A Top-Down Constraints-Driven Design Methodology for Powertrain Control System

M. Antoniotti[(a)], A. Balluchi[(a)], L. Benvenuti[(a)], A. Ferrari[(a)], R. Flora[(b)], W. Nesci[(b)],

C. Pinello[(a)], C. Rossi[(b)], A. Sangiovanni Vincentelli[(a,c)], G. Serra[(b)], M. Tabaro[(b)]

[(a)] PARADES G.e.i.e., Via S. Pantaleo, 66, 00186 Roma, ITALY

[(b)] Magneti Marelli S.p.a., Via del Timavo, 33, 40134 Bologna, ITALY

[(c)] Department of Electrical Engineering and Computer Science, University of California at Berkeley, CA 94720, U.S.

***Abstract**: In this paper a top-down, constraint-driven design methodology for powertrain control systems design is presented. We identify five levels of abstraction for this hierarchical methodology: at the highest level of abstraction, system specifications are captured by a FSM-like model that defines the desired behavior of the powertrain in a number of regions of operation covering the entire domain. In each region, the specifications are formally captured in terms of objectives (performance indices) and constraints. At a lower level of abstraction, a functional decomposition structures the system into a set of "functions", each with its own objective functions and constraints derived from the overall system specification. The design problems related to the implementation of function behaviors are solved by means of a set of basic algorithmic building blocks called operations. In parallel to the operation determination, we select a network of components (an architecture) that is the basis for the final implementation of the system. Estimation and analysis are used to select the most performing architecture that satisfies the constraints. At the end of the design process, the actual component design takes place if the components in the selected architecture are not available in a library. At this stage software design that customizes programmable components to implement a large number of operations is carried out. The methodology is exemplified by the application to the design of control algorithms for the cut-off problem.*

1. Introduction

This paper presents a top-down, constraint-driven design methodology for the design of embedded systems used in the control of automotive engines. The work presented in this paper is the outcome of a long standing working relationship among the University of California, Berkeley, Cadence Design Systems, a design services and tool company, and Magneti Marelli, a large European manufacturer of electronic subsystems for the automotive market.

The design problem addressed with the proposed approach can be described as follows. From a set of specifications given by a car manufacturer to the embedded system provider:

- find a set of algorithms that control the engine to match the specifications;
- implement the algorithms on a mixed mechanical-electrical architecture consisting of programmable components, such as microprocessors and DSP's, application specific integrated circuits, and physical devices such as sensors and actuators.

In current practice, the implementation architecture of the electronic component is fixed a priori, (micro-controllers with fixed-point arithmetic units). The specifications and the control algorithms are so much tuned to this architecture, that it becomes very difficult to verify in an effective way the behavior of the system. In addition, the control algorithms are based on simple,

intuitive and often open loop rules. Our team embarked on an ambitious project whose goal has been to redefine the entire design process according to these principles:

- the specification are given at a high level of abstraction so that they are independent on implementation decisions;
- the control algorithms are designed so that formal correctness and optimality properties are ensured by a rigorous mathematical framework;
- the abstract representation of the design is mapped to an architecture using estimation techniques that guide the selection process
- the implementation architecture of the electronic component is selected among a set of candidates (e.g. a 32-bit micro-controller with Floating Point Unit, a 32-bit micro-processor with a DSP co-processor, a multiprocessor architecture with an 8-bit I/O processor and a 16-bit micro-controller with a DSP unit) in such a way that cost, reliability and time to market are optimized.

The goal of the presented design process is to shorten the design cycle (now of the order of 5 years) by a very sizable amount (by 3 years and more!) while reducing the cost of the electronic system and increasing its quality and functionality. The conscious reuse of previously designed blocks and the capability of taking decisions at early stages of the design process are key points to this end. The design process, shown in figure 1, is described as a set of successive refinement steps from a level of abstraction as high as possible all the way down to the details needed for the final implementation. The process has been conceived to be applicable in almost all embedded design problems, however, to be more concrete, we will use the Engine Control Sub-system design to illustrate the basic principles. In our approach, we identify five main levels of abstraction:

System level. At this level, the specifications are captured and analyzed. The supplier and the customer, in the case of the engine control unit, a car manufacturer, in general agree upon the specifications. The system "car" is viewed as the system converting the driver's commands conveyed by a variety of input devices such as clutch, gas and brake pedals, gear stick, cruise control settings, into actual motion and other outputs of interest to the driver or to some regulatory body, (governments or car manufacturers associations) such as emission levels, noise, fuel consumption. Constraints are placed on these outputs and objective functions are set to determine the quality of the car as specified by market analyses performed by the car manufacturer. According to the market segment targets, a car in the high-performance range will be required to respond to a fast push of the accelerator pedal within a very short time, with little regard to the fuel consumption needed to achieve this goal, while an economy car will be required to respond as well as possible to the command maintaining the fuel consumption below a certain threshold. Since some of the key mechanical components of the car, such as the engine and the chassis, are in general not part of the embedded system design, they can be considered as given. Thus, the goal is to design the engine control sub-system so that the car with that particular engine and chassis performs as specified. We advocate that the specifications be clearly stated and negotiated between customer and supplier to make sure that they are realizable within the budget and time allowed for completing the design. Unfortunately, most often than not, the specifications are determined according to an informal process that leads to misunderstanding and late product delivery. A simulation environment together with an engineering spreadsheet tool is highly desirable to support the process at this stage.

Function level. Once the specifications are determined, the design process inside the supplier company begins. Because of the complexity of the problem, a decomposition of the system into interacting simpler sub-systems, called *functions*, is clearly a key step towards a good quality design. The decomposition is useful if it leads to a design process that can be carried out as independently as possible for each component. The decomposition process has to "spread" constraints and objectives among the components so that the composition of the behaviors of the components, made feasible and possibly optimal with their own constraints and cost functions, is

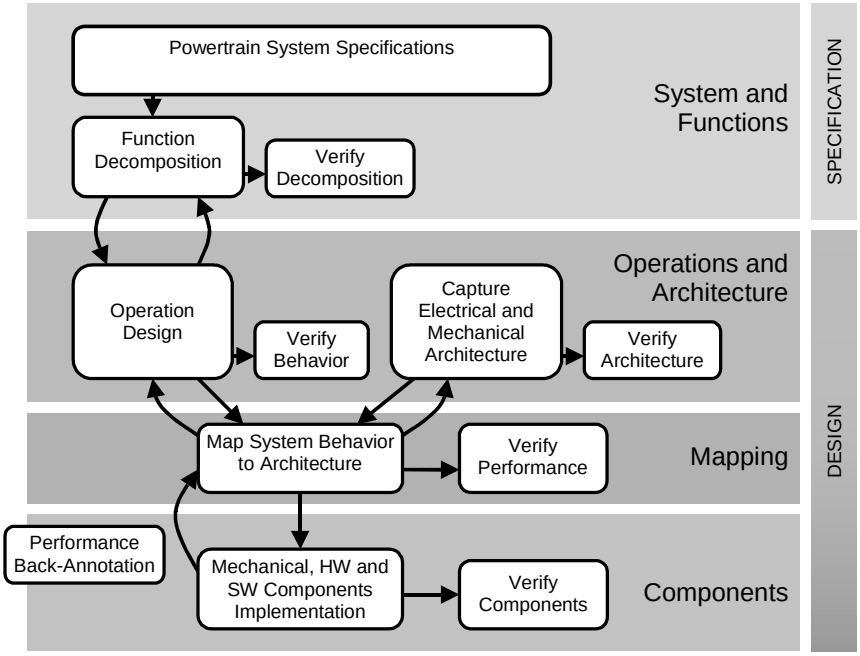

Figure 1: Overall view of the design methodology

guaranteed to meet the constraints and the objectives of the overall controlled system. Since in general it is difficult to decompose the system into independent parts, the determination of the local objectives and constraints has to be the result of a careful trade-off between the desire of maintaining optimality at the global level and the one of easing the design task for each of the components considered in isolation. This decomposition has to be based on the understanding of the physical process of interest. For the engine control case, the variables to be controlled are the air-fuel mix, the injection timing and the spark timing while the objective and constraints are stated in terms of emissions and motion of the car. Motion is the result of the application of torque to the driveline. The engine, via the combustion process, produces the torque. Emissions are the result of the chemical process in the catalytic converter that is fed the exhaust gases of the combustion process. The combustion process in turns is controlled by the quantities introduced above. The air-fuel mix is the result of two processes each determining the air and fuel quantity to mix. From this simple analysis, it appears that motion generation and emission generation are the results of a cascade of chemical-electrical-mechanical processes and that each can be controlled somewhat independently once the local constraints and objectives are set. In section 3, we will present this example with more details.

Operation level. Once the decomposition of the problem has been performed, each sub-problem has to be solved with its constraints and objectives. Each sub-problem solution is expressed in terms of basic building blocks, called *operations*, for re-use. Each operation is given a classification in terms of its nature, e.g., measurement, actuation, and control operations. At this

level, we start deciding “how to solve” the problem while at the system and function level we were dealing with “what is the problem” to solve. If, indeed, one or more of these sub-problems are not solvable, we need to revisit the previous design step to adapt the local objectives and constraints or to change the decomposition itself. Note that we try to close the design loop as early as possible to avoid major re-design steps at a later stage that can yield so much damage to schedule and profitability.

Architecture level. The operations are still defined abstractly and not directly related to physical devices. At the architecture level, we determine the actual interconnection of components (architecture) that implement one or more operations. This step consists of matching abstract operations with a collection of available or “to-be-designed” devices trying to optimize cost, size, reliability, time-to-market. Components can be classified as electrical, chemical, or mechanical devices. Some examples of components are the pressure sensor, the intake manifold, and the microprocessor. Note that the microprocessor or any other software programmable component is very flexible since it can be used to implement a large class of operations. At this level, the actual performance of the system can be more precisely estimated. If the estimation indicates that the objectives and constraints are not met, the choice of operations has to change accordingly. Once more the engineering change loop is closed as early as possible.

Component level. This step involves the actual design of the components of the architecture determined in the previous step if they are not available in the library or if their use requires some degree of customization as in the case of electronic programmable components, where software must be developed to customize the processor for the particular operation under consideration.

The operation and architecture design process is tightly linked. Decisions taken on what operations to implement have an obvious impact on the architecture and vice-versa. Hence it is very difficult to de-couple these two steps. Indeed, the mapping process from operations to architecture is critical to obtain a good design. In our methodology, mapping is performed continuously so that cost and performance estimation can guide the design decisions. In figure 1, the sub-steps corresponding to operation and architecture design all the way down to actual implementation are shown. Note that operation and architecture abstractions are shown in parallel to stress that the two design processes have to be tightly linked. It is in this phase that we address the key problem of software design. Software is the by-product of mapping a set of operations to the programmable components of the electronic architecture.

In summary, the overall methodology is hierarchical and consists of five main levels and of the mapping steps that take from one level to the next. Each step is characterized by analysis and optimization that can be carried out with the appropriate tools. The goal is to make the design process rigorous and as well defined as possible and to re-use as much as possible any of the design components for the present application as well as for future applications.

The paper is organized as follows: In Section 2, the system level is shown for powertrain control. In Section 3, the function level is described in more details with examples drawn from engine control. In Section 4, the operation level is presented. In Section 5, the architecture level and the associated mapping steps are described. In Section 6, the component level design problem is addressed with particular emphasis on software generation_Finally an example of application of the methodology for a particular region of operation of the engine, i.e., cut-off, is carried out.

2. System behavior specification: constructing the reference model

The car manufacturer gives the system specification in terms of how the vehicle should react to the inputs given by the driver. In this paper, we restrict our attention on the specifications related to force requests, i.e. we do not deal with requirements on trajectory control of the vehicle. The assumption is that the driver requests force by using the gas, brake and clutch pedals, and a manual gear shift. In general, specifications are stated in informal ways and are embedded in the language of the contract between supplier and customer. Not only, but the capture of the specifications as interpreted by the supplier is often obscured by other considerations involving the implementation of a system, thus making difficult to re-use part of the solution as well as verify whether the specifications are met.

In our approach, the specifications are captured using a hybrid model as shown in figure 2. The states of the top-level Finite State Machine (FSM) correspond to different regions of operations of the engine. The transitions are determined by the action of the driver on the input signals or by engine conditions. Each region of operation is characterized by a set of constraints related to driving performance, such as comfort and safety, or gas and noise emission, and a cost function that identifies the desired behavior of the controlled system. The controlled system is represented by a model that includes ordinary differential equations as well as discrete components. The goal of the controller is to act on the inputs to the plant (the throttle plate angle, the injection pulse duration and the spark advance angle) so that it behaves according to the specifications summarized in the FSM.

Figure 2 shows the FSM that specifies the behavior of the vehicle. The initial state is the Stop state that corresponds to the engine being off. From this state, the driver causes the transition to the Startup state, turning the ignition Key to the startup value. From the Startup state the FSM enters the Idle state, if the gas pedal signal is 0, otherwise enters the RPM Tracking state. The desired behavior of the controlled system in the Idle state, for example, is the following: the force F_G acting on the vehicle has to be zero and the crankshaft revolution speed n must be regulated to a reference value, e.g. 500 rpm, with an excursion of up to ±20 rpm while minimizing fuel consumption. Moreover, the transient response with respect to sudden loads of 25 Nm must have a 'settling time' of 2 s and an excursion less than 40 rpm. Last, the air-fuel ratio must remain within $\pm 1^{0}/_{00}$ of the stoichiometric desired level.

In the RPM Tracking state, the engine is supposed to adjust its RPM to match a prescribed relationship with the gas pedal signal. When the transmission is engaged, the Force Tracking state is entered. In this state, a particular torque profile, that depends on the gas pedal position, the gear and the RPM, has to be realized with constraints on the settle time and with cost function related to fuel consumption and drive comfort. The Fast negative force transient state is entered if the gas pedal is suddenly released. In this state, a step reference for the requested force is considered and a different control law is used. This control law should minimize the transient time subject to very tight drive comfort constraints. However, if no drive-by-wire control is available, a feasible solution may not exist. In this case, the optimization problem is reformulated so that the driving comfort constraint becomes the cost function. When the gas pedal is pushed again or the transient is elapsed, the transition to the Force tracking state is entered. If the gas pedal is completely released and the minimum RPM is approached, the transition to the Idle with transmission on state is enabled. In this state, the RPM is kept constant at a prescribed value until the gas pedal position or the RPM triggers the transition to

the Force tracking state. If the gas pedal is pushed quickly, then the transition to the Fast positive force transient state is enabled. This state is the mirror image of the Fast negative force transient state. Here the torque profile to be followed is a positive step and the control law minimizes the transient time subject to fuel consumption and comfort constraints. Finally, when a cruise control is available, a Speed tracking state is present. If the driver turns the Key to the stop value or if the engine turns off, the Stop state is reached.

Although, this scheme is very simple, it has quite an impact in clarifying the objectives of the design and how to decompose the problem into well defined sub-problems. In addition, this way of specifying the problem has an impact on the final implementation, reducing the amount of complexity and redundancy in the code. The requirements captured by the diagram can be considered as the reference model for the control problem. In addition, the *structure* of the FSM is the same independently of the car manufacturer or of the car range. Only the parameters of the cost and constraint functions change, thus favoring maximum re-use.

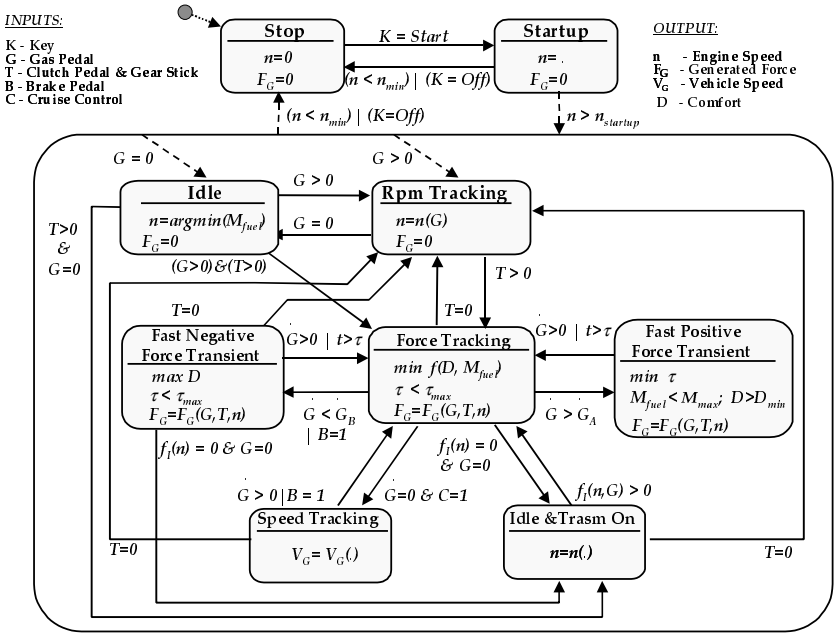

Figure 2. FSM of the regions of operation describing system specifications.

3. Function level design

In the previous section we specified the desired system behavior, by identifying several regions of operation characterized by different performance indices and constraints. In our design methodology we advocate the decomposition of the overall design problem into semi-independent sub-problems, called *functions*, to cope with complexity.

Each function is responsible for a particular physical process within the system. The set of functions we propose consists of *communications, motion generation, exhaust management, combustion, ignition, mixing, fuel management, and air management.* The communication function is related to electrical communications among components and is not tightly linked with the others. For this reason, it is not discussed here. The interactions among functions (except for communications) are described by a direct acyclic graph (see Figure 3). The arcs are associated

with the physical internal variables on the interface between functions. The source nodes are the motion generation and exhaust management functions, which are mostly responsible for the interaction of the system with the environment and are subject to constraints such as drivability, performance and pollution control. The drain nodes are related to the physical inputs of the plant, i.e. throttle, injection, ignition. The inverse graph, between the drain and the source nodes, represents the actual interaction of the physical processes. The reasons for this choice of functions for the engine control problem are the "weak" and well characterized dependency among the functions.

Once the functions are selected, we need to determine their constraints and cost functions so that solving the relative problems independently still guarantees a good performance of the overall system. to do so, we visit the graph of functions following a topological sort. At each step, the behavior of the current function is established by means of a tradeoff process with the functions associated with its successors in the graph. The selected behaviors are such that the requirements are fulfilled with the lowest cost. If this procedure does not yield a feasible solution then the graph sort is arrested and the procedure failure is reported at the predecessor nodes

In order to evaluate performance indices and variable constraints, the design at the function level makes use of models of the functions at a very high level of abstraction. In fact, the purpose is not the design of control algorithms, which is done at the operations level, but to assess whether or not system specifications can be met, by solving sub-problems independently.

Given the function interaction graph for the engine and powertrain control design problem, consider the gas pedal input as a force request, which is handled at the **Motion Generation** function. The force request is transformed by the function into a torque on the crankshaft requested from the **Combustion** function. This function, in turn, requests a given mass of the air/fuel mix, a given air/fuel ratio (λ) and other variable profiles from the **Mix** function to obtain the requested torque. The **Mix** function can request different air and fuel profiles to the

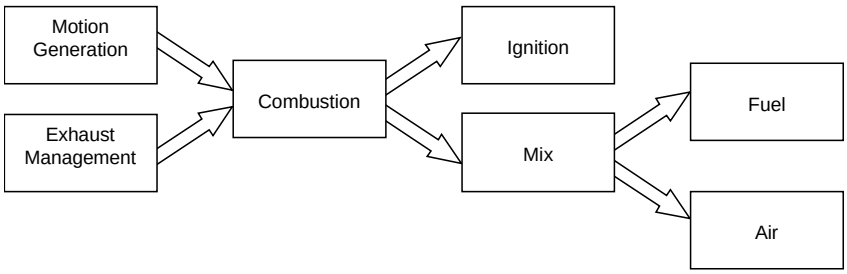

Figure 3: Graph of interaction among functions representing the propagation of specification constraints

downstream **Air** and **Fuel** functions for different operating conditions to fulfill the requests of the **Combustion** function. The fixed mechanical link between the gas pedal and the throttle in traditional cars, restricts the kind of air-flow profiles that can be produced. Thus it may very well be that, during this phase of the design, we discover that the requests are too tight and that there is no chance of making the bounds. If this is the case, the information has to be fed back into the graph structure until the source nodes are reached and a new sort process begins with different cost and constraints distribution. If we assume the presence of Drive-By-Wire (DBW) technology, the set of constraints that can be dealt with in the **Air** and **Fuel** functions is larger, i.e. the **Air** function can respond in a very flexible way in a wide range of operating conditions, since the throttle valve can be fully regulated by digital devices. The example shows how our

methodology can be used for different scenarios, where a key feature choice (with or without DBW) has a deep and widespread impact on the overall project activity.

The constraint and cost function selection and propagation is the key aspect of this level of design. No solution is yet proposed, but with this step we have been able to simplify the design problem considerably.

4. Operation level design

This is the first step of the actual design activity where solutions are selected and propagated at lower levels of the hierarchy. The term *operation* denotes a set of basic building blocks expressed abstractly, that are used to realize the overall behavior of given functions. We begin by devising algorithms that solve the control problems posed by the specifications of each of the function. The control algorithms operate on variables that are *measured* on the physical domain and produce values of variables that *act* on the physical domain. Each control algorithm as well as measurements and actuation can be expressed in terms of *elementary* operations. The operation design is concluded when the system behavior and the performance estimation of the overall system described in terms of operations, satisfy the imposed requirements. Then the detailed component design starts (figure 1).

The first common characteristic of operations is that they may be "reused" in order to realize the behavior of several functions, i.e. a given operation may produce a result usable by two or more functions. Thus to select the set of operations to implement we must pay attention to their *granularity* so that they are complex enough to yield sizable savings in design time and yet simple enough to be usable in several functions. The second characteristic of the operations (or better of this specific level of the methodology flow), is that they are the main objects which will be manipulated in the evaluation phase of different architectures. Realizing an operation fully or partially in software or hardware or mechanically has different impact on the performance of a given implementation as well as on the overall cost.

As argued before, the most important set of operations are related to the control aspects (control algorithm, measurements and actuations) but there are other aspects that have to be taken into account. For example, diagnostic testing of the engine and of other car components is essential due to an effort to standardize safety and reliability aspects. In this case, diagnosis requires the development of appropriate algorithms, measurements and actuations that make it similar to control from a design point of view. Finally, transfer functions of physical components of the car such as engines and pipes carrying physical quantities from location to location have to be considered. In summary, we classify the set of operations into these categories:

1. *Measurement*. These operations yield an interpreted measurement or an estimate of a physical quantity.
2. *Control*. These operations embody the realization of control algorithms.
3. *Command*. These operations translate a logical command into the necessary set of *actuations*, needed to act on the plant.
4. *Transport*. These operations correspond to physical processes, which realize the transfer of a given entity or quantity between two points of the plant. E.g. the transport of a fluid in a pipe.
5. *Storage*. These operations express the accumulation of a given quantity using a physical process or device.

6. *Transform.* These operations correspond to physical/chemical processes, which produce a new physical quantity starting from a set of given physical quantities. E.g. chemical processes involved in the catalytic converter.
7. *Diagnosis*. These operations perform a supervisor task on the system and may signal an abnormal condition.

These categories of operations encompass both software and hardware as well as physical/chemical processes and/or devices. This is very important because in the architecture selection and mapping phase (see Section 5) a given operation, which realizes part of a function, may be reclassified across different categories, to achieve an improved performance.

During the functions analysis phase a number of these basic operations is agreed upon. Each function is assigned a number of operations, which will be used to realize its overall goal. This constitutes the *de-coupled model* of each function. The assignment activity is called *synthesis* per operations of the function.

Each operation is defined in terms of its inputs, outputs and triggering conditions. Moreover each operation must comply with a given set of constraints on its behavior. These constraints are derived from the requirements imposed by each function, which uses the operation. Of course, this implies that the design flow must take into account a negotiation table between the engineering groups in charge of defining the functions specifications and the engineering groups in charge of defining the operations specifications, in order to take into account conflicting goals. The activity aimed at the sharing of operations among different functions is called *integration*. The overall aim is to produce an arrangement, which will achieve the best overall performance.

5. Architectural Level

The design process produces a mapping between the behavior that the system must realize (operations) and the chosen system architecture, i.e. an interconnection of mechanical and electrical components, e.g., sensors, actuators, micro-processors and ASICs. The set of components are either available in a library of existing parts or to be designed anew.

This architecture and component selection task is the subject of intense research in the system design community. The most difficult problem to solve is how to rank different alternatives in terms of performance, cost, power consumption, reliability, and fault tolerance. In our case, the problem is further complicated by the mix of mechanical and electrical components to be considered. Our methodology is based on the work reported in [1] (see figure 1) for pure electronic systems.

To rank different system architectures, the cost and performance of the mapped behavior must be estimated. The system cost is given by the sum of the component costs. Analysis and simulation are used to obtain an estimate of system performance. As shown in the figure 1, a non-detailed design for the mechanical domain is used to have a first rough figure for the cost-performance tradeoff. For the electrical components, the behavioral description is mapped to the electronic system architecture whose performance is then estimated. The interface components, mainly sensors and actuators, have cost and performance effects on both mechanical and electrical domains and must be taken into account. The behavior decomposition in terms of operations is partially related to the chosen architecture since the partitioning between mechanical and electronic components could require the design of new operations. For example, a measurement operation could be mapped directly to a sensor that measures the desired physical

variable or it could be estimated from other measurements thus implying a mapping of this operation to the electronic component. This design decision is based on the trade-off between performance and cost, where performance is the precision in value and time of the observation, while cost is sensor plus electronic driver costs for the first solution or computational cost for the second. Hence, only when the complete set of operations is mapped to the two main domains, the algorithmic development is definitely concluded.

As example of mapping and component selection, we consider the set of measurements related to the crankshaft position: crankshaft angle over the complete engine cycle (0-720°), revolution speed (RPM) and TDC (Top Dead Center) detection for each cylinder. Based on the precision in value and time imposed by higher levels of abstraction, these operations can be mapped to different components. Let us suppose that a sub-system maker produces a single mechanical-electrical sensor that measures the angle position over 720°, the TDC condition and satisfies the precision and performance requirements. The speed measurement is computed indirectly from this sensor. This is a first possible solution for the mapping and only the interface components between the sensor and the other electrical components (hardware and software) and the revolution speed algorithm need to be developed. A second architecture solution is a magnetic encoder, with 60 'teeth', measuring the crankshaft angle for half of the engine cycle [0,360°] and a phase sensor measuring if the reference cylinder is in the first or second part of the cycle. To reach the desired precision, a second order interpolation is needed, while to achieve the required fault tolerance, redundancy measurements must be considered. The phase sensor should measure each phase of the engine (an event every 180° of crankshaft rotation for a four cylinder engine), and the encoder should measure the zero reference position (identified by two teeth missing). The other measurements are computed from these sensors and so mapped to the electronic components yielding parts that are implemented in hardware and parts in software. In this second solution, the original set of operations has been detailed at a lower level of abstraction; the interpolation, fault tolerance algorithms and the other indirect measurements must be developed before the detailed design could start. To rank these two solutions a cost and performance estimation early in the design phase is mandatory.

For the electronic part of the design (see [1]), the estimation techniques were based on approximate timing models of the programmable units. From the high level description of the algorithms to be implemented, the code is automatically generated in optimal form and the performance of the selected architecture is evaluated.

This is a rather radical change with respect to the software development practice in the automotive industry where the software is not specified at the algorithmic level but rather at a much lower level of abstraction that is very close to the architecture selected for implementation. We have seen control software specifications expressed in terms of look-up tables, since look-up tables are the most efficient way of carrying out complex arithmetic operations with micro-controllers that have integer ALUs. Hence, contrary to top-down design principles, specifications are strictly related to implementations.

The de-coupled model of each function (i.e. its description at the operation level) must be specified in an implementation neutral way. This is essential in order to be able to perform a correct analysis for the selection of the best architecture to use. The software to be run on these architectures can be sub-divided in three parts:

- software related to reading sensors data and controlling actuators (interface software);

- software related to the implementation of the control algorithms;
- software related to the communication of the ECU sub-system to the rest of the car system.

Each part has characteristics that can be matched by different computing architectures. For example, control algorithms involve arithmetic operations, matrix and vector manipulations, which are characterized by streams of computation occasionally interrupted by exceptions. Hence, the computing architecture that has the best cost/performance trade-off is today a DSP. On the other hand, communication are best handled by micro-controllers as well as any exception handling. Finally, I/O processing requires interrupt handlings that are best performed by simple micro-controllers. We are presently evaluating a number of alternative architectures to select the implementation platform for the next generation ECUs (planned for the commercial market in 2001). The architectures under consideration include a multi-processor configuration with a 32-bit microprocessor, an 8-bit I/O microprocessor and a DSP, a configuration with the DSP replaced by a FPU unit, and finally a configuration based on a single 32-bit microprocessor. To be effective, the methodology needs an extensive set of models at different level of abstraction. These models can only be developed in collaboration with the companies that design and market these units.

Mapping algorithms to the architecture consists either in the generation of code that will run on the programmable components of the architecture (micro-processors and/or DSPs), or in the generation of a gate net-list to be mapped into an ASIC or an FPGA, or both. These generation phase can be done either automatically or by hand, and is based on complex optimization strategies. For each architecture under consideration, we have a set of algorithms in place to implement effectively the high level specification on the components of the architecture. The mapping process consists itself of several sub-steps. For example, if the selected architecture supports only fixed-point computations, then the control algorithms that are typically formulated using a floating-point notation, have to be mapped into a fixed-point representation. This operation is usually done by hand in a trial and error fashion and it is very time consuming. Designer experience and knowledge is today fundamental to obtain reasonable results. We have a research project in place that attempts at deriving a rigorous procedure to carry out this mapping step.

6. Component Level

Once high-level restructuring operations like the one described above are completed and the architecture of the implementation has been selected, we need to

1. represent the resulting description of the operations in terms of the basic operations (instructions) of the programmable components of the architecture (software design).
2. design the components of the architecture that are not available in the library.

"Hardware" component design follows standard design practices and for this reason it will not be addressed here. The software generation step is analogous to the compilation process in standard computing. However, in the embedded systems domain cost considerations are so important, that inefficiencies in terms of memory and CPU occupation are hardly tolerated. Hence, embedded software designers often resort to the use of low level programming techniques that may involve assembly language programming with the obvious negative effects on the long-term maintainability and traceability of the code. We have been able to devise software synthesis techniques that can produce object code that is competitive with hand designs

[1]. We believe that software synthesis is essential to improve software quality by a substantial amount.

Software has an architecture as well. Software architecture consists of the organization of the software in components that favor performance, re-use and debugging. In particular, the decomposition of software into application software and basic software is of great importance. Basic software is related to device drivers, operating systems. Application software is related to the implementation of the control algorithm. This decomposition allows the developer of application software to write his code independently of the peculiarities of the I/O devices used. Unfortunately, most of the code in use today is not organized in this way, so that device drivers and algorithm implementation are tightly intertwined so that it is very difficult to migrate from an architecture to another. In addition, the software should be transparent to the actual physical partition of I/O and processing. Today, sensors and actuators are distributed in the car since they are mostly in proximity with the mechanical components of the system, while the digital processing functions are centralized. The decision whether to decentralize the digital processing functions or to integrate the I/O functions should be completely transparent to the application software developer. Unfortunately, this is not the case today. A way of obviating to the drawbacks of the present software architecture is to use a standard operating system[1]. However, there are inefficiencies related to this choice that may make this avenue undesirable. We have developed algorithms and methods to automatically synthesize specialized scheduling algorithms that are highly optimized for the particular problem to be solved.

7. Example of functions and operations design: the Cut-off Problem.

The design problem for engine control is specified by the FSM shown in Figure 2. To exemplify our design methodology, we focus on a particular sub-problem, the so-called *cut-off control* problem[2], associated to the state of Fast negative force transient. The proposed methodology has been tested starting from the formal definition of system specifications to algorithm implementation on the ECU of a commercially available car.

7.1 System Specification

When the driver releases the gas-pedal, the commonly accepted interpretation of her/his action is that the engine should not provide any force to the driveline. In traditional "no drive-by-wire" systems, the throttle goes to its minimum opening position yielding a manifold pressure decrease which results in an upper bound for available engine torque. This limited engine torque may not be large enough to counteract the elastic behavior of the driveline, thus yielding unpleasant acceleration oscillation. The objective function for the Fast negative force transient region of operation is thus given in terms of comfort requirements, i.e. minimize the oscillation in acceleration: Let δ, p, p_0, u and $\tilde{a}$ respectively denote the manifold pressure, the manifold pressure in the idle regime, the engine torque, and the oscillating component of the acceleration. Let also $\tilde{a}_{th}$ denote a threshold of oscillation perception. Then the desired behavior in the Fast negative force transient is specified as follows:

---

1. We point out the activities of a European automotive consortium, OSEK/VDX, that has as goal the issue of a set of compliance rules that operating systems in the car electronic sub-system domain have to satisfy
2. Very recently we have attacked and solved the torque transient control problem related to the changes in torque required by the driver when the gas-pedal position is varied [3].

$$\min \sup_{0\le t\le T} |\tilde{a}(t)| \qquad (7.1)$$

$$\text{subject to: } \begin{cases} u(0) = U_0 \\ u(t) = 0 & \forall t \ge T \\ |\tilde{a}(t)| \le \tilde{a}_{th} & \forall t \ge T \\ p(t) = p_0 & \forall t \ge 0 \end{cases}$$

7.2 Function Level

The objective function expressed in (7.1) is based on the output of the **Motion generation** function that represents the transfer function between the torque generated by the combustion process and the motion of the car. The elastic behavior of the driveline, causing the oscillating acceleration, is modeled in the continuous-time dynamics of the driveline by a pair of conjugate complex poles.

$$\begin{cases} \dot{x}(t) = Ax(t) + bu(t) \\ \tilde{a}(t) = cx(t) \end{cases} \qquad (7.2)$$

with input torque $u(t)$ bounded to belong to the interval $[0, U_M]$, where U_M is the maximum torque achievable with manifold pressure at p_0. The target manifold is a sphere B_ρ in the x reduced state space, where under u=0 the acceleration stays under $\tilde{a}_{th}$. A torque signal minimizing the functional of problem (7.1) for trajectories of system (7.2) is obtained by standard technique [2]. Such optimal control is a bang-bang signal if the initial state is far enough from B_ρ. To produce the bang-bang control, the Motion generation function requires the Combustion function to generate either torque *0* or torque U_M.

The **Combustion** function model is typically a static model with the air-fuel mix parameters as inputs and the engine torque and exhaust gas as outputs. The requested torque appears to be achievable by the **Combustion** function by modulating the injection signal between zero (no fuel injection) and stoichiometric injection. At the function level, the models needed to describe the interaction between the **Motion generation** and **Combustion** function can indeed avoid details about cycle synchronization. Next, the requested mix has to be generated as outputs of the **Mix** function, which in turn, requires the appropriate outputs of the **Air** function and the **Ignition** function. The discussion about how to tune the requirements so that each function has a reasonable chance of success is not extended to these cases for lack of space.

7.3 Operation Level

Once the specifications for each function are generated, we have to develop appropriate control algorithms to achieve them. More accurate models that can capture the synchronization aspects are needed to synthesize the control law. To do so, we introduced a hybrid model of the plant (engine + drive-line) [2].

The plant to be controlled is the combination of the engine and of the driveline. The behavior of the whole plant is very difficult to capture in all its details because of the inherent complexity of the chemical and mechanical processes involved. For a specific control purpose, the behavior of each process can be abstracted at the appropriate level, showing only its essential features. For example, the model of the injection process can be abstracted as a simple gain if we are dealing with torque control. Otherwise, if we were designing the control strategy for the injectors, we would have to take into account the dynamics of the injector valve: a quite complex

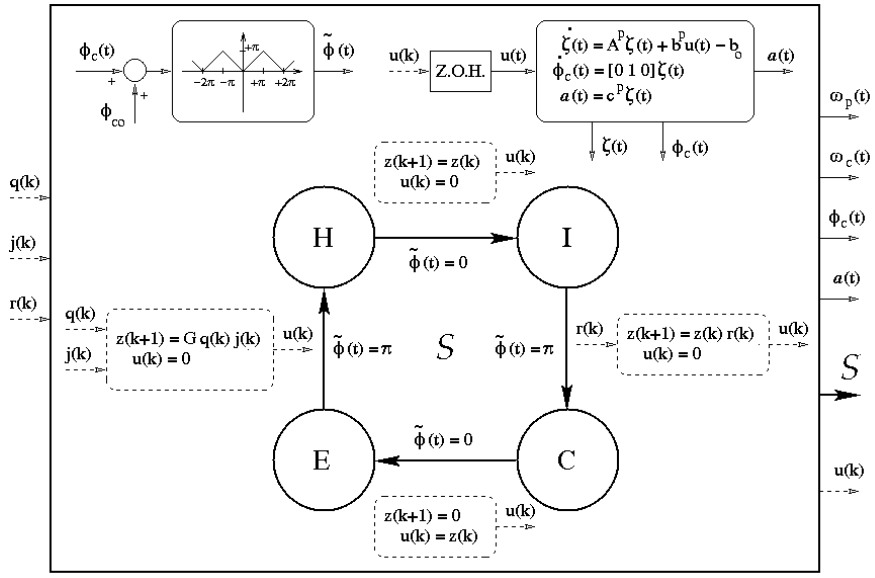

Figure 4: Hybrid model for a single cylinder

endeavor. For the control problem considered, the plant is represented by a hybrid system (see Figure 4) whose components are as follows:

Motion generation plant operation model

1. a Continuous Time system (CT) modeling the power-train;

Combustion plant operation model

2. a Finite State Machine describing the 4-stroke engine cycle;
3. a Discrete Event system (DE) modeling torque generation.

Motion generation plant operation model. The power-train model is described by a continuous-time linear system developed, identified and validated at Magneti Marelli Engine Control Division (continuous box in Figure 4). State $\zeta = [\alpha_e, \omega_e, \omega_p]^T$ represents the axle torsion angle, the crankshaft revolution speed, the wheel revolution speed, and ϕ_c represents the crankshaft angle. The input signal *u* is the torque acting on the crank. Vehicle acceleration *a* is the output. The linearized power-train dynamics is asymptotically stable with a real dominant pole λ_1, and a pair of conjugate complex poles $\lambda \pm j\,\mu$, responsible for the oscillating behavior.

Combustion plant operation model. The behavior of each cylinder in a 4-stroke gasoline engine is represented by the FSM shown in Figure 4. State *S* assumes values *I*,*C*,*E*,*H*, which respectively correspond to the *intake*, *compression*, *expansion* and *exhaust* strokes. In each state, the behavior of the system is subject to different dynamics depending on the evolving phenomena. The transitions of the FSM occur when a piston reaches the bottom or top dead center. The guard condition enabling the transition is expressed in terms of the piston position, given in terms of the crankshaft angle $\tilde{\phi}$.

In the model reported in Figure 4, *q* represents the quantity of air entering the cylinders during the intake stroke. The fuel quantity is taken proportional to air loaded. Control variable *j* $\in \{0,1\}$ indicates whether or not the fuel is presented in the mix. The time for spark ignition, usually referred to as *spark advance*, has the effect of a modulation $r \in [r_{min},1]$ of the maximum value of torque that can be generated. The generated torque *u* is a complex function of time during the expansion phase; in practice it is replaced with a constant equal to its average value.

The process of torque generation is characterized by the delay between the times at which combustion inputs q,j and r are set and the time at which such decisions have an effect. Such transport process on control signals is represented by a DE system (reported with dashed boxes in Figure 4) which is active at every FSM transition. The DE system output is the torque $u(k)$. At the FSM transition $E \rightarrow H$ the DE system reads its inputs $q(k)$ and $j(k)$, and stores in its state $z \in \mathrm{R}$ the maximum amount of torque achievable during the next E phase, obtained by the mix-to-torque gain G. Such value is corrected at the $I \rightarrow C$ transition by the modulation factor $r(k)$ due to the chosen spark advance. The DE output $u(k)$ is always zero except at the $C \rightarrow E$ transition when it is set to the value stored in z. Between two transitions of the FSM, occurring at times t_k and t_{k+1}, the input signal $u(t)$ to the powertrain model is given by $u(t) = u(k)$ for $t \in [t_k, t_{k+1})$.

Control algorithm synthesis (operation level). The intrinsic hybrid nature of the model at the operation level comes from the observation that the timing of the DE torque generation mechanism is determined by the angle of the crankshaft, which is a state component of the CT power-train dynamics whose evolution depends on the generated torque.

The hybrid model $M_{4\mathrm{cyl}}$ for a 4-cylinder engine is obtained from the hybrid model reported in Figure 4 by the composition of 4 FSMs and of *4* DE systems representing the behavior of each cylinder, and the ower-train CT dynamics. Model $M_{4\mathrm{cyl}}$ has input signals $\boldsymbol{j} = [j_1, j_2, j_3, j_4]^T$ and $\boldsymbol{r} = [r_1, r_2, r_3, r_4]^T$ properly synchronized with the corresponding DE models. Signal q is shared among the cylinders. Denote by $J_{4\mathrm{cyl}}$ and $R_{4\mathrm{cyl}}$ the classes of functions $\mathrm{N} \rightarrow \{0,1\}^4$ and $\mathrm{N} \rightarrow [r_{min}, 1]^4$, feasible for $\boldsymbol{j}$ and $\boldsymbol{r}$. The design at the operation level has been developed solving the following optimal control problem:

$$\min_{\substack{j \in J_{4\,cyl} \\ r \in R_{4\,cyl}}} \quad \sup_{0 \le t \le T} \quad |\tilde{a}(t)|$$

$$\text{subject to :} \begin{cases} \text{dynamics of hybrid model } M_{4\,cyl} \\ q(k) = q_0 \,\forall\, k \ge 0 \\ \text{FSM intial state } S_1 = H, S_2 = I, S_3 = C, S_4 = E \\ z(0) = G q_0 \\ \zeta(0) \notin C_\rho, \phi_c(0) = 0 \\ \zeta(T) \in \partial C_\rho \end{cases}$$

where C_ρ is the image of B_ρ in the ζ space. Because of the complexity of this problem, we actually found a "nearly" optimal solution by solving a relaxed problem and then mapping its solution to the original problem. An interesting aspect of our method is that we can find a rigorous bound on the performance degradation of the approximate solution with respect to the optimal one.

7.4 Architecture and Component Design

The architecture selection was, in this case, straightforward, since we decided not to change the original architecture selected by Magneti Marelli: a 68000 class 32-bit Motorola Althair ECU. The component design phase consisted of writing the software implementing the algorithm. Here care had to be exercised since the control algorithm was derived using floating point arithmetic while the available architecture had only fixed point arithmetic. In this phase, a first implementation had problems because of the conversion from floating point to fixed point was not done appropriately, resulting in variables saturation. A re-scaling fixed the problem. The

final results were exciting: cut-off control with the proposed algorithm had much better performance, in terms of oscillations and of time taken to reach cut-off, when compared to a heuristic algorithm in use at the time this research started. These results were not obtained at the expense of computing resources, always scarce when dealing with real time embedded controllers! Indeed, the implementation of the algorithm mounted on a VW Polo, showed a data size reduction of 50% and a code size reduction of 25% when compared with the heuristic algorithm. The CPU utilization was limited to 1% of the total capacity

8. Conclusion

We believe that a radical change in the way designs are carried out, is needed to solve the more and more challenging problems posed by the automotive industry. In particular, we have presented a new design methodology for powertrain control systems that is heavily based on abstraction and decomposition. Abstraction is a key aspect in complexity reduction and in tighter quality control. We have identified five levels of abstractions and given examples on how to use effectively these levels of abstraction. We have presented problems related to the implementation of control algorithms in real products and what are the intellectual challenges that have to be faced in this process. The implementation techniques are heavily based on an electronics system design methodology developed by part of our team and that has been exported to the Alta Group of Cadence. We expect to demonstrate the entire design process with a complete commercial product in the automotive domain based on this approach in the next two years.

Acknowledgements

We wish to acknowledge the support of Drs. Pecchini, Romeo, Mortara, Gaviani and Damiano of Magneti Marelli and of Jim Rowson, Patrick Scaglia and Shane Robison of Cadence who in their top management role allowed us to access important information and to influence the strategic directions of their companies. This research has been funded in part by CNR.

References

1. F. Balarin, M. Chiodo, A. Jurecska, H. Hsieh, A. L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich and B. Tabbara, eds., *Hardware-Software Co-Design of Embedded Systems: The polis Approach*, Kluwer Academic Press, 1997.
2. A. Balluchi, M. D. Di Benedetto, C. Pinello, C. Rossi and A. Sangiovanni-Vincentelli, *Cut-off in engine control: a hybrid system approach*, in 36th CDC, San Diego, CA, 1997.
3. _____, *Hybrid framework for torque regulation problem in automotive engine control*, submitted to the 37th CDC, 1998.
4. _____, *Hybrid control for automotive engine management: the cut-off case*, in Workshop on Hybrid Systems, University of California at Berkeley, Berkeley, CA, 1998.
5. A. Sangiovanni-Vincentelli, *Embedded system design and hybrid systems*, in Control using logic-based switching, A. S. Morse ed., vol. 222 of Lecture notes in Control and Information Sciences, Springer-Verlag, London, U.K., pp. 17-38, 1997.
6. A. Balluchi **,** L. Benvenuti, M. Di Benedetto, A. Ferrari, C. Pinello, A. Sangiovanni Vincentelli, *The Design of Embedded Controllers for Automotive Engine Management: the Cut-off Case,* in 37th CDC, 1998
7. T.Cuatto, C. Passerone, L.Lavagno, A. Jurecska, A. Damiano, C. Sansoè, A. Sangiovanni Vincentelli, *A case Study in Embedded System Design: an Engine Control Unit*. In 35th DAC, San Francisco, CA, 1998