Checkpointing in a real time OS

David Garriou

February 1, 2020

Plan

Many Thanks to

- Smile

  1800 geeks

  European expert in open source solutions

  `www.smile.eu`

  Smile did fund a part of the work
- LS2N – UMR CNRS 6004 (French National Centre for Scientific Research)

  École Centrale de Nantes, IMT Atlantique, Université de Nantes

  `www.ls2n.fr`

  Special thanks to my friends and former colleagues, Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou

Plan

The context

Some facts

- So many connected objects in the coming years
- So much raw data sent to the cloud
- Too much data will be stored in the cloud

Our motivation

- Decreasing the environmental impact of IoT
- Decreasing the maintenance cost of IoT

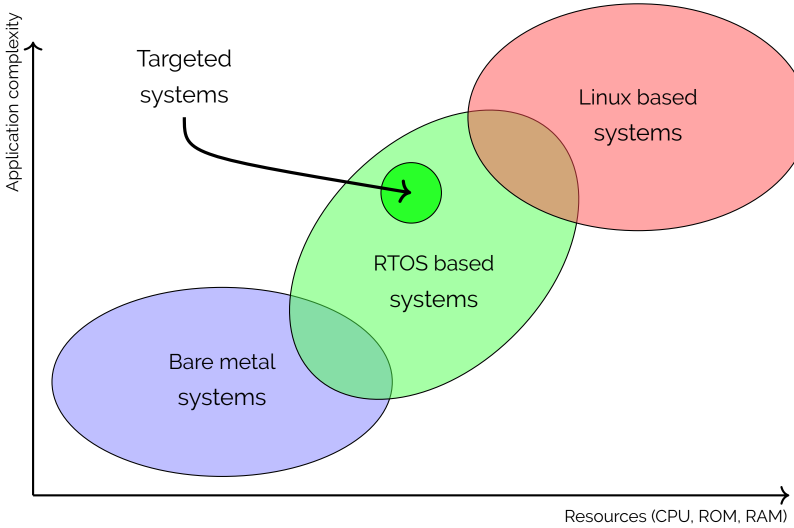
Our Goal

- Avoid transmission of raw data
- Provide a platform for usual sensing, transmission, “heavy” computing
- Have fun of course !

Targeted Systems

- Constrained embedded devices
- Application over an operating system
- Batteryless devices, so we need hypothesis :
    - Some non volatile RAM (NVRAM)
    - A supercapacitor

Targeted Systems



Plan

Real time energy estimation

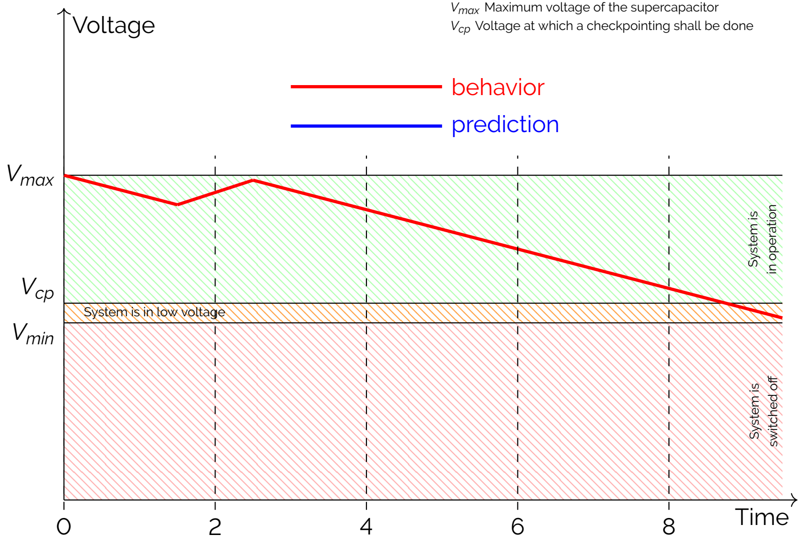
- System equipped with a supercapacitor
- Available energy in the capacitor stated with a relation on voltage
- Available energy depends on operating mode, clocked peripherals, microcontroller operations, etc.
- Estimate the operating time (the time before a checkpoint shall be done)
- May be predicted using a linear model
- Estimate a global slope (from different modules’ slope)
- Adjust the slopes dynamically at runtime

Remaining energy prediction

- By using the current voltage, V_c and the voltage drop slope over time, it is possible to predict when a checkpoint execution is required (When the voltage will cross V_{cp}).
- A timer is programmed so that an interruption (cp_{int}) occurs at this time.
- In addition, the exact voltage is read periodically to adjust the prediction. This can lead to a reprogramming of the timer.

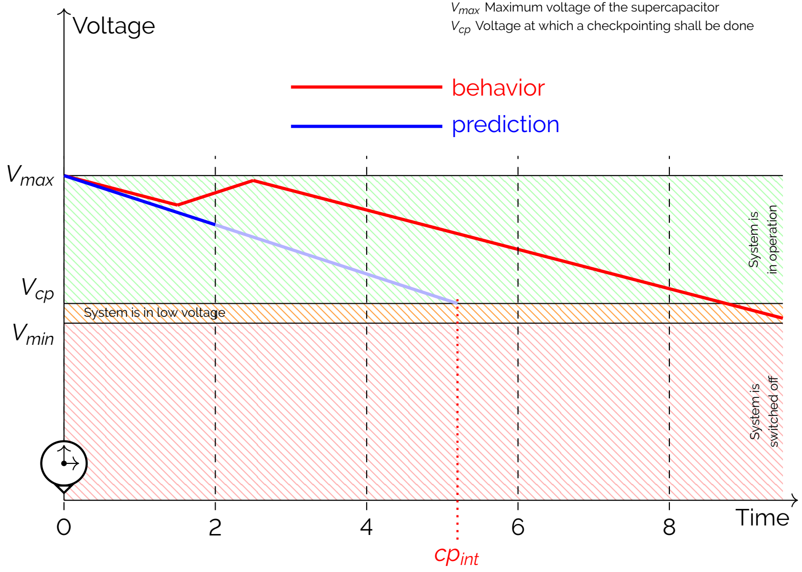
Operating time prediction

V_{min} Minimum voltage at which the microcontroller runs
V_{max} Maximum voltage of the supercapacitor
V_{cp} Voltage at which a checkpointing shall be done



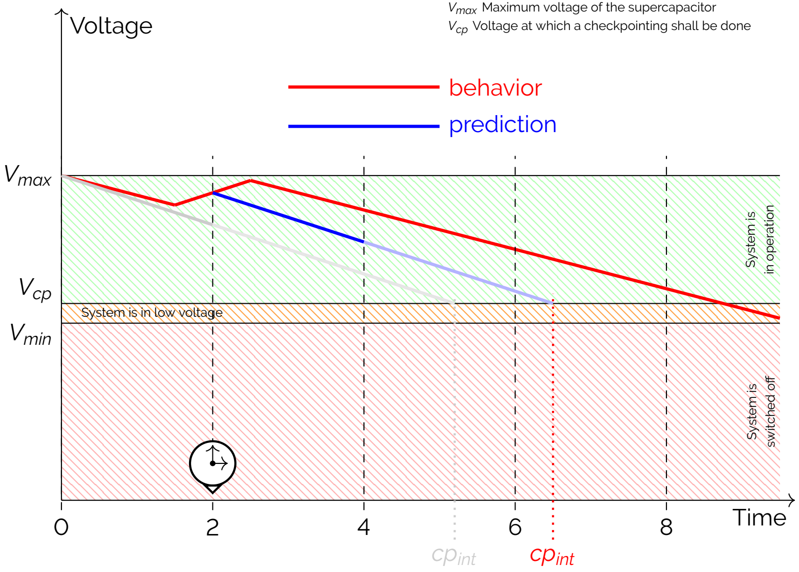
Operating time prediction

V_{min} Minimum voltage at which the microcontroller runs
V_{max} Maximum voltage of the supercapacitor
V_{cp} Voltage at which a checkpointing shall be done



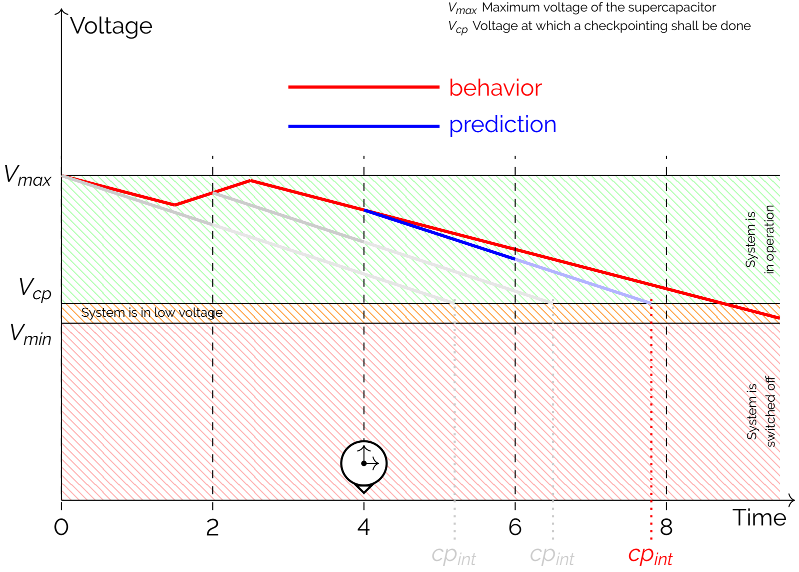
Operating time prediction

V_{min} Minimum voltage at which the microcontroller runs
V_{max} Maximum voltage of the supercapacitor
V_{cp} Voltage at which a checkpointing shall be done



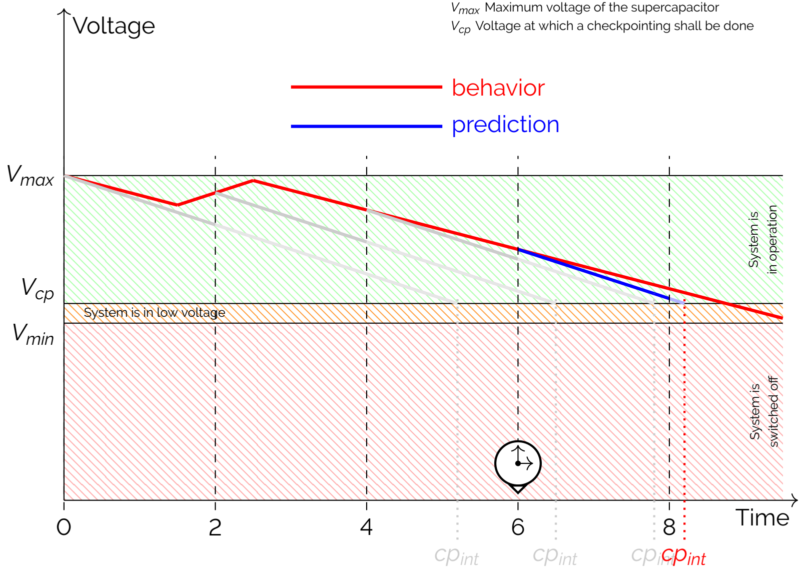
Operating time prediction

V_{min} Minimum voltage at which the microcontroller runs
V_{max} Maximum voltage of the supercapacitor
V_{cp} Voltage at which a checkpointing shall be done



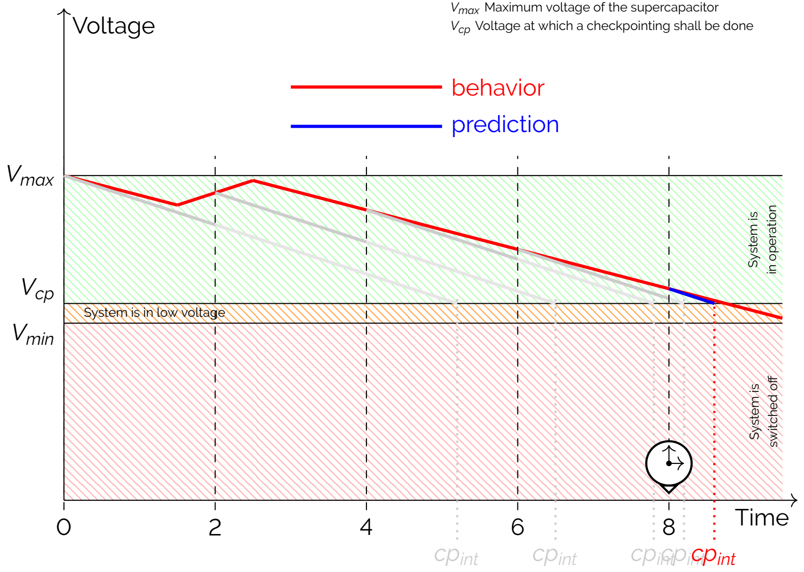
Operating time prediction

V_{min} Minimum voltage at which the microcontroller runs
V_{max} Maximum voltage of the supercapacitor
V_{cp} Voltage at which a checkpointing shall be done



Operating time prediction

V_{min} Minimum voltage at which the microcontroller runs
V_{max} Maximum voltage of the supercapacitor
V_{cp} Voltage at which a checkpointing shall be done



Plan

Trampoline

- Real-time Operating System, developed by the LS2N Real-time Systems Group
- Modular and portable
- Free Software License (GPLv2) + industrialized version
- Conforms to OSEK/VDX OS and AUTOSAR OS (fixed priority scheduling)
- Many targets including AVR, ARM-Cortex, PowerPC, Posix, MSP430
- Support for multicore architectures

Insights

Contributors	13 identified, but **4** in reality; Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, David Garriou
Commits	> 2600
Organization	No organization ! But I hope things will evolve

- First Trampoline sprint (à la Buildroot) on march.
- A lot of subjets to deal with; new features, build system, code generation, legacy code, documentation

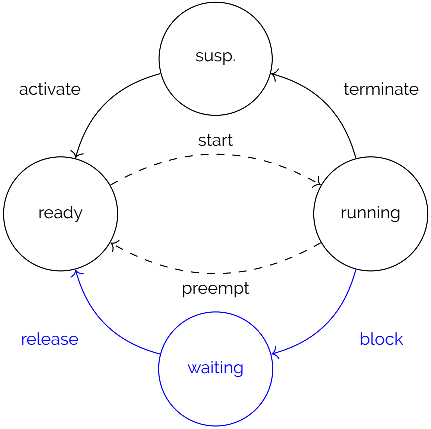
- As far as we know, used in industrial environment by two major undertaking

History

- Version 1
  - 2005: Start of development
  - 2007: AUTOSAR
  - 2008: Pass OSEK certification, AUTOSAR SC1&2
  - 2009: Integrated trace system
  - 2010: AUTOSAR SC3&4, MODISTARC & AUTOSAR test suite
- Version 2
  - 2013: Runtime Verification
  - 2014: Multicore
- Version 3
  - 2016: Hypervisor support (XVisor)
  - 2017: Runtime enforcement
  - 2018: Runtime verification on FPGA

Scheduling

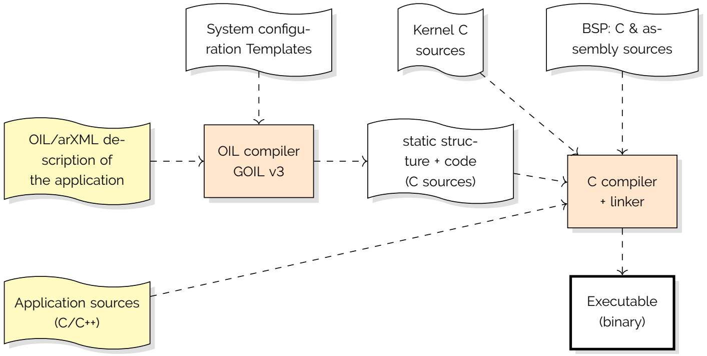
- Fixed priority scheduling
- Basic task : may not synchronize
- Extended task : may synchronize
- Critical sections use the ICPP protocol (no deadlock no priority inversion)
- Each task can be configured ad preemptable or non-preemptable



Trampoline is a static system

The configuration of Trampoline uses a dedicated language

- OIL or arXML
- extensible (based on a template engine)


System configuration Templates
Kernel C sources
BSP: C & assembly sources
OIL/arXML description of the application
OIL compiler GOIL v3
static structure + code (C sources)
C compiler + linker
Application sources (C/C++)
Executable (binary)

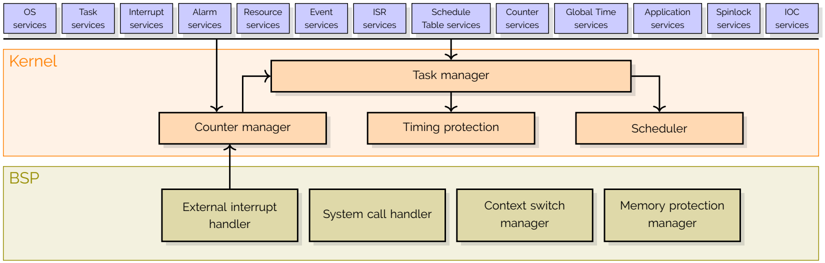
Example of OIL description (excerpt)

```
ALARM one_second {
   COUNTER = SystemCounter;
   ACTION = ACTIVATETASK { TASK = t0; };
   AUTOSTART = TRUE { APPMODE = std; ALARMTIME = 100; CYCLETIME = 100; };
};

TASK t0 {
   AUTOSTART = FALSE;
   PRIORITY = 3;
   ACTIVATION = 1;
   SCHEDULE = FULL;
   MESSAGE = s00;
};

MESSAGE s00 {
   MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
         CDATATYPE = "uint8";
    };
   NOTIFICATION = NONE;
};
```

Architecture



Some services

Service	Prototype	Description
OS services		
StartOS	*app_mode*	Starts the operating system
ShutdownOS	*app_mode*	Stops the operating system
Task services		
ActivateTask	*task_id*	Activates task *task_id*
TerminateTask		Terminates the caller
Alarm services		
SetRelAlarm	*alarm_id*, *offset*, *cycle*)	Starts alarm *alarm_id*
SetAbsAlarm	*alarm_id*, *date*, *cycle*)	Starts alarm *alarm_id*
CancelAlarm	*alarm_id*	tops alarm *alarm_id*
GetAlarm	*alarm_id*, *&remaining*)	Gets alarm state
Resource services		
GetResource	*rez_id*	The caller enters critical section
ReleaseResource	*rez_id*	The caller leaves critical section
Event services		
WaitEvent	*ev*	The caller waits for event *ev*
SetEvent	*task_id*, *ev*	Set event *ev* to task *task_id*
ClearEvent	*ev*	Clear event *ev* of the caller
GetEvent	*task_id*, *&ev*	Puts a copy of events received by task *task_id* in *ev*

Plan

Transcient computing platform with MSP430

Port of MSP430 to Trampoline

- Two boards; MSP430FR5969, MSP430FR5994 Launchpads
- Still in progress, not completely tested
- We need to better integrate it in Trampoline (e.g. templates)

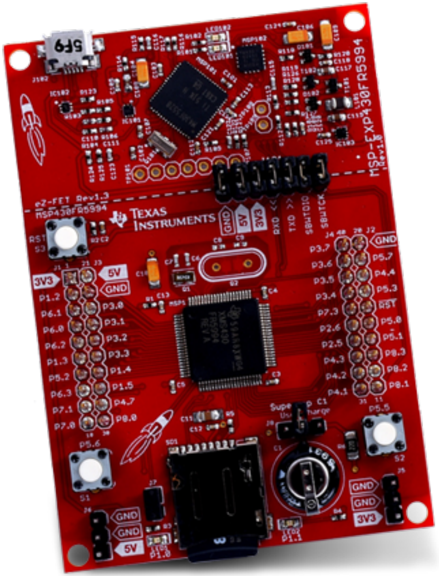
New services

Service	Prototype	Description
OS services		
RestartOS		Starts the operating system from a checkpoint
Hibernate		Terminates the current task and records a checkpoint in an available FRAM slot

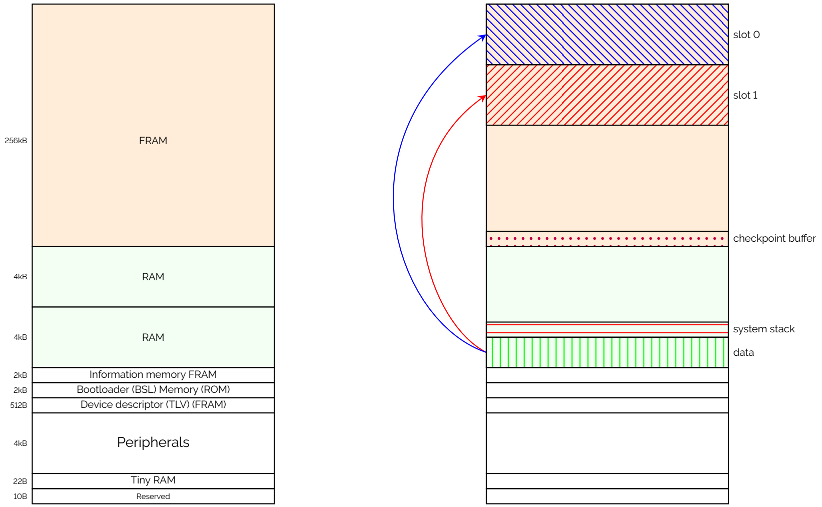
(MSP/EXP)430FR5994

- 16 bits micro-controller
- 16 registers 20 bits wide (CPUX)
- 16MHz
- 4kB + 4kB SRAM
- 512B + 256kB FRAM
- $118 \mu A / MHz$ active mode, Peripheral low-frequency (LPM3)
- 0.22F supercapacitor. Works x minutes with blink.

(MSP/EXP)430FR5994



MSP430FR5994 memory mapping



Status of work

Energy

- Remaining energy prediction strategy not implemented
- We periodically read the voltage
- Commit a checkpoint if voltage value is below a threshold

Checkpoints

- Two checkpoint slots, a mark selects the available checkpoint slot
- Copy a part of RAM to a slot in FRAM
- Exclude the system stack

Restart

- When energy comes back
- Get back from FRAM the last committed checkpoint

Future work

- Implement the strategy for operating time estimation, dynamically adjust the slopes
- Do not store unnecessary data, higher level RAM management
- Restore the state of some peripherals
- Thinking with the “normally off, instantly on” paradigm
- Integrate this paradigm in the configuration of Trampoline.
  - Checkpointing some tasks, not all
  - A task may specify the energy it would need to surely complete, a kind of worst case execution energy, the Worst Case Voltage Drop.

References

Trampoline	`https://github.com/TrampolineRTOS/trampoline`
Work	Work in progress on branch `checkpointing`
Documentation	Available on `checkpointing` branch, `documentation/manual/msp430/`