EPspectra: A Formal Approach to Developing DSP Software Applications

Hahnsang Kim, Thierry Turletti, Amar Bouali

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

EPspectra: A Formal Approach to Developing DSP Software Applications

Hahnsang Kim — Thierry Turletti — Amar Bouali

N° 4293

Octobre 2001

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4293--FR+ENG

INRIA
SOPHIA ANTIPOLIS

EPspectra: A Formal Approach to Developing DSP Software Applications

Hahnsang Kim, Thierry Turletti, Amar Bouali

Thème 1 — Réseaux et systèmes
Projet Planète

Rapport de recherche n° 4293 — Octobre 2001 — 19 pages

Abstract: Software approach to developing Digital Signal Processing (DSP) applications brings some interesting features, such as flexibility, re-usability of resources and easy upgrades of applications. However, it requires long tests and verification phases due to the increasing complexity of software. In this report, we present the EPSPECTRA toolkit, an ESTEREL-based development verification environment targeted to develop DSP applications on general purpose computers. We illustrate the description with an example of DSP implementation: the radio interface part of a GSM base station. Such DSP applications have complex control-paths since they have to obey strict scheduling rules.

The purpose of the ESTEREL formal method is to ease the programming of complex control-paths and the verification of safety properties of these control-paths. In this report, we show how we verify several critical safety properties of the radio interface of our GSM using the ESTEREL verification environment. Since the model used for the executable code generation and for the formal verification is the same, the EPSPECTRA toolkit achieves the faithful verification of the targeted application. By using EPSPECTRA, we obtain a higher confidence in the final implementation since traditional verification techniques based on simulation and testing can only partially verify the program. Moreover, the EPSPECTRA approach has shown the drastic reduction of verification/testing time while requiring relatively few expertise in formal language verification methods.

Key-words: synchronous language, ESTEREL, formal methods, formal verification, software safety,program correctness, DSP applications, GSM base station

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

EPspectra: Une Approche Formelle à Développer des Applications logicielle de DSP

Résumé : Utiliser une approche logicielle pour développer des applications de traitement du signal apporte de nombreux avantages, comme la possibilité de développer des algorithmes plus flexibles, et de les mettre à jour plus facilement. Cependant, la mise en œuvre de ces logiciels nécessite de longues phases de vérification du code et de tests en raison de la complexité accrue du logiciel. Dans ce rapport, nous présentons brièvement la toolkit EPSPECTRA, un environnement logiciel basé sur le langage formel ESTEREL pour développer des applications DSPs sur des stations PC classiques. Nous illustrons cette description avec comme exemple d'application, la partie INTERFACE RADIO d'une station de base GSM. Ce type d'application doit obéir à des règles d'ordonnancement strictes.

L'intérêt de l'utilisation du langage formel ESTEREL est de faciliter la mise en œuvre et la vérification d'algorithmes de contrôle complexes. Dans ce rapport, nous montrons comment vèrifier un ensemble de propriétés de fiabilité qui sont critiques pour l'application GSM. Les techniques de vérification traditionnelles basées sur la simulation du code et le test ne permettent qu'une vérification partielle du code. L'environnement de développement EPSPECTRA permet une vérification logicielle plus complète parce qu'il utilise le même modèle formel pour générer le code exécutable et pour vérifier les propriétés de l'application. Notre approche permet ainsi de réduire les temps de vérification et de test de développement du logiciel tout en ne nécessitant que relativement peu d'expertise en méthodes de vérification formelles.

Mots-clés : langage synchrone, ESTEREL, méthodes formelles, vérification formelle, Applications DSP, Station de base de GSM.

1 Introduction

It is well known that the number of transistors per integrated circuit roughly doubles every 18 months according to Moore's law[1]. This law has been maintained and still holds true by market factors. Consequently, programming environments for DSP applications are no more required to rely on specialised DSP hardware as the performance of general-purpose processor and peripheral equipment increase along with the high-tech curves. It naturally derives the shift of hardware-operation functions into software. A software approach to developing DSP applications allows the following advantages: re-usability of existing hardware, ease of upgrades, more flexible applications, etc. On the other hand, it makes the implementation of software applications more complex owing to requiring multi-disciplinary knowledge, such as software architecture, signal processing, real-time scheduling, networking protocols, validation, etc. Furthermore, it requires an appropriate development environment accessible to programmers.

Recently, we have developed an extension of the PSPECTRA development environment which has been developed by the SpectrumWare project[2] at MIT. PSPECTRA [Bos99, Vas00] was designed to implement software radio applications on general-purpose workstations. The ESTEREL extension of PSPECTRA, EPSPECTRA [HT01] is a component-based development verification environment which integrates debugging and verification phases within a component-based architecture. It makes it possible to facilitate the development of DSP software applications and verify their correctness.

In this report, we present EPSPECTRA, a development verification environment that allows the implementation of DSP software applications and the verification of their safety properties. As an example of EPSPECTRA applications, we describe a GSM Base Transceiver Station (BTS) [MM93]. In particular, we focus on the radio interface part of GSM. On top of that, we describe safety properties of the implementation in accordance with the concept of temporal logic for the verification phases of EPSPECTRA, and discuss the results obtained from the verification of safety properties.

The formal language ESTEREL [Ber99] is a synchronous programming language dedicated to reactive systems. ESTEREL programs perform an input-driven computation: wait for inputs and compute corresponding outputs in a cyclic manner, referred to as a reaction. ESTEREL provides powerful constructs: sequencing, parallelism, preemption, etc., which are suited for programming software and hardware controllers where the control-handling parts are predominant. We present the ESTEREL methodology which provides specification, simulation, automatic code generation, and verification.

Temporal logic [ZA92] supports the specification of temporal behaviors and properties required for reactive systems. It allows one for the formalisation with respect to the specification of properties required for the proof of the correctness. In this report, we show the ESTEREL translations of safety properties that can be expressed in temporal logic.

[1] See http://www.intel.com/research/silicon/mooreslaw.htm
[2] See http://www.sds.lcs.mit.edu/SpectrumWare/

Most of traditional verification methods are concerned with proving properties only of models of programs rather than programs themselves. These models, in the traditional verification techniques, are abstracted under certain limited conditions and written in the language dedicated to the proof system. Therefore, they may not faithfully represent the program that will be a final targeted program. On the other hand, the ESTEREL methodology allows one to directly verify the actual code of ESTEREL programs that corresponds to the actual targeted implementation. It guarantees that the ESTEREL programs satisfy the properties to be proved as far as all source code is correctly compiled to the targeted code. In [LCJ95], case studies have been done with a verification method with ESTEREL.

This report is structured as follows: section 2 gives an overview of our implementation environment, which includes the sequence of operations for the radio interface part of a GSM BTS and the EPSPECTRA software environment, in particular, focusing on its control part. In addition, we describe the verification properties that should be satisfied for the correctness of the GSM software application. Section 3 presents formal verification based on model-checking for Finite State Machine (FSM) model and Section 4 describes ESTEREL translations of the corresponding safety properties. Moreover, the combination of both the translation of properties and the implementation of programs is shown. Section 5 analyses the performance results of the verification process and discusses advantages and difficulties of this approach. Section 6, finally, concludes this report and introduces future work.

2 Overview of the Implementation Environment

This section briefly describes the radio interface implementation of a GSM BTS using EPSPECTRA. It accounts, not just for the connection of operations involved in the software application of the radio interface, but also for the control part of EPSPECTRA as well as verification properties of the implementation.

2.1 Sequence of Operations for a GSM BTS

The sequence of operations for the radio interface of a GSM BTS is shown in Figure 1. Basically, after having transformed speech into compressed data blocks, channel coding adds redundancy to the data blocks. The data blocks are interleaved and spread into pieces, which are combined with flags to build up the bursts. Ciphering is applied to these bursts and then the resulting data is used to modulate the carriers. The reverse transformations are performed on the receiver side.

- **Speech coding** algorithm, RPE/LTP (Regular Pulse Excitation with Long Term Prediction) [94e94] produces data blocks of 260 bits every 20ms.
- **Channel coding** introduces redundancy into the data flow, increasing its rate by adding information calculated from the source data, in order to allow the detection

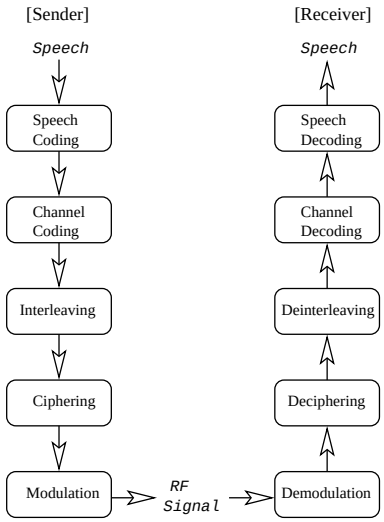

Figure 1: Sequence of operations from speech to radio waves and back to speech

or the correction of signal errors introduced during transmission. It forms a complete coded speech frame of 456 bits.

- **Interleaving** consists in mixing up the bits of several code words, which in the modulated signal are spread over several code words. GSM coding blocks are interleaved on 8 bursts each of which consists of 57 bits.
- **Ciphering** performs an XOR (exclusive or) operation between 2 bursts of each block and a secret recipe known only by the mobile station and BTS.
- **Modulation** transforms the binary signal into a GMSK (Gaussian Minimum Shift Keying) [gms81].
- Once radio waves are captured by the antenna, the portion of the received signal which is of interest to the receiver is determined by the multiple access rules. **Demodulation** takes place in this portion.
- **Deciphering** performs the same operations by reversing the ciphering algorithm.
- **Deinterleaving** merges two different 8-burst blocks into a 456-bit code word.
- **Channel decoding** involves reconstructing the source information from the output of the demodulator, using the added redundancy to detect or correct possible errors in the output from the demodulator.

- **Speech decoding** reconstructs the speech by passing the residual pulse first through the long-term prediction filter, and then through the short-term predictor.

2.2 The Software Environment

A software implementation of the above operations has already been described in [THD99]. Here, we describe the software environment used to re-design the radio interface of a GSM BTS using EPSPECTRA. Let us see Figure 2.

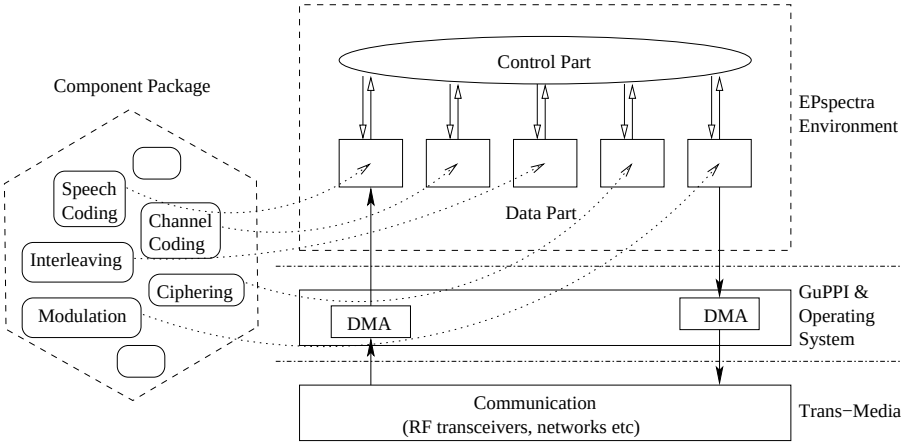

Figure 2: The Software Environment including EPspectra

Figure 2 is split into four parts: EPSPECTRA, the General Purpose PCI Interface (GuPPI) developed at MIT along with the operating system, the media for the transmission, and the components to be 'plugged & played' into EPSPECTRA. EPSPECTRA is composed of the ESTEREL-written control part including the scheduling of DSP software applications and the C/C++-written data part involved in computation-intensive functions represented as components. In subsection 2.3, we first details the main characteristics of the ESTEREL methodology. Then, the detailed description of the control part is provided in subsection 2.4. The data part consists of buffering and wrappers which encapsulate C++-written components to provide transparent access to them for the control part. Components are referred to as functions in which algorithms are performed. GuPPI allows the sampled signal data to be directly transferred in and out of memory of the workstation via Direct Memory Access (DMA).

2.3 The Esterel Methodology

ESTEREL belongs to the family of synchronous reactive languages, such as LUSTRE, SIGNAL, STATECHARTS, for the programming of reactive systems such as embedded systems,

software and hardware controllers, man-machine interfaces, etc., refer to [Hal93]. ESTEREL provides powerful constructs to express sequencing, parallel behavior, preemption. It also provides a communication mechanism by means of signal broadcasting. These constructs are particularly suited for the programming of control-dominated part of reactive systems. The ESTEREL language has a clean mathematical semantics that interprets an ESTEREL program as a Finite State Machine (FSM), a state-graph model with labels over the graph edges. The FSM model represents exhaustively all the possible states that the program can be in and all the behaviors that the program can perform between the states. The main features brought by the ESTEREL methodology are:

- **Specification**: Though the ESTEREL language is relatively simple, it is expressive and concise enough to program complex controllers.
- **Simulation**: The ESTEREL system provides symbolic debugging simulation by means of the symbolic debugging simulator XES [GE99]. The simulation environment is based on the FSM model and is fairly efficient. The simulator is coupled with the formal verification environment.
- **Automatic code generation**: The ESTEREL system compiles an ESTEREL program into an executable C program with a C interface that is easy to connect with hand-written C code. The C code implements exactly the FSM model.
- **Formal verification**: The FSM model allows one to perform model-checking to verify its properties. When a property is not satisfied, the verifier generates a counter-example input-sequence. This counter-example can be played back in XES. The more details of model-checking is given to Section 3.

Hence, the ESTEREL methodology is not only a programming language, but also a formal method, where there is no gap between specification, simulation, and execution. Using the ESTEREL methodology, the procedure verifying the properties of an ESTEREL program will be as follows:

i. describe the properties satisfying the correctness of an ESTEREL program, which is called ESTEREL observers,

ii. compile the ESTEREL program in parallel with the ESTEREL observers, and

iii. check for satisfaction or violation of the properties using the ESTEREL model-checker XEVE [Bou98].

2.4 The Control part of EPspectra

The control part of EPSPECTRA first initialises and connects component-driven processing modules and then triggers sources. Sources are specialised modules that have output connection ports with no input ones. Sinks are specialised modules that have input connection ports with no output ones. All applications implemented by means of EPSPECTRA must

have at least one or more sources and at least one or more sinks at the beginning and at the end of the sequence of operations, respectively.

After the initialisation phase, the processing loop including an *estimating method* and a *computing method* is performed. The estimating method stands for determining the amount of data corresponding to units at a time to be computed by processing modules such as speech coding, channel coding, interleaving, etc. In addition, it re-calculates the available amount being passed from a processing module to the following processing modules in reference with both the data unit and the remains of data consumed by the following processing modules. The computing method applies the computation algorithms to the available data by calling functions associated with the components and then dispatches it to the following modules. It also informs the previous modules of what amount of data has been processed for their estimating method.

2.5 Verification properties

In addition to the processing methods described above, the GSM programs have 3 requirements that should be satisfied. Basically, all processing modules do their behaviors in parallel not sequentially, that is, modules do not wait for terminating the whole process of other ones. Let us take a look at the following example.

```
module GSMsender
   ...
   run source/SOURCE
    ||
   run speechcoding/P_MOD
    ||
   run channelcoding/P_MOD
    ||
   run interleaving/P_MOD
    ||
   run ciphering/P_MOD
    ||
   run modulation/P_MOD
    ||
   run sink/SINK
   ...
end module
```

Apparently, all modules run in parallel, but unless carefully programming, the process of a module may block the process of the others from running simultaneously due to missing signals that should be received.

Let us see Figure 3. The performance of parallelism can be enhanced as the process of an inside module is sliced into two pieces (i.e. the estimating method and the computing

method) running in parallel according to the module's functionality. It requires the cautious synchronisation between the pieces of process of one module and the others.

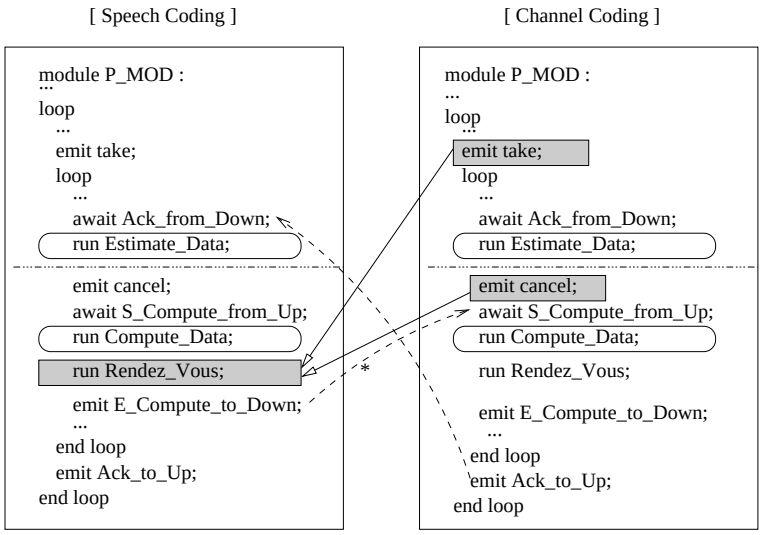

Figure 3: Signal passing diagram between two adjacent modules

The 'Ack_from_Down' signal of the speech coding module is synchronised with the 'Ack_to_Up' signal emitted by the channel coding module. As soon as 'Ack_from_Down' is received, the estimating method is performed on the speech coding module (i.e. run Estimate_Data). The 'S_Compute_-from_Up' signal of the channel coding module is synchronised with 'E_Compute_to_Down' signal emitted by the speech coding module. This synchronisation leads to perform the computing method on the channel coding module (i.e. run Compute_Data). However, Estimate_Data and Compute_Data modules contain loop statements including consuming *ticks*[3] and yet the number of consumed ticks will be determined at the run time execution. It may cause the channel coding module to miss the signal coming from the speech coding module corresponding to the $*$-given arrow in Figure 3. ESTEREL is a synchronous language but there is the case that the number of ticks being consumed is shaded in view of the programmers. Therefore, on purpose, we put a *explicitly* synchronising mechanism called *Rendez-Vous*. As it appears in Figure 3, the 'take' signal validates *Rendez-Vous* of the speech coding module and then *Rendez-Vous* suspends the successive process of the speech coding module, which leads to delay emitting the 'E_Compute_to_Down' signal of the speech coding module. Afterwards, on receiving the 'cancel' signal from the channel coding module, *Rendez-Vous* kills this suspending. It allows one to synchronise the $*$-marked arrow.

[3] *Tick* introduced in ESTEREL is thought of as logical time which represents the activation clock of reactive program.

The GSM programs process sample data; the data processed on the sources would be completed on the sinks in the end. All modules contain loop statements, which means that the programs may stall or may be in a situation in which some critical stage of a task is unable to finish. These must be verified for the safety of the programs.

Hereby, the requirements of above model that should be satisfied are as follows:

R1 The signal emitted by a module is always caught by the opposite modules (referred to as *Rendez-Vous*).

R2 The computed sample data on the sources will eventually be consumed on the sinks.

R3 When input signals on every module are received, the corresponding output signals are emitted within a *bounded* time.

3 Formal Verification

In this section, we explain what formal verification of ESTEREL programs means. As we said, ESTEREL is both a formal modeling language and a programming language. ESTEREL benefits a clear mathematical semantics that characterises a program as an FSM model. The ESTEREL FSM model is defined as a structure (I, O, S, s_0, T) where I is a set of input signals, O a set of output signals, S a set of states, s_0 the initial state, and T a transition relation: T is a set of 4-tuple (s, i, o, s') that means that there is a transition from s to s' whenever the input event i is true, generating the output event o. The FSM model is the one that is used for simulation, execution, and formal verification. A state of the FSM model is a stable configuration of the control points of the program. A transition from a system state is a reaction to some input event: the reaction leads to a new stable system. The FSM model has the advantage of exhaustively exhibiting the program behaviors.

3.1 Model-Checking

Formal verification is the activity of proving properties of programs and systems in a mathematical sense. When it is based on the FSM mathematical model, the activity is called model-checking.

Model-checking starts from a model of the program or system behavior and a set of properties expressed in a mathematical languages called a temporal logic. In addition to classical boolean operators, temporal logics offer specific operators to describe time and behaviors. Examples of such operators are $\Box$, $\Diamond$, and $\bigcirc$, respectively called *always*, *eventually*, and *next* operators. The semantics of the temporal logic is defined directly on the FSM model. We roughly sketch the semantics of the aforementioned operators:

- $\Box p$ means that p holds in every state of the FSM.
- $\Diamond p$ means that p holds in some future state of the FSM.

- $\bigcirc p$ means that p holds in the next state of the FSM.

Verification then consists in checking the validity of the temporal logic formulas over the FSM model.

Generally speaking, there are two types of properties that can be expressed: safety properties and liveness properties. Safety properties express the fact that *something bad will never happen*. Liveness properties express the fact that *something good will eventually happen*. In our experience, most of the properties are safety ones. When liveness is concerned, it is often reducible to *bounded* liveness, which is fundamentally a particular form of safety properties. Bounded liveness properties express the fact that *something good will eventually happen in at most k times units*. Let us look into a way to directly apply these properties to the ESTEREL system.

3.2 Observer Properties

In the ESTEREL system, a simple way to specify temporal properties without using a temporal logic syntax is offered. Indeed, the users directly express the properties using the ESTEREL language. Let us consider a simple property which requires the following condition: "At each state, if signal A and signal B are present, then signal C in the next state should be present unless signal R is present". In temporal logic, one would approximately write:

$$\Box((A \wedge B) \implies (\bigcirc C \vee \neg R))$$

In ESTEREL, this property is written as follows:

```
module OBSERVER:
input A, B, C, R;
output BUG;
loop
   present [A and B] then
      pause;
      abort
         present C else emit BUG end
      when R
   else
      pause
   end present
end loop
end module
```

The `pause` statement waits for one time unit. The `abort ... when cond` construct kills its body as soon as the condition `cond` is true. This formal verification consists in checking if the signal properties such as `BUG` above can be emitted in some reachable states and for some input events. If the property is violated, the XEVE model-checker generates an input sequence of events that has produced the error state.

4 Properties of the GSM Program

In this section, we detail the different requirements expressed in subsection 2.5 into safety properties. Each safety property then is translated into an ESTEREL observer. The safety properties and corresponding translations are as follows:

S1 Deadlock freedom: an important safety property is deadlock freedom. In the GSM programs, deadlock occurs when one misses signals that should be received, The *Rendez-Vous* mechanism aims to establish an explicit synchronisation between at least two signals of modules running in parallel. To guarantee that the program will never deadlock, it is sufficient to verify the *Rendez-Vous* mechanism, namely the following safety property: any state at which the module emits 'E_Compute_to_Down' is previously preceded by a state at which the opposite ones are ready to receive 'S_Compute_from_Up'. This is stated by the following ESTEREL observer:

```
module S1:
input ReadytoReceive, E_Compute_to_Down;
output S1_VIOLATED;
loop
   await E_Compute_to_Down;
   abort
      emit S1_VIOLATED
   when pre(ReadytoReceive);
end loop
end module
```

S2 Correctness: a major scheduling task of GSM programs is involved in correctly delivering certain sample data computed on the source to the sink by applying a sequence of operations to the corresponding data. The procedure begins from the source receiving 'Ack_from_Down' and ends upto the sink emitting 'Ack_to_Up'. We note that all sample data computed on the source is not always reached to the sink. Indeed, some amount of sample data can be skipped depending on the particular conditions, *e.g.*, a missed deadline happens, either a type of modulation algorithm or time variables applied to sample data are changed, or some events, such as the *reset* event, from the outside environment occur. Therefore, It is sufficient to check that this property is satisfied at the first time.

Each module consumes 1 or 2 ticks for an iteration of the loop statement from an input sample data to the corresponding output sample data. Each of the GSM sender and the GSM receiver consists of 7 modules including a source and a sink (refer to Figure 1). Suppose that each module consumes 2 ticks for a specific sample data, the sink finishes computing an amount of sample data derived from the source within 14 ticks ($= D$). However, in case that more than an iteration of the loop statement are done, D would be greater than 14. Therefore, D is specified as a heuristic number.

The correctness property is as follows: for a state receiving 'Ack_from_Down' on the source, a state emitting 'Ack_to_Up' on the sink is following within D position. This is stated by the following ESTEREL observer:

```
module S2:
constant D;
input Ack_to_Up, Ack_from_Down;
output S2_VIOLATED;
   await Ack_from_Down;
   abort
      await D tick;
      emit S2_VIOLATED
   when Ack_to_Up;
end module
```

S3 Safety-liveness: every module behaves as a sub-reactive program which consists of input signals and output signals and responds to input signals by setting or emitting a value of the corresponding output signal. Each module contains a loop statement with a certain condition to exit. There is, possibly, a situation in which some critical stage of a task is unable to finish, referred to as *livelock*. In case that one of modules has livelock, other modules would be blocked. The number of ticks consumed between receiving both input signals from the following module and ack-signals from the previous one and emitting the corresponding output signals in a module is proportioned to the length of a path between the module and the sink. Let us see Figure 4.

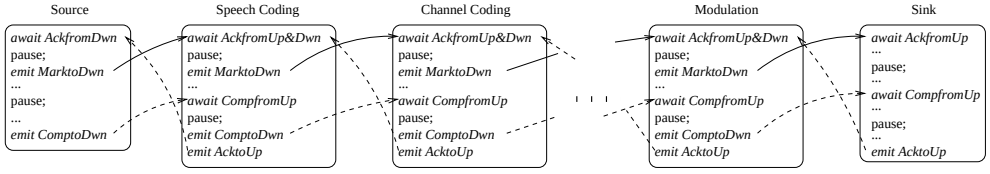

Figure 4: A signal-passing scenario of the GSM sender program

Figure 4 shows a signal-passing scenario of the GSM sender program. In terms of the speech coding module, it takes 12 'pauses' identified by 'await tick' from the step of *await AckfromUp&Dwn* to the step of *emit AcktoUp*, while it takes 10 'pauses' to the corresponding part in the speech coding module. Therefore, the longest path is the path between the source which receives input signals and waits for the corresponding ack-signals and the sink which sends the ack-signals through other modules to the source.

Considering that each module consumes 2 ticks in an iteration of a loop statement, the source receives ack-signals within D=14 ticks and yet D is also a heuristic number.

This property is separately applied to the source, the sink and the others. The general form of the property is as follows: if I_s holds at position j, then O_s holds at position k, for k, $j \leq k \leq j + D$. This is stated by the following ESTEREL observer:

```
module S3:
constant D;
input Is, Os;
output S3_VIOLATED;
loop
   await Is;
   abort
      await D tick;
      emit S3_VIOLATED
   when Os;
end loop
end module
```

This property has to be verified for the following (I_s, O_s) event predicate pairs:
(Ack_from_Down$_{source}$, E_Compute_to_Down$_{source}$),
(S_Mark_from_Up$_{mod}$ $\wedge$ Ack_from_Down$_{mod}$,
Ack_to_Up$_{mod}$), (S_Mark_from_Up$_{sink}$, Ack_to_Up$_{sink}$).

At the phase of combining the GSM programs with observers in the property verifying procedure, we define the following program including the `GSM` program in parallel with the `S1` `S2`, and `S3` observers.

```
module VERIFY_PROGRAM:
constant D:=14 : integer;
input <the program inputs>;
output <the program outputs>,
       S1_VIOLATED,
       S2_VIOLATED,
       S3_VIOLATED;

   run GSM
    ||
   run S1
    ||
   run S2
    ||
   run S3
end module
```

At the last phase of checking satisfaction of the properties, the occurrence of `S1_VIOLATED`, `S2_VIOLATED`, and `S3_VIOLATED` is checked using XEVE. We describe more details in the following section.

5 Performance of the Verification Process

XEVE takes as inputs the FSM model expressed as a set of boolean equations in BLIF format[4], as generated by the ESTEREL compiler. It makes use of the symbolic state space construction algorithm by means of Binary Decision Diagrams (BDDs) [Bry86] which are the internal representation of an FSM model for the reachable state space. XEVE provides two verification functions: minimising the number of states of the FSM model and checking the emission status of output signals. The former function is performed with respect to an equivalence notion called *symbolic bisimulation* [RA94], but, in this experiment, could not be applied to the sender and the receiver GSM ESTEREL programs due to the explosion of states space. The latter one checks two status for output signals: *possibly emitted* which means there exists a reachable configuration that the emitted output signals are derived from some inputs and *never emitted* which means there exists no reachable configuration that the emitted output signals are derived from some inputs.

We verified that all the properties that should be satisfied in the sender and the receiver GSM programs were kept safe by confirming *never emitted* the output signals including VIOLATED_DEADLOCK_FREEDOM, VIOLATED_CORRECTNESS, and VIOLATED_LIVENESS using XEVE. Figure 5 shows the verification result of checking the status of the output signals of the GSM sender program in XEVE.

Our performance analysis in the verification process has been carried out on a bi-PIII 935MHz machine with 500MB of core memory and 516MB of swap space. ESTEREL programs are compiled by the version 6.03 of the ESTEREL compiler[5] into BLIF formats and then optimised by REMLATCH [STB96] and SIS [E*et al*92].

Each produced reachable states space of the sender and the receiver GSM ESTEREL programs amounts to 25862 and 25863 states. The sizes of BDD of reachability are 15048 and 13186, respectively, to the sender and the receiver GSM programs. Each program takes about 34 second-CPU time to generate the reachable states space. Note that these amounts have been generated by the combination of the properties and the implementation of the ESTEREL programs.

Advantages: The software architecture of EPSPECTRA, whose features include a clear separation between the control part and the data part, and a component-based DSP library, makes easier the design and implementation of DSP applications. Moreover, it incorporates verification functionalities, which allow one to directly verify the actual code of ESTEREL

[4] *Berkeley Logical Interchange Format* is an ASCII format developed at the university of Berkeley to describe a logic-level hierarchical circuit in textual form.

[5] 6.0 or later version of ESTEREL compiler is now developed and maintained by ESTEREL-TECHNOLOGIES LTD.

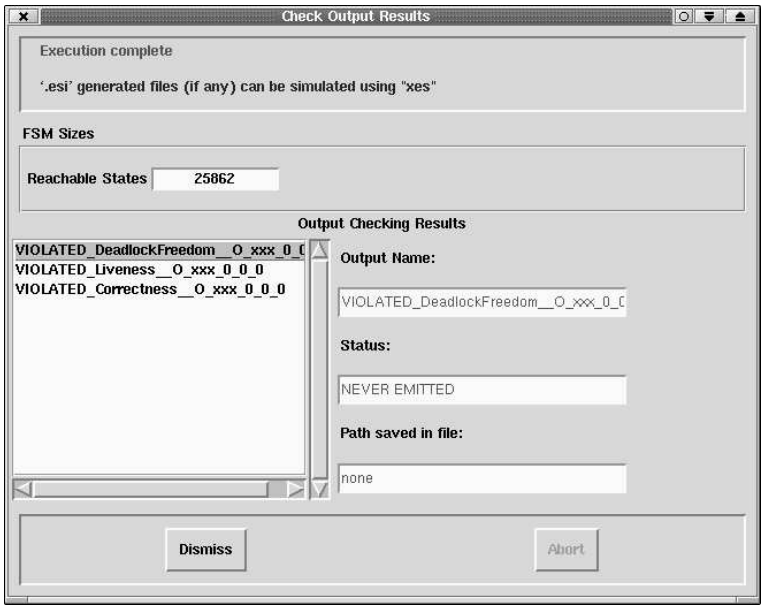

Figure 5: Screen-shot of a verification result of checking the status of output signals

programs that would straightforwardly be compiled into an executable code. It guarantees that the ESTEREL-implemented programs satisfy the safety properties to be proved as far as all source code is correctly proved and compiled to the targeted code.

Difficulties: The ESTEREL model-checker XEVE allows one to verify complex control-paths of DSP applications, but when it comes to real-time constraints, it has difficulty verifying time-sensitive scheduling behaviors of DSP applications. In addition, in spite of the advantage of exhaustively searching all states space, it leads to the explosion of states space that has been encountered in this experiment. It is required to enhance a verification function minimising the number of states, provided by XEVE.

6 Conclusion

In this report, we have described EPSPECTRA, a component-based development verification environment that makes easier the implementation of DSP software applications and allows the verification method to guarantee their safety. As we have seen, EPSPECTRA is partitioned into a computation-intensive data part and a control part. This structure allows programmers to easily develop novel software applications by means of the reuse of components or the minimal description of DSP algorithms for both the data part and the control

part. We have presented that the GSM programs implemented using EPSPECTRA satisfy their safety properties making the most of the ESTEREL methodology which brings a formal method: the actual code to specify, simulate, and verify and the targeted implementation code are identical.

In future work, we shall experiment this test-generation feature. The ESTEREL model-checker XEVE provides an automatic test-cases generation feature that can further reduce the time cost of the testing phase [LA*et al*99]: the generated test-cases are such that the ESTEREL FSM model's states are totally covered, that is, every state of the model is visited and stimulated at least once by the test cases. With these test cases, the developers can detect more potential tricky bugs called the corner cases for which the writing of a test case to reach them is particularly hard.

We also shall attempt to verify timing constraints considering that the applications developed by EPSPECTRA correspond to time-sensitive systems based on either hard real-time constraints or soft real-time constraints. The method introduced in [D*et al*01] will be available, which verifies quantitative timing constraints by being represented as a time-driven automata. In terms of EPSPECTRA itself, the variety of scheduling techniques that would be provided as a library are required in the control part and so different actions are possible when a deadline is missed. For example, in addition to aborting computation and dropping the remaining data in the current version, one could replace the remaining data by a special value or partially estimated data from the result, or start processing the next slice of data while the current slice is processing in parallel.

Acknowledgements

We gratefully acknowledge the discussions about ESTEREL Verification Techniques with Robert de Simone at INRIA-Sophia Antipolis and about Timing Constraints Verification Techniques with Daniel Weil and Jacques Pulou at France Telecom R&D. This research was supported by the DESS project associated with Information Technology for European Advancement (ITEA).

References

[94e94] European digital cellular telecommunication system, 1994.

[Ber99] Gérard Berry. The constructive semantics of pure esterel, 1999.

[Bos99] Vanu G. Bose. *Design and Implementation of Software Radios Using a General Purpose Processor.* PhD thesis, MIT, June 1999.

[Bou98] Amar Bouali. Xeve: an esterel verification environment. In *the 10th International Conference on Computer Aided Verification*, volume 1427. LNCS, 1998.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[D*et al*01] D. Weil, E. Closse and *et al.* Taxys: a tool for developing and verifying real-time properties of embedded systems. In *the 13th International Conference on Computer Aided Verification*, 2001.

[E*et al*92] E. M. Sentovich, K. J. Singh and *et al.* SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, Univ. of California Berkeley, 1992.

[GE99] Gérard Berry and Esterel team. *The Esterel v5.92 System Manual.* http://www.esterel.org, 1999.

[gms81] Gmsk modulation for digital radio telephony. *IEEE trans. on Communications*, com-29(7):1044–1050, 1981.

[Hal93] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, 1993.

[HT01] Hahnsang Kim and Thierry Turletti. An Esterel-based development environment for designing software radio applications. Technical Report RR. 4256, INRIA, September 2001.

[LA*et al*99] Laurent Arditi, Amar Bouali, and *et al.* Using esterel and formal methods to increase the confidence in the functional validation of a commercial dsp. In *FLoC'99, 4th ERCIM Workshop of Formal Methods for Inductrial Critical Systems*, 1999.

[LCJ95] L. J. Jagadeesan, C. Puchol, and J. E. Von Olnhausen. Safety property verification of Esterel programs and applications to telecommunications software. In *the 7th International Conference On Computer Aided Verification*, volume 939, pages 127–140. Springer Verlag, 1995.

[MM93] Michel Mouly and Marie B. Pautet. *The GSM System for Mobile Communications.* ISBN 2-9507190-07. Europe Media Duplication S.A, 1993.

[RA94] R. de Simone and A. Ressouche. Compositional semantics of esterel and verification by compositional reductions. In *the 5th International Conference on Computer Aided Verification*, page 818. LNCS, 1994.

[STB96] E. Sentovitch, H. Toma, and G. Berry. Latch optimization in circuits generated from high-level descriptions, 1996.

[THD99] T. Turletti, H. Bentzen, and D.L. Tennenhouse. Towards the software realization of a gsm base station. In *IEEE/JSAC, Special Issue on Software Radios*, April 1999.

[Vas00] Brett W. Vasconcellos. Parallel signal-processing for everyone. MS thesis MIT, February 2000.

[ZA92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* ISBN 0-387-97664-7. Springer-Verlag, 1992.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399