risc-v in vhdl

Entwickelt von David Heim im Rahmen des **Prozessorbau** Praktikums im SS22 an der **Uni Augsburg**

Setup

Das Vivado-Projekt wird automatisch aus TCL-Skripten generiert. Zum Generieren und Öffnen des Projekts:

```
$ vivado -mode tcl -source scripts/project.tcl
```

Achtung: Ein vorhandenes Projekt wird dabei überschrieben. Für ein bereits generiertes Projekt kann man in Vivado die Datei `scripts/project/risc.xpr` öffnen.

Zusätzlich zum Projekt werden auch die verwendeten IP-Module per Skript generiert. Bei Ausführung der `project.tcl` wird automatisch auch das `ip.tcl` Skript ausgeführt, welches die Konfiguration der IP-Module beschreibt. Bei Änderungen an einer Konfiguration muss der Ordner des Moduls unter `scripts/ip/[name]` gelöscht werden, damit das Modul neu generiert wird.

Abgabe

Die Abgabe enthält ein bereits generiertes Vivado-Projekt, wie in Setup beschrieben. Zusätzlich dazu sind drei Programme vorhanden:

1. `test106tid`: Prozessor konfiguriert mit 3 Threads und Thread Start `start_0`
2. `test107spawn`: Prozessor konfiguriert mit 3 Threads und Thread Start `spawn`
3. `test108offset`: Prozessor konfiguriert mit 3 Threads und Thread Start `start_offset`

Die Bitstream und Debug-Probes Dateien liegen im Projekt-Root.
Der Code des jeweiligen Test-Programms liegt im Ordner `src/test_files/src/`.

Multithreading

Multithreading ist eine Form von Thread-level Parallelismus bei der sich mehrere Threads die Ausführungseinheiten eines Prozessors teilen. Jeder Thread benötigt seinen eigenen State, d.h. einen eigenen Program Counter und ein eigenes Register Set.

Es gibt zwei verschiedene Ansätze für Multithreading bei einer Single-Issue CPU:

- Fine-grained Multithreading

  > Wechselt mit jeder Instruktion zwischen den Threads, meist im Round Robin Verfahren. Dadurch können bei Stalls eines Threads Instruktionen der anderen Threads ausgeführt werden und der Prozessordurchsatz aufrecht erhalten werden. Umgekehrt laufen einzelne Threads, welche ohne Stalls ausgeführt werden, langsamer, da sie sich ihre Ausführungszeit mit den anderen Threads teilen.

- Course-grained Multithreading

> Wechselt zwischen den Threads nur bei längeren, kostspieligen Stalls. Dadurch wird der Prozessor bei Stall-freien Ausführungen weniger verlangsamt, allerdings wird bei jedem Thread-Wechsel ein Pipeline-Bubble benötigt.

Dieser Prozessor implementiert Fine-grained Multithreading mit Round Robin und der zusätzlichen Erweiterung, dass im Modus `spawn` (s. Konfiguration) nur aktive Threads ausgeführt werden.

Konfiguration

Es gibt mehrere Konfigurationsmöglichkeiten, die als Generic-Parameter an der `Processor` Entität gesetzt werden:

- `ThreadCount`: Die Anzahl der Threads. Frei wählbar.
- `ThreadStart`:
  - `start_0`: Alle Threads werden sofort gestartet und führen die selben Instruktionen, beginnend bei Adresse 0, aus.
  - `start_offset`: Alle Threads werden sofort gestartet, führen jedoch unterschiedliche Instruktionen aus, beginnend bei einem festen Offset (z.B. Thread 0 bei 0x0, Thread 1 bei 0x100)
  - `spawn`: Nur Thread 0 wird gestartet. Weitere Threads können mit einer speziellen Instruktion an einer bestimmten Adresse gestartet werden (s. *Zusätzliche Instruktionen*). Bei einem `ret` legt sich der Thread wieder "schlafen".

Anpassungen an der Pipeline

- ThreadTag
  Jede Pipeline-Stufe muss wissen, welcher Thread gerade in ihr ausgeführt wird. Ein Signal `ThreadTag` wird dafür über alle Stufen durchgereicht.

- Register
  Jeder Thread benötigt seinen eigenen kompletten Registersatz. Die Register werden hierfür in einem zweidimensionalen Array gespeichert, welches über ThreadTag und Register indexiert wird. Zum Lesen wird der aktuelle ThreadTag der Decode-Stufe und zum Schreiben der ThreadTag der MEM-Stufe mitgegeben.

- Clear/Interlock/Forward
  Die jeweilige Aktion ist nur valide, wenn sie den selben Thread betrifft, der sie ausgelöst hat. Z.B. darf Thread X nur Daten geforwardet bekommen, welche von ihm selbst geschrieben wurden, auch wenn ein anderer Thread zur gleichen Zeit das gleiche Register schreibt. Auch hierfür werden die ThreadTags der jeweiligen Stufen zusätzlich zu dem eigentlichen Signal mitgegeben.

- Fetch-Stufe
  Damit für jeden Thread ein Program Counter gespeichert werden kann, wurde das `PC` Signal als Array implementiert, indexierbar mit dem ThreadTag. Jeden Takt speichert die Fetch-Stufe zuerst den nächsten PC (PC+4) für den aktuellen Thread in das Array und wählt dann den nächsten (aktiven) Thread zum Ausführen und liest dessen PC aus dem Array.
  Besondere Behandlung benötigen in der Fetch-Stufe Jumps. Auch hier muss wieder der ThreadTag des den Sprung ausführenden Threads mitgegeben werden, damit dessen PC entsprechend angepasst werden kann.

- Stalls bei Speicherzugriffen
  Da die Memory-Einheit nicht gepipelined ist muss der Prozessor entgegen der eigentlichen Idee von Multithreading bei Speicherzugriffen weiterhin komplett gestalled werden. Da das Thread-Wechseln aktuell in der Fetch-Stufe stattfindet, um einen möglichst schnellen Wechsel zwischen den Threads zu ermöglichen, hat die CPU zu dem Zeitpunkt noch keine Informationen über die Instruktion, die ausgeführt wird. Eine mögliche Erweiterung wäre es, nicht-Speicherzugriffe anderer Threads weiter auszuführen.

Zusätzliche Instruktionen

Es wurden zwei zusätzliche CSR-Befehle implementiert:

- ThreadTag/Thread-ID des aktuellen Threads in Register speichern:
  Lese CSR `71D` ("TID", "Thread-ID"):

  ```
  csrr  0x71D, #reg
  ```

- Spawnen des Threads mit `id` an der Adresse in Register `#reg`:
  Schreibe CSR `700+id` (z.B. `702` für `id`=2)

  ```
  la    #reg, #address
  csrw  0x702, #reg
  ```

  Dieser Befehl ist ähnlich wie ein Jump implementiert. Die EX-Stufe reicht der Fetch-Stufe den ThreadTag des zu spawnenden Threads, sowie die Spawn-Adresse weiter. Die Fetch-Stufe schreibt dann in das PC-Array.

Probleme

Während in der Simulation alle Features des Prozessors korrekt funktionieren, so haben sich beim Testen auf dem Board eine Vielzahl schwer zu debuggender Probleme ergeben. Bei bis zu 3 Threads verhält sich der Prozessor weitestgehend erwartungsgemäß, ab 4 Threads weicht das Verhalten ab. Selbst direkt verbundene Signale wurden teilweise nicht mehr korrekt weitergereicht. Aus diesem Grund werden alle Test-Programme nur mit 3 Threads ausgeführt.