Automatic implementation of TTEthernet-based time-triggered avionics applications

Raul A. Gorcitz, Thomas Carle, David Lesens, David Monchaux, Dumitru Potop, Yves Sorel
(CNES, INRIA, Airbus DS)
dumitru.potop@inria.fr
Open question from Nicolas

• The end of time in time-triggered systems research?

• Maybe not: Time-triggered systems and static scheduling are twin fields
  • Time-triggered as an abstraction (which it is anyway to some point, except in fully synchronous HW)
  • Platform abstraction issues
    – "Simplifying hypotheses" must go away at some point, especially the unsafe ones
      » No synchronization time, no interferences due to memory access, etc.
    – Precision
  • Scalability issues:
    – Will constraint solvers scale on 15000 elementary bocks, on 16 cores, taking into account allocation/scheduling/memory interferences/etc.?

• My take: RT scheduling seen as compilation
Embedded control systems

• Computing system that controls the execution of a piece of « physical » equipment, in order to ensure its correct functioning.

![Diagram of Embedded Control System](image-url)

- Actuators
- Sensors
- Embedded control system
- Plant (controlled « physical » system)

Domains:
- Analog domain
- Digital domain
Embedded control systems

• Example: Guidance, Navigation and Controle (GNC) in a plane/launcher.
  – Use sensors (GPS, accelerometers, gyros, etc...) to determine the position, speed, and attitude of the plane/rocket
  – Compute the correction (control algorithm)
  – Update actuators
Embedded control systems

Control engineering

Continuous-time model

Functional specification

Platform-independent real-time specification

Controller discretization

Other non-functional requirements

Plant+Controller

Other engineering fields

(HW design, reliability, etc...)

Cyclic control

Periods

Deadlines

Latency

Functional correctness

Schedulability analysis included

Implementation model

Code generation (including compilation)

Executable implementation

Criticality

Allocation

Partitioning

Off-line

CPU, bus, etc.

On-line

Topology

R-T calculus models

Respect of requirements
Our objective

Synchronous specifications
(Scade/Heptagon)

Release dates
Deadlines
(periods, latency
constraints)

Allocation
Partitioning
(criticality)
Preemptability

Platform-independent
real-time specification

Full automation
- Compilation-like
- Correct by construction

Other non-functional
requirements

Platform model
(HW+OS)

Toplogy
(CPUs,buses,...)

WCET/WCCT
IMA/ARINC 653
(VxWorks 653)
TTEthernet

Functional
specification

Off-line real-time
scheduling

Scheduling tables

Code generation

Time-triggered
implementation

Time-triggered
avionics
applications
(Airbus DS space launcher
case study)
Principles of our approach

• Off-line mapping
  – All scheduling decisions are taken before execution
    • Conditional execution allowed
  – Construction of scheduling tables

• Take into account both computations and communications during scheduling
  – Durations (WCETs, WCCT/WCTTs)
  – Various communication media
    • Buses, shared RAMs, complex networks (NoCs)
    • Each requires different communication and synchronization protocols
      – Differences in scheduling algorithms and code generation
      – Tools: Lopht, SynDEx
Principles of our approach

• Off-line mapping: ignition engine controller

HS: High speed mode
FS: Fail-safe mode
Principles of our approach

• Off-line mapping: ignition engine controller
  – 3 CPUs, 1 broadcast bus
    \[d_{P_0}(\text{read}_\text{input}) = 1\]
    \[d_{P_0}(F1) = 3\]
    \[d_{P_0}(F2) = 8\]
    \[d_{P_0}(F3) = 5\]
    \[d_{P_0}(V) = 1\]
  – WCETs/WCCTs
  – Allocation constraints
Principles of our approach

- **Off-line mapping: ignition engine controller**

<table>
<thead>
<tr>
<th>time</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HS_IN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FS_IN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>if(¬HS) F1</td>
<td></td>
<td></td>
<td>send(P0,FS)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>if(FS) N</td>
<td></td>
<td>send(P0,HS)</td>
</tr>
<tr>
<td>4</td>
<td>if(¬HS) G</td>
<td></td>
<td>if(HS&amp;¬FS)</td>
<td>send(P0,ID)</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>if(¬HS)&amp;¬FS</td>
<td>send(P1,ID)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>if(¬FS)M</td>
<td>if(¬HS)</td>
<td>send(P0,V)</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>if(¬HS) F3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Principles of our approach

- Compilation-like approach
  - Reliance on fast, efficient heuristics
    - List scheduling combined with
      - Software pipelining to improve throughput/schedulability
      - Deadline-driven scheduling to improve schedulability
      - Post-scheduling optimizations: to reduce the number of context/partition switches, reduce the number of tasks
  - Easy to take into account:
    - Execution conditions/modes in the functional specification
    - Complex non-functional properties
    - « OS costs » (synchronization, preemption, partition changes, driver cost)
      - Interferences due to access to shared resources (e.g. RAM)
  - Easy to report errors and integrate in an interactive approach
Airbus DS Case study

• Complex application:
  – 13 tasks, 7 partitions, 10 end-to-end flow constraints
  – Complex non-functional requirements: Partitioning, Real time (release dates, flow latencies), Allocation

• Initial architecture:
  – 4 processors, 1 broadcast bus, global time base

• Result
  – Processor charge: 82%, 72%, 72% and 10% (telemetry)
  – Bus charged at 81%

• Now, we need to consider a real, industrial time-triggered communication infrastructure ensuring the global time base
  – TTEthernet
TTEthernet

• Communication network (not a bus)
  – Unidirectional links connecting end-stations and switches
    • Each link has a separate arbiter/scheduler
  – Store-and-forward packet switching policy
  – All traffic organized in virtual links (VLs) with fixed route and transmission policy
    • Time-triggered (TT) – strictly periodic, one packet per VL period
    • Rate-controlled (RC) – AFDX-like
      – Include clock synchronization traffic (protocol control frames = PCF)
    • Best effort (BE)

• Global time reference, 3 time scales
  – Raster tick – all dates and deadlines are formulated in raster ticks
    • At most one TT packet/raster tick (minimum_delta_r = 1)
    • Recommended: 200usec if network speed is 100mbps
  – Integration Cycle – Periodicity of PCFs
    • multiple of raster tick
  – Cluster Cycle – Length of scheduling table and sync with VxWorks 653
    • multiple of the Integration cycle and of the period of all TT VLs
Lopht for TTEthernet

- **Objective**: joint mapping of tasks and TTEthernet communications
  - Choose VL routes
  - Choose TT packet scheduling for each link

- **Challenges**:
  - Build a global scheduling table including computations on processors and TT packets on links
  - Control interferences from non-TT traffic
  - Interoperation with TTE configuration tool (TTE-Plan)
VxWorks653

LoPhT

LoPhT Input

Scheduling

Scheduling Table

Code Generation

Bus Schedule
(TTEthernet Code)
.start date
.duration
.source
.destination

C code
1/partition on
every processor

ARINC 653 Configurations

LooperT Output

Compiler

Application Code
Arinc653/VxWorks653
TTEthernet
Libraries

Compilation

Executables

Execution on the Platform

Communication System Architecture
Single-Board Computers

TTE Tool-set

Network Description

TTE-Plan

Configuration Files

Architecture Specific Parameters
(synchronization domains, devices, physical links)

Virtual Links
.name
.periodicity
.sender
.receivers
...

Temporal Constraints
Lopht/TTE-Plan interoperability

- **TTE-Plan**
  - Input: VL requirements
    - VL type
    - Start and destination systems
    - Real-time requirements
  - Output
    - VL routes
    - Link scheduler configurations

- **Overlapping with Lopht functions**
  - Forget information when building TTE-Plan input
    - Problem if TTE-Plan cannot find a solution afterwards (as it exists)
    - Possible solutions
      - Enrich TTE-Plan input to allow specification of routing and scheduling constraints
      - Completely bypass TTE-plan?
Control interferences from non-TT traffic

- Hypotheses in TTEthernet use
  - Only TT data traffic
    - Remaining non-TT traffic: clock synchronization (PCF frames of RC type)
  - A TT packet must fit inside a single raster tick
    - 100mbps network => max 120usec/packet (1499-byte payload)
    - Usual raster tick = 200usec
Control interferences from non-TT traffic

- PCF traffic and sync. precision: can be neglected
  - TT packet (120us),
  - PCF (5us),
  - Raster (200us)
- Device induced delay and network sync precision assumed included in raster

- Multi-hop => overhead of at least one raster tick per link
Build global scheduling table

• Ressources = processors, TTE links
  – Only schedule tasks and TT traffic

• Extension of algorithms for mapping onto network-on-chips
  – Strict periodicity of communications
    • Start with single-period systems
  – Take into account the VxWorks 653/TTE interface
    • Use of queuing pseudo-ports and the critical frame output buffer of the TTE network card of each end system
Conclusion

• Compilaton-like approach in the construction of time-triggered real-time systems
  – Lopht tool working on complex IMA/ARINC 653-based systems (and other types of systems)

• Lopht extension to include TTEthernet networks
  – Already done:
    • Build a high-level model for use by the scheduling algorithms
      – Assumptions on platform use
    • Manual building of a case-study
  – Still to do:
    • Experimentation on the TTE platform
    • Extending the platform modeling language of LoPhT
    • Modifying scheduling and code generation algorithms
    • Tool integration