Partitioning of Mealy Finite State Machines

Arkadiusz Bukowiec * Luis Gomes **

* University of Zielona Góra, Institute of Computer Engineering and Electronics, Zielona Góra, Poland (e-mail: a.bukowiec@iie.uz.zgora.pl)
** Universidade Nova de Lisboa, Departament of Electrotechnical Engineering, Caparica, Portugal (e-mail: lugo@ieee.org)

Abstract: As widely accepted, the most popular way for realization of control units are finite state machines. Up-to-date control unit circuits very often are implemented using programmable logic devices. Microprocessors can be also considered as a solution taking costs into account. But very often microprocessors are too slow for realization control units of digital systems. The partitioning of state machines can be a solution for this problem allowing a parallel execution of sub-state machines, keeping performance, cost, and energy consumption at adequate levels. In this case, the time critical part of the control unit (associated with specific sub-state machines) can be implemented in fast FPGA device and other parts can be realized by cheaper platforms (namely based on microprocessors). Additional advantage of such solution is that each part can be synthesized using different methods. The problems and algorithms of partitioning of state machines are discussed in this paper. A CAD tool for partitioning implementing the proposed algorithm is also presented.

Keywords: control units, finite state machines, model partitioning, FPGAs, KISS

1. INTRODUCTION

Most digital systems can be decomposed into control unit and data path (De Micheli, 1994). The most popular way for specification and implementation of control unit is based on finite state machines (FSMs) (Hopcroft and Ullman, 1979; Baranov, 1994). Control unit circuits very often are implemented using programmable logic devices such as CPLD or FPGA (Jenkins, 1994), depending on complexity. Coping with tradeoffs between costs and performance, microprocessors/microcontrollers can also be considered as a solution. However, very often microprocessors are too slow for implementation of control units for critical parts of digital systems. On the other hand, implementation of very large state machines is very expensive and performance can be penalized (depending on implementation method).

The partitioning of FSM into a set of concurrent FSMs can be a solution for these problems (Jóźwiak, 1995; Gomes and Costa, 2003).

Also, when addressing hardware-software co-design of embedded systems, the partitioning of state diagram could be very effective in order to map different components into different implementation platforms, according with target costs and performance. Whenever it is needed to handle large state diagrams, it is mandatory to rely on some structuring mechanisms, like hierarchical representation, which is provided by different state-based modeling formalism. Statecharts (Harel, 1987) can be considered as a representative for those well-known set of formalisms supporting hierarchy and parallel composition of state diagrams.

The paper starts with a brief description of the proposed methodology and introduces the notation used for state diagram representation. In the following sections, the proposed partitioning algorithm and details about its implementation are presented. The next section presents an example in order to allow illustration of the referred procedures; validation of the results was achieved through implementation; associated simulation results are presented. At the end, final remarks regarding the example are produced and conclusions are presented.

2. METHODOLOGY OVERVIEW

Instead of addressing the whole methodology flow, the paper focuses only on the part related with the partitioning of state diagrams, more specifically Mealy state diagrams. It is important to note that considering only Mealy machines has no loss of generality compared with Moore machines (Hopcroft and Ullman, 1979), as both models have equivalent modeling capabilities and it is possible to obtain a behaviorally equivalent Mealy machine from a Moore machine.

Figure 1 summarizes the associated flow. Starting with a state diagram, and based on the definition of a cutting set composed by a set of transition arcs, a set of of parallel components is obtained. Each of these components is also a state machine (also referred as a sub-state machine, or sub FSM). The components will be executed in parallel with the support of additional synchronizing signals, which were created for inter-component communication purposes.

If one generates the state space associated with the execution of this set of generated communicating components,
Fig. 1. State Diagram partitioning

it is important to note that the generated state space is behaviorally equivalent (which means isomorphic) with respect to the initial state machine.

Figure 2 presents a high-level characterization of the implementation architecture, where each component (sub-state diagram) receives as inputs the inputs of the initial state machine (or a sub-set of it, depending of the dependencies of the specific state machine arc expressions) and a set of signals generated by the other components to allow synchronization of global evolution (as a matter of fact, only signals generated by adjacent components are relevant).

Fig. 2. High-level implementation architecture

After the partitioning of the initial state diagram, a set of components are ready to be deployed in a specific implementation platform. In this sense, the proposed procedures are applicable within different implementation scenarios:

- hardware-centric implementations, where only hard-
  ware implementations of the sub-state machines are foreseen (for instance, from asynchronous circuits to synchronous circuit with different clock signals to accommodate proper power consumption levels);
- hardware-software implementation using co-design techniques, where each component could be mapped into an hardware platform or a software process (Jóźwiak et al., 2006).

It is important to note that whenever different execution times (for instance, clock rates) are used for the different components, special care needs to be taken (and specific modules need to be used) to assure a secure and reliable inter-communication between parallel components (Jantsch, 2003). However, this is not a problem whenever a global clock is considered, as in the running example used in this paper (to be presented in following sections).

Using the referred partitioning procedures, the time critical part of the control unit can be implemented in fast FPGA device and other parts can be realized by cheaper devices or microprocessors. On the other hand each part of the system can be synthesized and implemented using different methods of synthesis (Barkalov and Titarenko, 2008; Bukowiec, 2006).

Unfortunately, the fully automated partitioning process based on mathematical graphs algorithms can not be achieved, as state diagram doesn’t give information about required execution time of the different transition functions (which means execution performance), probability of transition, etc.

We argue that this partitioning can be achieved starting with a set of transitions selected by designer and a computational tool to produce the partitioning itself. This tool, named as DivideFSM, is presented in this work.

3. BASIC DEFINITIONS

In the current section, two basic definitions are formally introduced presenting characteristics of initial state diagram and of sub-state diagram, as presented in previous section.

**Lemma 1.** A State diagram is a 7-tuple \( SD = (S, s_0, I, O, F, AC, AA) \), where

- \( S \) is a finite set of states,
- \( s_0 \) is the initial state,
- \( I \) is a finite set of input Boolean signals,
- \( O \) is a finite set of output Boolean signals (Mealy-type outputs),
- \( I \cap O = \emptyset \)
- \( F \) is the transition arc relation defining the source state, attached transition expression, destination state, and attached output signals for each transition arc \( F \subseteq S \times AC \times S \times AA \),
- \( AC \) is a finite set of input signals expressions; each expression is composed by a Boolean product term expression containing from none to all input signals in \( I \); the evaluation is evaluated to true whenever no input signals are involved,
- \( AA \) is a finite set of output signals expressions; each expression is composed by a list of output Boolean signals containing from none to all output signals in \( O \).

**Lemma 2.** A Sub-State diagram is a 9-tuple \( SSD = (S, s_0, I, IA, O, OA, F, AC, AA) \), where

- \( S \) is a finite set of states, composed by the union of a subset of the set of states of the main state diagram and an additional state representing inactivity of the sub-state diagram.
- \( s_0 \) is the initial state.
- \( I \) is a finite set of input Boolean signals (same as in main state diagram).
- \( IA \) is a finite set of input Boolean signals (coming from other sub-state diagrams, from \( OA \) set of output signals).
- \( O \) is a finite set of output Boolean signals (Mealy-type outputs) (same as in main state diagram).
4. REPRESENTATION OF FSMS

Behavior of FSM can be described using different notations but the state diagrams are the most common graphical representation (De Micheli, 1994). One common formal representation of a state diagram is direct structural table (DST) (Baranov, 1994). The advantage of this format is that it can be easily represented as text format. Typically such table consist of columns representing a initial state of FSM, a state of transition, logic conditions and the microinstruction formed under transition and each line describe one transition. One of the most popular text formats describing DST is the KISS2 format (Yang, 1991). A file in this format consists of a header and a table (Fig. 3).

Fig. 3. State diagram (a) and KISS2 description (b) of FSM S1

The header includes information about the number of inputs (.i), the number of outputs (.o), the number of table lines - products (.p), the number of states (.s). There can be also specified the optional information about the initial state (.i). The table describes the behavior (transitions) of FSM. The '-' sign in logic condition means that this input variable does not affect this transition. The '0' value means that negation of this variable should be placed in logic condition and the '1' value that its affirmation should be placed in logic condition. The '0' value of the output signal means that this microoperation does not belong to formed microinstruction and the '1' value means that it belongs.

5. ALGORITHM OF PARTITIONING

The goal of partitioning is to start with one initial FSM and ending-up with a set of concurrent FSMs, obtained from the partitioning of the initial model (Gomes and Costa, 2003). Several strategies can be used to identify the set of transitions to be included in the cutting set in order to split the model and to produce the different concurrent components: from completely ad-hoc techniques to identification of strong connected components. The only restrictions that this transitions cutting set have to obey is to produce a set of unconnected components after its removal. The procedure can be summarized as follows:

- Each of the decomposed FSMs (one component), also named as sub-FSMs, is composed by a set of states and transitions;
- In the sub-FSM associated with the first component (the one with the initial state), one new state is added. This state represents the other components (from the point of view of this sub-FSM, all others sub-FSMs collapses into this one new state);
- In the others sub-FSMs associated with others components also one new state is added. This state also represents the other components but in this case it is also the initial state of considered sub-FSM;
- New transitions are added between the new states and the other states of the component according with the transitions of the cutting set. Associated conditions include a dependency on the activation of the origin state.

Figure 4 illustrates the process of partitioning. On the right side, there is also presented scheme of implementation before and after partitioning.

6. IMPLEMENTATION OF ALGORITHM OF PARTITIONING

The referred procedures are implemented by a CAD tool, called DivideFSM, presented in this work, which associated algorithm is presented in Fig. 5.

This tool starts with a state machine description based on KISS2 (Yang, 1991) format and with the transition cutting set. The identification of the cutting set transitions is accomplished marking those transitions in the initial description with a specific directive (seen as a comment in KISS2 syntax). As result, a set of output KISS2 files describing each sub-FSM are produced. An initial state per sub-FSM is assured:

- if the initial state of original FSM belongs to states of the sub-FSM then the initial state is set to the original initial state,
- in other cases the initial state is set to the newly created state (wait-state).

Additional signals are also created in order to assure proper communication and synchronization between sub-FSMs and wake them up to work, and assure that they are able to return to a non-active state.

After reading the initial source file and extracting header data and DST, the DivideFSM tool enters main compilation. First step is to create special matrix that de-
Fig. 4. Idea of FSM partitioning and implementation

determines which states are connected between them. This table does not include transitions that belong to the transition cutting set - include only transitions that belong to components. Then, based on this table, lists of states per component are created; each list consists of names of states from each particular sub-FSM. After that, it is possible to check if the cutting set of transitions is valid; if for any transition belonging to the cutting set, the current state and next state of the transition belong to the same sub-FSM then the cutting set is invalid and an error message is generated and program is aborted. A "connection table" describing which sub-FSMs are connected between them is created afterwards. After this preparation steps, the main algorithm of creation KISS2 description of sub-FSMs starts. Every sub-FSM is created one-by-one using the same function. First, the number of inputs and the number of outputs of sub-FSM are calculated based on the connection table; also the number of states based on the list of states. The number of lines and the initial state of the sub-FSM are determined. If the initial state of original FSM belongs to list of states of sub-FSM then the initial state of original FSM; in other case the initial state of sub-FSM is set to a new wait-state. These steps create the header of KISS2 file of every sub-FSM. Then the DST of each sub-FSM is created. The algorithm reads each line of the original DST and:

- If nor current state neither next state do not belong to the list of states of this sub-FSM then this line is ignored;
- If current state and next state belong to the list of states of this sub-FSM then this line is copied to the DST of this sub-FSM;

7. ANALYSIS OF EXAMPLE

Algorithm and DivideFSM tool presented in previous sections are analyzed base on FSM S₁ with Mealy outputs (Fig. 3). Transitions from the cutting set are marked with
scissors icon in the state diagram (Fig. 6a) and directive #dvtr in the KISS2 file (Fig. 6b).

As a result of partitioning there were received three sub-FSMs. The First sub-FSM S₁ includes the initial state a₁ (Fig. 7).

There is also created new wait-state SW1. The additional signals in_a2 and in_a3 are generated as Mealy outputs only during transitions into wait-state. These signals are used to wake up other sub-FSMS into activity. Other two sub-FSMS S₂ and S₃ (Fig. 8) and S₁ (Fig. 9) have also new wait-states SW2 and SW3 but in this case these states are also initial states.

The sub-FSM S₀ generates additional in_a4 output signal base on the same way as for sub-FSM S₁. There are also additional input signals in_a2, in_a3 and in_a6 from others sub-FSMS. These signals are responsible for waking up this sub-FSM from wait-state SW2. Base on the same way there is additional input signal in_a4 in sub-FSM S₁ and this sub-FSM generates additional output signal in_a6.

Such sub-FSMS can be implemented using different technologies or methods in different circuits (Baranov, 1994; De Micheli, 1994). To satisfy proper functionality of the algorithm, and to assure that the execution of the set of parallel components is behaviorally equivalent to the execution of the initial FSM S₁, all sub-FSMS have to be properly connected together. In general, the top-level module should satisfy these conditions:

- All sub-FSMS should have the same Clock and Reset signals;
- All sub-FSMS should have the same input signals;
- Additional output signals (for synchronization) should be connected with corresponding (with the same name) additional input signals in others sub-FSMS;
- Corresponding output signals from all sub-FSMS should be connected together in "final" output signal via OR-gate.

For example, the top-level circuit associated with partitioning of FSM S₁ is shown in figure 10 with connected sub-FSMS.

For verification and simulation purposes, KISS2 description of FSM S₁ was converted into VHDL using a tool worked out at University of Zielona Góra (Figler, 2006). Design of FSM S₁ before partitioning consist of one file describing behavior of FSM. Design of FSM S₁ after partitioning consist of four files: three files describing behavior of each sub-FSM and one top-level module describing connections between sub-FSMS. Simulation results of both approaches are shown in figure 11.

As expected, the results of both simulations are the same. Output signals always have the same value. In simulation of FSM S₁ before partitioning (Fig. 11a) there is one variable describing current state. In simulation of FSM S₁ after partitioning (Fig. 11b) there are three variables describing current state - one variable per each sub-FSM. How you can see there is always only one sub-FSM in active state and other two are in wait-state. The sequence of states for both simulations are also the same.
seems to be good candidates to be considered for help in this case.

8. RELATED WORK

Usage of tools like the one presented in previous sections can be very helpful during designing big control units that cannot be implemented with the use of only one device, or that can have different modes of operation with different associated performances. In case of application of partitioning of FSMs only time-critical part of FSM can be implemented in fast FPGA device (that are rather expensive) using method of synthesis oriented to optimize speed (consume more hardware), or to optimize area, or other criterion (Jenkins, 1994). These methods can be optimized by applying functional decomposition (Luba et al., 2002) because these manipulations typically do not affect on time of execution of algorithm. Other components (considering lowering the cost of design) can be implemented in microprocessors/microcontrollers or in cheaper field-programmable devices (FPDs) using different methods of synthesis (Baranov and Titarenko, 2008; Bukowiec, 2009).

In general, partitioning of state machines allows to distribute control algorithm and lower costs of its realization without losing of performance. Additionally each sub-FSM can be implemented using different technology and method of synthesis and this method can be chosen adequate to needs: speed, time, etc.

9. CONCLUSIONS

The aim of this paper is to analyze and present algorithm of partitioning of finite state machines (Gomes and Costa, 2003) and CAD tool that implement this algorithm. This tool creates descriptions of each sub-FSM based on description of an initial FSM and the set of transitions cutting set.

A running example was used illustrating the development flow applying the proposed technique. The tool is publicly available, as well the examples presented in the paper, at the DivideFSM webpage (Bukowiec and Gomes, 2007).

In future works there could be created additional algorithm for automatically creation of the cutting set of transitions. But in this case, additional information should be analyzed, like required execution time of microinstruction, probability of transition or others. This task could be very difficult and genetic algorithms or data mining algorithms seems to be good candidates to be considered for help in this case.

REFERENCES