TECHNISCHE UNIVERSITÄT DRESDEN

Vortrag zum Ergebnis der Literaturrecherche

Fehlerinjektion mittels Trace-Architektur auf einem Mips-Prozessor

Matthias Brinker

Dresden, 27.10.2016

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Gliederung

Einleitung und Motivation

Einleitung und Motivation

Wieso ist Fehlerinjektion wichtig?

- Auswirkung von Fehlern auf das untersuchte System
- Auswirkung von Fehlertoleranzmaßnahmen

Trace-Architektur

Trace-Architektur

Was ist eine Trace-Architektur

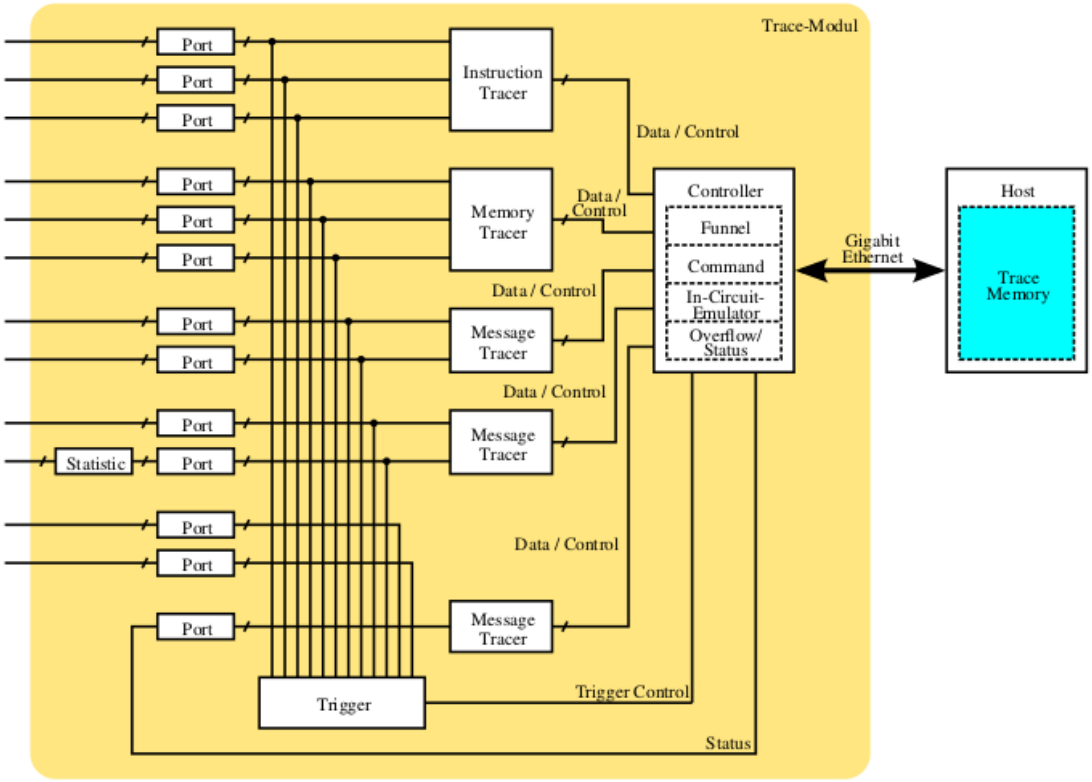
- Hardwareerweiterung des Prozessors zum Aufzeichnen relevanter Daten für Fehlerevaluation
- Benötigte Erweiterungen:
  - Schnittstelle zum Aufzeichnen der Daten
  - (Filter)
  - Kompressionsmodul
  - Übertragungskanal zum Host-PC

Trace-Architektur

Was sind die relevanten Daten

- Programm-Zähler
  - + niedrige Bandbreite, weil hohe Kompression durch PFCM
  - - Keine sichere Erkennung des Systemversagens
- Datenbus
  - + Erkennung falscher Schreibzugriffe in den Speicher (Systemversagen)
  - - hohe Bandbreite, da keine Kompression
- Wahlmöglichkeit: mit/ohne Zeitstempel

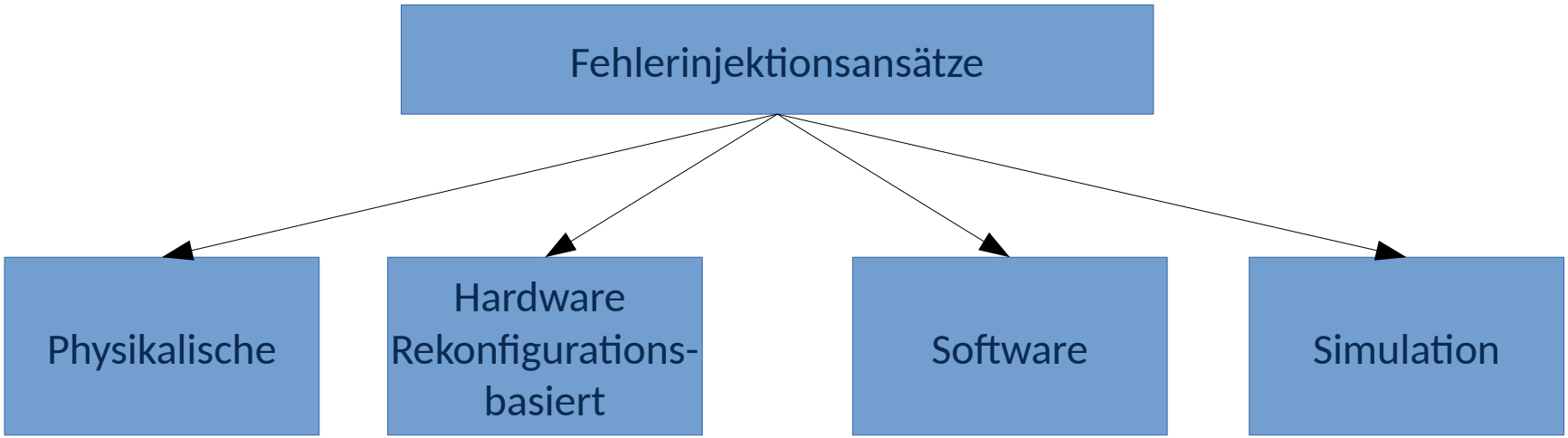
Trace-Architektur



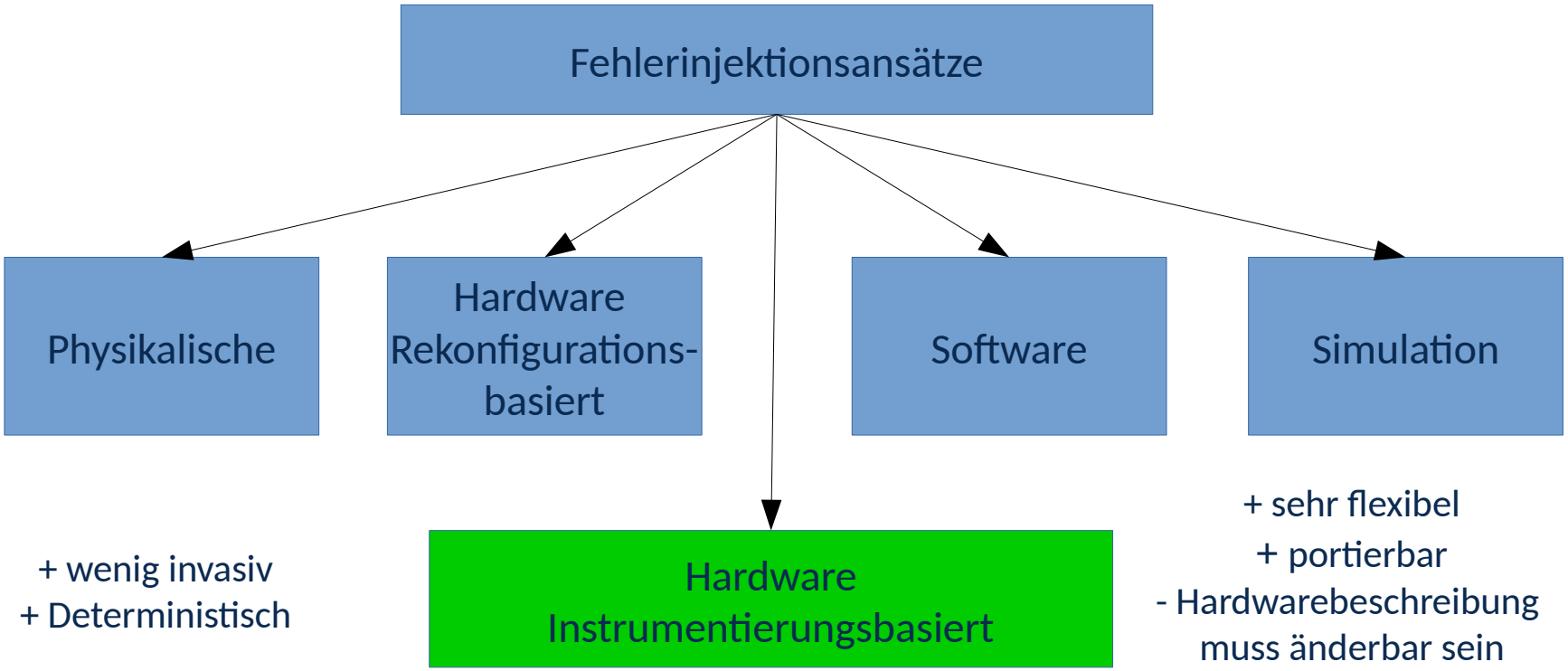
Quelle: [2]

Methoden zur Fehlerinjektion

Methoden zur Fehlerinjektion



Methoden zur Fehlerinjektion



Fehlerinjektion mittels Trace

Fehlerinjektion mittels Trace

Zielstellung

- Genaue Kontrolle über Ort und Zeit der Injektion
- Fehlerfortpflanzung beobachtbar
- Experiment muss wiederholbar sein
- Möglichst wenig invasiv
- Real Time

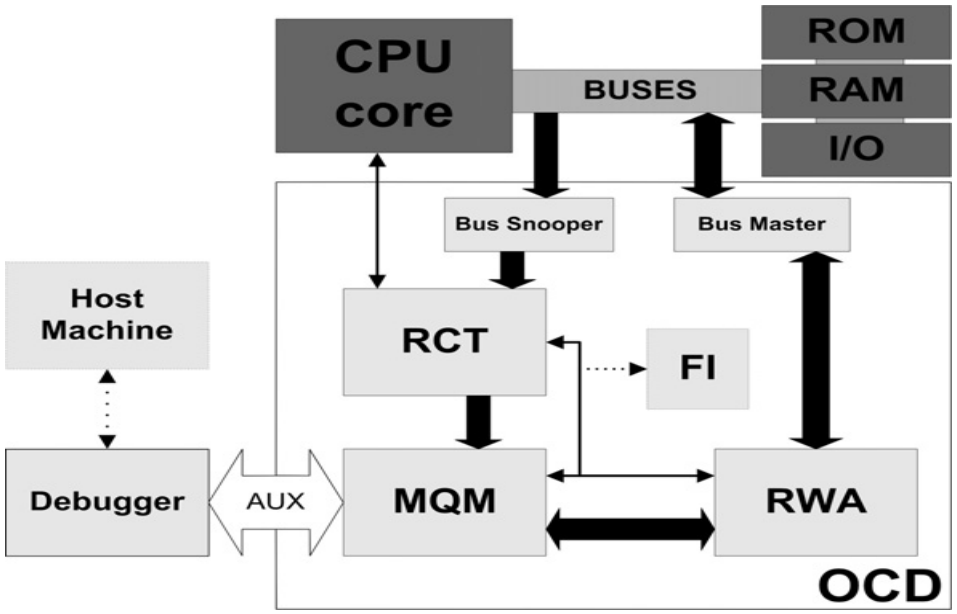
Fehlerinjektion nach Fidalgo[1]

Nötige Änderungen an der Hardware

- Nexus-Trace-Architektur (IEEE-ISTO 5001-2003)
- On-Chip-Debugger wird um ein FI-Modul erweitert
- Debugger wird erweitert, dass er Scripte ausführen und auf Anfragen reagieren kann
- (Optional) Erweiterung für Register-Faultinjection

Fehlerinjektion nach Fidalgo[1]

Ablauf einer Fehlerinjektion



- Trifft auf Breakpoint
- FI übernimmt Kontrolle über OCD
- Sendet die von RWA erhaltene Daten an Debugger
- Wendet Fehlermaske an
- FI injiziert den Fehler auf den BUS

Quelle: [1]

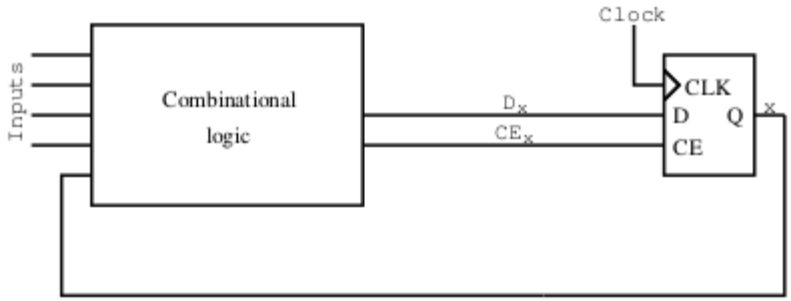
Fehlerinjektion nach Gunia[²]

Nötige Änderung der Hardware

- In den FlipFlops wird eine FI-Schnittstelle implementiert
- Trace-Architektur wird um ein FI-Control-Modul erweitert

Fehlerinjektion nach Gunia[²]

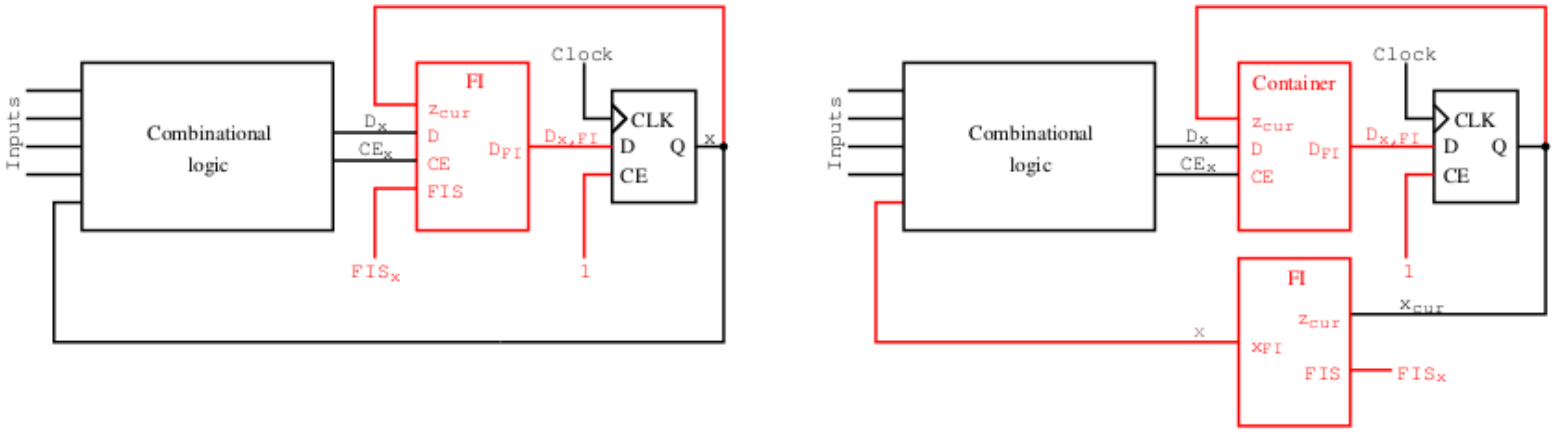
Flipflop ohne Fehlerinjektion



Quelle: [2]

Fehlerinjektion nach Gunia[²]

Flipflop mit Fehlerinjektion



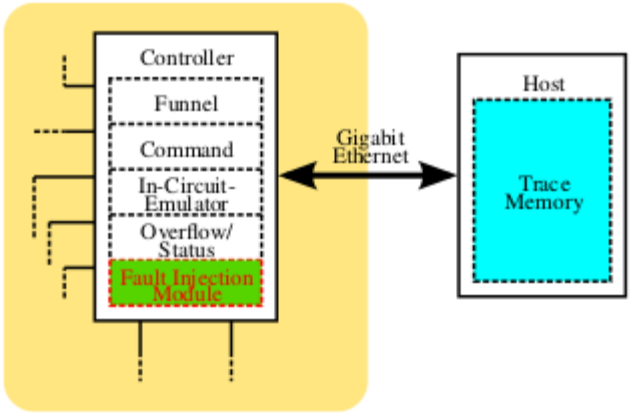
(a) Integration vor dem Flipflop

(b) Integration hinter dem Flipflop

Quelle: [2]

Fehlerinjektion nach Gunia[²]

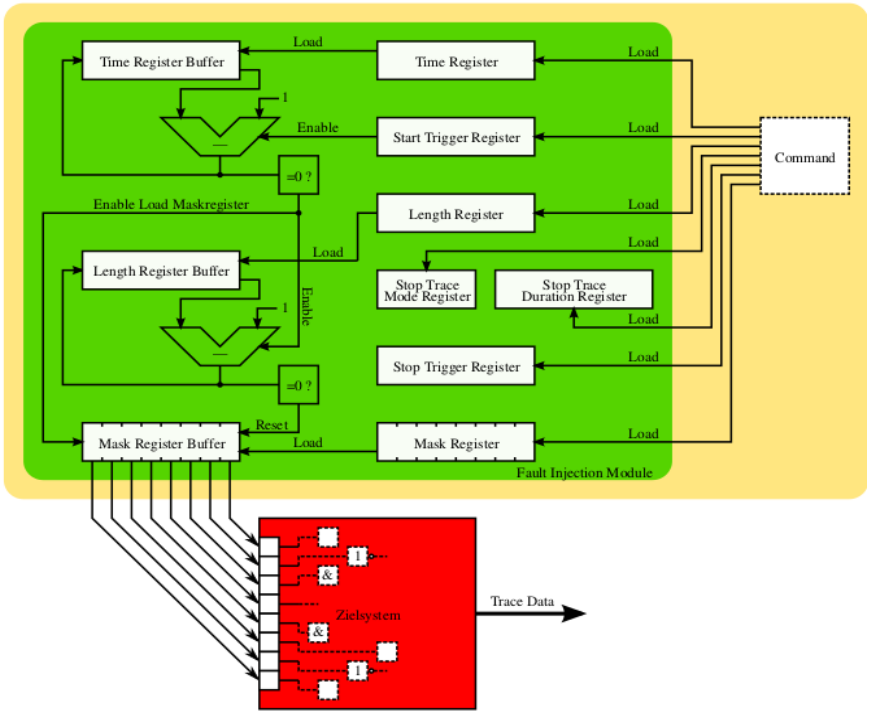
Vorhandene Architektur wird um Fault Injection Modul erweitert



Quelle: [2]

Fehlerinjektion nach Gunia[²]

Fault Injection Modul

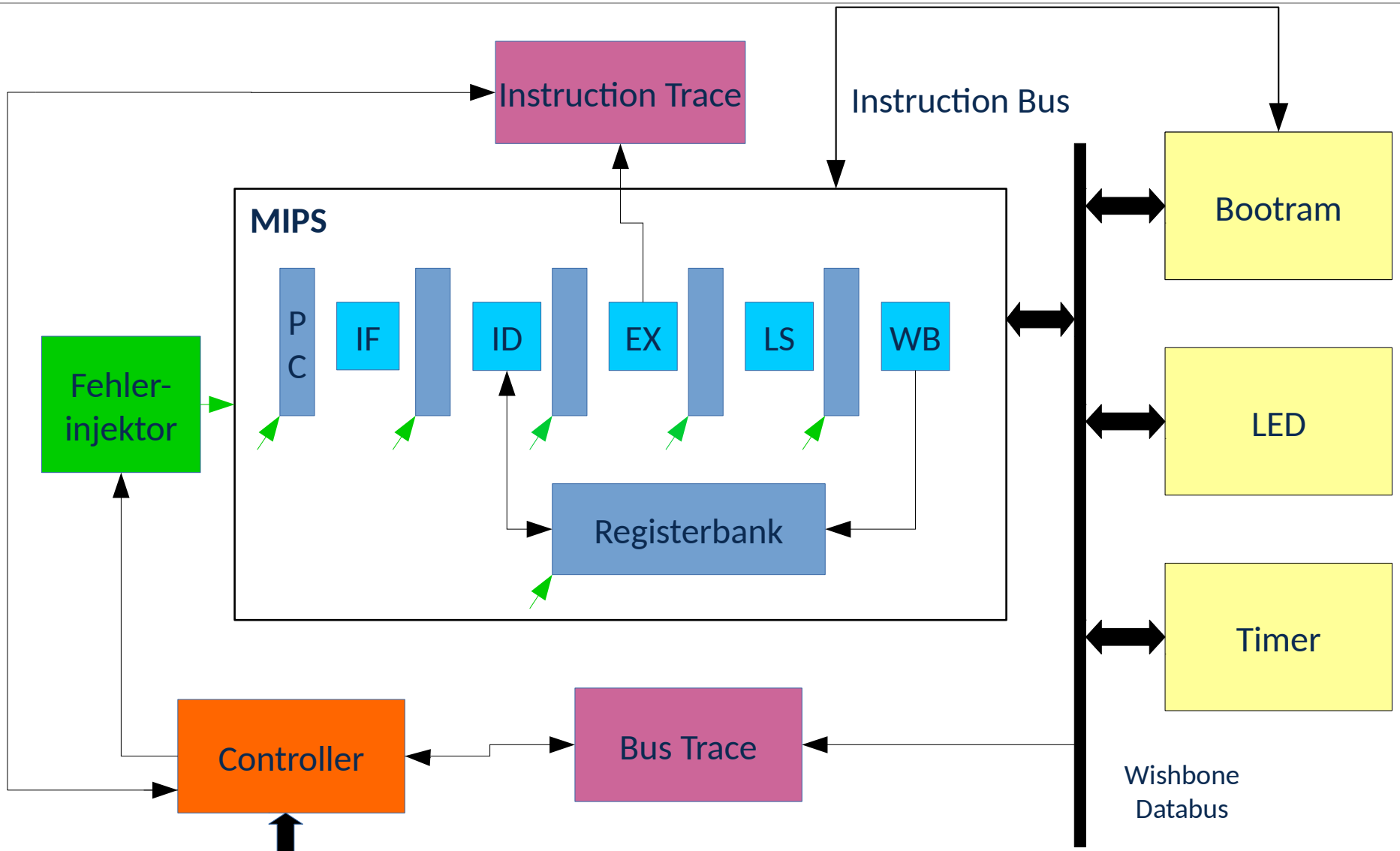


Quelle: [2]

Time Register:
Zeitpunkt der Injektion

Length Register:
(zeitliche) Länge der Injektion

Mask Register:
Ort der Injektion


Instruction Trace
Instruction Bus
MIPS
P C
IF
ID
EX
LS
WB
Registerbank
Fehler-injektor
Bootram
LED
Timer
Controller
Bus Trace
Wishbone Databus
Host

Vergleich der Fehlerinjektionen

Injektions-ansatz	nach Fidalgo[1]	nach Gunia[2]
Vorteile	• (theoretisch) auf jeder Nexus-Architektur anwendbar • Keine Änderung der HW nötig	• Sehr präzise Fehlerinjektion möglich • Injektionen in alle Bereiche möglich
Nachteile	• Hohe Latenz	• Hardwarebeschreibung muss änderbar sein

Testbedingungen

Testbedingungen

5 Verschiedene Fehlertypen

- Bit-Flip
- Flip-to-0
- Flip-to-1
- Stuck-at-0
- Stuck-at-1

Testbedingungen

Vorläufige Testprogramme

- Quicksort (viele Sprünge)
- Primzahlen-Berechnung (viele Lese/Speicherzugriffe)
- Fibonacci-Folge (viele Registerzugriffe)
- Knuth-Morris-Pratt-Algorithmus

Testbedingungen

Verschiedene Arten der Fehlerinjektion

- Injektion vor Flipflop
- Injektion nach FlipFlop

Testbedingungen

Verschiedene Trace-Einstellungen

- Mit/ohne Instruction-Trace
- Mit/ohne Daten-Trace

Nächste Schritte

- Bewertung der Testprogramme
- Auswertung der verschiedenen Testszenarien

Quellen

1) **Real-time fault injection using enhanced on-chip debug infrastructures** André V. Fidalgo, Manuel G. Gericota, Gustavo R. Alves, José M. Ferreira

2) **Erweiterung der Trace-Hardware eines Mikroprozessors für die Fehlerinjektion und Fehlerbeobachtung** Marco Gunia

Anhang

Fehlermodell	Schaltfunktion
stuck-at-0	$D_{FI} = (z_{cur}) \cdot \overline{CE} + (\overline{FIS} \cdot D) \cdot CE$
stuck-at-1	$D_{FI} = (z_{cur}) \cdot \overline{CE} + (FIS + D) \cdot CE$
Flipt-to-0	$D_{FI} = (\overline{FIS} \cdot z_{cur}) \cdot \overline{CE} + (\overline{FIS} \cdot D) \cdot CE$
Flip-to-1	$D_{FI} = (FIS + z_{cur}) \cdot \overline{CE} + (FIS + D) \cdot CE$
Bit-Flip	$D_{FI} = (FIS \hat{} z_{cur}) \cdot \overline{CE} + (FIS \hat{} D) \cdot CE$

Tabelle 5.2: Schaltfunktionen für die Integration vor dem Flipflop

Fehlermodell	Schaltfunktionen
stuck-at-0	$x_{FI} = \overline{FIS} \cdot z_{cur}$ und $D_{FI} = z_{cur} \cdot \overline{CE} + D \cdot CE$
stuck-at-1	$x_{FI} = FIS + z_{cur}$ und $D_{FI} = z_{cur} \cdot \overline{CE} + D \cdot CE$
Flipt-to-0	$x_{FI} = \overline{FIS} \cdot z_{cur}$ und $D_{FI} = z_{cur} \cdot \overline{CE} + D \cdot CE$
Flip-to-1	$x_{FI} = FIS + z_{cur}$ und $D_{FI} = z_{cur} \cdot \overline{CE} + D \cdot CE$
Bit-Flip	$x_{FI} = FIS \hat{} z_{cur}$ und $D_{FI} = z_{cur} \cdot \overline{CE} + D \cdot CE$

Tabelle 5.3: Schaltfunktionen für die Integration hinter dem Flipflop

Quelle: [2]