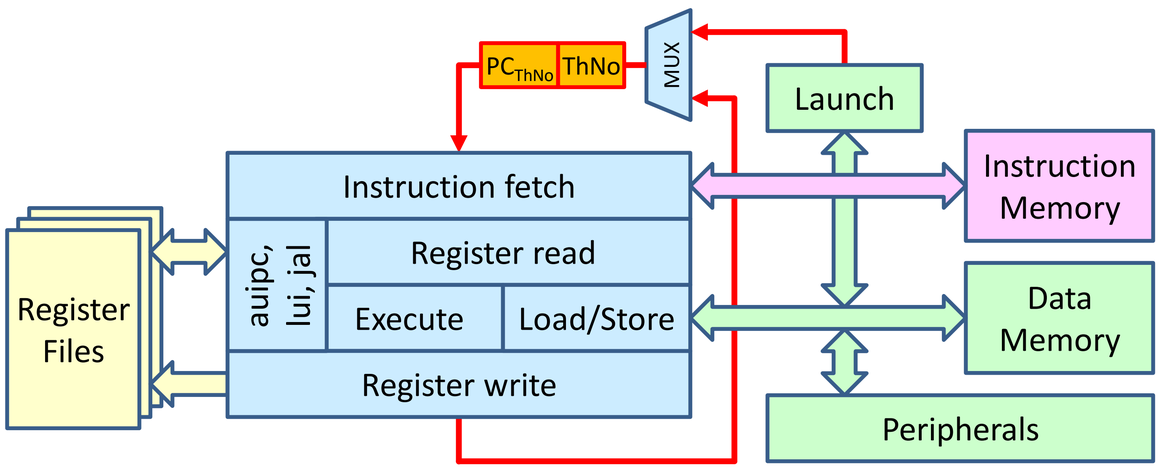
FGMT-RiscV:
A fine grained multi threading processor for FPGA systems

Prof. Dr.-Ing. Bernhard Lang
Hochschule Osnabrück, University of Applied Sciences, Faculty of Engineering and Computer Science

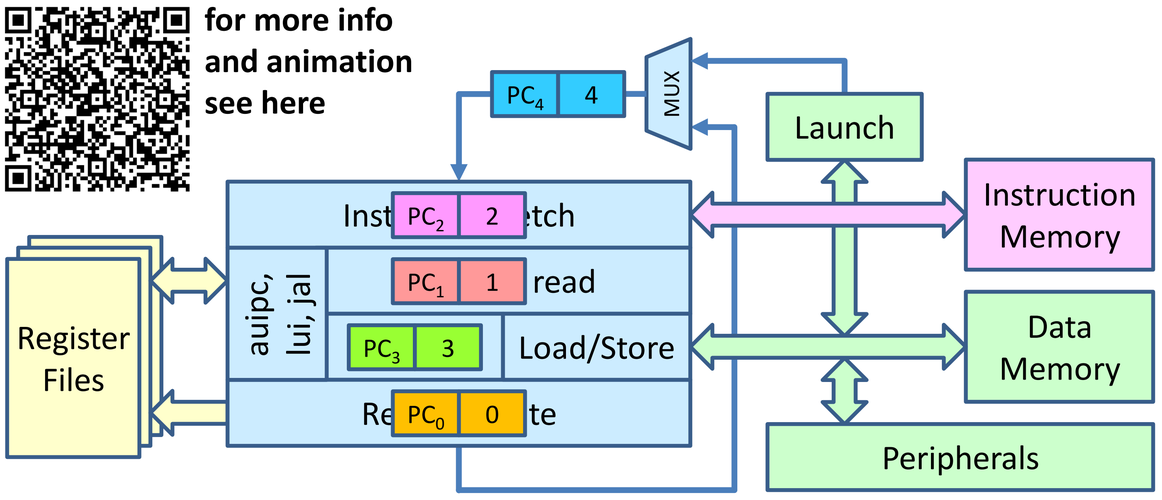
The Idea: Processor without a program counter register. Tokens formed by a program counter value (PC) and a thread number (ThNo) circulate through the instruction pipeline:



- First thread is launched by PowerOn, it can launch further threads by software.
- Program counter value is changed in the pipeline to match the instruction.
- Each thread has its own register set.
- RiscV is well suited for this approach because its pipeline is context free.

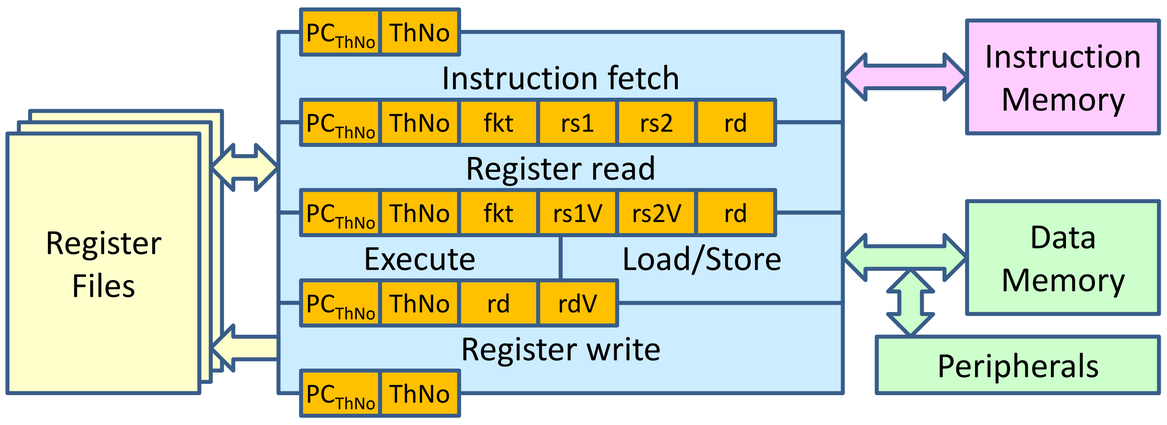
Multi Threading: Tokens of multiple threads circulate through the pipeline stages:

for more info and animation see here



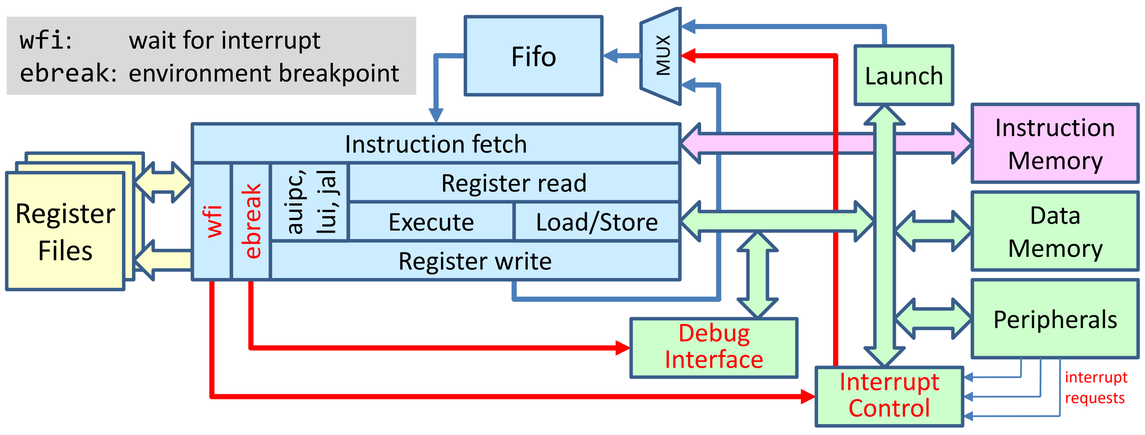
- The stages are modeled as an **AXI streaming** pipeline, so the throughputs of the interleaved threads are automatically synchronized with each other.

Token flow through the pipeline: The tokens are first expanded, then modified and finally reduced back to PC and thread number:



- Instruction fetch adds instruction information to the input token.
- Register read replaces source register addresses to source values.
- Execute and Load/Store determine a result value from the source.
- Register write stores the result value back into the destination register.

Interrupts and Debugging: `wfi` and `ebreak` related tokens are redirected to special pipeline paths



- ebreak tokens flow to a debug interface and are handled by a debug server.
- wfi tokens wait in an interrupt control block for relaunch by an interrupt.

Software: Programming the FGMT-System is similar to programming interrupt-controlled embedded systems. For real time processing, no software scheduler is required if the maximum number of hardware threads configured for this FGMT-Processor is sufficient.

Interrupt handling: Instead of interrupt handlers, individual threads handle interrupt events. Such a thread executes a `wfi` instruction which redirects its token to the interrupt controller. There it waits until an interrupt request occurs. Then the controller immediately relaunches the waiting thread so that it can handle the interrupt without delay.

Code snippets:

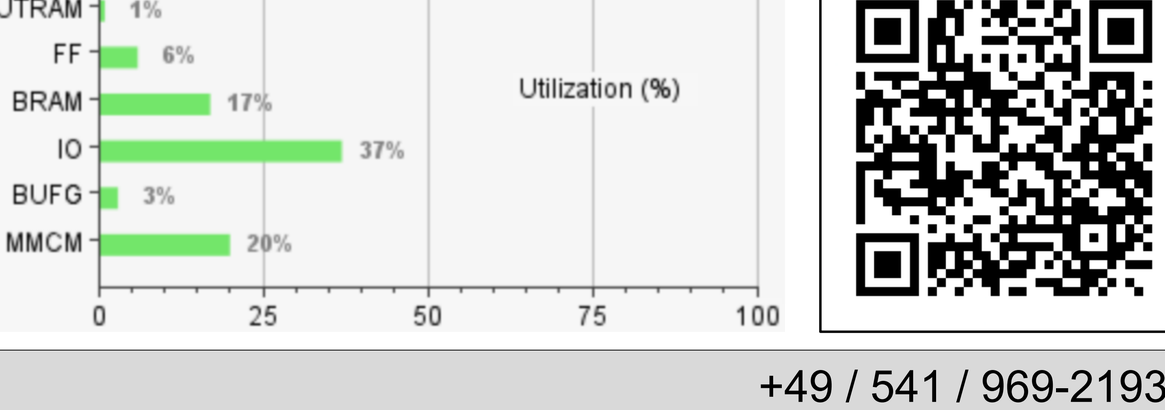
```
void Thread3(void) { // Normal processing
  ... // Thread3 initializations
  while (1) {
    ... // Thread3 loop
 }
}
unsigned int Thread3_Stack[TH3SSIZE]; // Stack for Thread3

void Thread2(void) { // Interrupt handling
  ... // Thread2 initializations
  while(1) {
    asm ("wfi"); // Let thread 2 wait for Interrupt
    ... // Interrupt Handling
  }
}
unsigned int Thread2_Stack[TH2SSIZE]; // Stack for Thread2

void riscv_Launch(void* pc, unsigned int thread, void* stack);

int main( void ) { // Normal processing
  riscv_Launch(Thread2,0x2, Thread2_Stack); // launch thread 2
  riscv_Launch(Thread3,0x3, Thread3_Stack); // launch thread 3
  ... // main initializations
  while ( 1 ) {
    ... // main loop
  }
}
```

Eclipse/GCC/GDB support: For programming and debugging the GCC compiler suite is used. A GDBServer that supports the RSP protocol of the GNU debugger can run as thread 0 of this FGMT system. Then a simple serial connection between the GDB and the GDBServer allows debugging the other FGMT-RiscV threads. For convenient operation, the Eclipse IDE can be used.



System realization: The FGMT-RiscV system is described in VHDL. A first example implementation uses the AMD XC7A35T FPGA on a Digilent BASYS3 board. It is operated at 50 MHz, higher speeds are possible. The System has been configured for 16 threads and set up with GPIO, UART, Timer and 32kBytes block memory. The following graph shows its utilization:

Contact:

B.Lang@hs-osnabrueck.de
+49 / 541 / 969-2193