An Approach to Dynamic Reconfigurable Transport Protocol Controller Unit Development

Elena Suvorova
Saint-Petersburg State University of Aerospace Instrumentation
Saint Petersburg, Russia
suvorova@aanet.ru

***Abstract*—Transport Protocol Controllers are parts of most terminal nodes of local networks widely used in information and telecommunication systems. Typically, a terminal node (node of local network) is exploited for several years. New applications, new versions of transport protocols or new transport protocols could be developed during this time. Support of these new features without change of equipment is very important today. Also set of tasks could be dynamically changed that leads changes in data flows. These changes in turn leads the changes of used transport layer protocols or their profiles. The Transport Protocol Controller unit should be dynamically reconfigurable to meet these requirements. Today dynamically reconfigurable components usually are developed with Field Programmable Gate Array (FPGA). However, power consumption, area, time characteristics (e.g. achievable clock frequency) of FPGA implementations are essentially worse than same parameters of Application Specific Integration Circuits (ASIC) implementations. These factors essentially constrain the application area of FPGA based dynamically reconfigurable systems.**

In this paper, we consider existing approaches for development of dynamically reconfigurable systems with ASIC, evaluate its applicability for Transport Protocol Controller Unit. We propose an approach to development of dynamically reconfigurable Transport Protocol Controller Unit. This approach allows us to take into account the specific requirements for this unit. In the paper we present several examples of the proposed approach. We have evaluated reachable parameters and overheads for these examples.

I. Introduction

One of main requirements for many modern local networks used in information and telecommunication systems is the ability to change of operating mode during exploitation. The mode can be changed, for example, due to change in tasks set in the system or due to faults in components [1], [2], [3], [4], [5]. These changes may lead the change of used transport layer protocols set, their profiles. In addition, new transport protocols and new versions of transport protocols, well suited for decided tasks, may appear during lifecycle of the system.

Correspondingly, the Transport Protocol Controller Unit should support different protocols. But area and power consumption of controllers is usually strongly constrained. Therefore, the possibility of dynamic reconfiguration is one of the most important features for this unit.

The Transport Protocol Controller Unit may be realized entirely in hardware, entirely in software or as hardware-software component. On one hand, entirely software realization provides very wide possibilities for dynamic reconfiguration. But on other hand this approach has several essential disadvantages. Typically, the dedicated processor core can not be used as the packet distribution unit due to too large area and power consumption (and power distribution). To process the packet flow in real time, the processor core should operate at frequency ten times higher than rate of packets arrival. Therefore, power consumption is unacceptably high.

The dynamically reconfigurable hardware can be realized on FPGA. The entire or partial change of bitstream allows us to change configuration of several units or whole device. But, power consumption and area of FPGA is essentially higher than that of ASIC realization due to its structure. Many parts of device do not require dynamic reconfiguration. Therefore resources of FPGA are used not efficiently.

In the paper we propose an approach to development of dynamically reconfigurable controller with ASIC that allows us to avoid the problems specific to software implementation and implementation on FPGAs and meet user constraints.

In the paper, we do not consider the ways of packet processing rules specification (corresponding to the Transport Layer Protocol). The specification used for Software Defined Network-on-Chip or other specifications can be used.

The paper is organized as follows. In section 2 we briefly describe the requirements to the Transport Protocol Controller Unit, its structure and architecture. In the next section we consider the existing methods for dynamic reconfiguration using ASIC technology. The proposed approach to development of the dynamic reconfigurable Transport Protocol Controller Unit is described in section 4. Several examples of this approach applying are presented in section 5. Section 6 concludes the paper.

II. The requirements to the transport protocol Controller unit, its structure and architecture

Transport Protocol Controller Unit may be used as part of transport layer units. Typical structure of transport layer is represented in Fig. 1. Number of Protocol Controller Units depends on quantity of packet flows that should be processed concurrently. The packet Distribution unit is used when Transport Layer Unit includes more than one Transport Layer

Controller Unit. We have proposed an approach to development of Dynamic Reconfigurable Packet Distribution Unit in [36].

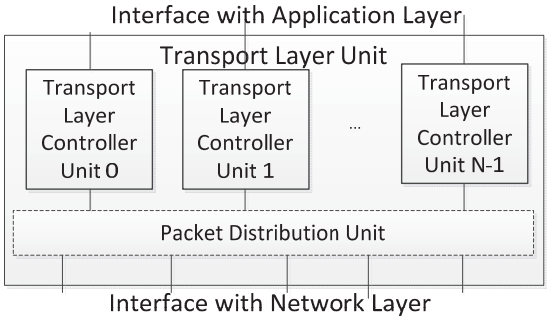

Fig. 1. Typical structure of transport layer unit

The following system level requirements and limitations should be considered when designing the packet distribution unit:

- support for a set of transport protocols that is known in advance;
- support for a set of new transport protocols;
- area and power constrains;
- specific system requirements.

Let's consider specific system requirements. Transport Layer Controller Unit should provide data streaming interface with the Network layer. Receiving data should be processed and sending data should be generated "on the fly". Several actions can be performed for every data words (e.g. CRC count, context check and etc.).

To implement most of the existing transport protocols, Transport Protocol Controller should process several event flows concurrently, perform several actions at the same time (e.g. receiving and transmission of data and service packets, different guardian timeout control, and etc.).

Thus, at each stage of the data processing, there is a need to perform several actions in parallel and decide to move to the next state under several conditions.

There are quite heavy restrictions on the area and energy consumption for many types of systems (e.g., for embedded systems). The presence of heavy restrictions on energy consumption, in turn, leads to the fact that the Transport Protocol Unit shall operate at a frequency corresponding to the frequency of receipt of data words. Therefore, most of the data words must be processed in one clock cycle, and several decisions should be made in this cycle. This requirement should be taken into account at the stage of development of the architecture and structure of controllers.

III. The brief overview of the methods for dynamic reconfiguration with ASIC technology

The dynamic reconfiguration can be provided at the technology (close to technological) layer or at higher layers, Fig. 2.

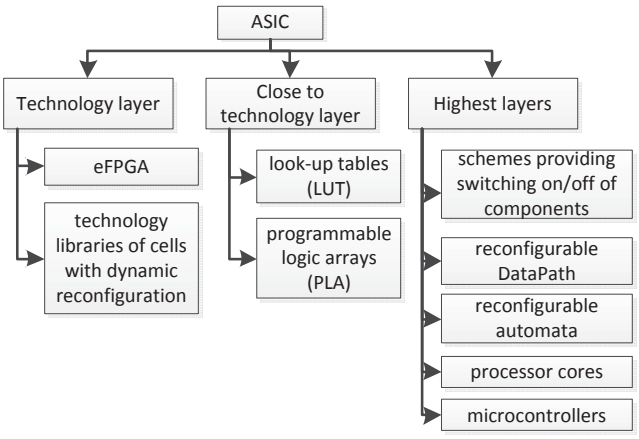

Fig. 2. Typical structure of transport layer unit

The dynamic reconfiguration at the technology layer can be achieved by using:

- The technology libraries of cells with dynamic reconfiguration [6], [7];
- eFPGA[8], [9], [10], [11].

The technology libraries with dynamic reconfiguration includes special cells. Every special cell can perform various functions (e.g. NAND, NOR, NOT) depending of the configuration. The area of such cells is not essentially bigger than area of cells without dynamic configuration. But the area of memory cells that need for store of configuration can be essential. In addition, existing CAD tools do not support synthesis with using of the technology libraries with dynamic configuration. Therefore using of these libraries is significantly limited.

The special library cells with FPGA structure (fragment of FPGA) - eFPGA can be included in design [8,9,10,11]. The eFPGA could include from several hundred to several thousands of slices (logic blocks), memory cells and interconnection structure. Generation of bitstreams for these eFPGA can be performed with the FPGA Design tools. Today eFPGA are widely used. However, the availability of eFPGA for a very limited list of technology libraries and large size (and area) of these cells essentially constrain their use.

At the close to technological layer, the dynamic reconfiguration is achieved by using:

- look-up tables implemented on ASIC library cells (memory blocks or flip-flops);
- programmable logic arrays (PLA) implemented on ASIC library cells (such as AND, OR, NOT).

This kind of structures could be described on HDL as soft IP-blocks. They can also be implemented as firm or hard IP-blocks for decreasing of area and power overheads.

Standard CAD tools can be used for synthesis of designs that includes such blocks. But use of these IP-blocks is limited by

area overheads and achievable working frequency. Typically these IP-block are used for implementation of relatively small components or as part of more complex reconfigurable components. These methods can be used for implementation of dynamic reconfiguration at higher layers.

At the highest layers can be used:

- schemes providing switching on/off of components;
- reconfigurable finite state machine (automata);
- reconfigurable DataPath;
- processor cores (typically RISC);
- microcontrollers.

The schemes, providing switching on/off of components, logically based on multiplexors. The main advantages of this approach are ease realization and possibility of fast switching between modes. When the unused units could be switched off the power dissipation of such scheme is practically equal to power dissipation of one unit. Nevertheless, area of this schema may be too large. Other essential disadvantage of the schema is impossibility to configure it for new modes.

Other way of dynamical reconfiguration providing is using of reconfigurable finite state machine. The next properties of reconfigurable state machine could be changed dynamically: the logical meaning and quantity of the states, the rules of transition between states, the values of output signals, associated with them. The physical realizations of reconfigurable finite state machine may be various. Typical realizations based on Look-up Tables or Programmable Logic Matrices (PLM). Let's briefly consider Look-up Table based realization. The input signals and the current state are feed to inputs of Look-up table. Strings of Look-up table contains the next state and the values of output signals (or values of output signals can be generated by combinational circuit as function of current state and input signals). Number of strings in Look-up table is determined by the formula 2(Ni+Ns), where Ni – total width of input signals, Ns – width of state vector. Therefore, the area of Look-up table grow essentially with growing of Ni or Ns. Developer can use next approaches to reduce area:

- decomposition of initial finite state machine to sub-automates [12];
- additional multiplexing schemes for input signals, that allow us to decrease the number of input signals (Ni) connected to Look-up table in comparison with the number of input signals of finite state machine [13], [14], [15].

When decomposition of initial finite state machine to sub-automates is used, the number of states, input and output signals for every sub-machine will be essentially less than for initial finite state machine. Correspondingly, total area of realization is also essentially less in comparison with initial realization. However, when developer decompose an initial finite state machine to sub-machines achievable degree of configurability (number of possible configurations) may be decreased [16]. This decline is especially significant when the decomposition makes formally. In this case, any new configuration may not corresponds to this decomposition. Therefore, when the structure of the initial finite state machine is permanent for all possible modes, the decomposition to sub-automates should correspond to this structure [17].

The value of Ni is decreased when the additional multiplexing scheme is used for input signals. Decomposition of finite state machine to sub-machines and additional multiplexing schemes could be used together [17].

Other way of realization of reconfigurable state machines based on PLM structures and a register for storing of finite state machine's state. In this case, the area overheads are also main factors that limit the number of inputs and states of finite state machine. Also, eFPGA cells could be used for realization of large reconfigurable automates.

The next state of finite state machine is selected every clock cycle correspondingly to values of input signals and current state (all possible conditions are considered). Therefore reconfigurable state machines well suited for parallel processing of many flows of events in real time. However, state machines do not intended to data flow processing. Therefore, reconfigurable finite state machines are often used in combination with dynamically reconfigurable Datapath.

Dynamically reconfigurable DataPath includes set of functional units (FU) and interconnections between these units. The FUs functions and the structure of interconnections could be dynamically reconfigurable. Dynamically reconfigurable DataPath typically used for data flow processing. It is poorly suited for realization of control logic due to large overheads [18].

The dynamic reconfiguration could be provided by using processor cores. In this case, dynamic reconfiguration is achieved by change software. Several cores, such as Xtensa, ARC [19], [20], [21], [22], [23], [24], [25] and others programmable processors, such as [34] provides additional reconfiguration possibilities. They may include reconfigurable units for implementation of specific data processing.

However, these cores do not suitable for processing of data flow "on the fly". Too high working frequency is required for processing of several event flows in real time due to context switching (it will be 10 - 20 times higher than data flow rate). Thin design rules are required for achieve such working frequency. Often it is not economically feasible. Other problem is high power consumption and dissipation when high working frequency is used. Due to these disadvantages processor cores cannot be used for the Packet distribution unit implementation.

Special microcontroller cores also can be used for realization of dynamic reconfiguration. Microcontrollers often have special components for data flow processing in real time.

Several classes of microcontrollers, such as streaming processors [28], [29], [30] are not suitable for the Transport Layer Controller Unit realization due to too large area. Other classes of microcontrollers, such as [31] are more compact, but too high working frequency is required for processing of several event flows in real time (it will be 10 - 20 times higher than data flow rate). Due to these disadvantages microcontroller cores cannot be used for the Packet distribution unit implementation.

The main advantages and disadvantages of analyzed approaches for realization of reconfigurable packet distribution unit are represented on Table I.

TABLE I. COMPARISON OF APPROACHES TO DYNAMIC RECONFIGURATION IMPLEMENTATION

	libraries of cells with dynamic reconfiguration	eFPGA	schemes providing switching on/off of components	Reconfigurable finite state machine+Datapath	Processor cores	Microcontroller cores
No specific requirement to CAD	-	+-	+	+	+	+
No specific requirements to tech libs	-	-	+	+	+	+
Area, power overheads	+-	-	+	+	-	-
Possibility of data flow processing in real time when working frequency is equal to data rate	+	+	+	+	-	+
Possibility of parallel processing for several event flows, when working frequency is equal to data rate	+	+	+	+	-	-
Possibility of new modes realization	+	+	-	+	+	+

This table show that using of reconfigurable finite state machines in combination with reconfigurable DataPath is the most appropriate approach to realization of Transport Layer Controller Unit.

IV. The Proposed Approach to Development of Transport Layer Controller Unit

We propose an approach to development of Transport Layer Controller Unit, based on dynamically reconfigurable finite state machine and DataPath.

The general structure of proposed dynamically reconfigurable unit is represented in Fig. 3. It includes Control Unit (reconfigurable finite state machine), reconfigurable DataPath, memory subsystem (storage of current configuration, storage for processed data, registers and flags) and Interface Unit (with structure similar to FPGA interconnect).

Several transport layer protocols require to store a large amount of data (e.g. for providing of retransmission when errors in network occur). It is possible to store data in outside memory when proposed approach is used.

The reconfigurable unit has a functional input and output interface and a configuration interface (interface for loading of new configuration). The functional interface includes an interface with network layer (streaming interface) and an interface with application layer (typically, an interface like AHB, AXI, WISHBONE, and an IRQ interface).

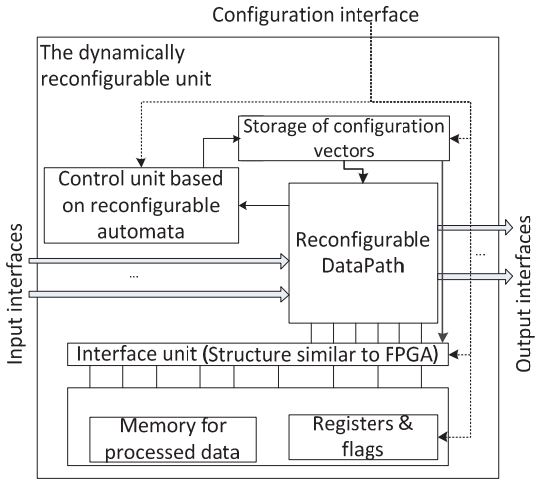

Fig. 3. General structure of the proposed dynamically reconfigurable unit

One configuration corresponds to one operating mode. Configuration (Fig. 4) includes:

- values, that written to Look-up table of reconfigurable finite state machine (automata);
- initial values for DataPath FU's;
- initial values of registers4;
- a set of configuration vectors.

Every Configuration Vector consist of two parts:

- configuration of the DataPath FU's;
- configuration of the Interface Unit.

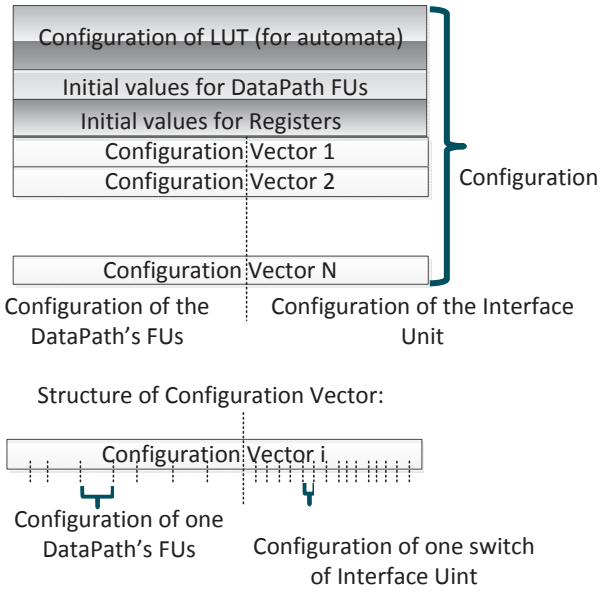

Fig. 4. Structure of configuration

Configuration of the DataPath FU's consists of configuration subvectors for every FUs of DataPath, which may be configured (one subvector corresponds to one FU). Configuration of the Interface Unit consists of subvectors for every switch.

The configuration vector determines the set of actions that are performed with input data words and internal events. Configuration vectors play the role of arithmetic, logic and load/store commands in processors architectures (like instructions in WLIV architectures).

The main function of Control Unit is choice of configuration vector (determining of configuration vector number) that shall be used in next clock cycle. In comparison with processor architectures, this unit implements conditional and unconditional branches. However, as noted above, the use of a finite state machine allows us in one clock cycle to realize the choice of a transition between several branches (more than two). This distinguish our approach from others (e.g. [35]), where implement classical jump with two branches in finite state machine.

The illustration of our approach is represented in Fig. 5. The example of algorithm's part contain two sequential conditions (Fig. 5, a). The first of these conditions may be complex. For example, the header of packet is correct when format of header is correct (Hformat_correct=true), the Address of packet is correct (PAddr_correct=true), the Key of packet is correct (PKey_correct=true), the CRC of header is correct (HCRC_correct=true).

Corresponding assembler code will include several comparison commands and jump commands for implementation of this algorithm. Implementation of this algorithm with state machine is represented in Fig. 5,b. In one clock cycle state machine may pass from Packet_header_CRC_state to one of three states (Deletion_state, Read_state, Write_state).

Special FUs based on PLA structure are used in DataPath to quickly and concurrently calculate values of conditions (they will be considered further).

As noted above, the main limitation when using a reconfigurable state machine is the area limitation. Decomposition to sub automates and multiplexing of inputs may be used for area decreasing.

We have analysed modern transport Layer protocols and its possible realizations. Corresponding structures and quantity of sub automates vary greatly. Therefore using of decomposition approach in this case will not give a tangible gain in area, or very significantly limit the possibilities for reconfiguration.

Consequently, we consider an approach of input multiplexing. During the analysis of modern Transport layer protocols, it was determined that the number of inputs that must be analysed to determine the next state can range from several tens to several hundreds. (Conditions of transitions typically are very complex, different groups of signals are used in different states of state machine).

Due to using special FU based on PLA for calculation of values for inputs of state machine, the maximum possible number of transitions from one state to the next is determined the number of input signals of the state machine. Analysis of the Transport protocol's implementations showed that the number of following states for each possible state does not exceed 10. Therefore, input multiplexing allows us very essentially decreasing quantity of inputs of the reconfigurable automata.

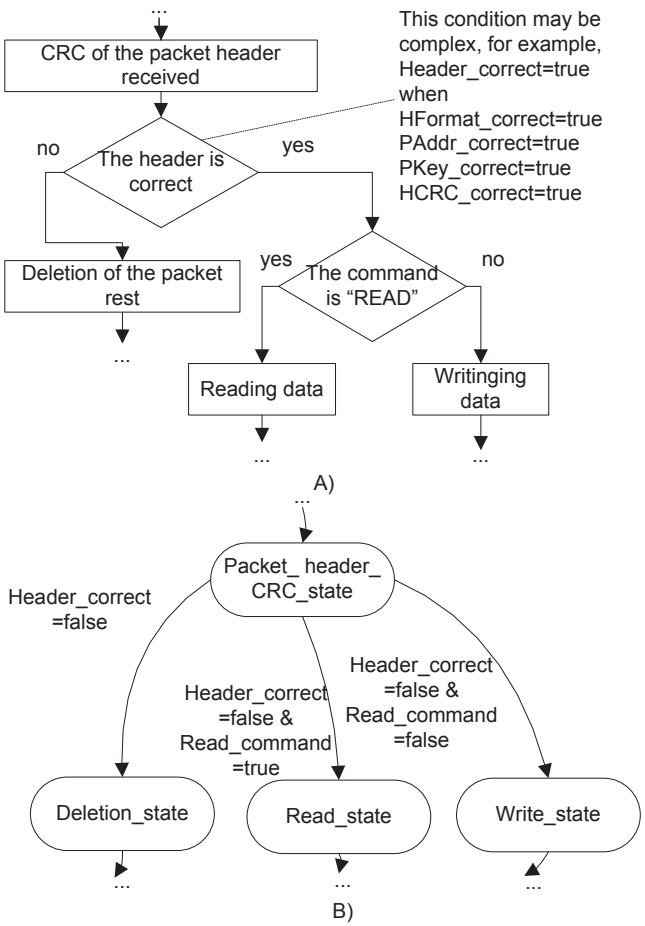

Fig. 5. An example (1) of implementation of transition between several branches

Let's consider these FUs and its using. The FUs with PLA structure allows us to implement functions presented in perfect disjunctive normal form. (Such FUs can be used to implement any function whose number of input variables does not exceed the number of FU's inputs and the number of disjunctions does not exceed the number of library cells with OR logic, includes into the FU.)

Let's consider an example of using such FU's for implementation of the part of algorithm (Fig. 5). In this implementation, we use next flags:

- Flag(0), for which a logic function HFormat_correct is assigned (it is set to one if the Header Format is correct);
- Flag(1), for which a logic function PAddr_correct is assigned (it is set to one if the Address of the packet is correct);
- Flag(2), for which a logic function PKey_correct is assigned (it is set to one if the Key field of the packet header has allowable value);
- Flag(3), for which a logic function HCRC_correct is assigned (it is set to one if the Header CRC is correct);

- Flag(4), for which a logic function Read_command is assigned (it is set to one if the received command is Write Command).

The Packet Header is correct when (Flag(0) = ‘1’) & (Flag(1)=’1’) & (Flag(2)=’1’) & (Flag(3)=’1’).

We denote inputs of the Control unit (state machine) as in(i). For implementation of considered part of algorithm we will use in(0), in(1). When its values are “X0” (in(1) may have any value, in(0)=0) the state machine should pass to Deletion_state. When its values are ‘11’, the state mashine should pass to Read_state. When its values are ‘01’, the state mashine should pass to Write_state.

We use FU with PLA structure for calculation values for these inputs. The next function is implemented with PLA: in(0)= (Flag(0) = ‘1’) & (Flag(1)=’1’) & (Flag(2)=’1’) & (Flag(3)=’1’). We do not need PLA for in(i) calculation (in(1)=Flag(4)).

We use this PLA based FU for implementation of other function in other part of considered algorithm. (This part is represented in Fig. 6). The state machine may pass from the Packet_command_state to Packet_header_CRC_state or to Idle state dependently on the next received data word. If the End of Packet (EP) received, the state machine goes to Reset_state. (EP is not expected symbol here and the received part of packet should be discarded.) If Data symbol is received the state machine pass to Packet_header_CRC_state. If nothing received, the state machine stays in Packet_command_state. The primary inputs and outputs of reconfigurable controller (data_in, valid_in, ready_out) are used for calculation of these conditions.

Nothing_received = (valid_in=’0’) U (ready_out =’0’)

EP_received = Data_in(8) (in this example Data_in(8) is using for signing data or EP word)

In this case in(0)= Nothing_received. Corresponding function is implemented with PLA based FU. In(1) is equal EP_received.

When its values are “X1” (in(1) may have any value, in(0)=1) the state machine stays in Packet_command_state. When values are “00” the state machine goes to state Packet_header_CRC_state. When values are “10” the state machine goes to state Reset_state.

These examples illustrate using the same FU for calculation of different conditions, and illustrate multiplexing of inputs of state machine.

To determine the whole set of FUs that should be included in DataPath, we have analysed the algorithms corresponding to modern transport layer protocols. Despite the fact that the protocols may vary greatly, they include a limited set of actions that are performed during the processing and generation of the packet’s header and body, generation and processing of various internal events (e.g. timeouts). In most cases, this set includes operations of comparison (=, <, >), shifts, arithmetic operations (+,-,*), CRC count, count of different events. Therefore, DataPath should include standard set of components for performing arithmetic and logical operations (e.g. ALUs) and special components (CRC counters, special components for calculation the values of conditions).

As discussed above, in proposed architecture special components for calculation the values of conditions need for forming of input signals (for determining of branch conditions) for Control Unit, also we use these components in combination with counters for count of different events quantity (e.g. timeouts, quantity of data words, quantity of errors and etc.)

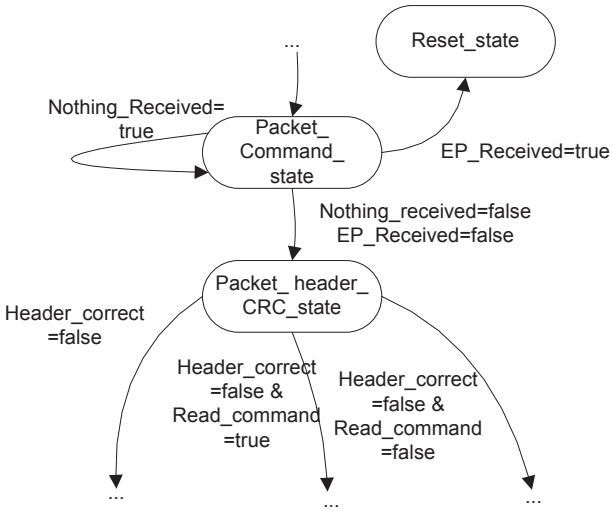

Fig. 6. An example (2) of implementation of transition between several branches

Let’s consider memory subsystem. Different components of this subsystem may be implemented on memory block (special library component) or as arrays of flip-flops. Area and power dissipation for memory blocks is essentially less than for an array of flip-flops with same capacity. Therefore, the size of flip-flops array is strongly constrained. However, memory block typically have only one or two interfaces (accesses to one or two memory cells may be performed in one time). Several (more than two) cells from flip-flop array can be read and written in one time.

Memory subsystem in the dynamically reconfigurable unit includes:

- Look-up tables in reconfigurable finite state machine;
- Storage for configuration vectors;
- Memory for processing data (outside memory can be used);
- Registers (for storage parameters need for packet header analysing) and flags.

The data to Look-up table and to Storage for configuration vectors is written, when new configuration is loaded. One word is read from these memories every clock cycle. Therefore, memory block can be used for it’s realization.

The processed data packets can be quite large (up to several KB, MB). Therefore, for their intermediate storage, it is necessary to use memory blocks. Memory external to the controller may be used.

Registers are used for storage of parameters, that need for packet header and body analysing (e.g. identificators of transport

protocols, addresses, identificators of applications, acceptable number of words in packet body), for internal events control (e. g. timeout values). Also, registers are used for storage different variables (e.g. current packet length). These parameters and variables can be used by different FUs of DataPath concurrently. Therefore they should be implemented as separate flip-flop groups (memory block is not suitable due to restriction to number of words that can be read and write concurrently).

Flags are used for storage a values of conditions. They are realized as separate flip-flops.

The Interface Unit provide connections between the Memory for processed data, the Registers, the Flags and DataPath (FUs). The Interface Unit consists of two parts:

- interconnection system between the flags and the FUs (width of channels – 1 bit);
- interconnection system between the Memory for processed data, the Registers and the FUs (width of channels – 1 word).

For example, the data inputs of comparator FU are connected to interconnection system with one word channels width, the data output is connected to interconnection system with one bit channels width.

The structure of both interconnection systems is same. Each Interconnection system includes some logical channels. The number of logical channels is equal to number of primary ports (ports of reconfigurable controller) of input ports of the registers, the flags and the FU's connected to this system (every input port connected to one logical channel). One logical channel support one connection of corresponding input port with any output port that connected to this interconnection system at one point of time. (One output port may connect to only one input port in same time.) The logical channel in implemented as multiplexor on Register Transfer Layer. The corresponding subvector of Configuration vector is forwarded to the control input of the multiplexor. Physical implementation of multiplexor is implemented by CAD tools correspondingly to used technology library.

The Transport Layer Controller Unit developed with proposed approach can be configured to perform any algorithm corresponds to next constraints:

- Set of functional units included in the DataPath support all functions need for algorithm
- Capacity of configuration vectors storage is enough for set of configuration vectors, Look-up Table size is enough to realization of finite state machine, corresponds to transport protocol
- Quantity of parameters and flags need for packet flow processing is not more than quantity of registers and flags

The area overhead of dynamic reconfigurable Transport protocol controller developed with using of proposed approach depends on area of the Look-up table, the storage of configuration vectors, Interface unit and PLA. (Other components also are used in realization without dynamic reconfiguration and therefore do not affect to overheads.)

V. Examples of reconfigurable Transport Layer Controller unit

We have selected several transport protocols that could be used for similar tasks in different operating modes for demonstration of our approach. These protocols are widely used in aerospace onboard networks.

We initially developed a Reconfigurable Transport Layer Controller Unit that can support the RMAP [30] and the STP [31] transport protocols with proposed approach. (The algorithm of Reconfigurable automata generation is beyond the scope of this paper.) Both these protocols can be used for transmission data from sensors to host system in different modes of data processing. The RMAP protocol is used in query mode (host send to sensors queries when need the data from them). STP protocol is used when the data packets are periodically transmitted from sensors to host.

We have included into the Reconfigurable Transport Layer Controller Unit 15 additional registers, 20 additional flags, and CRC16 FU for providing possibility of other protocols implementation.

Then we have generated the configuration corresponds to the ESDP (STP2) protocol [32] controller for this Reconfigurable Transport Layer Controller Unit. (ESDP protocol differs from the STP protocol in the format of the packet header and the procedure of setting up the transport connection, it includes some additional features for data transmission.)

Further we have generated the configuration corresponds to the STP-ISS 13 protocol [33] controller for the Reconfigurable Transport Layer Controller Unit. (This protocol also is used for data transmission from sensors to the host system.)

After that we have showed that the Reconfigurable Transport Layer Controller Unit includes not enough resources (FUs, registers, flags and memory blocks) for realization of STP_ISS 14 [33]. (This version of protocol is essentially complex than STP_ISS 13.) There were not enough registers, flags, and memory size to store Look-up table and configuration vectors. But we have generated configurations for several profiles of STP_ISS 14 (scheduling and command duplication). Only profile with transport connections can not be implemented.

Let's compare area of proposed Reconfigurable Transport Protocol Controller with area of other realizations:

- separate realizations of considered protocol's controllers (RMAP, STP, ESDP, STP_ISS 13, STP_ISS 14);
- realizations of dynamically reconfigurable controllers with schemes providing switching on/off of these controllers (three variants: RMAP+STP+ESDP, RMAP+STP+ESDP+STP_ISS13, RMAP+STP+ESDP+STP_ISS14).

We used Cadence design tools and different technology libraries (180 nm – 65 nm) for synthesis of these units. Area of units depends on technology library, timing constraints (clock period and others) and specific of implementations. Therefore, we used implementations of RMAP, STP, STP-ISS controllers developed for the same clock frequency. (Frequency is varied

from 125MGz for 180mn to 325MGz for 65 nm. These values were selected in accordance with the existing requirements to bandwidth of controller.)

We have evaluated hardware cost (area) of units and compared results. The ratio of the results obtained for different libraries varies slightly (within 5%). The relative areas of considered controllers are represented in Fig. 7.

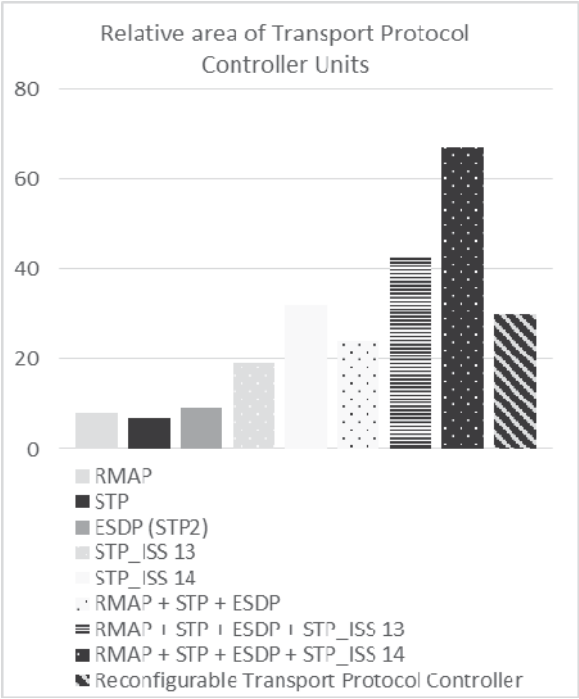

Fig. 7. Relative area of Transport Protocol Controller Units

This chart shows, that area of the Reconfigurable Transport Protocol Controller Unit is essentially bigger, than areas of the RMAP, STP, ESDP (STP2) Controller Units implemented separately. However, the proposed Reconfigurable controller is not essentially bigger, than the based on switching on/off scheme Reconfigurable Controller, that includes the RMAP, STP, ESDP Controller Unit. The area of proposed Reconfigurable controller is essentially less than area of based on switching on/off scheme Reconfigurable Controller, includes the RMAP, STP, ESDP, STP ISS 13 Controller Unit.

This example shows that the area overheads of the Unit, developed with our approach are acceptable, that we can add new configurations of the Reconfigurable controller for implementation of new protocols or new profiles of protocols.

We have compared area of the Reconfigurable Transport Protocol Controller with the area of the STP_ISS 13+ controller (profile of STP_ISS 14 with scheduling and commands duplication, without connections) implemented with Xilinx Virtex 7 (with using Vivado). We select Virtex 7 due to partial reconfiguration is supported for this FPGA. Virtex 7 is realized with 28 nm technology.

We select STP_ISS 13+ controller for comparison due to its area is bigger than area of RMAP, STP, ESDP, STP_ISS 13. Number of LUTs need for implementation of STP_ISS 13+ is enough for implementation any other considered controller. Therefore implementation of the STP_ISS 13+ controller may be replaced by implementation of other controller in SoC realized with Virtex 7.

Area of the Reconfigurable controller (ASIC 180 nm) is comparable with area of the STP_ISS 13+ controller (Virtex 7). But achievable working frequency of the STP_ISS 13+ controller with Virtex 7 is about 75MGz. That is less than achievable working frequency of the Reconfigurable controller (125 MGz). Area of the Reconfigurable controller (ASIC 65 nm) is in dozen times less, than area of the STP_ISS 13+ controller (Virtex 7).

This comparison show that implementation of the Reconfigurable controller with ASIC is more compact and has higher achievable working frequency (and throughput) than implementation of one special controller with FPGA.

VI. Conclusion

We consider existing approaches to realizing of dynamic reconfigurable units with ASIC, their advantages and disadvantages for Transport Protocol Controller Unit correspondingly to specific requirements to this Unit.

We propose an approach for realization of dynamic reconfigurable Transport Protocol Controller Unit, which is based on dynamically reconfiguration state machine (automata) and DataPath.

In frame of this approach, we use units with PLA structure in DataPath for calculation of complex conditions. In combination with using reconfigurable state machine, it allows us to select next state (which depends on several conditions) in one clock cycle that is not possible when processor core is used. This feature is very actual for Transport Protocol Controller Unit since processing of several event flows is required.

Besides, the complex conditions are used for decreasing of finite state machine inputs and for count of events.

The dynamically reconfigurable units developed with proposed approach can be configured to perform any algorithm corresponds to several constraints described in the corresponding part of the paper.

Several examples are considered where presented approach is used, and area overheads are evaluated. It's shown, that area of Reconfigurable Unit developed with our approach are typically less than when set of units without dynamic reconfiguration is used. Also we show that using of the approach allows us to realize new algorithms in existing Reconfigurable controller unit.

Acknowledgment

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation.

REFERENCES

[1] *A*. Azarian, M. Ahmadi, "Reconfigurable Computing Architecture", Survey and introduction 978-1-4244-4520-2/09 IEEE 2009, pp 269 - 274.

[2] M.B Stensgaard, "ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology", in Proceedings of Second ACM/IEEE International Symposium, 7-10 April 2008, pp.55 - 64

[3] E. Cota, Reliability, Availability and Serviceability of Networks-on-Chip / E. Cota, A. de Morais Amory, M. S. Lubaszewski. Springer, 2012, 209 p.

[4] S. Jafri, L. Guang, A. Hemani et al., Energy-aware fault-tolerant network-on-chips for addressing multiple traffic classes. Microprocessors and Microsystems, vol. 37, issue 8, 2013, pp. 811–822.

[5] K. Yoonjin, "Reconfigurable Multi-Array Architecture for Low-Power and High-Speed Embedded Systems", Journal of Semiconductor Technology and Science, Vol.11, N.3, 2011.

[6] I. O'Connor, I. Hassoune, D. Navarro, "Fine-Grain Reconfigurable Logic Cells Based on Double-Gate MOSFETs", IFIP AICT 313, 2010, pp. 97–113

[7] I. Hassoune, I. O'Connor, "Double-Gate MOSFET Based Reconfigurable Cells", Electronics Letters 43(23), 2007 , pp. 1273–1274

[8] Speedcore eFPGA Datasheet (DS003). Achronix Semiconductor Corporation, Web: https://www.achronix.com/documentation/speedcore-efpga-datasheet-ds003

[9] eFPGA IP and electronic diagnostics, Web: http://www.nanoxplore.com/categories/17-efpga.html

[10] Web: http://www.electronics-lab.com/taking-advantage-embedded-fpga-efpga

[11] Web: http://www.menta-efpga.com/efpga-ips.html

[12] V. Sklyarov and I. Skliarova, "Synthesis of parallel hierarchical finite state machines," in Proceedings of the 2013 21st Iranian Conference on Electrical Engineering, ICEE 2013, Iran, May 2013

[13] J. Glaser, M. Damm, J. Haase, and C. Grimm, "TR-FSM: Transition-based Reconfigurable finite state machine," ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 4, no. 3, article no. 23, 2011

[14] A. Karatkevich, Design of Reconfigurable Logic Controllers, vol. 45, springer, Berlin, Germany, 2016

[15] I. Garcia-Vargas and R. Senhadji-Navarro, "Finite state machines with input multiplexing: A performance Study," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 5, pp. 867–871, 2015

[16] S. Gupta, V. Pareek, S. C. Jain, and D. Jain, "Realization of sequential reversible circuit from finite state machine," in Proceedings of the International Computer Science and Engineering Conference, ICSEC 2014, pp. 458–463

[17] V. Salauyou, "Synthesis of high-speed finite state machines in FPGAs by state splitting," in Computer Information Systems and Industrial Management: 15th IFIPTC8 International Conference, CISIM 2016, Springer International Publishing, Vilnius, Lithuania, 2016, pp. 741–751

[18] S. Xydis, G. Economakos, D. Soudris, and K. Pekmestzi, "High Performance and area efficient flexible DSP data path synthesis", IEEE Trans. Very Large Scale Integr.(VLSI) Syst., vol. 19, no. 3, pp.429-442

[19] "The What, Why and How of Configurable Processors", Tensilica's White Paper. 2008. Web: https://ip.cadence.com/uploads/902/TIP_What_Why_How_Cust_Processors_WP_V3_FINAL-pdf

[20] Xtensa Instruction Set Architecture (ISA) Reference Manual. Tensilica, Inc., 2007

[21] Xtenta LX2 Microprocessor Data Book. Tensilica, Inc., 2008

[22] Xtensa LX7 Processor. Tensilica Datasheet, 2016, 13 pp.

[23] ARChitect Processor Configurator. Arc.com. Retrieved 2014-03-02.

[24] ARC Processor Core. Fujitsu Microelectronics America, Inc., 2016

[25] U. Kapasi, S. Rixner, W. Dally et al. "Programmable stream processors," IEEE Computer, 2003

[26] S.Venkatasubramanian, "The graphics card as a stream computer," SIGMOD DIMACS, 2003

[27] SHARC® Processor Programming Reference. Revision 2.4, 2013

[28] PSoC® 4: PSoC 4200 Family Datasheet. Cypress Semiconductor Corporation. Document Number: 001-87197 Rev. J. Web: https://www.cypress.com/documentation/datasheets/psoc-4-psoc-4200-family-datasheet-programmable-system-chip-0

[29] S. N. Pradhan, P. Choudhury, "Low power and high testable Finite State Machine synthesis," in Proceedings of the International Conference and Workshop on Computing and Communication, IEMCON 2015, Canada, 2015, pp. 1–5

[30] ECSS-E-ST-50-52C SpaceWire - Remote memory access protocol, 2010

[31] Y. Sheynin, E. Suvorova, F. Schutenko, V. Goussev, "Streaming Transport Protocols for SpaceWire Networks", International SpaceWire Conference, 2010

[32] A. Khakhulin, I. Orlovsky, Y. Sheynin, E. Suvorova, I Korobkov, V Olenev, I. Lavrovskaya, "Real Time Video Data Transmission in SpaceFibre Networks with the ESDP Transport Protocol", SpaceWire Conference, 2016

[33] V. Olenev, I. Lavrovskaya, Y. Sheynin, I. Korobkov, E. Suvorova, E. Podgornova, D. Dymov, S. Kochura "STP-ISS Transport Protocol for SpaceWire On-Board Networks: Development and Evolution", International Journal of Embedded and Real-Time Communication Systems, №5(4). IGI Global, 2014, pp. 45-76

[34] M. Safarpour, O. Hautala Silvén, "An Embedded Programmable Processor for Compressive Sensing Applications", NorCAS, 2018

[35] E. Hrynkiewicz, M. Mirosław Chmiel, "Programmable Logic Controller - Basic Structure and Idea of Programming" PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review), ISSN 0033-2097, R. 88 NR 11b, 2012, pp. 98-101

[36] E. Suvorova, V. Rozanov "Dynamic Reconfigurable Packet Distribution Unit for Embedded Systems", WECONF 2019, 8 p.