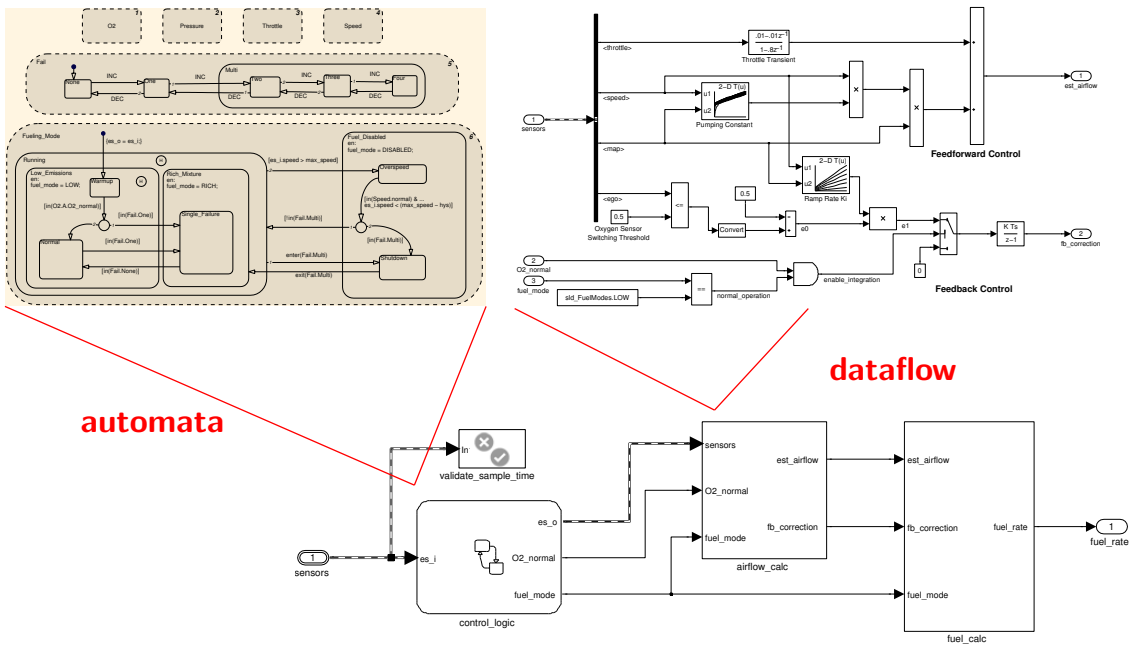
EMC^2 – Executable Application Models and Design Tools for Mixed-Critical, Multi-Core Embedded Systems

Technical Focus

ARTEMIS Technology Conference, EMC^2 Workshop, October 6 2016

Motivation: Reconcile *Computation* and *Control*

Context: smart embedded systems involve increasingly *complex tasks* and *parallel computing* resources, *in the control loop*



$\leftrightarrow$

Distribution?
Acceleration?
Scalability?
Efficiency?
Isolation?
Races?
Heisenbugs?

Block-diagram, synchronous languages

Motivation: Reconcile *Computation* and *Control*

→ Parallelizing compilation for block-diagrams and state machines (so-called synchronous programs), *integrating non-functional constraints*: e.g., *porting computational libraries to safety-critical environments is a burning challenge*

→ Modeling the system *and* providing guarantees that parts of the system achieve a desired contract: e.g., *timing composability*, *isolation*

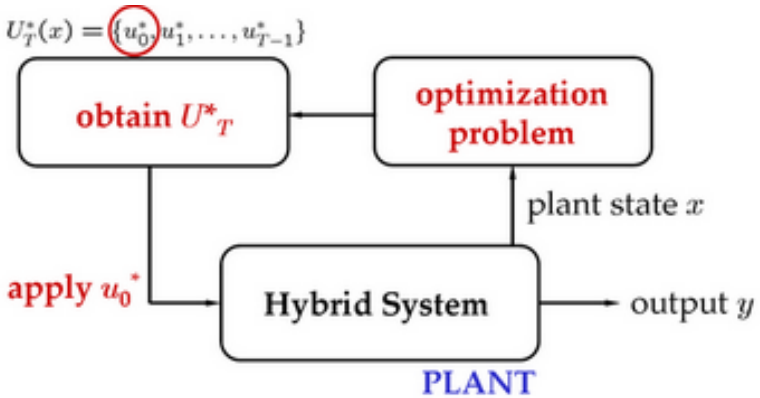
→ New applications such as *online optimization, high-performance simulation, monitoring, data analytics in control systems*

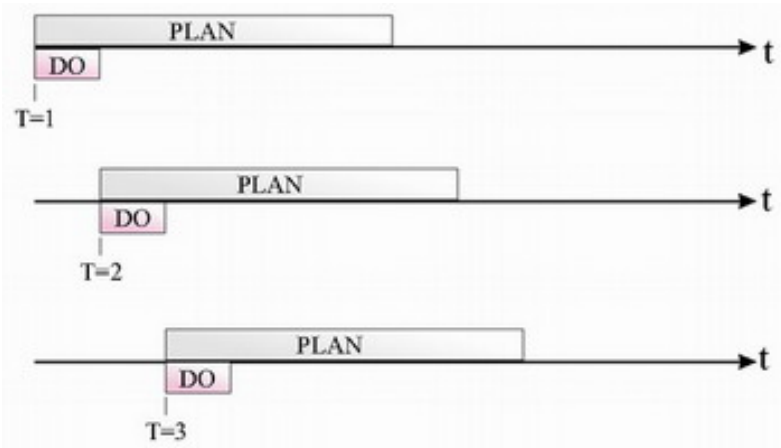
→ Also, security is a growing dimension, from logical/software bugs to side-channel attacks and hardware fault injection

Motivation: Reconcile *Computation* and *Control*

Prototypical scenario: online optimization in the loop of a safety-critical system

E.g., *(Fast) Model Predictive Control*: anticipate input sequence with *receding horizon*, relying on convex solvers with *incrementally refined precision*, possibly running in parallel with a safe but less optimized PID controler





(courtesy Morari, Hempel, ETHZ)

Correct and Efficient Concurrency *"By Construction"*

Between the hammers of
correction, *programmability*, *certification*, *performance*
and the anvil of *cost*

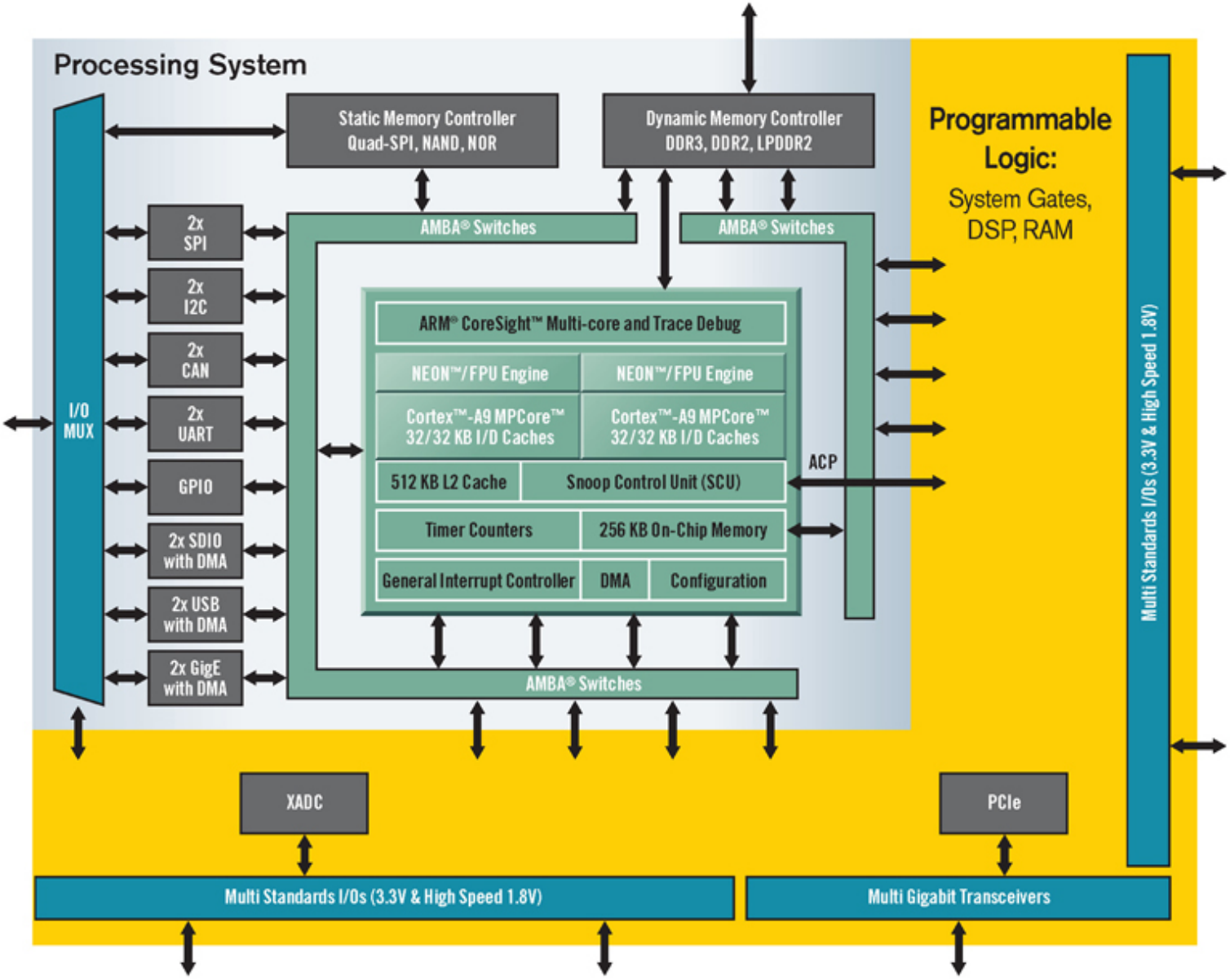
Proposed Approach
... or, when *reliability* meets *efficiency*

1. A *single source code* should serve
   - as an abstract model for *verification*
   - as an executable model for *simulation*
   - as the high level program from which target-specific *sequential/parallel* code is generated
2. Provide correctness and safety guarantees *throughout the design and refinement phases, and through compilation*
3. Rely upon *proven and efficient execution environnements*
4. Rely upon *formal models of SW stack and HW platform*, to improve quality and ease certification

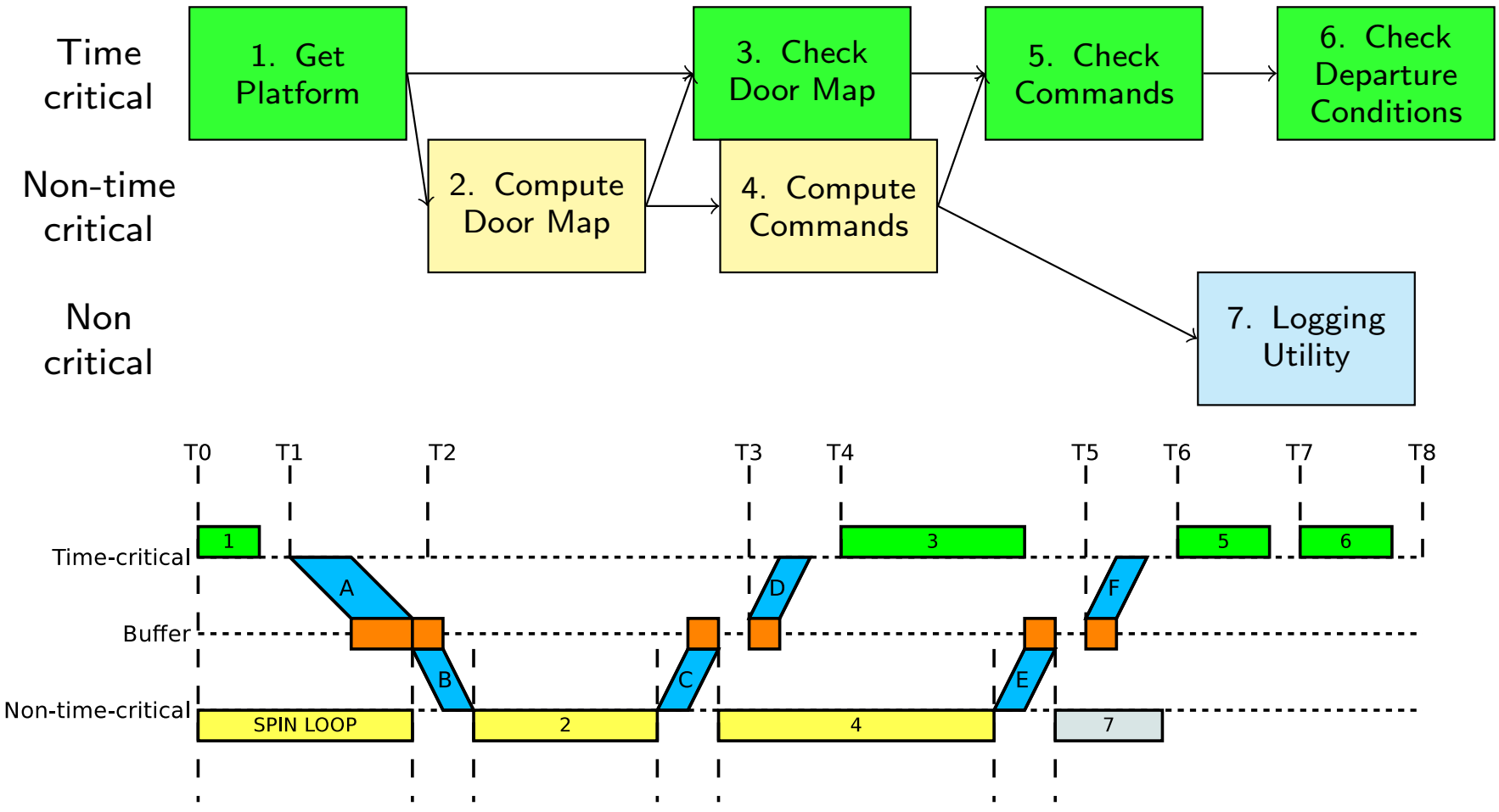
Case Study – Passenger Exchange

- Mission
  - Issue commands to open or close doors according to a given plan
  - Announce imminent door opening/closing to passengers
  - Warn the traffic supervision when the passenger exchange cannot be completed
- Safety
  - While the train has not stopped securely, the doors cannot be opened
  - Only properly aligned doors can be opened
  - The train is only allowed to leave when all doors are closed

Case Study – Experimental Platform: Zynq 7k

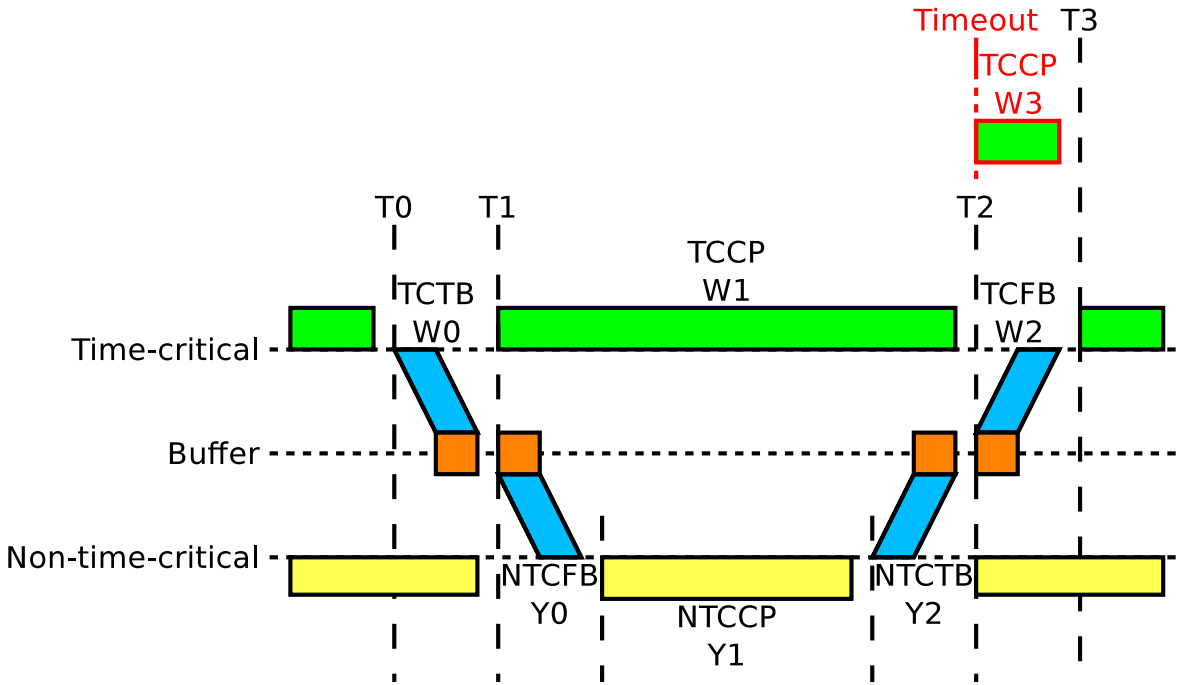


Case Study – Mixed-Time-Critical Chronogram



Case Study – Tasks Missing Deadlines

Time-triggered protocol



Time-critical

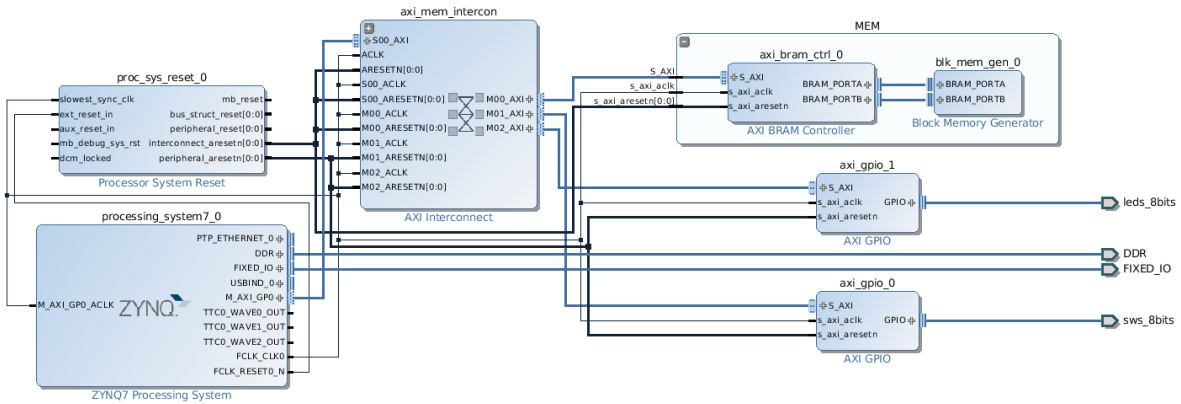
- Computing (TCCP)
- Copy to buffer (TCTB)
- Copy from buffer (TCFB)

Non-time-critical

- Computing (NTCCP)
- Copy to buffer (NTCTB)
- Copy from buffer (NTCFB)

Case Study – Platform Configuration

- ARM core 0: non-time-critical and non-critical tasks, Linux
- ARM core 1: time-critical tasks, bare-metal environment, software stack running in on chip memory of 256KB (code and data)
- FPGA: communication buffers



Hardware configuration (Vivado)

Resource	Used	Available	Util%
Slice LUTs	2345	53200	4.40%
Slice Registers	2815	106400	2.64%
Block RAM Tile	8	140	5.71%

Case Study – Modeling/Programming Flow

Aiming for SIL 4 certification $\longrightarrow$ synchronous programming methodology

Listing 1: Simplified check_commands in HEPTAGON

```
node check_command(door_command : command; door_map : int)
    returns (safe_command : command)
let
  safe_command = if door_map <> -1 then door_command else None;
tel

task check_commands(unpunctual door_commands : command^n; door_map : int^n)
    returns (safe_commands : command^n)
let
  if ontime door_commands then
    safe_commands = map<<n>> check_command(door_commands, door_map);
  else
    safe_commands = None^n;
  end
tel
```

Case Study – Modeling/Programming Flow

Aiming for SIL 4 certification → synchronous programming methodology

Listing 2: Snippet of the passenger exchange in HEPTAGON

```
node passenger_exchange(train_position : int)
    returns (safe_door_commands : command^n; departure_authorization : bool)
var
  platform : int;
  unpunctual door_map : int^n;
  safe_door_map : int^n;
  unpunctual door_commands : command^n;
let
  platform = get_platform(train_position);
  door_map = compute_door_map(platform);
  safe_door_map = check_door_map(door_map, platform);
  door_commands = compute_commands(door_map);
  safe_door_commands = check_commands(door_commands, safe_door_map);
  departure_authorization = check_departure_conditions(safe_door_commands);
tel
```

Case Study – Modeling/Programming Flow

Aiming for SIL 4 certification → synchronous programming methodology

Listing 3: Time-critical C code generated by the HEPTAGON compiler

```
void passenger_exchange_tc(int train_position,
                           command safe_door_commands[8],
                           bool* departure_authorization) {
  int platform, door_map[8], safe_door_map[8];
  command door_commands[8];
  bool ontime1, ontime2;

  get_platform(train_position, &platform);
  send(0, &platform, sizeof(int), TC);
  ontime1 = receive(1, door_map, sizeof(int) * 8, TC);
  check_door_map(ontime1, door_map, safe_door_map);
  ontime2 = receive(2, door_commands, sizeof(command) * 8, TC);
  check_commands(ontime2, door_commands, safe_door_map, safe_door_commands);
  check_departure_conditions(safe_door_commands, departure_authorization);
}
```

Case Study – Modeling/Programming Flow

Aiming for SIL 4 certification → synchronous programming methodology

Listing 4: Non-time-critical C code generated by the HEPTAGON compiler

```
void passenger_exchange_ntc() {
  int platform, door_map[8];
  command door_commands[8];

  receive(0, &platform, sizeof(int), NTC);
  compute_door_map(platform, door_map);
  send(1, door_map, sizeof(int) * 8, NTC);
  compute_commands(door_map, door_commands);
  send(2, door_commands, sizeof(command) * 8, NTC);
}
```

Case Study – Model/Code Metrics

Software specifications metrics	
Functions	≈ 30
Requirements	≥ 100

Code metrics	Files	LOC
Heptagon sources	27	2741
C generated from Heptagon	70	7014
Additionnal C code	11	611

EMC^2 – Executable Application Models and Design Tools for Mixed-Critical, Multi-Core Embedded Systems

Selected Scientific and Technical Challenges

ARTEMIS Technology Conference, EMC^2 Workshop, October 6 2016

Point of View – *Models of Computation*

Moving from abstract MoC to *concrete programs*

- ⋆DF models: lessons from research tools (Ptolemy, StreamIt) and production (COMPAAN, ΣC)
- What is the role of static analysis in the abstraction of concrete programs into manageable MoCs?
- What is the role of profile-driven and dynamic analyses (see RWTH and Silexica's MAPS framework)?
- Competition for resources: responsibility of the tool or programmer or both?
- What should be dealt with statically, dynamically, how to integrate non-functional constraints and objectives?
- Interaction with "concrete programs", including memory management?
- Issues with expressiveness, modularity, and fundamental research needed to clean the chaotic MoC landscape

Point of View – *Programming Languages*

Research program in *language design*

- Build on the synchronous hypothesis in control systems
  $\longrightarrow$ relaxation: synchronization up to bounded delay (n-synchrony)
  $\longrightarrow$ efficiency: capture fine-grain schedules into the clock calculus
- Explicit parallelism in synchronous/block-diagram languages
  $\longrightarrow$ multi-threading and/or explicit distribution
  $\longrightarrow$ automatic synthesis of (loosely) time-triggered communications

Tools and software platforms

- Full automation, rigorously established, correctness by construction
- Runtime environment, (asymmetric) multi-processor configuration

Hardware platforms

- Facilitate isolation, functional and non-functional
- Enable timing-composable implementations
  ... while preserving efficient and scalable common-case performance
  ... it can be done without radical changes on the (micro)architecture

Software Crisis – Opportunities

Focus on high-productivity, high value software

- Higher level, reactive programming langages
- Correct-by-construction approaches
- Ubiquitous parallelism
- Ubiquitous distribution: elasticity, heterogeneity (edge/fog)
- ‘Non-functional programming’: time, resources, faults...

Invest in tools and reference platforms:
larger businesses, virtually vertical organizations, and funding agencies need to understand the urge and value in supporting a sound ecosystem of tools and platforms

- **Develop new computing modalities in HW and SW:** dynamicity, adapation, learning and reasoning, accuracy, trust, predictability, agile development… without throwing validation, verification, certification, quality away