on its behavior that is observable to the host CPU
- The implementation has low overhead and can be used in field tests
- We demonstrated this technique by applying it to a USB Host Controller
- We were able to observe differences in system operation when the system was exposed to ESD

CONCLUSIONS

- We can predict, based on the operation of the system, whether it has experienced the effects of ESD
- This allows us to identify which components on a system have experienced ESD and may require repair or better shielding
- Software may be able to determine that an ESD event has occurred and to automatically recover from resulting errors
- This would enable systems to continue operating in hostile environments
- We have determined that a naive classification approach can detect ESD events in a system execution trace with 88% accuracy
  (work under review for IEEE Transactions on Electromagnetic Compatibility)

FUTURE WORK

- Expand this approach to other peripherals, enabling full-system monitoring of ESD effects
- Characterize the effects of ESD at specific injection locations, potentially allowing us to trace ESD through the board based on observed operation
- Develop software that can anticipate and recover from ESD events