Automated generation of time-predictable executables on multi-core hardware

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

Claire Pagetti, Julien Forget, Heiko Falk, Dominic Oehlert, and Arno Luppold

October 10th 2018

RTNS 2018

ONERA
THE FRENCH AEROSPACE LAB

TUHH
Technische Universität Hamburg-Harburg

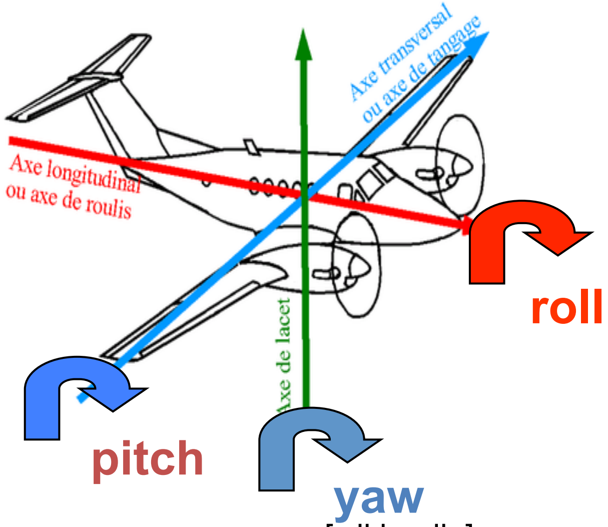
Outline

- ❑ **Introduction**
- ❑ Contribution
- ❑ Conclusion

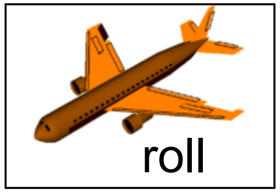
Context – control/command applications

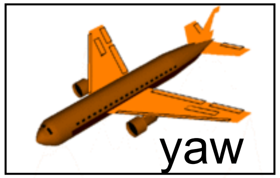
- Control / command applications
  - Safety-critical with DAL – Design Assurance Level A
  - Under certification, and certification development process

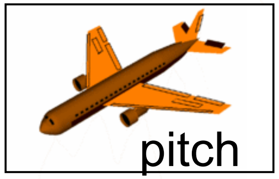
- Example: flight control system


Axe transversal ou axe de tangage
Axe longitudinal ou axe de roulis
Axe de lacet
roll
pitch
yaw

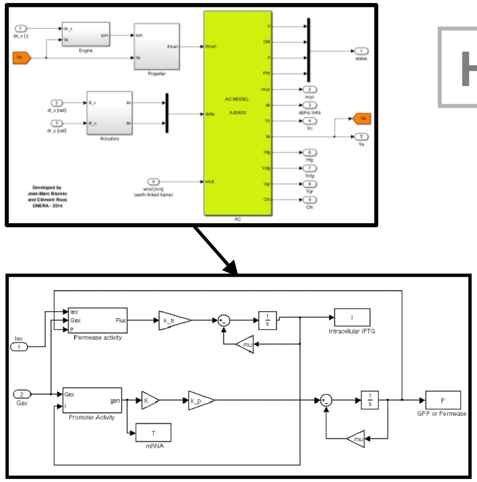
[wikipedia]







Current development cycle



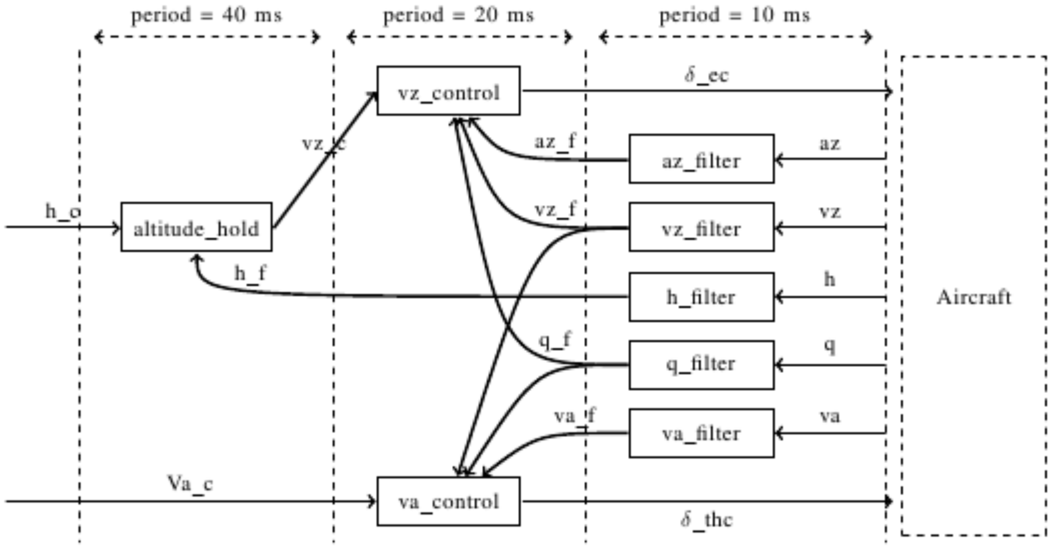
High-level design – control engineering

Implementation

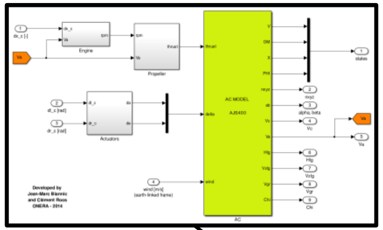
- Steps:
  - Coding of elementary blocks: Lustre/Scade
  - Coding of multi-periodic assemblies: ad hoc

- **Example: flight control systems**

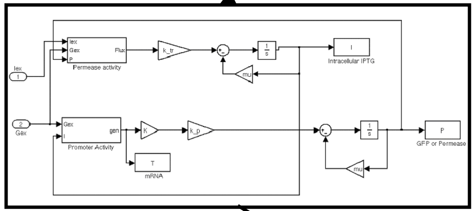
  multi-periodic, large size, under temporal and precedence constraints.



Current development cycle



High-level design – control engineering



Implementation

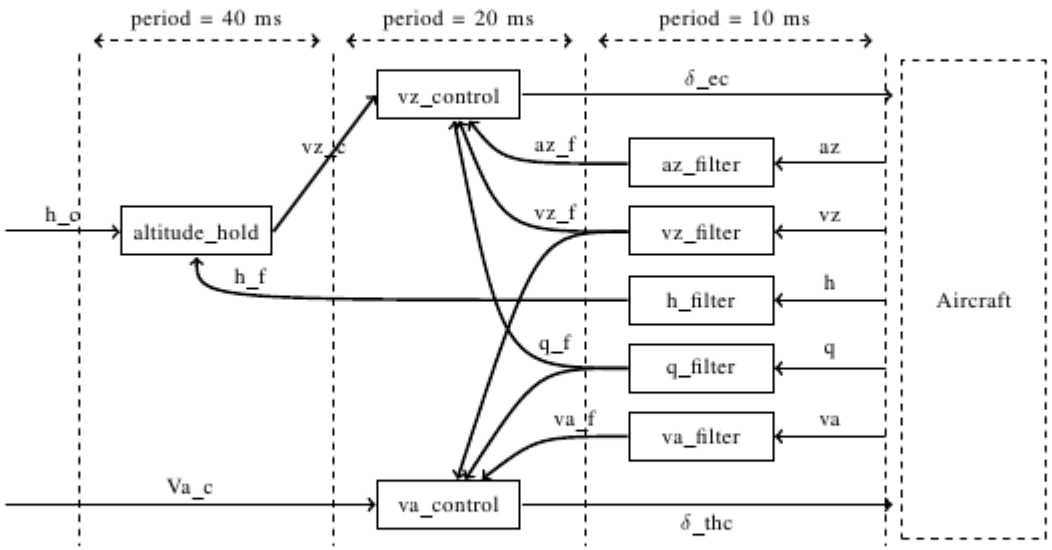
(Mono processor)

Integration on the target

- Steps:
  - Code generation:
    - Scade ➔ C: KCG
    - ad hoc ➔ scheduling + C
    - C ➔ executable: gcc
  - WCET: aiT from Absint

Prelude – multi-periodic language

- ❑ Synchronous language

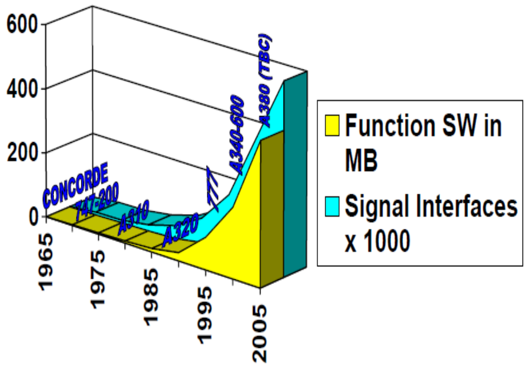


```
imported node h_filter (h :real)
returns (h_f :real) wcet 25;
…
node assemblage (h_c : real rate(100,0)
                                  Va_c : real rate(100
 returns ( delta_x_c  ,   delta_e_c )
var vz_c, va,  az, q, vz , va_f, vz_f,
        az_f , q_f :real;
 let
                  va_f  = va_filter(va/^ 2) ;
                  delta_x_c = va_speed_control(Va_c/^ 20 , va_f/^ 2 ,q_f/^ 2 ,vz_f/^ 2) ;
                  vz_f  = vz_filter(vz/^ 2) ;
                  delta_e_c = vz_speed_control( vz_c ,vz_f/^ 2 ,q_f/^ 2 ,az_f/^ 2) ;
                  az_f  = az_filter(az/^ 2) ;
                  h_f  = h_filter(h/^ 2) ;
                  q_f  = q_filter(q/^ 2) ;
                  vz_c = altitude_hold(h_c/^ 20 , h_f/^2) ;
                  (va,  az, q, vz , h)  = aircraft_dynamics( (41814.0000000000 fby delta_x_c)*^ 4 ,
                                                                                  (0.0120000000 fby delta_e_c)*^ 4) ;
tel
```

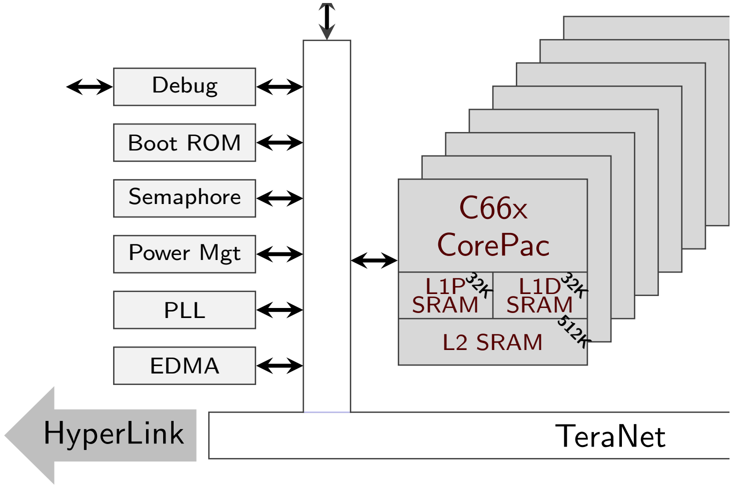
Context – multi-core COTS

Use of multi/many-core COTS in safety critical systems. Needs in terms of:

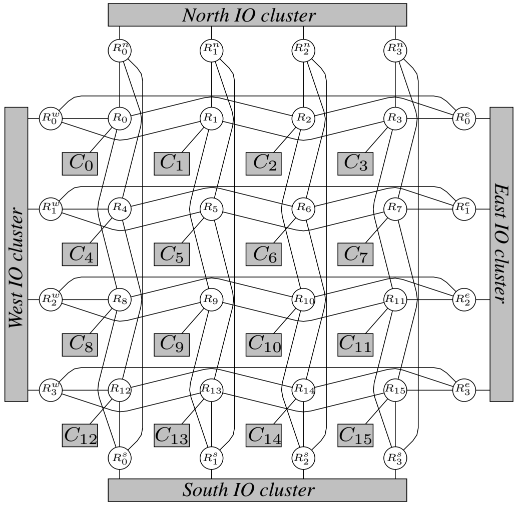
- Performance
- maturity
- affordable cost
- predictability (WCET computable)
- dependability
- programmability



"Open Integrated Modular Avionic (IMA): State of the Art and future Development Road Map at Airbus Deutschland", Airbus Deutschland Gmbh



Texas Instruments TMS320C6678



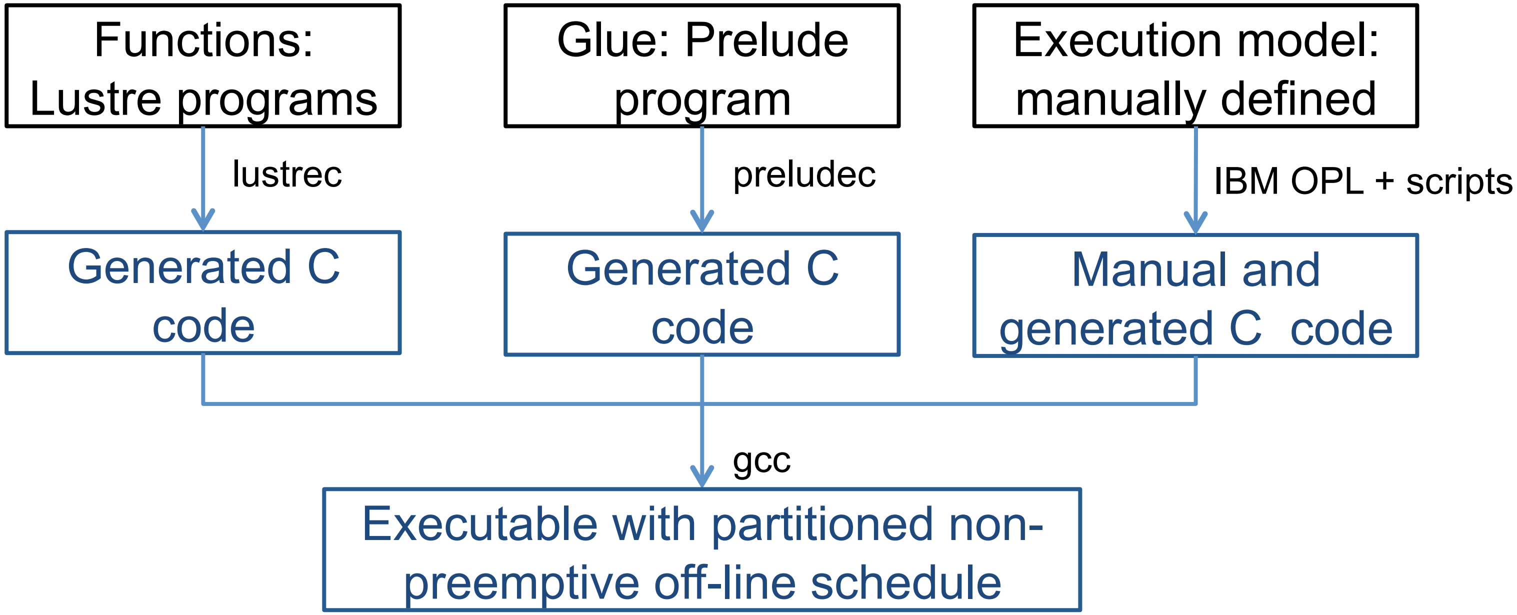
Kalray MPPA

Multi-core certification problem

- ❑ Aeronautic – certification standards
  - DO178 B/C, 1992 /2012
  - Position Paper CAST-32A Multi-core Processors, 2014 - 2016
  - White Paper FAA on Issues Associated with Interference Applied to Multicore Processors, 2017
- ❑ Purposes: set of guidances for software planning and verification on multi-core chips, with a particular emphasis on timing considerations and error handling
- ❑ The compilation framework is in the scope of the high level objective
  - « Interference channels and resource usage »
  - Issue: Shared resources on a platform can lead to unexpected delays or loss of data
  - Argumentation: the applicant has to identify all the interference channels in the final configuration and shall argue that the resource demand does not exceed the resource availability

Former solutions at ONERA

- ❑ Multi-periodic assembly expressed in Prelude
- ❑ Execution model
  - – to reduce or avoid any temporal interferences
  - – A set of programming rules, based on off line mapping and scheduling
- ❑ Script to generate the glue code
- ❑ WCET measured based



Functions: Lustre programs

lustrec

Generated C code

Glue: Prelude program

preludec

Generated C code

Execution model: manually defined

IBM OPL + scripts

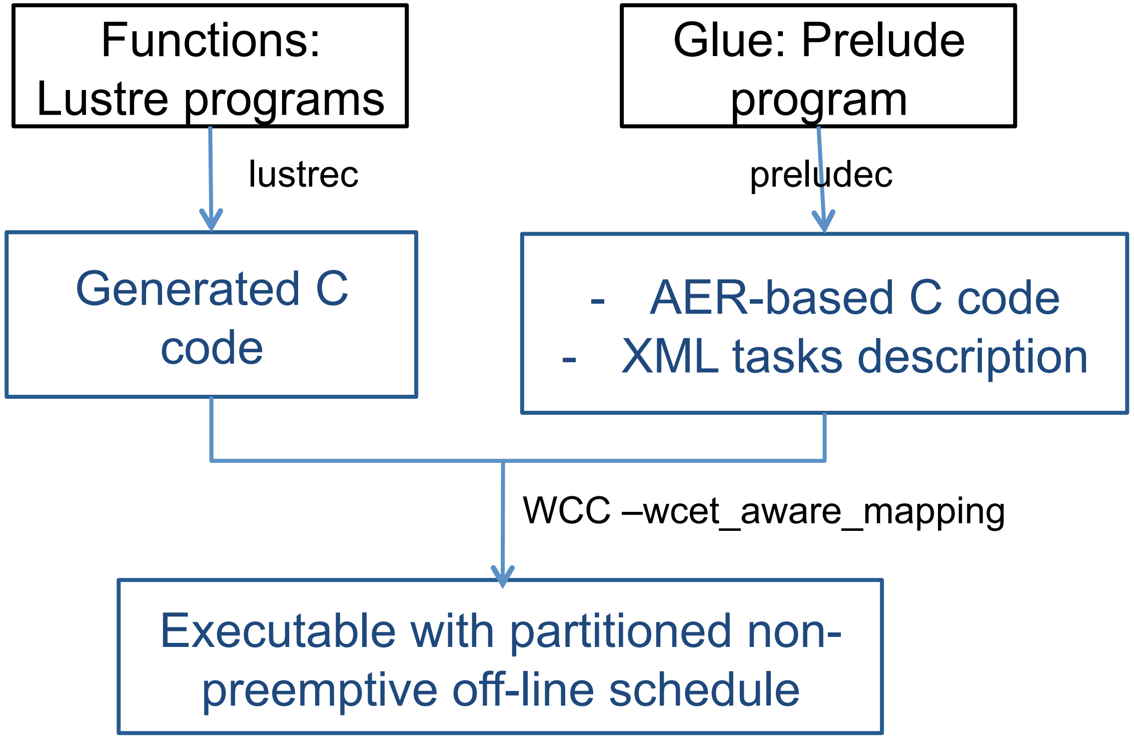
Manual and generated C code

gcc

Executable with partitioned non-preemptive off-line schedule

Overall new approach

1. Definition of an execution model for the target (AER)
2. Modification of Prelude compiler
3. Modification of WCC to generate mapped and scheduled applications

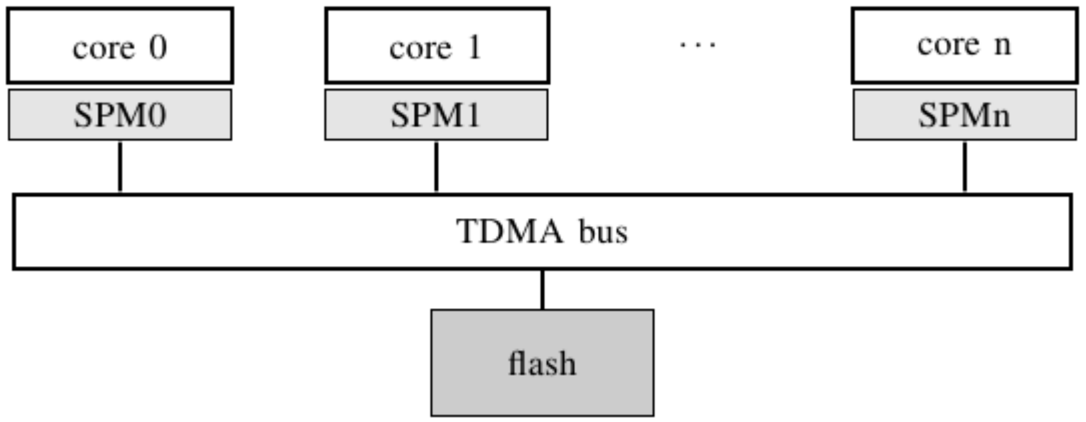


Outline

- ❑ Introduction
- ❑ **Contribution**
  - **Design choice: AER model**
  - Prelude extension with AER function generation
  - WCC extension for AER functions
- ❑ Conclusion

Processors supported by WCC

- ❑ TriCore (single core) and ARM (1 to 8 cores)
- ❑ ARM architecture
  - Core at 1 GHz
  - Private local SPM (scratchpad memory)
  - only local addressing on local SPM is supported, meaning that a core i cannot access the SPM of core j.
  - Bus arbitrated with a TDMA (Time division multiple access) protocol.

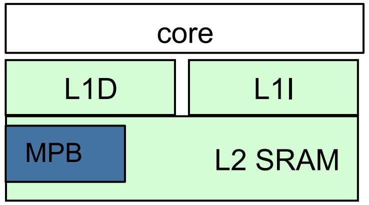


➔ next generation of embedded processors for automotive may share similar features.

Predictable solution – Execution model

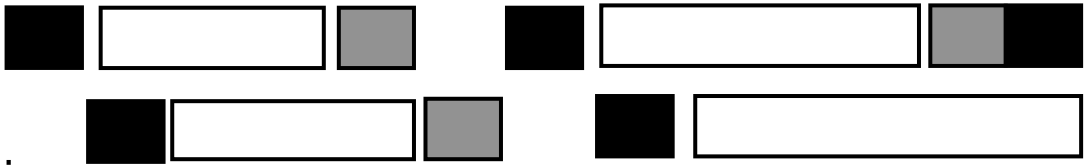
- ❑ Execution model
  - – Set of rules to be followed by the designer to avoid or at reduce the temporal interferences
  - – Separate the moment of pure execution and shared resource access

- ❑ AER model [Durrieu et al, 2014]
  1. Memory management
     - ▪ Codes and data stored statically and locally
     - ▪ Exchanged variables stored in specific zones MPB



ex config TMS

  2. Mapping scheduling strategies
     - ▪ Differentiate
       - • Acquisition ,
       Execution, Restitution
     - ▪ Static sequencing & mapping



ARM execution model

- ❑ **Rule 1:**
  - non preemptive partitionned off-line pre-computed schedule
- ❑ **Rule 2:**
  - all sections are stored in the local SPM
  - except the exchanged data which are in the flash
- ❑ **Rule 3:**
  - each function is split in 3 parts AER. During A, each “global variable” is copied in a local variable. During R, the value of a local variable is assigned to the produced variable
- ❑ **Rule 4:**
  - A and R phases always occur during the TDMA slots of the core hosting the function.

Outline

❑ Introduction

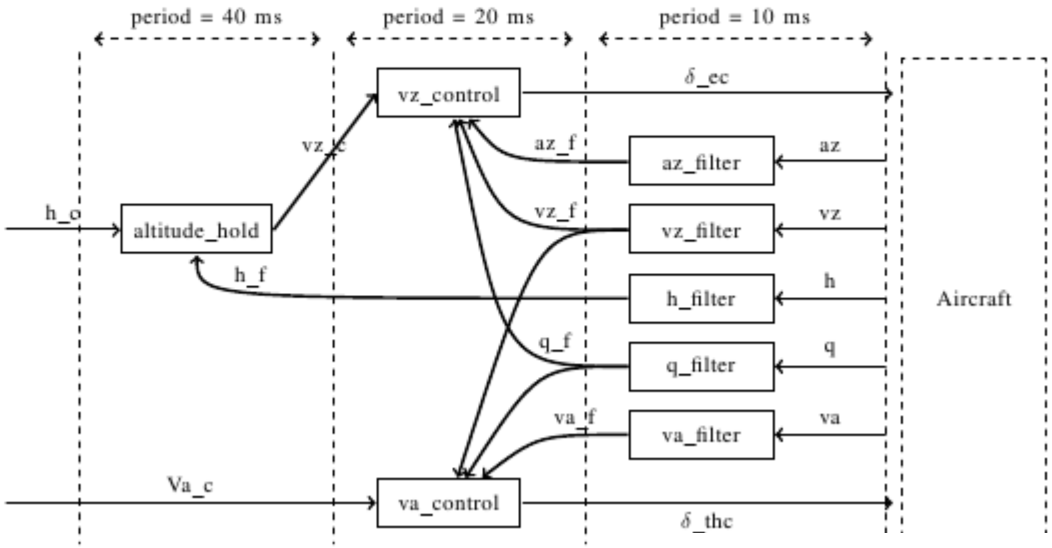
❑ **Contribution**

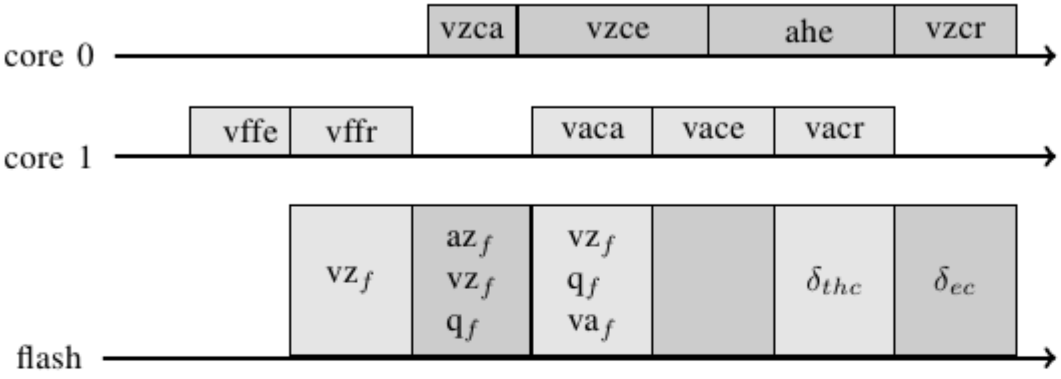
- Design choice: AER model
- **Prelude extension with AER function generation**
- WCC extension for AER functions

❑ Conclusion

Example of AER execution

❑ For the ROSACE controller





Legend:
vzca = vz_control_A
vzce = vz_control_E
vzcr = vz_control_R

Code generation – step 1

- Wrapping lustreC output as imported node C➔ C: genwrapper (ONERA/ LIFL)
- Assembly ➔ C: preludec
- For each function f, generation of f_A, f_E and f_R

```
static double h_filter110_fun_h_locread; /* local copy of a consumed data */
static double h_filter110_fun_h_f_locwrite; ; /* local copy of a produced data */

int h_filter110_A(void* args)
{
  static int h_rcell=0;
  static int instance=0;

  read_val(aircraft_dynamics73_h_h_filter110_h_id, h_rcell, sizeof(h_filter110_fun_h_locread),
                &h_filter110_fun_h_locread); /* copy of global variable in the local copy */
  h_rcell=(h_rcell+1)%2; /* communication protocol management */
  instance++;
  return 0; }
```

Code generation – step 2

- ❑ Global variables generation and link with the buffers id

```
enum {
 h_filter110_h_f_altitude_hold79_hf_id,
 aircraft_dynamics73_h_h_filter110_h_id,
 altitude_hold79_Vz_c_vz_speed_control104_Vz_c_id,
…, PLUD_BUFFER_NUMBER}

double  aircraft_dynamics73_h_h_filter110_h [2];
double  h_filter110_h_f_altitude_hold79_hf [2];
…

void * table_address [PLUD_BUFFER_NUMBER] =
{(void *) h_filter110_h_f_altitude_hold79_hf,
(void *) aircraft_dynamics73_h_h_filter110_h,
…}
```

Outline

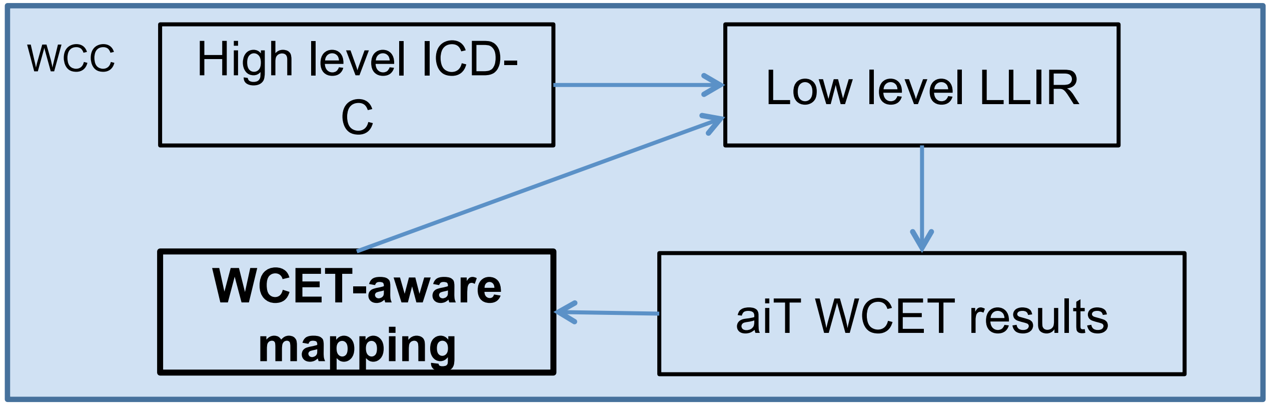
- ❑ Introduction
- ❑ **Contribution**
  - Design choice: AER model
  - Prelude extension with AER function generation
  - **WCC extension for AER functions**
- ❑ Conclusion

Interaction with WCC

- ❑ Input description
  - – Architecture description in an xml file (**hard coded**)
  - – Application description in an xml file (**generated by preludec**)

```
<task>
 <sources>
  <file>h_filter.c
   <entrypoint>
     <function>h_filter_a</function>
     <period> 10 </period>
   </entrypoint>
   <entrypoint> …
</file>…
```

- ❑ Extension in wcc



WCC

High level ICD-C

Low level LLIR

aiT WCET results

WCET-aware mapping

Algorithm – Integration strategy

```
procedure WCET aware mapping (Config appli)
    get SPM size
    get nb core, bus slot
    for function : t in appli do
        get t.period, t.name, t.subfunctions
        call aiT
        get t.wcetx, t.sizex (all sections, $x \in \{A, E, R\}$)
    end for
    call OPL IBM solver to solve the mapping problem
    for function : t in appli do
        get t.core, t.startx
    end for
    for core : c in Cores do
        generate C local scheduler
        generate new xml file (with the correct mapping and scheduling)
    end for
```

Step 1: hardware and application information

Algorithm – Integration strategy

```
procedure WCET aware mapping (Config appli)
    get SPM size
    get nb core, bus slot
    for function : t in appli do
        get t.period, t.name, t.subfunctions
        call aiT
        get t.wcetx, t.sizex (all sections, $x \in \{A, E, R\}$)
    end for
    call OPL IBM solver to solve the mapping problem
    for function : t in appli do
        get t.core, t.startx
    end for
    for core : c in Cores do
        generate C local scheduler
        generate new xml file (with the correct mapping and scheduling)
    end for
```

Step 2: off-line mapping and schedule

Conditional time-intervals

- ❑ OPL IBM constraint programming modelling with Conditional Time-Intervals
  - Very efficient for non preemptive schedules
  - Presented by Quentin Perret at RTNS 2016 (and a paper of this year)

- ❑ **Inputs**
  - Architecture
    - *Cores*, *SPMsize*
    - *MAF of TDMA, StartBusSlot[nbCores]*
  - Application
    - *TaskList, TaskProps[TaskList] (e.g. TaskProps[t].period*)
  - Pre-processing unrolling of tasks in *Jobs, JobProps[Jobs]*

- ❑ **Decision variables**
  - interval *phaseX[j in Jobs]*
  - optional interval *phaseX_c[j in Jobs][c in Cores]*

Formalization in OPL

- **Constraints**
  - Specific to conditional time intervals

$\forall j \in \textit{Jobs},\ \textit{alternative}(\textit{phaseX}[j], \textit{all}(c \in \textit{Cores})\ \textit{phaseX\_c}[j][c])$

$\forall c \in \textit{Cores},\ \Sigma_j\ \textit{pulse}(\Sigma_X\ \textit{phaseX\_c}[j][c], 1) \leq 1$

  - Scheduling (A before E and E before R)

$\forall j \in \textit{Jobs},$

$\quad \textit{endBeforeStart}(\textit{phaseA}[j], \textit{phaseE}[j])$

$\quad \textit{endBeforeStart}(\textit{phaseE}[j], \textit{phaseR}[j])$

  - All phases on the same core

$\forall j \in \textit{Jobs}, c \in \textit{Cores}, X \in \{A,E,R\}$

$\quad \textit{presenceOf}\ (\textit{phaseA\_c}[\textit{JobProps}[j].\textit{function}][c])$

$\quad == \textit{presenceOf}\ (\textit{phaseX\_c}[j][c])$

Formalization in OPL

- **Constraints**
  - Memory constraints

$\forall c \in \mathit{Cores},$

$\Sigma_t\ \mathit{presenceOf}(\mathit{phaseA\_c}[t][c]) \times (\Sigma_x\ \mathit{TaskProps}[t].\mathit{size}_x) \leq \mathit{SPMsize}$

  - A and R on the TDMA

$\forall j \in \mathit{Jobs}, c \in \mathit{Cores}, X \in \{A,R\}$

$\mathit{presenceOf}(\mathit{phaseX\ c}[j][c]) \Rightarrow$

$((\mathit{startOf}(\mathit{phaseX}[j]))\mathit{mod\ MAF} == \mathit{StartSlotBus}[c])$

Algorithm – Integration strategy

```
procedure WCET aware mapping (Config appli)
    get SPM size
    get nb core, bus slot
    for function : t in appli do
        get t.period, t.name, t.subfunctions
        call aiT
        get t.wcetx, t.sizex (all sections, $x \in \{A, E, R\}$)
    end for
    call OPL IBM solver to solve the mapping problem
    for function : t in appli do
        get t.core, t.startx
    end for
    for core : c in Cores do
        generate C local scheduler
        generate new xml file (with the correct mapping)
    end for
```

Step 3: generate C schedule on each core

Experiments

- ❑ Works well on several controllers, e.g. ROSACE

preludec	Step 1	OPL	Step 3
0m0.114s	1m45.132s	0m0.601s	0m20.481s

- ❑ WATERS 2017 challenge
  - – 1250 runnables, 10000 labels

preludec	Step 1	OPL	Step 3
0m9.163s	3550m15.365s	0m36,074s	30m22.548s

Conclusion

- ❑ Complete framework from synchronous programs to predictable executables

- ❑ More experiments
- ❑ Execution on a real target
- ❑ We followed a “bottom-up” approach ➔ re-think the internal representations to support AER and synchronous semantics features

Thanks for your attention