Implementing reactive programs on circuits
A hardware implementation of LUSTRE

Frédéric Rocheteau
DEC / Paris Research Laboratory

Nicolas Halbwachs
IMAG / LGI - Grenoble

Abstract. Synchronous languages constitute effective tools for programming real-time systems as far as they can be efficiently implemented. Implementing them by hardware is of course a good way for increasing their performances. Moreover, configurable hardware is now available which makes practical such an implementation. This paper describes an implementation of the synchronous declarative language LUSTRE on a “programmable active memory”.

Keywords: Reactive systems, synchronous languages, silicon compilation

Contents
1 Introduction
2 Time in synchronous languages
3 Overview of LUSTRE
4 Programmable Active Memories
5 Implementing boolean LUSTRE on the PAM
6 Extending LUSTRE for programming the PAM

1 Introduction

Synchronous programming [BCG87] has been proposed as a paradigm for designing reactive systems. It is an abstract point of view about real-time, which consists of assuming that a program instantly reacts to external events (synchrony hypothesis). It allows providing programs with precise, deterministic and machine-independent semantics. Several programming languages have been designed according to this point of view, e.g., STATECHARTS [Har84], ESTEREL [BG85], SML [BC85], SIGNAL [BL90] and LUSTRE [CPHP87].

In practice, an implementation on a given machine satisfies the synchrony hypothesis if the reaction time is always shorter than the minimum delay separating two successive external events. So, the only real-time problem with a synchronous program is to minimize and measure its reaction time. A specific compiling technique has been proposed [BG85] for synchronous languages, which synthesizes the control structure of the program as a finite automaton. This technique has been applied to ESTEREL and LUSTRE and has been shown to produce very efficient sequential code.
In this paper, we consider an other, more radical way for minimizing the reaction time of a synchronous program, which consists of translating it directly into a circuit. Synchronous languages are especially good candidates for such a translation, because usual circuits behave synchronously, from some reasonable level of abstraction (SML was designed as a hardware description language). And among synchronous languages, LUSTRE is perhaps the one for which this translation is the most natural: LUSTRE is a data-flow language, and one goal we had when designing it, was to be able to describe hardware. As a matter of fact, one solution considered for translating ESTEREL into circuits [Ber91] was to translate ESTEREL into LUSTRE.

One can wonder whether the hardware implementation of reactive systems is of general and practical interest, considering the cost of circuit manufacturing. A first answer is that many reactive systems — for instance low level communication protocols — are actually implemented by special purpose circuits. An other answer is provided by configurable hardware. The prototype compiler described in this paper configures a Programmable Active Memory (PAM [BRV89]), designed in the Paris Research Laboratory of Digital Equipment. By loading a bitstream — an operation performed in about 20 milliseconds — the PAM can be configured into any digital circuit.

The paper is organized as follows: In section 2 we explain the notion of time in synchronous languages, in order to show the importance of minimizing program reaction times. Section 3 recalls the main aspects of LUSTRE and the PAM is briefly presented in Section 4. In Section 5, we show how a boolean LUSTRE program can be translated into a circuit description which is accepted as input by standard CAD tools. Then, we describe some extensions to LUSTRE which are needed for using it as a programming language for the PAM (Section 6). These extensions concern arrays and only affect the surface level of the language.

Throughout the paper, we shall consider a very simple example of real-time program implementing a watchdog.

## 2 Time in synchronous languages

Let us first recall how synchronous languages pretend to express real-time constraints without making reference to a global physical notion of time. In the synchronous world, the notion of physical metric time is replaced by a simple notion of order and simultaneity between events. The physical time (measured in seconds, e.g.) will be considered as an input event, among others, and will not play any privileged role. We say that time is multiform. For instance, consider the two following constraints:

"The train must stop within 10 seconds"
"The train must stop within 100 meters"

There is no conceptual difference between them, and there is no reason to express them by means of different primitives, as it would be the case in languages where the metric time has a special status. In a synchronous language, they will be expressed by analogous precedence constraints:

"The event STOP must precede the 10th (100th) next occurrence of the event SECOND (METER)"