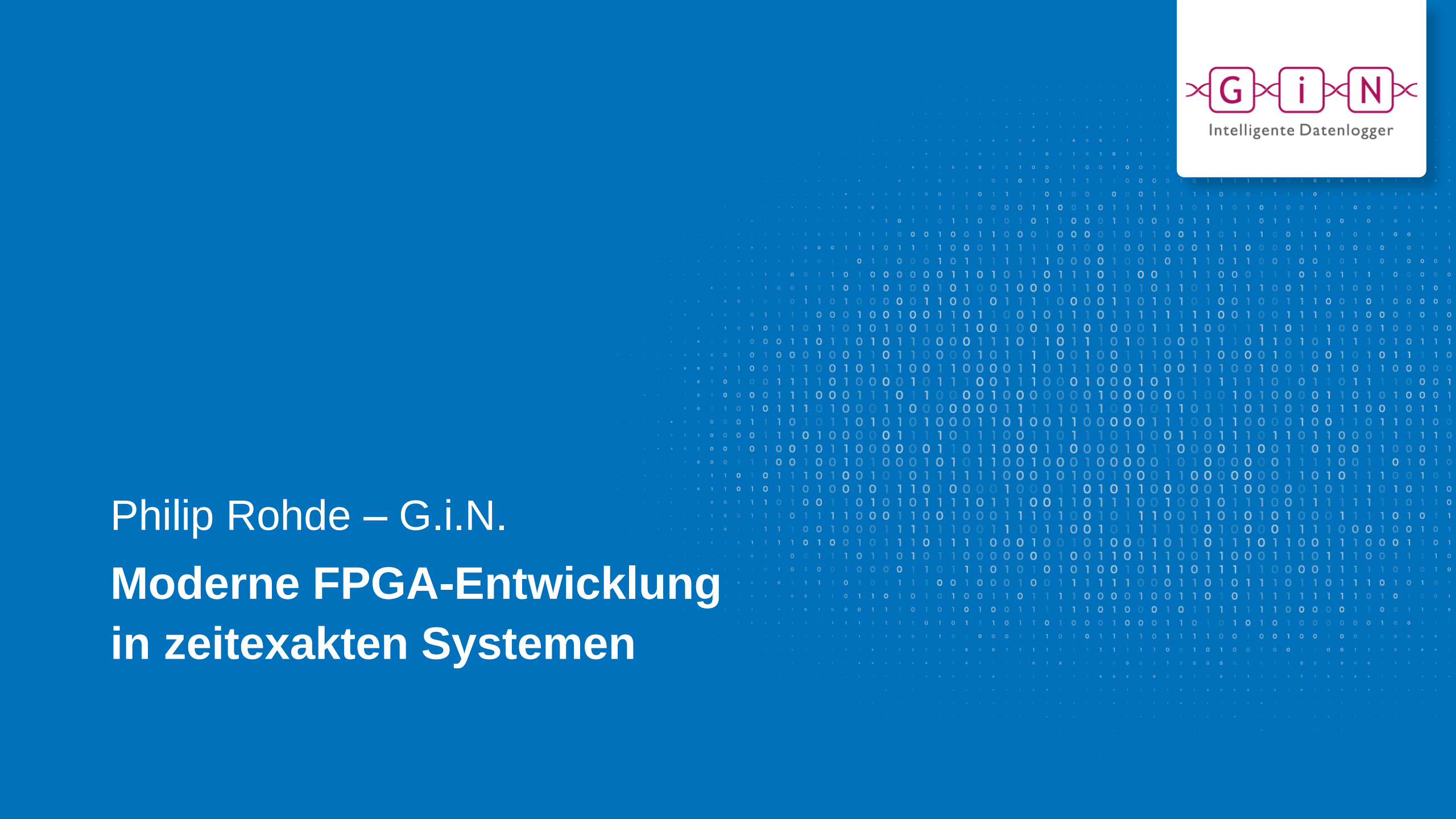
Intelligente Datenlogger

G.i.N. GmbH
Gesellschaft für industrielle Netzwerke

Industriekolloquium Datentechnik SoSe 2023

www.gin.de



Philip Rohde – G.i.N.

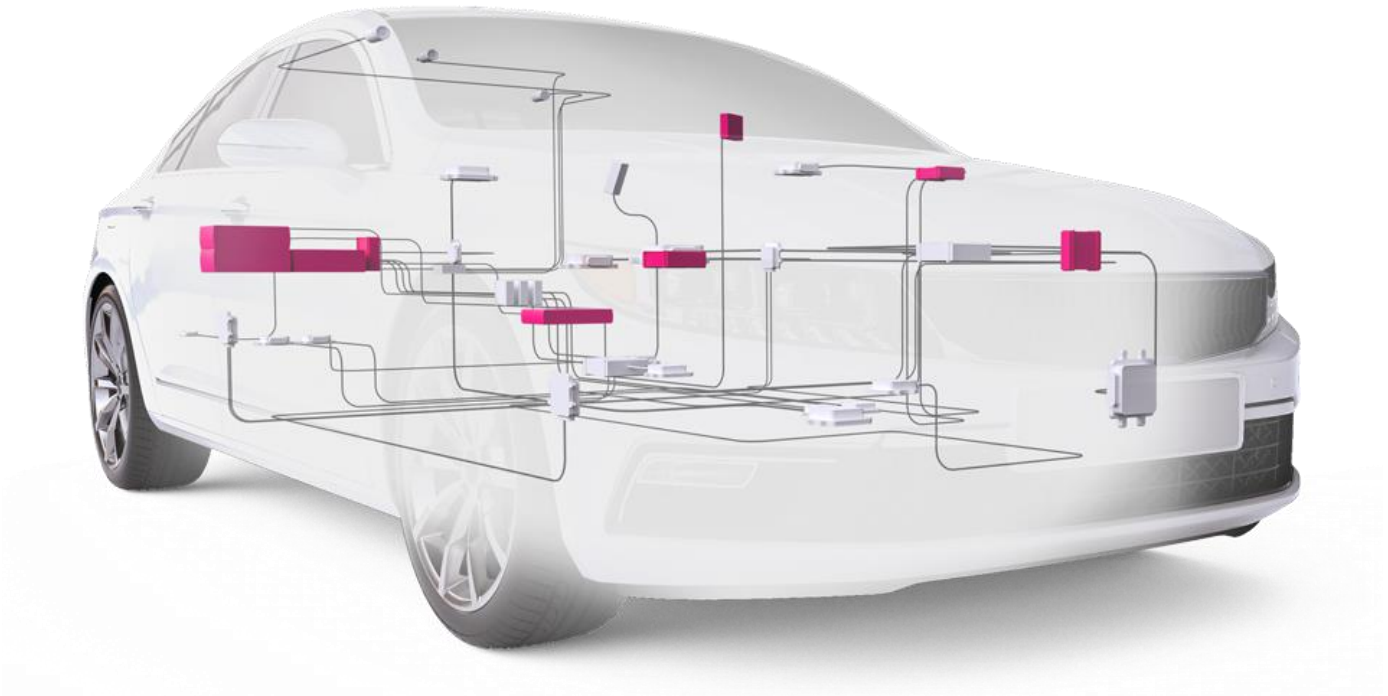
Moderne FPGA-Entwicklung in zeitexakten Systemen

Agenda

1. Was entwickelt G.i.N
2. Wofür setzen wir FPGAs ein
3. Herausforderungen in der Entwicklung
4. Entwicklungstools und -prozesse
5. Werbung!

Was entwickelt G.i.N?

Intelligente Datenlogger



Was entwickelt G.i.N?

- Datenlogger für
  - CAN / CAN-FD
  - LIN
  - MOST
  - RS232
  - FlexRay
  - Analogwerte
  - Automotive Ethernet (100 Mbit/s, 1 Gbit/s)
  - …

Intelligente Datenlogger

GL5450

- 20 x 100Mbit/s (10 Taps)
- 6 x 1Gbit/s (3 Taps)
- Theoretisch maximale Last 1GB/s
- Zwei SSDs mit 1TB
- 240MB/s kontinuierliche Aufzeichnung
- 2GB DDR für Lastspitzen

Was heißt intelligent?

Intelligente Datenlogger

LTL

```
CONST
   reply_msg = hCAFE[16]
VAR
   speed_raw  = CAN1 DATA 100 [3 2]
   speed_kmh = FREE[10]

CALC
   speed_kmh = (speed_raw / 128)

EVENT
   ON CYCLE (100) BEGIN
      TRANSMIT CAN1 DATA 200 [reply_msg] WHEN speed_kmh > 200
   END
```

Warum setzen wir FPGAs ein?

Viele Schnittstellen

Bspw. 24x CAN, 8x RS232, 2x LIN

Oder 20x 100Mbit/s + 6x 1Gbit/s

Hohe Genauigkeit

Zeitstempel im ns-Bereich

Hardware-Abstraktion

CAN, LIN, RS232, FlexRay

Zeitliche Sortierung von Daten

Reihenfolge der Nachrichten ist essentiell

Geringe Leistungsaufnahme

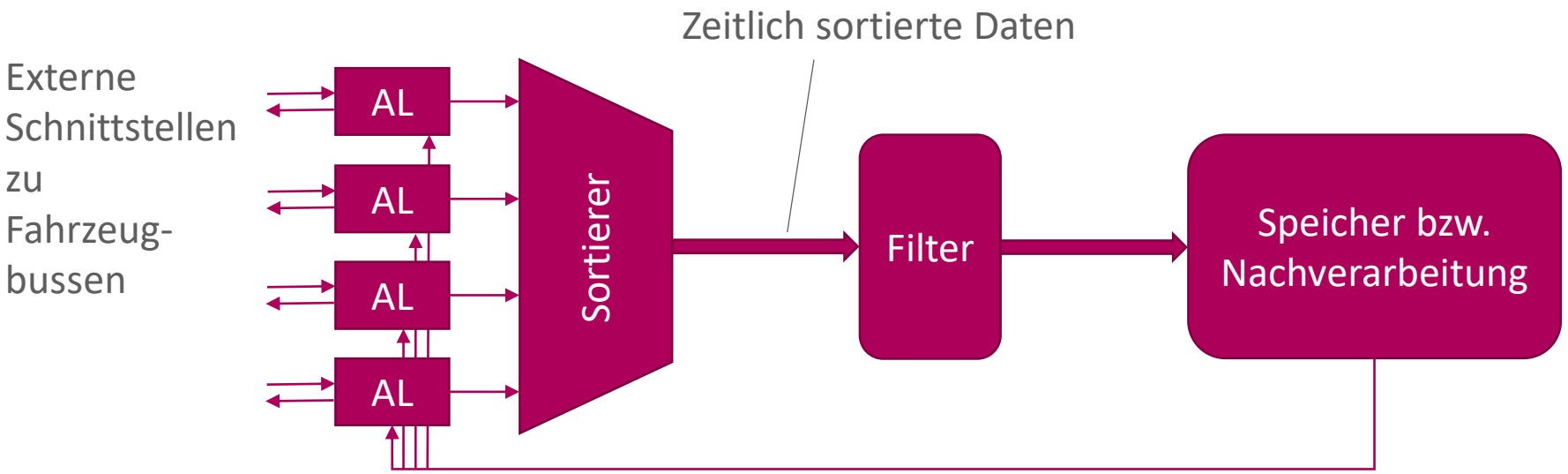
Halbschlaf mit Weckfähigkeit durch transceiver

Hohe Bandbreite trotz geringer Taktrate

Kurze Startzeit

Logging ab der ersten Nachricht

Hauptfunktionalität im FPGA



Sortieren? Klingt jetzt nicht so spannend…

Intelligente Datenlogger

- Bei 48 Bit-Zeitstempeln kostet jeder Vergleich Hardware und Zeit
- Bei 26 Eingangsströmen
  - … ist ein Modul, das alle parallel betrachtet nicht (sinnvoll) implementierbar
  - Was passiert mit Eingängen, an denen gerade keine Daten anliegen?
  - Durchsatz von 1GB/s bedeutet 64 Bit bei 125MHz
- Grenzbedingungen?
  - Overflow?
  - Kommt bei 48 Bit mit 64ns-Auflösung nur alle 208 Tage vor
  - Bei 32 Bit wäre es allerdings schon alle 4,5 Minuten

Herausforderungen

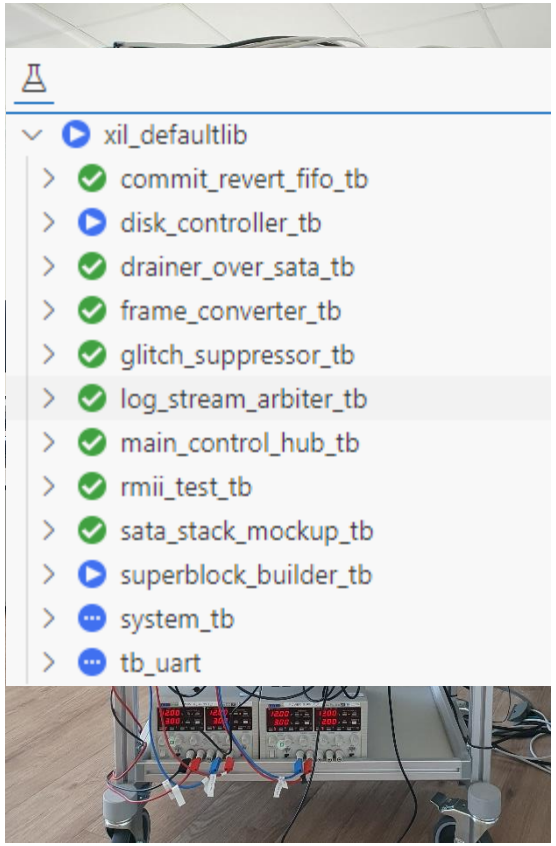
Intelligente Datenlogger

- HW-SW-Codesign
- Auswahl des FPGA bei Neuentwicklungen
- High-Speed Strecken und damit verbundene Timing-Anforderungen
  - On-Chip ist man in einer relativ heilen Welt
- Design Entscheidungen, deren Auswirkungen man nicht vollständig überblicken kann
  - Erweiterbarkeit vs. KISS
  - Infrastruktur (Bussystem, Schnittstellen, Projekttyp)
  - IP-Core nutzen oder selbst schreiben?
- Fehleranalyse
- Sicherstellen der korrekten Funktion

Wie stellen wir korrekte Funktion sicher?

- Whiteboxtests in der Entwicklung
  - Wir kennen die Randfälle
  - … die technischen Limitierungen
  - … die genaue Implementierung

- Blackboxtests im Testcenter
  - Datenstimulierung am Gerät
  - Ausgabe gegen erwartetes Ergebnis prüfen
  - Verschiedene Funktionalitäten/Betriebsmodi des Geräts



Testen in Hardware

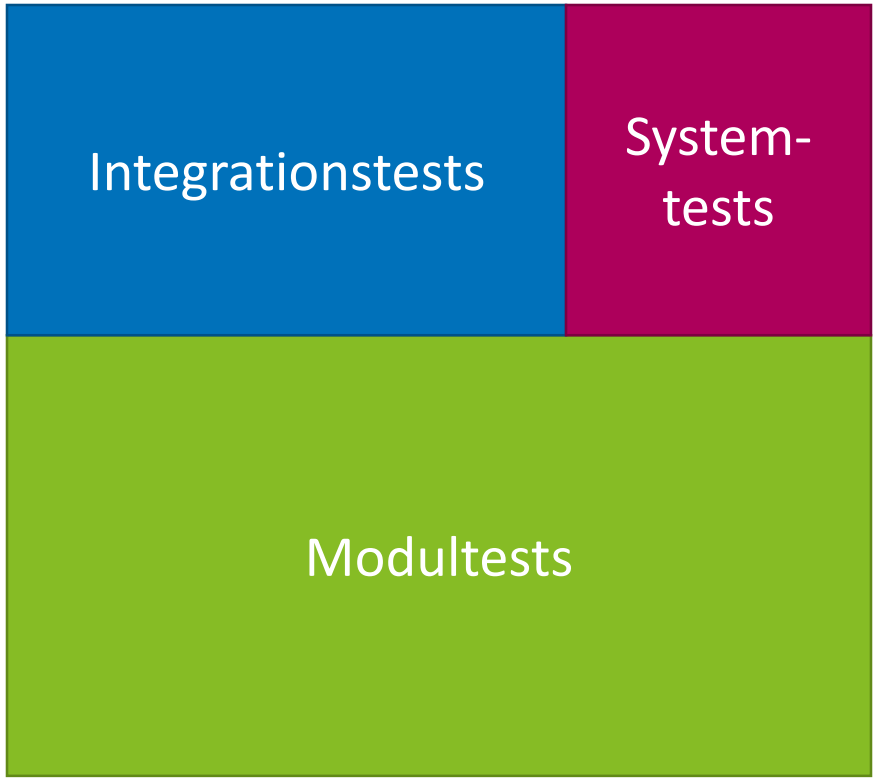
- Synthesezeiten von ca. bis zu 4 Stunden
  - De-Facto Coding-Pause
  - Ein Fehler → nochmal 4h
  - Fehler werden erst bei Instrumentierung sichtbar
- Randfälle sind nicht gut kontrollierbar
- Nur geringe Testabdeckung möglich
- Fehlerfälle nicht immer nachstellbar (z.B. defekte Hardware)
- Manche Funktionen nur in Kombination mit angepasster Firmware möglich
- Vorteil: Bei echter Hardware fallen Timing- und Initialisierungsfehler schnell auf

Testen in Simulation

Intelligente Datenlogger

- Hardware muss nicht mal verfügbar sein
- Erfordert Simulationsmodelle externer Komponenten
  - ADC, CAN, MII, DDR, SSD
  - Benötigt Entwicklungszeit, die sich aber auf mehrere Projekte aufteilt
- Für jedes Modul/Komponente lassen sich Tests schreiben
  - Kleine Module lassen sich schnell umfangreich testen
  - Randfälle können gezielt angesteuert werden
  - Fehler direkt in Simulator sichtbar
  - Regressionstests durch self-checking Testbenches!
- Nachteil: Timing-Probleme fallen komplett unter den Tisch
- Tests sind nur so gut, wie sein Autor und die Modelle
- Systemtests dauern extrem lang

Testen in Simulation



Wie gehen wir es an

Intelligente Datenlogger

- Simulationsmodelle
  - Über die Zeit weiter verfeinert
  - Hoher Grad an Wiederverwendbarkeit
  - Nutzung zum deterministischen Nachstellen von Fehlern

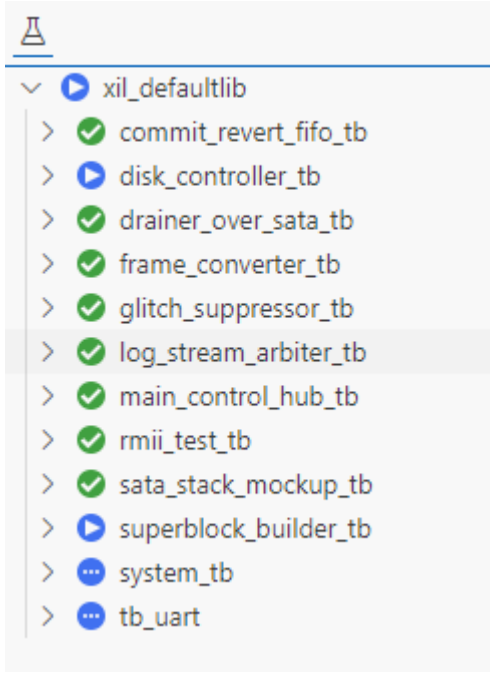
- Software-Entwicklungsmethodik
  - Coding-Style
  - Source code in Git
  - Reviews für Änderungen (Pull-Request)
  - Test-Framework (VUnit)
  - Regressionstests auf Build-Server in Docker-Containern

- In dem Moment, wo Legacy-Code modifiziert wird, werden Testbenches gefordert

Test-Framework

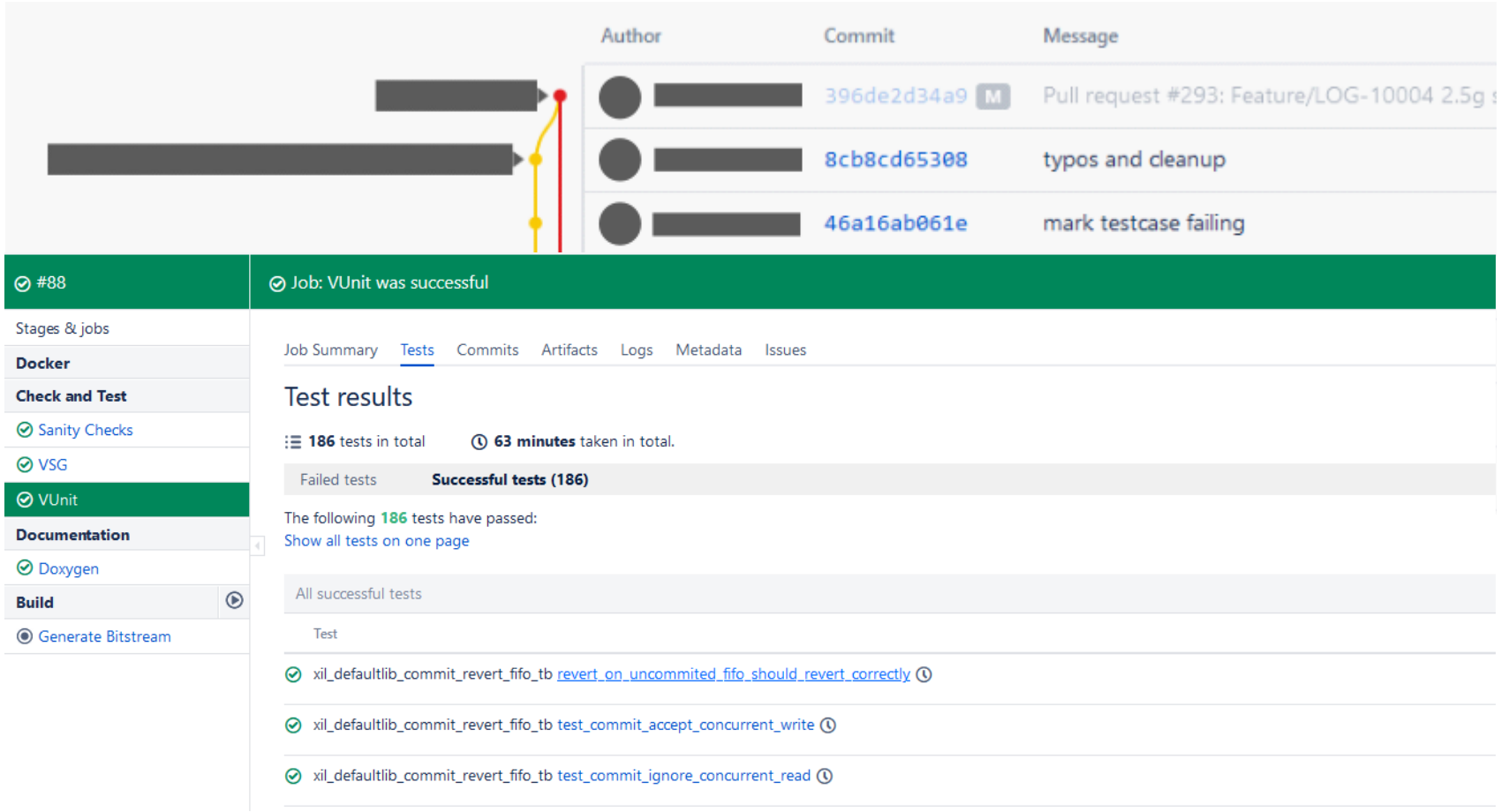
Intelligente Datenlogger

```
Run | Debug | Show in Test Explorer
elsif run("read_write_regs_should_return_written_value") then
    -- check for every rw-register that a written value can also be read
    helper_data := rnd.RandSlv(helper_data'length);
    write_bus(NET, BUS_HANDLE, REG_INTERRUPT_MASK, helper_data);
    check_bus(NET, BUS_HANDLE, REG_INTERRUPT_MASK, helper_data);

    helper_data := rnd.RandSlv(helper_data'length);
    write_bus(NET, BUS_HANDLE, REG_INTERRUPT_STATUS, helper_data);
    check_bus(NET, BUS_HANDLE, REG_INTERRUPT_STATUS, std_logic_vector'(32x"00000000"));

    helper_data := rnd.RandSlv(helper_data'length);
    write_bus(NET, BUS_HANDLE, REG_READ_ERROR_COUNT, helper_data);
    check_bus(NET, BUS_HANDLE, REG_READ_ERROR_COUNT, helper_data);

    helper_data := rnd.RandSlv(helper_data'length);
    write_bus(NET, BUS_HANDLE, REG_WRITE_ERROR_COUNT, helper_data);
    check_bus(NET, BUS_HANDLE, REG_WRITE_ERROR_COUNT, helper_data);

    helper_data := rnd.RandSlv(helper_data'length);
    write_bus(NET, BUS_HANDLE, REG_READ_ERROR_THRESHOLD, helper_data);
    check_bus(NET, BUS_HANDLE, REG_READ_ERROR_THRESHOLD, helper_data);

    helper_data := rnd.RandSlv(helper_data'length);
    write_bus(NET, BUS_HANDLE, REG_WRITE_ERROR_THRESHOLD, helper_data);
    check_bus(NET, BUS_HANDLE, REG_WRITE_ERROR_THRESHOLD, helper_data);
```



Regressionstests

Intelligente Datenlogger



Größte Probleme in Realität

- Alter Legacy-Code ohne Tests
  - Sehr schwer zu erfassen (Benamung, State-Diagramm, Parallelität)
  - Erweiterung ohne alte Funktion zu beeinflussen
  - Definitionslücken
- Es können zwei oder auch mal vier Wochen vergehen, bis ein Fehler nachstellbar ist
  - Ist das Gerät einfach defekt oder liegt ein systematisches Problem vor?
  - Umgebungsprobleme?
  - Unklar, wer der Verursacher ist (Firmware, FPGA, Hardware)
  - Bei der Untersuchung verschwindet der Fehler
  - Manche Fehler treten nur bei sehr kalten oder sehr hohen Temperaturen auf
- Zugelieferte Arbeit erfordert Reverse Engineering
- Gute Entwickler finden

Was macht gute FPGA-Entwickler aus?

Intelligente Datenlogger

- Tellerrand
  - Tooling abseits von HDL (TCL, Bash, Python, Git, Docker)
  - Datenblätter/User-Guides lesen und interpretieren
  - Gutes Verständnis in beide Richtungen: Firmware und Hardware
- Frustrationstoleranz / hoher Pain-Threshold
- Erarbeitung eines eigenen Workflows
- Hohes Maß an Analysefähigkeit
- Motivation und Spaß daran, neues zu Lernen und auszuprobieren!

Noch Fragen?


GL2400
CAN FD Data Logger
GL5450
GL5370
GL3400
CAN FD Data Logger
Powerbank GLA710
GLA618

Klingt ganz spannend?