Agenda

ComRV – Cacheable Overlay Manager for RISC-V

– Introduction: Solving code space limitations with Software
– Basic concepts and use case
– Building blocks: SW and Tool chain blocks
– Deployments
Introduction
ComRV – Introduction

Solving code space limitations with Software

• In the early days of embedded computing there was a technique to load code on Real-Time at the moment it was needed for execution.

• This technique was named Overlay, and it was threaded with the toolchain (compiler, liker, etc.) providing easy application-interface for the SW developers.

• Today, IoT devices (Internet of things) are quite strict with memory size and power, alongside requirements for simple HW implementation which does not contain MMU or high-end operation system, to manage it (linux/windows)

• Those needs, alongside RISC-V code density challenges, rises a need to revive the overlay concept to fit to RISC-V ISA, and use its toolchain to support it.
Basic Concepts
ComRV – Basic concepts

Run-Time Module/Engine

• A run-time module operates on the fast memory deciding which function to load or not from a storage device, to fast memory heap

• Code is dynamically loaded to “cache” and executed according the program flow

• The run-time module manages the cache and responsible for invoking the overlay functions
ComRV – Basic concepts

User usage

Example **code without** ComRV:

```c
void bar(void);

void foo(void)
{
    bar();
}
```

Toolchain generated code:

```assembly
jal x1, 0x12345678       ; bar()
```

Example **code with** ComRV:

```c
void __attribute__((overlaycall)) bar(void);

void foo(void)
{
    bar();
}
```

Toolchain will generate code:

```assembly
li    x31, 0x04C38835   ; bar() “token”
jalr  x1, x30          ; ComRV RT-engine
```

compiler creates special calls
ComRV – Basic concepts

Functions Group

• Overlay group is a container for functions & read-only-data

• When the RT-Module decides to load a function, a full group will be loaded.

• Size of a group is between 512Bytes – 4kBytes

• Gathered functions to groups is done:
  – Manually: User “register” function to group
  – Automatically: Done by external toolchain utility on link-time, triggered by the linker
ComRV – Basic concepts

Multi Grouping

• Sometimes different SW scenarios can run the same function

• Example case study on overlay:
  – We have small cache that only fits one group
  – MyFooFunc is in Group A, it is used by Func42 and also needed by Func1003 which is Group B.
  – Meaning we need to evict A when B is running, and back to A when we call MyFooFunc from Func1003.
  – Results = too many loads = too much time

• Multi Groups will give the option to put MyFooFunc both in Group A and B.
ComRV – Basic concepts

Logic flow

• Each overlay function will be passed through ComRV engine
• The engine is written in C/Asm so it is threaded with RISC-V ISA
• The engine manages the load/evict
Building blocks
**ComRV – Building blocks**

**SW and Tool chain blocks**

- For ComRV we need few changes in the toolchain and support utilities.
- Compiler [LLVM and clang]:
  - The compiler creates special call for overlay func.
- Linker [GNU-LD]:
  - Create descriptors (tokens) for functions, and offset tables for overlay functions.
- Debugger [GDB]:
  - Provides easy interface to the users for debugging overlay function and overlay core.
- Utilities:
  - Grouping tool, extend .map file, other service utilities…
ComRV – Building blocks

Grouping – Finding Overlay Activity (time based)

Initialization stage
repeated itself on each power cycle
(bring up time)

FW Steady State
Perform device specific operations
ComRV – Building blocks

Grouping – Finding Overlay Activity (time based)

Function calls per time

Function A

Function B

Function C
ComRV – Building blocks

Grouping – Finding Overlay Activity (time based)

Merge – Functions calls per Time

Initialization stage

repeated itself on each power cycle (bring up time)

operation 1

FW Steady State

Perform device specific operations
Deployment
ComRV – Deployment

How we can get access to ComRV?

• ComRV is **open sourced**

• It is designed to fit bare-metal software and with RTOS based software (FreeRTOS supported on phase)

• Fully deployed with all toolchain support

• Following next: More demos of usage on each ComRV API

• ComRV code can be found on the following link on github:

https://github.com/westerndigitalcorporation/riscv-fw-infrastructure