Programming Mixed Critical and Weakly Hard Real-Time Systems: It's about time

CASTOR Workshop on Dependable and Secure Systems
December 9, 2020

David Broman
Associate Professor, KTH Royal Institute of Technology
Associate Director Operations, Digital Futures

Acknowledgement (involved people)

Timed C
Saranya Natarajan
Mitra Nasri
Björn B. Brandenburg
Geoffrey Nelissen
David Broman

FlexPRET
Michael Zimmer
David Broman
Chris Shaver
Edward A. Lee

SPM Management
Yooseong Kim
David Broman
Jian Cai
Aviral Shrivastava

digital futures

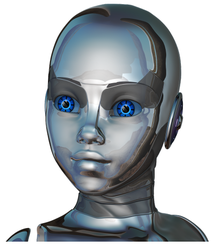
WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Financially supported by the Swedish Foundation for Strategic Research.

With funding from: VINNOVA Sweden's Innovation Agency

Vetenskapsrådet (VR)

Research Group and Areas



Machine Learning

Intersection of Research Areas

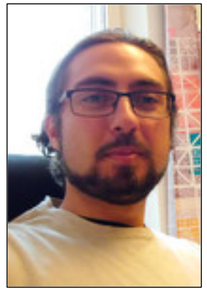
Cyber-Physical and Real-Time Systems

$\lambda^{<?>}$

Programming Languages and Compilers

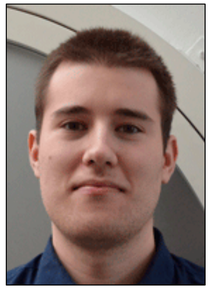
David Broman PI

Elias Castegren Postdoc



Mauricio Chimento Postdoc

Saranya Natarajan PhD Student



Daniel Lundén PhD Student

Viktor Palmkvist PhD Student

Linnea Ingmar PhD Student

Gizem Çaylak PhD Student

Oscar Eriksson PhD Student

Lars Hummelgren PhD Student

Joey Öhman Student Engineer

Programming Mixed Critical and Weakly Hard Real-Time Systems: It's about time

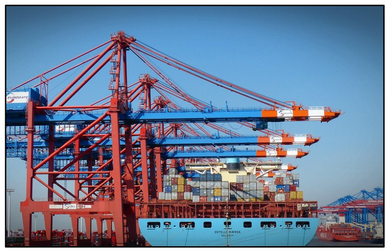
David Broman, KTH Royal Institute of Technology, dbro@kth.se

Cyber-Physical and Real-Time Systems

Autonomous Vehicles

Smart Industrial Automation

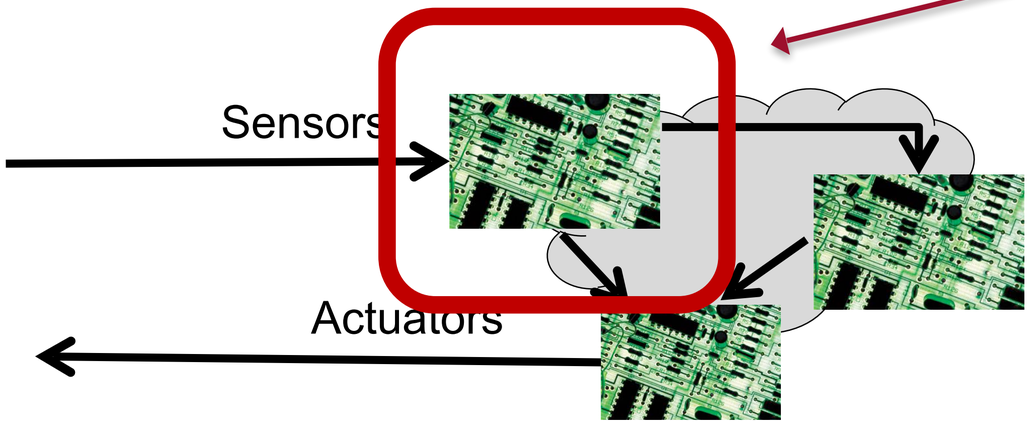
Satellites



Container Automation

Real-time systems, running on embedded platforms

System



Physical system (the plant)

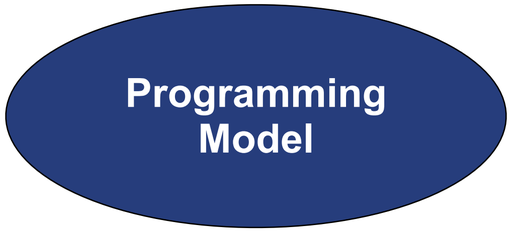
Cyber system: Computation (embedded) + Networking

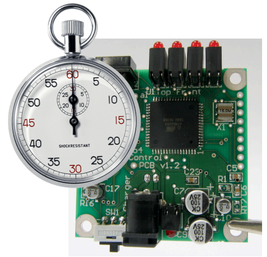
David Broman, KTH Royal Institute of Technology, dbro@kth.se

Programming Model and Time

Timing is not part of the software semantics

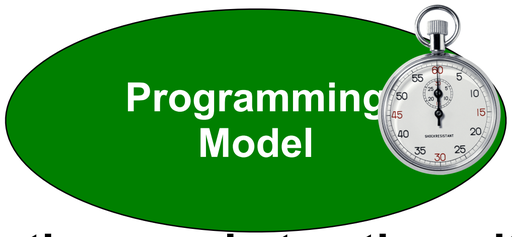
Traditional Approach



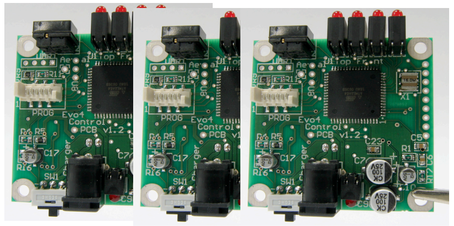


Timing Dependent on the Hardware Platform

Our Objective



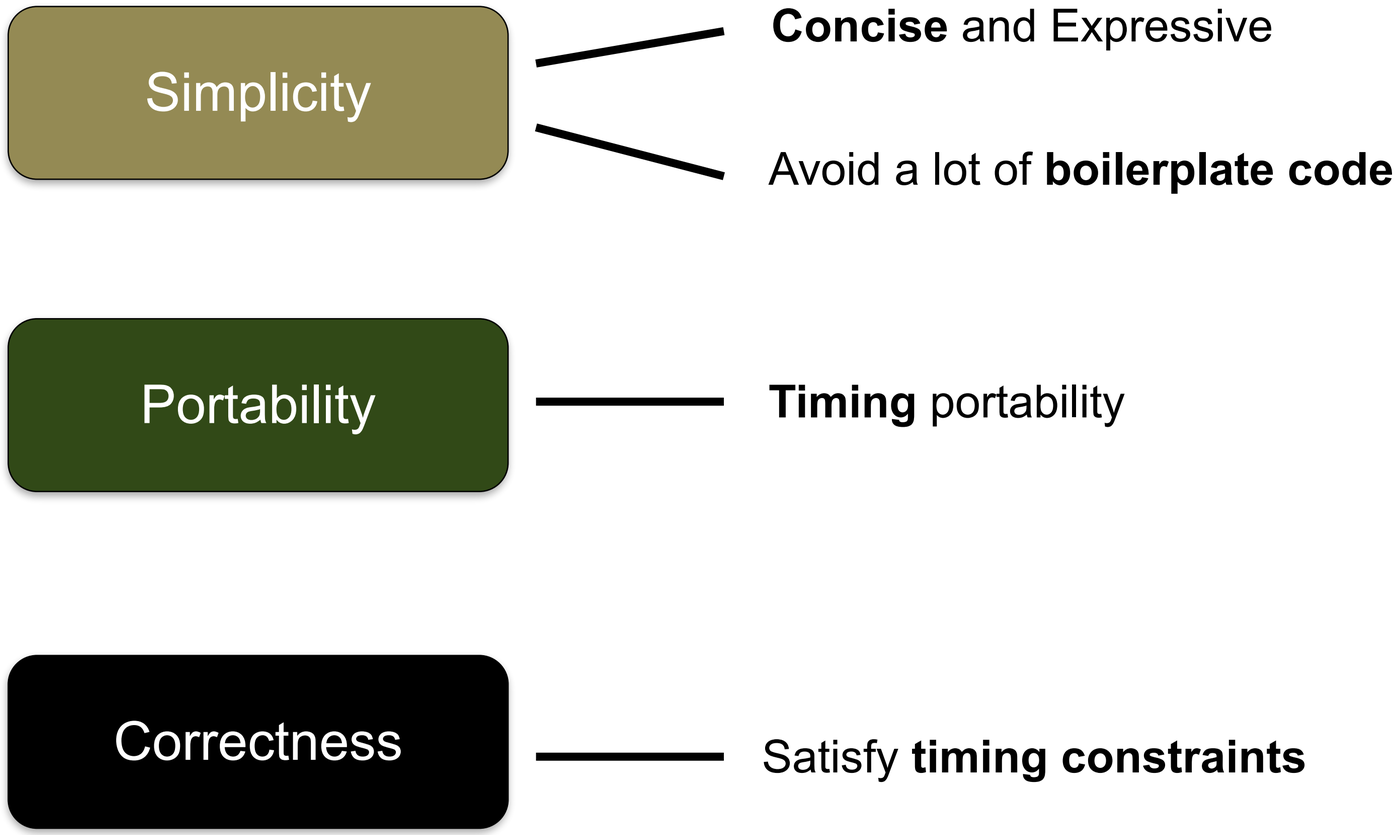
Make time an <u>abstraction</u> <u>within</u> the programming model



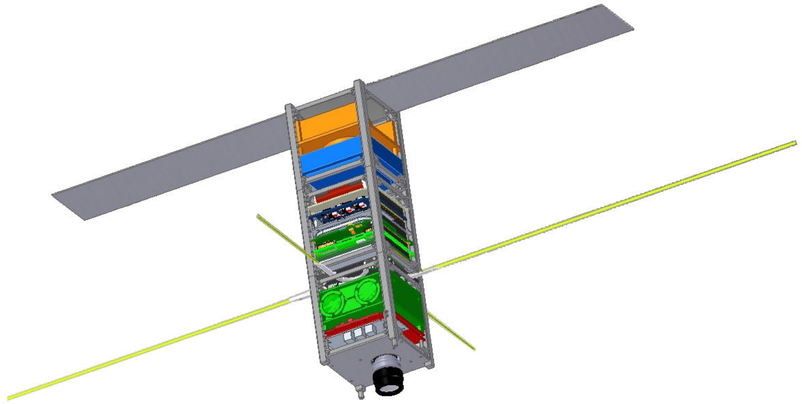
Timing is independent of the hardware platform (within certain constraints)

David Broman, KTH Royal Institute of Technology, dbro@kth.se

Programming with Time: Design Objectives and Challenges



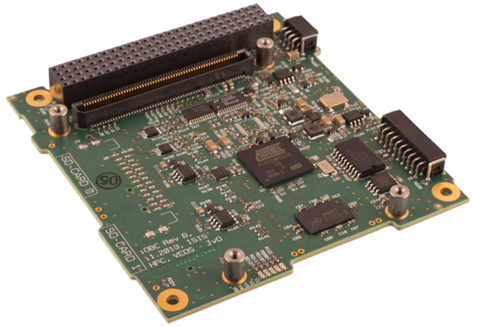
MIniature STudent Satellite (MIST)



- 3U Cubesat (≤ 4 kg).
- Sven Grahn (Project Leader)
  Christer Fuglesang (KTH Space Center)
- Built by students at KTH
- 7 scientific experiments on board
- On-board computer (OBC) coordinates experiments and communicates with the ground station on earth.

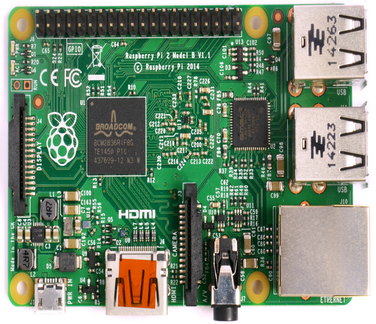
David Broman, KTH Royal Institute of Technology, dbro@kth.se

Motivation for Timing Portability



Flight qualified OBC hardware (iOBC)

OBC flight software



Raspberry Pi

exp1 exp2 exp7

State of The Art

Safety Critical

- Synchronous languages (Lustre, Esterel, Singal). SCADE commercial success.
- Timed research frameworks (PTIDES and Giotto)
- Modeling languages (Modelica, UML MARTE, Simulink, Ptolemy, Labview etc.)
- Formalisms for verification (Process algebras with time, Timed Automata etc.)
- Automation (PLC standard IEC 61131-3, e.g. structured text)
- Other specialized languages (Real-Time Euclide and PEARL)

Ada

Safety Critical

- Direct primitives for Real-time
- Delay primitives (`delay`, `delay_until`)
- Firm deadlines using `select-abort` and use of interrupts

Real-time extension to Java (RTJS)

- Direct primitives for Real-time
- Instances of special classes (`RealtimeThread`)

C language

Commodity Real-Time

- C programs using RTOS APIs (e.g. FreeRTOS, Vx
- Arduino programs
- Real-time concurrent C
- Real-time POSIX C
  - Pthreads
  - clock_nanosl
  - timers

Simple but limited

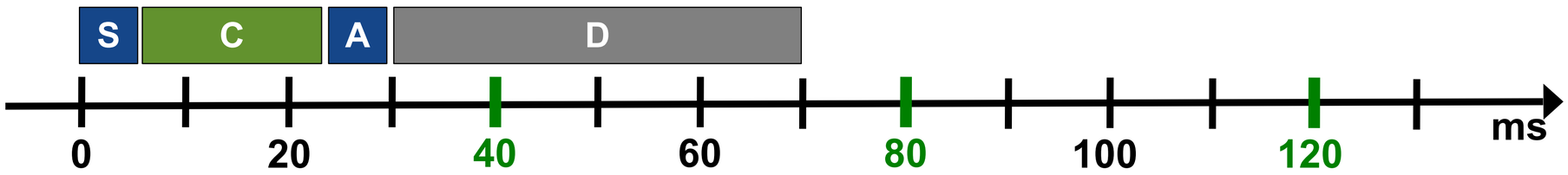
Expressive but verbose

Simplicity and Correctness? (Arduino)

A naïve implementation of a periodic loop in Arduino. What is the problem with this approach?

```
void loop() {
  sense();
  compute();
  actuate();
  delay(40);
}
```

There is a drift.



Simplicity and Correctness? (POSIX C)

Same, but with firm deadline

```
 1:   /*Code using POSIX API*/
 2:int waiting_for_signal;
 3:jmp_buf env;
 4:void timer_signal_handler(int sig, siginfo_t*
       extra, void* cruft){
 5:   if(waiting_for_signal == 1){
 6:     siglongjmp(env, 3);
 7:   }
 8:   waiting_for_signal = 0;
 9:}
10:void main(){
11:   struct timespec start_time, interval_timespec;
12:   long interval;
13:   char* unit;
14:   int  ret_jmp;
15:   struct itimerspec i;
16:   struct sigaction sa;
17:   struct sigevent timer_event;
18:   timer_t mytimer;
19:   convert_to_timespec(&interval_timespec,3,"ms");
20:   sa.sa_flags = SA_SIGINFO;
21:   sa.sa_sigaction = timer_signal_handler;
22:   if(sigaction(SIGRTMIN, &sa, NULL) < 0){
23:     perror("sigaction");
24:     exit(0);
25:   }
26:   timer_event.sigev_notify = SIGEV_SIGNAL;
27:   timer_event.sigev_signo = SIGRTMIN;
28:   timer_event.sigev_value.sival_ptr=(void*)&
          mytimer;
29:   if(timer_create(CLOCK_REALTIME,&timer_
          mytimer)<0){
30:     perror("timer_create");
31:     exit(0);
32:   }
33:   clock_gettime(CLOCK_REALTIME,&start_time);
34:   add_timespec(&(i.it_value ), start_time,
          interval_timespec);
35:   i.it_interval.tv_sec = 0;
36:   i.it_interval.tv_nsec = 0;
37:   if(timer_settime(mytimer, TIMER_ABSTIME, &i,
          NULL) < 0 ){
38:     perror("timer_setitimer");
39:     exit(0);
40:   }
41:   while(1){
42:     ret_jmp = sigsetjmp(env, 1);
43:     waiting_for_signal = 1;
44:     if(ret_jmp == 0){
45:       sense(); //read from sensor
46:     }
47:     waiting_for_signal = 0;
48:     clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME
            ,&i.it_value, NULL);
49:     add_timespec(&(i.it_value ), i.it_value,
            interval_timespec);
50:     i.it_interval.tv_sec = 0;
51:     i.it_interval.tv_nsec = 0;
52:     timer_settime(mytimer, TIMER_ABSTIME, &i,
            NULL);
53:   }
54:}
```

Timed C – A Timed Extension to C

```
task foo(){
  while(1){
    sense();
    compute();
    actuate();
    fdelay(40, ms);
  }
}
```

Simplicity

As simple as Arduino

Portability

Source-to-source compiler. Different target RTOSs

Correctness

Timing verification (End-to-end tool chain)

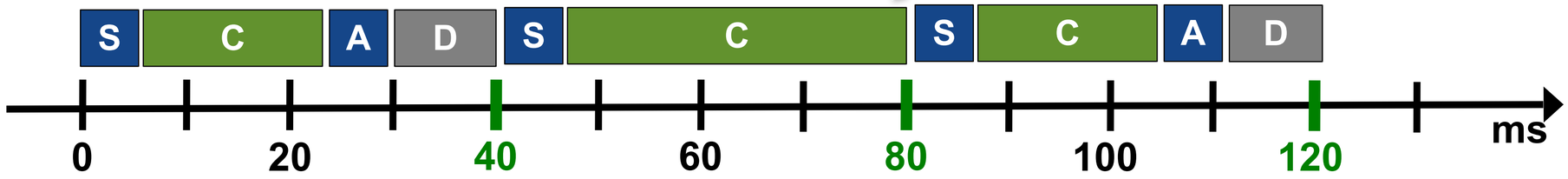
Open Source

https://github.com/timed-c/

Delay, but not more than to 40ms

Interrupt if overrun (critical section is coming up)

Natarajan and Broman (RTAS 2018)



David Broman, KTH Royal Institute of Technology, dbro@kth.se

Firm delay revisited

```
 1:void computePath(int* a){
 2:   int b[100];
 3:   initialize(a);
 4:   while(1){
 5:     computeAnytime(b, a);
 6:     critical{
 7:       memcpy(a, b, 100);
 8:     }
 9:   }
10:   fdelay(100, ms);
11:}
```

Initialize computation

Compute a new, improved value

Protect copying using critical section

Firm delay. Note, outside while loop

Programming Mixed Critical and Weakly Hard Real-Time Systems: It's about time

David Broman, KTH Royal Institute of Technology, dbro@kth.se

Weakly Hard Real-time Systems

Soft Deadline

Run to completion, utility even if overrun

Firm Deadline

No utility if overrun, Abort execution at deadline

Hard Deadline

Deadlines must not be missed

Weakly Hard Real-Time Deadlines

(m,k) model

Originally by Bernat, et al. (2001)

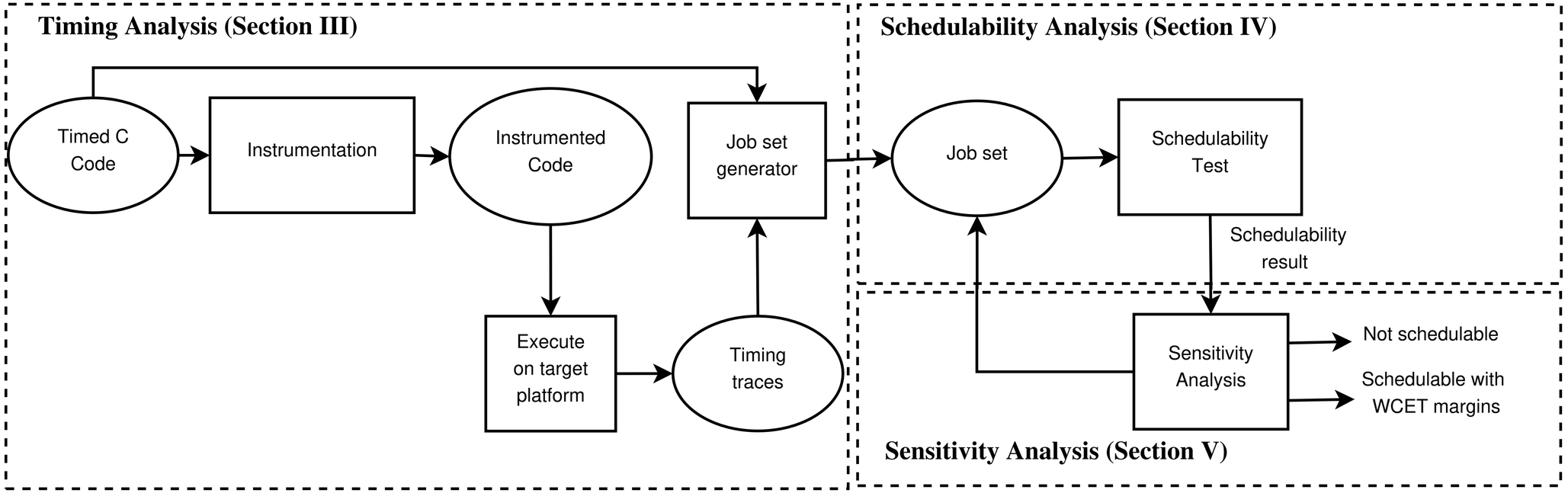
m number of deadline misses

in a **k** window

(Original definition, m is the number of met deadlines)

Idea: (m,k) - Sensitivity Analysis
To find the strongest still-satisfied (m, k) constraint (or general weakly-hard constraint).

(m,k) Sensitivity Analysis



Given a window size of (k)

Find all WCET scaling factors where m inverse changes

Natarajan, Nasri, Broman, Brandenburg, and Nelissen (RTSS 2019)

Example

Alpha value	WCET Margin	Total misses (sum of tasks)
1.11	11%	0
1.25	25%	1
2.30	130%	2

Modern Systems with Many Processor Platforms

Aerospace

Modern aircraft have many computer controlled systems

- Engine control
- Electric power control
- Radar system
- Navigation system
- Flight control
- Environmental control system

etc…

Automotive

Modern cars have many ECU (Electronic Control Units)

- Airbag control
- Door control
- Electric power steering control
- Power train control
- Speed control
- Battery management.

etc.. Over 80 ECUs in a high-end model (Albert and Jones, 2010)

Programming Mixed Critical and Weakly Hard Real-Time Systems: It's about time

David Broman, KTH Royal Institute of Technology, dbro@kth.se

Mixed-Criticality Systems

Issues with too many processors

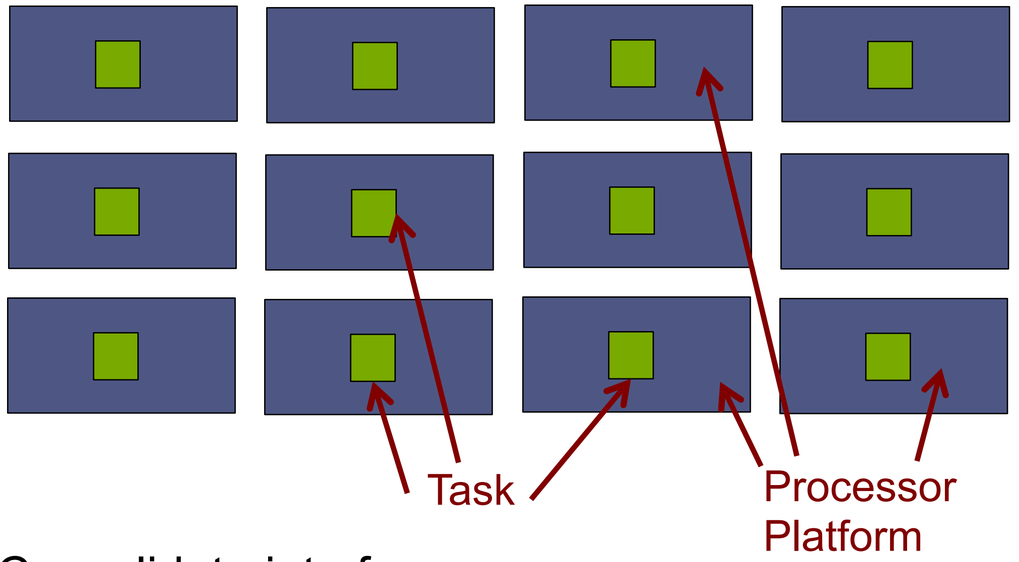
- High cost
- Space and weight
- Energy consumption

Required for Safety

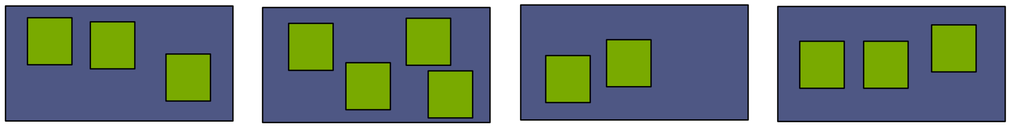
- Spatial isolation between tasks
- Temporal isolation between tasks (necessary to meet deadlines)

Federated Approach

Each processor has its own task



Consolidate into fewer processors



Mixed-Criticality Systems

Issues with too many processors

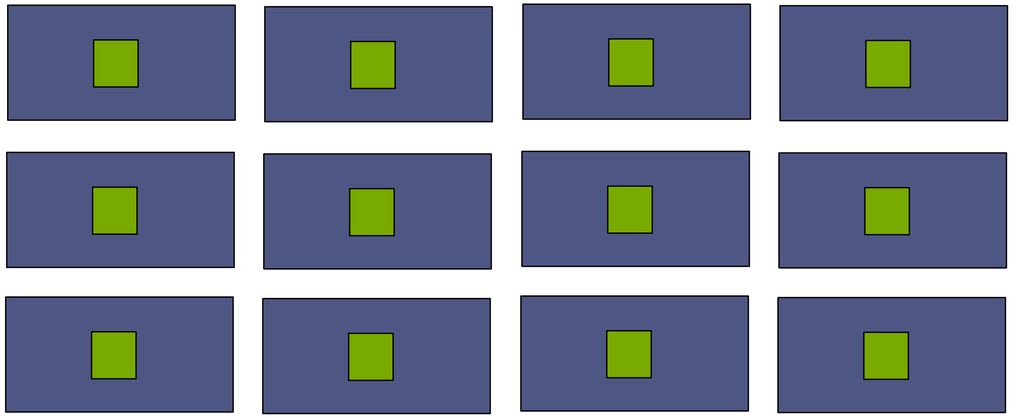
- High cost
- Space and weight
- Energy consumption

Required for Safety

- Spatial isolation between tasks
- Temporal isolation between tasks (necessary to meet deadlines)

Federated Approach

Each processor has its own task



…but such safety requirements are only needed for highly critical tasks

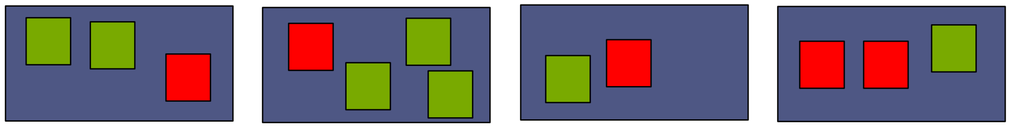
Mixed-Criticality Challenge

Reconcile the conflicting requirements of:

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

(Burns & Davis, 2013)

Consolidate into fewer processors

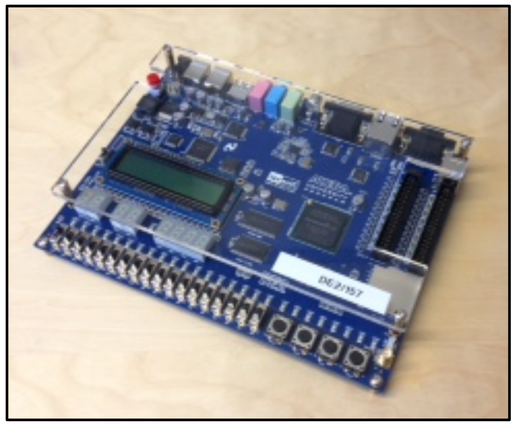


Hardware and Compiler Solutions

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 pipleine (no pipeline hazards)
  - Scratchpad memory instead of cache
- Soft real-time threads (SRTT) with cycle stealing from HRTT

Zimmer, Broman, Shaver, and Lee (RTAS 2014)



WCET-Aware Scratchpad Memory (SPM) Management

- Automatic DMA transfer of code to SPM
- Optimal mapping for minimizing WCET

Kim, Broman, Cai, and Shrivastava (RTAS 2014, TECS 2017)

Conclusions

Some key take away points:

- Timed C is an experimental research language, with the design goals of simplicity, portability, and correctness
- An (m,k)-sensitivity analysis is potentially a practical alternative to exact (sometimes impossible) timing analysis
- FlexPRET and Software Managed Scratchpad Memories are potential approaches to achieve better timing predictability

Thanks for listening!

Saranya Natarajan and David Broman. **Timed C: An Extension to the C Programming Language for Real-Time Systems**. In the *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2018)*, Porto, Portugal, IEEE, 2018. **(Outstanding Paper Award)**

Michael Zimmer, David Broman, Chris Shaver, and Edward A. Lee. **FlexPRET: A Processor Platform for Mixed-Criticality Systems**. Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS 2014), Berlin, Germany, April 15-17, 2014.

Saranya Natarajan, Mitra Nasri, David Broman, Björn B. Brandenburg, and Geoffrey Nelissen. **From Code to Weakly Hard Constraints: A Pragmatic End-to-End Toolchain for Timed C**. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS 2019)*, ACM, 2019.

Yooseong Kim, David Broman, Jian Cai, and Aviral Shrivastava. **WCET-Aware Dynamic Code Management on Scratchpads for Software-Managed Multicores.** Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS 2014), Berlin, Germany, April 15-17, 2014.