Oberseminar TAMS

Vergleichende Untersuchungen zur effizienten VHDL-Simulation

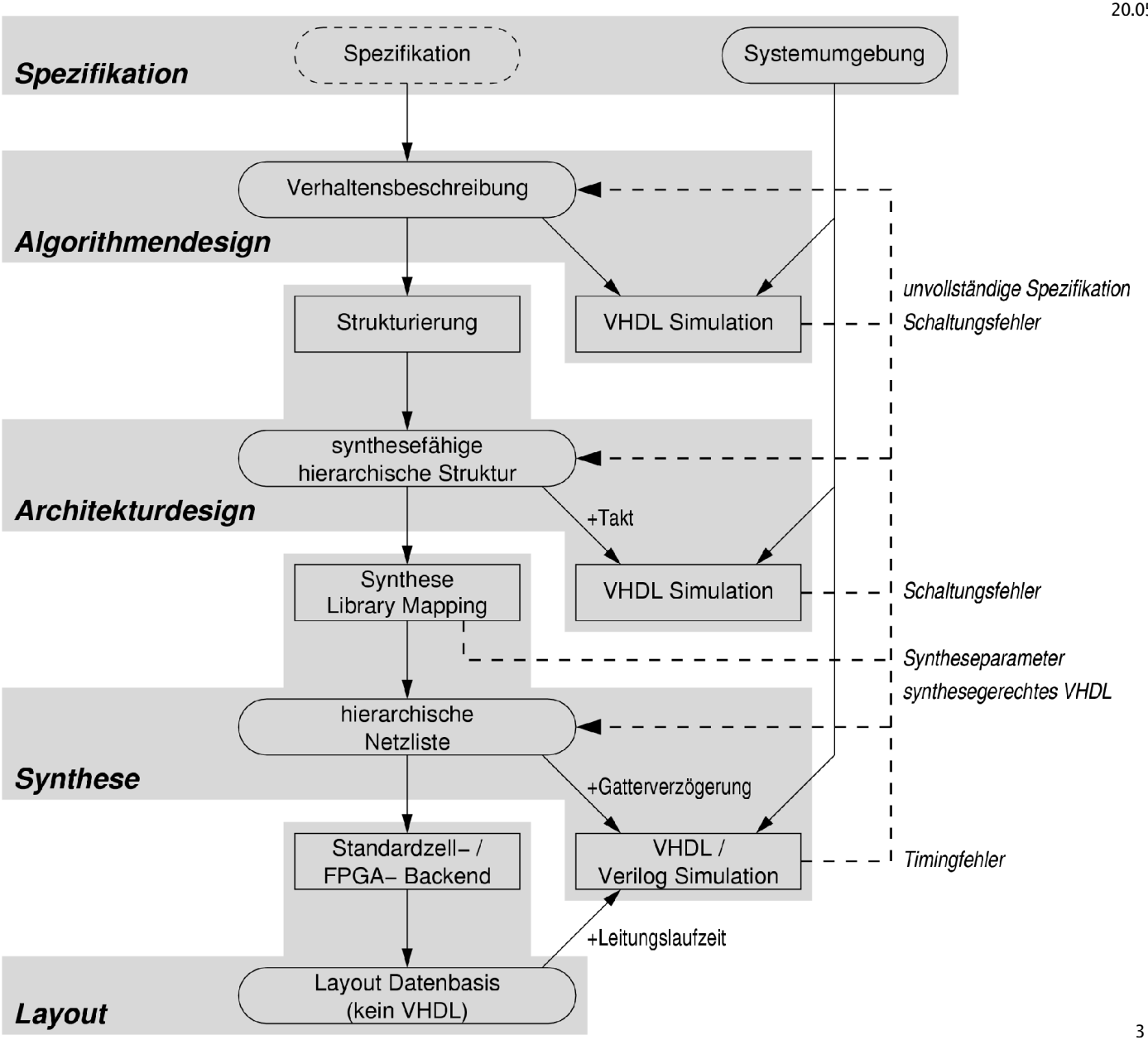
AB TAMS
Technische Aspekte Multimodaler Systeme
Andreas Mäder
maeder@informatik.uni-hamburg.de

Inhalt

- Motivation
  - Entwurfsablauf
  - Ziele der Simulation
- VHDL-Simulation
  - Simulationszyklus
  - Algorithmen
  - Durchführung der Simulation
- Methoden & Werkzeuge
  - Benchmarks
  - Programme
- Ergebnisse
  - Algorithmen
  - Parameter
  - Skalierung
  - Codierung

Andreas Mäder
maeder@informatik.uni-hamburg.de

Top-down Entwurf



Ziele der Simulation

- Detektion von Entwurfsfehlern ⇒ Nachweis der Funktionalität
- Erarbeitung/Ergänzung der Spezifikation
- Untersuchung von Alternativen
- Dynamisches Systemverhalten

  Simulation mit *richtigen* Anwendungsdaten als Stimuli
  - Timing-Analyse
  - Power-Analyse
    Schaltverhalten aus der Simulation statt geschätzter Umschalthäufigkeiten

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ziele der Simulation

- Konsistenzsicherung
  Correctness by Construction, beim Übergang von einer simulierten Eingabe der Synthese zu der Netzliste, gilt nicht!
  - Unzureichende Simulation auf höheren Ebenen
  - Bedienungsfehler
  - Design-Flow z.B.: als Wechselwirkung zwischen Syntheseprogramm und Bibliotheken
  - Programmfehler

Andreas Mäder
maeder@informatik.uni-hamburg.de

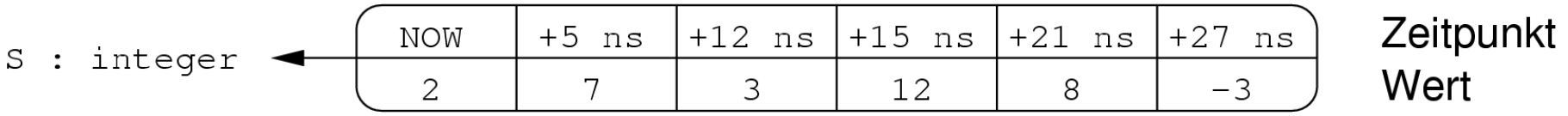
VHDL Simulation

- Menge konkurrent aktiver Codefragmente
  - Prozesse
    begrenzen sequenziell abzuarbeitenden VHDL-Code
  - Konkurrente Anweisungen
    Prozesse mit nur einer Anweisung
  - Hierarchien
    Ersetzung der Instanzen führt zu deren konkurrentem Code
- Signale verbinden diese Codeteile (Prozesse)
  - Signaltreiber
  - Signalaktivität

Andreas Mäder
maeder@informatik.uni-hamburg.de

Signaltreiber

- Zuweisung an ein Signal durch eine konkurrente Zuweisung / einen Prozess
- Mehrere Zuweisungen innerhalb *eines* Prozesses bilden einen Signaltreiber
- Sortierte Liste: Wert-Zeit Paare

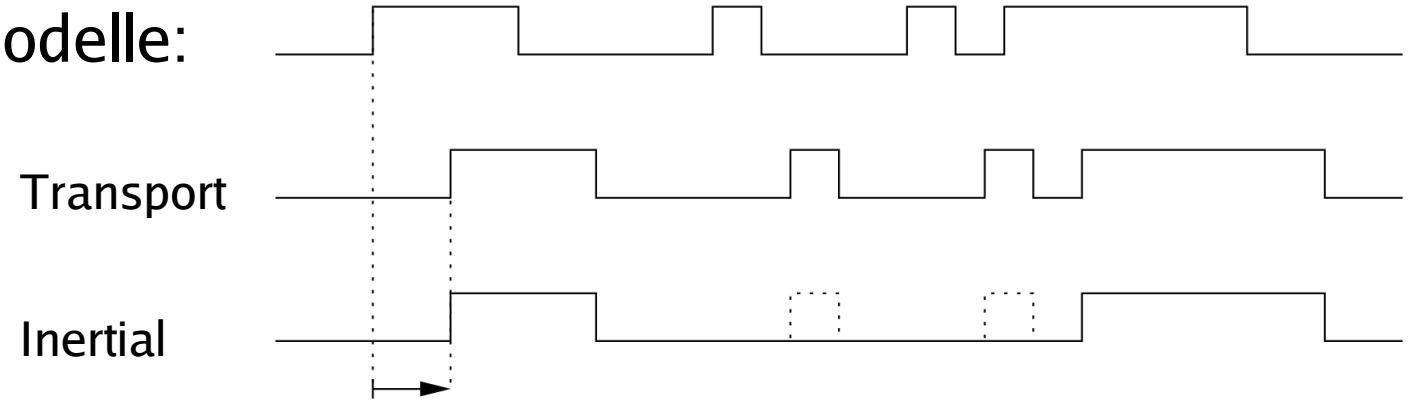


- Aktualisierung durch den Simulationslauf als Ergebnis von Signalzuweisungen

Andreas Mäder
maeder@informatik.uni-hamburg.de

Signaltreiber

- Verzögerungsmodelle:



Transport
```
S <= transport I after 2 ns;
S <= reject 0 ns inertial I after 2 ns;
```
Inertial
```
S <= I after 2 ns;
S <= inertial I after 2 ns;
S <= reject 2 ns inertial I after 2 ns;
```

- Delta-Delay: Zuweisung ohne Signalverzögerung

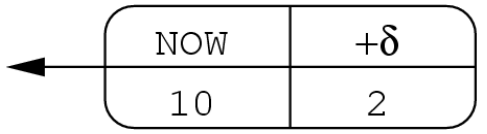
```
S <= I;
```


Andreas Mäder
maeder@informatik.uni-hamburg.de

Signaltreiber

1. Ein Treiber, eine Signalzuweisung
   - Aktualisierung der Treiberliste für Zeiten > Verzögerungszeit
   - Ersetzung bei δ-Delay

```
S <= 2;
```



2. Ein Treiber, mehrere Zuweisungen (in einem Prozess)
   - sequenzielle Aktualisierung der Treiberliste
3. Mehrere Treiber ⇒ Busstruktur
   - Erfordert spezielle Datentypen und Auflösungsfunktion, die aus allen Treibern einen *effektiven Wert* des Signals berechnet

Signalaktivität

- Der simulierte Zeitpukt `now` ändert sich
- Eine Signalzuweisung der Treiberliste wird wirksam
- Bei Signalauflösung (mehrere Treiber) wird ein neuer effektiver Wert berechnet
- VHDL-Attribute:

`'active`	Signalaktivität
`'event`	+ der Wert ändert sich

Andreas Mäder
maeder@informatik.uni-hamburg.de

VHDL Simulation

- Ablauf der Simulation durch Ereignisse gesteuert
- Simulationsereignis:
  - Werteänderung eines Signals

    $$Ereignis_{Signal} = (Zeitpunkt, Signal, Wert)$$

  - (Re-) Aktivierung eines Prozesses nach `wait for`

    $$Ereignis_{Prozess} = (Zeitpunkt, Prozess, Einsprungstelle)$$

- Prozessaktivierung:
  - Direkt $Ereignis_{Prozess}$
  - Triggerfunktion sensitivity-Liste, `wait on`, `wait until`

    $$Trigger_{Prozess}(Ereignis_{Signal}) \rightarrow Ereignis_{Prozess}$$

Andreas Mäder
maeder@informatik.uni-hamburg.de

VHDL Simulationszyklus

- Initialisierung
  - $t_0.zeit$ — t_c = `now := 0 ns;`
  - $t_0.signal_{explizit}$ — effektive Werte explizit deklarierter Signale
  - $t_0.signal_{implizit}$ — implizite Signale (durch Attribute) initialisieren
  - $t_0.prozess_{normal}$ — alle Prozesse bis `wait` / Prozessende, ausführen
  - $t_0.prozess_{postponed}$ — postponed Prozesse (letzter δ-Zyklus) abarbeiten
  - $t_0.zeitschritt$ — nächster Zeitschritt niedrigster Wert des Schedule

$$t_n = min(\texttt{time'high}, Ereignis_{Signal}.Zeitpunkt, Ereignis_{Prozess}.Zeitpunkt)$$

Andreas Mäder
maeder@informatik.uni-hamburg.de

VHDL Simulationszyklus

- Zeitschritt: effektiv oder δ-Zyklus
  - $t_c.zeit$ t_c = now := t_n;
    `time'high` markiert das Ende der Simulation
  - $t_c.signal_{explizit}$ die Treiber aktiver Signale werden aktualisiert
  - $t_c.signal_{implizit}$ und erzeugen: $Ereignis_{Signal}$
  - $t_c.prozess_{aktivierung}$ alle zu aktivierenden Prozesse werden bestimmt:
    $Ereignis_{Prozess} \cup Trigger_{Prozess}(Ereignis_{Signal})$
    `postponed`-Prozesse bilden eine eigene Menge
  - $t_c.prozess_{normal}$ aktivierte Prozesse (nicht postponed)
    bis `wait`, bzw. Prozessende, ausführen

Andreas Mäder
maeder@informatik.uni-hamburg.de

VHDL Simulationszyklus

- $t_c.zeitschritt$ nächster Zeitschritt niedrigster Wert des Schedule
  $t_n = min(\texttt{time'high}, Ereignis_{Signal}.Zeitpunkt, Ereignis_{Prozess}.Zeitpunkt)$
- $t_c = t_n \Rightarrow$ es folgt der nächste δ-Zyklus: $t_c.zeit$
  $t_c \neq t_n \Rightarrow$ dies ist der letzte δ-Zyklus: $t_c.prozess_{postponed}$
- $t_c.prozess_{postponed}$ die postponed Prozesse abarbeiten,
  dabei werden keine weiteren δ-Zyklen erzeugt
- $t_c.zeitschritt_{post}$ nächster Zeitschritt niedrigster Wert des Schedule
  $t_n = min(\texttt{time'high}, Ereignis_{Signal}.Zeitpunkt, Ereignis_{Prozess}.Zeitpunkt)$

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulationsalgorithmen

- Ereignisorientiert, interpretierend
  - Folgt direkt dem Algorithmus
  - Dynamische Verwaltung der Ereignisliste(n) (Signaltreiber)
  - Vorteile: Konformität
    Debugging
  - Nachteile: langsam

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulationsalgorithmen

- Ereignisorientiert, compilierend
  - Umsetzung von VHDL in Hochsprachencode (C, C++)
  - Simulatorkern Verwaltung der Ereignisliste
    Schnittstellen zu Datenstrukturen Signalwerte, Aktivierung...
  - Compiler erzeugt Maschinencode
  - Der Simulator wird *zusammengelinkt*
  - Vorteile: Simulationsgeschwindigkeit
  - Nachteile: Debugging teilweise eingeschränkt
    Compileroptimierung ↔ Geschwindigkeit

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulationsalgorithmen

- Zyklenbasiert, compilierend
  - Simulationszyklus statt dynamischer Verwaltung von Ereignissen:
    - diskretes Zeitraster — Takt bei Register-Transfer Code
    - In jedem Zyklus werden alle Beschreibungen simuliert
      Optimierungen möglich
    - Erfordert Sequenzialisierung der konkurrenten Prozesse, dem Datenfluss entsprechend
  - Vorteile: Simulationsgeschwindigkeit
    Codeabhängig: unnötige Berechnung ↔ dynamische Ereignisverwaltung

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulationsalgorithmen

- Nachteile: Debugmöglichkeiten teilweise eingeschränkt
  Einschränkungen im Sprachstandard:
    Zuweisungen mit Verzögerungszeit
    `wait`-Anweisungen
  Sonderbehandlung von Testumgebungen
  einheitliches Taktschema (disjunkte Takte?)

Andreas Mäder
maeder@informatik.uni-hamburg.de

Durchführung der Simulation

1. Codeanalyse
   - syntaktische Analyse
   - erzeugt Elemente in einer VHDL-Bibliothek
   - simulatoreigene Binärformate
     native Objektdateien (compilierend)
2. Elaboration
   - Hierarchie auflösen
   - Executable des Simulators zusammenbauen (compilierend)
   - Datenstrukturen aufbauen (in Verbindung mit Simulation)
3. Simulation

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulation – weitergehende Konzepte

- Mehrere Algorithmen — nur VHDL
  - Kopplung ereignisorientierter und zyklenbasierter Simulation
  - Teile der Hierarchie erhalten *ihren* Simulatorkern
- Mixed-mode Simulation — + analog = VHDL-AMS
  - analoge Komponenten
    Systemumgebung
    MEMS
  - Schnittstellenproblematik

Wert	diskret ↔ kontinuierlich
Zeit	diskret / (Takt-) Zyklen
Wirkung	*Richtung* der Ports

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulation – weitergehende Konzepte

- Simulatorkopplungen + digital (Verilog)
  - Gatterbibliotheken
    IP-Modelle die nur in einer HDL vorliegen
  - Timing-Checks in Verilog enthalten ⇒ besser optimiert
    (VITAL-Standard in VHDL)
  - Realisierungskonzepte Master-Slave
    getrennte Codeanalyse bei compilierenden Simulatoren
  - „golden Simulation“ VHDL-Systemumgebung + Verilog-Netzliste + SDF-Timing

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulation – weitergehende Konzepte

- Hardwarebeschleuniger FPGA-Boards
  - Hardwareemulation von Gatternetzlisten (Quickturn, IKOS ...)
    z.B. Nvidia: 160Mio.$ tools, 40Mio.$ Emulatoren, >8000CPUs
  - Vorteile: Geschwindigkeit
    macht viele Systemsimulationen erst möglich
  - Nachteile: Kosten
    eingeschränktes Timing
- Programmiersprachen-Schnittstellen (VHPI, Verilog-PLI)
  - Externe Modelle IP-Komponenten, System
  - Benutzerschnittstellen Ein-/Ausgabe, Sound, Grafik...

Andreas Mäder
maeder@informatik.uni-hamburg.de

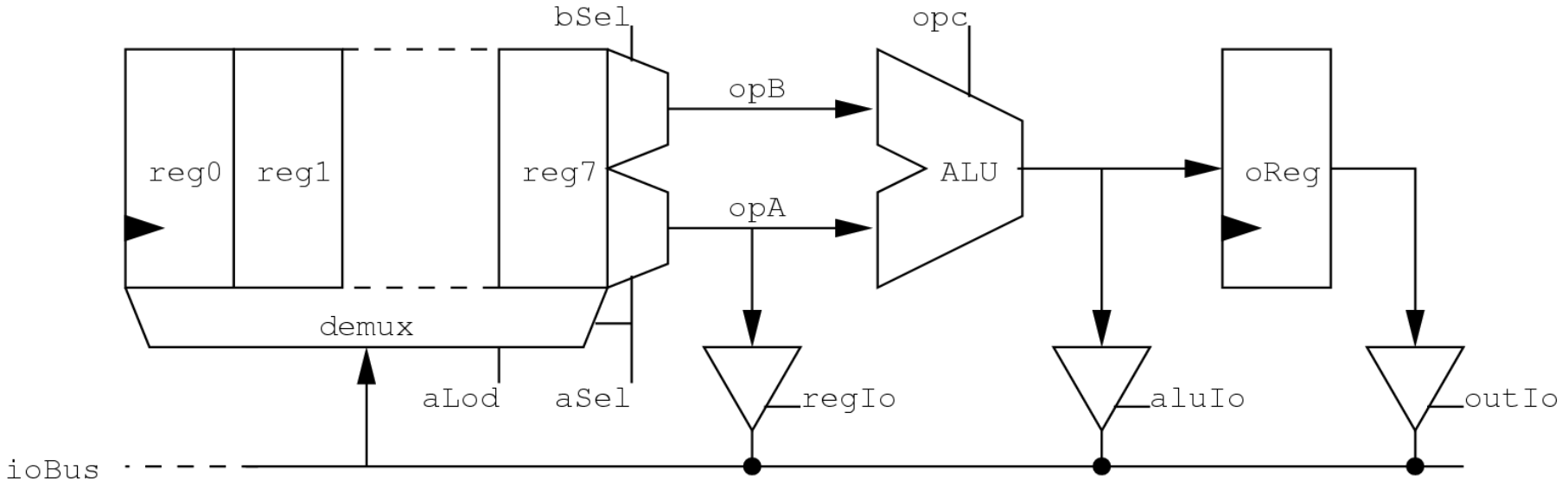
Ziel der Arbeit

- Zeitbedarf von Simulationen ist ein entscheidender Faktor für Entwurfsdauer
- Welche Faktoren beeinflussen die Simulationsgeschwindigkeit?
  - Der Simulator, bzw. dessen Algorithmus
  - Die Parametrisierung und Handhabung der Programme
  - Die Gatterbibliotheken
  - Die Art der VHDL-Codierung
  - Die Schaltungsgröße
- Für VHDL-Simulation gibt es keine Benchmarks!

Andreas Mäder
maeder@informatik.uni-hamburg.de

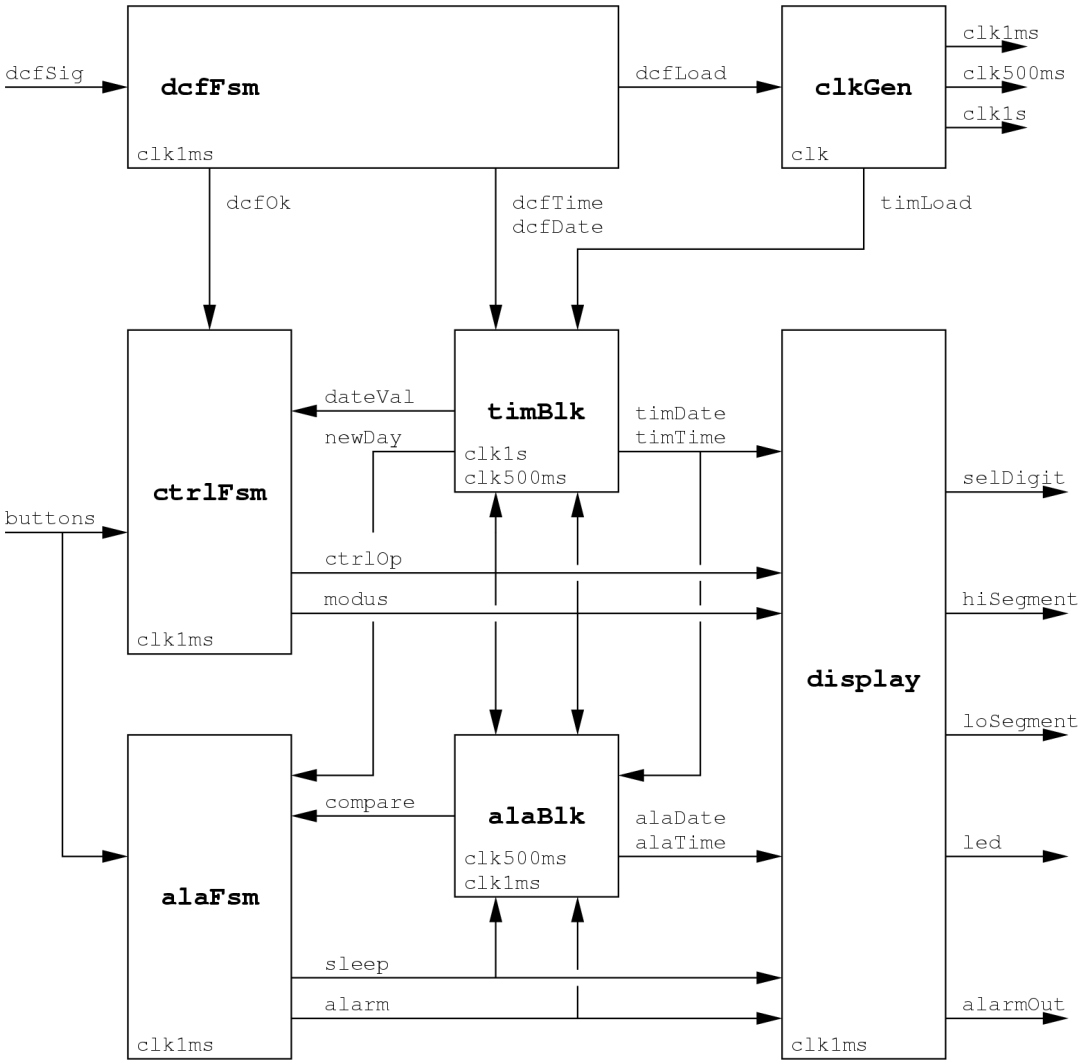
Benchmarks

- Operationswerk
  - 16-bit Prozessor
  - RISC-Befehlssatz: Laden, Speichern, 2- und 3-Adress ALU



Benchmarks

- Funk-Wecker
  - Kleines ASIC
  - Interner Taktteiler
  - Autonome Automaten



Andreas Mäder
maeder@informatik.uni-hamburg.de

Benchmarks

- Hardware-Multiplizierer
  - Radix-4 Booth-Multiplizierer
  - „von Hand“ codiert: Booth-Encoding
    Wallace-Tree CSA
    CPA-Addition
    statt Operator: *
  - Parametrisierbar: $m \times n$-bit
    Beispiele ⇨
    8 × 8, 16, 24, 32
    16 × 16, 24, 32
    24 × 24, 32
    32 × 32
- Ampelschaltung

Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulationsprogramme

- Programme
  - EDA-Anbieter
  - HDLs
  - Platformen
- Zellbibliotheken
  - Prozesse
  - Modelle

	Version	VHDL	Verilog	Sun	PC
VSS	2000.12	*		*	
	2001.09	*		*	
Cyclone	2000.12	*		*	
	2001.09	*		*	
Scirocco	2000.12	*	+VCS	*	
	2001.10	*		*	
	2002.06	*	+VCS	*	
VCS	6.0.1	+Scirocco	*	*	
	6.2R4	+Scirocco	*	*	
Leapfrog	2.84-p001	*	+Verilog-XL	*	
NCSim	3.11.p1	*	*	*	
Verilog-XL	3.11.p1	+Leapfrog	*	*	
ModelSim	5.3d	*	*		*
Simili	1.5b17	*			*

AB TAMS
Technische Aspekte Multimodaler Systeme
Andreas Mäder
maeder@informatik.uni-hamburg.de

Simulationsläufe

- VHDL-Testumgebung
  - keine Benutzereingaben nötig
  - Prüfsummen zeigen korrektes Verhalten der Schaltung
- 62 Parametersätze für die Simulatoren
  - Debug-Simulation, Voreinstellung, Performance
  - Codeanalyse, Elaboration, Simulation
- 10 Parametersätze für die 2 Zellbibliotheken
- Skriptgesteuert
  - Zeitmessung durch Betriebssystem, z.T. Simulatorintern
  - Mittel aus mehreren Läufen

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ergebnisse – allgemein

- Algorithmus
  - ereignisorientiert, interpretierend — OPW
  - ereignisorientiert, compilierend — Clock
  - Zyklenbasiert, compilierend
- Parametervariationen
- Benutzerschnittstelle — VSS
- Abstraktion — NCSim
  - Register-Transfer Code — OPW
  - Gatternetzliste — Clock

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ergebnisse – Skalierung

- Simulatoren skalieren linear mit der Schaltungsgröße
  - Profiling Information aus Elaboration
  - Beispiel: Gatternetzliste des Multiplizierers
- Maßeinheit der Komplexität: *Anzahl skalarer Signale*
  - Unabhängig von der Abstraktion
    - Netzlisten ≙ Anzahl Prozesse
    - Verhalten ≙ Aktivität der Prozesse
  - Maß gilt für alle Simulationsalgorithmen
    - Ereignissteuerung Eingangssensitivität + Wirkungseffektivität
    - Zyklenbasiert Anzahl der Prozesse + Komplexität

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ergebnisse – Codierung

- Simulation: möglichst wenig, möglichst große Prozesse
  kurze Sensitivity-Liste
  - Ereignissteuerung ⇒ weniger Events
  - Zyklenbasiert ⇒ mehr Compileroptimierung
    ⇒ weniger Prozesse anzuordnen
- Design + Synthese: Trennung funktionaler Einheiten, beispielsweise
  ein Prozess pro Register
  einfache Prozesse, wenig Variablen:
  übersichtlicherer Code, weniger Fehler
- Tradeoff: Simulation ↔ Design

Ampelschaltung

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ergebnisse – Netzlistensimulation

- Simulationsmodelle — OPW, Clock
  - Verschiedene Zellmodelle ⇒ VITAL als „Standard“
  - Parameter bei Bibliotheksgenerierung
- Zielbibliothek — OPW, Clock
  - Timing-Modellierung
- Mixed-mode Simulation — OPW, Clock
  - VHDL + Verilog
- Verilog Simulation

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ergebnisse – Programm

- Parametrisierung der Programme — Multiplizierer
- Auswahl des Simulators
  - Starker Konkurrenzkampf – Produkte gleichwertig
  - Unterschiede durch die Konzepte
  - Neue Versionen — OPW
- EDA-Tools auf PCs sind konkurrenzfähig — OPW
  - Sun: Ultra60 2 × 360 MHz 1280MB — Clock
  - PC: AMD K6III 400 MHz 160MB

Andreas Mäder
maeder@informatik.uni-hamburg.de

Ergebnisse – Fehler

- Timing-Fehler ⇒ undefinierte Werte
- Bibliotheksmodelle ⇒ undefinierte Werte
  ⇒ falsche Werte **kritisch!**
- Programmfehler ⇒ Generic-Auswertung
  ⇒ Programmabbrüche
- Verschiedene Simulationsalgorithmen
- Verschiedene Hardwarebeschreibungssprachen

Andreas Mäder
maeder@informatik.uni-hamburg.de

Zusammenfassung

- Simulation ist von zentraler Bedeutung im VLSI-Entwurf und macht einen erheblichen Teil der Entwurfszeit aus
- Zeitaufwand abhängig von
  - Algorithmus: Compiliert
    ereignisorientiert ↔ zyklenbasiert
  - Benutzung: Parameter
    Debug-Option
  - VHDL-Codierung: Partitionierung im Code
    Testumgebung
    Schaltungsgröße
- Gemischte Ansätze VHDL-Testumgebung + Verilog-Netzliste

AB TAMS
Technische Aspekte Multimodaler Systeme
Andreas Mäder
maeder@informatik.uni-hamburg.de

```
mainP: process (clk, rst)
    type stateTy is (Gr, Yr, Rr, Rg, RYr);
    variable   timer      : integer range 0 to maxWalkC;
    variable   state      : stateTy;
    variable   request    : boolean;
begin
  if rst = '0' then                                        -- async. reset
    lightCar  <= "001";
    lightWalk <= "10";
    state     := Gr;
    timer     := 0;
    request   := false;
  elsif rising_edge(clk) then                              -- clock
    case state is
    when Gr =>                                             -- Green + red
      lightCar  <= "001";
      lightWalk <= "10";
      if (reqWalk = '1')  then  request := true;           -- store reqest
      end if;
      if (timer > 0)      then  timer   := timer - 1;      -- no timeout
      elsif request       then  state   := Yr;             -- timeout and req.
      end if;
    when Yr =>                                             -- Yellow + red
      lightCar  <= "010";
      lightWalk <= "10";
      timer     := maxWalkC-1;                             -- init. timer
      ...
```

Selektive Signalzuweisung

```
with OP select
  RES <= A + B    when ADD_I,
         A - B    when SUB_I,
         A + 1    when INC_I,
         ...
         A or  B  when  OR_I,
         A and B  when AND_I;
```

```
process (A, B, OP)
begin
  case OP is
    ADD_I =>    RES <= A + B;
    SUB_I =>    RES <= A - B;
    INC_I =>    RES <= A + 1;
    ...
     OR_I =>    RES <= A or  B;
    AND_I =>    RES <= A and B;
  end case;
end process;
```

Bedingte Signalzuweisung

```
SIGN <=  1 when (A > B) else
         0 when (A = B) else
        -1;
```

```
process (A, B)
begin
  if    (A > B)   then  SIGN <=  1;
  elsif (A = B)   then  SIGN <=  0;
  else                  SIGN <= -1;
  end if;
end process;
```

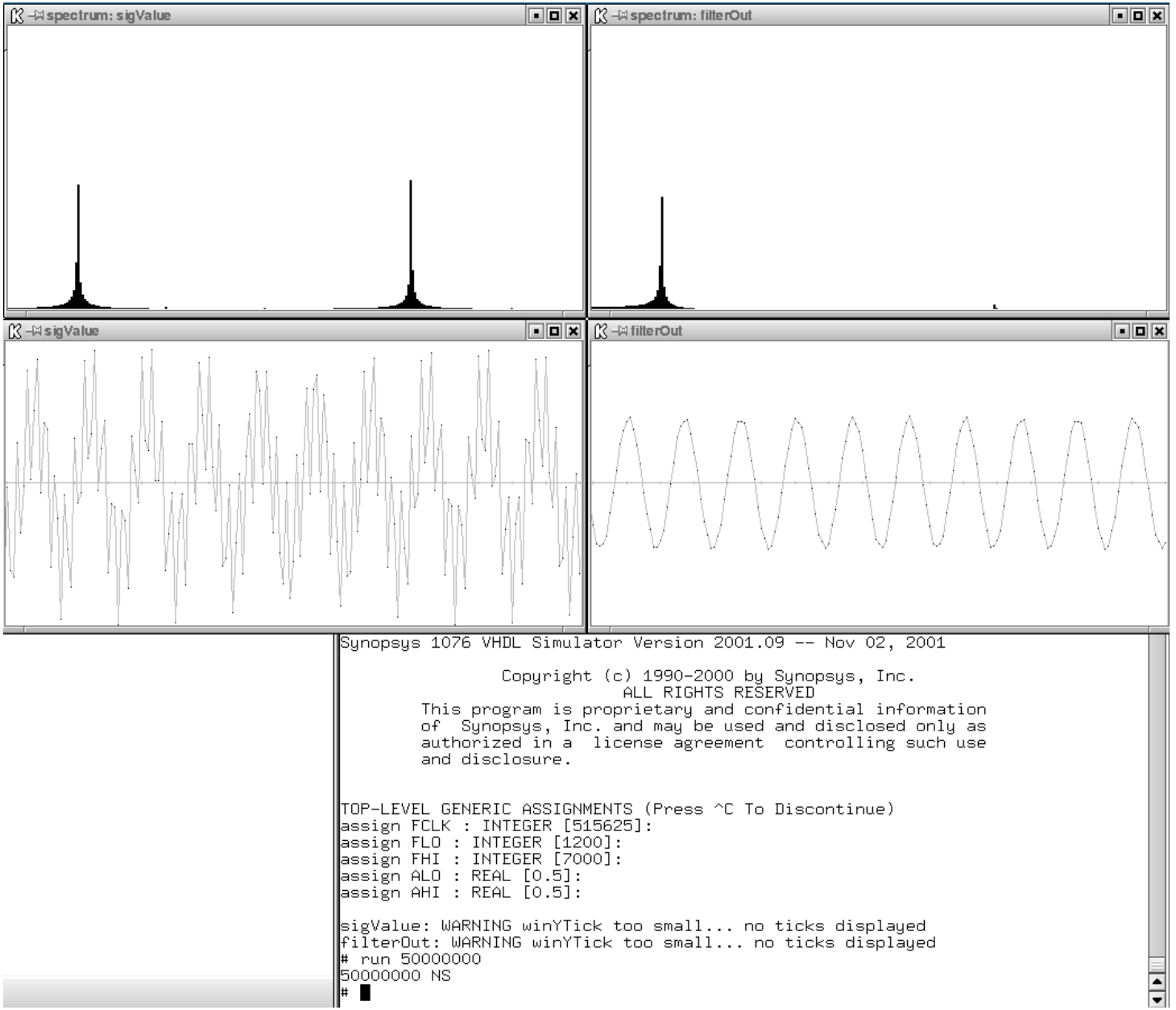
```
-- stimuli generator for user interface
  stiP: process
    procedure pushBut(  pushT, runT    : in  time;
                      signal button : out std_logic) is
    begin
      button <= '1';    wait for pushT;
      button <= '0';    wait for runT-pushT;
    end procedure pushBut;
  begin
    -- send pattern:  Mo. 24.12.01 22:54-- FSM          sec
    wait for 1 sec;                        --             1

    -- set alarm2:    Mo. 22:54
    pushBut(300 ms, 1 sec, butMod);     -- alarm1         2
    pushBut(300 ms, 1 sec, butMod);     -- alarm2         3
    pushBut(300 ms, 1 sec, butMin);     -- minute         4
    pushBut( 3 sec, 4 sec, butMinus);   -- minute -6      8
    pushBut(300 ms, 1 sec, butHour);    -- hour           9
    pushBut( 1 sec, 2 sec, butMinus);   -- hour   -2     11
    ...

    pushBut( 1 sec, 2 sec, butSleep);   -- sleep         64
    wait;
  end process stiP;
```

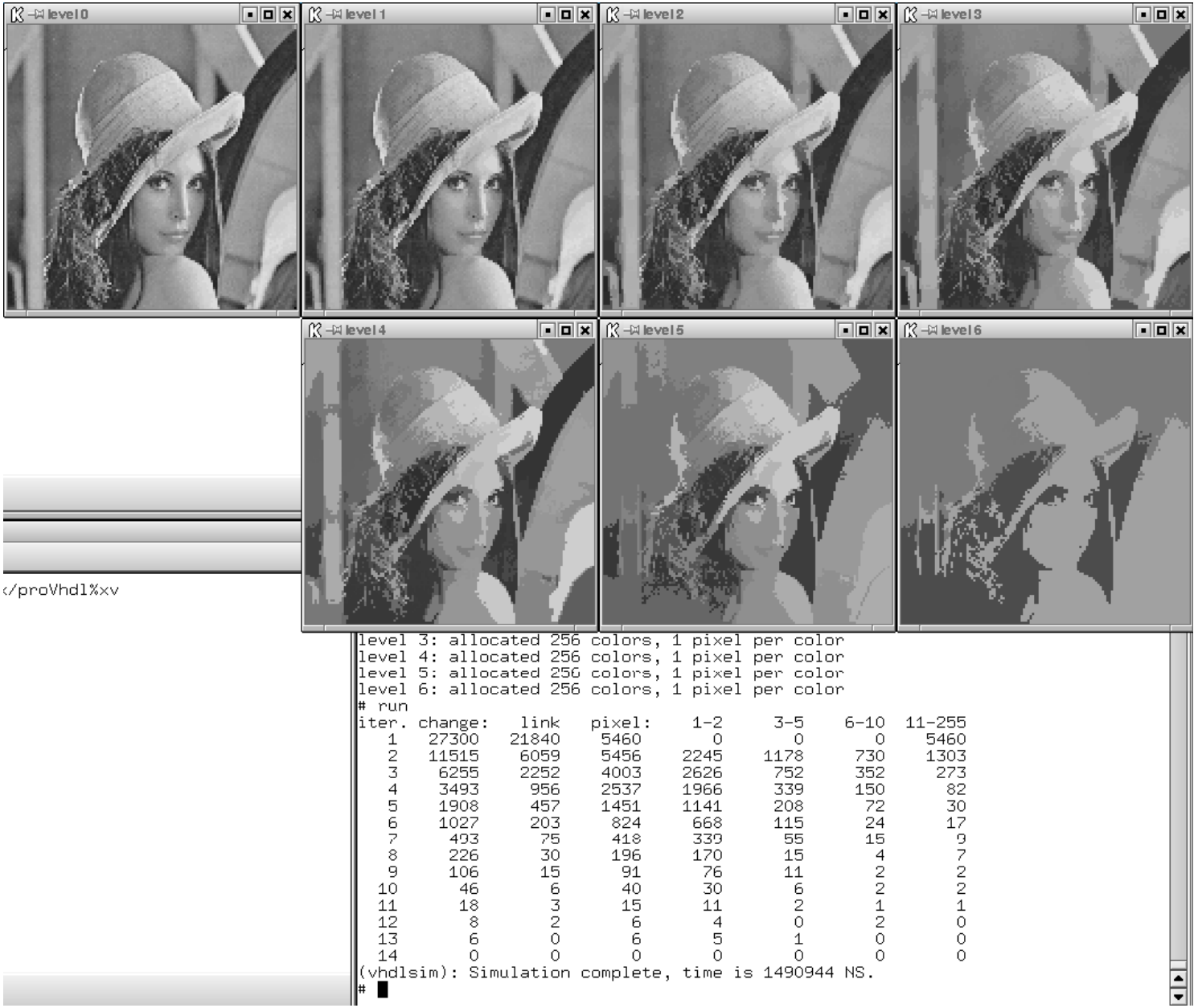

Andreas Mäder
maeder@informatik.uni-hamburg.de

VSS
+Spektrum
+Scope



AB TAMS
Technische Aspekte Multimodaler Systeme
Andreas Mäder
maeder@informatik.uni-hamburg.de

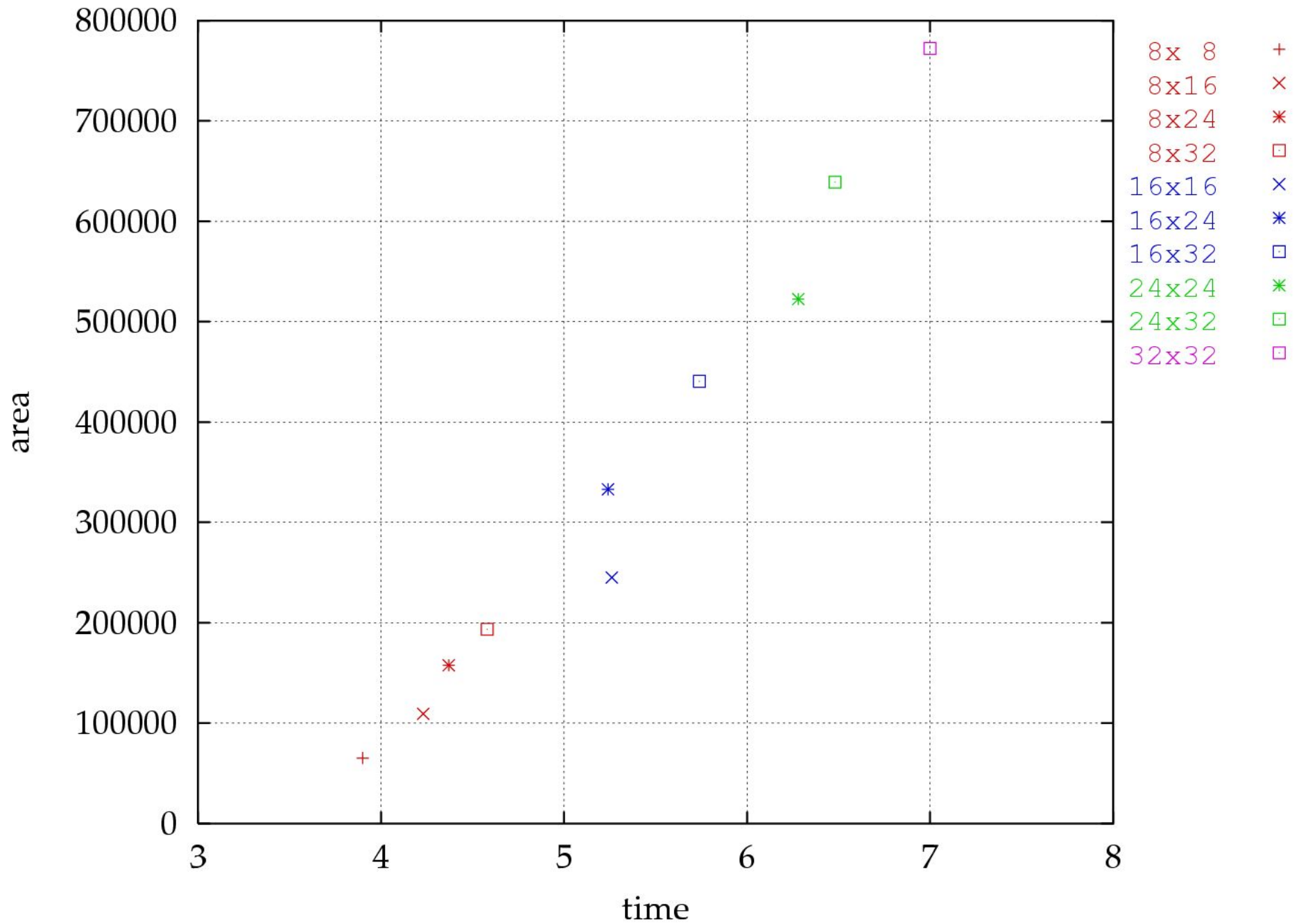
VSS +Display



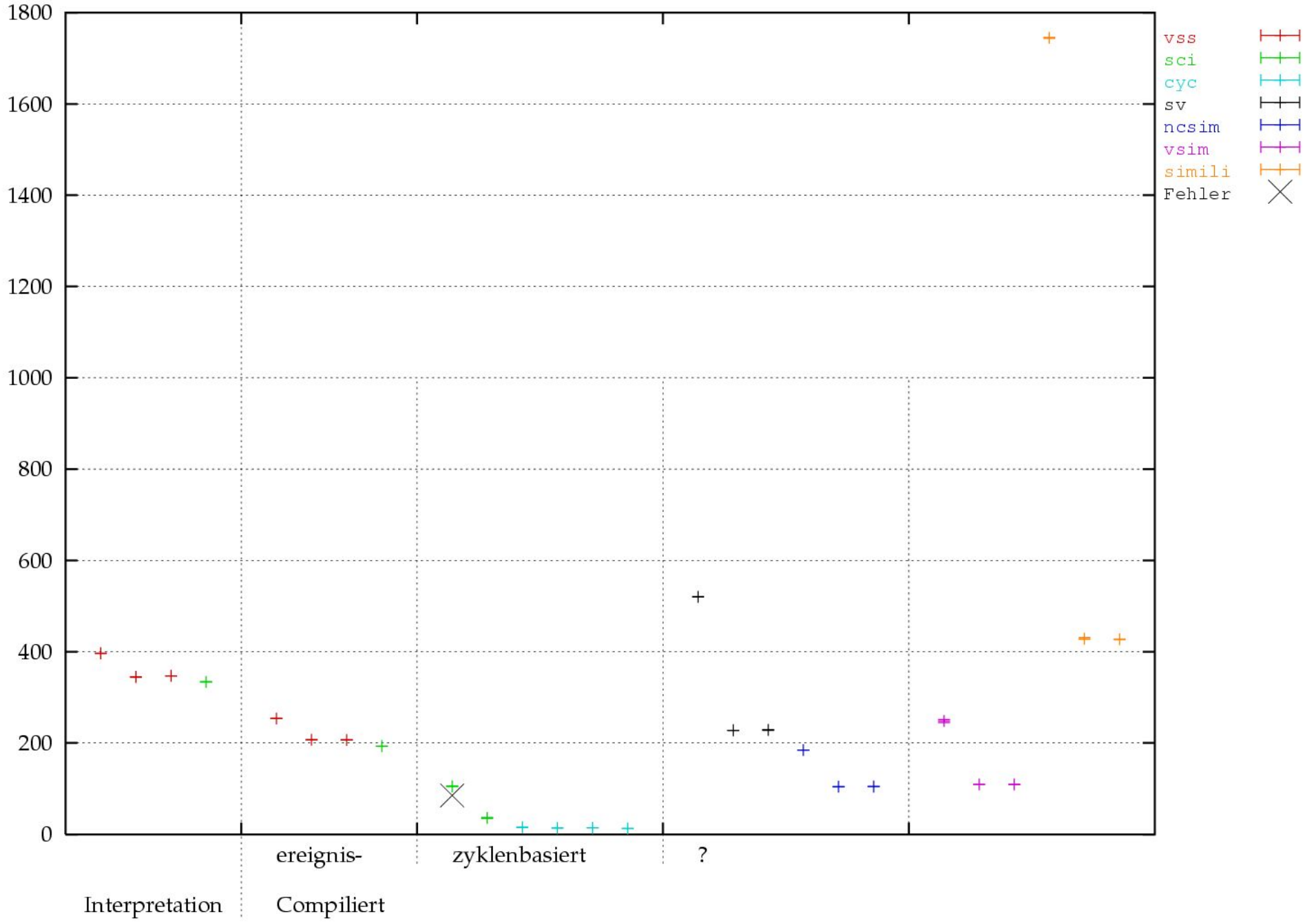

```
</proVhdl%xv
level 3: allocated 256 colors, 1 pixel per color
level 4: allocated 256 colors, 1 pixel per color
level 5: allocated 256 colors, 1 pixel per color
level 6: allocated 256 colors, 1 pixel per color
# run
iter. change:   link   pixel:    1-2     3-5    6-10  11-255
   1   27300   21840    5460       0       0       0    5460
   2   11515    6059    5456    2245    1178     730    1303
   3    6255    2252    4003    2626     752     352     273
   4    3493     956    2537    1966     339     150      82
   5    1908     457    1451    1141     208      72      30
   6    1027     203     824     668     115      24      17
   7     493      75     418     339      55      15       9
   8     226      30     196     170      15       4       7
   9     106      15      91      76      11       2       2
  10      46       6      40      30       6       2       2
  11      18       3      15      11       2       1       1
  12       8       2       6       4       0       2       0
  13       6       0       6       5       1       0       0
  14       0       0       0       0       0       0       0
(vhdlsim): Simulation complete, time is 1490944 NS.
#
```


AB TAMS
Technische Aspekte Multimodaler Systeme
Andreas Mäder
maeder@informatik.uni-hamburg.de

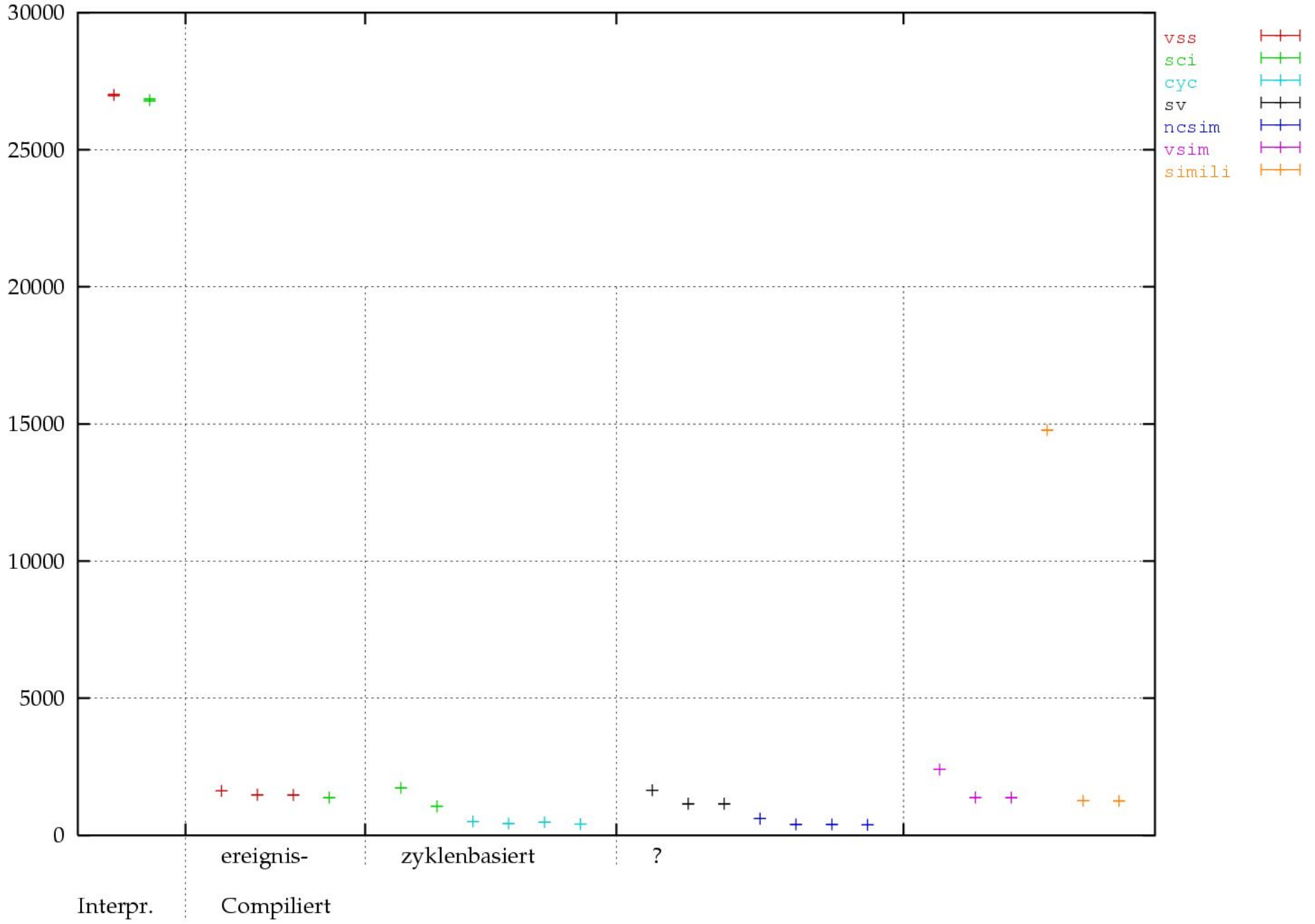
Multiplizierer Flächen- und Zeitbedarf



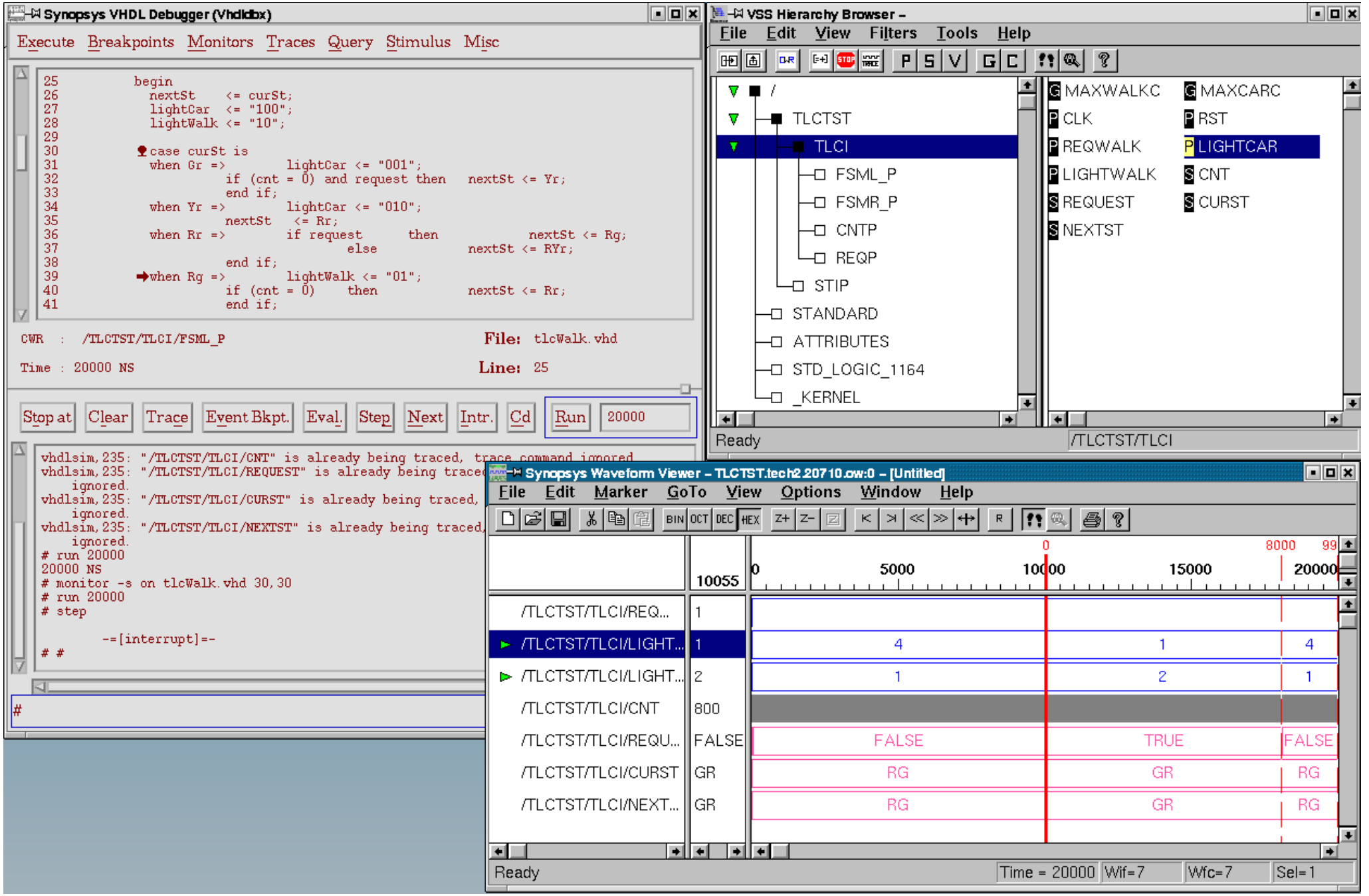
OPW RT-Code


1800
1600
1400
1200
1000
800
600
400
200
0
vss
sci
cyc
sv
ncsim
vsim
simili
Fehler
ereignis-
zyklenbasiert
?
Interpretation
Compiliert

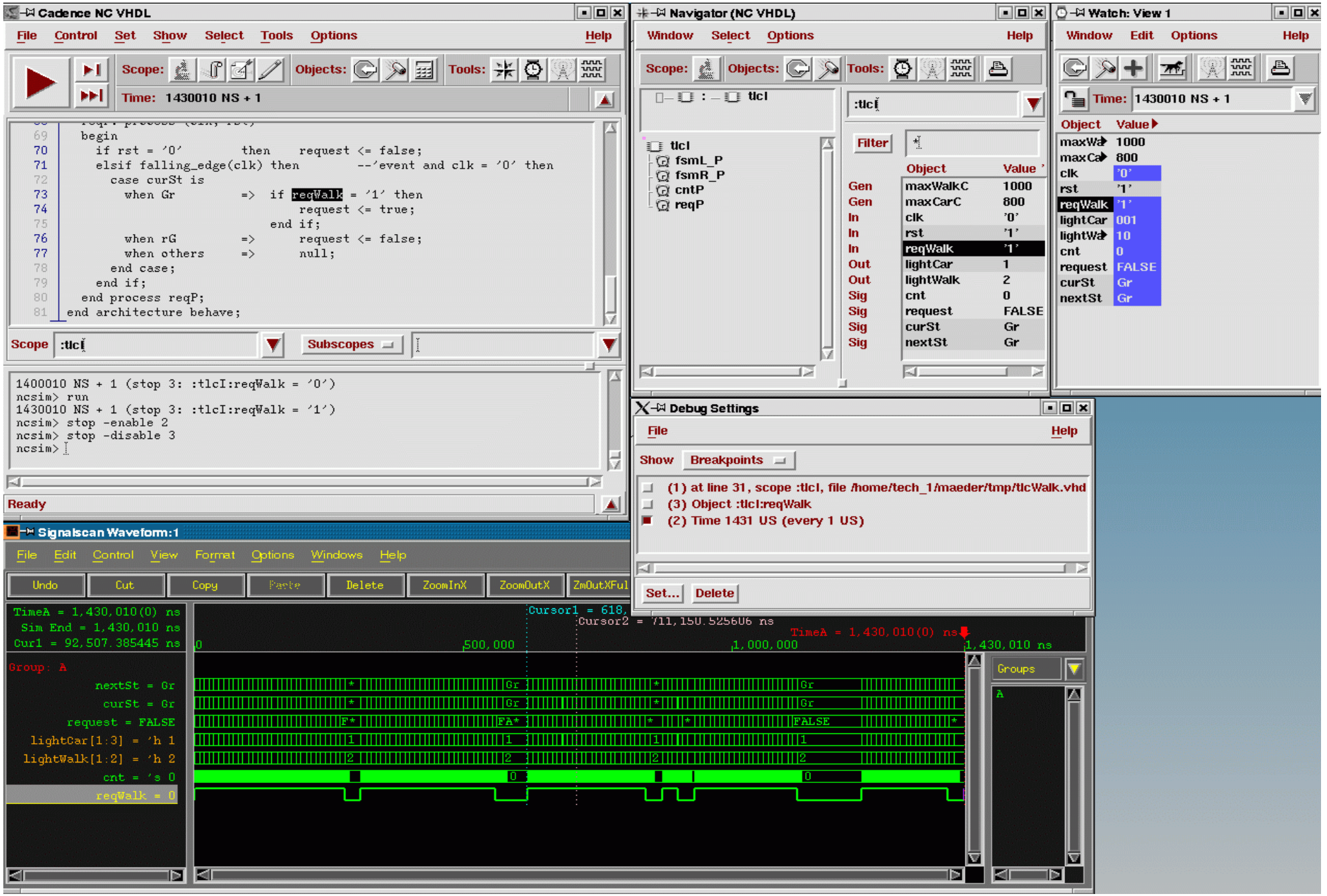
Clock RT-Code



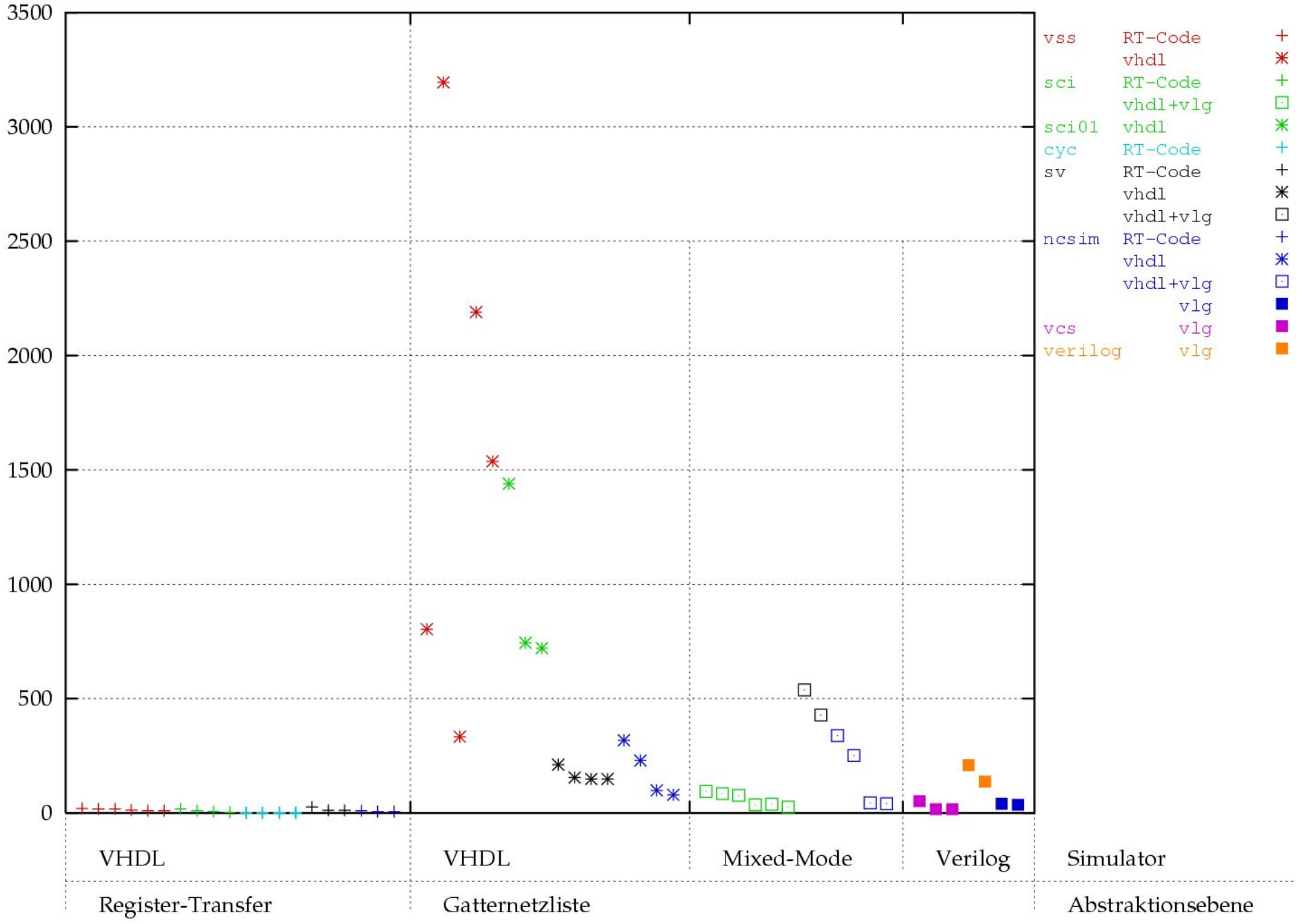
VSS


Synopsys VHDL Debugger (Vhdldbx)
Execute Breakpoints Monitors Traces Query Stimulus Misc
25 begin
26 nextSt <= curSt;
27 lightCar <= "100";
28 lightWalk <= "10";
29
30 case curSt is
31 when Gr => lightCar <= "001";
32 if (cnt = 0) and request then nextSt <= Yr;
33 end if;
34 when Yr => lightCar <= "010";
35 nextSt <= Rr;
36 when Rr => if request then nextSt <= Rg;
37 else nextSt <= RYr;
38 end if;
39 when Rg => lightWalk <= "01";
40 if (cnt = 0) then nextSt <= Rr;
41 end if;
CWR : /TLCTST/TLCI/FSML_P
File: tlcWalk.vhd
Time : 20000 NS
Line: 25
Stop at Clear Trace Event Bkpt. Eval. Step Next Intr. Cd Run 20000
vhdlsim,235: "/TLCTST/TLCI/CNT" is already being traced, trace command ignored.
vhdlsim,235: "/TLCTST/TLCI/REQUEST" is already being traced, trace command ignored.
vhdlsim,235: "/TLCTST/TLCI/CURST" is already being traced, trace command ignored.
vhdlsim,235: "/TLCTST/TLCI/NEXTST" is already being traced, trace command ignored.
run 20000
20000 NS
monitor -s on tlcWalk.vhd 30,30
run 20000
step
-=[interrupt]=-
# #
#
VSS Hierarchy Browser –
File Edit View Filters Tools Help
/
TLCTST
TLCI
FSML_P
FSMR_P
CNTP
REQP
STIP
STANDARD
ATTRIBUTES
STD_LOGIC_1164
_KERNEL
MAXWALKC
MAXCARC
CLK
RST
REQWALK
LIGHTCAR
LIGHTWALK
CNT
REQUEST
CURST
NEXTST
Ready
/TLCTST/TLCI
Synopsys Waveform Viewer – TLCTST.tech2.20710.cw:0 – [Untitled]
File Edit Marker GoTo View Options Window Help
10055
0
5000
10000
15000
20000
8000
/TLCTST/TLCI/REQ... 1
/TLCTST/TLCI/LIGHT... 1
/TLCTST/TLCI/LIGHT... 2
/TLCTST/TLCI/CNT 800
/TLCTST/TLCI/REQU... FALSE
/TLCTST/TLCI/CURST GR
/TLCTST/TLCI/NEXT... GR
4 1 4
1 2 1
FALSE TRUE FALSE
RG GR RG
RG GR RG
Ready
Time = 20000
Wif=7
Wfc=7
Sel=1

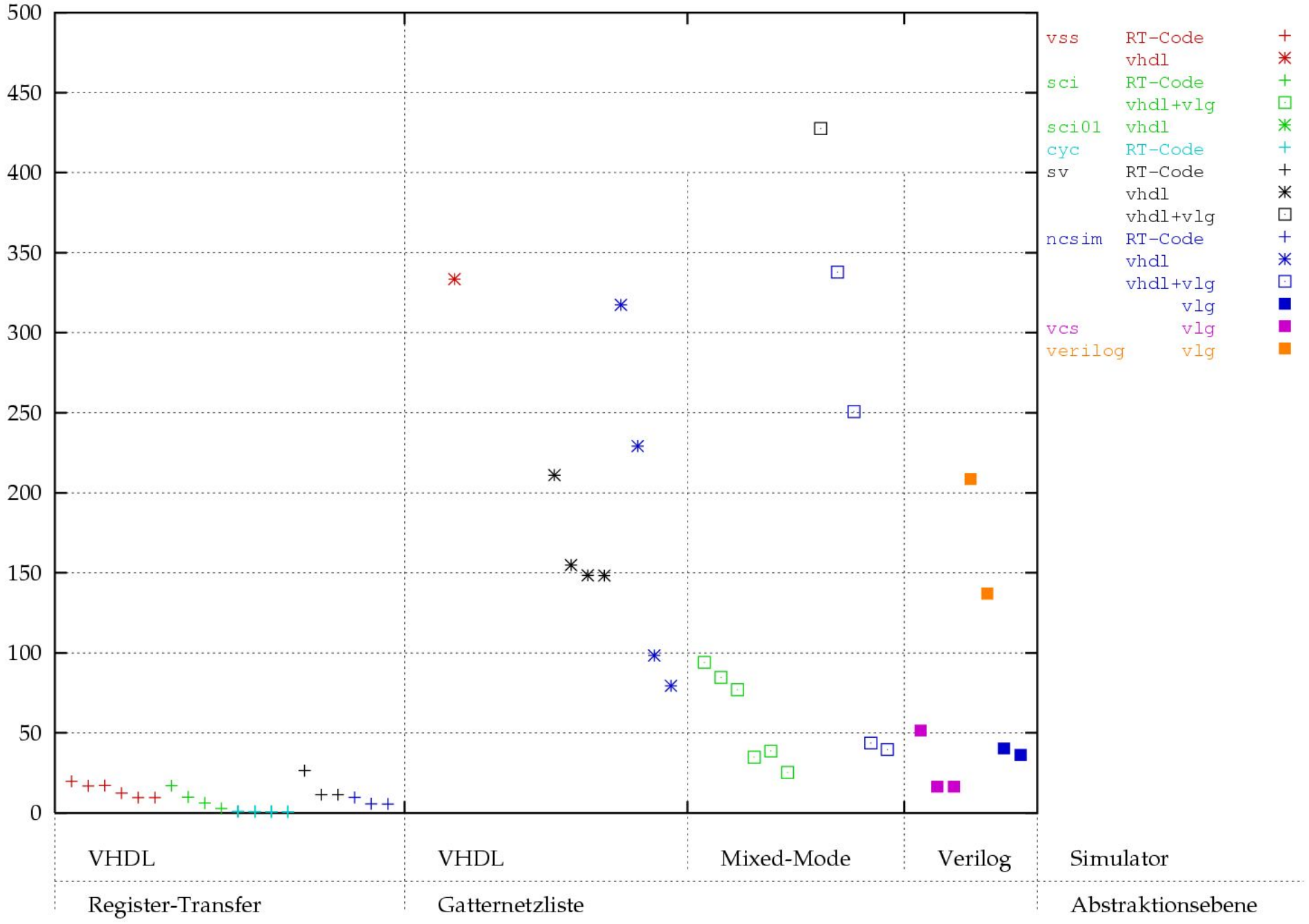
NCSim


Cadence NC VHDL
File Control Set Show Select Tools Options Help
Scope: Objects: Tools:
Time: 1430010 NS + 1
begin
if rst = '0' then request <= false;
elsif falling_edge(clk) then --'event and clk = '0' then
case curSt is
when Gr => if reqWalk = '1' then
request <= true;
end if;
when rG => request <= false;
when others => null;
end case;
end if;
end process reqP;
end architecture behave;
Scope :tlcI Subscopes
1400010 NS + 1 (stop 3: :tlcI:reqWalk = '0')
ncsim> run
1430010 NS + 1 (stop 3: :tlcI:reqWalk = '1')
ncsim> stop -enable 2
ncsim> stop -disable 3
ncsim>
Ready
Navigator (NC VHDL)
Window Select Options Help
Scope: Objects: Tools:
tlcI
fsmL_P
fsmR_P
cntP
reqP
Filter
Object Value
maxWalkC 1000
maxCarC 800
clk '0'
rst '1'
reqWalk '1'
lightCar 1
lightWalk 2
cnt 0
request FALSE
curSt Gr
nextSt Gr
Watch: View 1
Window Edit Options Help
Time: 1430010 NS + 1
Object Value
maxWa 1000
maxCa 800
clk '0'
rst '1'
reqWalk '1'
lightCar 001
lightWa 10
cnt 0
request FALSE
curSt Gr
nextSt Gr
Debug Settings
File Help
Show Breakpoints
(1) at line 31, scope :tlcI, file /home/tech_1/maeder/tmp/tlcWalk.vhd
(3) Object :tlcI:reqWalk
(2) Time 1431 US (every 1 US)
Set... Delete
Signalscan Waveform:1
File Edit Control View Format Options Windows Help
Undo Cut Copy Paste Delete ZoomInX ZoomOutX ZmOutXFul
TimeA = 1,430,010(0) ns
Sim End = 1,430,010 ns
Cur1 = 92,507.385445 ns
Cursor1 = 618,
Cursor2 = 711,150.525606 ns
TimeA = 1,430,010(0) ns
0
500,000
1,000,000
1,430,010 ns
Group: A
nextSt = Gr
curSt = Gr
request = FALSE
lightCar[1:3] = 'h 1
lightWalk[1:2] = 'h 2
cnt = 's 0
reqWalk = 0
Groups
A

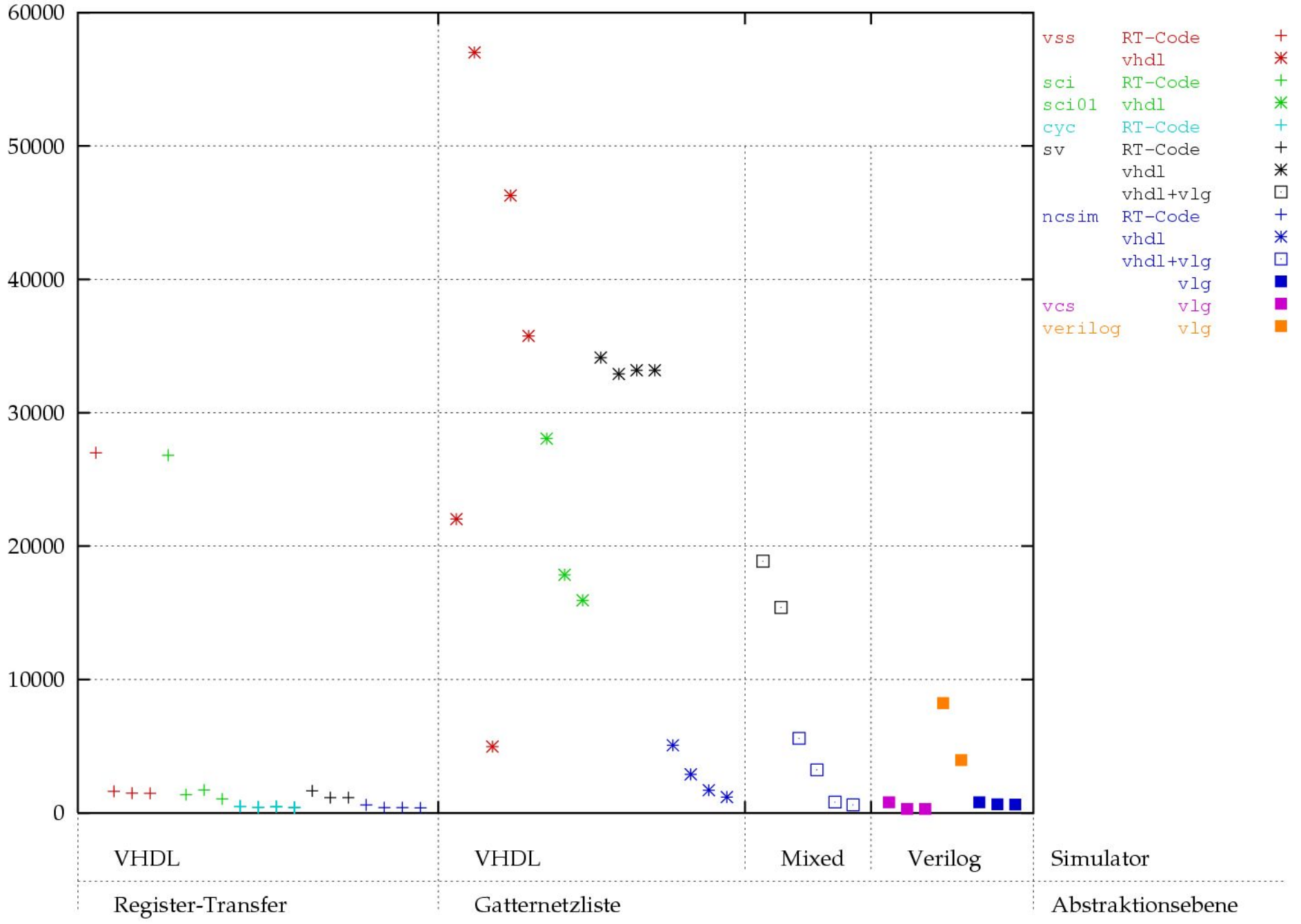
OPW


3500
3000
2500
2000
1500
1000
500
0
vss RT-Code
vhdl
sci RT-Code
vhdl+vlg
sci01 vhdl
cyc RT-Code
sv RT-Code
vhdl
vhdl+vlg
ncsim RT-Code
vhdl
vhdl+vlg
vlg
vcs vlg
verilog vlg
VHDL
VHDL
Mixed-Mode
Verilog
Simulator
Register-Transfer
Gatternetzliste
Abstraktionsebene

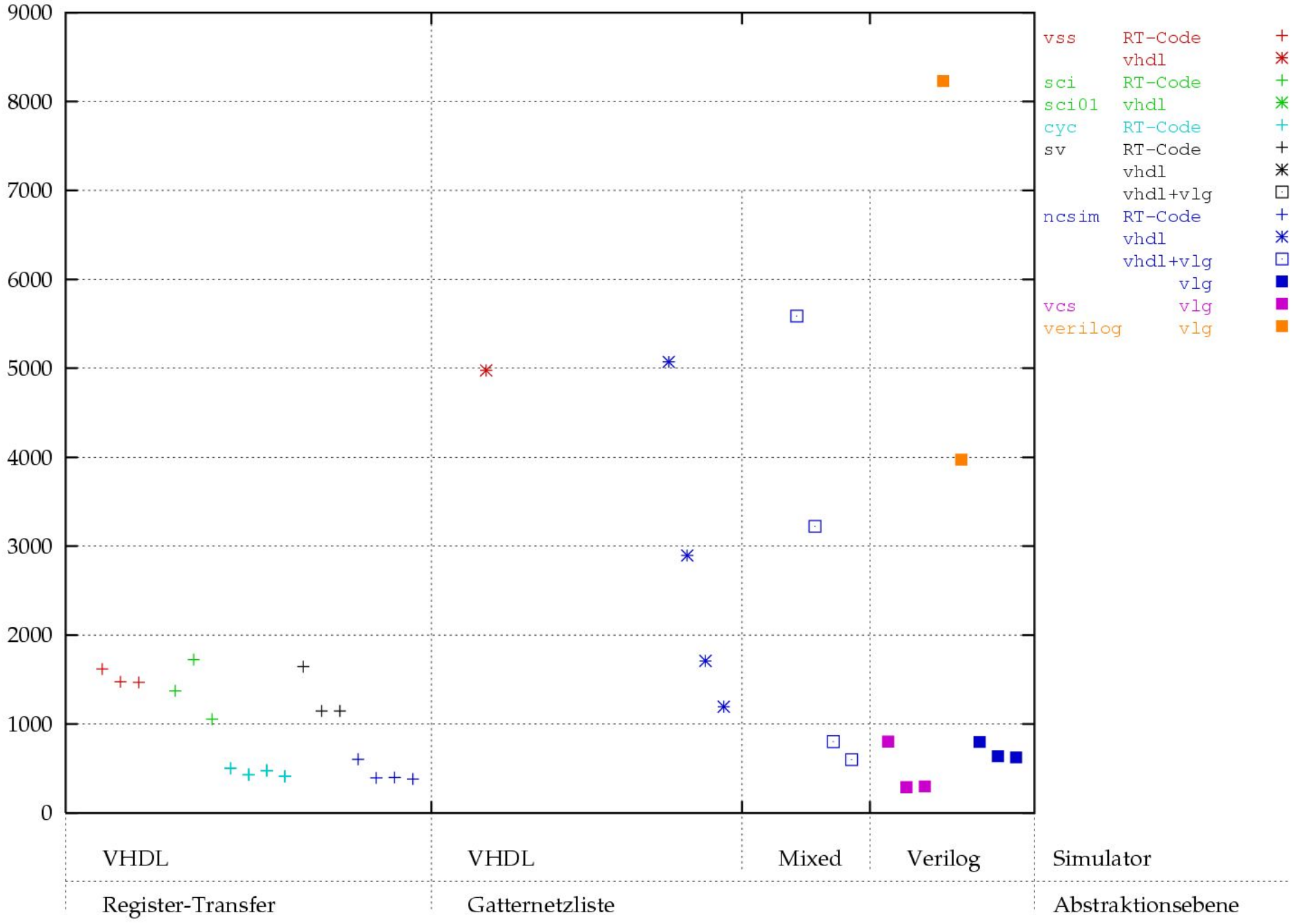
OPW


500
450
400
350
300
250
200
150
100
50
0
vss RT-Code
vhdl
sci RT-Code
vhdl+vlg
sci01 vhdl
cyc RT-Code
sv RT-Code
vhdl
vhdl+vlg
ncsim RT-Code
vhdl
vhdl+vlg
vlg
vcs vlg
verilog vlg
VHDL
VHDL
Mixed-Mode
Verilog
Simulator
Register-Transfer
Gatternetzliste
Abstraktionsebene

Clock


60000
50000
40000
30000
20000
10000
0
vss RT-Code
vhdl
sci RT-Code
sci01 vhdl
cyc RT-Code
sv RT-Code
vhdl
vhdl+vlg
ncsim RT-Code
vhdl
vhdl+vlg
vlg
vcs vlg
verilog vlg
VHDL
VHDL
Mixed
Verilog
Simulator
Register-Transfer
Gatternetzliste
Abstraktionsebene

Clock


9000
8000
7000
6000
5000
4000
3000
2000
1000
0
vss RT-Code
vhdl
sci RT-Code
sci01 vhdl
cyc RT-Code
sv RT-Code
vhdl
vhdl+vlg
ncsim RT-Code
vhdl
vhdl+vlg
vlg
vcs vlg
verilog vlg
VHDL
VHDL
Mixed
Verilog
Simulator
Register-Transfer
Gatternetzliste
Abstraktionsebene

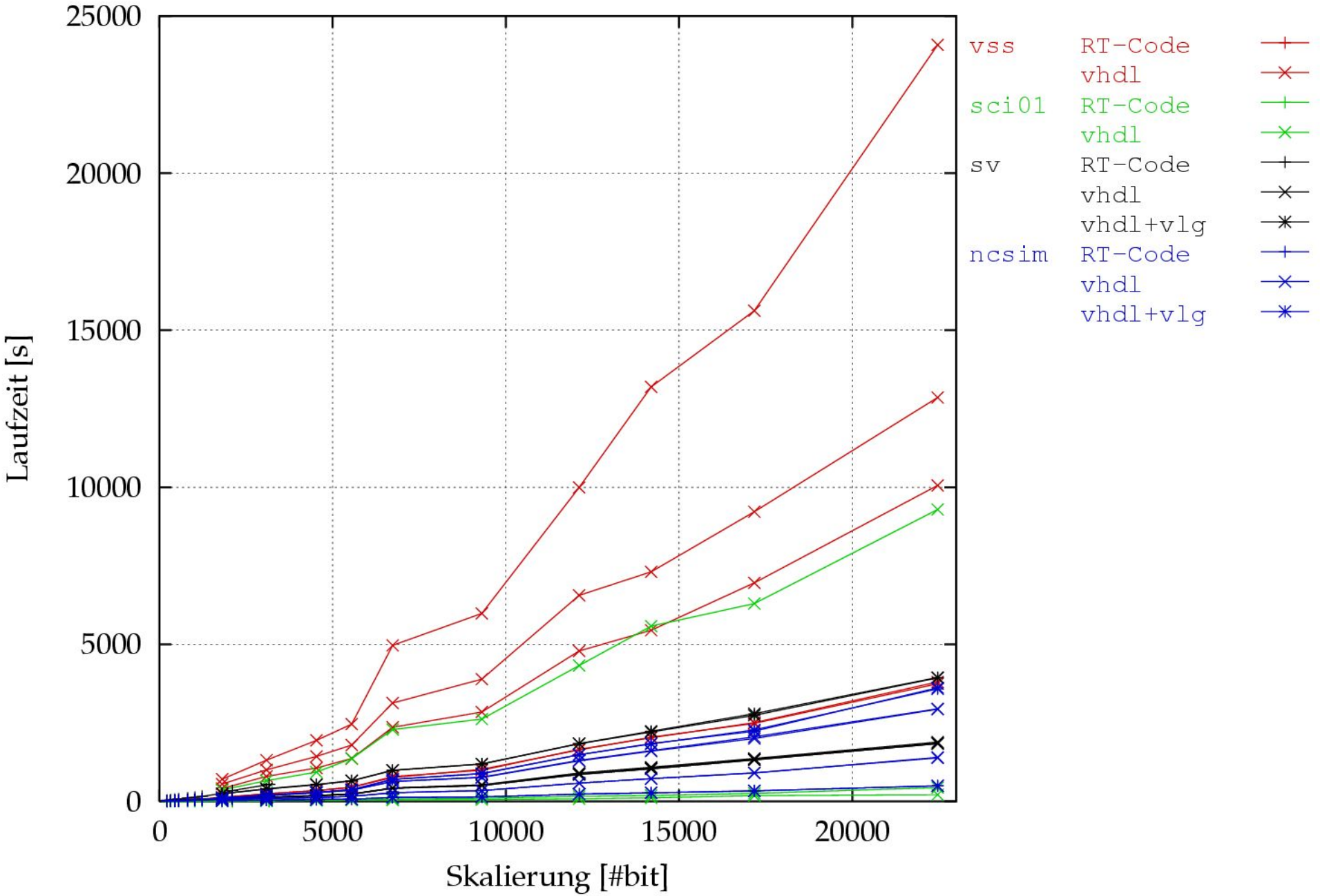
Schaltungsgröße

20.05.03

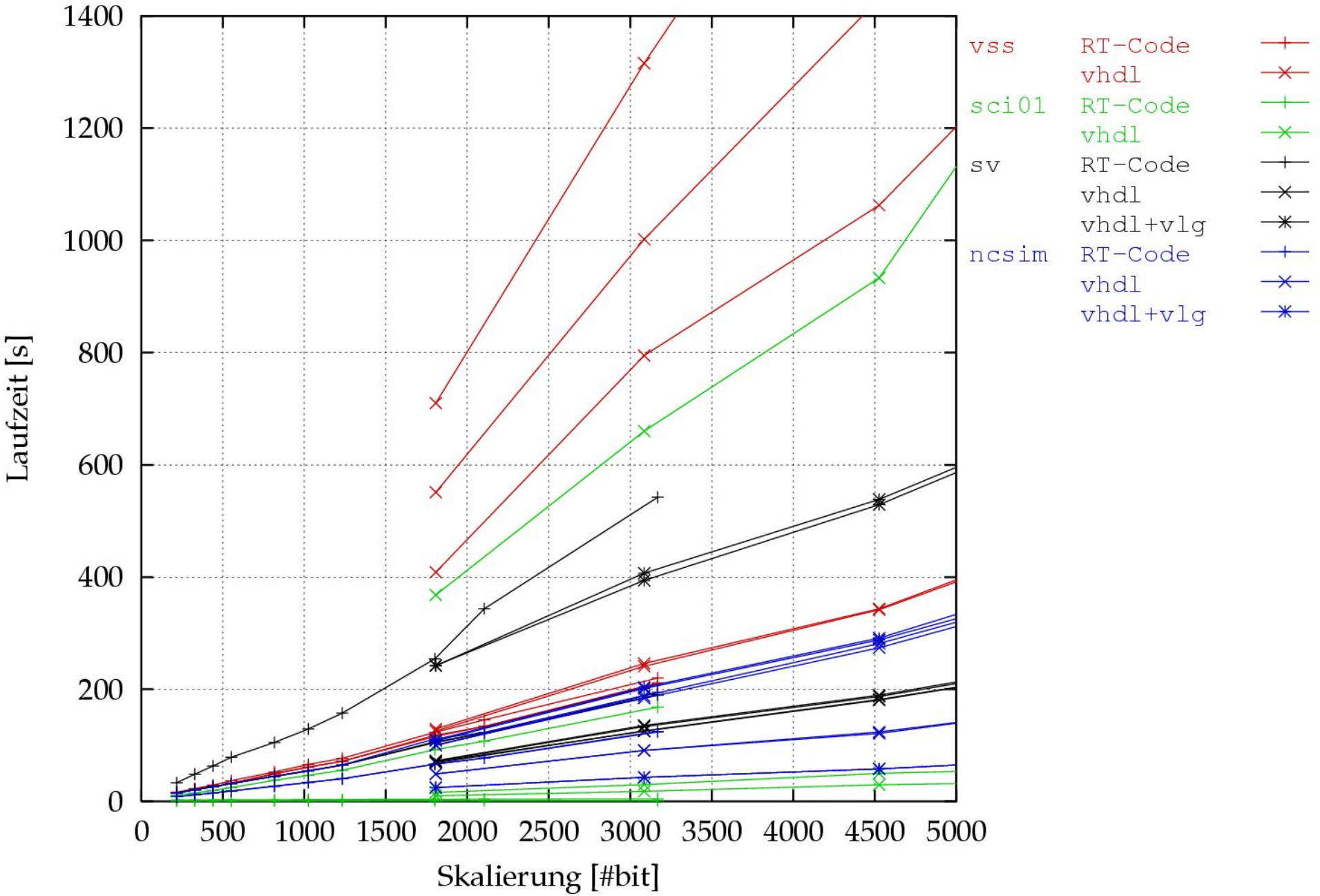
			Instanzen	Prozesse	Signale	Bit
Fußgängerampel	RT-Code	4 Proc.	1	5	9	12
		1 Proc.	1	2	5	8
Operationswerk	RT-Code		2	8	22	272
	Netzliste	AMS	779	3489	3610	3632
		UMC	823	3202	3179	3201
Funkwecker	RT-Code		9	48	130	405
	Netzliste	AMS	1620	6472	6811	6876
		UMC	1726	6388	6502	6560
Multiplizierer	RT-Code	8×8	1	14	15	216
		8×16	1	14	15	328
		8×24	1	14	15	440
		8×32	1	14	15	552
		16×16	1	26	27	816
		16×24	1	26	27	1024
		16×32	1	26	27	1232
		24×24	1	38	39	1800
		24×32	1	38	39	2104
		32×32	1	50	51	3168
	Netzliste AMS	8×8	603	1760	1762	1806
		8×16	1039	3015	3017	3085
		8×24	1517	4433	4435	4527
		8×32	1830	5429	5431	5547
		16×16	2266	6636	6638	6730
		16×24	3109	9188	9190	9306
		16×32	4036	11971	11973	12113
		24×24	4707	14048	14050	14190
		24×32	5614	17003	17005	17169
		32×32	7359	22271	22273	22461

Andreas Mäder
maeder@informatik.uni-hamburg.de

Multiplizierer



Multiplizierer

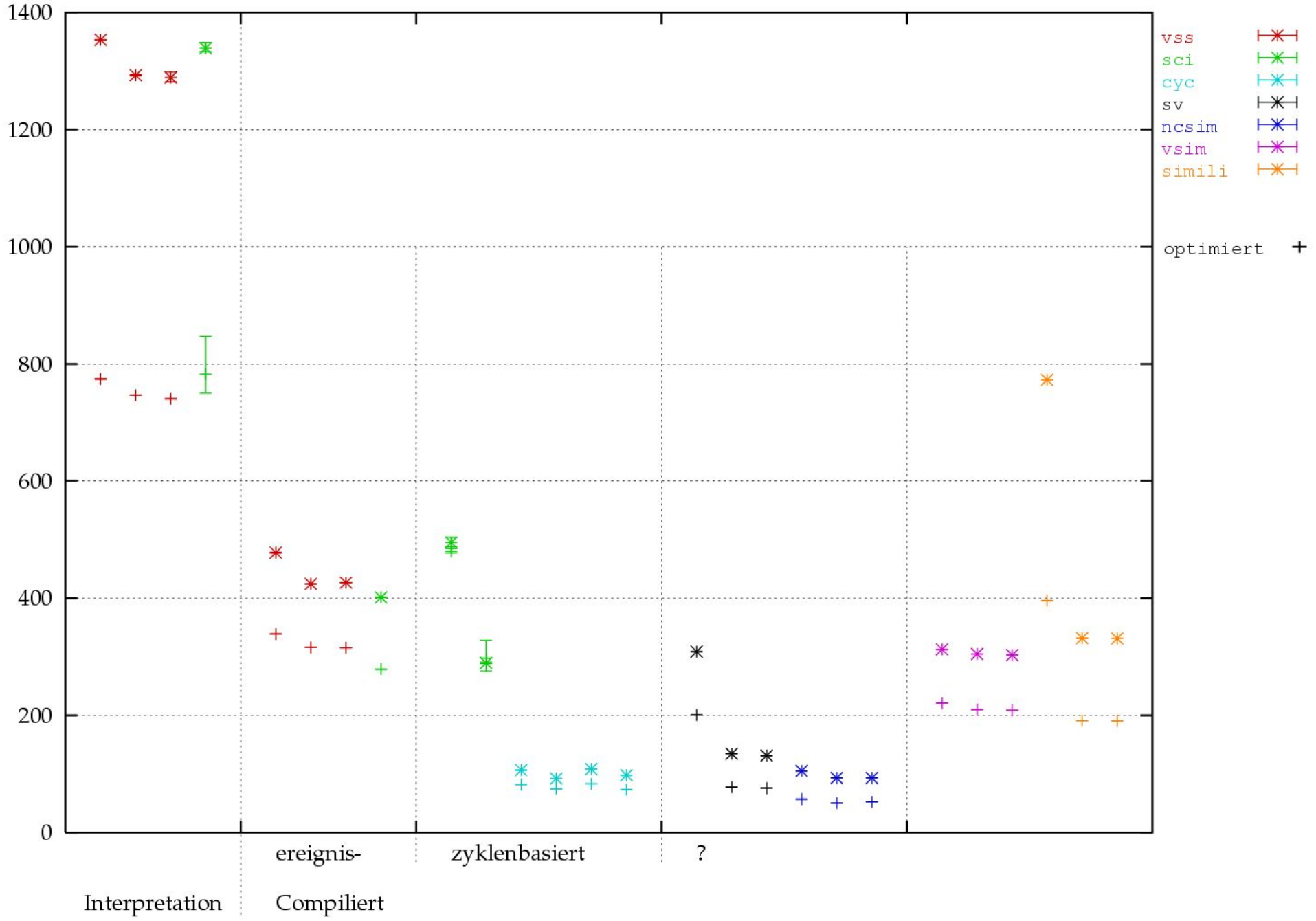


Ampelschaltung RT-Code

	4 Prozesse avg. $[s]$	1 Prozess avg. $[s]$	speed-up
VSS	1353,29	774,38	1,75
	1292,97	746,63	1,73
	1288,92	740,62	1,74
	477,68	339,02	1,41
	424,59	316,02	1,34
	426,68	315,43	1,35
Scirocco	1339,06	782,76	1,71
	401,41	278,88	1,44
	495,27	479,92	1,03
	289,33	297,6	0,98
Cyclone	106,89	82,1	1,3
	92,2	74,82	1,23
	108,13	83,02	1,3
	96,5	73,93	1,31
Leapfrog	308,79	200,54	1,54
	134,37	77,24	1,74
	131,13	75,79	1,73
NCSim	105,33	57,09	1,84
	93,28	50,4	1,85
	93,26	52,22	1,79
ModelSim	312,64	221,05	1,41
	305,09	210,02	1,45
	303,15	208,82	1,45
Simili	772,88	396,34	1,95
	331,82	190,72	1,74
	331,36	190,46	1,74

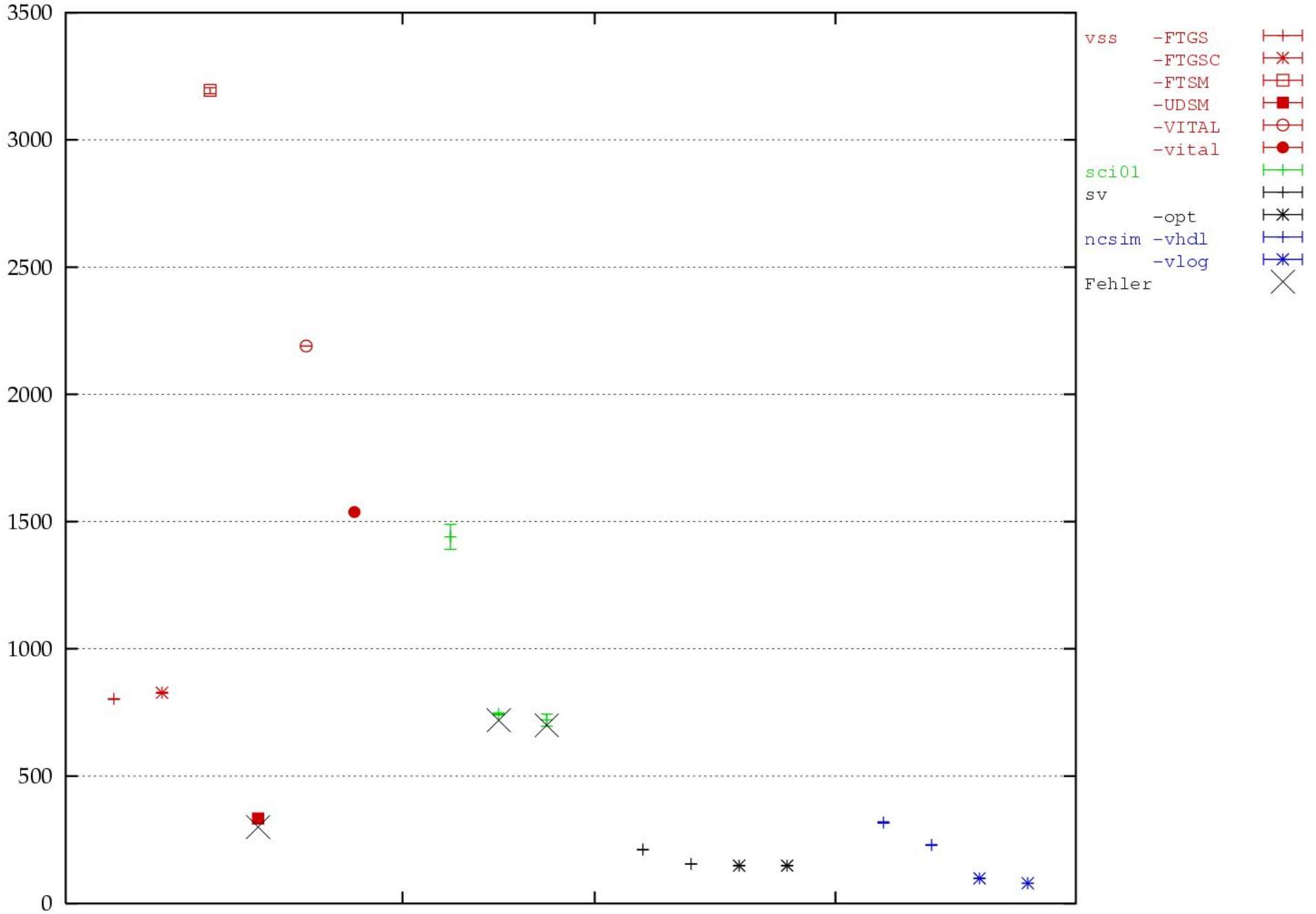
Andreas Mäder
maeder@informatik.uni-hamburg.de

Ampelschaltung RT-Code

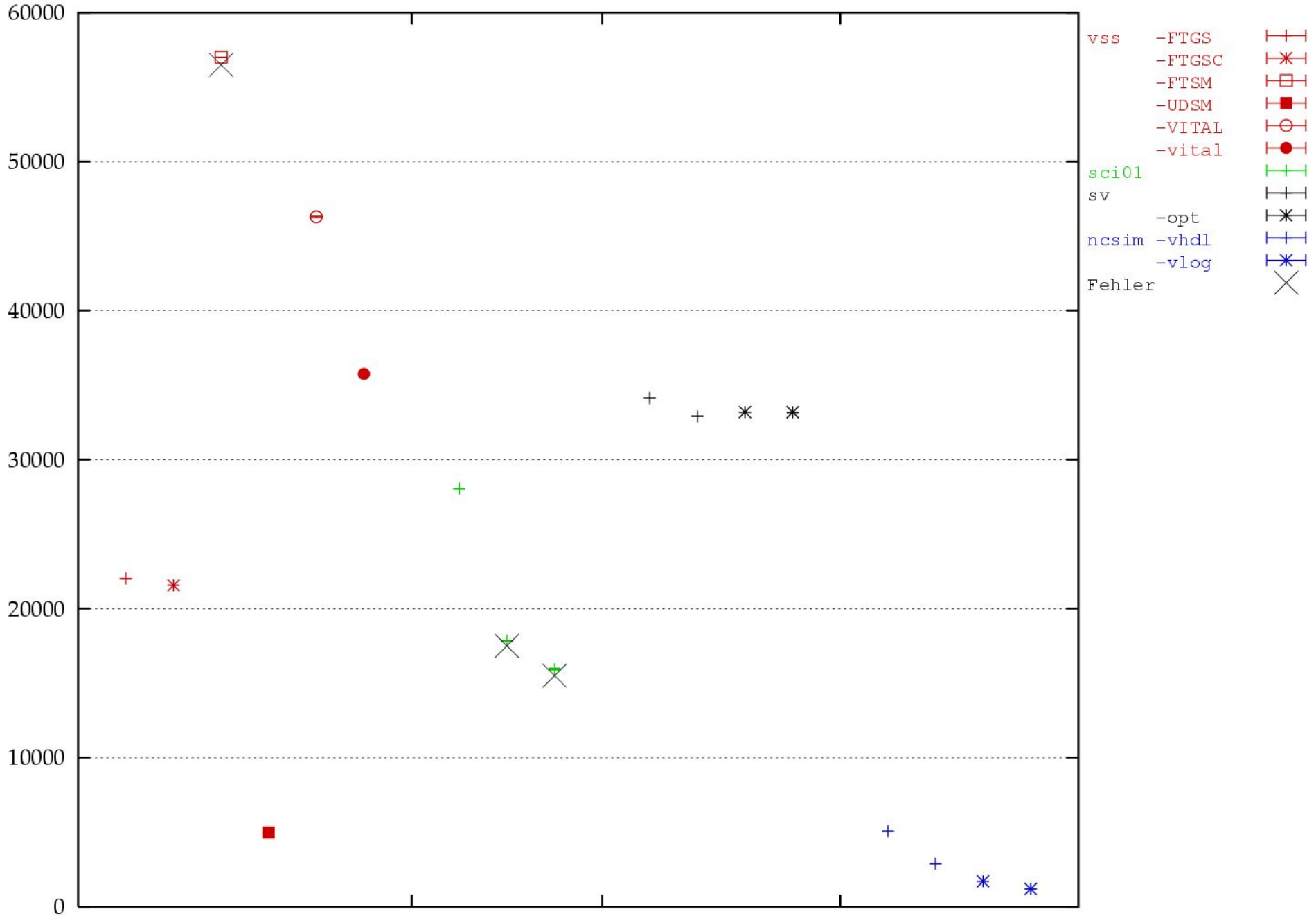


20.05.03

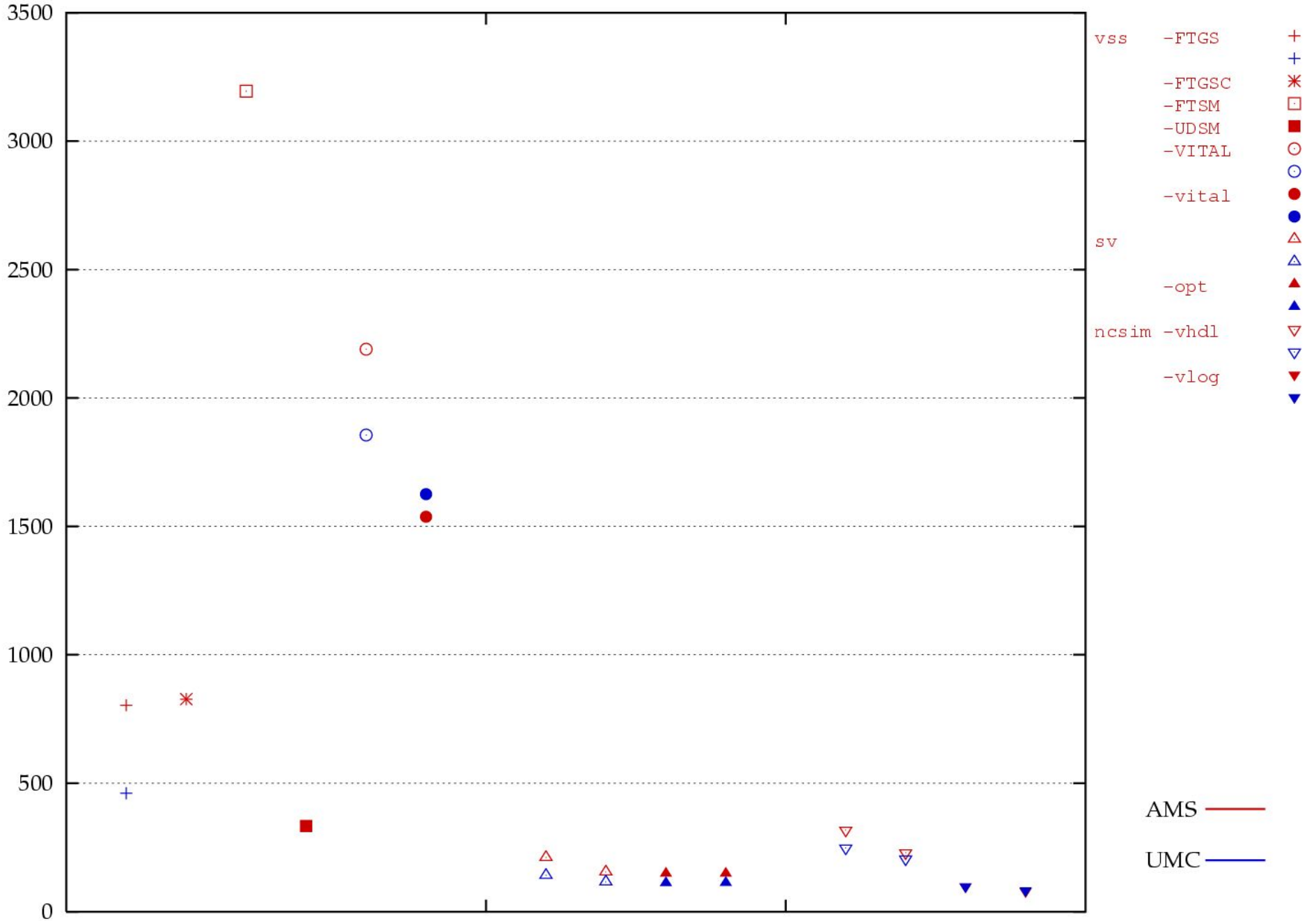
OPW Netzliste



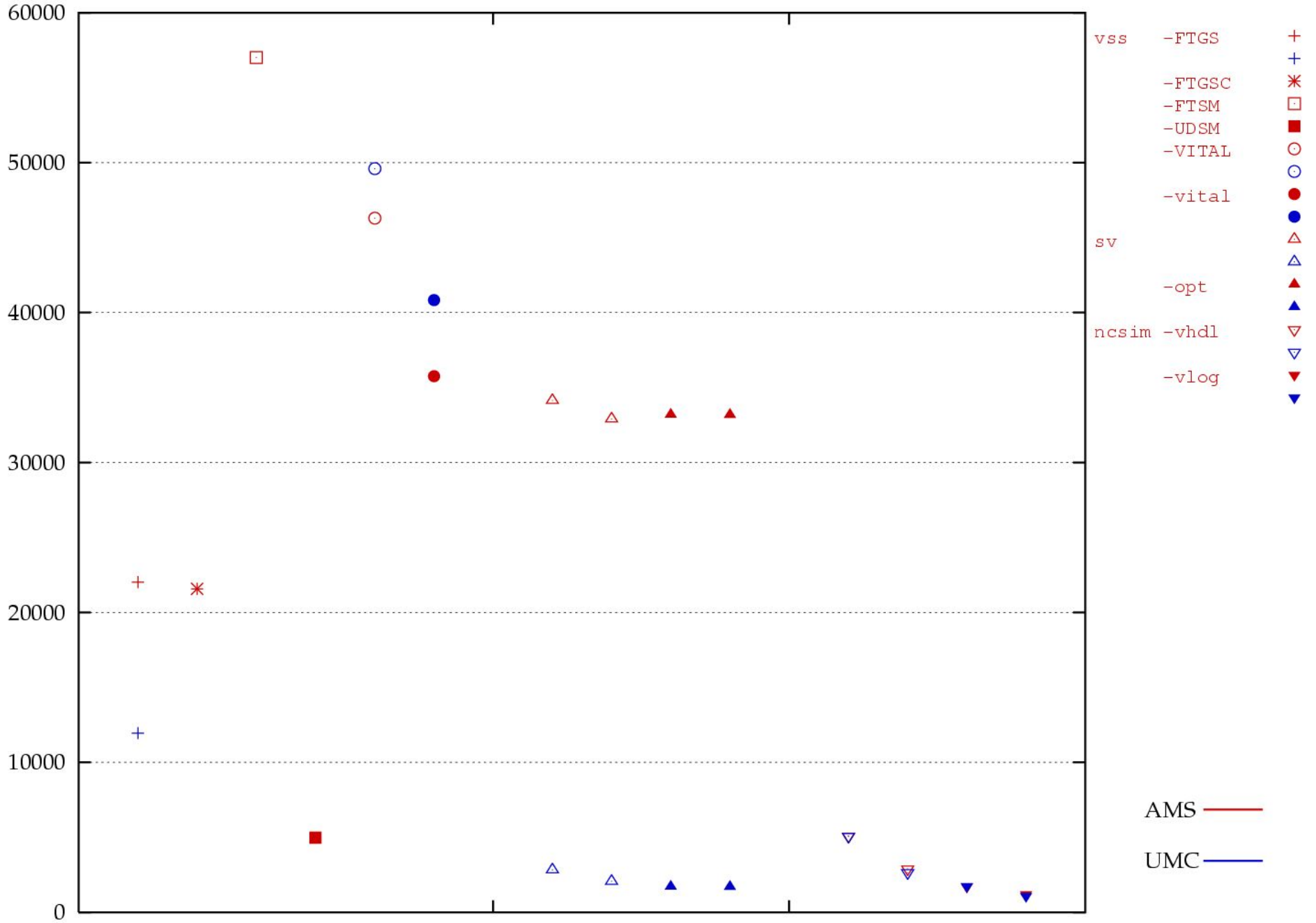
Clock Netzliste



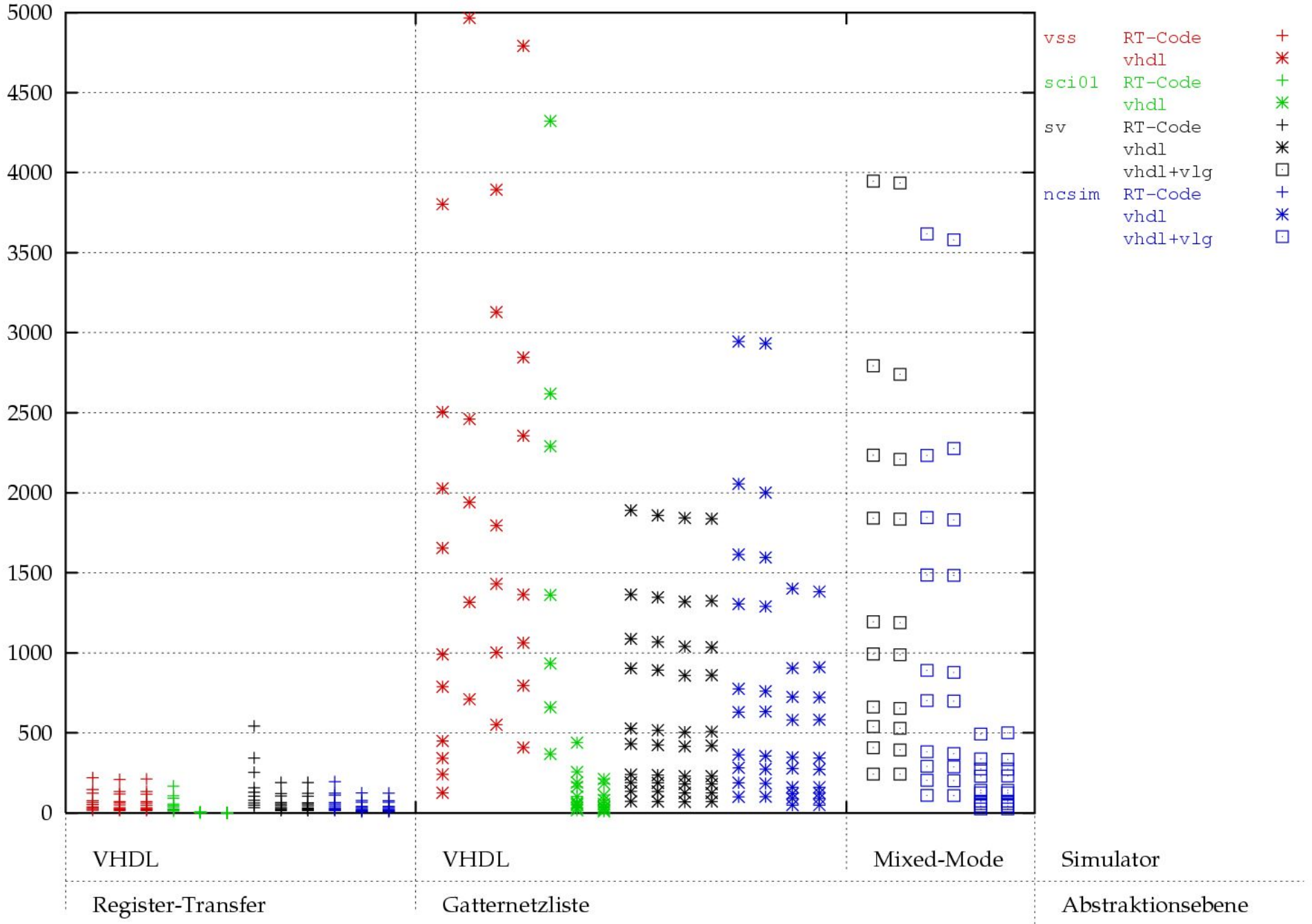
OPW Netzliste



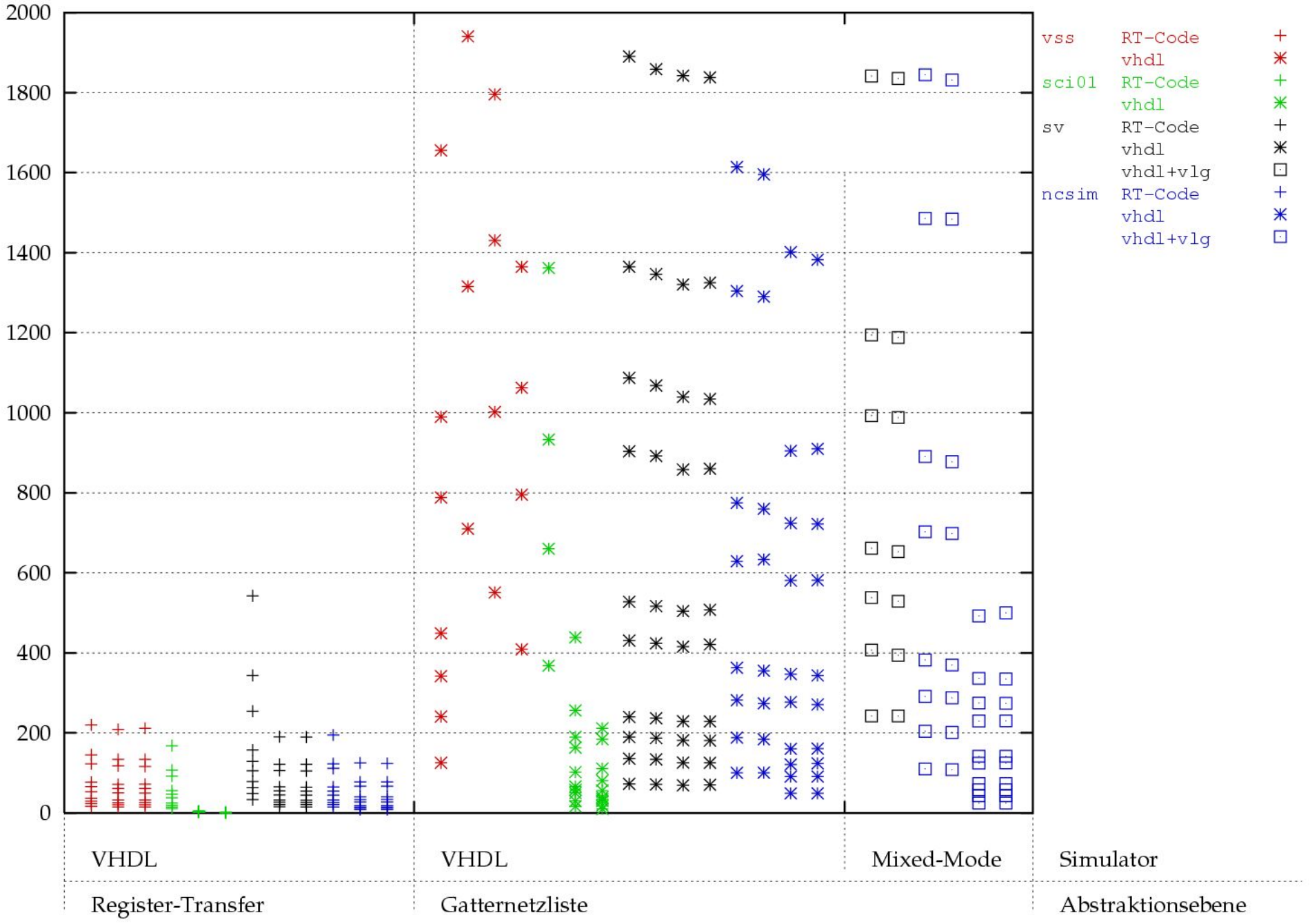
Clock Netzliste



Multiplizierer

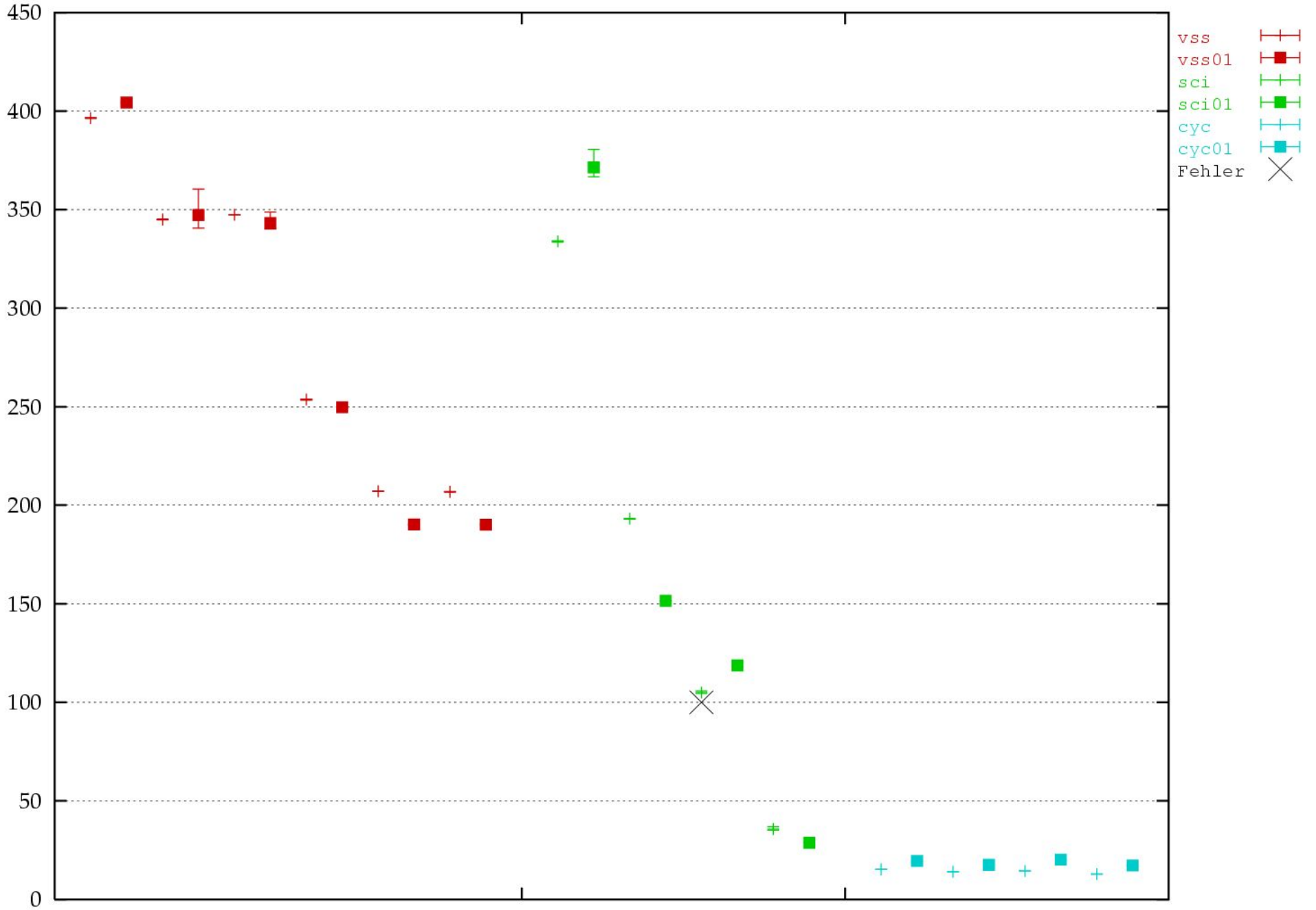


Multiplizierer

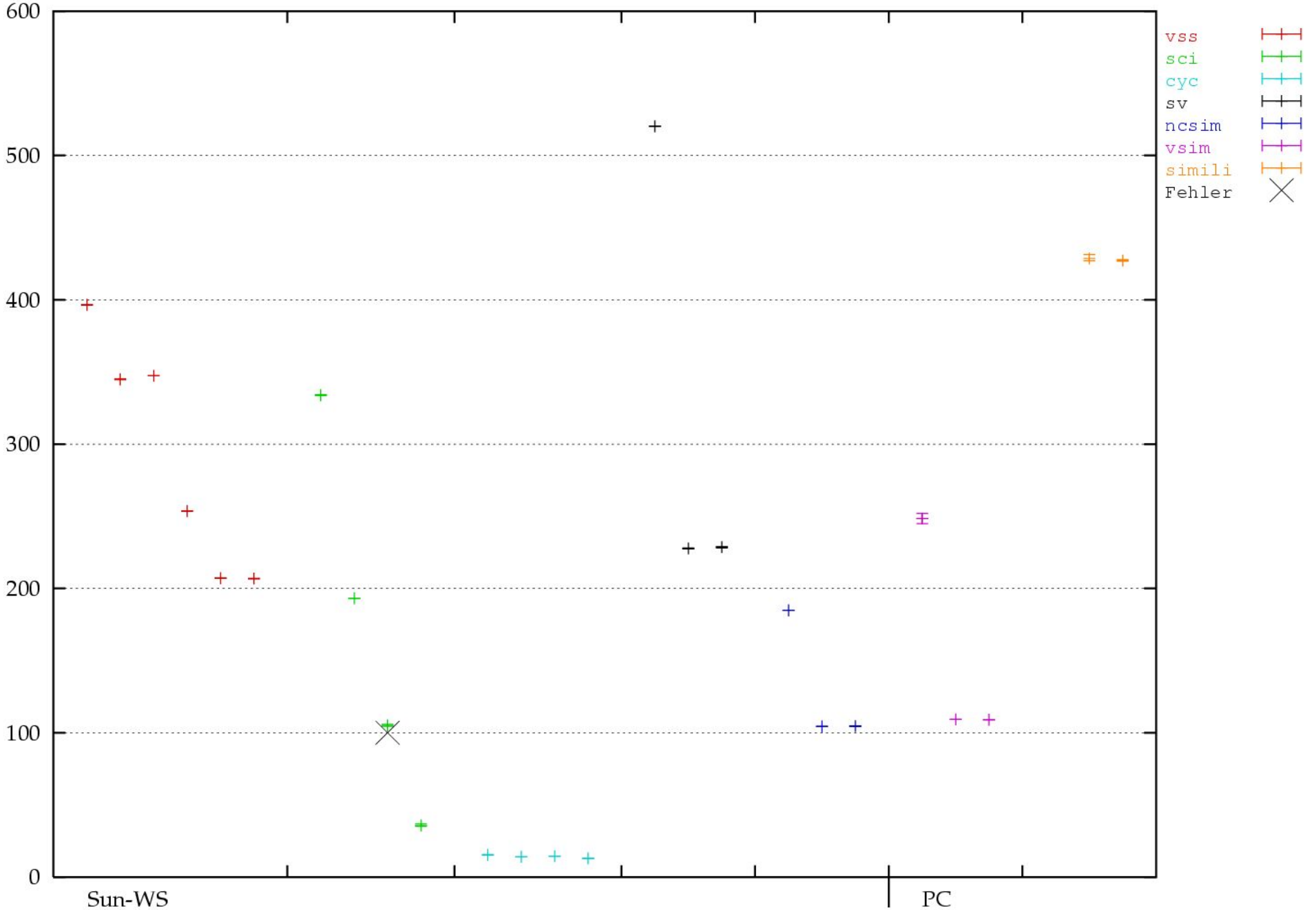


20.05.03

OPW RT-Code



OPW RT-Code


600
500
400
300
200
100
0
Sun-WS
PC
vss
sci
cyc
sv
ncsim
vsim
simili
Fehler

Clock RT-Code