The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution

Claire Pagetti\textsuperscript{1}, David Saussié\textsuperscript{2}, Romain Gratia\textsuperscript{3}, Eric Noulard\textsuperscript{1} and Pierre Siron\textsuperscript{4}
\textsuperscript{1}ONERA - Toulouse, France
\textsuperscript{2}Polytechnique - Montréal, Canada
\textsuperscript{3}TELECOM ParisTech - Paris, France
\textsuperscript{4}ISAE - Toulouse, France

20th IEEE Real-Time and Embedded Technology and Applications Symposium RTAS 2014
Outline

- Introduction
  - Context
  - Objective

- ROSACE (Research Open-Source Avionics and Control Engineering) case study

- Predictable implementation

- Conclusion and perspectives
Control design level
- Steps: non linear $\Rightarrow$ linearization around a flight condition $\Rightarrow$
  controller synthesis $\Rightarrow$ digitalization (sampling periods)
- Tools: Matlab / Simulink

Implementation level
- Coding of elementary blocks: Scade, Lustre, Signal…
- Coding of multi-periodic assembly: home made language, manual coding, …

COTS hardware integration
- Automatic code generator
- Manual code
- Low level services
Avionic use case: Longitudinal Flight Controller

- Longitudinal motion of a medium-range civil aircraft in *en-route* phase
  - *Cruise*: maintains a constant altitude \( h \) and a constant airspeed \( V_a \)
  - *Change of cruise level* subphases:
    commands a constant vertical speed \( V_z \) (rate of climb)
    - Ex: FL300 → FL320 → FL340 → FL360
    - FL300 = pressure altitude of 30000 ft

- **Performance requirements for change of cruise levels**
  - **P1 settling time**: time required to settle within 5% of the steady-state value
  - **P2 overshoot**: maximum value attained minus the steady-state value
  - **P3 rise time**: time to rise from 10% to 90% of the steady-state value
  - **P4 steady-state error**: difference between the input and the output at \( t \to \infty \).
- **36 tiles**
  - Single core clocked at 1.2 GHz
  - 32 KB program memory (L1P)
  - 32 KB data memory (L1D)
  - 256 KB level 2 memory (L2)

- **NoC (network on chip)**
  - five full-duplex sub-networks
  - Shared Dynamic Network (SDN) for exchanging data between tiles, e.g., for cache coherency
  - reQuest Dynamic Network (QDN) for write requests to the DDR
  - Response Dynamic (RDN) Network for read requests to the DDR

- **Development environment**
  - Zero Overhead Linux (ZOL): no non-user interrupts
  - Shared memory with cache homing policy

- **Timing issues**
  - local time-stamp counter
  - cores start independently => unpredictable offsets
Objective

Control design level
- Identification of performance requirements

Implementation level
- Coding of multi-periodic assembly: Prelude
- Analyse several multi-periodic assemblies wrt to the performance requirements and real-time efforts

COTS hardware integration
- A predictable execution model in ZOL
- Compliance with the high level requirements

Open source case study
https://svn.onera.fr/schedmcore/branches/schedmcore-RTAS2014/Case_Study_RTAS
Outline

- Introduction
- **ROSACE case study**
- Predictable implementation
- Conclusion and perspectives
### Longitudinal flight controller architecture

**Flight condition:**
- $h = 10000 \text{ m}$, $V_a = 230 \text{ m/s}$

<table>
<thead>
<tr>
<th>Outputs</th>
<th>$V_z$</th>
<th>vertical speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$V_a$</td>
<td>true airspeed</td>
</tr>
<tr>
<td></td>
<td>$h$</td>
<td>altitude</td>
</tr>
<tr>
<td></td>
<td>$a_z$</td>
<td>vertical acceleration</td>
</tr>
<tr>
<td></td>
<td>$q$</td>
<td>pitch rate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filtered outputs</th>
<th>$V_{zf}$</th>
<th>vertical speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$V_{af}$</td>
<td>true airspeed</td>
</tr>
<tr>
<td></td>
<td>$h_f$</td>
<td>altitude</td>
</tr>
<tr>
<td></td>
<td>$a_{zf}$</td>
<td>vertical acceleration</td>
</tr>
<tr>
<td></td>
<td>$q_f$</td>
<td>pitch rate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reference inputs</th>
<th>$h_c$</th>
<th>altitude command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$V_{ac}$</td>
<td>airspeed command</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commanded inputs</th>
<th>$V_{zc}$</th>
<th>vertical speed command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\delta_{ec}$</td>
<td>elevator deflection</td>
</tr>
<tr>
<td></td>
<td>$\delta_{thc}$</td>
<td>throttle command</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Aircraft inputs</th>
<th>$\delta_{ec}$</th>
<th>elevator deflection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T$</td>
<td>engine thrust</td>
</tr>
</tbody>
</table>

- 5 filters consolidate the measured outputs provided by the sensors
- 3 controllers track accurately: altitude ($h_c$), vertical speed ($V_{zc}$) and airspeed commands ($V_{ac}$)
- rate choices
  1. for controllers:
     - closed-loop system with the continuous-time controller can tolerate a pure time delay of 1 s before destabilizing $\Rightarrow$ sampling period $\leq 1 \text{ Hz}$
     - performances $\Rightarrow$ sampling period $\leq 10 \text{ Hz}$
  2. for environment: 200 Hz to model a continuous-time phenomenon
1. Analysis of $V_a$ and $V_z$ loops with separate step demands

   - Airspeed variation of 5 m/s
   - Vertical speed demand of $V_{zc} = 2.5$ m/s

2. Analysis of P4: input is a step climb

   - Altitude change of 1000 m
     - First phase: constant vertical speed demand
     - Second phase: altitude reaching
Requirements validation at Simulink level

Results for the decoupled approach

<table>
<thead>
<tr>
<th>Property</th>
<th>Objective</th>
<th>Results in SIMULINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 5% settling time</td>
<td>$V_z \leq 10,\text{s}$</td>
<td>8.22 s</td>
</tr>
<tr>
<td></td>
<td>$V_a \leq 20,\text{s}$</td>
<td>17.22 s</td>
</tr>
<tr>
<td>P2 Overshoot</td>
<td>$V_z \leq 10%$</td>
<td>4.72%</td>
</tr>
<tr>
<td></td>
<td>$V_a \leq 10%$</td>
<td>3.65%</td>
</tr>
<tr>
<td>P3 Rise time</td>
<td>$V_z \leq 6,\text{s}$</td>
<td>5.09 s</td>
</tr>
<tr>
<td></td>
<td>$V_a \leq 12,\text{s}$</td>
<td>11.6 s</td>
</tr>
<tr>
<td>P4 Steady-state error</td>
<td>$V_z \leq 5%$</td>
<td>0.83%</td>
</tr>
<tr>
<td></td>
<td>$V_a \leq 5%$</td>
<td>0.11%</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- ROSACE case study
- Predictable implementation
  - Coding
  - Integration on the Tilera
- Conclusion and perspectives
Coding of the basic blocks

- In the paper, discretization and manual coding in C
  - Forward Euler method for the environment
  - Forward difference for the controllers
  - Zero-order hold approximation for the filters

- Since then, the discretized model has been written in Lustre and the C code generated with [KGHT14]. Prelude compiler has been extended to wrap automatically the 
  lustrec outputs to be compliant with Prelude format.

Prelude in a nutshell

- Prelude: a real-time synchronous language. Allow to express real-time behaviours in a synchronous style
  1. By using relaxed synchronous hypothesis (Curic 2005)
  1. By adding the notion of real-time periodic clock
  1. By generating dependent task sets with extended precedence constraints
  4. By reusing Sofronis et al. communication protocol

- SchedMcore framework
  - Simulation of Prelude programs
  - Script that verifies if a trace fulfils the requirements

A clock $\alpha=(t_i)$ is strictly periodic iff
\[ \exists n \in \mathbb{N}^*, \forall i \in \mathbb{N}, t_{i+1} - t_i = n \]
\[ \pi(\alpha)=n \text{ is the period, } \varphi(\alpha)=t_0 \text{ is the offset.} \]

eg: (120, 1/2) is the clock of period 120 and offset 60.

F (10, 5, 10, 0)
S (40, 15, 40, 0)
F[4k] $\rightarrow$ S[k]
S[k] $\rightarrow$ F[4k+4]

http://www.lifl.fr/~forget/prelude.html
First coding in Prelude

```haskell
node assembly (h_c, Va_c: real rate (1000,0))
returns (delta_e_c, delta_th_c : real)
var Va, Vz, q, az, h: real;
    Va_f, Vz_f, q_f, az_f, h_f : real;
    Vz_c, delta_e, T : real;
let
    h_f = h_filter(h/^2);
    Va_f = Va_filter(Va/^2);
    Vz_f = Vz_filter(Vz/^2);
    q_f = q_filter(q/^2);
    az_f = az_filter(az/^2);
    Vz_c = altitude_hold (h_f/^2, h_c*^5);
    delta_e_c = Va_control (Va_f/^2, Vz_f/^2, q_f/^2, Va_c*^5);
    delta_th_c = Vz_control (Vz_f/^2, Vz_c, q_f/^2, az_f/^2);
    T = engine (delta_th_c*^4);
    delta_e = elevator (delta_e_c*^4);
    (Va, Vz, q, az, h)= aircraft_dynamics
    ((0.018645918123716 fby delta_e), (43219.8575 fby T));
tel
```

Basic clock (1,0) at 100 µs

<table>
<thead>
<tr>
<th>Node</th>
<th>Frequency</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude_hold</td>
<td>50Hz</td>
<td>(200, 0)</td>
</tr>
<tr>
<td>Va_filter</td>
<td>100Hz</td>
<td>(100, 0)</td>
</tr>
<tr>
<td>az_filter</td>
<td>100Hz</td>
<td>(100, 0)</td>
</tr>
<tr>
<td>Vz_control</td>
<td>50Hz</td>
<td>(200, 0)</td>
</tr>
<tr>
<td>elevator</td>
<td>200Hz</td>
<td>(50, 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Frequency</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_filter</td>
<td>100Hz</td>
<td>(100, 0)</td>
</tr>
<tr>
<td>q_filter</td>
<td>100Hz</td>
<td>(100, 0)</td>
</tr>
<tr>
<td>Va_control</td>
<td>50Hz</td>
<td>(200, 0)</td>
</tr>
<tr>
<td>engine</td>
<td>200Hz</td>
<td>(50, 0)</td>
</tr>
<tr>
<td>ac_dynamics</td>
<td>200Hz</td>
<td>(50, 0)</td>
</tr>
</tbody>
</table>
- Relaxing precedences:

\[ T = \text{engine} \left( (1.640222296162316 \ \text{fby} \ \delta_{\text{th}} c)^{4} \right); \]
\[ \delta_{e} = \text{elevator} \left( (0.018645918123716 \ \text{fby} \ \delta_{e} c)^{4} \right); \]

- Relaxing periods:
  - \( h_{\text{filter}}, \ Va_{\text{filter}}, \ Vz_{\text{filter}} \) at 50Hz; \( \text{altitude\_hold} \) at 10Hz.

\[ h_{f} = h_{\text{filter}}(h^{4}); \]
\[ V_{a_{f}} = V_{\text{a\_filter}}(V_{a}^{4}); \]
\[ V_{z_{f}} = V_{\text{z\_filter}}(V_{z}^{4}); \]
\[ V_{z_{c}} = \text{altitude\_hold} \left( h_{f}^{5}, h_{c} \right); \]
\[ \delta_{\text{th\_c}} = V_{a\_control} \left( V_{a_{f}}, V_{z_{f}}, q_{f}^{2}, V_{a_{c}}^{5} \right); \]
\[ \delta_{e_{c}} = V_{z\_control} \left( V_{z_{f}}, V_{z_{c}}^{5}, q_{f}^{2}, az_{f}^{2} \right); \]
Objective:
- Simulation of the specification
- Real-time constraints are assumed correct

Input:
- Prelude program
- Code of imported nodes

Results
- no significant degradation and only the settling time is slightly impacted by the modifications.
Outline

- Introduction
- ROSACE case study
- Predictable implementation
  - Coding
  - Integration on the Tilera
- Conclusion and perspectives
Predictable rules for the Tilera

- **Stressing benchmarks** to assess the variability of the platform
  - impact on the execution times when several tiles access concurrently the shared resources (read and write), such as DDR, local caches and NoC.
  - time to access the local clock

- **Mapping rules**
  1. No more than 10 tiles must simultaneously access in writing the same [shared] cached memory location
  2. No more than 5 tiles must simultaneously access in writing the DDR
  3. No more than 30 tiles must simultaneously access in reading the same [shared] cached memory location
  4. Offline partitioned non preemptive schedule
  5. Memory footprint fits in the local caches
Mapping the flight controller on the Tilera

- **WCET in isolation**

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Task</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>aircraft_dynamics</td>
<td>200 µs</td>
<td>elevator</td>
<td>100 µs</td>
</tr>
<tr>
<td>altitude_hold</td>
<td>100 µs</td>
<td>engine</td>
<td>100 µs</td>
</tr>
<tr>
<td>h_filter</td>
<td>100 µs</td>
<td>q_filter</td>
<td>100 µs</td>
</tr>
<tr>
<td>Vz_filter</td>
<td>100 µs</td>
<td>az_filter</td>
<td>100 µs</td>
</tr>
<tr>
<td>Va_filter</td>
<td>100 µs</td>
<td>Va_control</td>
<td>100 µs</td>
</tr>
<tr>
<td>Vz_control</td>
<td>100 µs</td>
<td>delta_e_c, Va_c, delta_th_c h_c</td>
<td>500 µs</td>
</tr>
</tbody>
</table>

- **2 offline mapping (manual computation)**
  1. 1 core – 1 task
  2. Tile 1

```
Aircraft_dynamics
```

```
elevator
```

```
engine
```

---

```
Tile 2
```

```
h_filter
az_filter
Vz_filter
q_filter
Va_filter
```

---

```
Tile 3
```

```
altitude_hold
Vz_control
Va_control
```

- April 17th – RTAS 2014
Mapping the flight controller on the Tilera

- Tick-based scheduling
  - Decisions taken only at discrete instants

- Communication delay on the NoC

- Clock precision

⇒ Enforce WCET

35 ns (communication delay) + 500 ns (clock precision)

- **Objective:**
  - Run the specification on the Tilera
  - WCET extended with tick gap

- **Input:**
  - Prelude program
  - Code of imported nodes
  - Mapping rules

- **Results**
  - Almost the same as those obtained by the functional ones
Conclusion & perspectives

- Design of a parallel avionic longitudinal controller on a multi/many-core target
- Illustration of discussion between control engineers and integrators can be leveraged, in order to find a compromise between both sides constraints.

Future work:
- Designer kit (constraint programming approach to compute static mapping and scheduling)
- Use of “don’t care” operators in order to generate many assemblies
- Mixed-critical systems: integration of fault-tolerant mechanisms

Thank you for your attention