An Event-Driven Multi-Threading Architecture for Embedded Systems

Reinhard Gerndt[1], Rolf Ernst[2]
[1] *IAM FuE-GmbH,* Richard-Wagner-Str. 1, D-38106 Braunschweig, Germany
[2] *Institut für Datenverarbeitungsanlagen, TU Braunschweig,* Hans-Sommer-Str. 66, D-38106 Braunschweig, Germany

Abstract

In this paper we present an event driven multi-threading architecture and its underlying event flow system model of computation as a framework for the implementation of complex reactive and communication systems. Existing process oriented specification languages can be used to specify the system and embedded in the model. The target architecture covers a wide variety of architectures, varying from small FSMs to large processors, which are interconnected by a network template which performs dynamic scheduling and communication for different levels of process granularity and timing. Interconnect and module implementation and optimisation is based on an event flow graph model (EFG). In this paper we present our system model and the architectural template and show how they can be applied to an industrial application example.

1 Introduction

Increasing demand on high speed real-time control systems (fig. 1) and the necessity for an optimal exploitation of the design space, cause the need for new design methodologies and new architectures. The major optimization criteria are speed and costs. One way to control speed and area is to optimize parallelism over sequential processing. To fully exploit the design space, a variable granularity of parallelism within a design approach is needed. It is an observation, that most control schemes, as current methodologies and most processors, support only a fixed granularity of parallelism. Therefore only part of an embedded system may be implemented optimally. The limitations may be overcome by an event-driven system, that allows an arbitrary granularity.

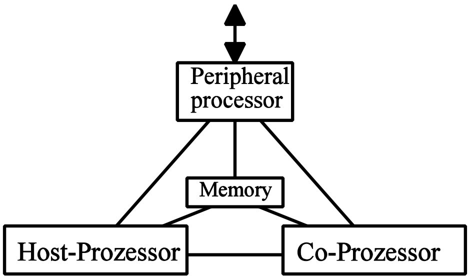

Figure 1 Embedded system.

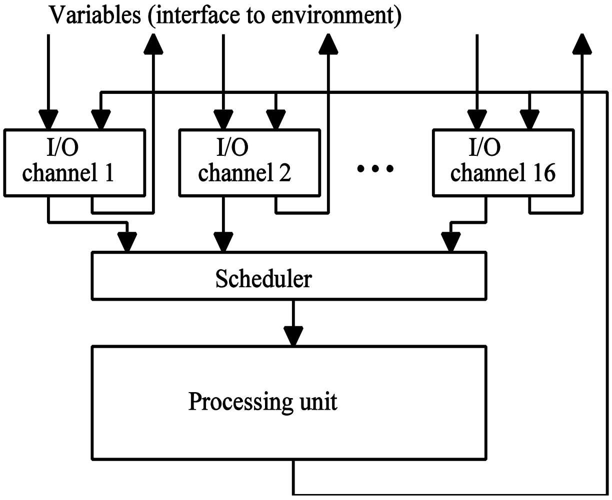

Figure 2 TPU structure

Examples for event-driven design methodologies and architectures are Motorolas time processing unit (TPU) [1] (fig. 2) and the PBS-communication controller [2]. The TPU shows a high amount of parallelism in event detection, but schedules its processes sequentially. It is supplied with 16 I/O channels, that detect events by monitoring input variables. As a reaction to an event, the I/O channel issues a service request to the scheduler and may change an output variable. The scheduler dynamically activates the programmable processing unit. The PBS exploits fixed instruction level and task level parallelism.

This paper is organized as follows: in section 2 we describe the underlying system model of computation. The architecture is described in section 3. We then present an implementation example in section 4 and our conclusions in section 5.

2 System Model of Computation

In a reactive system, processing is stimulated by possibly asynchronous external events [3, 4]. The embedded system processes data from input variables to determine new values for output variables or internal process states. We propose an event flow model, which is composed of a set of processes P, a set of event places E and a set of variables V. Functions I_{VP}, O_{PV}, I_{VE} and O_{EP} define the flow between P, V and E:

$I_{VP}: V \rightarrow P^*$, $P^* \subset \mathcal{P}(P)$: process input variable,

$O_{PV}: P \rightarrow V^*$, $V^* \subset \mathcal{P}(V)$: process output variable,

$I_{VE}: V \rightarrow E^*$, $E^* \subset \mathcal{P}(E)$: event detection input,

$O_{EP}: E \rightarrow P^*$, $P^* \subset \mathcal{P}(P)$: process activation input.

The event flow hypergraph EFG (fig. 3) is defined as a 7-tuple:

$EFG = (V, P, E, I_{VP}, O_{PV}, I_{VE}, O_{EP})$.

Hyperedges correspond to multiple copies of variables and events in the implementation.

A process p $\in$ P has at least an activation input, a data input and a data output. There is no restriction on the maximum number of inputs and outputs. Functionality of a process may range from hard-wired mapping functions to complex sequential operations. Depending on functionality and input data, the process may not compute a new value for its output data. The processes are created at compile-time and are activated by events. A process is executed upon activation and then terminates. Input data must be available prior to activation. The process is activated only once by an event. If a process is controlled by more than one event it is activated whenever any of them occurs (OR-condition). A number of processes may be activated by the same event concurrently.

Event places e $\in$ E detect events, which activate the related

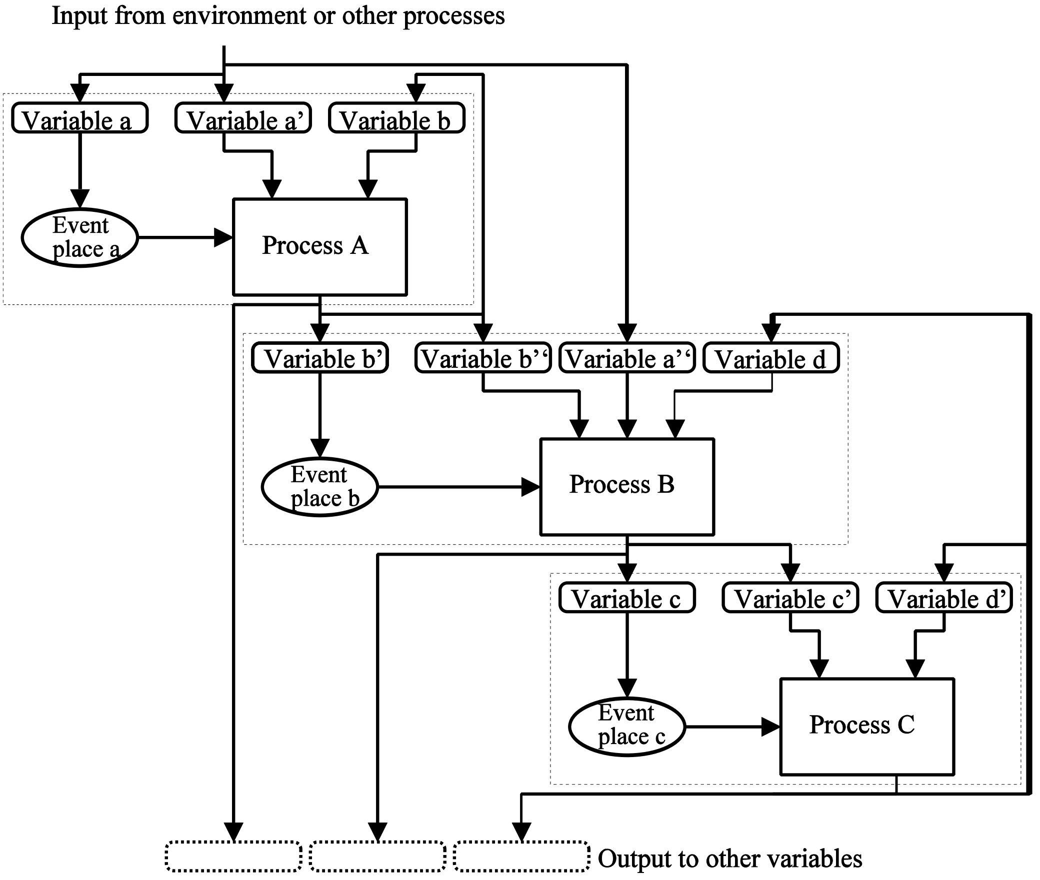

Figure 3 EFG system model, example, part of [5]

processes. An event is defined as a predefined transition of a variable. For a binary variable this can be a 0-1-transition. For extended variables more complex transitions can be defined. If a following event occurs prior to the execution of a process, the event may be ignored. This may be avoided by explicitly modeling an event handshake among processes. An event may control arbitrarily many processes concurrently.

Variables v ∈ V are used to store data and supply the interface between modules and to the environment. The set of vari-ables can be structured into the two subsets of event variables and data variables. Data variables supply input data for processes, event variables for event places. Variables must not be changed by more than one concurrent process at the same time.

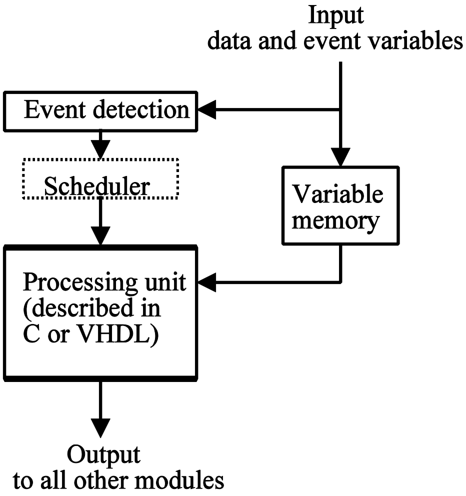

Figure 4 Implementation module structure.

Initially the system is modelled as a set of independent event trees, each with an input event as root and the possible output events as leafs. An input event, caused by a change of an input variable, propagates through the event tree and will possibly cause a change of either the state (set of all variables) or of an output signal. This way the event tree can be analyzed and implemented independently for each input event. In a second step, resource sharing is applied by tree merging.

The model has some similarity with colored Petri Nets [6]. However it is better suited to the reactive flow in our target architecture and leads to more compact descriptions.

The structure of the event flow model will now be illustrated with figure 3, showing the main functionality of the framing module taken from the application example in chapter 4. The modules basic functionality is to preprocess telegrams prior to transmission. Process A is activated by the availability of telegram data in variable a (with copies a' and a'') via event a. It controls the character sequence count (variable b) of the telegram. Process B is activated every time the sequence count (variable b) is changed via event b. Process B then routes either a single character of the telegramm data (variable a) or the error check character (variable d) or a framing character to the sequential output (variable c). (It serialises the telegram according to the sequence count (variable b)). Process C computes the error check character (variable d) from the sequential character stream (variable c). It is activated every time the output (variable c) is changed via event d.

The individual processes are described in a software or hardware language such as 'C' or VHDL. We can map the model onto hardware or software using the implementation module structure as shown in figure 4. Variables are stored in a local memory. Event variables are evaluated by the event detection unit. Data processing is performed by the processing unit. Processes may also be grouped to clusters to be implemented on the same module in order to control speed (resource sharing) and costs (local communication). To resolve concurrent events on a shared processing unit, a scheduler has to be added.

Concurrently executing data and control processing and communication not only offers a large design-space but also makes the approach potentially suitable for co-synthesis.

3 Target Architecture

Figure 5 shows the general outline of our target architecture. The processes are clustered on a number of concurrent processor nodes. This is currently done manually. A processor node may contain a set of sub-processors to form a hierarchy of independent processing units. The processors are interconnected such as to implement the EFG-edges, thereby preserving the order of messages. The communication network can be tuned to different applications, e.g. as a bus, for standard applications or with a function specific topology.

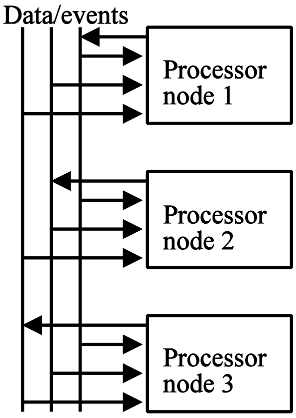

Figure 5 Target architecture

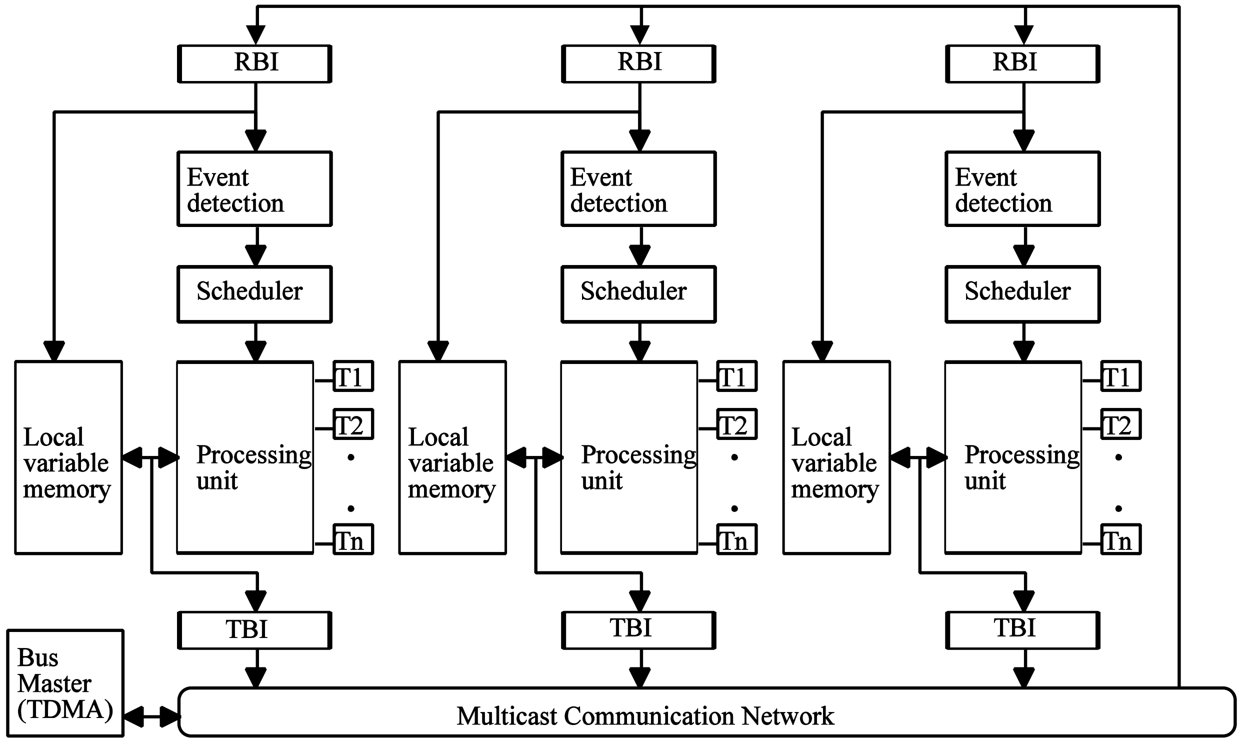

Figure 6 Detailed target architecture.

The detailed target architecture is shown in figure 6. The processing units are interconnected by a multicast communication network. Access is controlled by a bus master, e.g. with a TDMA strategy. The messages contain variable identifiers and values. The receive bus interfaces (RBI) filter the message stream to suppres irrelevant messages. The event detection unit evaluates the event variables for event conditions to possibly issue a request to the scheduler. Event conditions may be any changes of a variable or comparison with a given value or other more complex conditions. This covers the standard peripheral units in micro-controllers such as edge detection and capture and compare units [7]. The scheduler will then activate the processing according to the detected event. The type of processing unit (PU) may range from a small hardwired module as shown in the following example to a full size microprocessor or even a multi-processor system. When activated, it executes a predefined process (T1 - Tn) according to the system model and possibly computes new values for its output variables. The PU may be capable of executing a number of different processes as long as constraints are met under all conditions. Input data is read from the local variable memory. The output variables are buffered and transmitted to all other nodes trough the transfer bus interface (TBI).

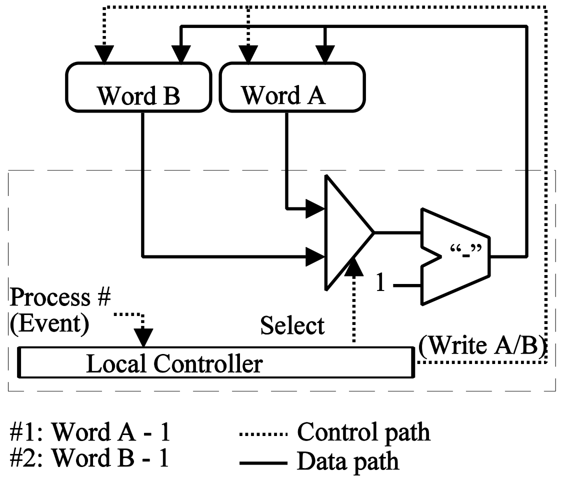

Figure 7 Processing unit example.

An example processing unit, capable of executing two decrement processes is given in figure 7 (For clarity we omitted the external data paths to words A and B). Process 1 decrements data word A, process 2 decrements word B. Both processes may be activated by different events, which must be mutually exclusive if a shared processing unit is used. The processing unit may e.g. be used for a cascaded down-counter. An external event may cause decrementing word A. If word A equals 0 this would be interpreted as an internal event, causing word B, the high byte, to be decremented.

4 Implementation results

The system model and architecture have been applied to a communication hardware controller design [5]. The functionality of the controller is to preprocess parallel I/O data and transmit them according to a given protocol [8]. The system was initially modeled using ‘C’, with independent processes of different granularity. The next step was to cluster processes with respect to concurrency, speed and costs. The clusters then where translated to a VHDL description of independent processor nodes and synthesised according to the architectural template (fig. 6).

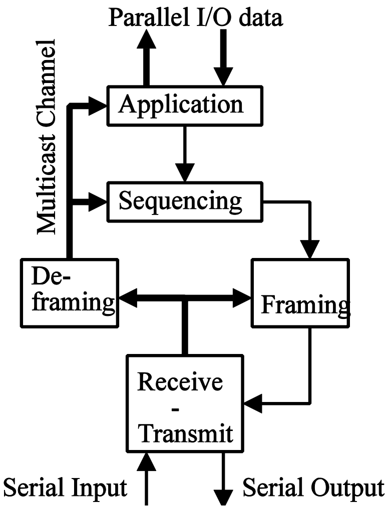

Figure 8 Application example

The hardware implementation contains 7 modules, figure 8 is showing the major ones. They are connected with an application specific topology of explicit communication channels. The receive-transmit-module monitors time and serial input data. Received data is transmitted to the de-framing module and thereby activates processing. De-framing outputs data and events to the sequencing and the application module. The application module controls the parallel output, if activated. In the opposite direction parallel input data is preprocessed and transmitted according to the event thread.

The size of the software, including some library functions, is 35 kbytes of executable code when the ‘C’-sources are compiled for an Intel personal computer. Supplied with the necessary resources, like a physical interface, it works as a real communication controller. Translated to VHDL and synthesized, the design amounts to 15 k gate equivalents of a 0.7μ gate array technology. The speed ratio of hardware over software was over 1000.

5 Conclusions

In this paper we presented a new event-driven system model and a target architecture for the co-design of highly dynamic embedded control systems.The approach allows the design of architectures with a low control overhead due to the event-flow concept. An industrial grade communication controller is given as an example.

References

[1] Motorola: ‘TPU - Time Processing Unit Refernce Manual’, 1990.

[2] R. Gerndt: ‘A Case Study in Co-Design of Communication Controllers’,International Workshop on Hardware/Software Codesign, Pittsburgh, Pennsylvania, 1996.

[3] N. Halbwachs: ‘Synchronous Programming of Reactive Systems, Kluver Academic Publishers, 1993.

[4] R. Kumar, V.K. Garg: ‘Modeling and Control of Logical Discrete Event Systems’, Kluwer Academic Publishers, 1995.

[5] M. Rudert: ‘Entwicklung einer Entwurfssystematik am Beispiel der FPGA-Implementierung eines Feldbus-Moduls’, Diplomarbeit, Fachhochschule Braunschweig-Wolfenbüttel (‘Application of a Design Method to an FPGA Implementation of a Fieldbus Module’, Master Thesis, Fachhochschule Braunschweig-Wolfenbüttel).

[6] J. L.Peterson: ‘Petri Net Theory and the Modeling of Systems’, Prentice-Hall, 1981.

[7] Siemens: ‘Microcomputer Components’, User’s Manual 6.90.

[8] EN 50170, European Fieldbus Standard.