Introducing Multi-Core at Automotive Engine Systems

D Claraz, F Grimal, T Leydier, R Mader, G Wirrer

Introducing Multi-Core at Automotive Engine Systems

D. Claraz[1], F. Grimal[1], T. Leydier[2], R. Mader[3], G. Wirrer[3]

1: Continental Automotive France SAS, 1, av. Paul Ourliac, BP 1149, Toulouse - France
2: Virtualised Reeled, 4, rue du romarin - 31650 Saint Orens de Gameville - France
3: Continental Automotive Germany AG, Siemensstr.12, Regensburg - Germany

1. Introduction / Motivation (why Multi-Core)

With the introduction of the new Euro 6, and Euro 7 emission standards for passenger cars, the combustion process of Engine Management Systems (EMS) needs to be controlled with an increased precision.

In addition, new vehicle architectures are introduced (increased integration of functions inside an Engine Management System), as well as new SW architectures concepts like AUTOSAR or the support of ISO26262.

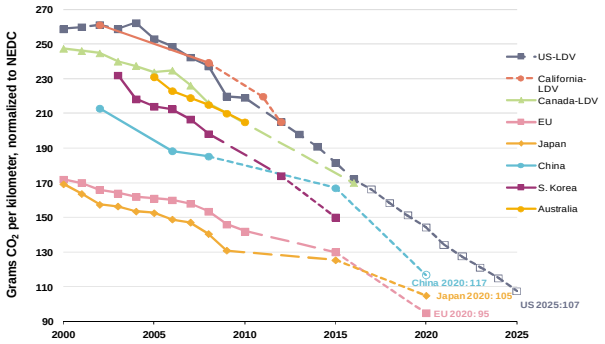

Fig. 1: Evolution of Emission standards (Source: European Commission, EPA)

All these evolutions result in an increased need for computation power.

To face this challenge, the increase of the CPU frequency (today up to 300 MHz), which has widely been used in the past, is not anymore an option, due to power dissipation limitations.
Therefore, the solution emerging on the market now, and promoted by our microcontroller suppliers, is the use of Multi-Core technology.

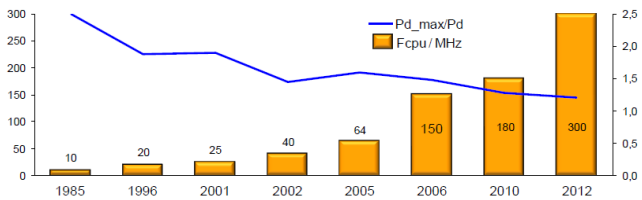

Fig. 2: Evolution of cpu freq vs. Power dissipation

Our objective in this paper is to present the status of Continental Automotive Engine Systems, concerning the introduction of Multi-Core technology.

In a first part, we will describe the challenges we have to face to make a legacy engine systems SW Multi-Core compatible.
In a second part, we will give some elements of the solution we have developed.
In a last part, we will give a status on where we are concretely.

Originally, our plan was to introduce the Multi-Core approach together with our new AUTOSAR-based architecture concept EMS3-PowerSAR®. But the migration of a full project to this new PowerSAR® architecture is not possible in a short time frame, and we foresee that our next projects will have to deal with a mixture of different architectures: AUTOSAR (different releases, different uses of same release, ...), PowerSAR®, Continental EMS2-MCR, Continental EMS2, OEM specific, etc..

Then, since December 2011, we focus on a solution to make our EMS2 “legacy” SW platform Multi-Core compatible. This is called EMS2-MCR (“Multi-Core Ready”).

EMS2-MCR is based on a PowerSAR® / AUTOSAR BSW layer, on which the migrated legacy ASW is integrated, together with components of the new ASW PowerSAR® architecture. Our objective was to distribute a legacy project on a 3-Cores ECU, and run a demonstrator-engine by end of 2012/beginning of 2013.

2. Challenges

a. Methodological challenges

The methodological challenges are multiple:

Backward compatibility:

Our SW-development process is highly reuse-oriented (“reuse by reference”), based on a concept

of reusable aggregates stored in generic libraries, and developed by generic teams. Customer projects, managed by customer-oriented teams, select solutions from these generic libraries, and configure them for their own context (configuring being different than modifying).

Such an approach requires that the generic library is an “asset” common to all projects, and means that the functions are developed and maintained once, in the generic teams.

Consequently, our first objective is to ensure that a function migrated to the new EMS2-MCR context must be still reusable as it is in a classical EMS2 Single-Core context (for instance w/o RTE), as all projects will not migrate simultaneously on the new HW platform. Furthermore, the effort to maintain 2 parallel branches of the same function (one Single-Core branch and one Multi-Core branch) would be too high: duplication of teams, of competence, …

Independence vs. Core allocation:

The design of the function must not take any assumption about its own and other’s core allocation. This is due to the fact that we are not able to define, once forever, the distribution of the functions on the Cores. The distribution will change over the time, due to new incoming constraints, due to our gain of experience, ISO-26262 considerations, load balancing, RAM consumption, customer requests, diversity of projects (Single-Core projects dual core projects, triple core projects reusing the same functionality), diversity and evolution of the HW (1 to 3 cores today, 5, 6 or more in the future?), …

Automatic protection of code:

The problem of concurrent access to shared resources, a classical problem in real time systems, is kept under control, in Single-Core systems, by two main hypothesis: scheduling strategy (preemptive, non preemptive), and priority scheme. These Single-Core hypothesis are just blown-up in a Multi-Core context, reaching to a special high degree of complexity.

Firstly, intra-core communication is dependant on task priorities, and is directionnal: Depending on the relative priority of 2 tasks, we can know if they interact with each other, and even better, we can identify which one interrupts the other one.
With inter-core communication, this is not anymore true: a low priority (low criticality) task of Core 1 can “interact with” a high priority (high criticality) task of Core 2. Furthermore, the interaction is bi-directionnal, now.

The second aspect is the type of interaction between the tasks: In intra-core communication, they are of three types: preemptive, non-preemptive and cooperative[1]: depending on the type of interaction, the risk is mitigated. Finally only in few cases, preemptive scheduling is used, which requires special attention for the developers, in term of data handling: one executable might be executed concurrently to another one (or to itself), and therefore concurrently access to a shared resource (race condition met).
In Multi-Core context, the inter-core communication is similar to a preemptive behavior, everywhere.

Another point is that the same aggregate might be integrated differently in different projects with different architectures, and still needs to be properly protected. A manual and a priori protection cannot comply with a big diversity of architectures, and it is likely that a protection pattern works for a context, but not for another one. At the end, the correct handling of such race conditions is in general error prone, and requires additional effort, increasing development time & cost.

For all those reasons, we have no other alternative than an automatic mechanism to protect the SW.

Efficiency of the protection:

Considering the particularity of Engine Systems - high coupling between the control functions – the efficiency of the solution gets high importance.

Basically, the resources to protect here are data, and they must be protected in an efficient way: The air mass entering the combustion chamber is consumed more than 6.000 times a second, while the engine rotation speed (rpm) is used 30.000 times a second, and in more than 300 different modules. In total an Engine System Software has to deal with 20.000 variables in around 600.000 of lines of code.

Different concepts are introduced to handle such situation in an efficient way. For instance, a standard approach of buffering data at a beginning of a task, or disabling interrupts at each access point, similar to what AUTOSAR implicit & explicit communication[1] proposes is not an option here.

[1] Cooperative scheduling could be called also « controlled preemptive scheduling », as interruption points are fixed at design time.

Improved integration:

The difficulties linked to integration in the Engine System domain have already been addressed, in a Single-Core context[2]. The introduction of Multi-Core inevitably increases the complexity of the architecture. A project with 30 operating system tasks in Single-Core will get up to 70 to 80 tasks in the future, or even more depending on the number of cores.

New integration constraints and artefacts will raise up, which will be defined at function level, to be considered at integration level. Different projects will follow different criteria for the distribution of the SW across cores. The total control of integration and architecture will be necessary for the control of data protection.

Human factor / applicability:

The solution should be understandable, and usable by the whole organization in a short time-period. We cannot make every SW developer (~600 World Wide) an expert of Multi-Core architecture. In particular they are used to architectural patterns and design paradigms on which our current platform is founded. The tools, method and processes are based also on this and the transition should be smooth enough to be easy to use and understand. Finally, the migration to the new concept should be fast and supported.

b. Migration

As already mentioned, our purpose is to migrate our SW library to the new Multi-Core Standard. As this migration is done while all projects are in development, we need to ensure that there is no regression introduced with the migration. A default Single-Core behaviour shall be ensured.

In order to speed-up the migration of hundred thousands of lines of code, a tool has been developed, which analyses the legacy SW, and introduces the necessary material into the code to make it Multi-Core compatible.
The basic cases, most frequent, are treated automatically, while some more complicated cases need to be treated manually. In all cases, a review of the migrated code is done by the function experts.

This has required an exhaustive identification of the use cases, in the code, quite a challenge knowing that some of the SW modules are more than 10 years old, and taking into account that architecture rules, coding rules might evolve with time.

Complex code structures and complex design patterns have been considered, and their applicability in a Multi-Core context has been studied. For some of them, new patterns have been introduced. For instance, the classical « double buffering » pattern has been replaced by exclusive areas, as it does not apply in all cases of core distribution.

(A) In low priority task:

```
// 32 bit counter
u16 shared_data[2];
u16 temp_lo, temp_hi;

BEGIN_CRITICAL_SECTION;
   temp_hi = shared_data[1];
   temp_lo = shared_data[0];
END_CRITICAL_SECTION;
```

(B) In preemptive high priority task:

```
// increment 32 bit counter
if( shared_data[0]==0xFFFF )
 {shared_data[0] = 0;
  shared_data[1]++;
 }
else
shared_data[0]++;
```

Fig. 3: Example of design pattern valid for preemptive Single-Core, but not applicable on Multi-Core

A last point to consider in the migration has been the training of all the function developers to the new standard. More than 200 developers have been trained in the last 2 years, not only on the code migration but on the whole Multi-Core concept, which means new methods, architecture, and associated tools. On the project side, Project Architects have also been trained to these new concepts.

c. Technical challenges

Which distribution?

The first question to answer when we switch from Single-Core to Multi-Core (e.g. 3 cores) is the distribution strategy: Do we have a dynamic, or a static distribution? What is the rationale to distribute this or that Aggregate on this or that Core?

As the objective of the migration to Multi-Core is to gain CPU load, the basic question is to define which policy is the best to reach a balanced load. Angle vs. Time computations? Short deadlines vs. long deadlines? Vehicle functions vs. engine functions? ASIL modules vs. QM modules?

A first approach would be a functional approach: The distribution over the cores is defined by the functional partitioning. For instance, all inlet related functions on 1st core, all setpoints related functions on 2nd core, and all exhaust related functions on 3rd core.
Another topology could be based on the dynamic architecture of the SW: all engine-angle related

aggregates (a clear specificity of combustion engines) on one core, all pure time dependant aggregates on another core.
A third approach could be based on the speed / deadlines of functions: Short deadlines on the first core, middle deadlines on the 2nd core, and long deadlines distributed on the 3rd core.

Each of these topologies has advantages and drawbacks. Due to the strong coupling of the functions (another clear specificity of combustion engines), they are not easy to apply, in particular when computation sequences matter. In effect, a classical 10ms Task might gather up to 150 Runnables which in many cases need to be executed in a well defined sequence. Therefore, the distribution of such task over 3 different cores can be done, only if the execution order is kept.

At the end, the code distribution is a multi-dimensional problem[2], and the allocation will be a balance between the buffering effort, the sequence needs, the customer request, etc... We foresee that we will need a high flexibility, and we will not be able to ensure for a long time that this or that function will always be mapped to Core 1.
Finally, the objective is to distribute the CPU load in an equilibrated way across the cores.

Therefore, we target a distribution based on runnables, rather than modules, or even compositions. At integration time, the project will have the possibility to move one runnable from one core to another, taking into account its own policy.

Split of components?

Another question we faced concerns the ability to split SW-Components on different cores, something today not authorized by AUTOSAR.

As most of the SW-Components are not monolithic in terms of dynamic architecture, they need to be integrated in different Tasks / Integration containers. If we mix this status with the need to ensure a correct sequence between runnables, we see that it shall be possible to locate different runnables of the same component on different cores, depending on the integration Task.

Moreover, complex cases like multi-rate data (data produced at different rates, e.g. top dead centre and 10ms), or even multi-rate executables needs to be supported. This includes executables called by different tasks on the same core, or even on different cores.

Scheduling strategy

A last question concerns the Task scheduling: Is a dual core architecture a simple duplication of the simple core architecture? By evidence, no, because the complexity of the whole system would not be controllable : A typical Single-Core project contains up to 30 OS Tasks, plus some 20 containers for system transitions, which means around 50 integration containers, where individual runnables need to be plugged. A simple duplication of these containers on each core would lead to a high complexity, and probably an unfeasible system, in term of deadline fulfilment.

So, an adequate architecture is set-up, with new concepts. For instance, for a given rate, some tasks might be either parallelized, or chained. Both techniques have advantages and drawbacks.

d. Starting point : Legacy SW

A classical approach, when introducing a new technology like Multi-Core would be to start from a white page, design all functions new, and even take some assumptions concerning their Core allocation. But, the corresponding effort and delay would be too high, and as already mentioned, a fixed allocation does not fulfil our needs. Therefore, our starting point is a legacy SW. But what does it mean exactly?

A typical legacy Engine Systems SW project is built of around 2.000 SW modules (1.300 for asw), more than 8.000 executables, and 1.500[3] of them to be integrated in a task. Around 150.000 data access points need to be considered for data access protection.

In term of mechanisms, ahead of the multi-rate data and executables already mentioned, the reality of a legacy SW is a complex call structure sometimes reaching in some cases more than 10 nested call levels, with indirect calls, with complex data types, etc…

One typical topic is the « consolidation » of the data and control flow in such « non flat » architecture.

[2] Will be addressed in a further conference.

[3] Might be divided by a factor 3 by means of adequate design pattern. See [3]

Another difficulty, inherent to legacy-SW is the mixture of formats: AUTOSAR files mixed up with legacy files, and/or with object code from our OEMs.

3. Solution : Integration & Protection

In this chapter, we will show the main principles of the solution developed to integrate runnables and to protect data in the EMS2-MCR SW, satisfying the above constraints. As a prerequisite, the legacy code had to be prepared to be MCR, which means (roughly) the replacement of all direct accesses to global variables by GET/SET APIs, to allow their protection by an adequate process.

a. Concepts

This solution is organized around 2 main axes: Integration and Protection.

Runnable Integration

Runnable integration gets a higher degree of complexity in Multi-Core context, compared to a Single-Core context:
- The total number of tasks and integration containers is largely increased compared to Single-Core (even if not multiplied by the number of cores)
- The criteria to choose between one and the other task on one or the other core might be more complex (multi-dimensional problem)

In addition, we want to benefit from the "new" platform to improve our integration process.

So, the solution to this problem has 2 facets:

At Component or Composition level, integration constraints are specified: System Events (stimulus), , Phases[3], Timing properties (periods, deadlines …), Sequence Needs, Core Affinity Needs are defined, using a component modelling tool, CoMod.

On project side, these constraints are reused by a runnable integration tool, RunIn, enabling the integration of Runnables at the correct position in the correct Task. Conflicts are shown and can be solved by the integrator. Runtime of Runnables can be imported in order to properly balance the computations across the cores. Additional constraints can be defined at integration time as well.

Data access Protection

As explained before, data protection in Multi-Core context becomes also much more complex, compared to the "automatically protected" mode of Single-Core Cooperative context.
We remind here, that cooperative scheduling is widely used at Continental, as it has the following benefits: First of all, in most of the cases, it is largely sufficient to fullfil our timing constraints. Secondly, it is resource efficient (limited context switches). And last but not least, it provides us a data protection "free of charge": no particular design constraint, no tool is needed to ensure data protection. Computations simply do not interrupt each other.

With Multi-Core now, as any task on any Core can disturb any other task on any other Core, the situation is quite different.

Two kinds of problems are identified: stability (an information gets a stable value along a code sequence , like 2 different Runnables), and coherency (2 information are coherently acquired at a certain point in time).

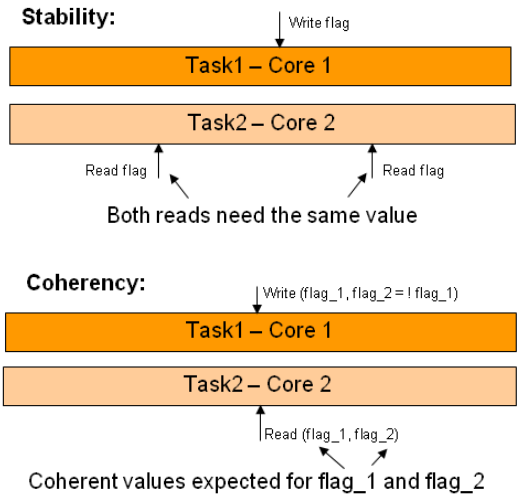

Fig. 4: Stability & Coherency issues

Here again, our solution has 2 facets:

At component development time, protection constraints are specified, and at project integration time, the constraints are used to protect the SW.

One cornerstone of the strategy is that only those artefacts which require protection will be protected. We made this choice because we estimate the cost of protecting everything too high, due, once again, to the high coupling and complexity of Engine Systems. In addition, an unnecessary buffering may have

negative impacts on the SW behaviour, not only on the resource consumption point of view. Finally, not all data and data accesses are critical, and we have to focus on those ones.

So, the component modelling tool, CoMod, supports the specification of so-called “Consistency Needs”. To be noted that this concept of Consistency Needs has been promoted and then introduced in AUTOSAR (and available in 4.1.1).

At project side, these constraints are taken into account by the integration tool. RunIn analyses the project and computes all required protection, taking into account the complete call and data graph, the specified ConsistencyNeeds and the project architecture. It generates all necessary C and ARXML files for the further SW build.

The strategy chosen to protect the data is based on a buffering concept:

When a variable is accessed in one executable, if we identify a race condition, and if a protection is required for this variable, then it is copied into a buffer, and the executable works on this local copy. If the executable modifies the data, the modification is done on the buffer, and the value of the buffer is copied in the global variable later on.

The principle is similar to the AUTOSAR concept of implicit communication.

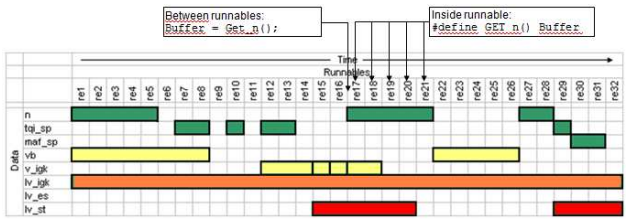

Fig. 5: Buffering concept

To enable this, in the migrated code, special access APIs have been introduced, which allow an eventual re-direction.

```
void aggr_module__1s(void)                    - Calculation at 1s
#define EXECUTABLE_ID EXECUTABLE_ID_aggr_module__1s
#include <aggr_module_mcr.h>
void aggr_module__1s(void)
  if (GET_var1())
        then                                  else
  SET_var2(NC_IDX_1, GET_var_4());            /*Do Nothing*/
  SET_var3(NC_IDX_2, GET_var_5());
```

Fig. 6: Migrated code.

Depending on the conditions, the GET and SET macros will be redirected to buffers. Fill and flush routines are added to the Runnable scheduling, in order to initialize the Buffers, and to copy back the value in the global memory.

In order to minimize the buffer consumption, the same Buffer can be reused across different executables, for different data, in the same task. This is part of our optimization process.

The code patching strategy, similar to what is done by a classical RTE, presents several advantages compared to the object file patching strategy, also used at automotive engine systems. Some of these advantages are:

- It can be used early in the process, in particular before the sw Build. Therefore, the integration choices can be influenced by the Buffering results.
- The output can be verified easily
- The process is independent of any compiler version, or compiler option
- It is an open solution which easily supports complex use cases, like multi-rate executables

b. Architecture control

One element of the strategy is to identify and control properly the project architecture. The tasks and integration containers, their allocation to the cores, their timing properties, and their priorities have to be defined, as they are important factors for the integration, as well as for the protection. For the data protection, for instance, the interactions between the tasks have to be identified (direction, type, …). The resulting buffering and copy routines will depend on it.

System Event Imp…	Core	Min. Period	Deadline	Priority	…	Cooper..
Task_P0_1000msT0	Primary_0	1000.000	500.000	4	1	Group 1
Task_S1_10msT1	Secondary_1	10.000	10.000	6	2	Group 1
Task_S2_10msT1	Secondary_2	10.000	10.000	6	2	Group 1
Task_S1P_1msT0	Secondary_1	1.000	0.200	10	1	None

Fig. 7: Task configuration

In order to ensure that the theoretical view on the architecture, used for integration and protection, is consistent with the real implementation in the ECU, different configurations are generated in a seamless environment: typically RTE and OS configurations, as well as all the required « glue code ».

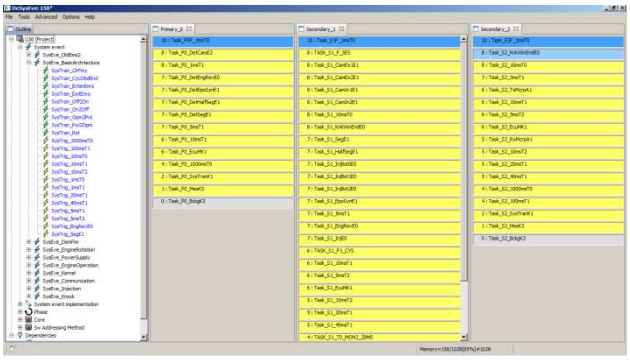

Fig. 8: Project Architecture: Task allocation to cores

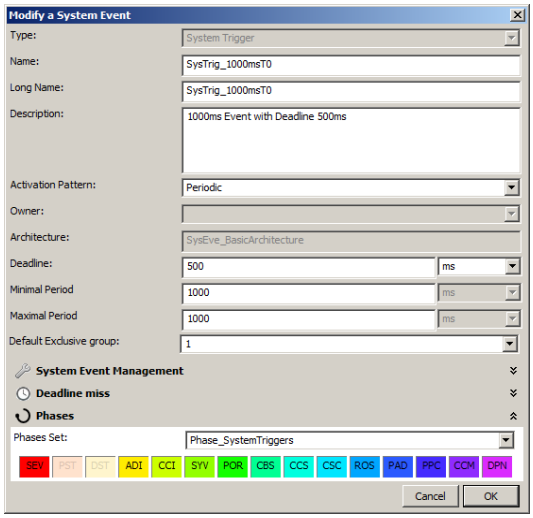

Fig. 9: A SystemEvent specifies timing properties and Phases of stimuli

One last element of the architecture control, particularly important in a strong reuse context, is the introduction of a Reference Architecture, which is used as well by the Function Developers when the specify their integration needs, as well as the projects when they build their project architecture. The Reference Architecture is managed in a centralized way, and defines the main choices of the new Multi-Core platform.

c. Need specification

On the developer's side, 2 types of Needs can be specified: Protection Needs and Integration Needs.

Based on an analysis of the data & control flow of its module, the function developer is able to specify the coherency or stability needs. Roughly either he specifies 2 (or more) variables which have to be read in a coherent way in one Runnable. Or he specifies 2 (or more) Runnables which need to get the same value for a given variable. This represents a new kind of activity for the developers, which means not only a new task, but a new kind of topic, of concept he has to think of.

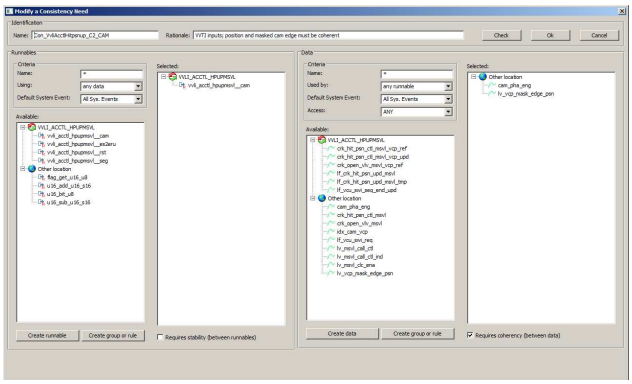

Fig. 10: A Coherency Need

Concerning the integration, for each of its Runnables, the developer specifies the required timing and integration properties. In order to avoid a big variety of needs, a set of standard System Events is provided through a Reference Architecture. The developer has only to select the one of interest, as well as the Phase the Runnable corresponds to. Doing this, he defines a « RunnableEvent », which is a similar concept than the AUTOSAR concept of RteEvent (with the notable difference that no invocation mechanism is specified here, which gives more flexibility on how to solve this invocation).

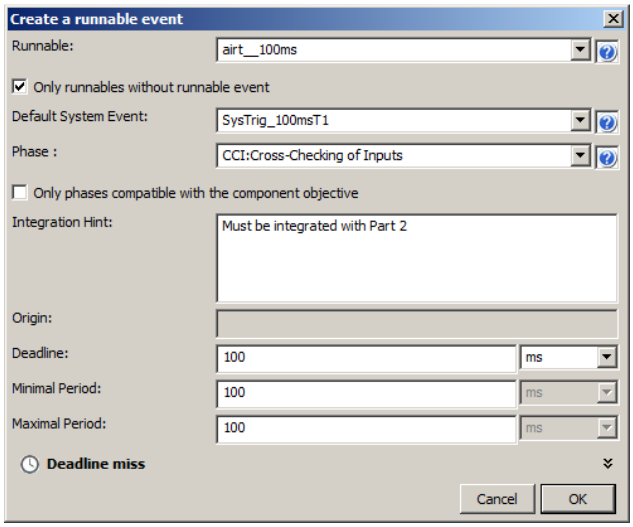

Fig. 11: A Runnable Event (Integration Need)

d. Integration & Protection

The integration and protection process takes as input the previously defined project architecture, the specified Needs of the individual modules, and the SW cartography. This last element consists in resolving for the whole SW all data accesses. All

types of accesses have to be identified (arrays, pointers, structures, …), as well as their multiplicity and consolidation.

The previously defined RunnableEvents are used by the integrator to select the right task for each runnable. The specified Phase and Sequence constraints are used to define the position in the task. In case a mistake is done, the integrator is informed by special warnings.

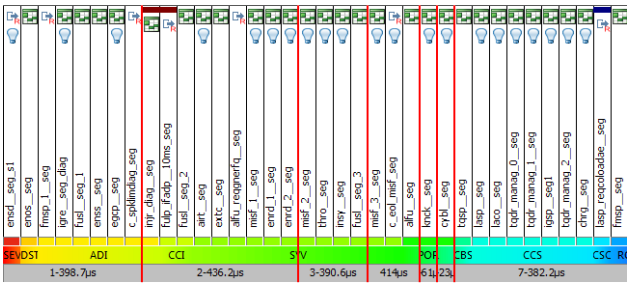

Fig. 12: Integration of Runnables in a task

In the above picture, we see the Runnables integrated in a task, and their respective Phases.

When all Runnables are integrated, the protection process is launched. The tool takes into account all parameters, and defines which variables need to be protected, how, and at which point in the task.

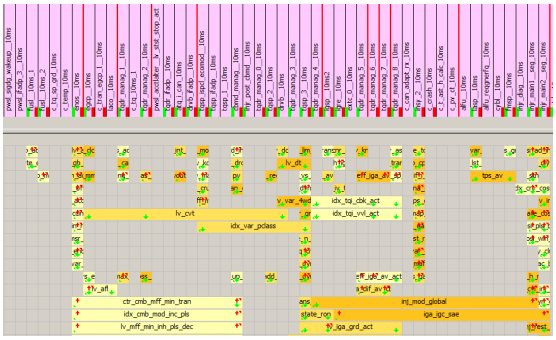

Fig. 13: Data protection by Buffering

Ahead of the simple graphical view, all the necessary files for the build process are generated: the buffer fill and flush routines; the redirection of the GET and SET macros in the modules; the task bodies, etc...

For instance, in the below picture, we see how the fill routines are inserted at certain points in the tasks. Like already mentioned, we see that the buffers are not initialized only at beginning/end of a task.

```
void Task_S1_1000msT0_01(void) {
dacoFill_01() ;
   airt__1000ms() ;
   c_n_max_tol_st() ;
   c_t_ast_obd_1s() ;
   enrd__1s() ;
   fusl__1000ms() ;
dacoFill_03() ;
   insy__1000ms() ;
dacoFill_11() ;
   egtr__1000ms() ;
}
```

Fig. 14: Fill and Flush routines inserted in tasks

Then, the bodies of the copy routines are produced:

```
void dacoFill_02(void) {
  pdaBuffer_01=inh_igc ;
  pdaBuffer_02=pwm_etc ;
}

void dacoFlush_03(void) {
  pwm_etc=pdaBuffer_02 ;
  vp_tps_1=pdaBuffer_11 ;
  vp_tps_2=pdaBuffer_13 ;
}
```

Fig. 15: Bodies of Fill and Flush routines

And the accesses to the global data in the code are redirected to accesses to the buffers:

```
/* Patch Area */
#if (PDA_EXECUTABLE_ID==RUNNABLE_ID_c_acp_10ms)
#undef  GET_lv_igk
#define GET_lv_igk() ((flag) pdaBuffer_071)
#undef  SET_tq_loss_acc_inp
#define SET_tq_loss_acc_inp(x) pdaBuffer_004=(s16)(x)
#endif
```

Fig. 16: Redirection of data access within Executable

In addition to these 3 elements, various configurations are generated, which ensure a perfect fit between model and real implementation in the ECU.

4. Status

Since we started to develop this approach, in December 2011, most of the 1.300 Sw-modules have been migrated, in the scope of a first pilot (gasoline) project. A new (diesel) project is currently started, which will reuse partly the already migrated aggregates, and which will require new migrations.

The migrated aggregates are reused in the running Single-Core projects, w/o noted bad effects: That demonstrates their backward compatibility.

In the first pilot project, more than 400 consistency needs and 2700 RunnableEvents are specified.

When applied, together with complementary automatic protection strategies, near to 2.000 buffers are used to buffer around 8.000 accesses (out of 130.000). The measured overhead is in the range of 6% of CPU load, distributed over the 3 cores, which is an acceptable value.

The first distribution applied on our pilot shows the following results, in term of CPU load evolution:

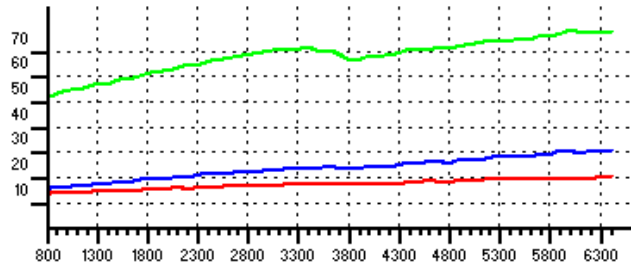

Fig. 17: Core-wise CPU load (as function of engine rotation speed) after distribution

The different curves correspond to the CPU load of the different cores. This current distribution is only a first step, and under permanent evolution. Thanks to our methodology, we have a high flexibility to re-organize the distribution. Nearly just a question of “drag-and-drop” .

Most of our engineers are getting trained to the concepts of Multi-Core, and are able to specify Protection and Integration Needs, while the project Teams are able to cope with the new architectural artefacts, and integration means.

Concerning real customer projects, a first project with a premium german OEM started to use this method by beginning of 2013 (SOP End 2014, 200ku/year). This first project is split over 3 cores, and uses preemptive scheduling. With the same OEM, we start another project, which also uses 3 cores, but with a different distribution of the SW. Another project with another German premium OEM is also started based on our pilot, also distributed on the 3 cores. Finally, another project for a US OEM will use also this technology, but in a limited way, by using only 2 cores.

5. Conclusion

Ahead of a simple Tool Suite, the EMS2-MCR solution is a methodology based on a series of new EMS3--PowerSAR® concepts implemented coherently across the process: Reference Architecture, System Events, Synchronized Transitions, Phases, Runnable Events, Consistency Needs, Sequence Needs, Core Affinity Need etc ...

These concepts, originally planned to be used within an AUTOSAR-based platform, have been quickly tailored and applied on a legacy non-AUTOSAR Platform. This has been made possible by an abstraction of our needed concepts vs. AUTOSAR.

These concepts, and the method and tools supporting them are now at common use at Continental, and different Customer projects are already developped following this new standard.

But, this is the beginning of the story, and we need to improve and enrich the solution. For instance, the integration process has to be re-considered due to Multi-Core needs. Schedulability, SW distribution, ASIL introduction, shared development, validation, are on our table for the coming year.

Of particular importance will be the integration of AUTOSAR SW-Components in our framework. The questions will be about the mixture between legacy non-AUTOSAR SW and AUTOSAR SW, the efficiency of the RTE (and therefore, which use?), the introduction of the new concepts (Consistency Needs, ...), and improved integration means, the maturity of AUTOSAR in term of Multi-Core, ...

Finally, the step to Multi-Core suddenly provides us a duplicated computation power, for the same ECU price. This opens the door to an increased integration of PowerTrain domain, at least. The Multi-Core revolution is to be expected in the full system architecture, much ahead of the pure SW area.

6. References

[1] AUTOSAR, *AUTOSAR 4.1.1 Specification of RTE - AUTOSAR_SWS_RTE 3.3.0*, 2013.

[2] D. Claraz and M. Niemetz, “Engine management software dynamic architecture versus integration,” in *ERTS-2008*, 2008.

[3] D. Claraz, S. Kuntz, U. Margull, M. Niemetz and G. Wirrer, “Deterministic Execution Sequence in Component Based Multi-Contributor Powertrain Control Systems,” in *ERTS2-2012*, 2012.

[4] R. Davis and A. Burns, “A Survey of Hard Real-Time Scheduling for Multiprocessor Systems,” *ACM Computing Surveys*, 2011.