# Automatisches Bauen und Parallelisieren von Microblaze Systemen


**Master Thesis**

**Christian Hohenbrink**

Betreuer: Kris Heid

Datum: 23.01.2017

TU Darmstadt
Fachgebiet Rechnersysteme
Prof. Dr.-Ing. Christian Hochberger

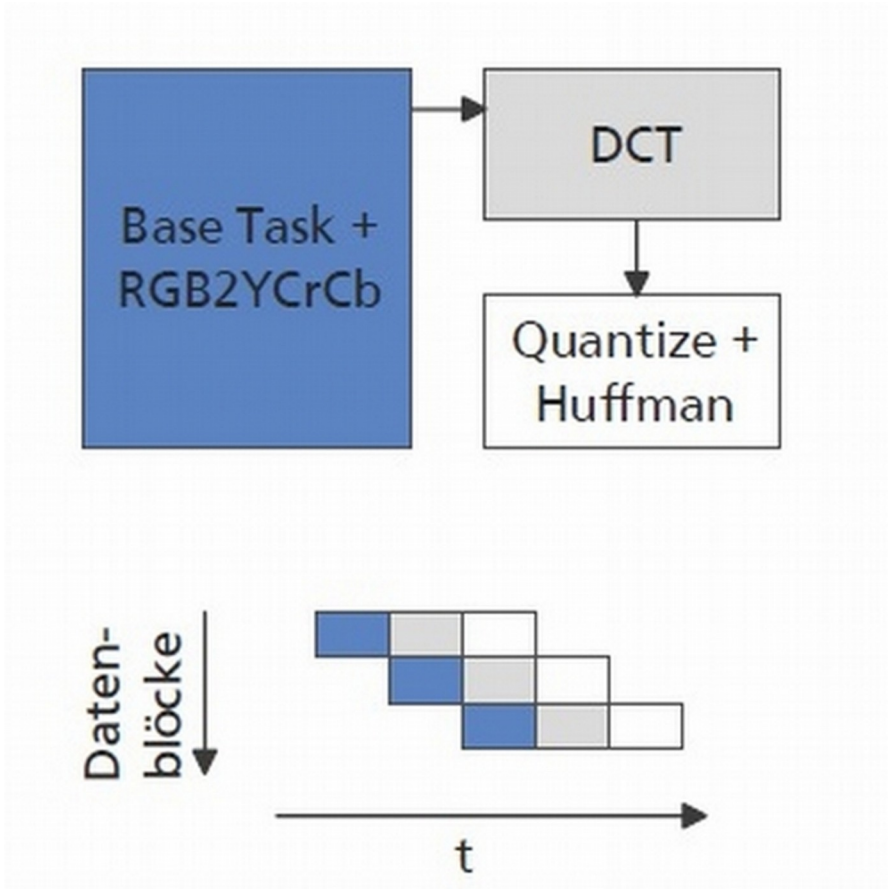
# Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motiviation
- Ziel der Arbeit
- Grundlagen
- Konzept und Implementierung
- Evaluation
- Fazit und Ausblick

# Motiviation

- Im Fachbereich Rechnersysteme wird an Manycore SoC geforscht
- Steigerung der Performance
- Ein Ansatz stellt das Projekt µStreams dar
- Source-To-Source Compiler mit Task Pipelining
- µStreams nur mit SpartanMC → Prinzip auch auf andere Prozessoren anwendbar



# Ziel der Arbeit

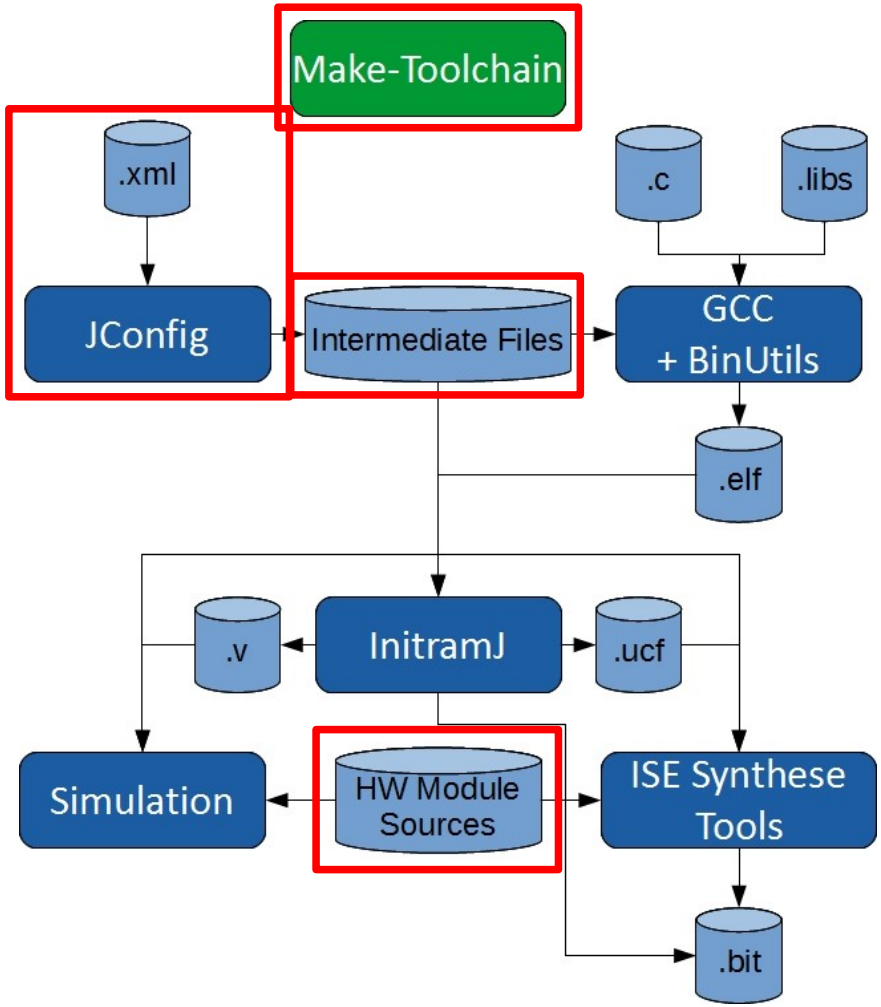
TECHNISCHE UNIVERSITÄT DARMSTADT

- Xilinx Microblaze in die SpartanMC Entwicklungsumgebung integrieren
- Außerdem:
  - UART als Peripherie
  - Fast Simplex Link (FSL) Bus  als Pendant zu den Core Konnektoren
- Die Toolchain gegebenenfalls anpassen
- Optional: Erfolgreiche Parallelisierung eines Microblaze Programms mit µStreams zeigen

# Grundlagen – Die SpartanMC Entwicklungsumgebung

TECHNISCHE UNIVERSITÄT DARMSTADT

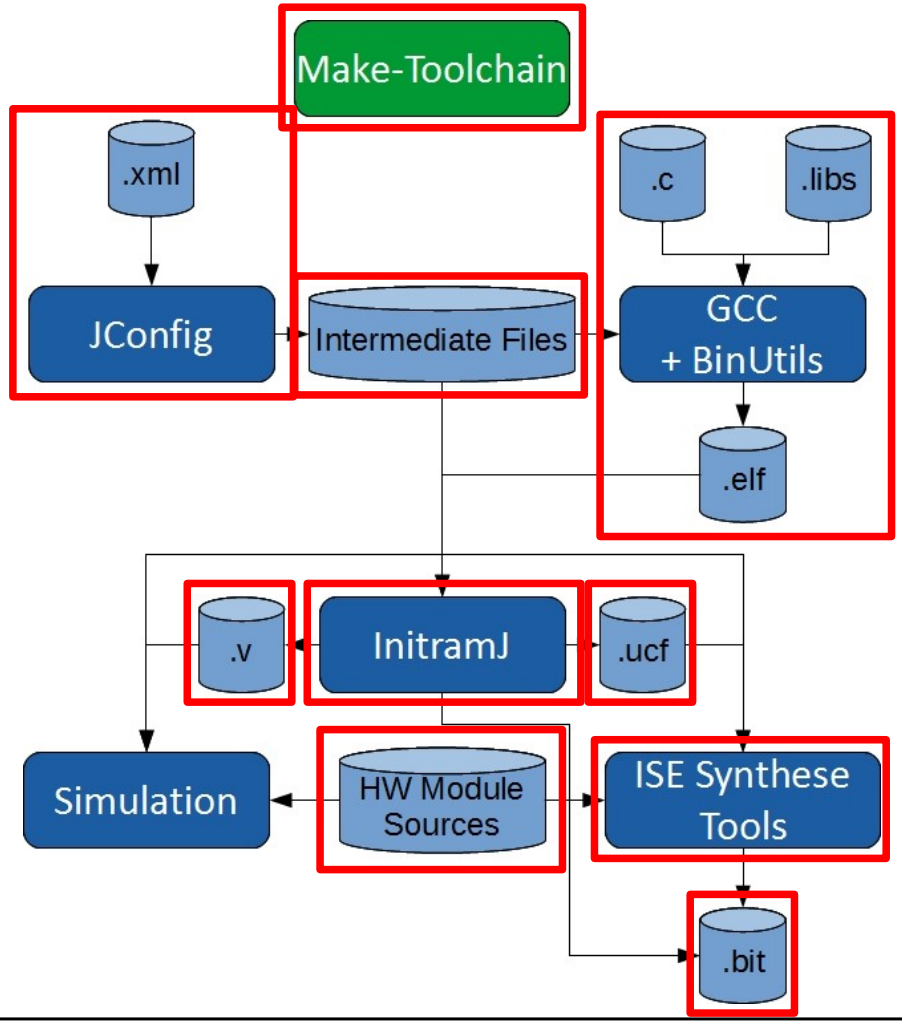
- Projektmanagement und Steuerung des Workflow durch Makefiles
- Systembuilder JConfig
- Intermediate Files: Sammlung von Dateien, die von der Make-Toolchain benötigt werden.



# Grundlagen – Die SpartanMC Entwicklungsumgebung

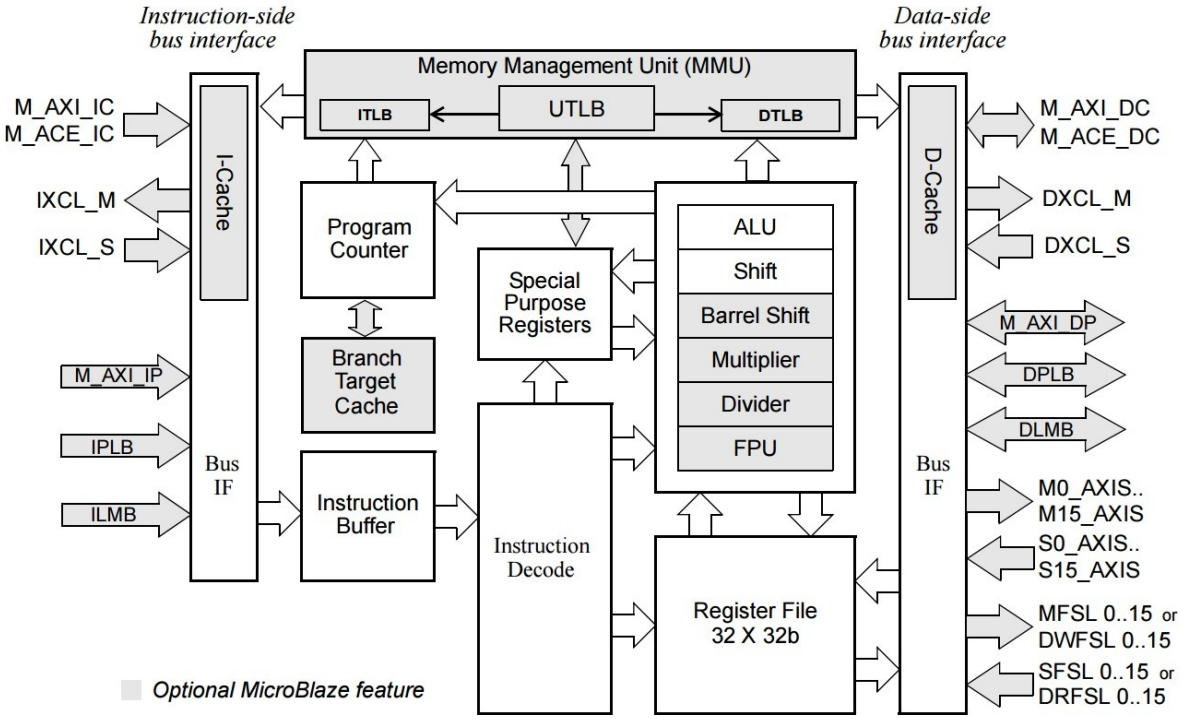
TECHNISCHE UNIVERSITÄT DARMSTADT

- Software in C
- Synthese Tools aus der ISE Design Suite
- InitramJ für die Speicherinitialisierung
  - Synthese: UCF-Datei
  - Simulation: Verilog-Datei
  - Aktualisierung des Bitfiles



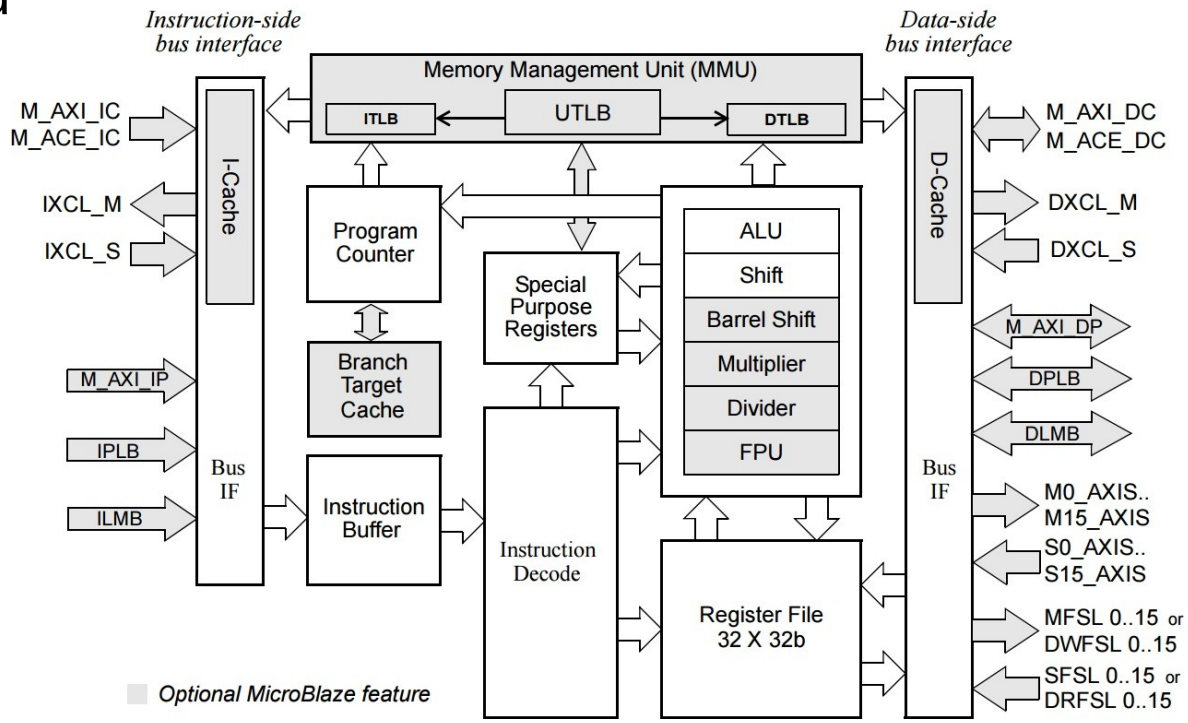
# Grundlagen – Der Microblaze

- 32-Bit RISC-Prozessor
- Feste Eigenschaften:
  - 32 32-Bit General Purpose Register
  - 32-Bit Instruktionen mit drei Operanden
  - Zwei Adress-Modi
  - 32-Bit Adressbus
  - Singel-Issue Pipeline
- ~70 Parameter zur freien Konfiguration

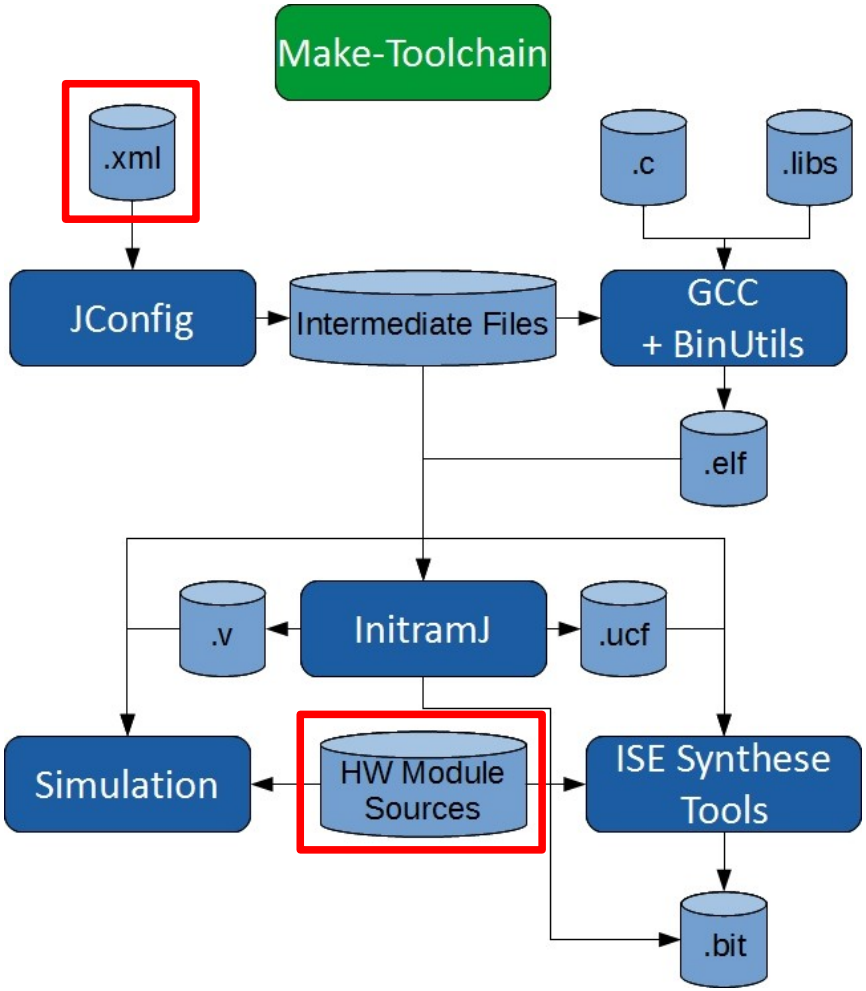


# Grundlagen – Der Microblaze

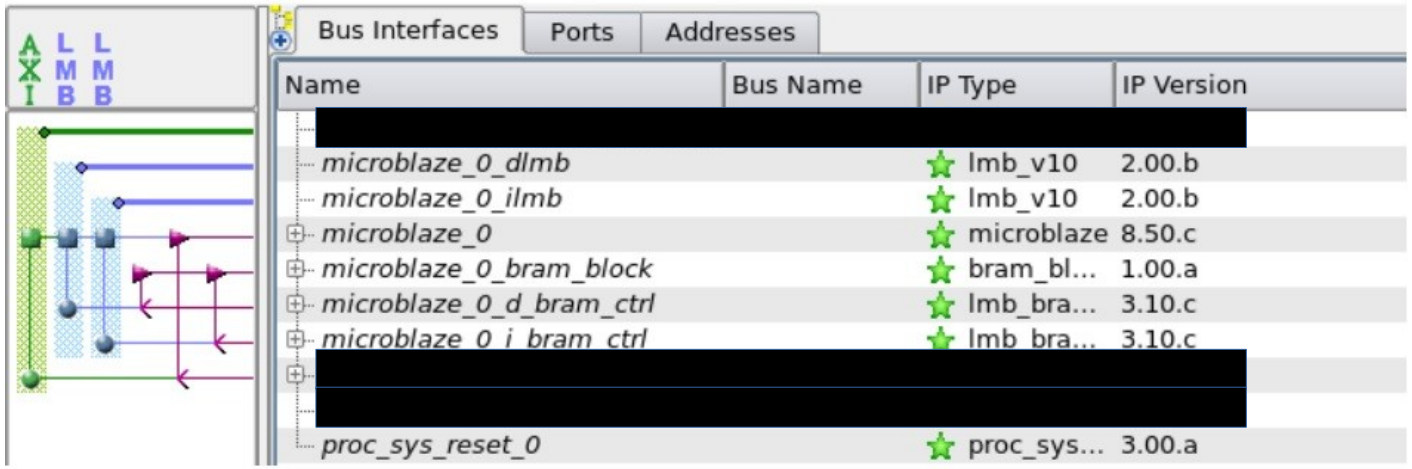
- Microblaze Features:
  - Hardwarebeschleuniger
  - MMU
  - Caches
  - Diverse Busse
  - Hardware Debugging
  - Streaming Interfaces
  - Branch Target Cache



# Konzept und Implementierung – HDL-Dateien und XML-Modulbeschreibungen



# Konzept und Implementierung – HDL-Dateien

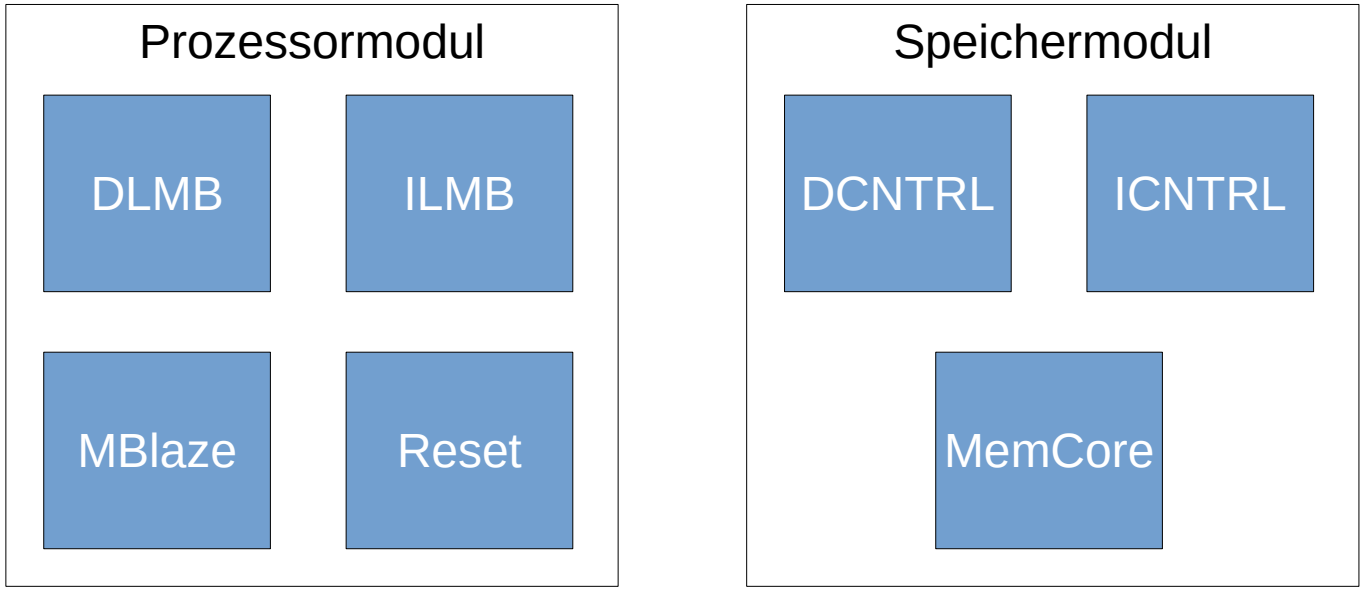



- Komplexität des Systems wird verringert:
  - AXI4-Bus, Clock-Modul und Debug-Modul werden entfernt
- XPS kann Wrapper-Dateien erzeugen
- Wrapper-Dateien haben keine Parameter im Interface
- Prozessor und Speichermodul werden getrennt

# Konzept und Implementierung – HDL-Dateien

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aufteilung in Prozessor- und Speichermodul



# Konzept und Implementierung – XML-Modulbeschreibungen

TECHNISCHE UNIVERSITÄT DARMSTADT

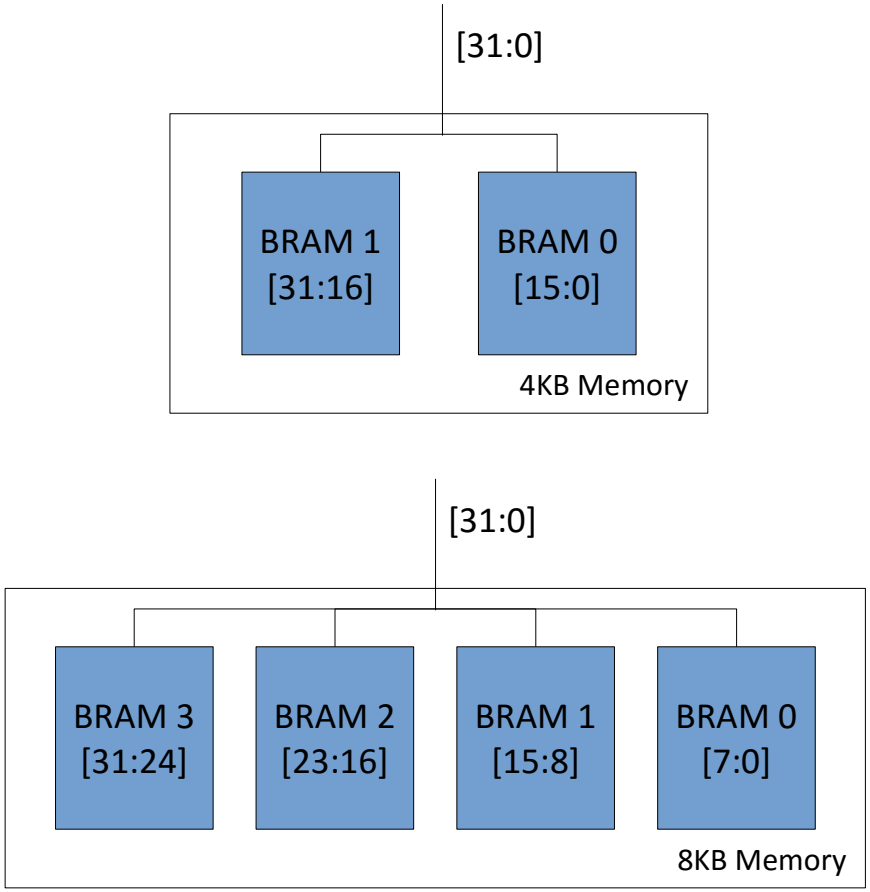
- Bisher nur Wrapper-Dateien
- Implementationen im ISE-Verzeichnis zu finden
- JConfig hat keine Möglichkeit externe Pfade anzugeben
- Lösung → ExternalRoot-Scope

```
<hdl lang="vhdl" toplevel="fast_simplex_link">
    <sources>
        <file>fast_simplex_link.vhd</file>
        <externalRoot path="$XILINX_ROOT/ise/14.7/ISE_DS/EDK/hw/XilinxProcessorIPLib/pcores/fsl_v20_v2_11_f/hdl/vhdl/" library="fsl_v20_v2_11_f">
            <file>async_fifo.vhd</file>
            <file>async_fifo_bram.vhd</file>
            <file>fsl_v20.vhd</file>
            <file>gen_srlfifo.vhd</file>
            <file>gen_sync_bram.vhd</file>
            <file>gen_sync_dpram.vhd</file>
            <file>sync_fifo.vhd</file>
        </externalRoot>
    </sources>
</hdl>
```

# Konzept und Implementierung – Speicherimplementierung

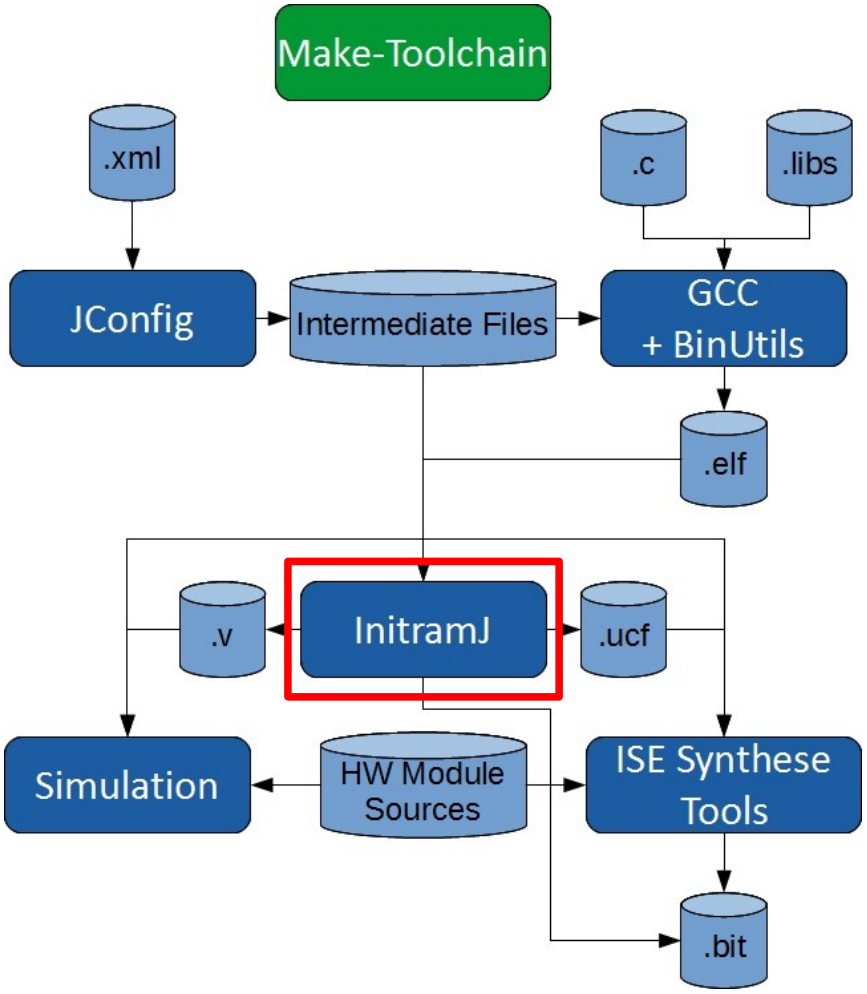
TECHNISCHE UNIVERSITÄT DARMSTADT

## Microblaze Speicher



# Konzept und Implementierung – Speicherinitialisierung

- Unterschiedliche Arten der Speicherinitialisierung mit InitramJ:
  - Synthese: UCF-Datei
  - Simulation: Verilog-Datei
  - Aktualisierung des Bitfiles
- Zwei Möglichkeiten:
  - InitramJ anpassen
  - data2mem verwenden
- → data2mem



# Konzept und Implementierung – Speicherinitialisierung

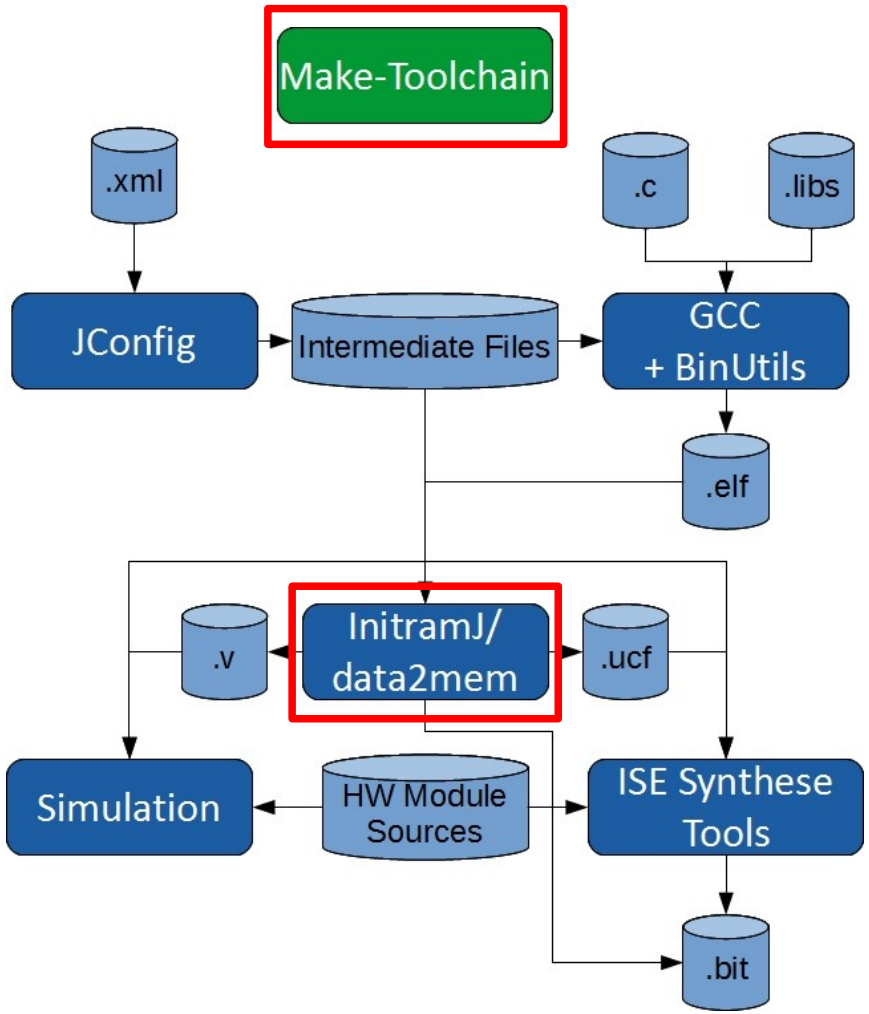
TECHNISCHE UNIVERSITÄT DARMSTADT

- InitramJ für SpartanMC und data2mem für Microblaze
- BMM-Datei als neue Memory-Map für Microblaze
- Neuer Output-Provider in JConfig
- Flags um Prozessorarten zu unterscheiden

```
ADDRESS_MAP microblaze_spmc_0 MICROBLAZE-LE 1
   ADDRESS_SPACE microblaze_spmc_0_bram_block_combined COMBINED [0x0:0x1fff]
      ADDRESS_RANGE RAMB16
      BUS_BLOCK
         microblaze_spmc_0_microblaze_mem_local_0/mem_core/MEM_BL[0].DPRAM [31:24];
         microblaze_spmc_0_microblaze_mem_local_0/mem_core/MEM_BL[1].DPRAM [23:16];
         microblaze_spmc_0_microblaze_mem_local_0/mem_core/MEM_BL[2].DPRAM [15:8];
         microblaze_spmc_0_microblaze_mem_local_0/mem_core/MEM_BL[3].DPRAM [7:0];
      END_BUS_BLOCK;
      END_ADDRESS_RANGE;
   END_ADDRESS_SPACE;
END_ADDRESS_MAP;
```

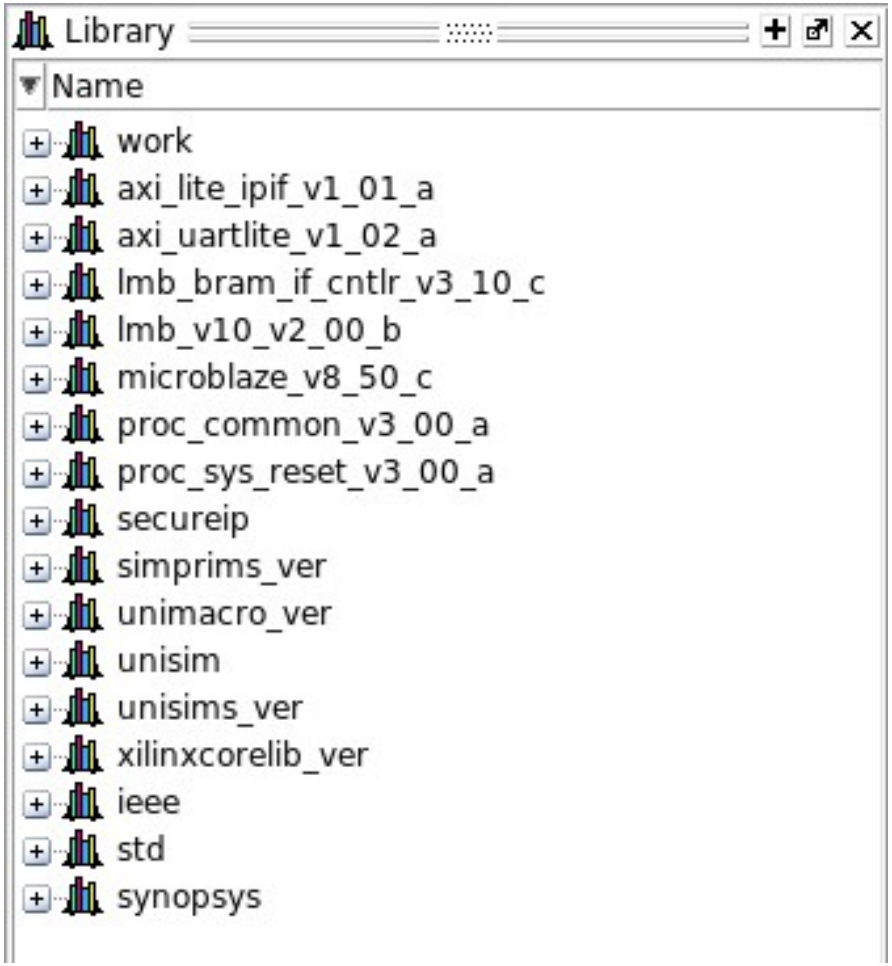
# Konzept und Implementierung – Speicherinitialisierung

- Anpassungen der Makefiles
- Neue Regeln die data2mem aufrufen
- Nur ein Speicher pro Aufruf von data2mem
- Skript merged UCF-Dateien
- Neues Skript für das Mergen der Verilog-Dateien



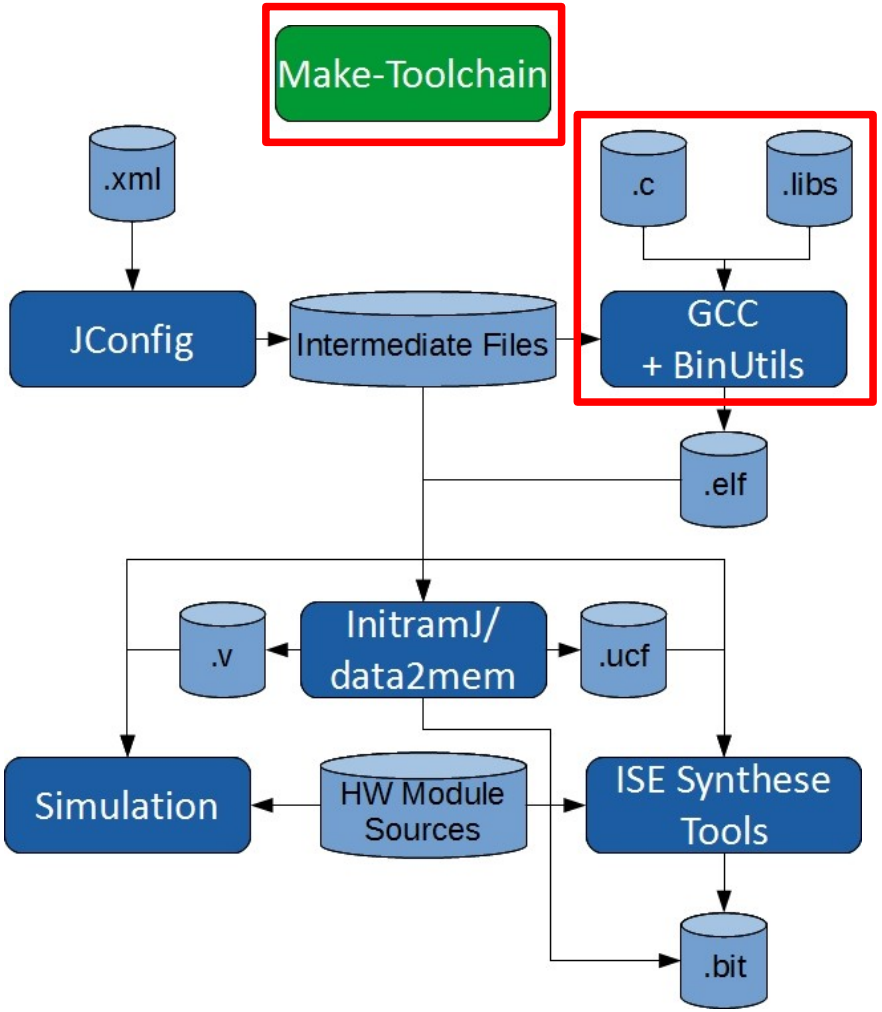
# Konzept und Implementierung – Simulations-Libraries

- Xilinx Tool compxlib zum Erstellen von vorkompilierten Libraries
- Notwendig für verschlüsselten Microblaze
- Tcl-Skript erweitern, welches die Simulations-Libraries definiert
- Pfad zu den Libraries muss in JConfig angegeben werden

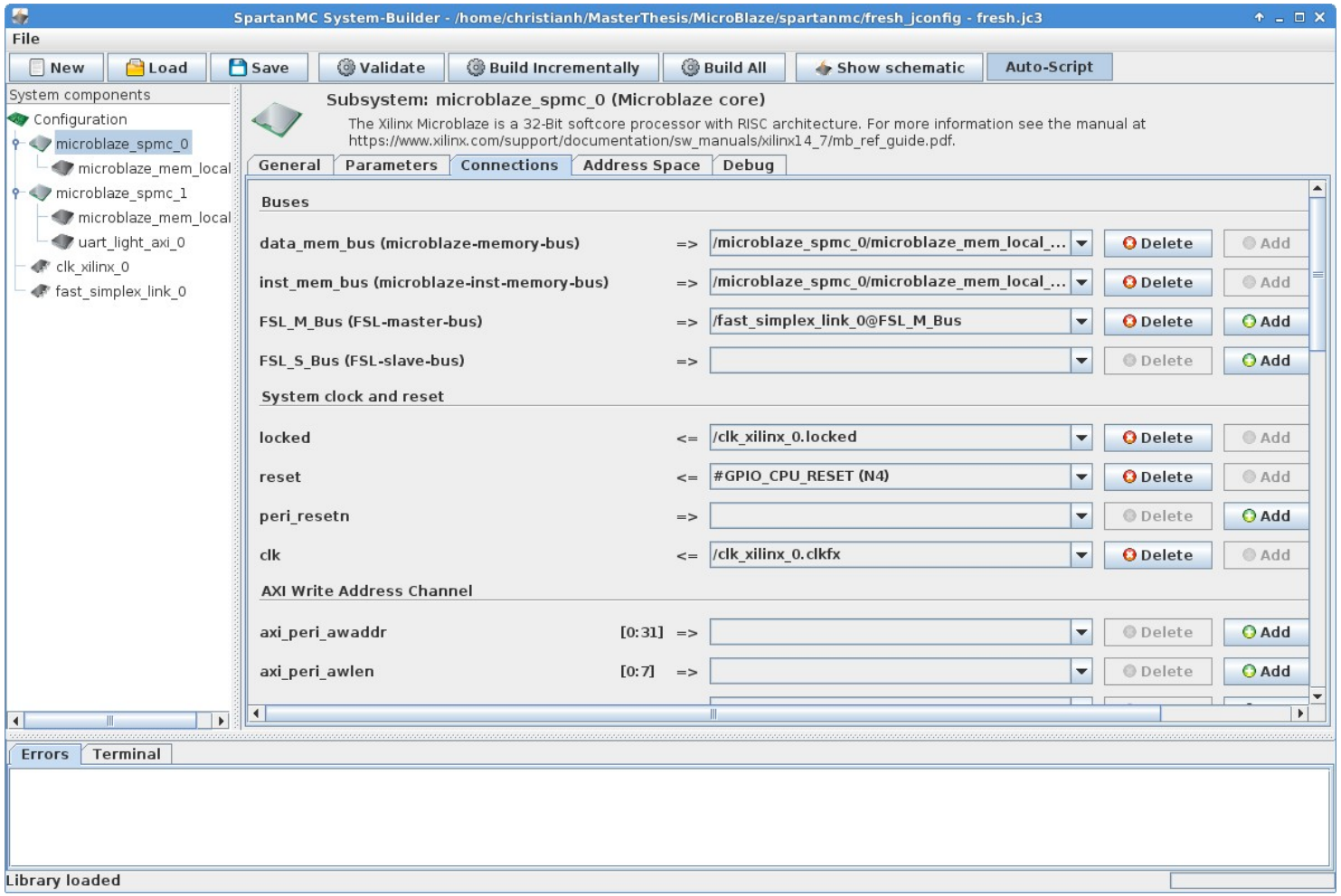


# Konzept und Implementierung – Microblaze GCC

- Microblaze GCC notwendig um kompatible ELF-Dateien zu erzeugen
- wurde nicht integriert
- Hinweise zur möglichen Umsetzung in der Ausarbeitung



# Evaluation – Testsystem



# Evaluation – Testssoftware

TECHNISCHE UNIVERSITÄT DARMSTADT

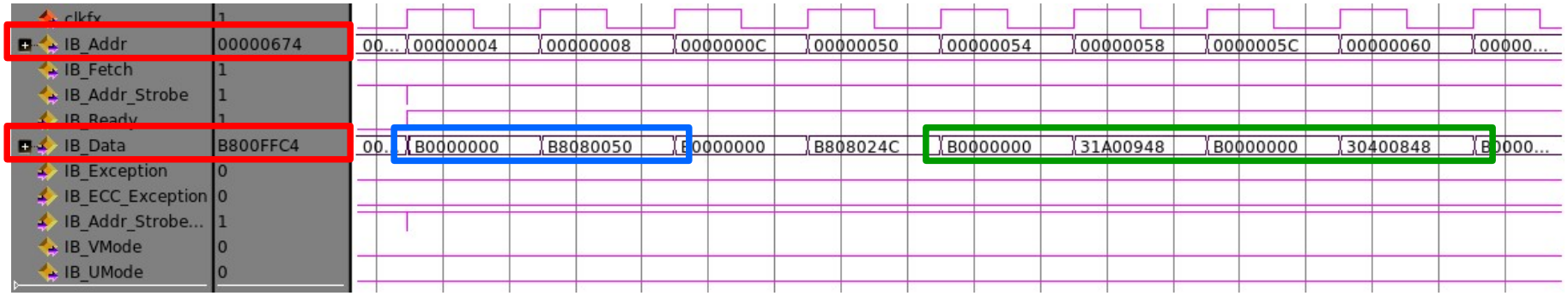
- Software Prozessor 1

```
int main()
{
   int i = 0;
    init_platform();

    char* string = "Hello World!\n\r";
    while(1){
      for (i=0;i<13;i++){
         putfslx(string[i],0,FSL_NONBLOCKING);
      }
    }
    return 0;
}
```

- Software Prozessor 2

```
int main()
{
   int i = 0;
    init_platform();

    char* string = "";
    while(1){
      for (i=0;i<13;i++){
         getfslx(string[i],0,FSL_DEFAULT);
      }
      print(string);
    }
    return 0;
}
```

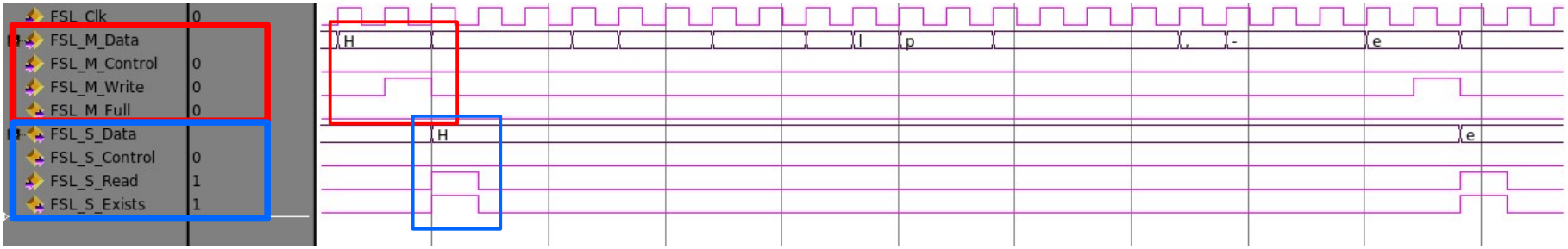
# Evaluation – Simulation des Instructionbuffers




```
void
disable_caches()
{
   0:   b0000000    imm 0
   4:   b8080050    brai    80  // 50 <_start1>
```

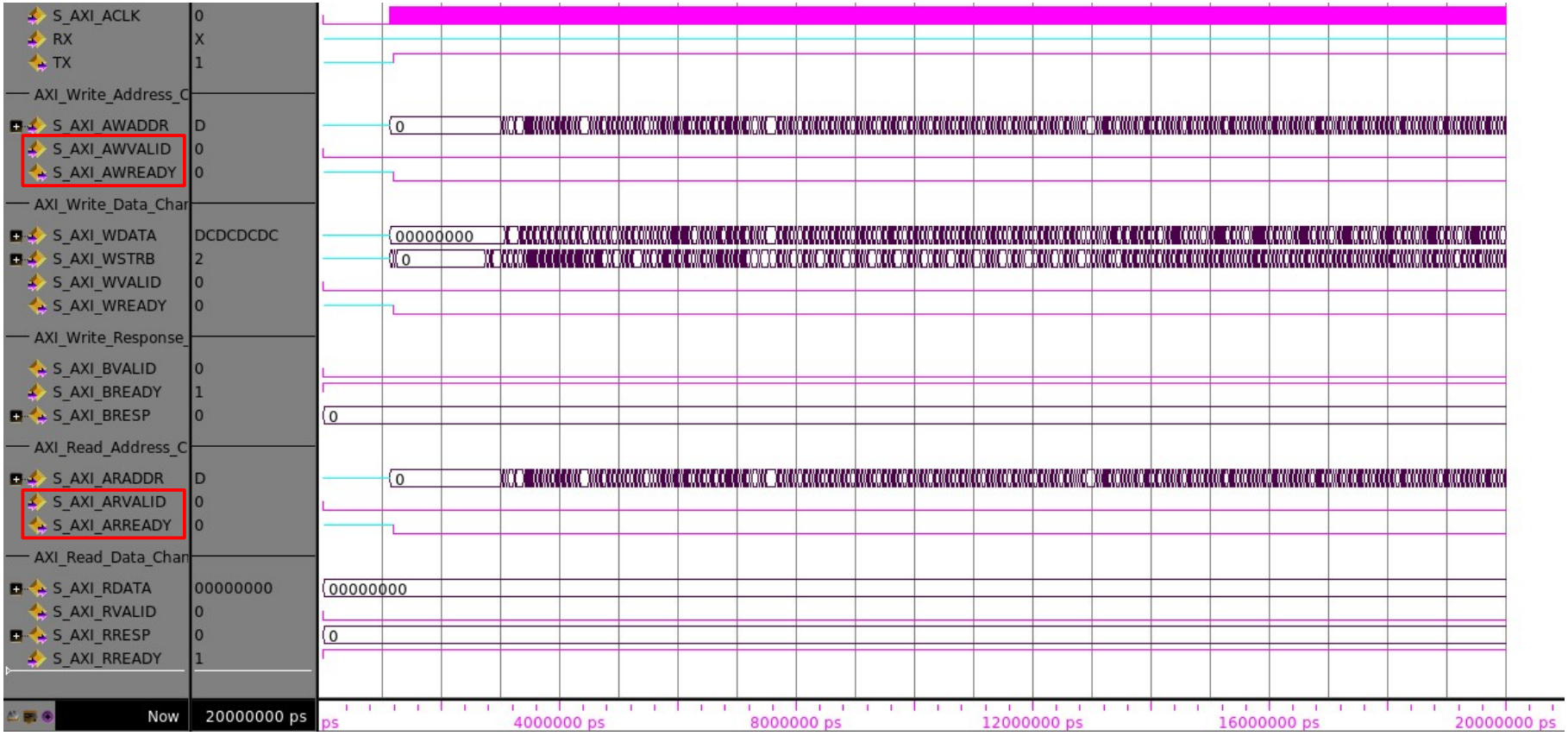
```
00000050 <_start1>:
  50:   b0000000    imm 0
  54:   31a00948    addik   r13, r0, 2376   // 948 <_SDA_BASE_>
  58:   b0000000    imm 0
  5c:   30400848    addik   r2, r0, 2120    // 848 <_SDA2_BASE_>
```

# Evaluation – Simulation einer FSL-Übertragung

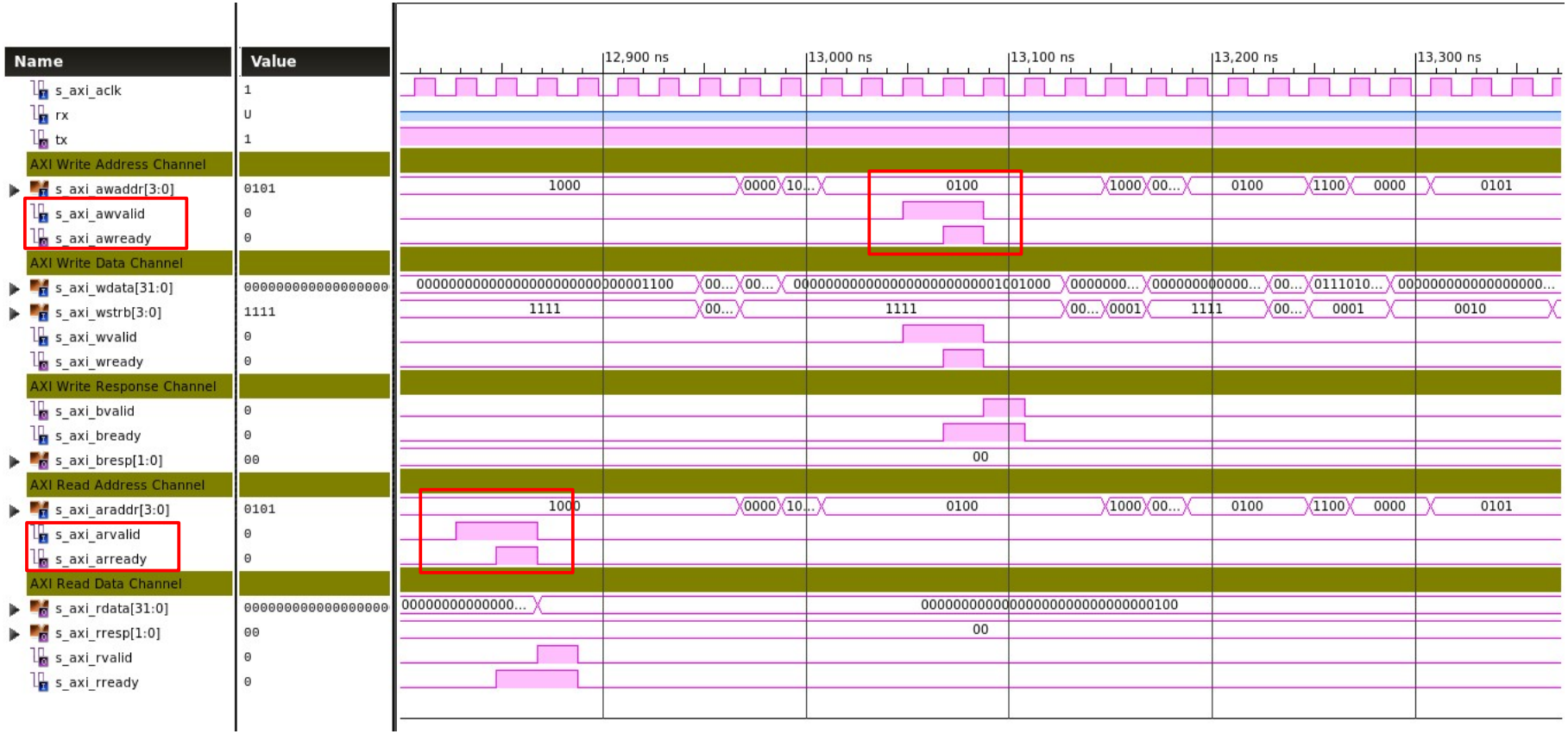



- Übertragung von ‚H‘ und ‚e‘ in „Hello World!“
- Write: FSL_M_Full ist nicht aktiv, MB1 setzt FSL_M_Data und FSL_M_Write
- Read: FSL_S_Exists wird aktiv, MB2 setzt daraufhin FSL_S_Read

# Evaluation – Simulation der UART

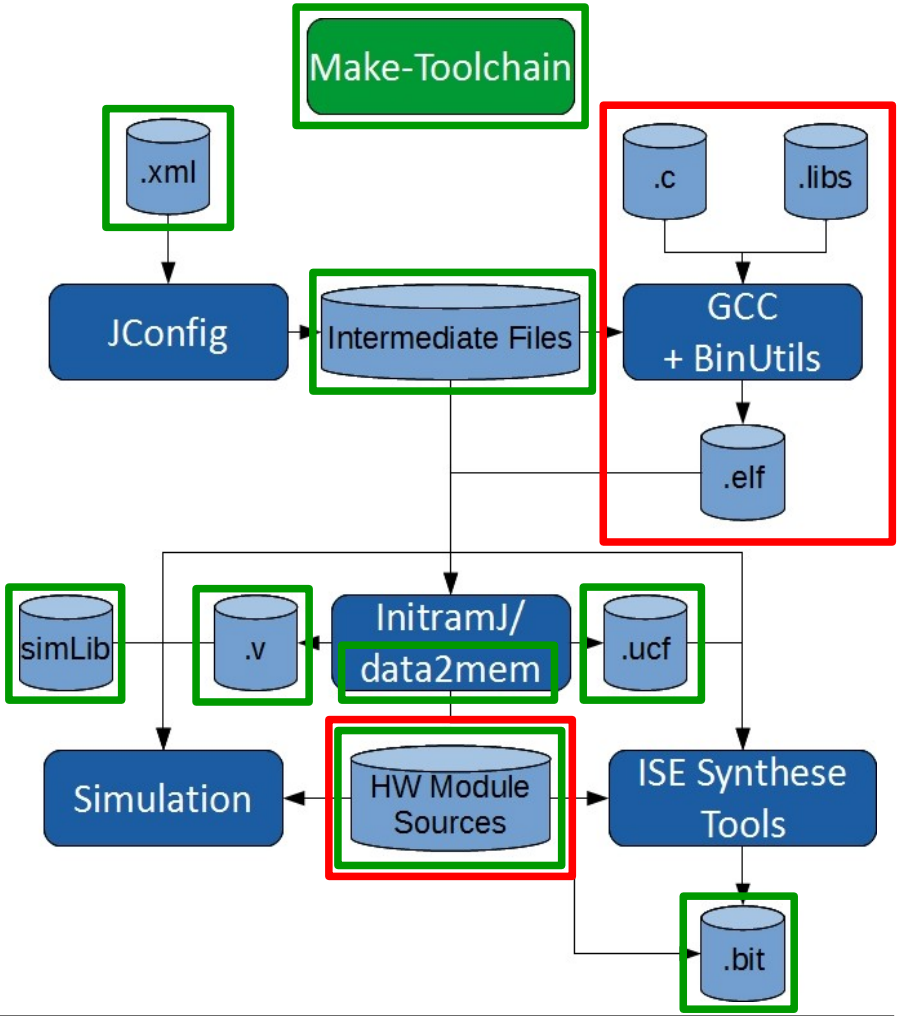


# Evaluation – Simulation der UART im Referenzsystem mit ISIM



# Fazit

- Microblaze, Speicher und Peripherie hinzugefügt
- Speicherinitialisierung mit data2mem
- Anpassungen der Make-Toolchain
- Simulations-Libraries
- GCC noch nicht integriert
- Verwendung des AXI-Busses konnte nicht verifiziert werden



# Ausblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- Integrieren des Microblaze GCC
- Probleme mit dem AXI-Interface beheben
- Optionales Ziel der Arbeit: Parallelisieren eines Microblaze Programms mit µStreams
- AXI-Bus IP-Core hinzufügen für mehr Peripherie
- Debug-Modul
- Microblaze mit unterschiedlichen Konfigurationen verwenden

# Ende der Präsentation

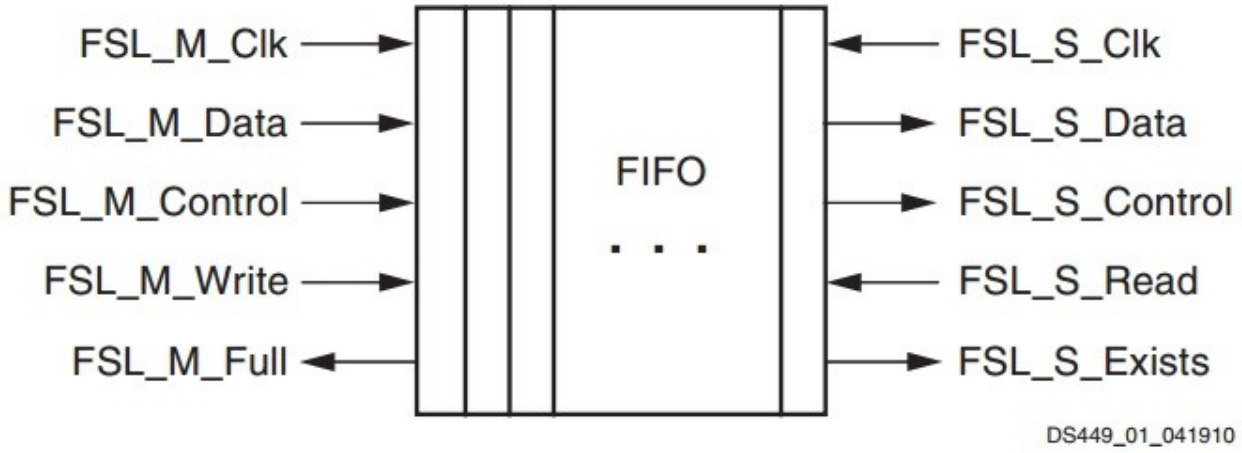
TECHNISCHE UNIVERSITÄT DARMSTADT

Fragen

# Grundlagen – Fast Simplex Link Bus

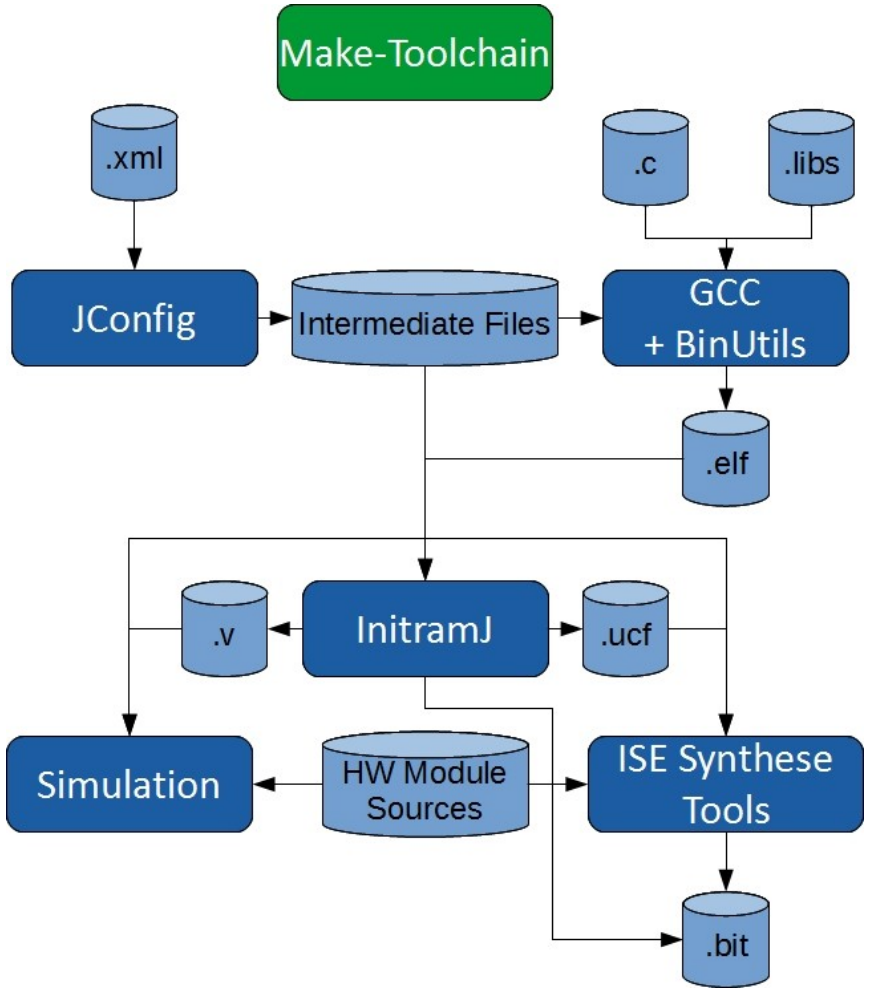
- Xilinx FSL Bus:
  - Unidirektionale Punkt-zu-Punkt Verbindung
  - FIFO-basiert
  - Synchroner oder asynchroner Betrieb
  - Extra Control-Bit

- Write Operation:
  - Setzen von FSL_M_Write falls FSL_M_Full nicht aktiv ist
- Read Operation:
  - Setzen von FSL_S_Read wenn FSL_S_Exists gesetzt ist



# Konzept und Implementierung – CoreGenerator oder HDL-Dateien

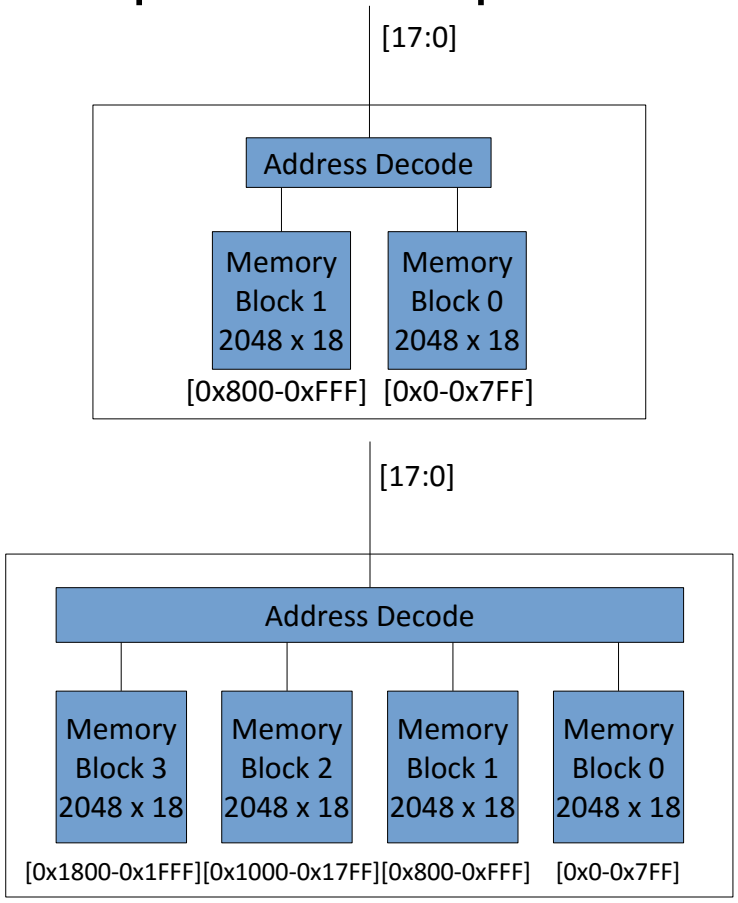
- CoreGenerator oder HDL-Dateien hinterlegen?
  - CoreGenerator verfügt nur über Microblaze MCS
  - MCS hat stark eingeschränkte Parametrisierbarkeit
  - → HDL-Dateien hinterlegen



# Konzept und Implementierung – Speicherimplementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

## SpartanMC Speicher




## Microblaze Speicher