FGMT-RiscV: A fine grained multi threading processor for FPGA systems

Bernhard Lang[1] *

[1]Faculty of Engineering and Computer Science, Osnabrück University of Applied Sciences

Abstract

A fine-grained multi-threaded processor designed for FPGA applications is presented. Its heart is a processing pipeline which is able to process multiple instruction streams, each of which is represented by a thread token containing a program counter and a thread identifier which selects the related register set. Thread tokens enter the pipeline and are emitted at the output with modified program counter. Instruction and data memories are accessed via streaming interfaces at the side of the pipeline. Inside the processor the thread tokens circulate in a closed ring formed by the processing pipeline, thread handling infrastructure and a token fifo. This ring enables monitoring and manipulation of thread tokens and provides resources for connecting high-level debuggers. Interrupts are handled by threads that execute a wfi instruction that routes its token to a wait stage of the interrupt controller. As soon as an interrupt occurs, the token is restarted without delay and the thread can handle the interrupt event. An example system even suited for small FPGAs consists of the processor, block ram, Wishbone bus and some peripherals. Programming and debugging the system is comparable to the typical software development for embedded systems. The usual GCC compiler suite is used as the programming environment. However, instead of defining interrupt handlers, here related threads are defined and started.

Introduction

This paper presents a special RiscV [1,2] variant, a fine-grained multi-threaded processor, which is referred to as FGMT-RiscV in the sequel. Its heart is a ramified Processing Pipeline with the usual stages: instruction fetch, operands fetch, instruction execution and result store. It is realized as an AXI streaming pipeline [3], which inherently controls the flow of data through the individual stages.

The program counter is fed in at the input of the pipeline and the modified program counter is emitted at the output. The instruction and data memories are accessed via streaming busses on the side of the pipeline. The number of pipeline stages is not fixed; it can be adapted to meet timing specifications by inserting AXI-streaming synchronization components.

The pipeline can handle several threads, each with its own program counter and register set. The program counter together with a thread identifier form a token which represents the associated thread. Typical FPGA block RAMs (4–32 kbit) can accommodate 4 to 32 RiscV register banks; if more threads are required, multiple block RAMs are needed.

In the FGMT-RiscV, the Processing Pipeline is combined with infrastructure (thread launching, monitoring, interrupt and error handling, etc.) and a Token Fifo to form a closed ring in which the thread tokens circulate. The ring incorporates components for manipulating the token stream including debug functionality to enable source-level debugging in the system.

A small system consisting of FGMT-RiscV, block RAM, Wishbone bus [6], GPIO, Timer and UART has been designed which allows high-level debugging of the processor's threads with the GNU Debugger [4] via serial interface.

The Processing Pipeline

Figure 1 shows an overview of the Processing Pipeline. Remind that all data connections in the pipeline are designed as AXI streams, they always contain a *Valid* signal plus *Data* signals in the forward direction and a *Ready* signal in the backward direction.

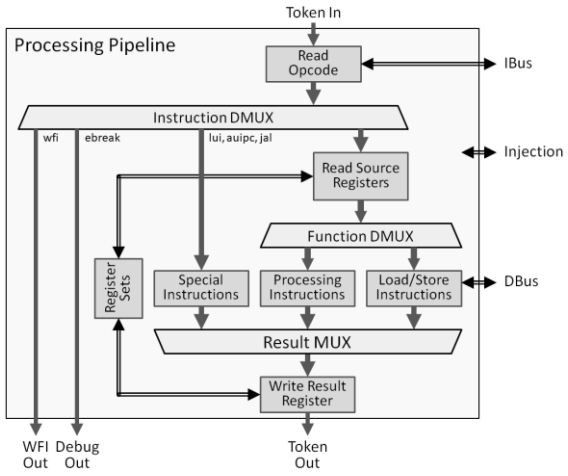

Figure 1: FGMT processing pipeline

* Corresponding author: B.Lang@hs-osnabrueck.de

Thread tokens enter the processing pipeline and first reach the Read Opcode stage in which the instruction is read using the thread's program counter. The instruction information is added to the thread token.

The subsequent Instruction DMUX forwards the expanded tokens to the related one of the four outputs. Most instructions require source register values, their tokens are passed to the Read Source Registers block. The next output is used for special instructions (lui, auipc, jal). Finally, tokens with ebreak instruction are forwarded to the debugger output and tokens with wfi (wait for interrupt) instruction to the wfi output.

The Function DMUX further subdivides between processing and load/store instructions. The Processing Block currently supports the RV32I instruction set, the Load/Store block is able to read and write external data memory using request/response streams.

Finally, all instructions that generate a result value are merged with the Result MUX towards the Write Result Register block that writes the result value to the selected register.

Injection of commands for debugging is supported but only indicated in the figure, and handling of errors is implemented but not shown in the figure.

The Processor

Figure 2 shows the internal structure of the processor. The main processing ring is formed by the Thread Fifo via Processing Pipeline, Thread Filter and a MUX back to the Thread Fifo. In this ring tokens circulate during normal execution, with each token representing one of the independent threads.

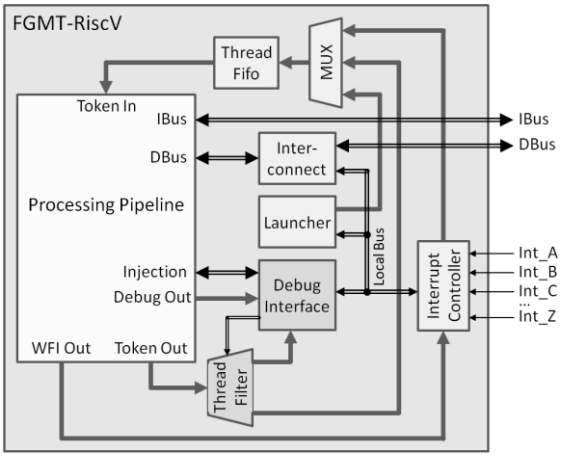

Figure 2: FGMT-RiscV processor

Interrupt handling is initiated by a thread that executes a `wfi` instruction. The Processing Pipeline emits this token to WFI Out, which takes it to the Interrupt Controller. There it waits until an assigned interrupt occurs. Triggered by the interrupt, the token leaves the controller immediately and the thread handles the interrupt.

Threads are started using the Launcher block. At startup this block emits the token of the initial thread; further threads can be launched by software.

For debugging, the Thread Filter redirects the tokens of a selected thread to the Debug Interface. Another thread working as debug server can read the tokens of the selected thread and gains access to its registers via inject. Finally the server can continue the stopped thread via the Launcher.

The System

As an example, a small system consisting of the FGMT-RiscV, some block RAMs as system memory, Wishbone Bridge and the Wishbone peripherals GPIO, Timer and UART is implemented in an XC7A35T FPGA device using Digilent's BASYS3 board. Its block diagram is shown in Figure 3. This system requires only about one fifth of the FPGA resources.

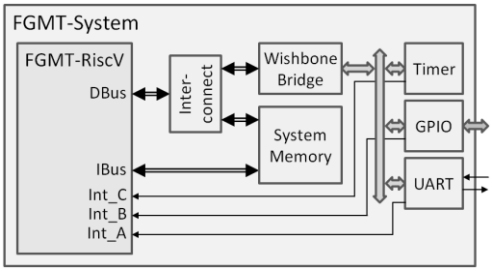

Figure 3: FGMT example system

Programming and debugging the system is comparable to software development for embedded systems. The GCC compiler suite is used as programming environment. However, instead of defining interrupt handlers, now interrupt handling threads are defined and started.

As an extra goodie, an embedded GDBServer software is installed as thread 0, which serves GDB's RSP protocol [5] via serial interface. It thus allows direct connection to a development PC running the GNU Debugger [4] to debug the processor's other threads.

References

[1] The RISC-V Instruction Set Manual Volume I, Unprivileged Architecture, Version 20240411

[2] The RISC-V Instruction Set Manual: Volume II, Privileged Architecture, Version 20240411

[3] AMBA® 4 AXI4-Stream Protocol Specification, Version: 1.0. ARM, 2010.

[4] Debugging with GDB, Version 17.0.50.20250122. Free Software Foundation, Inc., 2024.

[5] Implementing a Remote Stub: Chapter 20.5 in Debugging with GDB, Version 17.0.50.20250122. Free Software Foundation, Inc., 2024.

[6] WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Revision: B.3. opencores.org, 2002.