The Trampoline Handbook

release 2.0

Jean-Luc Béchennec
Mikaël Briday
Sébastien Faucou
Pierre Molinaro
Florent Pavin

CONTENTS

Part I

The Real-Time Operating System

CHAPTER

ONE

Porting Trampoline

In this chapter *arch* is used to designate the instruction set of the target like PowerPC®, ARM® or AVR®; *chip* is used to designate the name of an implementation of the architecture like a PowerPC 5516; *board* is used to designate the name of a development board that uses the chip. *compiler* is used to designate the compiler and *linker* is used to designate the linker used to link the project and produced the executable file.

1.1 Adding files to the directory structure

Doing a port of Trampoline on a new target requires

- data structures
- code, some is in C and some is in assembly language of the target
- code templates
- memory mapping templates (depend on the compiler)
- link scripts templates (depend on the linker)

Data structures declarations and code related to the instruction set are located in the '`machines/`*`arch`*' directory.

Code templates are located in the '`goil/templates/code/`*`arch`*' directory.

Memory mapping templates are located in the '`goil/templates/compiler/`*`compiler/arch`*' directory.

Link scripts templates are located in the '`goil/templates/linker/`*`linker/arch`*' directory.

For instance, if the goal is to port Trampoline to a Freescale® ColdFire® CPU with the gcc compiler and the gnu ld linker, you have to create a directory '`coldfire`' inside the '`machines`' directory, inside the '`goil/templates/code`' directory and inside the

'goil/templates/code/gnu_ld' directory. For gcc, memory mapping templates are common to all *arch* and exist already but, for instance, the Metrowerks® C compiler uses different #pragma according to the *arch*. So memory mapping templates for the Metrowerks C compiler for PowerPC would be located in 'goil/templates/compiler/mwc/powerpc' and for HCS12 would be located in 'goil/templates/compiler/mwc/hcs12'

In addition, some code or link scripts may be specific to the *chip* or the *board*. In this case, create sub-directories in the various *arch* directories using the pattern '*arch/chip/board*' to put the corresponding files.

1.2 Using a target with goil

The `-t` or `--target` option of goil selects the target by using a *arch/chip/board* path. Goil will look at the code, compiler and linker templates in the corresponding paths. Goil looks for a template at the deeper path first and goes up until it find it or gives an error when it does not find it. This way, a generic *chip* level template may be overridden by a more specific *board* level template for instance.

If you use the `-g` or `--generate-makefile` option, goil generates a Makefile that includes the Makefiles that exist along the path.

The link script templates (linker) and the memory mapping templates (compiler) are used only if a project is built using memory mapping. `MEMMAP` is a boolean attribute of the OS object in the OIL file. `COMPILER` and `LINKER` are sub-attributes of `MEMMAP` when it is `TRUE`. For instance, a `MEMMAP` using gcc and gnu ld would described like that:

```
MEMMAP = TRUE {
  COMPILER = gcc;
  LINKER = gnu_ld { SCRIPT = "script.ld"; };
  ...
};
```

Using this description and the target option, goil will look for link script templates in 'goil/templates/gnu_ld/*arch/chip/board*' path and for memory mapping templates in 'goil/templates/gcc/*arch/chip/board*' path.

The `SCRIPT` sub-attributes gives the name of the generated link script file.

1.3 Architecture specific code and structures

The following informations require you use a software interrupt to call the system services.

This code should be located in the 'machines/*arch*' directory or in a sub-directory ('*chip*' or '*board*') if you want to implement a feature that rely on a specific chip or board (for instance to put peripheral devices in sleep mode in the `tpl_sleep` function). Anyway, you should put the relevant code at the corresponding level. If in the rare instances you may need to use conditional compiling, you may use the C macros `TARGET_ARCH`, `TARGET_CHIP` and `TARGET_BOARD` that contains the *arch*, *chip* and *board* respectively as character strings.

1.3.1 Functions called by Trampoline

The following functions are needed by Trampoline:

```
extern FUNC(void, OS_CODE) tpl_init_context(
  CONST(tpl_proc_id, AUTOMATIC) proc_id);
```

`tpl_init_context` may be written in C. It is called when an activated task runs for the first time. It initializes the context of the task by setting the 'at start' values of registers. Setting at least the values of the stack pointer at the beginning of the stack zone of the task and the return address at the entry point of the task code are required.

```
extern FUNC(void, OS_CODE) tpl_init_machine(void);
```

`tpl_init_machine` is called at the beginning of `StartOS` before calling the `StartupHook` and starting the scheduling. `tpl_init_machine` should do the hardware related initializations that are needed to run the OS (for instance starting the timer of the `SystemCounter`).

```
extern FUNC(void, OS_CODE) tpl_sleep(void);
```

`tpl_sleep` is called from the idle task. It should implement a loop around an instruction that put the CPU in a waiting for interrupt mode. If the *arch* does not have such an instruction, an empty loop may be used.

`tpl_sleep` should never return.

```
extern FUNC(void, OS_CODE) tpl_shutdown(void);
```

`tpl_shutdown` is called from `ShutdownOS`. It should disable all interrupts and put the CPU in sleep mode. If no sleep mode exists, an empty loop may be used.

`tpl_shutdown` should never return.

1.3.2 Data types and structures

A file named '`tpl_machine.h`' should exists in the '`machines/`*`arch`*' directory. This file should contain the declarations and definitions of:

- the `tpl_stack_word` type
- the `tpl_stack_size` type
- the `tpl_context` structure
- the `tpl_stack` structure
- the `IDLE_ENTRY` macro that should set to `tpl_sleep`
- the `IDLE_STACK` macro
- the `IDLE_CONTEXT` macro

The `tpl_stack_word` type is used to achieved a correct alignment of the stack

The `tpl_context` context structure contains one of more pointers to structures where all the registers needed for the execution context are stored. More than one pointer may be needed because on some architectures, contexts may be split in 2 or 3 parts to store the integer context, the floating point context and the vector context for instance. This way a task doing only integer computation needs the integer context only. The other pointers are set to NULL and the context switching code does not save or restore contexts for the NULL pointers. A `tpl_context` field is included in the static part of a task descriptor which may be stored in ROM. For instance, on an AVR, the context structure is declared as follow:

```
struct TPL_CONTEXT {
  avr_context *ic;
};
typedef struct TPL_CONTEXT tpl_context;
```

and an `avr_context` is defined as follow:

```
struct AVR_CONTEXT {
  u8 *sp;
  u8 regist[33];  // registers: R0-R15,R17-R31,SREG,R16
};
typedef struct AVR_CONTEXT avr_context;
```

The `tpl_stack` stack structure contains one or more pointers to the stack and one or more stack sizes. Some ABI may use more than one stack (an example is the Infineon C166). A `tpl_stack` field is included in the static part of a task descriptor. The AVR stack structure is as follow:

```
struct TPL_STACK {
  tpl_stack_word  *stack_zone;
  tpl_stack_size   stack_size;
};
typedef struct TPL_STACK tpl_stack;
```

The `IDLE_STACK` macro should expand to a `tpl_stack` initialization. This macro is used to initialize the stack in the idle task descriptor. For instance, the AVR `IDLE_STACK` and the component it uses are defined like this:

```
#define SIZE_OF_IDLE_STACK  50

extern VAR(tpl_stack_word, OS_VAR)
  idle_stack[SIZE_OF_IDLE_STACK/sizeof(tpl_stack_word)];

#define IDLE_STACK { idle_stack, SIZE_OF_IDLE_STACK }
```

The `IDLE_CONTEXT` should expand to a `tpl_context` initialization. This macro is used to initialize the context in the idle task descriptor. For instance, the AVR `IDLE_CONTEXT` and the component it uses are defined like this:

```
extern avr_context idle_task_context;

#define IDLE_CONTEXT {&idle_task_context}
```

Part II

The Goil system generator

BIBLIOGRAPHY

[1] *Programming Environments Manual for 32-Bit Implementations of the PowerPC^{TM} Architecture*, chapter 8, pages 8–157. Freescale semiconductor, rev. 3 edition, September 2005.

[2] AUTOSAR. Specification of compiler abstraction. Technical report, AUTOSAR GbR, August 2008. http://autosar.org/download/R3.1/AUTOSAR_SWS_CompilerAbstraction.pdf.

[3] AUTOSAR. Specification of memory mapping. Technical report, AUTOSAR GbR, June 2008. http://autosar.org/download/R3.1/AUTOSAR_SWS_MemoryMapping.pdf.

[4] Microcontroller Applications IBM Microelectronics. Developing powerpc embedded application binary interface (eabi) compliant programs. Technical report, IBM, Research Triangle Park, NC, September 1998.

[5] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, 1990.