A Methodology for Interrupt Analysis in Virtual Platforms

Puneet Dhar

SYNOPSYS®

Silicon to Software™

accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Agenda

- Introduction
- Interrupts in Virtual Prototyping
- Challenges in Debugging Interrupts
- Proposed Analysis Methodology
- RH850 Adaptation
- Conclusions

Pre-Requisites

- Basic understanding of
  - System C scheduling
  - Temporal Decoupling
  - Interrupt Handling in Instruction Set Simulators (ISS)

Introduction

- An interrupt is a request to the processor to suspend its current program and transfer control to the *Interrupt Service Routine* (ISR).
- A developer must know if the design is
  - Processor intensive
  - Interrupt intensive
- Interrupt handling should be
  - Timely (esp. when safety critical)
  - Memory and processor efficient

Interrupts in Virtual Prototyping

- A key usecase of any Virtual Prototype(VP) is system software development
- Use of Interrupts is common in system software
- Important for developers to understand interrupts and prevent associated issues – interrupt overloading, stack overflows etc.
- Empirically its known, interrupts
  - Hard to get completely right
  - Difficult to track down

With right means, VP can be a powerful tool to design efficient software for interrupt based systems

Challenges in Debugging Interrupts

- Peculiar/rare preconditions
- Multiple sources and sinks
- Interrupt Enable/Disable
- Masks/Priority settings
- Missed Interrupts
- Separate Execution Flow
- Nested Interrupts, Reentrancy
- Context management/Stack Corruption
- Multi-level ISR, Delayed Procedural Calls (DPC)
- Unnoticed Faults/Violations/Traps

Additional challenges in a VP

If the VP uses temporarily decoupled ISS :

- Interrupts are only handled at quantum boundaries.
- Interrupts external to VP (e.g. pulse from USB) can be missed
- A large static quantum leads to
  - Loss of Inter Processor Communication
  - Missed Peripheral Interrupts
  - Timing Issues

Shortcomings in existing methodology

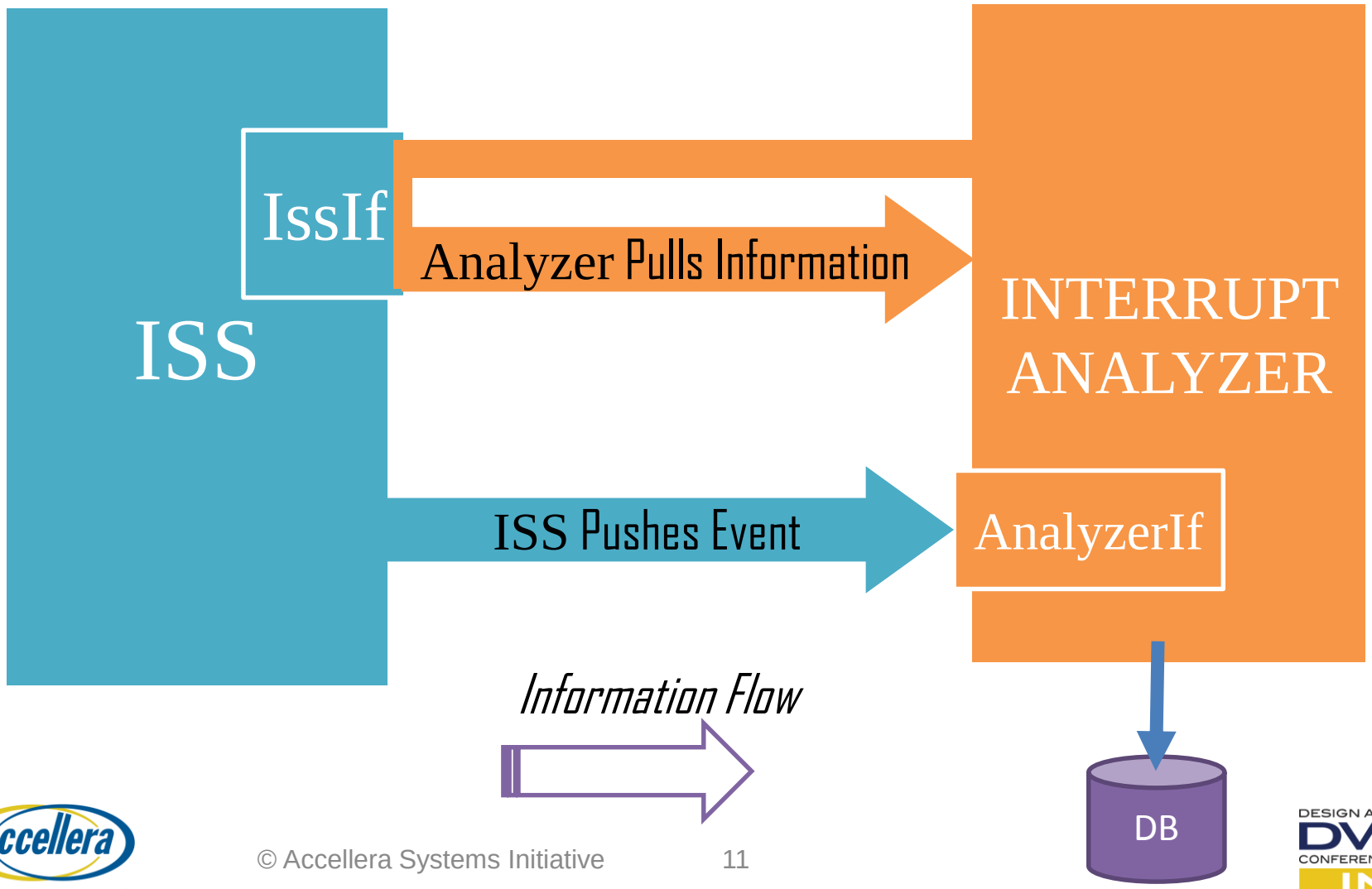
- Investigate various traces
  - Debug Log trace
  - Instruction, function, register trace
  - Pin Value trace
- Verify correct execution by
  - Assembly level debugging
  - Putting breakpoints on pins/registers/instructions
  - Examining values in pins, registers
- Manual, time and effort intensive
  - Sparsely distributed information
  - Time to converge to actual issue may take multiple days

A solution helps if it

- Is easy to integrate with an ISS
- Meaningfully presents the distributed information on
  - Events, register settings and actual handling
  - Latencies and execution of ISR
  - Missed/Pending/Masked interrupts
- Helps developer assess if it's an interrupt related problem
- Helps developer to profile the software

Proposed Interrupt Analyzer based Methodology

A Quick Glance



How to integrate an ISS

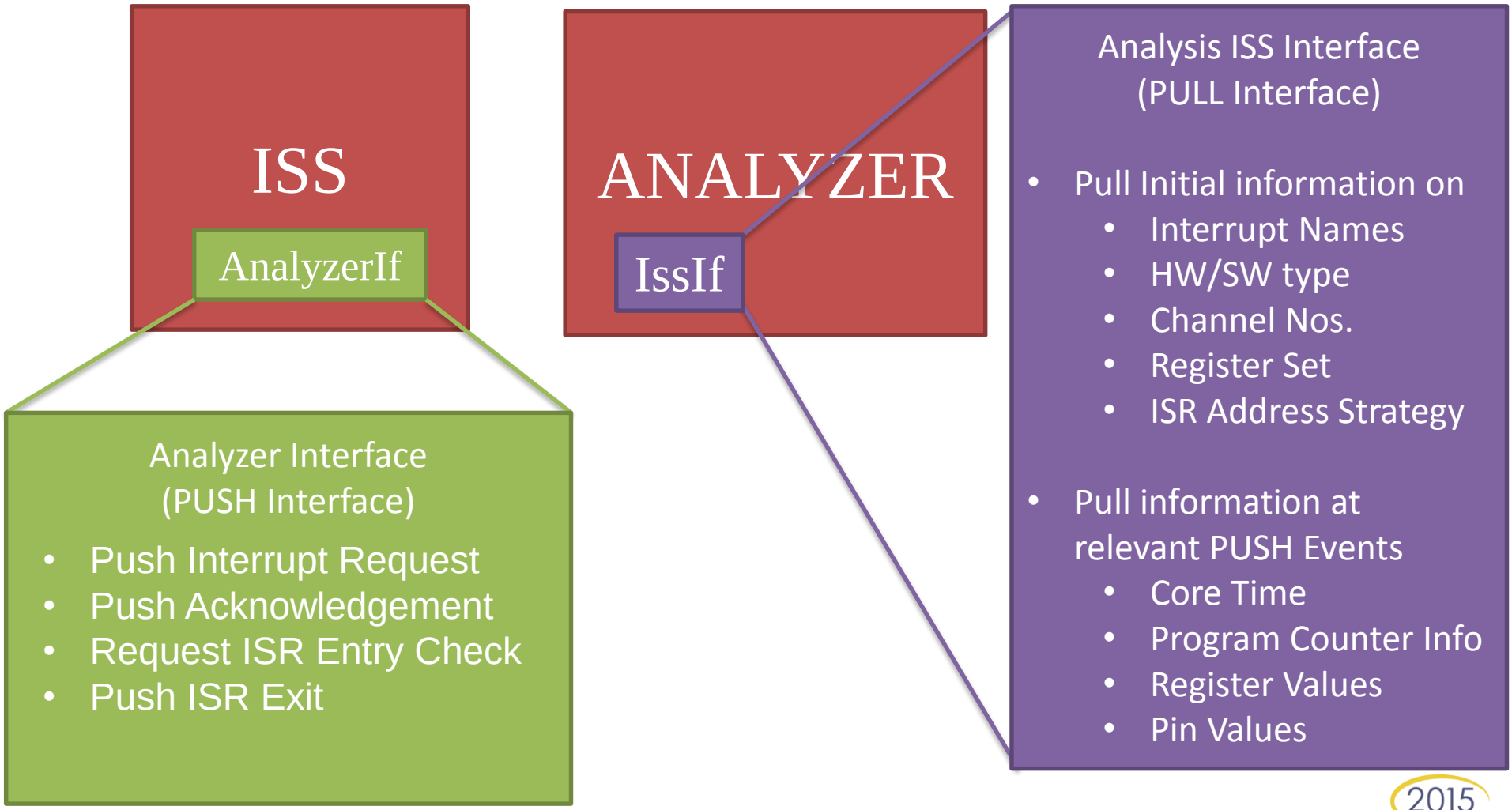
Implement an IssIf to

- Register interrupt capabilities
  - HW , SW Interrupts
  - No. of channels for HW
  - HW IRQ pins
  - HW ACK pins
  - ISR Addresses
- Allow Analyzer to fetch
  - Core Time
  - Register Values
  - Program Counter
  - Pin Values

Use AnalyzerIf to

- Signal Pin Events:
  - IRQ
  - ACK
- Signal ISR Event:
  - Request check/Notify for ISR entry
  - Notify returns from ISR

Implementation Overview

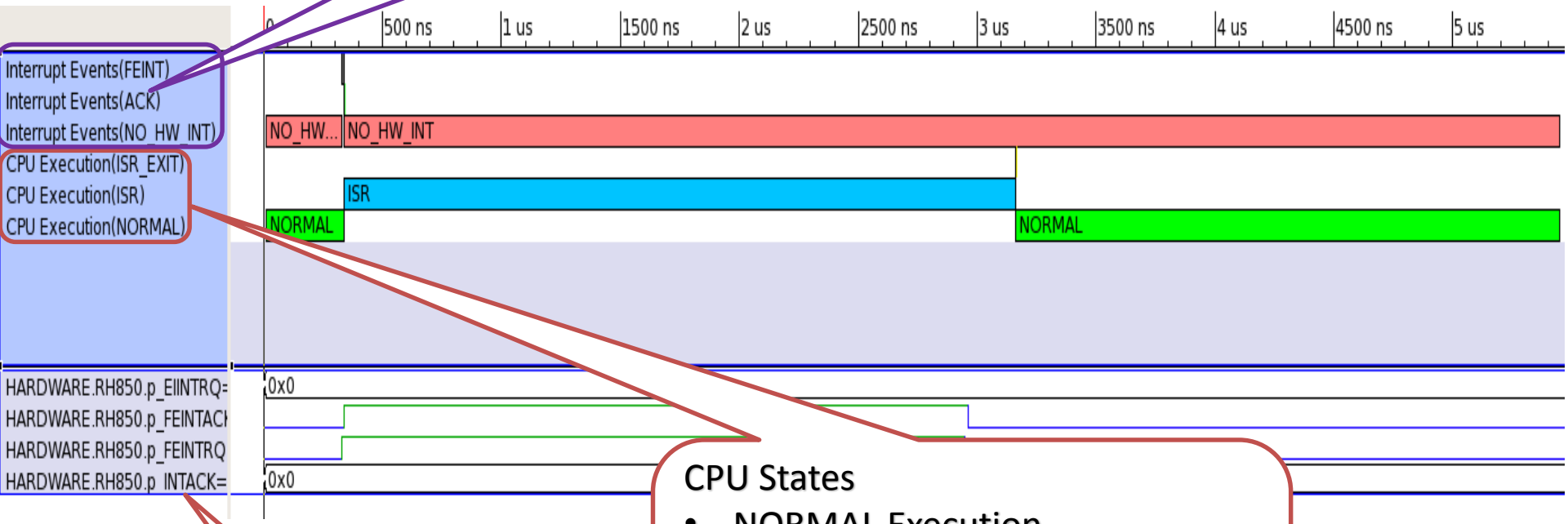


Output: Informatics

Analysis Elements	Details covered
Pin Events	IRQ, ACK, Messages
ISR Events	Entry, Exit, Fall-Through
Latencies	ACK Response, ISR Duration (DPC not covered)
IRQ Special Events	Masked IRQ, Pending IRQ, Missed IRQ
Register Set	Mask, Status and Cause registers

Timeline Chart

Interrupt states
- Interrupt Request (e.g. For RH850 FEINT, EIINT)
- Acknowledgement (ACK)
- No Hardware Interrupt(NO_HW_INT)



Pin Events

CPU States
- NORMAL Execution
- ISR
- ISR_EXIT
- ISR_FALLTHROUGH

accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Information Table

Example 1 from Use Cases

Object	State	CoreTime	FEIC	FEPC	FEPSW	ISRTT	LastPC	PC	PinName	RT	Count	% Duration	Aver
CPU Execution	NORMAL										1	6.06%	
Interrupt Events	NO_HW_INT										1	5.87%	
Interrupt Events	FEINT	330000	0	0	32	0	66988	66994	FEINTRQ	0	1	0.18%	
CPU Execution	ISR	340000	240	66996	196616	0	66996	240	0	0	1	51.74%	
Interrupt Events	ACK	340000	0	0	32	0	66994	66996	FEINTACK	10000	1	0%	
Interrupt Events	NO_HW_INT	340000	0	0	32	0	66994	66996	FEINTACK	0	0	93.76%	
CPU Execution	ISR_EXIT	3160000	240	66996	196616	2820000	250	66992	0	0	1	0%	
CPU Execution	NORMAL	3160000	240	66996	196616	0	250	66992	0	0	0	42.02%	

CoreTime

Registers

ISR Duration computed at ISR Exit

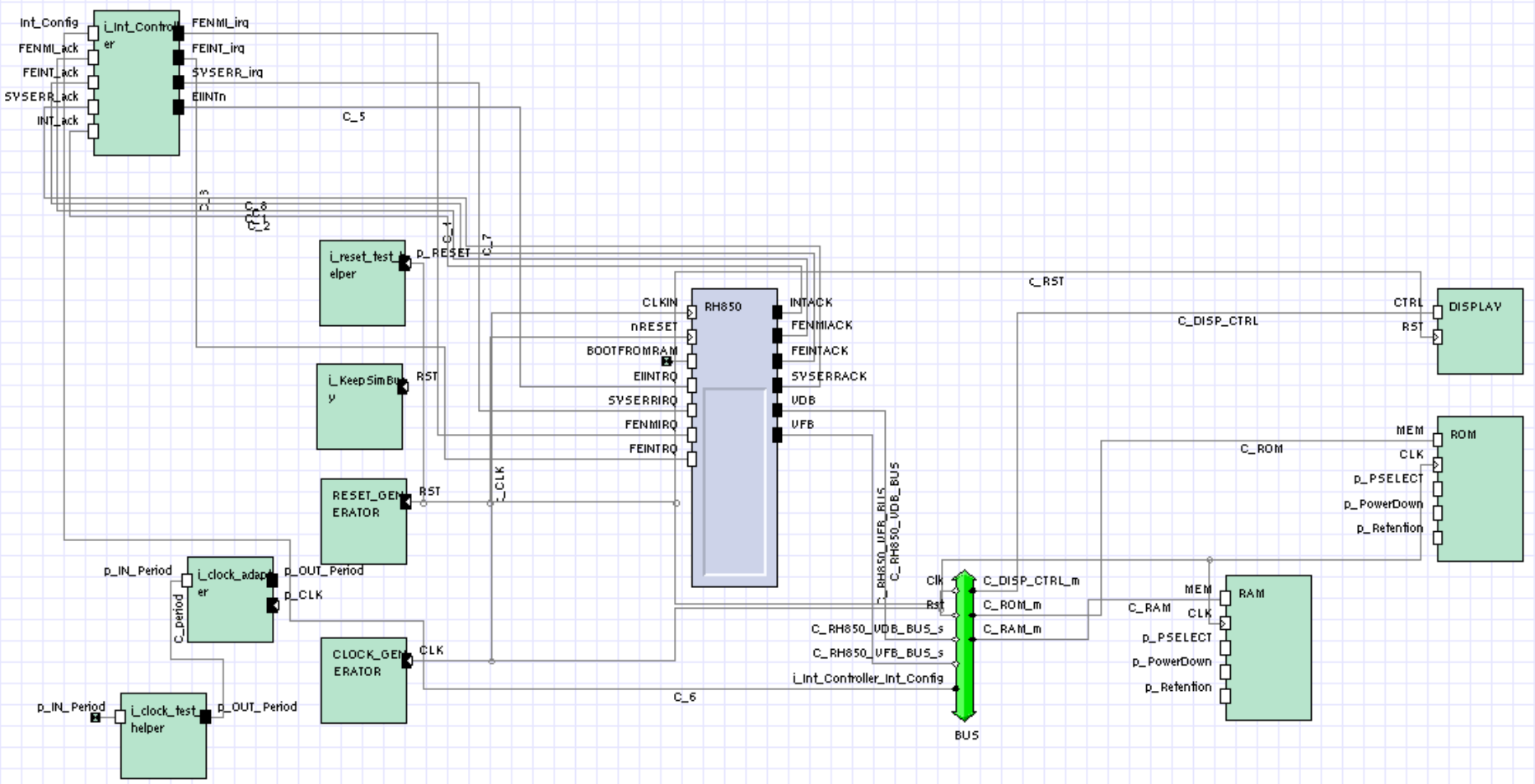
Response Time at Acknowledgment

Utilization

RH850 Based Adaption

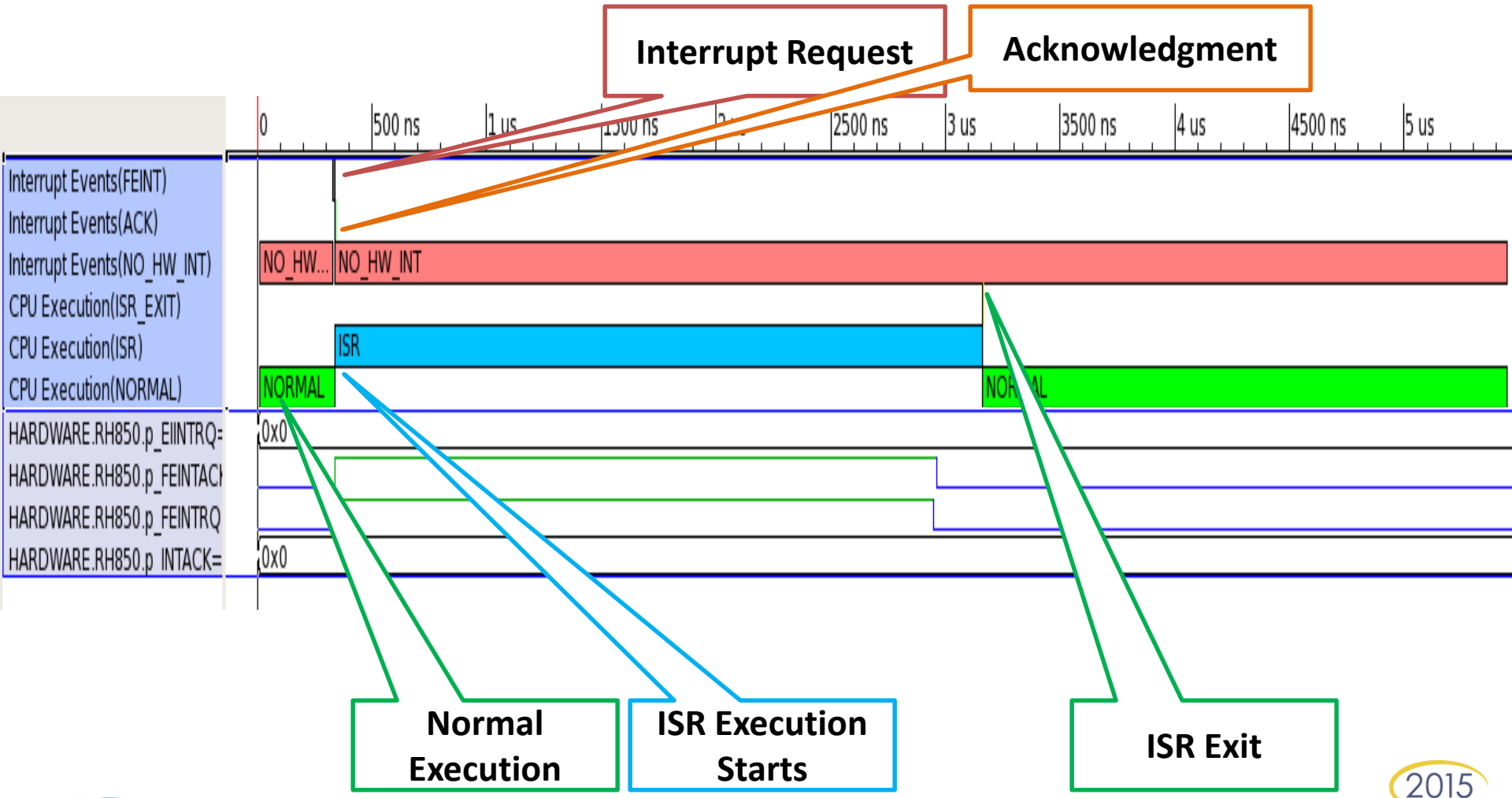
- A platform with Renesas RH850 ISS interfaced with an interrupt controller and peripherals
- Quantum: Both dynamic and static configurations
- Virtual Platform is designed running in Synopsys Virtualizer environment

Platform in use

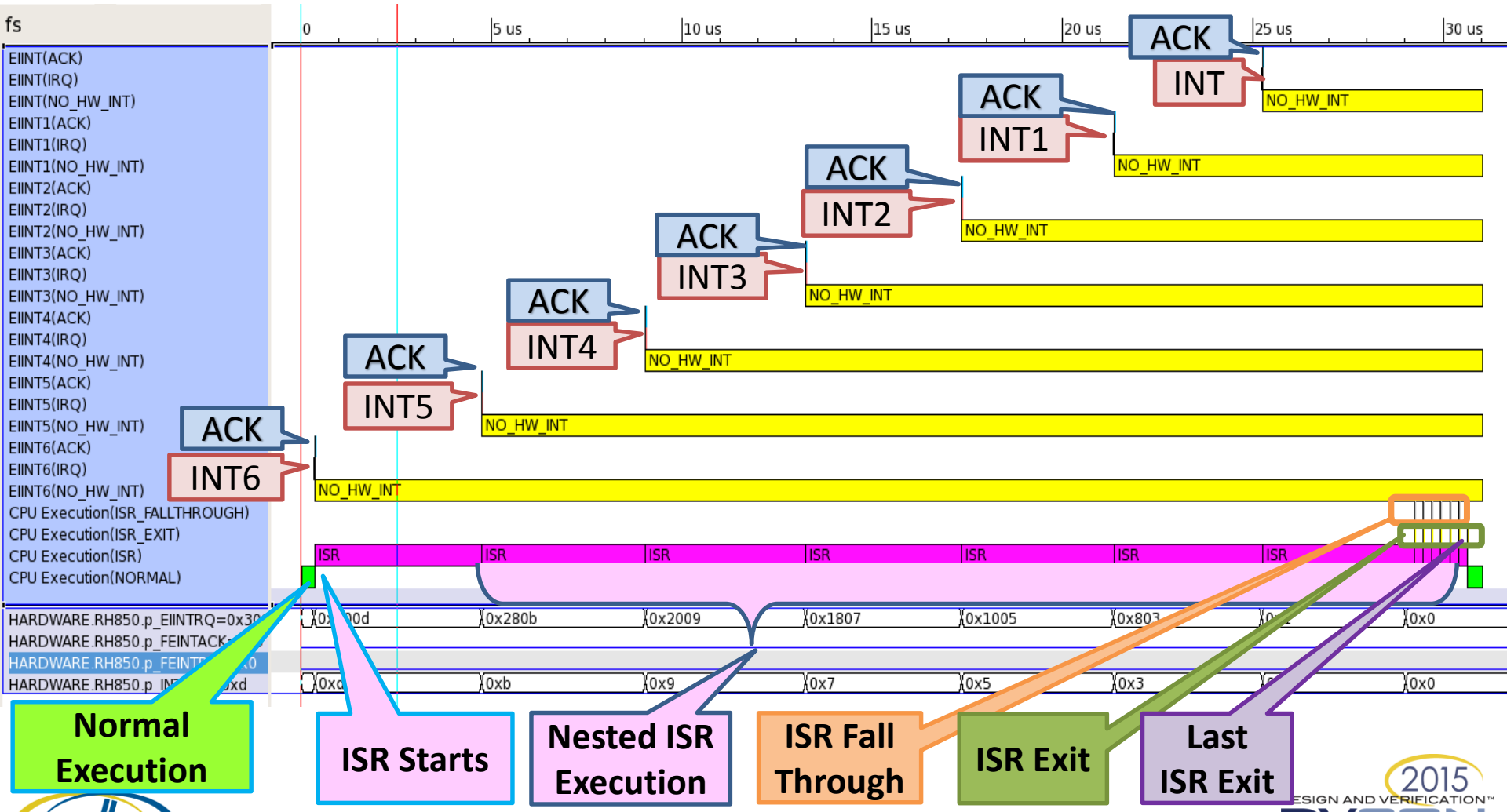


Example Use Cases

(1) Analyzing a HW Interrupt:



(2) Analyzing Nested Interrupts



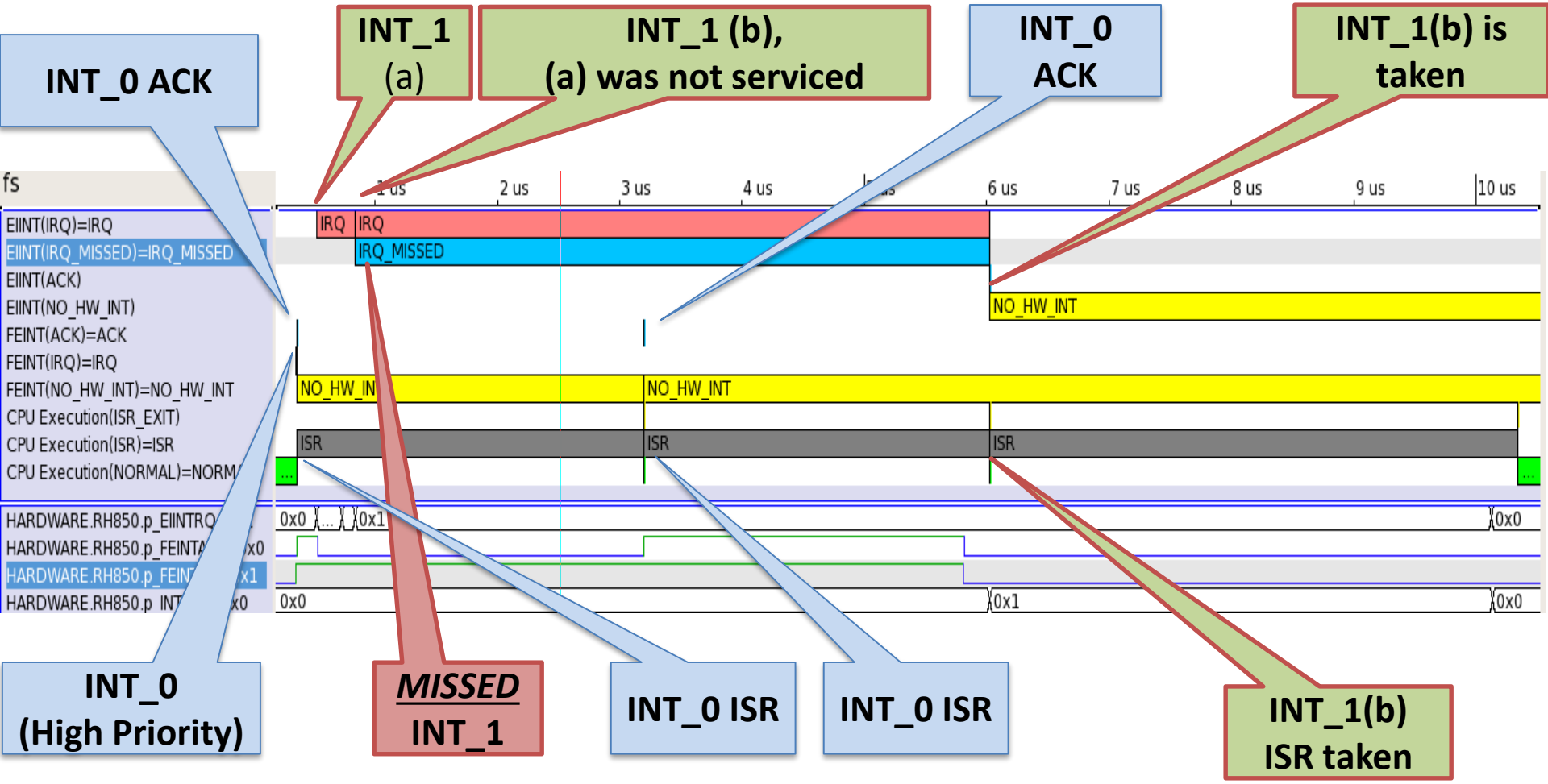
(2) Analyzing Nested Interrupts

Object	State	CoreTime	ISRName	ISRTurnaroundTime
CPU Execution	ISR_EXIT	30400000	EIINT5	25650000
CPU Execution	ISR_EXIT	30170000	EIINT4	21120000
CPU Execution	ISR_EXIT	29940000	EIINT3	16690000
CPU Execution	ISR_EXIT	29710000	EIINT2	12360000
CPU Execution	ISR_EXIT	29480000	EIINT1	8130000
CPU Execution	ISR_EXIT	29250000	EIINT	4000000

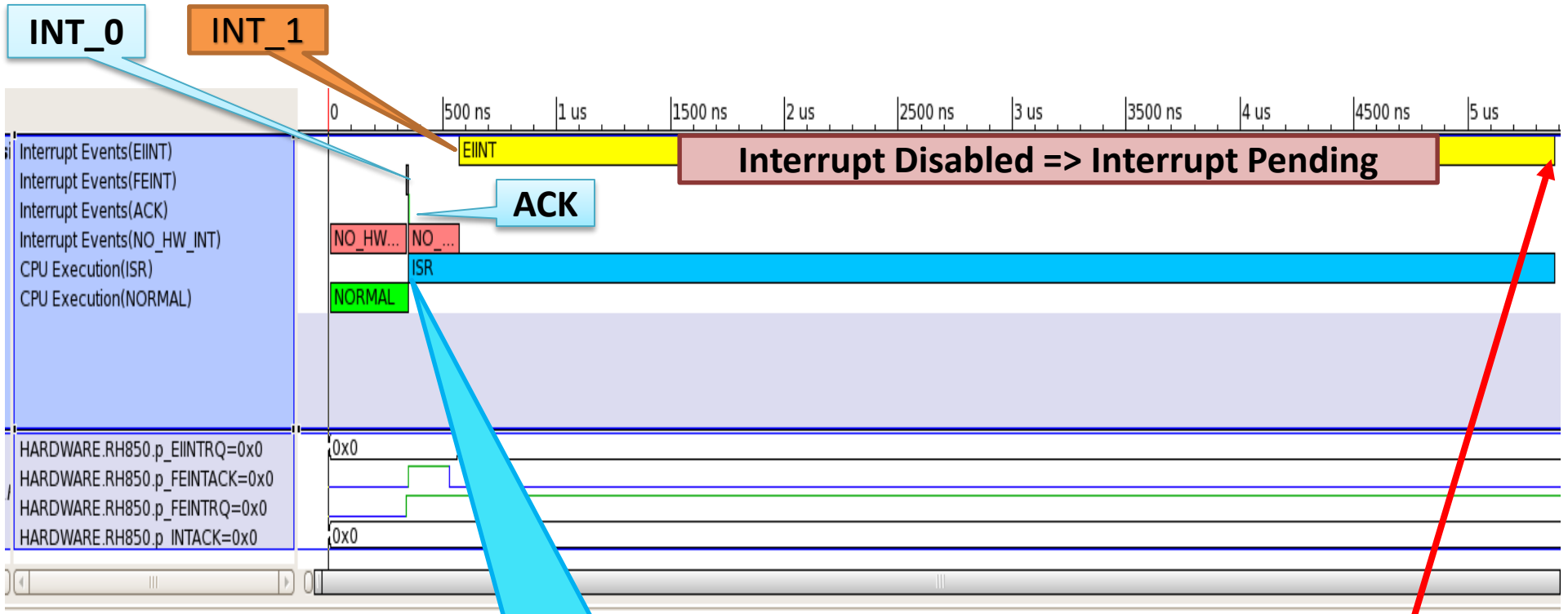
Object	State	CoreTime	PinName	ResponseTime
EIINT	ACK	25250000	EIINTACK	10000
EIINT1	ACK	21350000	EIINTACK	10000
EIINT2	ACK	17350000	EIINTACK	10000
EIINT3	ACK	13250000	EIINTACK	10000
EIINT4	ACK	9050000	EIINTACK	10000
EIINT5	ACK	4750000	EIINTACK	10000
EIINT6	ACK	350000	EIINTACK	10000

Object	State	CoreTime	ISRName	% Duration
CPU Execution	ISR	350000	EIINT6	14.18%
CPU Execution	ISR	4750000	EIINT5	13.86%
CPU Execution	ISR	9050000	EIINT4	13.54%
CPU Execution	ISR	13250000	EIINT3	13.21%
CPU Execution	ISR	17350000	EIINT2	12.89%
CPU Execution	ISR	21350000	EIINT1	12.57%
CPU Execution	ISR	25250000	EIINT	12.89%

(3) Analyzing Missed Interrupts



(4) Analyzing Pending Interrupts



Interrupt assertion activates a HW timer.

ISR Starts, disables interrupts. HW timer stopped

INT_1 assertion reactivates HW timer

Timer expires, terminates simulation

Conclusions

Analyzer consolidates the distributed debugging techniques and helps

- Organize interrupt data in tables
- Graphically assess interrupt handling
- Point out interrupt anomalies
- Track software exceptions
- Expedite debugging and profiling
- Expedite setting of optimal static quantum
- Present a post-silicon use case for debugging interrupt issues

Questions