MSP430X port - basic checkpointing design documentation

David Garriou, Jean-Luc Béchennec

November 21, 2019

1 Normal MSP430X startup sequence

The MSP430X startup sequence is as follow:

1. After a reset, the `tpl_reset_handler` executes (see `tpl_startup.S` file). It:
   (a) stops the watchdog timer;
   (b) disables the interrupts;
   (c) sets up the stack;
   (d) calls `tpl_continue_reset_handler` C function.
2. `tpl_continue_reset_handler` (see `tpl_startup.c` file) does:
   (a) initialization of the .bss section to 0 (uninitialized variables) and initialization of the .data section by copying the initial values from FRAM (initialized variables)[1];
   (b) clock setup by calling `tpl_set_mcu_clock`;
   (c) initialization of the MPU;
   (d) a call to `main`.
3. `main` is responsability of the user but usually it:
   (a) initializes application level devices;
   (b) calls `StartOS`.
4. `StartOS`:
   (a) calls `tpl_init_machine` that calls;

[1] Why not use the DMA to do that, it would use less energy

i. `tpl_init_machine_generic` that calls:

A. `tpl_init_mpu` which is not implemented yet (and would be redundant with 2c).

ii. `tpl_init_machine_specific` that calls:

A. `tpl_set_systick_timer`.

(b) calls `tpl_start_os` that does a system call to `tpl_start_os_service` that.

i. calls `tpl_init_os`;

ii. calls `tpl_enable_counters`;

iii. calls `StartupHook` if any;

iv. calls `tpl_start_scheduling`.

when returning a task is schedule.

2 Modified sequence to restore a checkpoint

Following items shall be modified when restarting from a checkpoint:

- item 2a should be replaced by a copy from the checkpoint data in FRAM to SRAM.
- item 2b should use the replaced by an init with the clock frequency when the checkpoint was done. This can be done by having a variable (in .data segment) to store the clock frequency so that it would restored as part of the checkpoint data.
- item 2d should be replaced by a call to a mandatory function used to initialize the devices for the application (UART for instance) and to a new service, let's call it `RestartOS`. `RestartOS` would:

  1. Call `tpl_init_machine`;
  2. Call `tpl_restart_os` that does a system call to `tpl_restart_os_service` that does a `tpl_start`. `tpl_start` moves the highest priority task from the ready list to the elected slot of `tpl_kern`. Conditions shall be `NEED_SWITCH` true and `NEED_SAVE` false.

When returning from the `RestartOS` service, the highest priority task is scheduled and the system continues execution.

A boolean variable stored in FRAM, let's call it `tpl_checkpoint_available`, shall be used to select, when true, the modified sequence instead of the normal one.

3 Checkpointing

A new service is necessary, let's call it `Hibernate`. When called, `Hibernate`, terminates the caller. It copies the SRAM to the FRAM (checkpoint), stops the Systick, stops application

interrupts (a user function shall be provided for that), programs the RTC to emit an interrupt every x seconds, enables interrupts and goes in LPM3.

The RTC ISR check the voltage of the MCU. If it is above a threshold (`RESUME_FROM_HIBERNATE_THRESHOLD`, 1.9 V ?), it exists from LPM3. If this never happens, after a while, the MCU will power off. When in the future the MCU restart, what is described in section 2 applies.

If `Hibernate` resumes because the RTC ISR exited from LPM3 then it disables interrupt, starts application interrupts, starts the Systick, stop the RTC. It calls `tpl_start` to elect the highest priority task and returns.

In addition, a periodic basic task, `energy_task` checks (every 5 seconds ? more ?) the voltage. If the voltage drops below a threshold (`HIBERNATE_THRESHOLD`), `energy_task` calls `Hibernate` (and terminates):

```
TASK(energy_task)
{
  uint16 voltage = readPowerVoltage();
  if (voltage < HIBERNATE_THRESHOLD) {
    Hibernate();
  }
  else {
    TerminateTask();
  }
}
```

`HIBERNATE_THRESHOLD` shall be chosen so that the worst voltage drop between 2 executions of `energy_task` plus the voltage drop due to checkpointing is lower than the threshold.