



HW-SW Codesign

Lab Course Introduction

*Jakob Lechner,
Thomas Polzer*

http://ti.tuwien.ac.at/ecs/teaching/courses/hwswcode_lu

Overview



- ▶ HW/SW-Codesign concept
- ▶ Spear2 processor core
- ▶ Extension modules
- ▶ Software tools
- ▶ Course organization

HWSW-Codesign Concept



- ▶ Starting point: Software application running on a processor core
- ▶ Insufficient performance of SW
- ▶ Objective: „Optimize“ the current implementation by:
 - Partitioning application into HW and SW tasks
 - Software tuning of existing implementation
 - Adding at least one „hardware module“
- ▶ Optimization criteria based on cost function

Constraints & Trade-Offs



► Constraints

- Available HW and SW resources
 - FPGA: available logic cells and memory blocks
- Power consumption
 - also SW consumes energy
- ...

► Trade-Offs

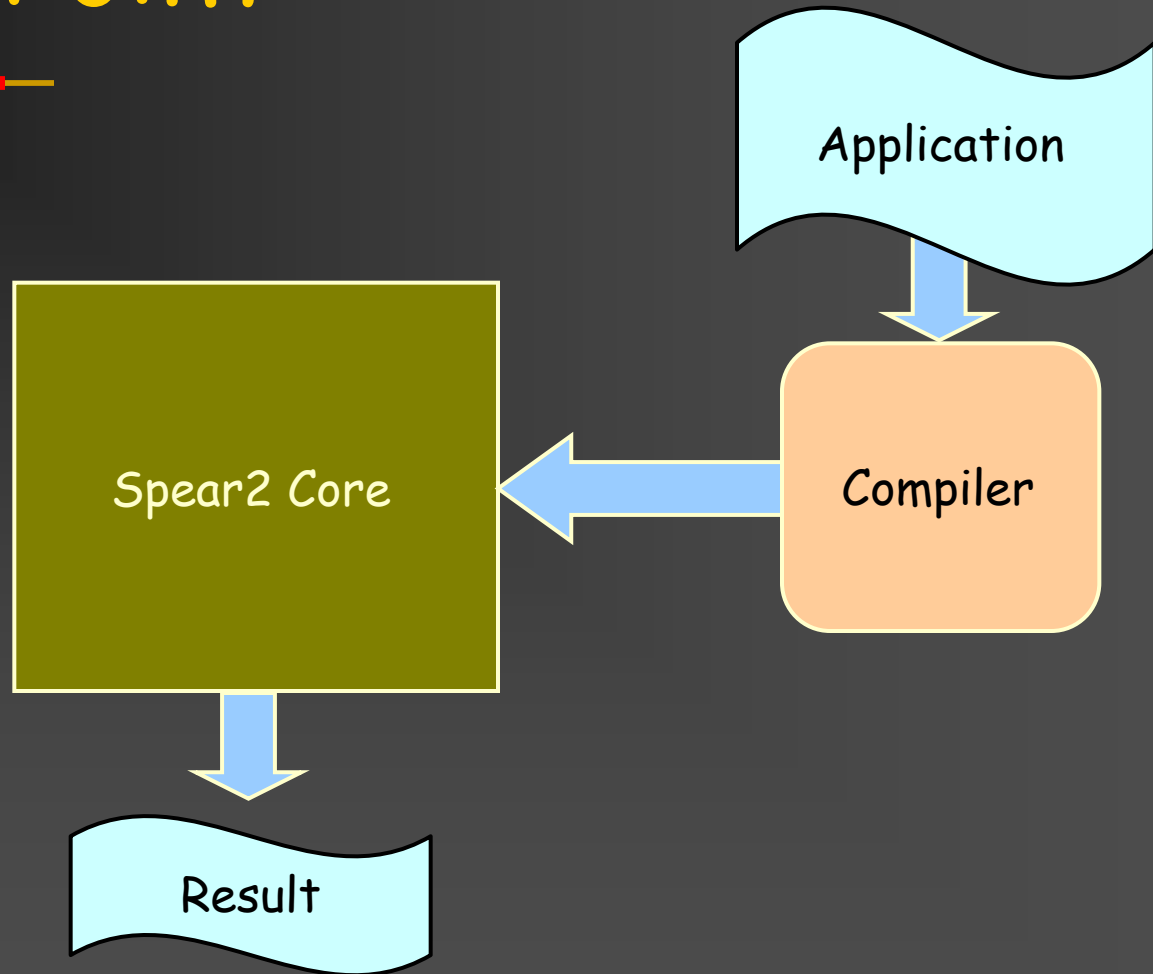
- Performance vs. size
- Hardware costs vs. software costs
- ...

Optimization Parameters



- ▶ Hardware resources (Spear + Extensions)
 - # LUTs
 - # Register-Bits
 - # Memory-Bits
- ▶ Software resources
 - # Instruction memory usage
- ▶ Performance (i.e., execution time)
- ▶ Power consumption

Starting Point



A horizontal bar chart with a red bar and a yellow bar. The red bar is significantly longer than the yellow bar.

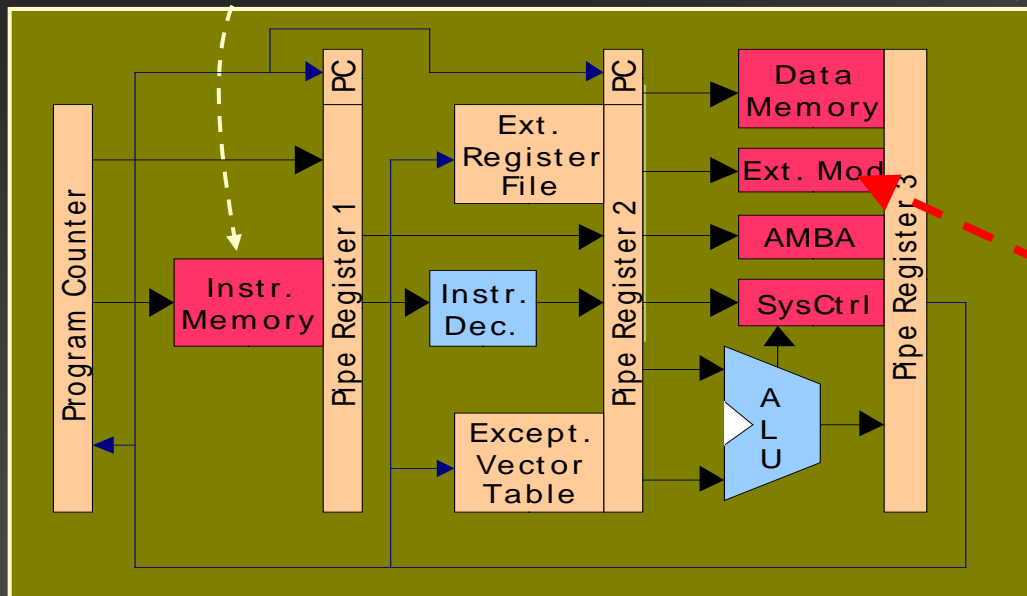


```
int main (int argc,
          char **argv)
{
    ...
    SlowFunction(...);
    ...
}
```

- ▶ Performance analysis (profiling)
 - ▶ HW-Counter for counting elapsed cycles

Analysis

C-compiler

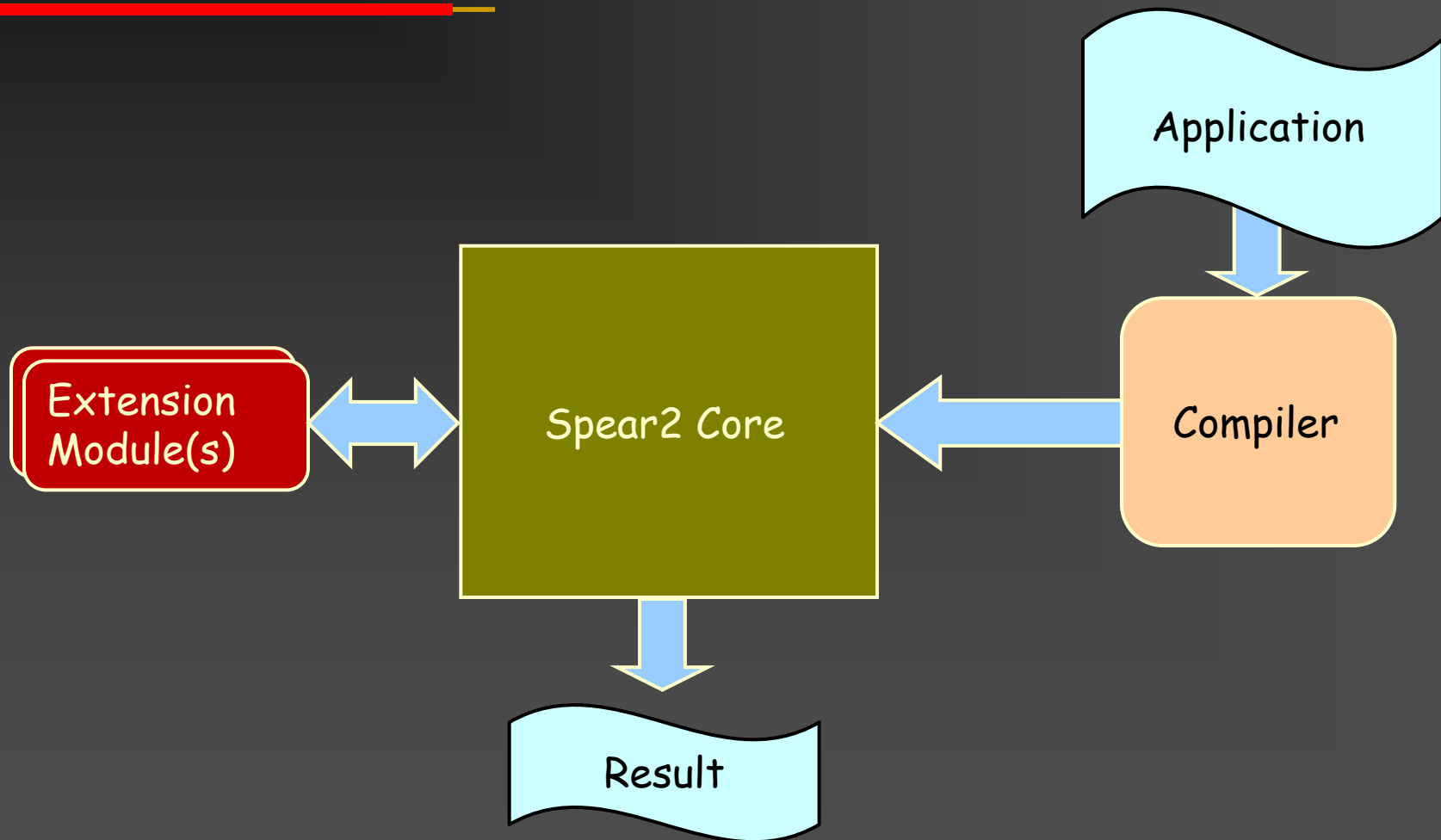


```
int main (int argc,  
          char **argv)
```

```
{  
    ...  
    SlowFunction(...);  
    ...  
}
```

- ▶ Performance analysis (profiling)
- ▶ HW-Counter for counting elapsed cycles

Optimized Architecture



SPEAR2 Features



- ▶ 16 bit instructions - 122 instructions
- ▶ Conditional instructions
- ▶ Configurable 16 or 32 bit data path
- ▶ 16 registers
 - 2 special purpose registers: RTE, RTS
- ▶ 16 interrupts and 16 traps
- ▶ Configurable data- and instruction memory size
- ▶ 4 framepointer / stacks
- ▶ All hazards resolved in hardware

Source Directory - Spear2

VHDL

ext_modules

spear2_core

...

workspace

amba_hws

modelsim

quartus

VHDL

Available Extension Modules

VHDL-Source of Spear-Core-Units

Pre-configured Spear-Instance for this course

ModelSim scripts

Quartus Project Directory

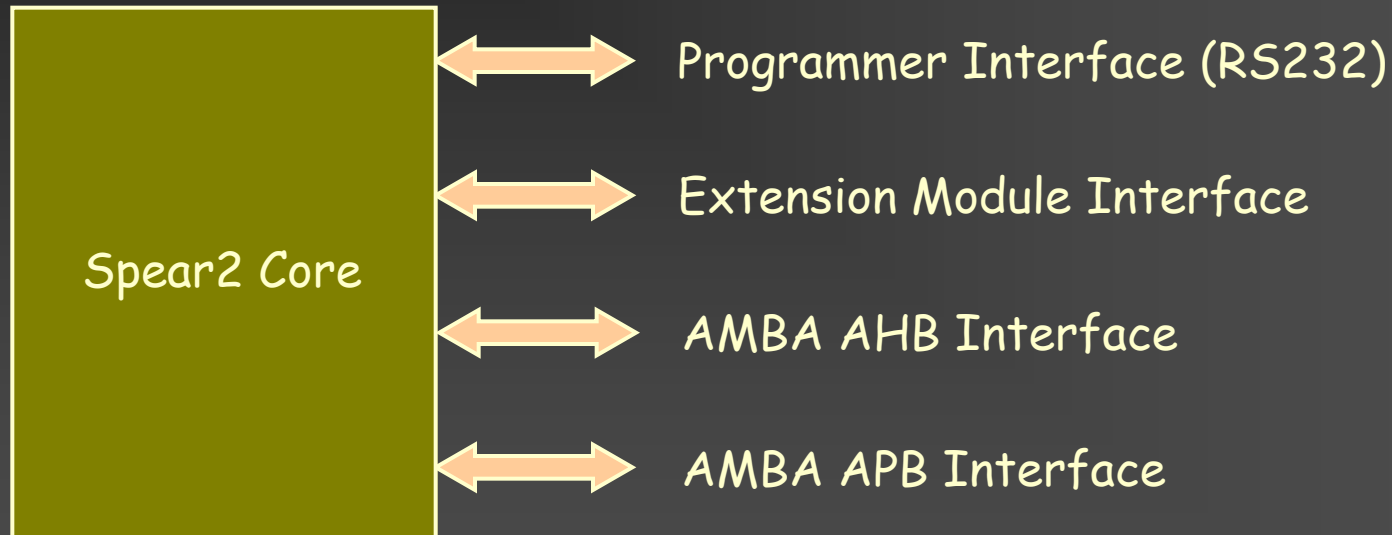
Top-Level-Entity (contains Spear-Core and Extension Module components)

Build-Proecss - Spear2

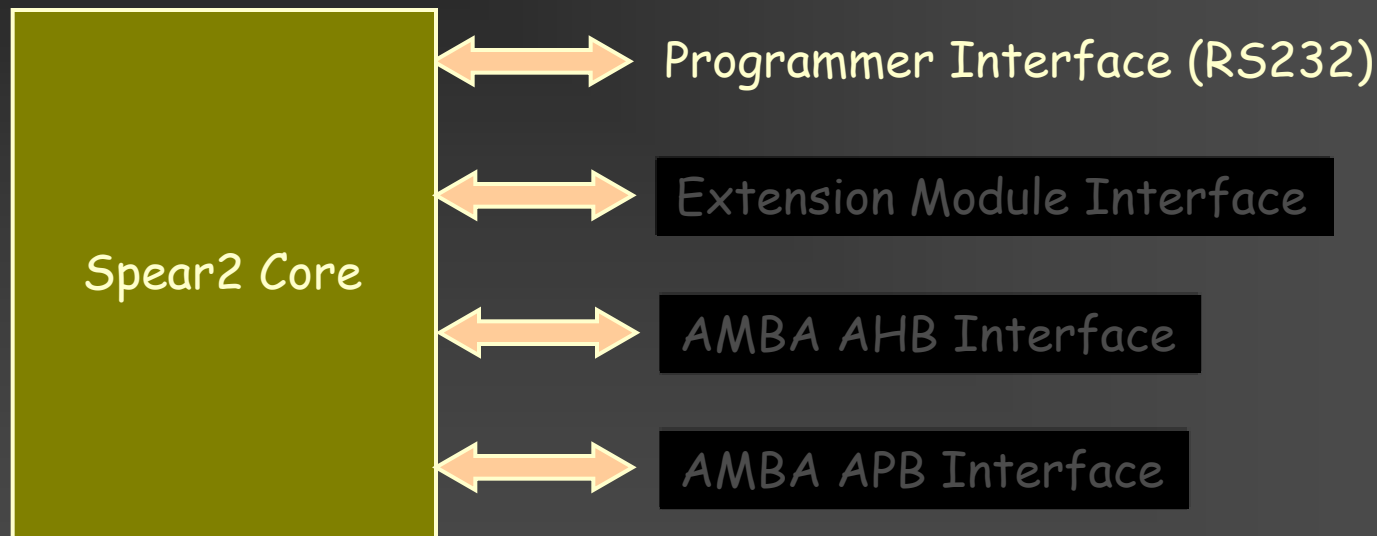


- ▶ Get Spear2 project from our website
- ▶ Unzip the archive in some working directory
- ▶ Open the Quartus project and hit "compile"
- ▶ Download the produced sof-file on the target

Spear2 Interfaces

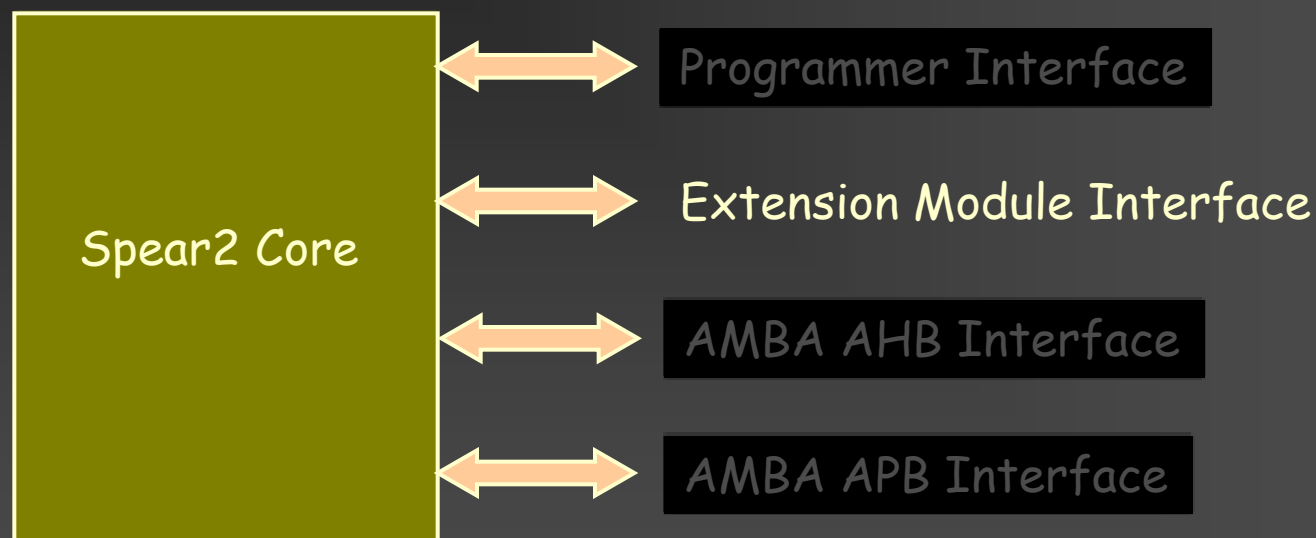


Programmer Interface



- After reset bootloader waits for download of program
- File format: Motorola SREC

Extension Module Interface

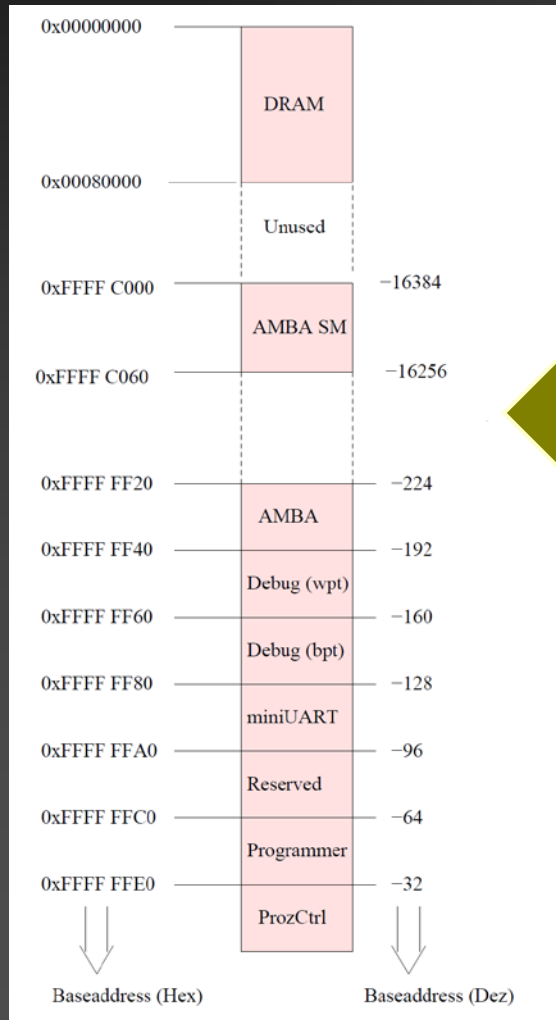


Extension Modules 1



- ▶ Memory Mapped Hardware Modules
- ▶ Typically 32 bytes per module (i.e., 8 words)
 - ▶ Larger address range possible, e.g. for directly mapping shared memory
- ▶ First word holds *status* and *config* registers
- ▶ Exchanging data with ordinary load/store instructions on assigned addresses

Extension Modules 2



Predefined address range for own extension modules

Extension Modules 3

► Hardware-Interface (defined in Spear-Package)

```
type module_in_type is record  
  reset    : std_ulogic;  
  write_en : std_ulogic;  
  byte_en  : std_logic_vector(3 downto 0);  
  data     : std_logic_vector(31 downto 0);  
  addr     : std_logic_vector(14 downto 0);  
end record;
```

```
type module_out_type is record  
  data     : std_logic_vector(31 downto 0);  
  intreq   : std_ulogic;  
end record;
```

Extension Modules 4

- ▶ Entity declaration of module
- ▶ Module needs to be instantiated as component in top-level unit of Spear
- ▶ *extsel* signal activates module => pull signal high if your module's address is applied to address bus

```
entity demo_mod is
port (
    clk      : in std_logic;
    extsel   : in std_ulogic;
    exti     : in module_in_type;
    exto     : out module_out_type;
    -- other I/O ports
    ...
);
End demo_mod;
```

Extension Modules 5

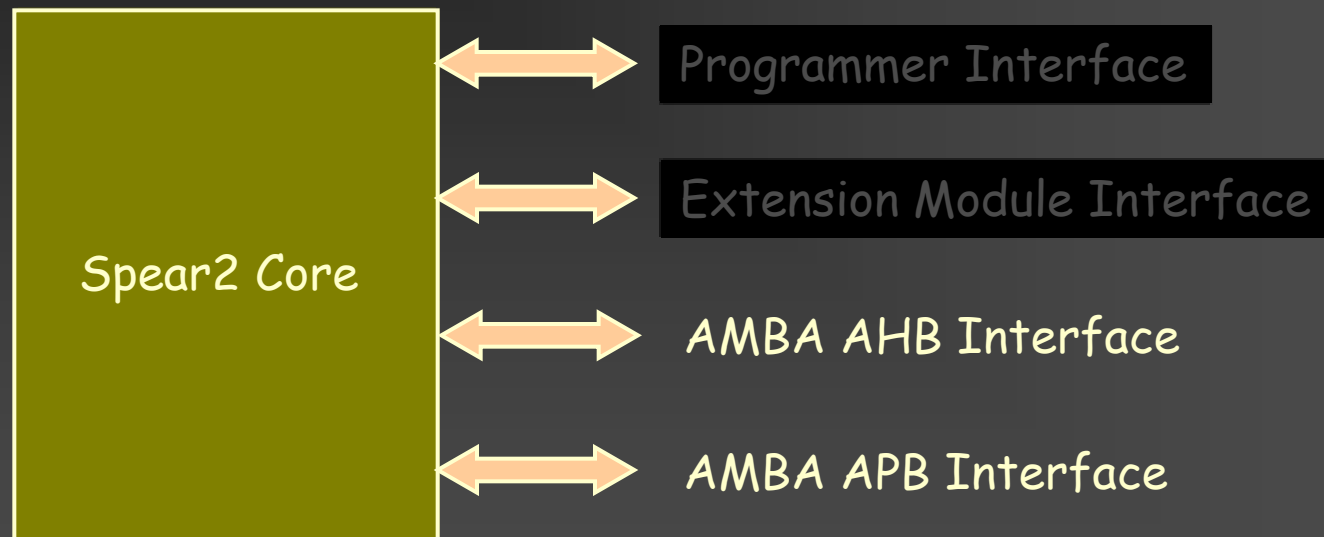
- ▶ Write software drivers for communication with extension modules
- ▶ Use function inlining for frequent operations

```
// Defines for memory addresses
#define DEMO_BASE (0xFFFFFEA0)
#define DEMO_STATUS (*(volatile int16_t *const) (DEMO_BASE))
#define DEMO_OP1 (*(volatile int8_t *const) (DEMO_BASE+6))
#define DEMO_OP2 (*(volatile int8_t *const) (DEMO_BASE+7))
```

```
inline static int8_t myfunction(int8_t x, int8_t y)
__attribute__((always_inline));
```

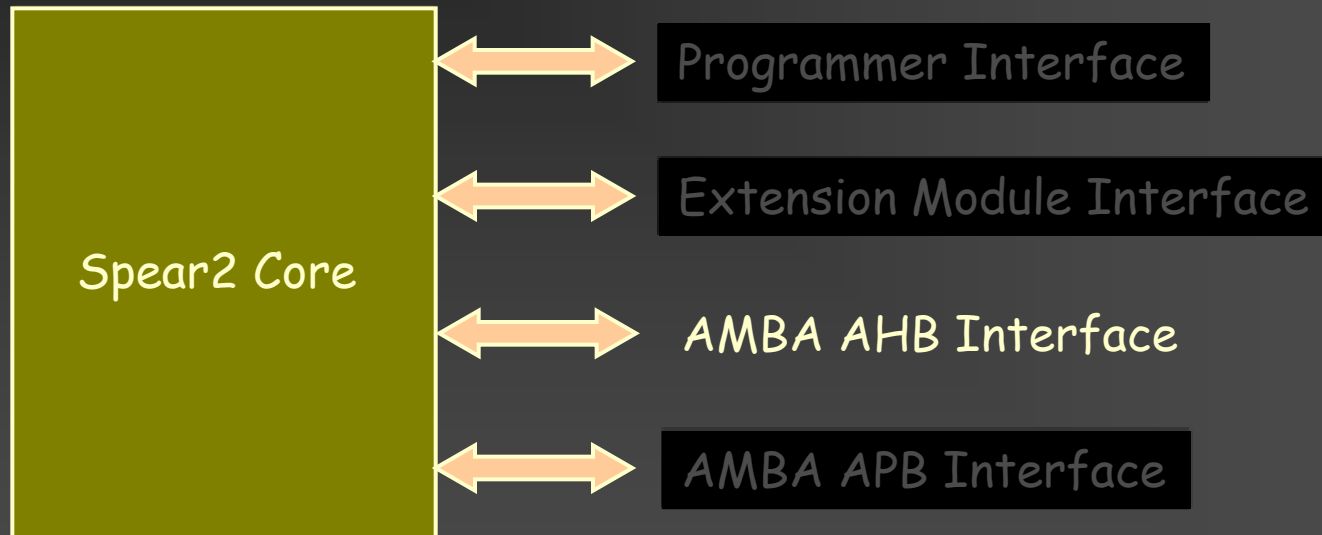
```
inline static int8_t myfunction(int8_t x, int8_t y) {
    DEMO_OP1 = x;
    DEMO_OP2 = y;
    ...
}
```

AMBA Interfaces



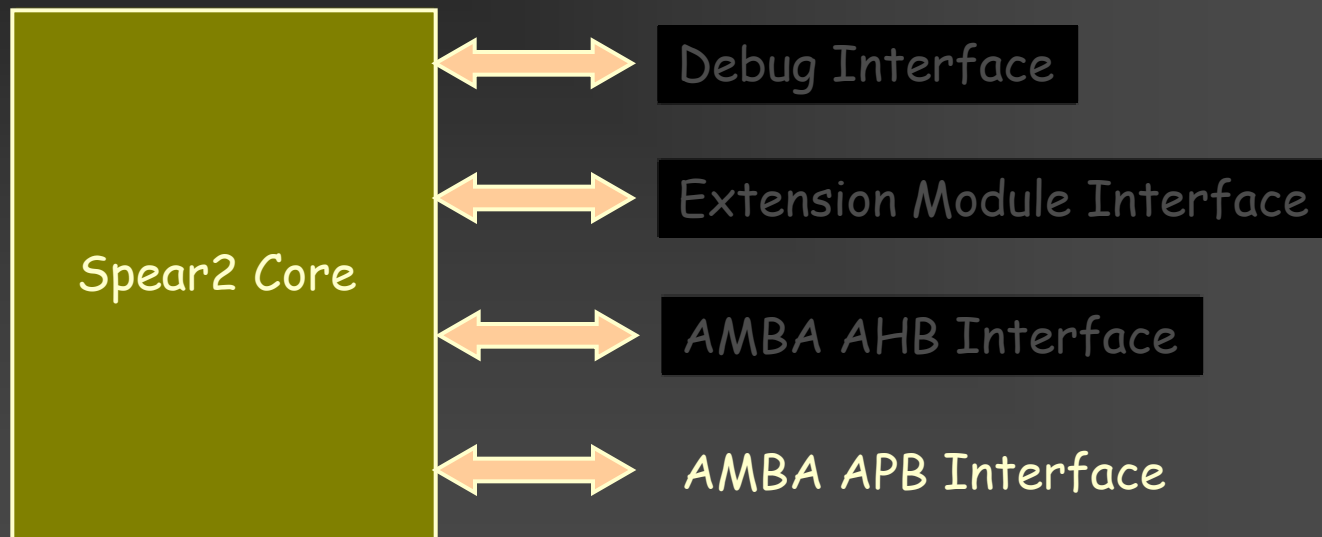
- AMBA modules are memory mapped
- Direct write/read access possible
- „Complex“ access using a shared memory

AMBA AHB



- AMBA Advanced High-performance Bus interface
- IP Cores available at Gaisler Research (GRLIB)

AMBA APB



- AMBA Advanced Peripheral Bus interface
- IP Cores available at Gaisler Research (GRLIB)

Software



- ▶ Binutils (Assembler, Linker, ...)
- ▶ GCC-Compiler
- ▶ GDB
 - Offline Simulator
 - Frontends: DDD, Eclipse

Course Organization



▶ Where?

- ECSLAB, Treitlstraße, 2nd floor
- No lab slots, free working hours

▶ Submission deadline for solution: 24.06.2011

▶ Group presentations

- 15-20 min
- 27.6. / 29.6. 12:00 – 14:00, EI8

▶ Grading

- Oral exam at end of semester

References



▶ LVA-Homepage

- http://ti.tuwien.ac.at/ecs/teaching/courses/hwswcode_lu

▶ Spear2 Project

- <http://trac.ecs.tuwien.ac.at/Spear2>

▶ HWSW Codesign Forum