Software Overlay for RISCV - HLD

Revision 0.1

January 6, 2021

Document Revision History

Revision	Date	Contents	Author(s)
0.1	06/Jan/2021	Draft – Engine HLD	Ronen Hean Ofer Shinaar

Table of Contents

List of Figures

List of Tables

Reference Documents

Item #	Document	Revision Used	Comment
1			

Abbreviations

Abbreviation	Description

1 Overview

Some systems (mostly embedded systems) have limited memory resources and as a result the total code footprint is bigger than the available memory. The concept of arranging code in ‘code overlays’ is quite old but still valid these days to resolve the code size issue. The following document specifies the requirement and design of an overlay manager engine for RISC-V.

2 High-Level Design

2.1 Block Diagram

The following figure describes a general firmware block diagram with the RT-Engine:

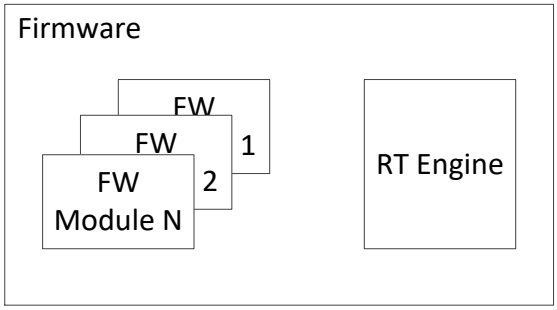

Figure 1: Generic Firmware Block Diagram

Firmware – this is the software being executed by the RISC-V core. It may contain several firmware modules that interact with each other or with different hardware components through firmware drivers. The firmware code can reside/execute in different memories, e.g., ROM, RAM, FLASH, etc.

FW Module – Firmware module that responsible for a specific operation in the program. One Firmware module may reside/execute in a different memory region than other firmware modules.

RT-Engine – this is the module responsible for managing the run time code load and execution. It is compiled and linked as all other firmware modules, and as such, it may also reside/execute in different memories. Any existing firmware module wishing to invoke a function defined as an overlay function will indirectly use the RT-Engine to dynamically load the function (if not already loaded) and invoke (call) it.

2.2 Overlay functions

When developing firmware code for systems with memory constraints, the engineers will program the code to define which function is designated to be an overlay function. All marked overlay functions are gathered into overlay groups with a size range of 512B-4K each. An overlay group may contain one or more overlay functions, and it is the responsibility of the toolchain to create the overlay groups encapsulating overlay functions.

2.2.1 Overlay group structure

An overlay group size ranges from 512B – 4K and may contain several functions. Since an overlay group's boundary is always 512B, the group will be padded with the Overlay `Group ID` up to the upper 512B boundary.

The structure of an overlay group is –

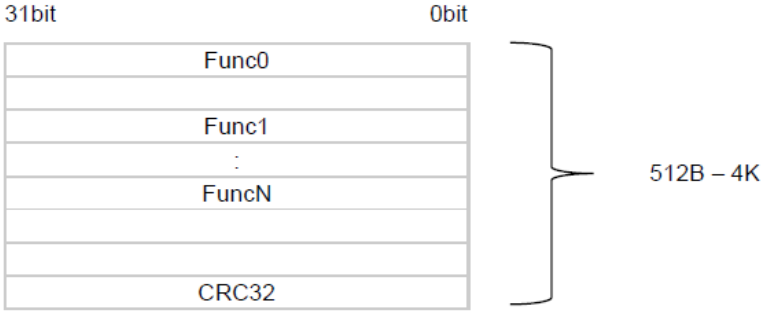

Figure 2 - Overlay group structure

2.3 Overlay function call

On regular operation, when a given function *foo()* performs a call to function *bar(),* the toolchain generates a core-specific 'jump' instruction code and resolving the jump 'address'. In overlay design, if *bar()* function is defined as an overlay function, the compiler can generate a 'jump' instruction, but the linker will not be able to resolve the symbol's address since the 'jump' is not referring a fixed address in memory.

2.3.1 Implicit RT Engine invocation

Since the linker can't resolve the actual address of the overlay function *bar()*, and it does know the address of the RT Engine entry point, the compiler shall plant a 'jump' instruction to the RT Engine entry point instead of a 'jump' to *bar()*. To distinguish which overlay function is to be loaded and invoked, the linker will use an address token defining the *bar()* overlay function instead of the actual *bar()* address. Sharing a token will allow the RT Engine to prepare (load/invoke) the correct overlay group in memory along with the *bar()* function offset within the overlay group.

Example code w/o overlay manager:

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

Toolchain generated code:

```
:
:
jal 0x12345678      ; bar()
:
```

Example code with overlay manager:

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

Toolchain generated a call RT Engine:

```
:
li  x31, 0x04C38835 ; bar() token
jr  x30             ;  RT Engine
:
```

Note: x30, x31 are reserved for RT Engine per the overaly standard

Figure 3 - Overlay operation example

2.3.2 Implicit RT Engine invocation for a non-overlay function

When a function *foo()* is declared as an overlay function, and it is calling a non-overlay function *bar()* there is a chance that when returning from *bar(), foo()* will already be evicted.
That could be if additional overlay functions were loaded due to calling *bar()* or in another scenario, an OS context switch occurred, and overlay function calls were done from that context.

Returning to an "already evicted" caller means that all non-overlay function calls that are made from within an overlay function must be done through the RT Engine. The toolchain will replace the call to *bar()* with a call to the RT Engine, and will set the token value to point to *bar()* address.

When the RT Engine is invoked, it will check if the token is a real token or an actual address; in this case it is an actual address the RT Engine will directly jump to that address. When *bar()* completes, it will return to RT Engine, which will load *foo()* if not loaded, and return to it.

2.3.3 Address Token

An address token is an *overlay function descriptor* providing all the needed information for the RT Engine to load and invoke an overlay function. A regular address will always be an even number. Therefore, to differentiate a token address from a standard address, the least significant bit of the address token shall be set to 1 (odd).

2.3.3.1 Overlay address token structure

The overlay address token is a 32bit value defining a specific overlay function as follows –

31	29	28	27	17	16
Multi-group token	Heap ID	Reserved		Function offset	Overlay group ID →
				1	0
← Overlay group ID					Overlay address token

B31	Multi-group token	B31 [1] – B16:1 specify a multi-group overlay ID B31 [0] – B16:1 specify a regular overlay group ID
B30:29	Heap ID	Heap region identification
B28	Reserved	
B27	Thunk call	Calling an overlay function through a function pointer
B26:17	Function offset	Value defining the function offset from the beginning of the group; value expressed in 4 bytes granularity
B16:1	Overlay group ID	Overlay group ID: regular overlay group ID (function resides in) or multi-group overlay ID (ID to a list of groups the function resides in)
B0	Overlay address token	Overlay token indication: B0 [1] – B31:0 define an overlay token address B0 [0] – B31:0 define a memory address

Table 1 - Overlay token structure

2.4 RT Engine Management Tables

The following management tables are required for the RT Engine operation:

2.4.1 Overlay offset table

This table is an array of overlay offsets prepared by the linker. A table index represents an overlay group ID; a table entry holds a specific overlay group's offset. For example, entry #1 defines the location offset of overlay-group ID #1. The offset is relative to the beginning of all existing overlays (Per overlay standard - *"overlay area"*). There can be a case where several Overlay Offset Tables exist, and each such table refers to a different overlay heap location (Heap ID Table 1 – Overlay token). In run-time, the RT Engine shall get the overlay group ID from the address token and use it with this table to determine the overlay offset to be loaded. The overlay offset granularity is expressed in 512B units.

2.4.1.1 Overlay offset table structure

An entry in the overlay offset table is defined as follows –

15		**0**
Group offset		
B15:0	Group offset	Offset from the begging of the overlay section; value expressed in 512B granularity.

Figure 4 - Overlay offset table structure

Table size (number of entries) shall be equal to the number of overlay groups plus one unused entry[1]; a single table entry represents each overlay group. The group offset value is accumulative, and the overlay group size is calculated by subtracting the overlay offset of the x+1 entry with the overlay offset of x entry.

e.g., in the following table, overlay group ID 2 is in offset 5632B (11 * 512B) from the beginning of the overlay section, and its size is 1024B:

Entry	overlay offset table [size]
0	0 [3*512]
1	3 [8*512]
2	11 [2*512]
3	13 [X*512]
	:
n-1	
n	

Figure 5 - Overlay offset table example

[1] Entry n doesn't represent an actual overlay, it exists to calculate the size of overlay ID n-1.

The 'Overlay Group ID' field of the Overlay Address Token is used to access an entry in the table.

2.4.2 Overlay multi-group table

Any given function can be defined as a multi-group function meaning it will reside in more than one overlay group. The toolchain prepares this table; the table index defines the multi-group identifier; table entries are sub-lists of address tokens specifying all multi group overlay functions; each sub-list defines the overlay groups of one function. In run-time, RT Engine shall use the input address token to determine if the token describes a multi-group token. If so, the overlay ID token field specifies the first index of the sub list in the overlay mult-group table; RT Engine will iterate through the sub list and check if one of the address tokes in the sub list is already loaded. If none of them are loaded, the first entry of the sub list is used to specifies the default address token. TBD several groups are loaded – which one to select

2.4.2.1 Overlay multi-group table structure

An entry in the overlay multi-group table is an Overlay Address Token (see table 1 – Overlay table).

The table size depends on the number of multi-groups and the number of occurrences per function. Each mutli-group token list is separated by a zeroed Address Token.

e.g. if there is only one multi-group and the multi-group function appears in 3 overlay groups, this means we'll have a single multi-group ID (ID 0) and the Overlay multi-group table shall contain 4 entries (last entry will be zero).

In the following example, we see that there are 4 multi-groups with the IDs – 0, 3, 8, 11, and each multi-group contains 2, 4, 2, and 3 occurrences of each function, respectively.

Entry	Overlay Multi-Group table
0	Some Address Token [1]
	Some Address Token
	0
3	Some Address Token [1]
	Some Address Token
	Some Address Token
	Some Address Token
	0
8	Some Address Token [1]
	Some Address Token
	0
11	Some Address Token [1]
	Some Address Token
	Some Address Token
	0

Figure 6 - Overlay Multi-Group example

Upon receiving an Overlay Address token, with 'Multi-group token' field set, it takes the 'Overlay Group ID' field of the Overlay Address Token to access a sub token list in the Overlay Multi-group table.

[1] Each first entry is the default entry in case none of the Address Tokens of a specific group is loaded

2.5 Reserved registers

2.5.1 RT Engine Entry Point Address register

2.5.2 RT Engine Stack Frames Pool register

2.5.3 RT Engine Stack register.

2.5.4 RT Engine Overlay Address Token register

2.5.5 RT Engine COM-RV return value

2.6 RT Engine

2.6.1 High-level flow

2.6.1.1 Address token query

2.6.1.2 Search for an already loaded overlay group

2.6.1.3 Eviction

2.6.1.4 Load

2.6.1.5 Saving the callee token and caller return address

2.7 Debugger awareness for overlays