Unleash the Full Potential of Your Waveforms

From Extra-functional Analysis to Functional Debug via Programs on Waveforms

Daniel Große, Lucas Klemmer
Institute for Complex Systems (ICS)
Web: jku.at/ics | wal-lang.org
Email: daniel.grosse@jku.at, lucas.klemmer@jku.at

2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

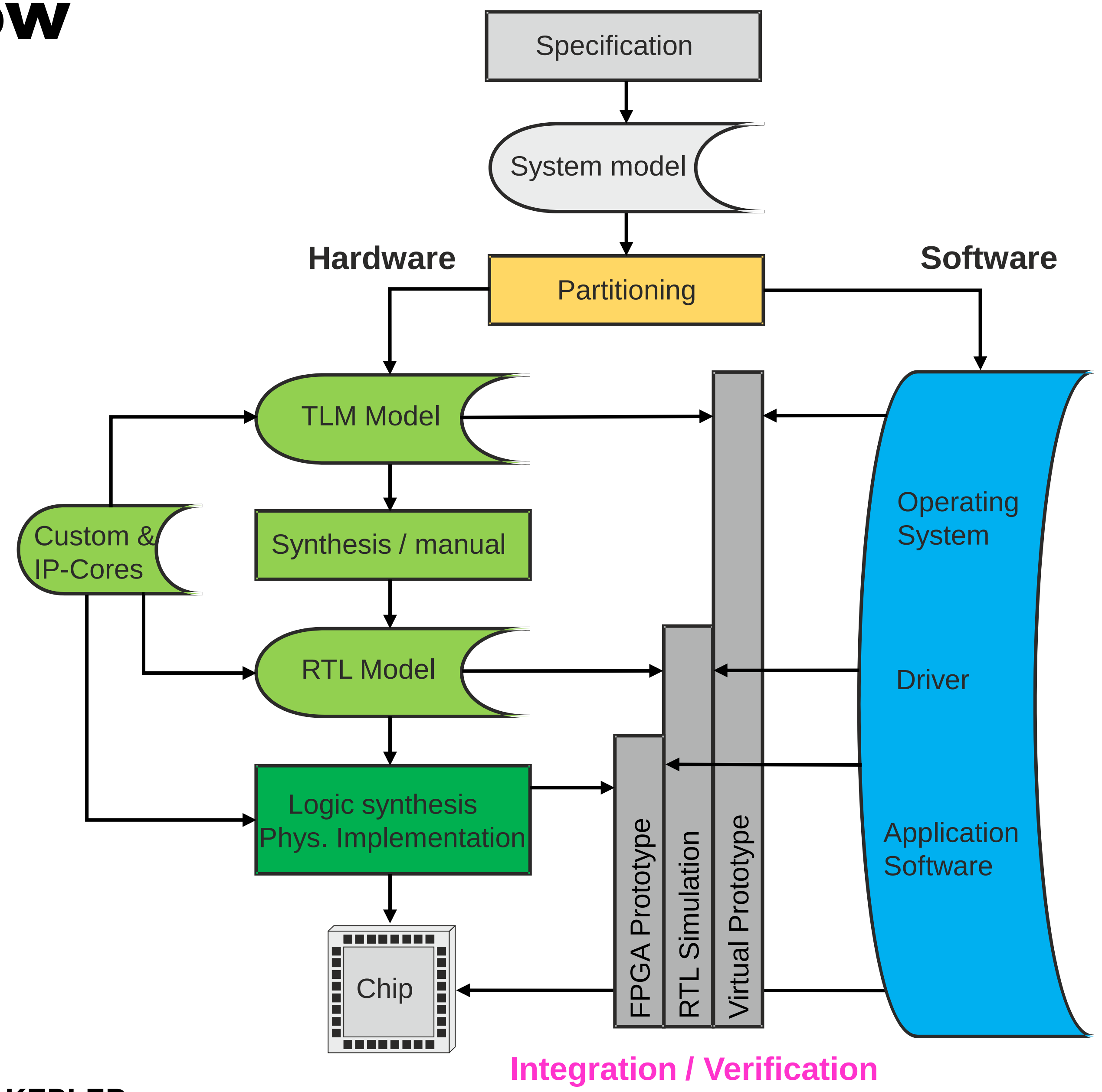
JOHANNES KEPLER UNIVERSITY LINZ

... Potential of Your Waveforms ...

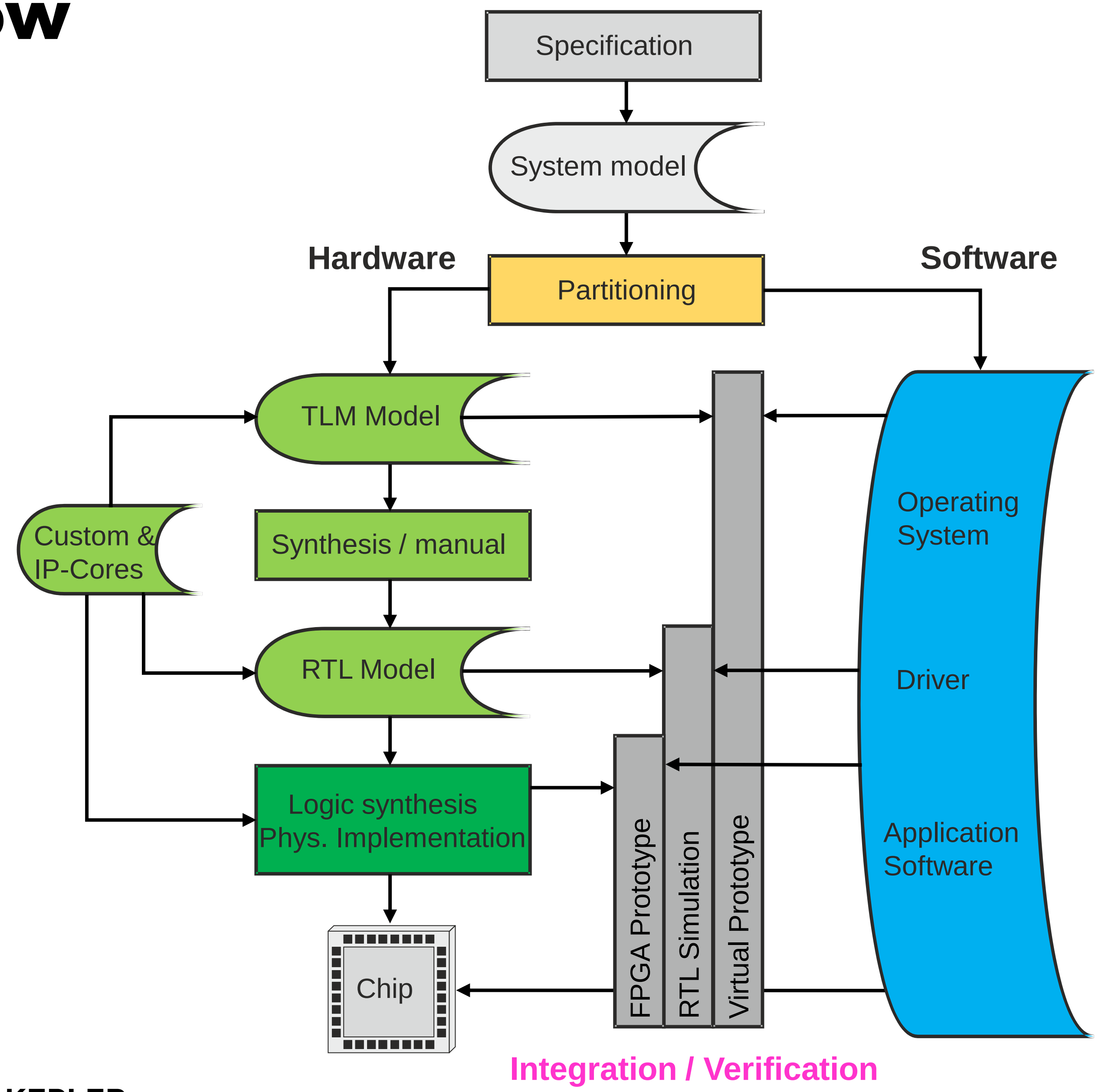
JOHANNES KEPLER UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

Design Flow


Specification
System model
Hardware
Partitioning
Software
TLM Model
Custom & IP-Cores
Synthesis / manual
RTL Model
Logic synthesis Phys. Implementation
Chip
FPGA Prototype
RTL Simulation
Virtual Prototype
Operating System
Driver
Application Software
Integration / Verification

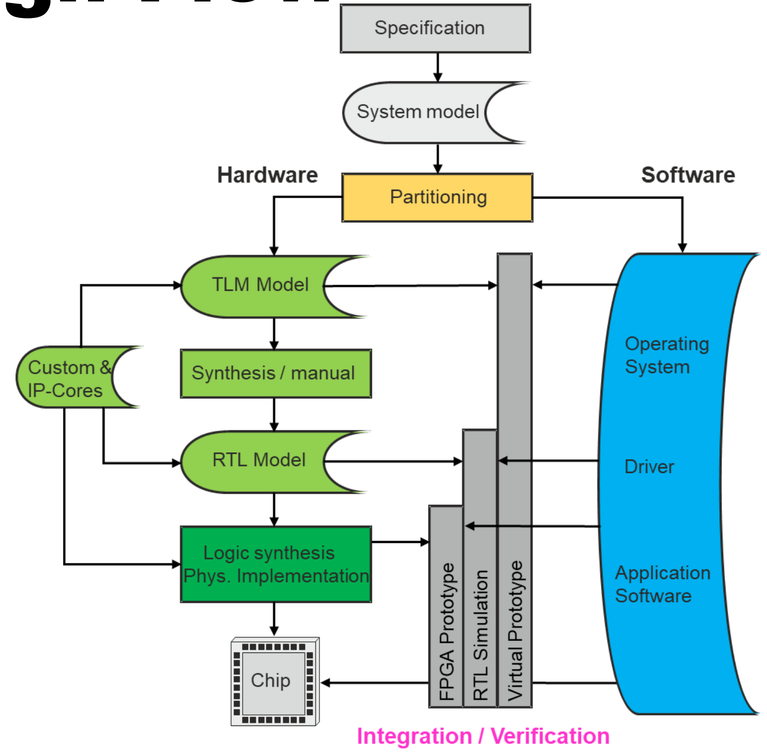
Design Flow



JOHANNES KEPLER UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

Design Flow



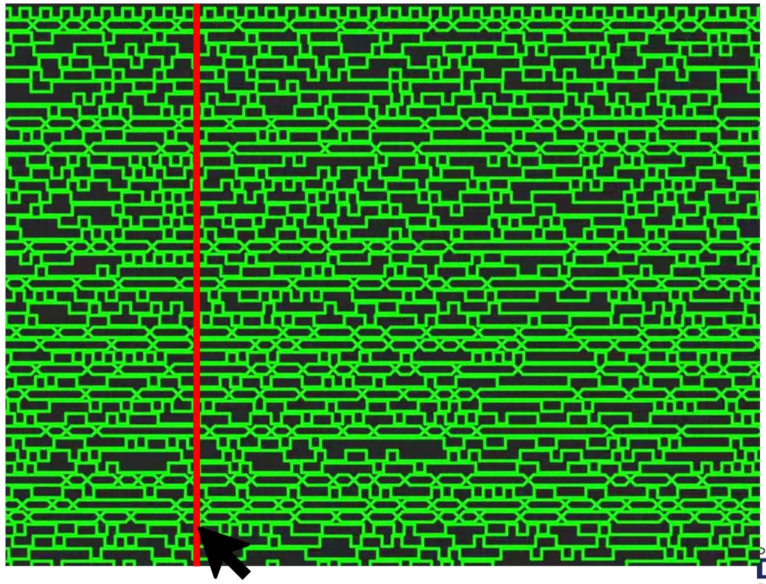
Waveforms

- HW block is alive
- HW shows expected behavior
- Communication works
- Assembler instructions run
- Performance as expected
- …

Waveforms

- Waveforms are great!
- A central data format for HW development
  - Produced by simulators, formal tools, logic analyzers, …
- They contain incredible amounts of information
  - performance, correctness, data/control flow, optimization, …
- However
  - 100% manual process
  - Only small slice of data visible at once
  - Only for “simple” signal relations
  - Analysis not automated
  - Data without analysis just noise

Data is not Information



WAL: Waveform Analysis Language

- WAL is *Domain Specific Language* (DSL) to express HW analysis problems
- Specialized language constructs for HW domain:
  - Waveform signals, Time, Hierarchy, Signal relations (bus interfaces)
- Not just true/false expressions, much more than SVA, PSL, …
- Full capabilities of scripting languages (functions, external libraries, …)
- Quickly analyze waveforms
- **Alternative to**
  - **Custom testbench extensions**
  - **Custom scripts**

How to Read WAL Expressions

- This is a **number**
  - `5`
- These are also numbers
  - `0xff, 0b1101`
- This is a **variable**
  - `my_var`
- And these are also variables
  - `RD-START, top.core1.run`
- This is a **string**
  - `“hello, DVCON Europe!”`

- The same in Python
  - `5`
  - `0xff, 0b1101`
  - `my_var`
  - `RD-START, top.core1.run?`
  - `“hello, DVCON Europe!”`

JOHANNES KEPLER UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

How to Read WAL Expressions (2)

- This is a **list**
  - `(5 1 abc)`
- If the first element is a function name the list is a function application
  - `(+ 1 2)`
  - `(+ 1 2 3 …)`
  - `(print “hello”)`
  - `(print “Sum: “ (+ 1 2))`

- The same in Python
  - `[5, 1, abc]`
  - `1 + 2`
  - `1 + 2 + 3 + . + ..`
  - `print(“hello”)`
  - `print(“Sum: “, 1 + 2)`

How to Read WAL Expressions (2)

- This is a **list**
  - `(5 1 abc)`
- If the first eleme
  the list is a func
  - `(+ 1 2)`
  - `(+ 1 2 3 …)` + ..
  - `(print “he`
  - `(print “Su` 1 + 2)

```
function(a, b)
```

$\Downarrow$

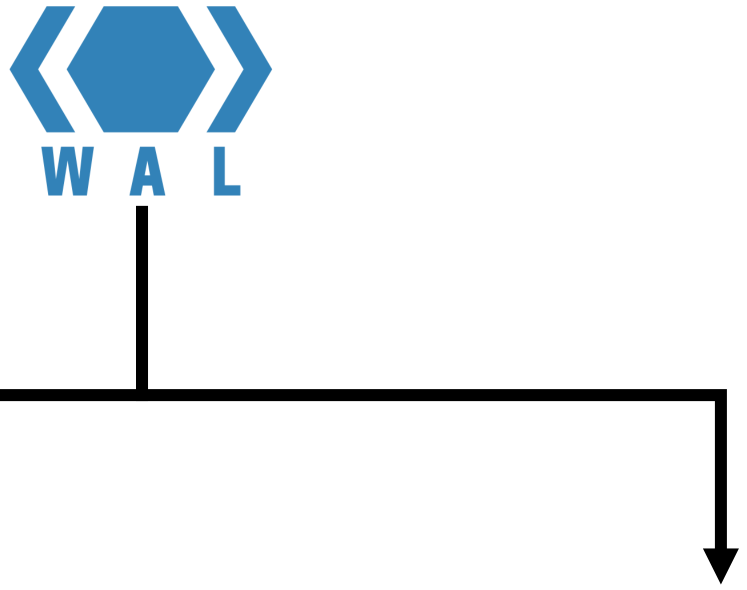
```
(function a b)
```

Arithmetic and Logic Operators

- Arithmetic Operators
  - `+`, `-`, `*`, `/`
  - `(+ 1 2) => 3`
  - `(+ 1 (- 4 2)) => 3`

- Logic Operators
  - `!`, `&&`, `||`, `=`, `!=`, `>`, `<`, `>=`, `<=`
  - `(&& #t #t) => #t`
  - `(! (&& #t #t)) => #f`
  - `(> 5 4) => #t`

How to get WAL



- The “original” WAL Interpreter
- Written in Python
- Basis for our research
- Easy to extend, experiment
- Limited performance but proven to be useful
- https://github.com/ics-jku/wal/

- The “new” Interpreter
- Written in Rust
- Goals: Top performance and usability
- Focus on real-life problems
- Wider range of platforms

Tutorial Website

- This tutorial is **very hands on**
- Online Waveform Explorer
  - Integrated waveform viewer
  - Prepared examples
  - Run programs, read output
  - Everything runs locally
  - Drop in your own waveforms
- No installation, no downloads
- Enabled by web assembly and Rust
- Runs on phones too!

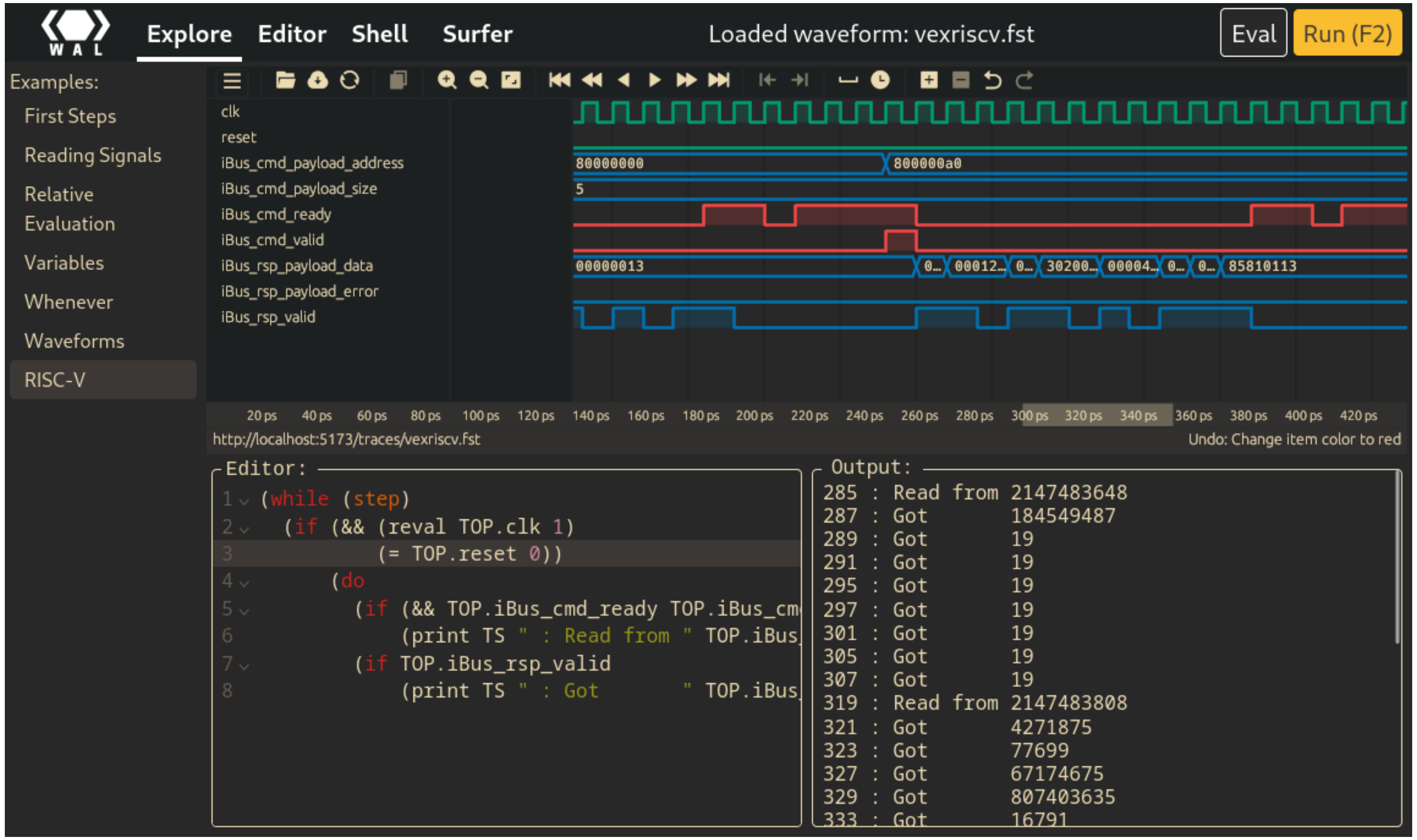
Only available during talk

https://[redacted]al-lang.org

Only available during talk

Install the Python version to follow examples

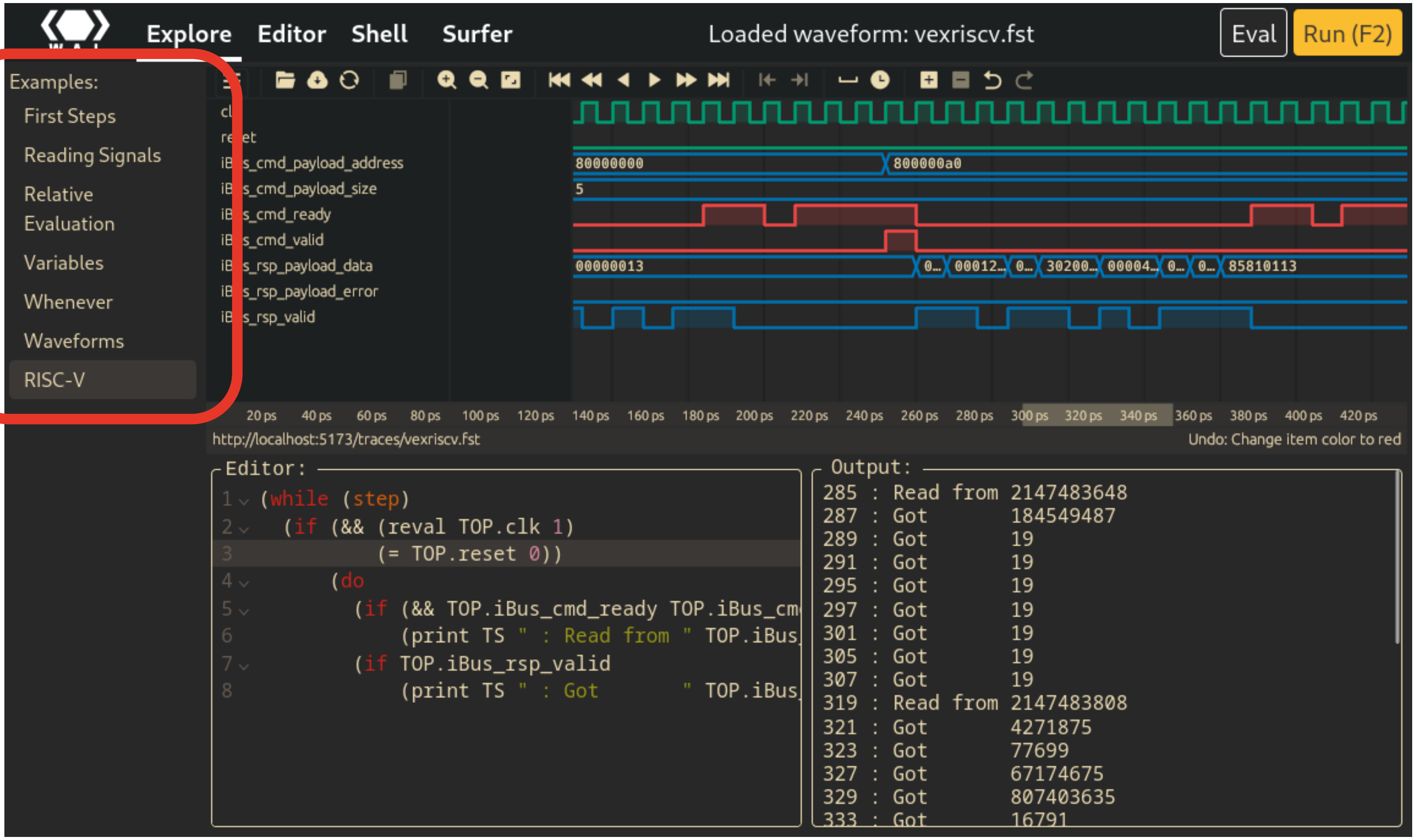
Tutorial Website



```
(while (step)
  (if (&& (reval TOP.clk 1)
          (= TOP.reset 0))
      (do
        (if (&& TOP.iBus_cmd_ready TOP.iBus_cm
            (print TS " : Read from " TOP.iBus
        (if TOP.iBus_rsp_valid
            (print TS " : Got       " TOP.iBus
```

```
285 : Read from 2147483648
287 : Got       184549487
289 : Got       19
291 : Got       19
295 : Got       19
297 : Got       19
301 : Got       19
305 : Got       19
307 : Got       19
319 : Read from 2147483808
321 : Got       4271875
323 : Got       77699
327 : Got       67174675
329 : Got       807403635
333 : Got       16791
```

Tutorial Website



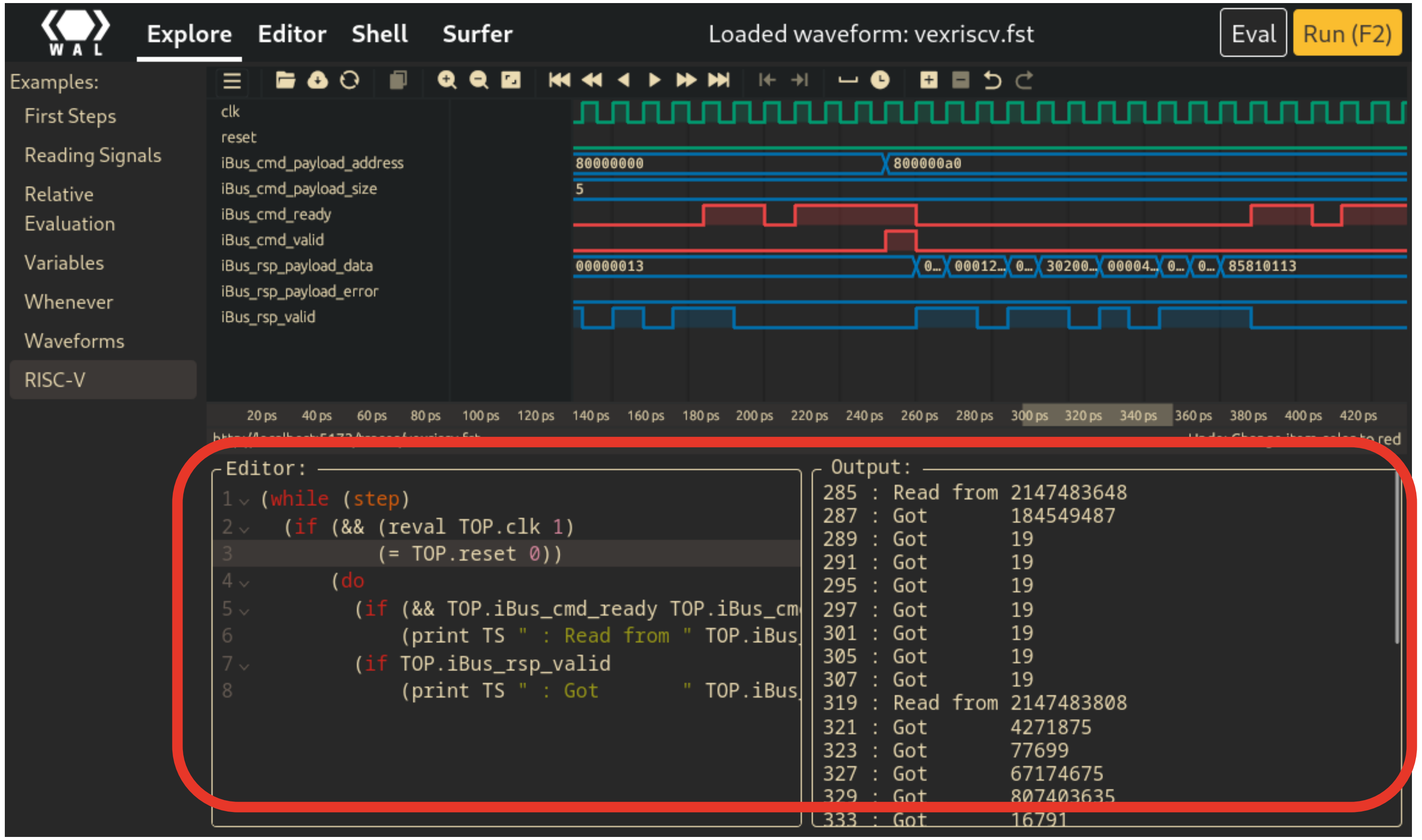
Editor:

```
(while (step)
  (if (&& (reval TOP.clk 1)
              (= TOP.reset 0))
      (do
        (if (&& TOP.iBus_cmd_ready TOP.iBus_cm
            (print TS " : Read from " TOP.iBus_
        (if TOP.iBus_rsp_valid
            (print TS " : Got       " TOP.iBus_
```

Output:

```
285 : Read from 2147483648
287 : Got       184549487
289 : Got       19
291 : Got       19
295 : Got       19
297 : Got       19
301 : Got       19
305 : Got       19
307 : Got       19
319 : Read from 2147483808
321 : Got       4271875
323 : Got       77699
327 : Got       67174675
329 : Got       807403635
333 : Got       16791
```

Tutorial Website



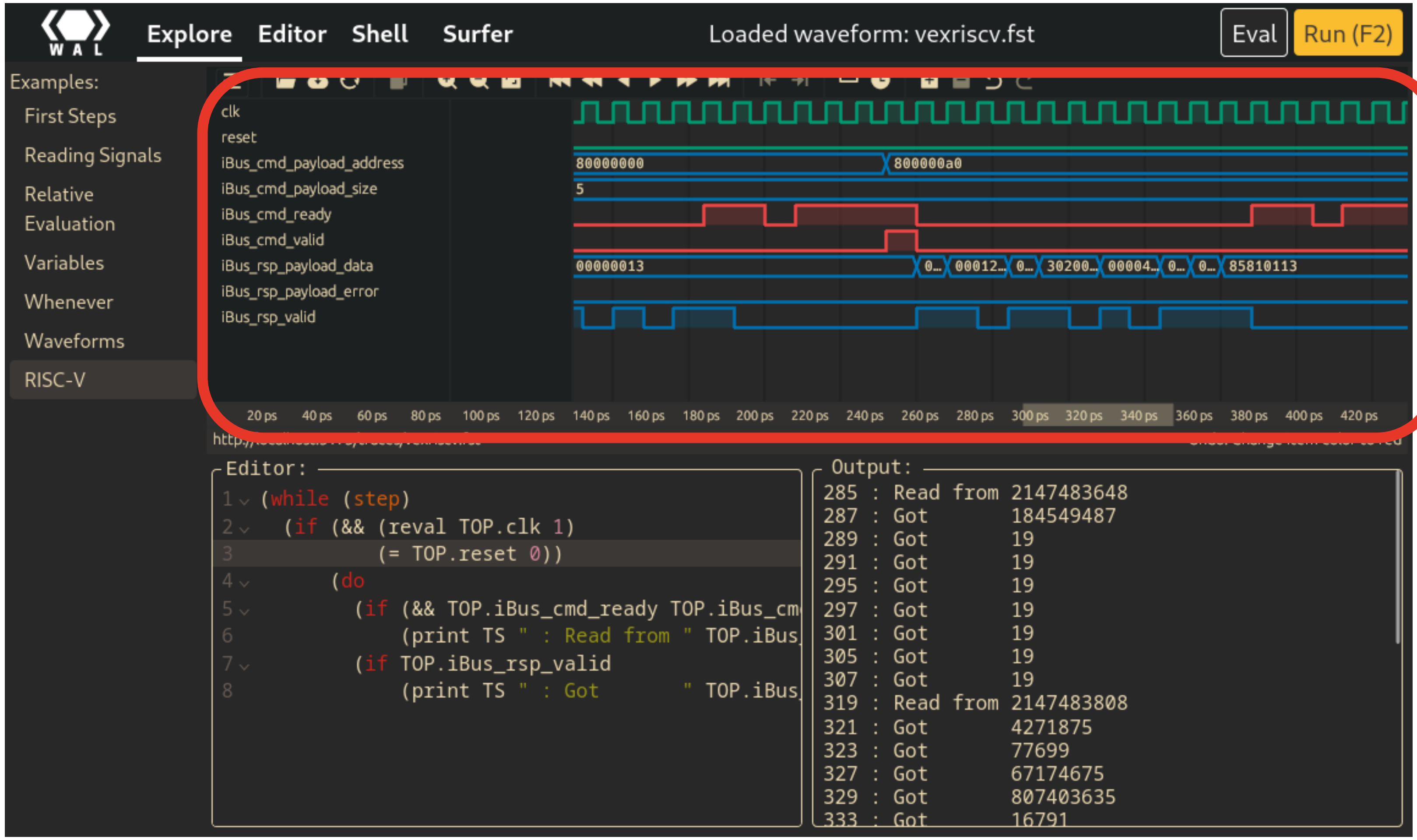
Editor:

```
(while (step)
  (if (&& (reval TOP.clk 1)
          (= TOP.reset 0))
      (do
        (if (&& TOP.iBus_cmd_ready TOP.iBus_cm
            (print TS " : Read from " TOP.iBus
        (if TOP.iBus_rsp_valid
            (print TS " : Got       " TOP.iBus
```

Output:

```
285 : Read from 2147483648
287 : Got       184549487
289 : Got       19
291 : Got       19
295 : Got       19
297 : Got       19
301 : Got       19
305 : Got       19
307 : Got       19
319 : Read from 2147483808
321 : Got       4271875
323 : Got       77699
327 : Got       67174675
329 : Got       807403635
333 : Got       16791
```

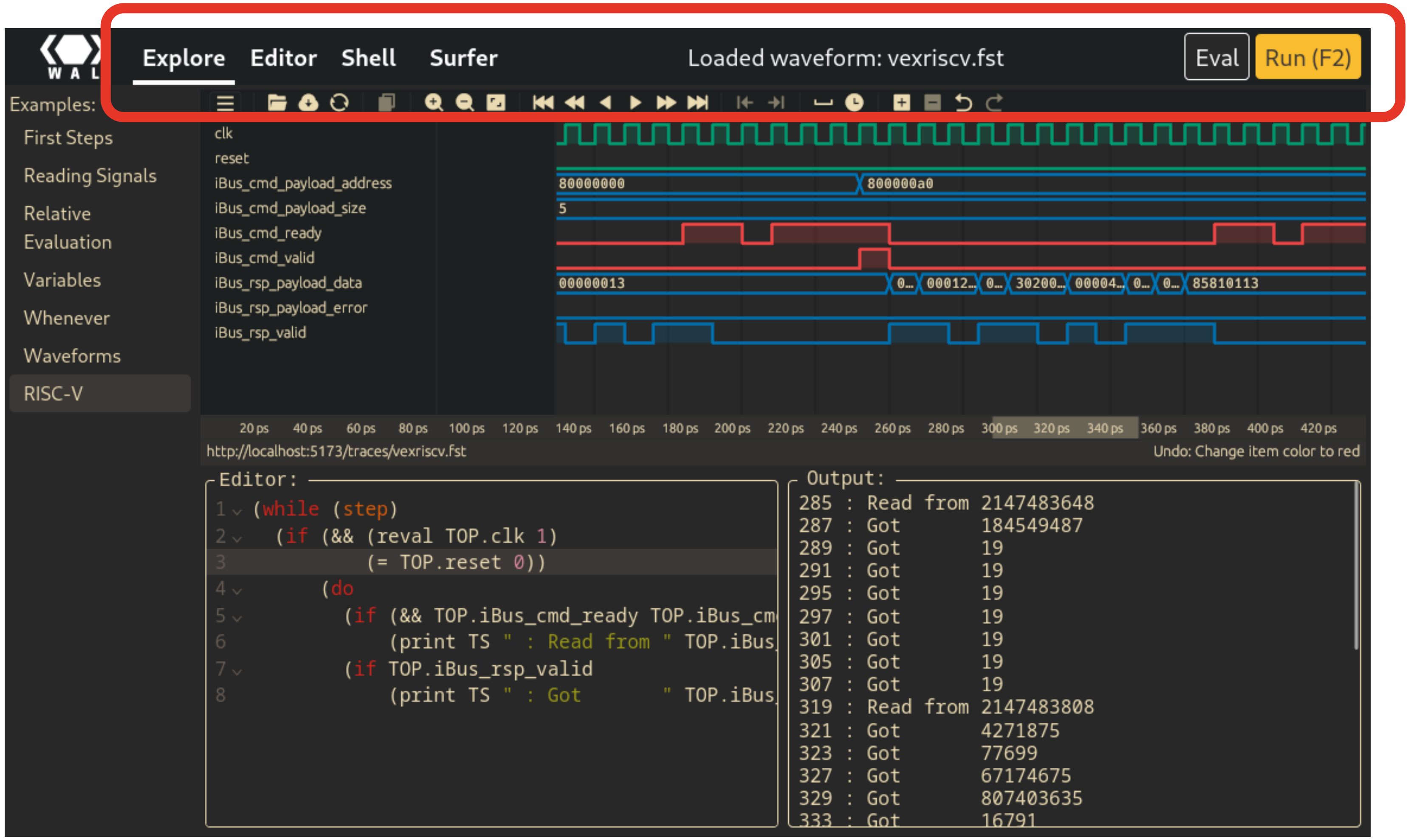
Tutorial Website



```
(while (step)
  (if (&& (reval TOP.clk 1)
          (= TOP.reset 0))
      (do
        (if (&& TOP.iBus_cmd_ready TOP.iBus_cm
            (print TS " : Read from " TOP.iBus
        (if TOP.iBus_rsp_valid
            (print TS " : Got       " TOP.iBus
```

```
285 : Read from 2147483648
287 : Got       184549487
289 : Got       19
291 : Got       19
295 : Got       19
297 : Got       19
301 : Got       19
305 : Got       19
307 : Got       19
319 : Read from 2147483808
321 : Got       4271875
323 : Got       77699
327 : Got       67174675
329 : Got       807403635
333 : Got       16791
```

Tutorial Website



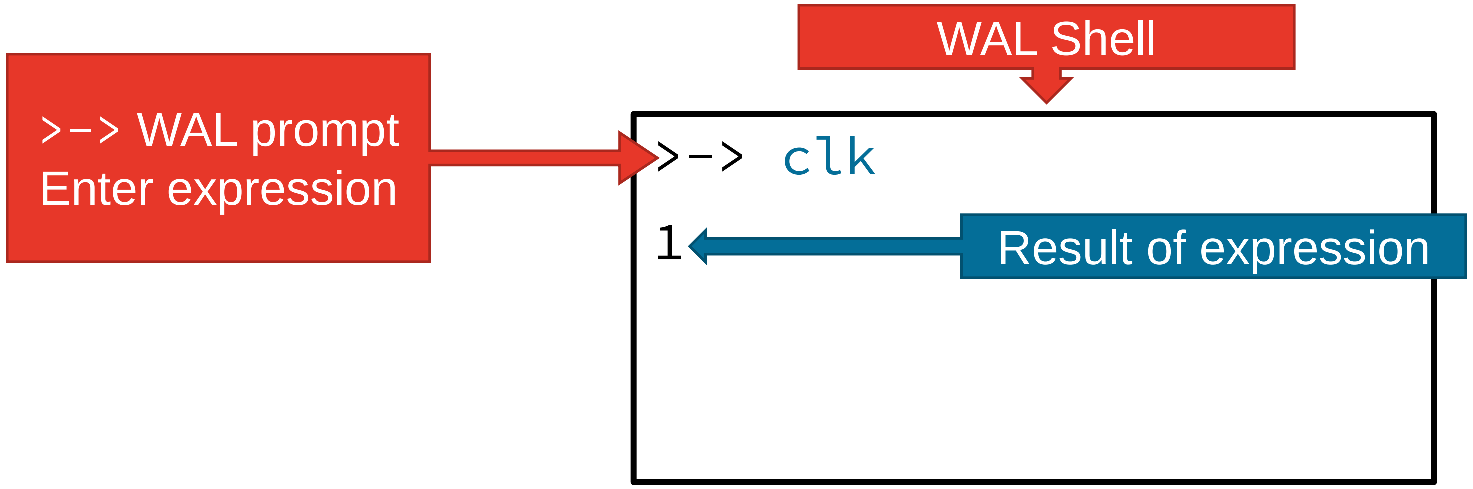
```
(while (step)
  (if (&& (reval TOP.clk 1)
          (= TOP.reset 0))
      (do
        (if (&& TOP.iBus_cmd_ready TOP.iBus_cm
            (print TS " : Read from " TOP.iBus
        (if TOP.iBus_rsp_valid
            (print TS " : Got       " TOP.iBus
```

```
285 : Read from 2147483648
287 : Got       184549487
289 : Got       19
291 : Got       19
295 : Got       19
297 : Got       19
301 : Got       19
305 : Got       19
307 : Got       19
319 : Read from 2147483808
321 : Got       4271875
323 : Got       77699
327 : Got       67174675
329 : Got       807403635
333 : Got       16791
```

JOHANNES KEPLER UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

Hands-On: Shell Examples

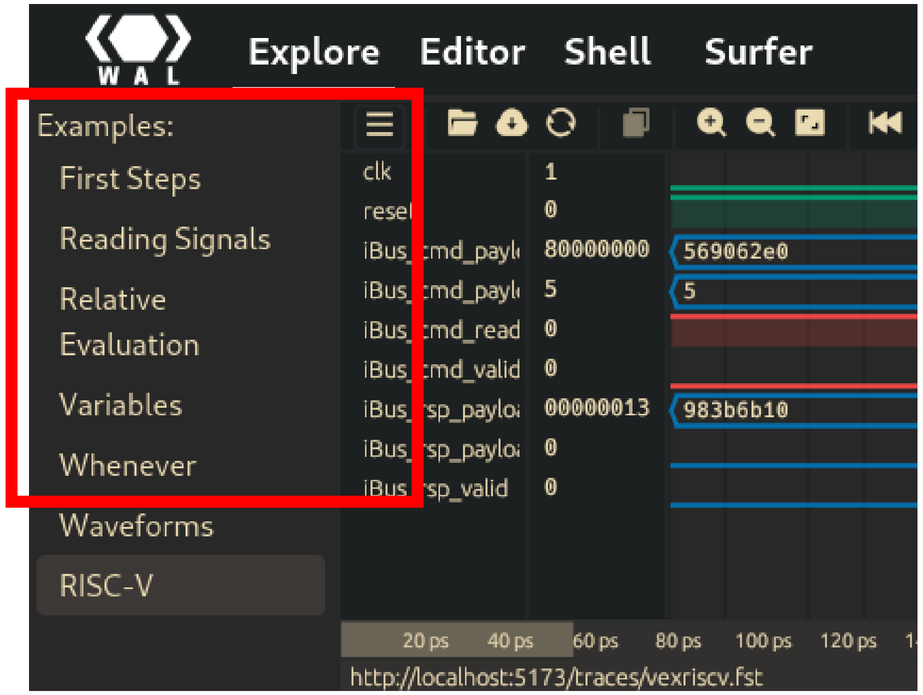


```
>-> clk
1
```

Hands-On: First Steps

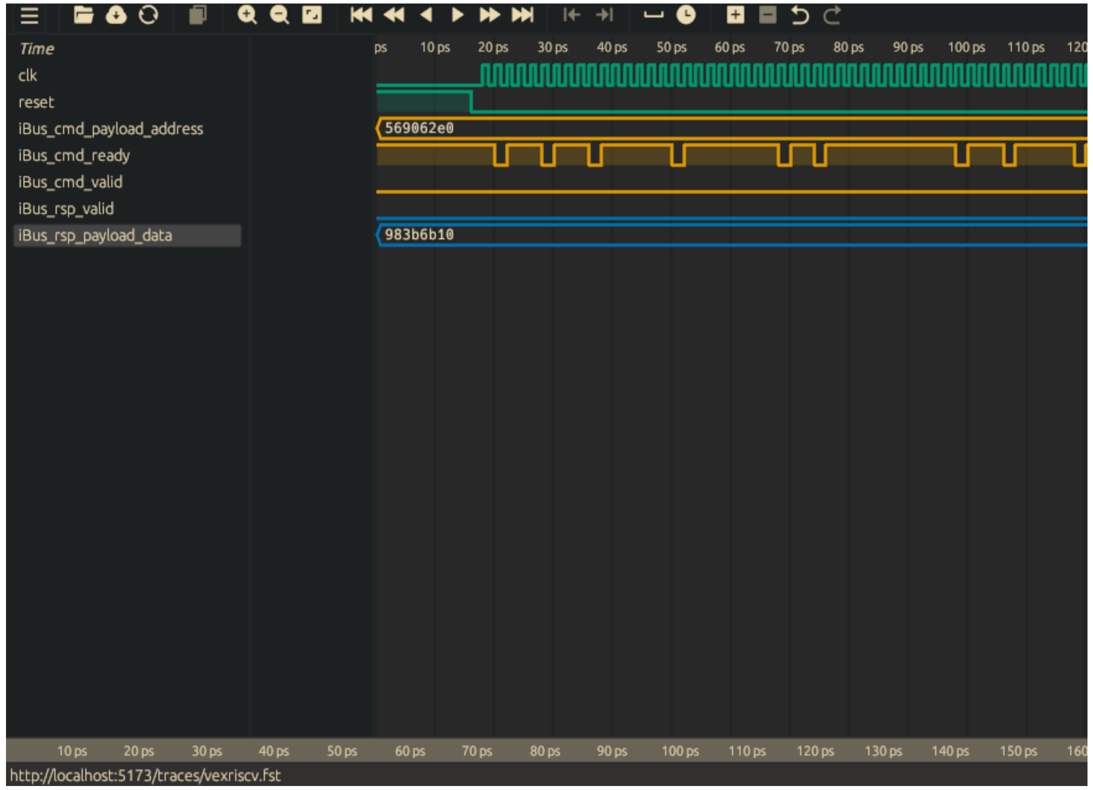
```
>-> 1
1
>-> (+ 1 2)
3
>-> (= 1 2)
#f
```

Selecting a different example:



http://localhost:5173/traces/vexriscv.fst

Side note: Surfer Waveform Viewer



- Modern open-source waveform viewer
  - Co-developed with LIU, Sweden
- Very fast, customizable, flexible
- Traditional mouse based navigation
- **OR**
- **Keyboard driven UI**
- VSCode inspired command line
  - press <SPACE>
    - variable_add …
    - scope_add …

- Visit https://surfer-project.org

JOHANNES KEPLER UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

The WAL Idea

- This is a signal access!

```
(define a 5)
(print (+ a b))
```

- What does this do?

```
a = 5
print(a + b)
```

- **Free variables are signals in waveforms**
- Value depending on:
  - Loaded waveform
  - Time index in the waveform
- In simple terms:
  - **WAL programs run over a waveform and collect information**

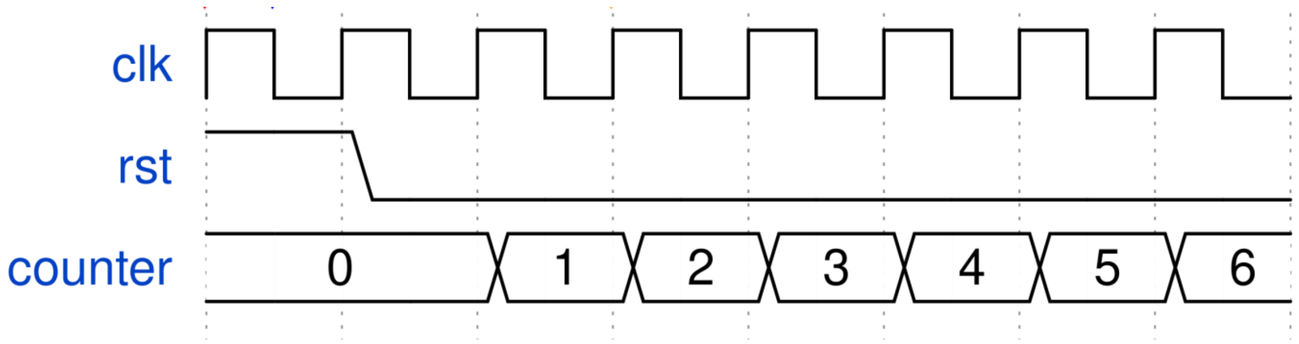
```
Traceback (most recent call last):
  File "error.py", line 2, in <module>
    print(a + b)
NameError: name 'b' is not defined
```

Ouch!

Reading Signal Values (Example)

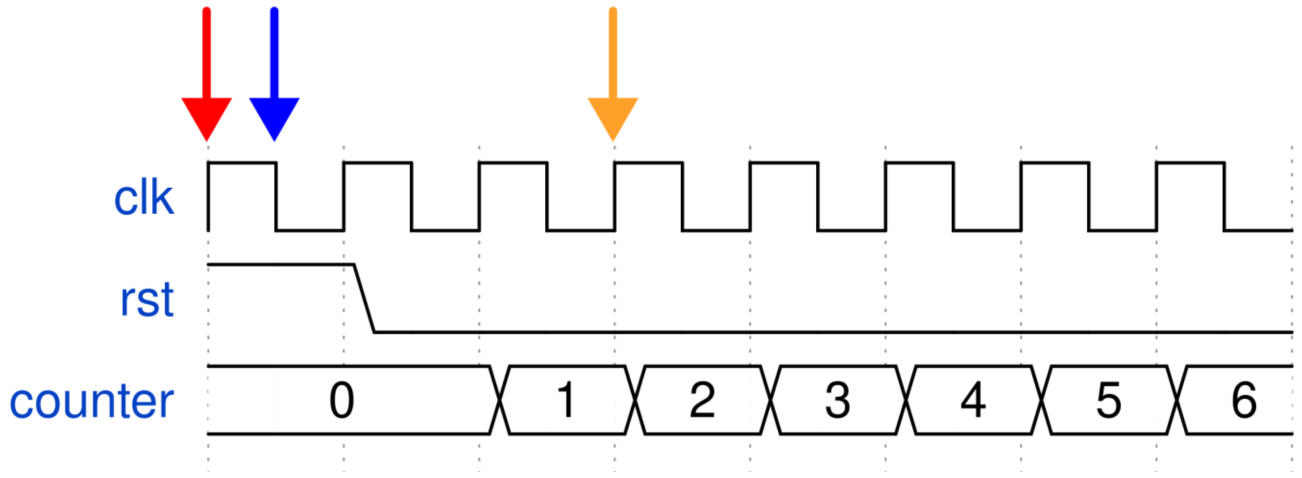
- We have a simple counter
- **index = 0**, after waveform is loaded
- Read a signal by typing it's name
- Move the index with `(step)`

```
0: >-> clk ⇒ 1
   >-> (step 1)

1: >-> clk ⇒ 0
   >-> (step 5)

6: >-> clk ⇒ 1
   >-> counter ⇒ 2
```



Hands-On: Reading Signal Values

```
>-> clk
1
>-> (step 1)
#t
>-> INDEX
1
>-> clk
0
>-> (step 5)
#t
>-> counter
2
```



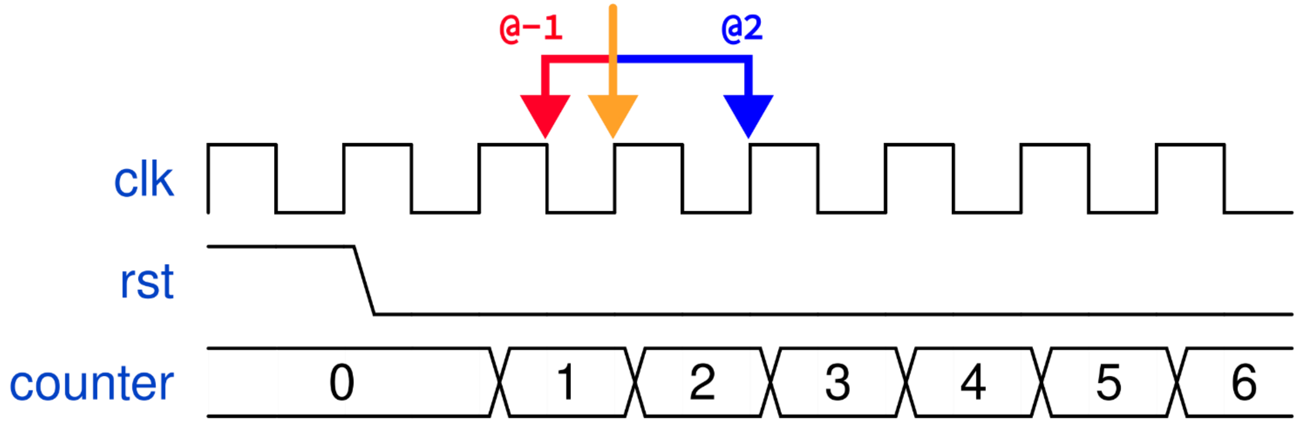
Relative Evaluation

- Index can be locally modified with `expr@offset` syntax
- Evaluate at next timestamp $\Rightarrow$ `signal@1`
- Detect value change $\Rightarrow$ `(!= signal signal@1)`

- @ can be applied to every expression (not just signals)
- Is `x` larger than 5 two indices ahead? $\Rightarrow$ `(> x 5)@2`

Hands-On: Relative Evaluation

```
>-> counter
2
>-> counter@-1
1
>-> counter@2
3
>-> (= counter 4)@2
#f
```



Variables

- Define a new variable using `define`
  - `(define x 5)`
- Change variables using `set!`
  - `(set! [x 22])`
- Create local bindings using `let`
  - `(let ([x 10]) x)`
  - `(let ([x 10] [y 20]) (+ x y))`

Hands-On: Variables

```
>-> (define x 5)
5
>-> x
5
>-> (+ x 1)
6
>-> (set! x “DVCON”)
“DVCON”
>-> x
“DVCON”
>-> (+ x 1)
“DVCON1”
```

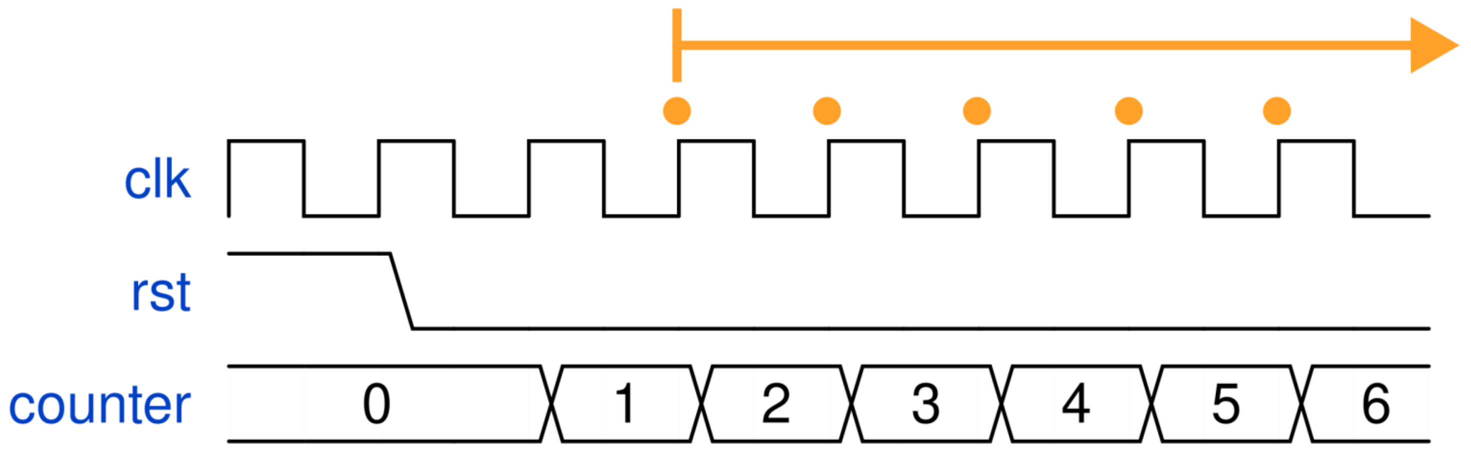
JOHANNES KEPLER
UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

Special Functions

- Signal events
  - `(rising x)` => `(&& (= x 1) (= x@-1 0))`
  - `(falling x)` => `(&& (= x 0) (= x@-1 1))`
  - `(stable x)` => `(= x x@-1)`
- Step over waveform and evaluate body whenever condition is true
  - Starts at the current `INDEX`
  - `(whenever condition body+)`
- Find all indices at which condition is true
  - `(find condition)`
- Count how often condition is true
  - `(count condition)`

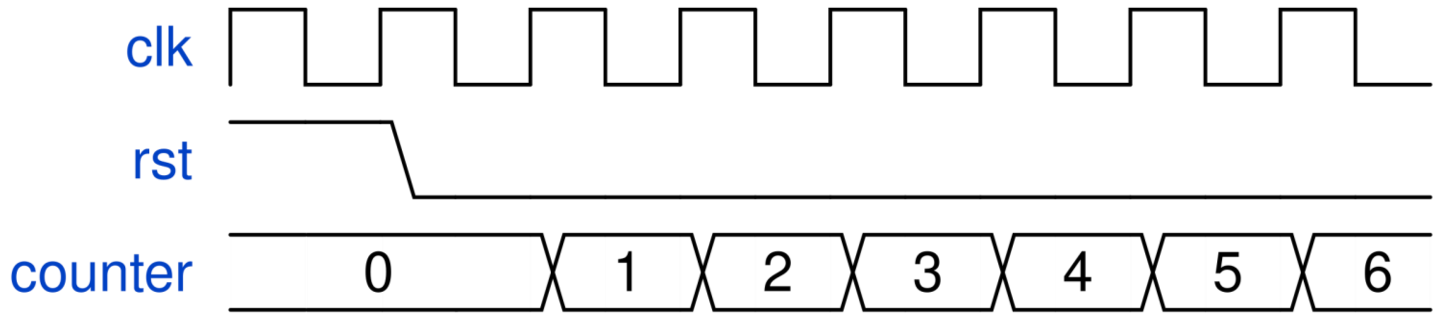
Hands-On: Whenever



```
>-> (whenever clk (print INDEX “ “ counter))
6 2
8 3
10 4
…
```

JOHANNES KEPLER
UNIVERSITY LINZ

Hands-On: Find, Count



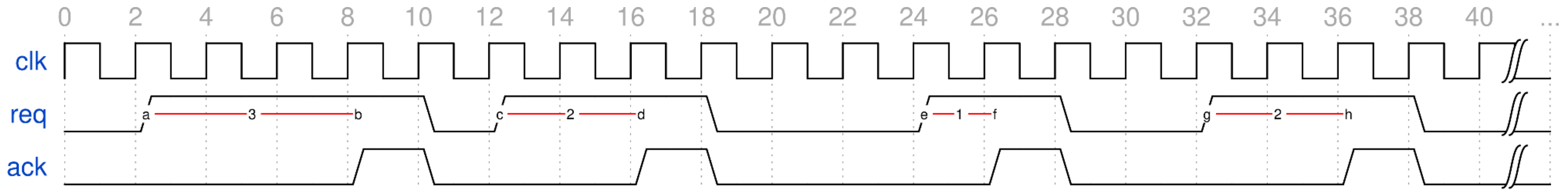
```
>-> (find (= counter 2))
(6 7 38 39 70 71)
>-> (count (= counter 2))
6
```

JOHANNES KEPLER
UNIVERSITY LINZ

Example: Average Delay

- Calculate average delay on handshaking bus
- Two states:
  - Waiting: `(&& req (! ack))`
  - Sending: `(&& req ack)`
- Count states
- Result = |waiting| / |sending|

```
(whenever clk
     … always evaluated when clk = 1 …)
```

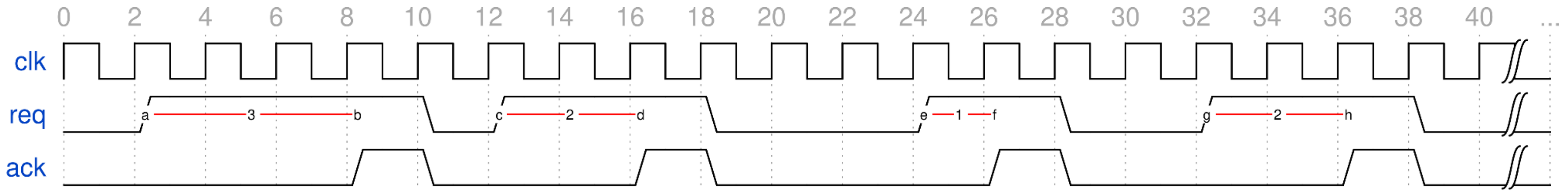


JOHANNES KEPLER UNIVERSITY LINZ

Example: Average Delay (1)

- Calculate average delay on handshaking bus
- Two states:
  - Waiting: `(&& req (! ack))`
  - Sending: `(&& req ack)`
- Count states
- Result = |waiting| / |sending|

```
(whenever (rising clk)
     (when (&& req (! ack)) (inc wait))
     (when (&& req ack) (inc packets)))

(print (/ wait packets))
```



(3+2+1+2) / 4 = 8/4 = 2

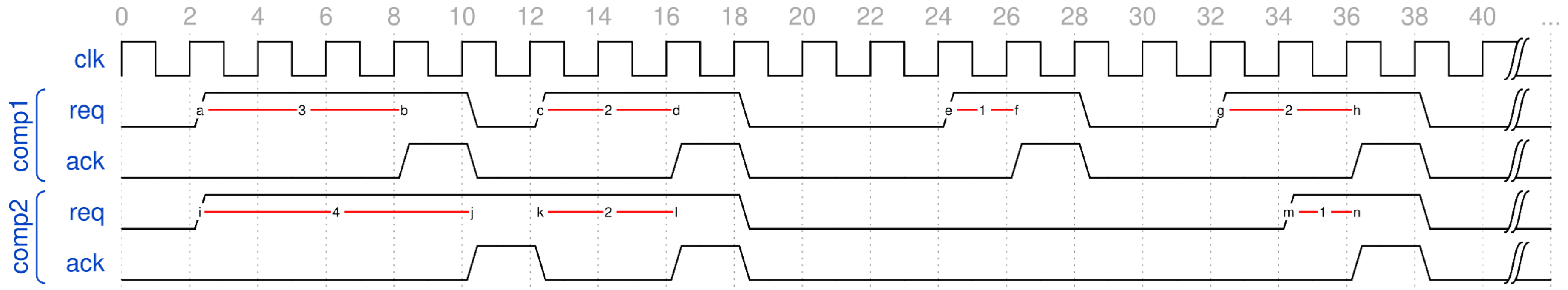
JOHANNES KEPLER UNIVERSITY LINZ

Groups

- HW designs ideal for writing generic code!
  - Handshaking is common
  - Standardized interfaces (AXI, AHB, Wishbone, SPI, …)
- For example, two instances of the handshaking bus
- Write expressions only using the shared suffix of the name
- Expand `#suffix` to full name
  - `#req` => either `comp1.req` or `comp2.req`

- `clk`
- `comp1.req`
- `comp1.ack`
- `comp2.req`
- `comp2.ack`

- `comp1.`
- `comp2.`



Hands-On: Groups

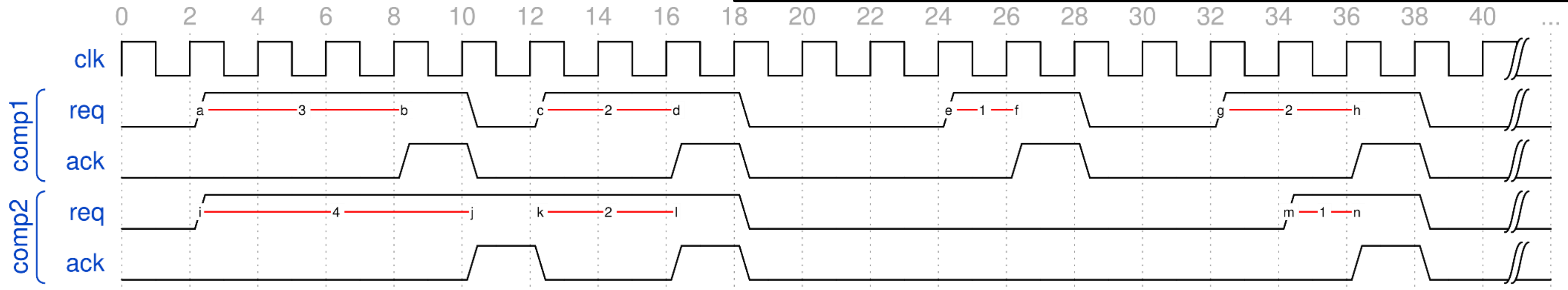
```
>-> SIGNALS
(… "comp1.clk" "comp1.ready" "comp1.valid"
   "comp2.clk" "comp2.ready" "comp2.valid")
>-> (groups clk ready valid)
("comp1." "comp2.")
>-> (groups clk)
("" "comp1." "comp2.")
```

Example: Average Delay (2)

- Wrap analysis in `in-groups` function
- Expression evaluated in each group
- `#signal` expanded to full name

```
(groups req ack) ⇒ (comp1. comp2.)
```

```
(in-groups (groups req ack)
  (whenever (rising clk)
    (when (&& #req (! #ack)) (inc wait))
    (when (&& #req #ack) (inc packets))))

(print (/ wait packets))
```

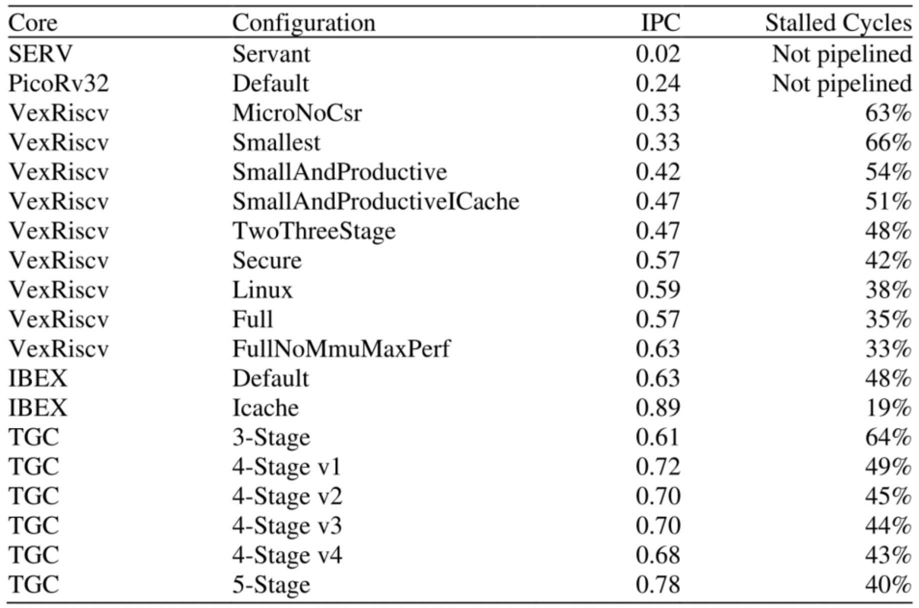


((3+2+1+2) + (4+2+1)) / 7 = (8 + 7) / 7 = 15/7 ≈ 2.1

Other WAL Features

- Data Structures
  - Lists:
    - `(first list)`, `(second list)`, `(rest list)`, …
    - `list[i]`, `list[h:l]`
    - `fold`, `map`, `for` , …
  - Hashmaps:
    - (geta symbol key1 key2 …)
    - (seta symbol key1 key2 … data)
- Extracting bits from signals
  - `signal[i]`, `signal[h:l]`
- WAL as a compilation target from other languages

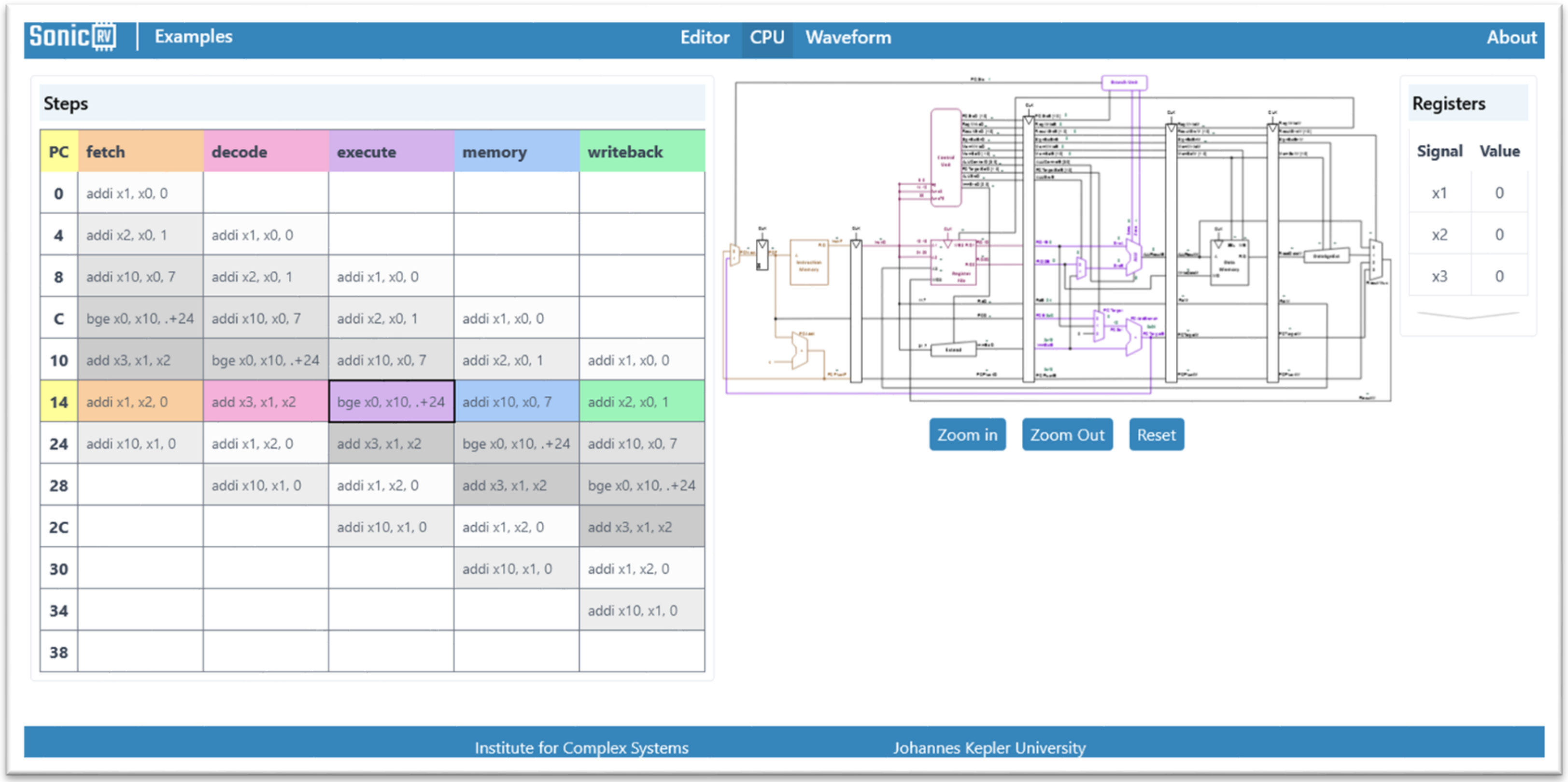
Applications: Reports of Processors, Buses



Core	Configuration	IPC	Stalled Cycles
SERV	Servant	0.02	Not pipelined
PicoRv32	Default	0.24	Not pipelined
VexRiscv	MicroNoCsr	0.33	63%
VexRiscv	Smallest	0.33	66%
VexRiscv	SmallAndProductive	0.42	54%
VexRiscv	SmallAndProductiveICache	0.47	51%
VexRiscv	TwoThreeStage	0.47	48%
VexRiscv	Secure	0.57	42%
VexRiscv	Linux	0.59	38%
VexRiscv	Full	0.57	35%
VexRiscv	FullNoMmuMaxPerf	0.63	33%
IBEX	Default	0.63	48%
IBEX	Icache	0.89	19%
TGC	3-Stage	0.61	64%
TGC	4-Stage v1	0.72	49%
TGC	4-Stage v2	0.70	45%
TGC	4-Stage v3	0.70	44%
TGC	4-Stage v4	0.68	43%
TGC	5-Stage	0.78	40%

```
"tb.dut.mem_wrapper.axi4_source1": {
  transactions: [
    {
      "id": 0,
      "start": 1511,
      "addr": f028,
      "duration": 5234,
      ...
    }
    {
      "id": 1,
      "start": 1541,
      "addr": f032,
      "duration": 3234,
      ...
    }
  ]
}
"tb.dut.mem_wrapper.axi4_source2": {
  transactions: [
    ...
  ]
}
```

Applications: Pipeline Explorer



Steps

PC	fetch	decode	execute	memory	writeback
0	addi x1, x0, 0				
4	addi x2, x0, 1	addi x1, x0, 0			
8	addi x10, x0, 7	addi x2, x0, 1	addi x1, x0, 0		
C	bge x0, x10, .+24	addi x10, x0, 7	addi x2, x0, 1	addi x1, x0, 0	
10	add x3, x1, x2	bge x0, x10, .+24	addi x10, x0, 7	addi x2, x0, 1	addi x1, x0, 0
14	addi x1, x2, 0	add x3, x1, x2	bge x0, x10, .+24	addi x10, x0, 7	addi x2, x0, 1
24	addi x10, x1, 0	addi x1, x2, 0	add x3, x1, x2	bge x0, x10, .+24	addi x10, x0, 7
28		addi x10, x1, 0	addi x1, x2, 0	add x3, x1, x2	bge x0, x10, .+24
2C			addi x10, x1, 0	addi x1, x2, 0	add x3, x1, x2
30				addi x10, x1, 0	addi x1, x2, 0
34					addi x10, x1, 0
38					

Registers

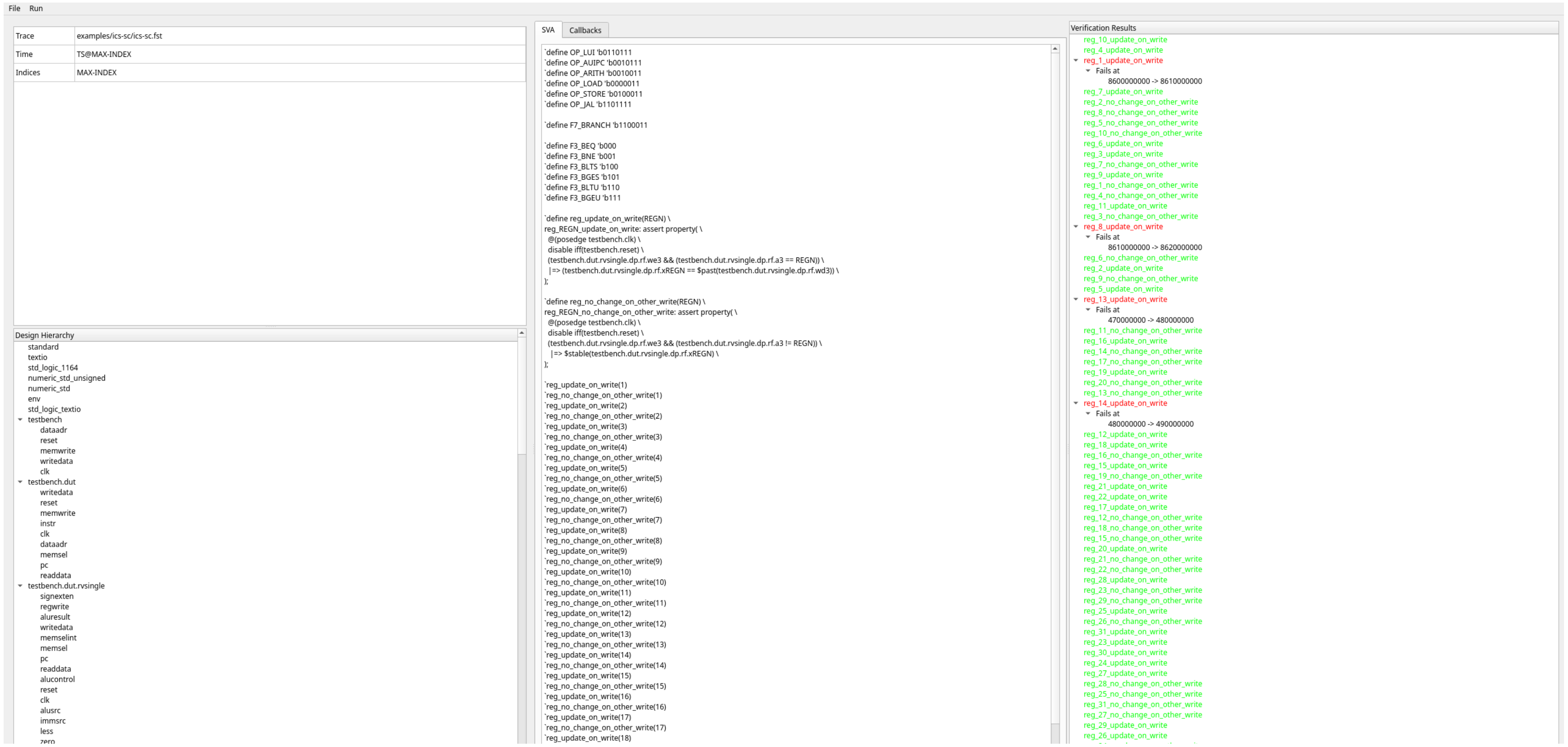
Signal	Value
x1	0
x2	0
x3	0

Institute for Complex Systems Johannes Kepler University

```
(require pipeline)

(stage fetch
       (value tb.dut.dp.instrf@1)
       (stall tb.dut.dp.stallf)
       (log stallf
tb.dut.dp.stallf)
       (log pc tb.dut.dp.pcf))

(stage decode
       (update (!
tb.dut.dp.stalld))
       (stall tb.dut.dp.stalld)
       (flush tb.dut.dp.flushd)

       (log pc fetch-pc@-1)
       (log rd tb.dut.dp.rdd)
       (log rs1 tb.dut.dp.rs1d)
       (log rs2 tb.dut.dp.rs2d))

(stage execute
       (update (!
tb.dut.dp.flushe))
       (flush tb.dut.dp.flushe)
       (log pc decode-pc@-1))

(stage memory)

(stage writeback)
```

JOHANNES KEPLER UNIVERSITY LINZ

Applications: SVA on Waveforms



Trace	examples/ics-sc/ics-sc.fst
Time	TS@MAX-INDEX
Indices	MAX-INDEX

```
standard
textio
std_logic_1164
numeric_std_unsigned
numeric_std
env
std_logic_textio
testbench
    dataadr
    reset
    memwrite
    writedata
    clk
testbench.dut
    writedata
    reset
    memwrite
    instr
    clk
    dataadr
    memsel
    pc
    readdata
testbench.dut.rvsingle
    signexten
    regwrite
    aluresult
    writedata
    memselint
    memsel
    pc
    readdata
    alucontrol
    reset
    clk
    alusrc
    immsrc
    less
    zero
```

```
`define OP_LUI 'b0110111
`define OP_AUIPC 'b0010111
`define OP_ARITH 'b0010011
`define OP_LOAD 'b0000011
`define OP_STORE 'b0100011
`define OP_JAL 'b1101111

`define F7_BRANCH 'b1100011

`define F3_BEQ 'b000
`define F3_BNE 'b001
`define F3_BLTS 'b100
`define F3_BGES 'b101
`define F3_BLTU 'b110
`define F3_BGEU 'b111

`define reg_update_on_write(REGN) \
reg_REGN_update_on_write: assert property( \
 @(posedge testbench.clk) \
 disable iff(testbench.reset) \
 (testbench.dut.rvsingle.dp.rf.we3 && (testbench.dut.rvsingle.dp.rf.a3 == REGN)) \
  |=> (testbench.dut.rvsingle.dp.rf.xREGN == $past(testbench.dut.rvsingle.dp.rf.wd3)) \
);

`define reg_no_change_on_other_write(REGN) \
reg_REGN_no_change_on_other_write: assert property( \
 @(posedge testbench.clk) \
 disable iff(testbench.reset) \
 (testbench.dut.rvsingle.dp.rf.we3 && (testbench.dut.rvsingle.dp.rf.a3 != REGN)) \
  |=> $stable(testbench.dut.rvsingle.dp.rf.xREGN) \
);

`reg_update_on_write(1)
`reg_no_change_on_other_write(1)
`reg_update_on_write(2)
`reg_no_change_on_other_write(2)
`reg_update_on_write(3)
`reg_no_change_on_other_write(3)
`reg_update_on_write(4)
`reg_no_change_on_other_write(4)
`reg_update_on_write(5)
`reg_no_change_on_other_write(5)
`reg_update_on_write(6)
`reg_no_change_on_other_write(6)
`reg_update_on_write(7)
`reg_no_change_on_other_write(7)
`reg_update_on_write(8)
`reg_no_change_on_other_write(8)
`reg_update_on_write(9)
`reg_no_change_on_other_write(9)
`reg_update_on_write(10)
`reg_no_change_on_other_write(10)
`reg_update_on_write(11)
`reg_no_change_on_other_write(11)
`reg_update_on_write(12)
`reg_no_change_on_other_write(12)
`reg_update_on_write(13)
`reg_no_change_on_other_write(13)
`reg_update_on_write(14)
`reg_no_change_on_other_write(14)
`reg_update_on_write(15)
`reg_no_change_on_other_write(15)
`reg_update_on_write(16)
`reg_no_change_on_other_write(16)
`reg_update_on_write(17)
`reg_no_change_on_other_write(17)
`reg_update_on_write(18)
```

```
   reg_10_update_on_write
   reg_4_update_on_write
▾  reg_1_update_on_write
     ▾ Fails at
          8600000000 -> 8610000000
   reg_7_update_on_write
   reg_2_no_change_on_other_write
   reg_8_no_change_on_other_write
   reg_5_no_change_on_other_write
   reg_10_no_change_on_other_write
   reg_6_update_on_write
   reg_3_update_on_write
   reg_7_no_change_on_other_write
   reg_9_update_on_write
   reg_1_no_change_on_other_write
   reg_4_no_change_on_other_write
   reg_11_update_on_write
   reg_3_no_change_on_other_write
▾  reg_8_update_on_write
     ▾ Fails at
          8610000000 -> 8620000000
   reg_6_no_change_on_other_write
   reg_2_update_on_write
   reg_9_no_change_on_other_write
   reg_5_update_on_write
▾  reg_13_update_on_write
     ▾ Fails at
          470000000 -> 480000000
   reg_11_no_change_on_other_write
   reg_16_update_on_write
   reg_14_no_change_on_other_write
   reg_17_no_change_on_other_write
   reg_19_update_on_write
   reg_20_no_change_on_other_write
   reg_13_no_change_on_other_write
▾  reg_14_update_on_write
     ▾ Fails at
          480000000 -> 490000000
   reg_12_update_on_write
   reg_18_update_on_write
   reg_16_no_change_on_other_write
   reg_15_update_on_write
   reg_19_no_change_on_other_write
   reg_21_update_on_write
   reg_22_update_on_write
   reg_17_update_on_write
   reg_12_no_change_on_other_write
   reg_18_no_change_on_other_write
   reg_15_no_change_on_other_write
   reg_20_update_on_write
   reg_21_no_change_on_other_write
   reg_22_no_change_on_other_write
   reg_28_update_on_write
   reg_23_no_change_on_other_write
   reg_29_no_change_on_other_write
   reg_25_update_on_write
   reg_26_no_change_on_other_write
   reg_31_update_on_write
   reg_23_update_on_write
   reg_30_update_on_write
   reg_24_update_on_write
   reg_27_update_on_write
   reg_28_no_change_on_other_write
   reg_25_no_change_on_other_write
   reg_31_no_change_on_other_write
   reg_27_no_change_on_other_write
   reg_29_update_on_write
   reg_26_update_on_write
```

You tried WAL and we listened

- WAL is available now for 3+ years
- We got a lots of feedback since then
- Now we want to make production-ready waveform analysis real
- We are working on a new Product based on your feedback
  - Intuitive C-style syntax
  - Support for much larger waveforms
  - Ready-to-use libraries (AMBA, Ethernet, …)
  - Tight waveform viewer integration
- Initial release early next year
- Interested, ideas, wishes? Let’s talk over a coffee/beer later!

Take-home Message

If you work with waveforms try WAL online or the Python version, reach out, and stay tuned!

JOHANNES KEPLER UNIVERSITY LINZ

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE MUNICH, GERMANY OCTOBER 15-16, 2024

Unleash the Full Potential of Your Waveforms

From Extra-functional Analysis to Functional Debug via Programs on Waveforms

Daniel Große, Lucas Klemmer
Institute for Complex Systems (ICS)
Web: jku.at/ics | wal-lang.org
Email: daniel.grosse@jku.at, lucas.klemmer@jku.at

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

JOHANNES KEPLER UNIVERSITY LINZ

Papers

- Lucas Klemmer and Daniel Große. An Extensible and Flexible Methodology for Analyzing the Cache Performance of Hardware Designs. In *FDL*, 2024. https://ics.jku.at/files/2024FDL_WAL-Cache-Performance-Analysis.pdf
- Lucas Klemmer and Daniel Große. WAVING Goodbye to Manual Waveform Analysis in HDL Design With WAL. In *IEEE Transactions on Computer Aided Design of Circuits and Systems (TCAD)*, 2024. https://ieeexplore.ieee.org/document/10496480 (**open access PDF**).
- Lucas Klemmer and Daniel Große. Towards a highly interactive design-debug-verification cycle. In *ASP-DAC*, 2024. https://ics.jku.at/files/2024ASPDAC_WAL-VirtualSignals.pdf
- Lucas Klemmer, Frans Skarman, Oscar Gustafsson, and Daniel Große, "Surfer: a waveform viewer as dynamic as RISC-V," In *RISC-V Summit Europe*, 2024. https://ics.jku.at/files/2024RISCVSummit_Surfer.pdf
- Daniel Große, Lucas Klemmer, and Dominik Bonora, "Using formal verification methods for optimization of circuits under external constraints," in *DATE*, 2024. https://ics.jku.at/files/2024DATE_FSYN.pdf
- Lucas Klemmer and Daniel Große. Waveform-based performance analysis of RISC-V processors: late breaking results. In *DAC*, pages 1404-1405, 2022. https://ics.jku.at/files/2022DAC_LBR-Waveform-based-Performance-Analyisis-for-RISC-V.pdf
- Lucas Klemmer, Eyck Jentzsch, and Daniel Große. Programmable analysis of RISC-V processor simulations using WAL. In *DVCon Europe*, 2022. https://ics.jku.at/files/2022DVCon_Programmable_Analysis_of_RISC-V_Processor_Simulations_using_WAL.pdf
- Lucas Klemmer and Daniel Große. A DSL for visualizing pipelines: A RISC-V case study. In *RISC-V Summit Europe*, 2023. https://ics.jku.at/files/2023RISCVSummit_DSLforVisualizingPipelines.pdf
- Frans Skarman, Lucas Klemmer, Oscar Gustafsson, and Daniel Große. Enhancing compiler-driven HDL design with automatic waveform analysis. In *FDL*, 2023. https://ics.jku.at/files/2023FDL_Enhancing-Compiler-Driven-HDL-Design-with-WAL.pdf
- Lucas Klemmer and Daniel Große. WAL: a novel waveform analysis language for advanced design understanding and debugging. In *ASP-DAC*, pages 358-364, 2022. https://ics.jku.at/files/2022ASPDAC_WAL.pdf