Programming with Time for Mixed Criticality Systems

Dagstuhl Seminar, March 16-20, 2015
Mixed Criticality on Multicore/Manycore Platforms

David Broman

Associate Professor, KTH Royal Institute of Technology

Assistant Research Engineer, University of California, Berkeley

Slide 9 was updated July 8, 2015.

What is mixed criticality?

Mixed-Criticality Systems (MCS) Challenge

Reconcile the conflicting requirements of:

- Partitioning (for safety assurance)
- Sharing (for efficient resource usage)

(Burns & Davis, 2013)

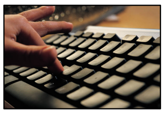
This talk focuses on the **time** and **timing** aspects of the problem

Mixed **Time**-Critical Systems

Other aspects are equally important (hardware failures, network aspects etc.), but are not considered here.

Viewpoints on the MCS timing aspect

Viewpoint I
The Implementation View

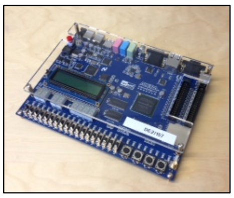


- **Software Scheduling**
  Vestal's model (and variants thereof) with different WCET numbers for different criticality levels.
- **Hardware Scheduling**
  For instance, the FlexPRET approach (Zimmer et. al 2014) with predictable and less predictable hardware threads.

Viewpoint II
The Specification View

- **A Task Model with Bounded Frequencies**
  Yip et al. (2014) on relaxed the synchronous approach for MSC.
- **Programming with Time**
  Express **timing constraints** and **fault handling** explicitly in a programming language.

Hardware Scheduling with FlexPRET



FlexPRET Softcore

Fine-grained Multithreaded Processor Platform (thread interleaved) implemented on an FPGA

Flexible schedule (1 to 8 active threads) and scheduling frequency (1, 1/2, 2/3, 1/4, 1/8 etc.)

Hard real-time threads (HRTT) with predictable timing behavior
- Thread-interleaved pipeline (no pipeline hazards)
- Scratchpad memory instead of cache

Soft real-time threads (SRTT) with cycle stealing from HRTT

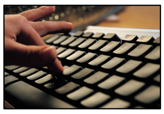
Note: Not limited to 8 tasks. Can schedule several tasks on the same hardware thread using software scheduling.

Open Source:
https://github.com/pretis/flexpret

Zimmer, Broman, Shaver, and Lee. "FlexPRET: A Processor Platform for Mixed-Criticality Systems" (RTAS 2014)

Viewpoints on the MCS timing aspect

Viewpoint I
The Implementation View



- **Software Scheduling**
  Vestal's model (and variants thereof) with different WCET numbers for different criticality levels.
- **Hardware Scheduling**
  For instance, the FlexPRET approach (Zimmer et. al 2014) with predictable and less predictable hardware threads.

Viewpoint II
The Specification View

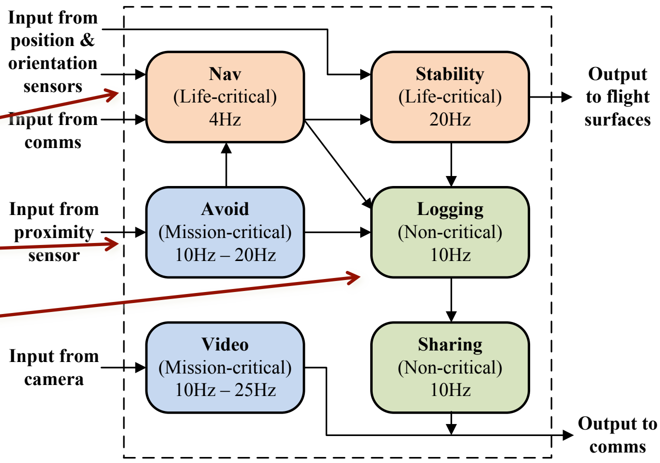
- **A Task Model with Bounded Frequencies**
  Yip et al. (2014) on relaxed the synchronous approach for MSC.
- **Programming with Time**
  Express **timing constraints** and **fault handling** explicitly in a programming language.

A Task Model With Bounded Frequencies

Each periodic task has two frequency parameters: f_{max} and f_{min}.

- **Life Critical Tasks**
  $f_{max} = f_{min}$.
- **Mission Critical Tasks**
  $f_{max} > f_{min}$.
- **Non-Critical Tasks**
  f_{max} is the goal. $f_{min} = 0$

Example: Unmanned Aerial Vehicle (UAV)

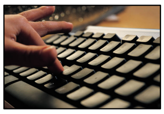


Note:
The task model does not say <u>anything</u> about the implementation technique or WCETs for specific platforms.

Eugene, Kuo, Roop, and Broman. "Relaxing the Synchronous Approach for Mixed-Criticality Systems" (RTAS 2014)

Viewpoints on the MCS timing aspect

Viewpoint I
The Implementation View



- **Software Scheduling**
  Vestal's model (and variants thereof) with different WCET numbers for different criticality levels.
- **Hardware Scheduling**
  For instance, the FlexPRET approach (Zimmer et. al 2014) with predictable and less predictable hardware threads.

Viewpoint II
The Specification View

- **A Task Model with Bounded Frequencies**
  Yip et al. (2014) on relaxed the synchronous approach for MSC.
- **Programming with Time**
  Express **timing constraints** and **fault handling** explicitly in a programming language.

Programming with Time

Motivation

- **Timing Specification:** Be able to describe different task models within one framework
- **Formal:** To have an unambiguous formal semantics with precise meaning
- **Fault handling**: Be able to express precise run-time behaviors when e.g. deadlines are missed.

Some related work

- Giotto by Henzinger et al. (2001)
- Ptides by Eidson et al. (2012)
- Timing constraint logic by Lisper and Nordlander (2012)
- Synchronous approach for MSC by Cohen et al. (2015)

A Timed Lambda Calculus (unpublished work)

Syntax

Variables	$x, y \in \mathbb{X}$
Constants	$c \in \mathbb{C}$
Time	$t \in \mathbb{N} \cup \infty$
Expressions	$e ::= x \mid \lambda x.e \mid e\ e \mid c \mid \texttt{overrun} \mid \texttt{time} \mid \textbf{within}\ t\ \textbf{to}\ t\ \textbf{do}\ e\ \textbf{else}\ e$
Values	$v ::= \lambda x.e \mid c$
Frames	$F ::= \Box\ e \mid v\ \Box \mid \textbf{within}\ t_1\ \textbf{to}\ t_2\ \textbf{do}\ \texttt{overrun}\ \textbf{else}\ \Box$

Dynamic Semantics

$$\frac{\delta(c, v, s, t) = (v', s', t') \qquad \nexists d \in D.\, t' > d}{c\ v \mid s, t, D \longrightarrow v' \mid s', t'}\ \text{(E-DELTA)} \qquad (\lambda x.e)v \mid s, t, D \longrightarrow [x \mapsto v]e \mid s, t \quad \text{(E-BETA)}$$

$$\frac{\delta(c, v, s, t) = (v', s', t') \qquad \exists d \in D.\, t' > d}{c\ v \mid s, t, D \longrightarrow \texttt{overrun} \mid s', t'}\ \text{(E-OVERRUN)} \qquad \texttt{time} \mid s, t, D \longrightarrow t \mid s, t \quad \text{(E-TIME)}$$

$$\textbf{within}\ t_1\ \textbf{to}\ t_2\ \textbf{do}\ v\ \textbf{else}\ e \mid s, t, D \longrightarrow v \mid s', max(\{t, t + t_1\}) \quad \text{(E-WITHIN)}$$

$$\textbf{within}\ t_1\ \textbf{to}\ t_2\ \textbf{do}\ \texttt{overrun}\ \textbf{else}\ v \mid s, t, D \longrightarrow v \mid s, t \quad \text{(E-OVERRUN-HANDLING)}$$

$$\frac{e_1 \mid s, t, D \cup \{t + t_2\} \longrightarrow e_1' \mid s', t'}{\textbf{within}\ t_1\ \textbf{to}\ t_2\ \textbf{do}\ e_1\ \textbf{else}\ e_2 \mid s, t, D \longrightarrow \textbf{within}\ t_1 - t' + t\ \textbf{to}\ t_2 - t' + t\ \textbf{do}\ e_1'\ \textbf{else}\ e_2 \mid s', t'}\ \text{(E-CONG-WITHIN)}$$

$$\frac{e \mid s, t, D \longrightarrow e' \mid s', t'}{F[e] \mid s, t, D \longrightarrow F[e'] \mid s', t'}\ \text{(E-CONG)} \qquad F[\texttt{overrun}] \mid s, t, D \longrightarrow \texttt{overrun} \mid s, t \quad \text{(E-OVERRUN-PROP)}$$

The `within` construct

Lower timing bound for a specific resolution (e.g., microseconds)

Upper timing bound (to be verified statically and checked at runtime)

Computation to be done within the bound.

Fault handling if a deadline is missed

$$\textbf{within}\ 5\ \textbf{to}\ 10\ \textbf{do}\ e_1\ \textbf{else}\ e_2$$

Constructs can be **nested**

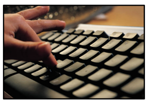
In this case, specifies the timing bounds for releases.

$\textbf{within}\ 5\ \textbf{to}\ 10\ \textbf{do}$
$\quad \textbf{within}\ 0\ \textbf{to}\ 3\ \textbf{do}\ ()\ \textbf{else}\ ();$
$\quad computation()$
$\textbf{else}$
$\quad errorHandling()$

Construction can be put within loops or have conditions.

Conclusions

Some key take away points:



- **Implementation view of MCS**
  - Software Scheduling
  - Hardware Scheduling

- **Specification view of MCS**
  - Bounded Frequencies Task Model
  - Programming with Time

Thanks for listening!

David Broman
dbro@kth.se