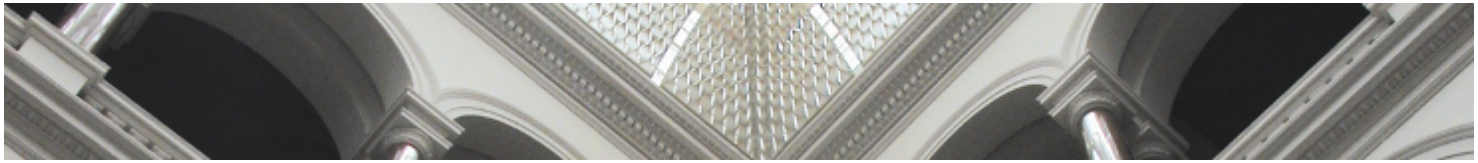
Technische Universität Berlin

8. Praktikumstermin

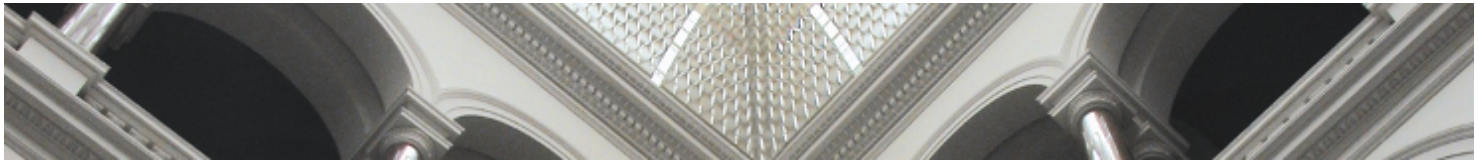
Rechnerorganisation Praktikum WiSe23/24 | Architektur eingebetteter Systeme | Debugging



Technische Universität Berlin

Debugging

- Testbenches automatisieren das Überprüfen aller Möglichkeiten
- Ein Fehler im Design führt oft zu vielen fehlgeschlagenen Tests
- Die Ursache liegt dabei nicht immer gleich auf der Hand
- Wir müssen also *debuggen* (die Fehlerursache finden)
- Dazu bleibt uns nur der Blick ins Wave-Fenster von *GtkWave*
- Darüber können wir die exakte Belegung jedes Signals zu jedem Zeitpunkt der durchgeführten Simulation nachvollziehen



Technische Universität Berlin

Wiederholung: std_logic

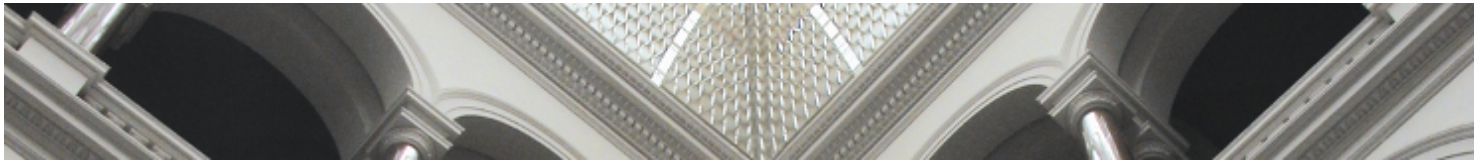
– Dabei macht sich die Verwendung von `std_logic` statt `bit` bezahlt:

- `U` (Uninitialized): Das Signal besitzt keinen Treiber

	U	X	0	1	Z	W	L	H	-	
U	U	U	U	U	U	U	U	U	U	(Auflösungsfunktion)
U	U	U	0	U	U	U	0	U	U	(`and`-Funktion)
U	U	U	U	1	U	U	U	1	U	(`or`-Funktion)

- `X` (Forcing Unknown): Das Signal besitzt mehrere Treiber

	U	X	0	1	Z	W	L	H	-	
X	U	X	X	X	X	X	X	X	X	(Auflösungsfunktion)
X	U	X	0	X	X	X	0	X	X	(`and`-Funktion)
X	U	X	X	1	X	X	X	1	X	(`or`-Funktion)



Technische Universität Berlin

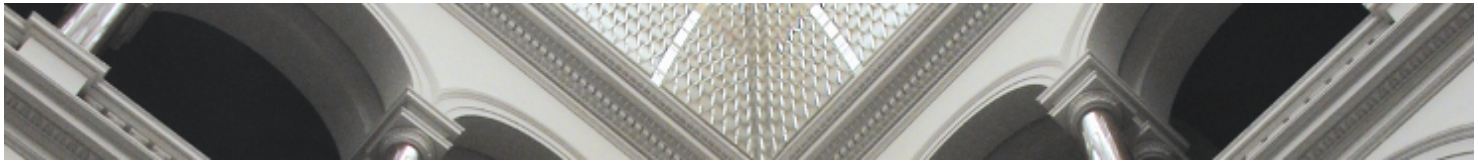
Fehlersuche

Fall 1: Gute Testbench

- Zeigt an, wo/wann genau sich der Fehler bemerkbar macht
  $\Rightarrow$ Dort ansetzen und Fehler auf seine Ursache zurückverfolgen!
- Aus welchem Schaltbaustein stammt das falsche Ergebnis?
- Wieso kommt es dort zu einem falschen Ergebnis?
- Woher kommen dessen Eingabesignale?

Fall 2: Schlechte/keine Testbench

- Es ist unklar, wo/wann genau sich der Fehler bemerkbar macht
  $\Rightarrow$ Alle Signale schrittweise ab t = 0ns auf Korrektheit prüfen!
- In der Regel deutlich aufwändiger als Fall 1



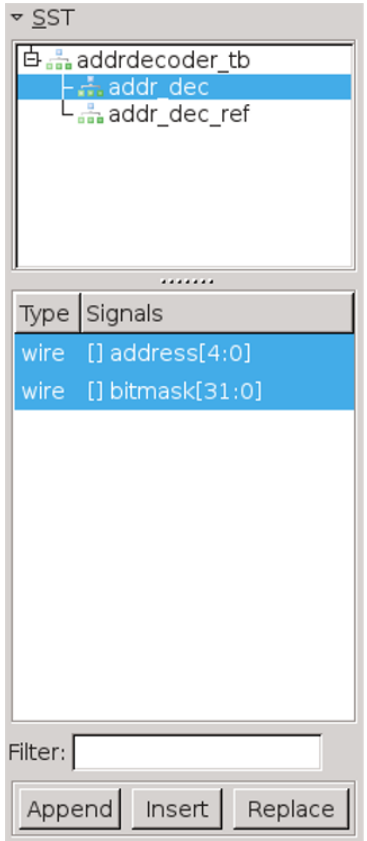
Technische Universität Berlin

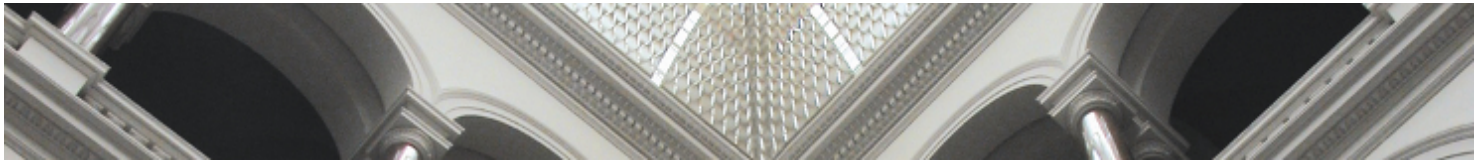
GtkWave

- Öffnen von *GtkWave* mittels

```
make clean all
make view_wave
```

- Strukturübersicht der Entities links oben
- Liste der Signale der ausgewählten Entität links unten
- Hier können wir auswählen, welche Signale welcher Bausteine wir beobachten wollen
- Dazu ziehen wir das Signal einfach bei gedrückter Maustaste ins offene Wave-Fenster
- oder benutzen den Button *Insert*.





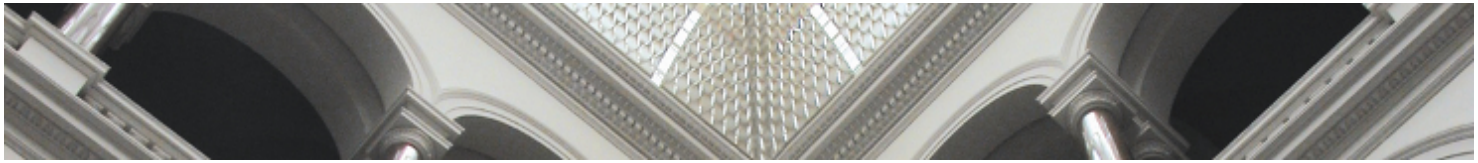
Technische Universität Berlin

Transcript-Fenster

- Anzeige der meisten Ausgaben der Konsole
- Doppelklick auf Eintrag setzt Cursor im Wave-Fenster auf Zeitpunkt der Ausgabe
- Nur verfügbar auf Universitäts-Systemen

Transcript

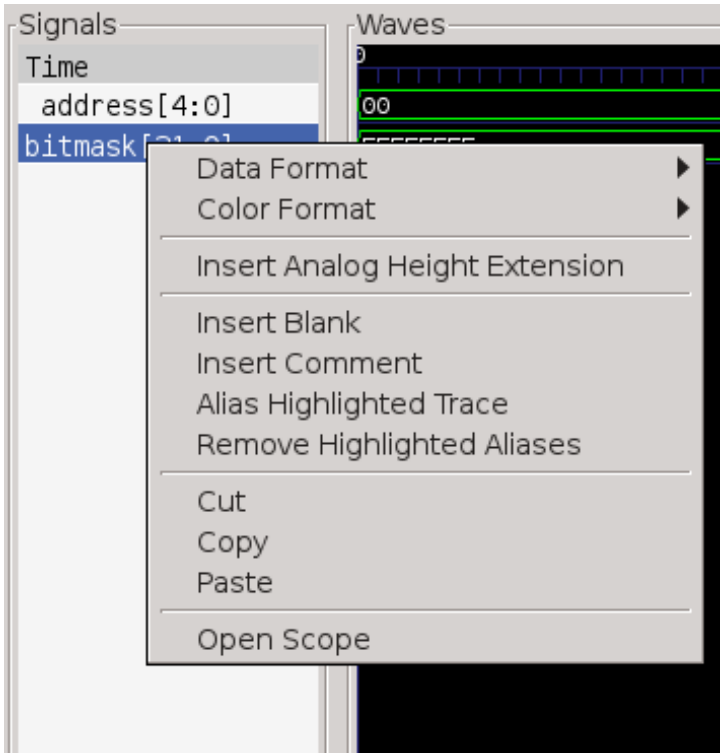
Timestamp	Type	Message	Location
0.000 ns+0	Warning	NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0	/afs/tu-berlin.de/home/p/phab
0.000 ns+0	Warning	NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0	addrDecoder.vhd:20:/addrdec
5.000 ns+0	Note	5000 ps: Falsches Ergebnis am Adressdekoder: "FFFFFFFF" (erwartet: "00000001")	addrDecoder.vhd:20:/addrdec
15.000 ns+0	Note	15000 ps: Falsches Ergebnis am Adressdekoder: "FFFFFFFF" (erwartet: "00000002")	addrDecoder.vhd:20:/addrdec
25.000 ns+0	Note	25000 ps: Falsches Ergebnis am Adressdekoder: "FFFFFFFF" (erwartet: "00000004")	addrDecoder.vhd:20:/addrdec
35.000 ns+0	Note	35000 ps: Falsches Ergebnis am Adressdekoder: "FFFFFFFF" (erwartet: "00000008")	addrDecoder.vhd:20:/addrdec
45.000 ns+0	Note	45000 ps: Weitere Fehler werden nicht angezeigt ...	addrDecoder.vhd:20:/addrdec
320.000 ns+0	Failure	Der Adressdekoder funktioniert nicht einwandfrei! (addrDecoder: 0/2 Punkte)	addrDecoder_tb.vhd:67:/addrd
>=320.000 ns+0	Note	Break in Process testbench at addrDecoder_tb.vhd line 67	addrDecoder_tb.vhd:67:/addrd
>=320.000 ns+0	Note	Stopped at addrDecoder_tb.vhd line 67	addrDecoder_tb.vhd:67:/addrd



Technische Universität Berlin

Signaleigenschaften

– *Linksklick auf Signal, dann Rechtsklick → Kontextmenü:*



– **Alias Highlited Trace:** Alias, der statt des Entity-Bezeichners im Wave-Fenster angezeigt wird, falls gesetzt.

– **Data Format:** Darstellung der Werte eines Signals, nützlich insbesondere bei größeren Vektoren, z.B.:
  - binär
  - hexadezimnal
  - unsigned

– **Color Format:** Signalverlaufs-Farbe