Hierarchical state diagram and the transition to synthesis of telecommunications protocol automaton

Osama Ahmad Salim Safarini

Computer Engineering Department
University of Tabuk,
Tabuk, KSA
osafarini@ut.edu.sa
usama.safarini@gmail.com

Abstract— this article describes the approach to the organization computing in the telecommunications machine called serialization. It provides for a hierarchical DST partially formalized based automaton telecommunication specifications. Hierarchical DST described limited set of transitions allowable species-specific telecommunication systems. Implicit transitions and states are determined at compile time for DST implementation in the target system.

On the basis of this approach we created a language specification of hierarchical DST, designed for quickly and efficiently create implementations of telecommunication automaton in target systems for various purposes.

Index Terms— DST - Diagrams of states and transitions

INTRODUCTION

Currently, there are large-scale development telecommunication systems, caused by the expansion of the spectrum communication services provided by a wide class of consumers. New services on the part of communication systems based on the new telecommunication protocols. The development of such protocols is an expensive technical task automation which still has no generally accepted solution, in including the final stage of development − implementation of the protocol in the target telecommunication system.

The main difficulties automating the process of implementation telecommunications protocol in the target system are related, primarily to two aspects:

1. lack of complete specifications entire protocol in a formal language (although there are rare exceptions);
2. Parallel computing**.**

The first aspect is related to the fact that usually a telecommunication protocol is described as a set of formally specified fragments and informal description of their interaction. This feature of description leads to that implementation of the protocol in the target runtime system is made by the developer of telecommunication systems and is not subject to formalization.

In other words, the transition from partially formalized description of the telecommunication protocol automaton to its software implementation is carried out in most using heuristic methods.

Automating the generation of the resulting code requires a completely presence of formalized description of the protocol automaton at a high level language. The absence of a fully formalized description of the protocol automaton causes, in the implementation, that arise of implicit state and transitions to be processed in the resulting system. Synthesis algorithm protocol automaton shall include monitoring of implicit states and transitions, and preferably automated redefine them.

The second complex set of tasks associated with the organization of parallel computing, traditional and goes back to the theory of finite automaton. The logical part of the telecommunication protocol traditionally described using state diagrams and transitions. These diagrams are widely used for the synthesis of finite automaton based on combinational circuits with memory elements. However, for the past tens of years of implementation telecommunication protocol automaton for the most part uses a software platform based on microcomputers. The transition from circuit implementations to software creates a number of difficulties associated with the provision of parallel computing. Usually this problem is solved by software emulation of parallel computing combinational circuit with varying degrees of effectiveness. In this paper we propose an alternative approach, which consists in organizing sequential signal processing (serialization computing).

1. Diagrams of states and transitions

Diagrams of states and transitions (DST) are often used to visualize finite automaton. In literature DST often identified with the finite automaton, that is not quite true [1-5]. Broadly, a finite automaton is any digital device, whether combinational logic scheme or program-controlled computer. Accordingly, under DST should understand certain visual form of representation of a class of algorithms.

Very close to the DST are the graphical flowcharts of algorithms. Flowcharts are commonly used to describe software algorithms, while DST often used for synthesis electronic circuits. Nevertheless, both these graphic presentations of the algorithms have much in common, and often are interchangeable.

Not all algorithms and devices are described using DST and yet the scope of the DST is very wide. To algorithms traditionally described using DST include:

1. electronic logic circuits of moderate complexity;
2. protocol automaton;

3. control algorithms for technological processes.

The latter can be attributed to software intelligent devices including machines and household appliances.

Note that all of these classes of algorithms combine the necessity of processing multiple external signals that occur regardless of the current state of the algorithm. Often these signals are called asynchronous signals, or real time signals. It can be assumed that the use of DST in these areas due to the peculiar features of the structural decomposition of DST, - namely, the ability to partitioning of the system by groups perceived external signals, as well as by groups of states receiving these signals.

Such a method of decomposition suggests the possibility of determining a plurality of subsystems that operate practically independently. These subsystems are executed in parallel branches of the general algorithm, or sub processes. In this case, each individual sub-system may be represented by a separate procedural program; however, the whole system requires the use of any parallel programming tools.

The finite state automaton may be implemented in hardware (such as electronic circuitry, or any other physical device) or as a computer program. The essential difference of these implementations is the use in the second case unified hardware platform computers, versus specificity of the electronic circuit. In other words, the software implementation of the algorithm makes it possible to perform various algorithms on the same hardware platform or one and the same algorithm - in various hardware devices, while the hardware implementation tougher associated with a specific algorithm.

Usually, the development of algorithms for hardware and software implementations is performed by different means. However recently, in practice there are problems increasingly replacing hardware implementation of the algorithm to its software equivalent and vice versa, especially in the development stage of the algorithm. Modern hardware implementation of algorithms for the most part provide for software simulation of hardware.

Modeling can held at different levels of detail. The most upper ("logical") levels of simulation are close to the software implementation of the algorithm. Accordingly, modeling tools of hardware systems are very closely related to the Software implementation of algorithms. The most significant difference between software implementations of the algorithm from the hardware is the technology of processing asynchronous signals.

For a hardware implementation of the algorithm each input signal generally corresponds to a separate input line that contains certain circuit elements that operate independently of the rest of the circuit. Therefore, for this type of implementation the separate structural part of the device operates in parallel and independently (asynchronously) from each other. Interaction between the parts of the device are logically does not differ from their interaction with the outside world.

Software implementation of the algorithm involves the execution of the program on the von Neumann machine. Von Neumann machine in turn, supposes consecutive execution of predefined instructions, otherwise referred to as procedural programs. Procedural program can receive data from the outside only at predetermined locations, typically - at the beginning of program execution.

Thus, the software implementation of the algorithm, unlike the hardware can not contain completely independent functional parts:

All components of the software algorithm will inevitably be synchronized with each other that inevitably reflected in the processing algorithm of asynchronous signals.

The need to handle asynchronous signals in sequence (procedural) program - serialization results to the creation of parallel branches of the algorithm: asynchronous signal should be interpreted independently of the state of the main program, and then correctly transmitted to the main program, in other words, "synchronized" or delayed until a certain point of execution, in which the main program will be ready to accept new data.

Thus, support for asynchronous signals causes to life concepts such as real-time signals, interruption, parallel execution, the division of time, preemptive multitasking, synchronization, blocking, atomic operation, etc... .

2. Hierarchy

The degree of complexity of modern protocol automaton in particular used in wireless telecommunication systems has been steadily increasing. For the design of complex systems is a necessary condition of the possibility of decomposition into simpler parts and the possibility of independent development of independent parts.

Partitioning systems into simpler parts are often called hierarchical structural decomposition of the project. This implies that at every level of system representation the developer has to deal with a number of independent entities manageable lower levels, the so-called primitives, each of which in turn may be regarded as an independent subsystem of lower level primitives.

Note that when the decomposition of the protocol automaton, held at the stage of development of the telecommunication protocol, there is a problem appearance of implicit transitions. This is due to the fact that the formal description of the fragments of an automaton interaction between the fragments is not formalized. We explain this situation as an example.

One of the fragments of an automaton telecommunications protocol produces the connection. This starts the timer, which is tasked with tracking failure physical signal. In the case of a signal from the timer transition is made to re-establish the connection. The following fragment of the protocol automaton (at a higher level) provides data exchange, and it does not provide signal processing of the above timer, since the interaction of the fragments is not formally specified. However, the appearance of the given signal protocol automaton fragment that is responsible for the transfer of data must be completed correctly and to transmit control in a fragment of connection establishment. Thus, there is the appearance of an implicit transition is not directly described in the formal specification of the protocol automaton, but implied in the informal part of the description.

The objective of the software code is generation of code protocol automaton in the target language and tracking such implicit transitions and their inclusion in the general scheme of the transitions of the protocol automaton. To ensure traceability of such transitions in the language of the description of the protocol automaton shall be provided special means to set the rules of interaction between the fragments of the automaton.

We propose to describe the protocol automaton introduce the concept of hierarchy.

To do this, we introduce the concept level fragment automaton in the overall structure of the automaton. For each level, all of the signals are described, which can be obtained at this level, furthermore, are described transition rules (functions) from level to level. All protocol automaton is represented as a hierarchy (possibly branching) these fragments.

Target code generation system must keep track of all a plurality of input signals, regardless of the level at which the signal is received. And in the case where the signal causes automaton transition to another level of hierarchy correctly implement rules of transition throughout the chain.

In the case of such a structure in the above example ambiguity does not arise with the transition of the signal timer. Upon receipt of this signal in the fragment, that is responsible for the transmission of data, will be caused by the transition rule that provides the transition to the lower layer. This function correctly completes the lower layer of the data transmission algorithm, and then transfers control to fragment of establishing a connection. If the transition is carried out through several levels of hierarchy, rules for the transition between the levels will be called consistently for all intermediate levels that will correctly complete all parts of protocol automaton.

There is a similar situation with the transition to a higher level. Transition rule performs a necessary component initialization and start the next fragment of the protocol automaton.

Language for describing the protocol automaton that provides the ability to set the level of the hierarchy of a fragment of the automaton, as well as the rules of transition between levels allows you to fully formalize the specification of the protocol automaton that will provide the ability to automate the translation into the language implementation.

3. Hierarchical Language DST

Existing formal description languages DST have significant limitations in the applicability, related to the specific requirements of the runtime, cumbersome and inconvenient text description DST, and structuring difficulties. Procedural languages are not very useful for the implementation of the DST. The main obstacle to the creation of an adequate Language DST is a contradiction between the parallel (or asynchronous) nature of DST and consistent course of execution programs written in procedural languages. In other words, the problem of software and traditional consistent implementation has not yet received a decent resolution.
In this paper we propose to create a subset of the language based on the existing description languages DST for efficient synthesis software telecommunications protocol machines and control software for microcontrollers based on the hierarchical of language constructs. Such a language should provide translation of hierarchical DST into procedural program suitable for sequential execution.
The initial data for this should be a description of DST algorithm in the form of a compact text form. Graphic form DST can be linked to the text using pragmas. It is of interest the automatic synthesis of a graphical representation of the text, and vice versa.
Hierarchical Language DST must be meta-language that combines three specialized languages:

1. procedural language for creating procedural blocks, mainly language C;
2. data description language (to create a protocol automata, where the protocol data units can be specified in a machine-independent format, such as ASN.1 or TLV);
3. Actual language DST, unifying and structuring procedural blocks written in a procedural language.

Syntactic language DST must be clearly separated from the procedural language. It should allow the layout separately compiled parts of a program using a minimum amount of external common definitions. In addition, the language must allow representing the algorithm on several levels with the necessary level of detail for each of the levels.
In other words, one state of the system at some level if necessary should be independent subsystem with a set of their states.
We illustrate this with two examples. In Many protocol automata a number of protocol procedures single level objects is described as a sequence of exchange certain messages. At the same time, the transmission and reception of messages are themselves complex algorithmic tasks. Therefore, at different levels of representation needed to see the transmission of the message or as an atomic action, or as a deployed algorithm. Another example: the device in interactive input state of the input data must be some way to respond when buttons are pressed. It is obvious that at the same level of representation is the "input state" should be displayed simple state, but on the other – as independent algorithm with input and correction of data.

Conclusion

Thus, consider a method of serialization computing providing for the creation of a hierarchical diagram of states and transitions telecommunication automaton. Hierarchical DST described by a set of transitions allowable kind-specific for telecommunication systems.
Implicit transitions and states are determined by the DST compiler for implementation in a specific target system.
To describe hierarchical DST we proposed language specifications for fast and efficient creating implementations of telecommunication automaton in the target systems for various purposes.

References

1. Specification and Description Language (SDL). Recommendation Z.100. - Moscow: International Telecommunication Union, 1992.
2. Armstrong J.R. Simulation of digital systems in the language VHDL. - Moscow: Mir, 1992.
3. Zubin V.E. PLC programming: languages for Simulation Electronic Computer 61131-3 and possible alternatives // Industrial Automated Control Systems and controllers. -2005. - №11. - pp. 31-35.
4. The nesC language: A holistic approach to networked embedded systems. 2003. http://nescc.sourceforge.net/papers/nesc-pldi-2003.pdf
5. *Paul Jay Lucas*. An object-oriented language system for implementing concurrent, hierarchical, finite state machines. MS Thesis. — University of Illinois, 1993.