RISC V Based Reconfigurable Manager for Event Transmission in SpaceFibre Networks

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

***Abstract*—In many aerospace networks based on the SpaceFibre standard, there is a need to transfer information about events occurring in various network nodes (terminal nodes, routers) for processing to other nodes. For example, information about faults during operation should be transmitted to a node performing system administration functions. In different networks, the set of events, information about which must be transmitted, the level of criticality of events for the functioning of the system, may be different. The required transmission characteristics may also vary. The SpaceFibre standard includes several mechanisms for transferring information about events from sources (terminal nodes and routers) to handler nodes. Different characteristics can be achieved (guaranteed delivery, guaranteed delivery time, the ability to transfer additional parameters, network load) with using different mechanisms. When designing a specific network, the system developer should be able to determine for each terminal node and router which events should be transmitted for processing to remote nodes and using which mechanisms this should be done. Terminal nodes and routers are developed as universal devices (systems-on-a-chip), which should provide the ability to send information about various events using various mechanisms. However, there are usually limits on the time that elapses between the occurrence of an event and sending information about it. Implementation overheads (area) are generally tightly constrained. The article proposes approach to designing an reconfigurable Event transmitting manager based on the RISC V ISA. For the proposed variants of implementation, the achievable time characteristics and overhead costs are evaluated.**

I. INTRODUCTION

In terminal nodes and routers of aerospace networks based on the SpaceFibre standard [1], various events can occur, information about which must be transmitted to other nodes for processing. Faults, information about which must be transmitted to the system administrator node are one example of such events. Other example are events generated by various sensors, measuring equipment, information about which must be transmitted to remote handler nodes.

The Fig. 1 shows example of typical structures of a terminal node intended for connecting external devices and Fig. 2 shows example of typical structures of a router. In these and follow figures, units labelled with "S" correspond to AXI slave interfaces; units labelled with "M" correspond to AXI master interfaces. In these figures, units that may be sources of events are marked with a dark gray background.

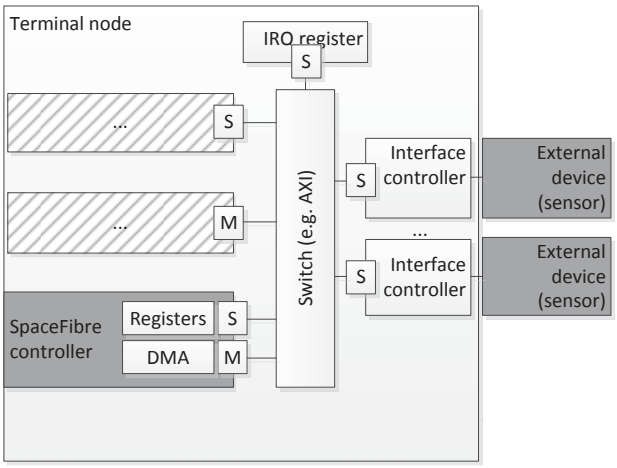


Fig. 1. Example of Terminal Node structure

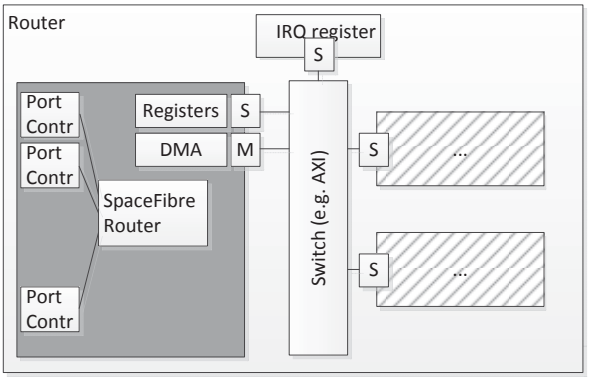


Fig. 2. Example of Router structure

Potentially possible set of events that can occur inside a terminal node, inside a router, is determined during its development. Typically, it includes events defined in the Management Layer specification of the SpaceFibre standard. This set includes connection and disconnection events on SpaceFibre ports, credit errors, errors of accessing non-existent virtual channels, events of expiration of timeouts for receiving/transmitting data, etc. In specific implementations, this set can be supplemented, for example, by events for the expiration of arbitration timeouts, events for receiving packets whose length exceeds the allowed length, etc.

ISSN 2305-7254

Possible set of events, the source of which may be external devices, is not rigidly predetermined at the design stage of the terminal node. When developing a terminal node, a set of external interfaces is defined that are used to connect external devices. Data transmission standards for these interfaces may be defined in whole or in part. But, at the same time, there may still be a large variability in the use of interfaces, determined by their specific settings during operation.

For example, GPIO is often used as an interface for connecting external devices. The roles of the external contacts of this interface can be changed during operation. Information received through external contacts can have different meanings and be interpreted in different ways.

Also, such interfaces as SPI, I2C, and I3C can be used to connect external devices. The purpose (roles) of the external contacts of these interfaces can not change, but the data received by these contacts can be interpreted in different ways.

There are several mechanisms in the SpaceFibre standard that can be used to transmit information about such events [1],[2].

The first mechanism is transmission using SpaceWire control codes (time markers, distributed interrupt codes). These codes are broadcast from the source to all network nodes. Codes have the highest priority. During their transmission in the data link, the transmission of other data objects (Broadcast and data packets) is temporarily suspended. As a result, a minimum delivery time and a minimum delivery time jitter is achieved compared to using other mechanisms. SpaceWire control codes can be transmitted over the network in situations where the transmission of data packets is not possible, due to faults in the network (for example, the occurrence of a Deadlock). If there is at least one living path between the source and the receiver (handler) of an event in the network, then the SpaceWire control code will be delivered through it. However, these codes are very short, they do not contain fields that allow transferring any additional information about events, any additional parameters. In addition, there can be no more than 32 sources of such codes in one network. (The size of the code identifier field is 5 bits, which allows us to use up to 32 identifiers. Each code (code with a single identifier) can have only one source.)

The second mechanism is transmission using Broadcast messages. These messages, as well as SpaceWire control codes, are broadcast from the source to all network nodes. The priority of these messages is lower than the priority of the SpaceWire control codes, but higher than that of the data packets. At the time of their transmission in the data link, transmission of data packets is temporarily suspended. The minimum delivery time and delivery time jitter for Broadcast Messages is slightly higher than for SpaceWire control codes, however, significantly lower than for data packets. Just like SpaceWire control codes, Broadcast Messages can be transmitted over the network in situations where the transmission of data packets is not possible. If there is at least one living path between the source and the receiver (handler) of the event in the network, then the Broadcast message will be delivered via it. Broadcast messages include a data field with a length of 8 bytes, which allows to transfer additional information, additional event parameters. One network cannot have more than 256 Broadcast Messages sources.

The third mechanism is transmission using data packets with one of the transport protocols, for example, RMAP, STP-ISS, ESDP. For data packets, guaranteed delivery times can be provided if required (using the scheduling mechanisms supported in SpaceFibre). However, in this case, the delivery time and delivery time jitter will be significantly higher than for the SpaceWire control codes and Broadcasts. If in the network transmission of data is blocked due to failures (deadlocks), then information about the event may not be delivered to the receiver. The length of the data field of packets can be significantly longer than the length of the data field of Broadcast Messages, so they are well suited in cases where you need to transfer a large number of parameters associated with an event. The number of sources of data packets in the SpaceFibre network is not limited.

Some of these mechanisms focused on meeting the requirements of hard real time. These mechanisms provide guaranteed delivery time with very low jitter. Information reaches the handler if at least one healthy transmission path exists between the source and the destination. But they do not allow the transmission of additional information about events. With using other mechanisms, the transmission time and the transmission time jitter will be significantly longer, there is no guaranteed delivery. However, these mechanisms allow transferring additional parameters characterizing the event.

Networks based on the SpaceFibre standard can vary significantly in their purpose and size.

They can be used to build small systems that include 5 to 10 terminal nodes and 2 to 4 routers and to build large systems that include several hundred terminal nodes and routers.

Various equipment can be connected to terminal nodes. This equipment can be source of various events, the criticality of which for the functioning of the system as a whole can be different. The criticality of the functioning of the terminal nodes and routers themselves (the criticality of faults that occur during their operation) in different networks, in different parts of the same network, can vary significantly. In accordance with this, the system developer should be able to define for each terminal node and router information about what events should be sent for processing to remote nodes and with the use of what mechanisms this should be done.

For example, in one system, a sensor (connected to a terminal node) monitors the value of a parameter that is very important for the functioning of the system as a whole. The system developer chooses a distributed interrupt to transmit event information from this sensor. In this case, this interrupt will be transmitted only if the parameter value goes beyond the permissible range. No additional parameters need to be passed.

In another system, a sensor (connected to the same terminal node) measures the value of a parameter that is not so critical for the functioning of the system as a whole. For processing, the measured value and the timestamp corresponding to the moment of measurement must be transmitted to the remote

terminal node. Measurements are carried out quite often, with an interval of 0.5 μs, and there are several hundred terminal nodes in the system with the same sensors. In this case, the system developer can choose one of the transport protocols, for example, RMAP or ESDP, to transfer information about events.

The developer should be able to define the priority levels of events. (Information about high-priority events should be sent first if multiple events occur near the same time.)

Terminal nodes and routers are designed as universal devices (systems-on-a-chip) that must provide the necessary reconfigurability - the ability to send information about various events using different mechanisms. At the same time, the requirements for the time between the occurrence of an event and sending information about it must be met within the node, on the one hand, on the other hand, the overhead costs for implementing mechanisms are usually strictly limited.

As a rule, for a network designer, the most important time parameters are the (maximum) time between the moment the event occurs and the arrival of information about this event in the handler and the jitter of this time (for hard real-time systems). The time between the occurrence of an event in the terminal node (router) and the transmitting of information about it to the network and the jitter of this time are integral parts of these parameters. For different systems, the permissible values of these parameters can vary widely. For some of them it can be milliseconds, tens of milliseconds, for others - microseconds.

The overhead of implementing mechanisms for transmitting information about events is usually very limited. In most cases, it should not exceed 5-10% of the terminal nod (router) total area .

In our implementations, we focus on the use of ASIC technology with design rules from 180 nm to 65 nm. (These design rules are available or will be available in the near future for use in Russian technology factories.) The proposed Event transmitting manager is focused on operating at a frequency of 62.5-250 MHz (depending on the selected design rules). Such operating frequencies are typically used in the nodes of the SpaceFibre network with physical channel rates of 1.25Gbps, 2.5Gbps, 3.125Gbps.

In this paper, we propose an implementation of the Event transmitting Manager based on the RISC core. The use of the RISC core provides very wide opportunities for dynamic reconfiguration due to the possibility of complete or partial software replacement. We chose RISC V ISA [3],[4] for our implementation.

This architecture has the following features, which are significant advantages for the considered application. The license (unlike, for example, ARM [5]) allows free use of the architecture and modification within a wide range. This allows us to optimize the RISC core in terms of area and power consumption in accordance with the set of algorithms/tasks for which it is being developed. Also freely distributed is a toolkit for testing and verifying processor cores developed on the basis of this architecture. RISC V ISA provides the ability to optimize the length and format of instructions, which allows us to optimize the size of command memory, which is very important for the implementation of *Event Transmitting Manager*.

The base set of registers defined in this architecture can be supplemented with optional registers (CSR). The standard defines a number of registers that can be used as additional registers. For example, these include the register in which the system time is displayed. Other additional registers can be defined according to the specific application.

An additional set of commands is defined for working with CSR registers. This instruction set is focused on working with the mode and status registers used to monitor the state and control various automata, which is necessary when implementing microcontrollers.

In RISC V ISA architecture (similar to the Extensa architecture [6]), it is possible to add new instructions that allow us to implement specific functionality necessary for data processing.

The paper is organized as follows. In section 2 we describe functions of the *Event Transmitting Manager*. Section 3 is about proposed implementations of the *Event Transmitting Manager*. In section 4 we evaluate and compare characteristics of proposed variants. Section 6 concludes the paper.

II. Event Transmitting Manager functions

Let's consider the functions that the *Event Transmitting Manager* should perform. Functions can be divided into two groups:

- functions for monitoring events that occur, determining the order of sending information about them;

- functions for generating and transmitting information about the selected event that occurred.

Potentially, the software for the *Event Transmitting Manager* can be implemented based on the real-time Operating System (RTOS) kernel. Actions to monitor ongoing events, determine the order of transmitting information about them can be performed by the RTOS kernel, and actions to generate and send information about an event that have occurred can be implemented as user processes(or threads) - event handlers. However, an OS-based implementation requires a fairly large amount of memory. The real-time OS kernel can occupy 20-30 Kbytes [7],[8],[9]. The area of such memory can be ten times larger than the area of a terminal node that includes 2 SpaceFibre ports and 1.5-2 times larger than the area of a SpaceFibre router with 8-10 ports.

The reaction of the RTOS kernel (the delay between the occurrence of an event and the call of the corresponding handler process/thread) at the considered frequencies of the RISC kernel can be more than ten μs. The jitter of this delay will also be more than ten μs, since potentially the process/thread can be called almost immediately after the occurrence of the event [7],[8],[9],[10],[11],[12]. This reaction delay and jitter can also exceed limits defined by system developer.

Therefore, the software for the *Event Transmitting Manager* was developed without using the OS as bare metal.

The software consists of a scheduler (monitors the occurrence of events and launches the corresponding handlers in the order corresponding to the priority of events) and event handlers (transmit information about events).

It could be necessary to interrupt the execution of a low priority event handler to handle a higher priority event. (It makes sense to perform these actions only if the execution time of the event handler significantly exceeds the time of switching between the event handler and the scheduler.) However, in this paper, we will not consider this mechanism, its implement tation; it is the subject of further research.

Let's consider the algorithm of the scheduler and its implementation. Fig. 3 shows a diagram of the algorithm (without interruption the execution of event handlers).

Terminal nodes and routers usually include an interrupt register (IRQ_REG), which represents information about all events that are controlled inside the terminal node (router). Also, the interrupt register represents information about the external events, which sources are devices connected to terminal node. The scheduler reads this register, and then looks for the set flags corresponding to events, information about which should be sent to remote nodes for processing. Looking is carried out in the order corresponding to the priority of events, defined by the system developer. If all flags are checked and none of them is set, the register is read again and the actions are repeated. If the flag is set, then the corresponding event handler is launched. After the event handler returns control to the scheduler, the IRQ_REG is read again and scanning begins with the event for which the highest priority is determined. (This provides faster response times and less jitter in response times to higher priority events.)

Let's consider typical algorithms for generating and transmitting information about an event.

A typical algorithm for the implementation of the first mechanism (transmission using SpaceWire control codes) is shown in Fig. 4. At the beginning, it waits for the SpaceFibre controller of the terminal node or router to be ready to send the next control code. The ready indication bit according to the SpaceFibre Management Layer specification is mapped to the SpaceFibre Controller/Router Status Register (SpFi_STATUS_REG). Next, the number of the SpaceWire code to be sent is written into the SpaceWire sent code register (SpFi_SendCCode_REG) of the SpaceFibre controller/Router. Writing to this register automatically initiates transmitting the code to the network.

A typical algorithm for the implementation of the second mechanism (transmission using Broadcast message) is shown in Fig. 5. At the beginning, the Broadcast Message parameters are generated. (Its values are written to the RISC core general purpose registers.) For example, the current time value can be fixed. (The Time_CSR value is written to one of the general-purpose registers.)

The parameters can be read from the SpaceFibre controller/Router registers or from the Interface Controller registers. The values from the registers can be used in their original form or some transformations can be performed on them, for example, sampling of individual bits.

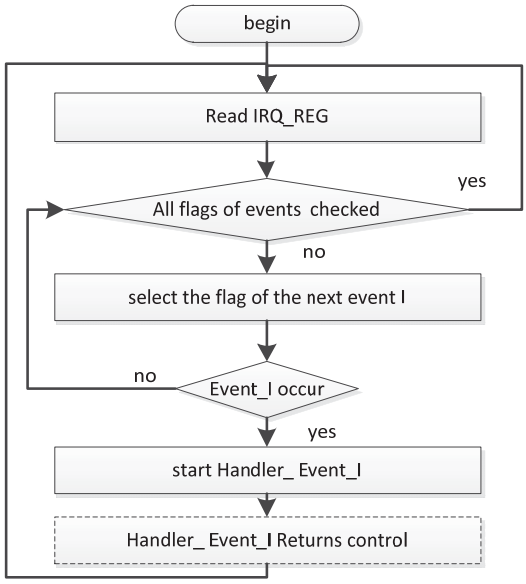


Fig. 3. Typical algorithm of scheduler

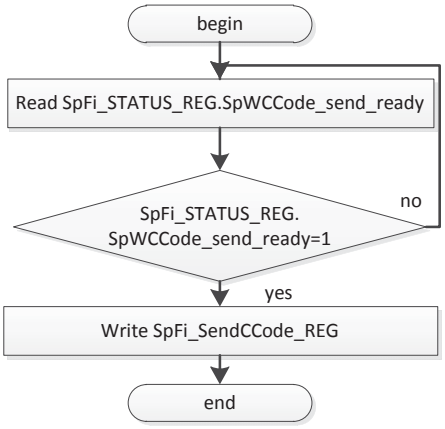


Fig. 4. Typical algorithm for transmission using SpaceWire control codes

Next, it waits for the SpaceFibre controller of the terminal node or the Router to be ready to send the next Broadcast Message. The ready indication bit according to the SpaceFibre Management Layer specification is mapped to the SpaceFibre Controller/Router Status Register (SpFi_STATUS_REG). Next, the parameters of the sent Broadcast Message must be written to the registers of the controller/Router. (Parameters transmitted in data field are written to SpFi_SendBroadcastData0_REG, SpFi_SendBroadcastData1_REG. Parameters transmitted in Header are written to SpFi_SendBroadcastHead_REG. Writing to the SpFi_SendBroadcastHead_REG automatically initiates the sending of the Broadcast Message to the network, so it should be done last.

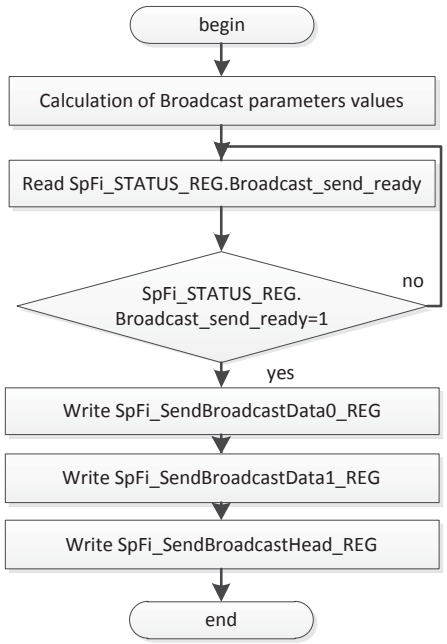


Fig. 5. Typical algorithm for transmission using Broadcast messages

When using the second scheme, the packet is written word by word into the data register for sending to the network of the SpaceFibre controller (SpFi_Send_Data_REG). The SpFi controller/router sends these data words to the SpaceFibre network. The next word is written to the register only after the previous one has been sent.

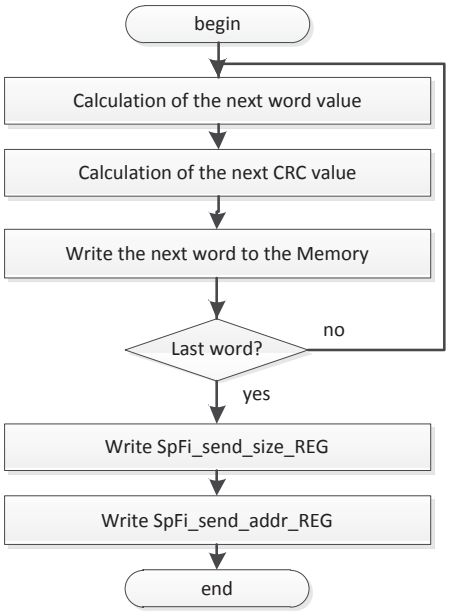


Fig. 6. Typical algorithm for transmission using Data packets (the first scheme)

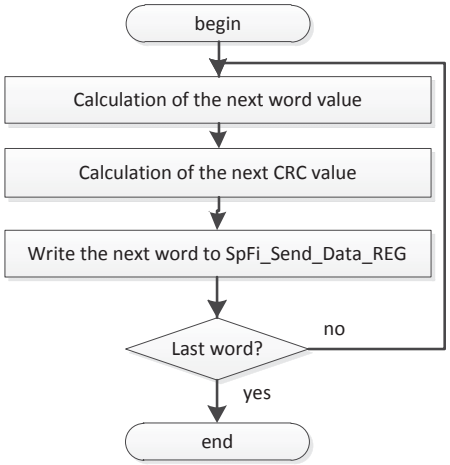


Fig. 7. Typical algorithm for transmission using Data packets (the second scheme)

III. Proposed Implementations of Event Transmiting Mananger

The place of the *Event Transmitting Manager* in the structure of the terminal node is shown in Fig. 8, in the structure of the router - in Fig. 9. *Event Transmitting Manager* has a direct connection with the memory in which its software is stored. This provides a strictly guaranteed and short time to retrieve the next instruction from memory, which is necessary to ensure predictable (and low) jitter of the execution time.

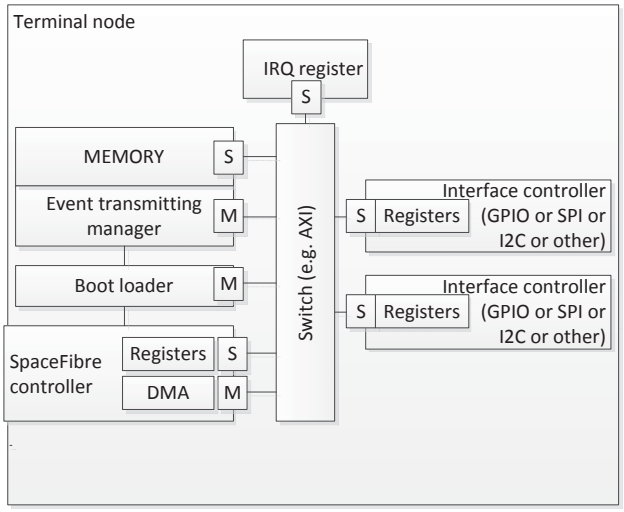


Fig. 8. Place of Event Transmitting Manager in structure of Terminal node

It is possible to boot the *Event Transmitting Manager* software from external memory connected directly to the terminal node or router. The initial boot is also possible by writing the program to the RAM of the *Event Transmitting Manager* by a remote device over the network. (In the current implementation, the program is transmitted using RMAP transport protocol packets over the SpaceFibre network.) The initial loading is done under the control of the initial loading automata. This automata disables the *Event Transmitting Manager* during the booting of software and allows it after this process is successfully completed.

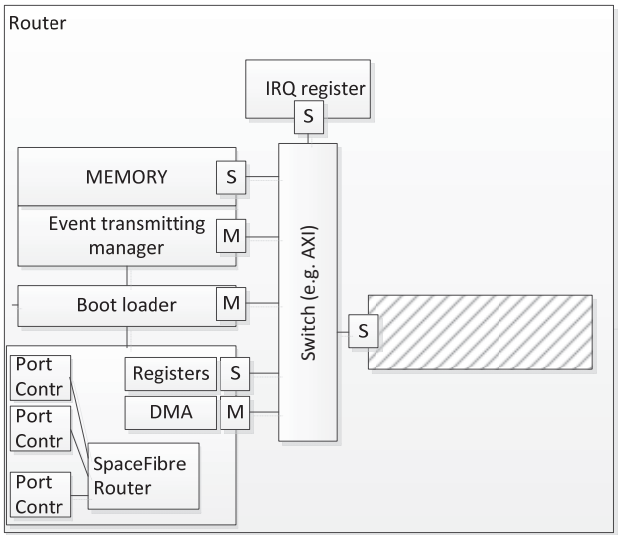


Fig. 9. Place of Event Transmitting manager in structure of Router

We propose several variants of the *Event Transmitting Manager* implementation.

A. The first proposed variant

In the first variant, all actions are performed in software on the processor core. The processor core supports a set of logical operations, shift operations, a set of arithmetic operations, a set of conditional and unconditional jump operations defined in RISC V ISA. The current implementation does not support debugging functions, since the software being developed is very small (several dozen commands), and its debugging can be performed on the RTL model.

The System time CSR register is included in the set of registers, since in most cases the time marker is one of the parameters of the event.

The processor core for this variant has a pipeline structure. It is shown in Fig. 10.

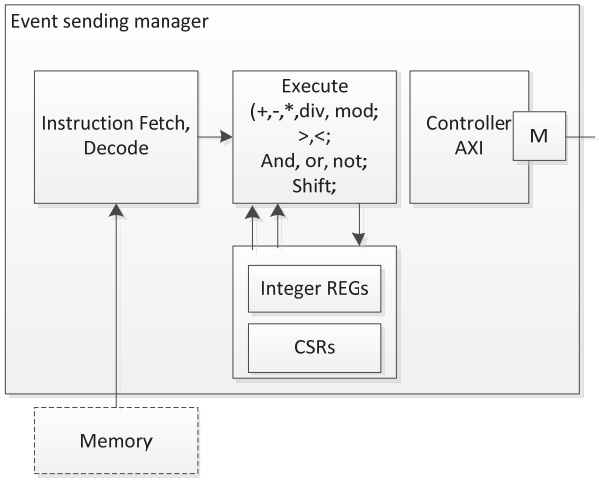


Fig. 10. Event Transmitting Manager structure

The first stage of the pipeline fetches the next instruction from memory and decode it. These actions in our implementation are carried out at one stage (used technological libraries make it possible to ensure this when using the selected operating frequencies).

At the second stage of the pipeline, arithmetic-logical operations are performed and the result is written to the registers that belongs to the *Event Transmitting Manager* register set. Time to access external registers and memory is 2 clock cycles in our implementation (this time can be vary for different implementations of terminal nodes and routers).

The main timing parameters of proposed implementation are represented on table I.

TABLE I. THE MAIN TIMING PARAMETERS OF THE EVENT TRANSMITTING MANAGER IMPLEMENTATION

	Time (clock cycles)	**Constant|variable**
Access to instruction memory	1	constant value
Access to internal registers (general purpose, CSRs)	1	constant value
Access to external registers, to external memory	1 - N	Variable value (depends on the implementation, on the competition for the resources of the communication system)
Performing arithmetic and logical operations, calculating of jump address	1 2(mul) 3(div, mod)	constant value

Most actions take deterministic time, they are a source of latency, but not a source of jitter. The source of jitter can only be access to external memory and external registers.

For this (and other) proposed implementations, we estimated the minimum time between the occurrence of an event and the transmitting of information about it, and an estimate of the maximum time between the occurrence of a high-priority event and the invocation of the event handler. (These parameters allow us to evaluate the best and worst characteristics of the proposed implementation).

The minimum time between the occurrence of an event and the transmitting of information about it (Trmin) can be estimated using the following formula:

$$Tr\min = Tos\min + Th\min \quad (1)$$

where Tosmin – minimum time required for the scheduler to detect the event and call the handler;

Th min – minimum handler execution time

The maximum time between the occurrence of a high-priority event and the call of the event handler (Trhmax) can be estimated using the following formula:

$$Trh\max = Tos\max + \max_{i=0}^{H-1} Th\max \quad (2)$$

Where

Thmax is the maximum execution time of the event handler (among all available handlers, the handler with the maximum value of this time is selected).

Tosmax – maximum execution time of actions by the scheduler

To perform numerical estimates of these parameters, we implemented a scheduler and several variants of handlers. We chose variants that include very few instructions (the simplest possible algorithms) and variants that include a relatively large number of instructions (instructions with a fixed execution time and with a variable execution time). When performing the estimates, we considered systems with different latency when accessing external registers, external memory. This allows us to monitor the main trends, functions and actions that are the source of the main delays.

The results of the evaluations are shown in Figure 12 (the points on the graphs corresponding to variant 1).

In the worst case scenario, the time between the occurrence of a high-priority event and the start of the high-priority event handler will be 820 clock cycles. At a local frequency of 125 MHz (the duration of the period is 8 ns), this will be about 6.6 microseconds.

This time is comparable to the time of transmission of the SpaceWire control code and Broadcast message through the network (5 – 10 transit routers). For a number of tasks, such a delay in the terminal node may be unacceptable.

Based on the obtained estimates of time characteristics, we have identified the functions and actions that are the main sources of delays.

First of all, saving and restoring registers values when switching between the scheduler and the handler has a large delay. For the examples we have considered, these actions take longer than the execution time of the handler.

The next most important source of delays is the function of calculating CRC8 and CRC16 (CRCs are used in all data transport protocols). The execution time of these actions is the majority of the execution time of the handlers that form the data packets.

If the handlers perform a large number of accesses to external registers and external memory, then these actions are also a source of rather large delays. These actions are also a source of jitter.

Next, we propose implementations in which these actions are performed in hardware.

An estimate of the area for the implementation of the first variant is shown in Fig. 13, 14.

B. The second proposed variant

In the second variant, saving the register values and loading the saved register values (necessary for context switching) are implemented entirely in hardware. To implement these actions, K “shadow” register sets have been added to the processor core, which allow storing K contexts. The K value at the RTL model level is set parametrically. The RISC core (Execute stage) works with registers from the main set, (it has a number 0). The values of the registers of the main set can be overwritten in any of the K shadow sets. The values of any of the K shadow sets can be overwritten into the main set. The rewriting actions are performed by hardware-implemented automata. The execution time is 1 clock cycle.

One instruction with two parameters has been added to the instruction set. The first parameter of the instruction is the number of the register set from which the values will be overwritten. The second parameter is a set of registers to which the values will be overwritten.

The implementation of this variant is represented in Fig. 12. A gray background marks the additional unit.

The timing characteristics obtained for this variant are shown in Fig. 13, estimate of overhead costs by area - in Fig. 14,15.

Using this variant significantly reduces the time between the occurrence of a high-priority event and the launch of the high-priority event handler by about 2 times compared to the first variant.

However, the area of this option is noticeably larger. It increased by 25-40% compared to the first option. (Depends mainly on the number of shadow register sets.)

C. The third proposed variant

In this variant, the units for hardware counting of CRC8 and CRC16 added. (These CRCs are used in the transport protocols used in the SpaceFibre network.) We included these units to the Execute pipeline stage. CRC counting, as a rule, is performed for a certain sequence of words (sequence of words included in the packet header; packet data fields). Therefore, we added automata that track the moments of the generation of a new word for which the CRC must be calculated. This automata can track writing to external memory, to an external register (for example, to SpFi_Send_Data_REG) or to one of the internal registers of the *Event Transmitting Manager*. (The programmer can choose one of these options depending on the selected data transmission algorithm.)

The next values of CRC8 and CRC16 for data words are calculated in 1 clock cycle. The CSR registers for CRC value CSR_CRC8, CSR_CRC16) and for tracked address (CSR_CRC8_ADDR, CSR_CRC16_ADDR) have been added to the register set of the *Event Transmitting Manager*.

We added CRC control instruction to the instruction set. The first parameter of the instruction specifies the register number from which the next data word is taken to calculate the CRC or a sign that it is necessary to track memory accesses. The value 0 is used as such a sign (in the RISC V architecture, the register with this address 0 is used to store the constant 0).

The second parameter contains the command code. It can take four values:

- starting CRC counting without incrementing the controlled address (applicable if the word for counting the CRC is located in the internal register, external register);

- starting the CRC counting with the increment of the monitored address (applicable if the word for CRC counting is located in memory);

- stop the CRC count;

- reset the CRC count.

The third parameter allows us to set the type of CRC for which the control is performed (it can take 2 values, corresponding to CRC8 and CRC16).

The initial value for the CRC counting should be written to the CSR_CRC8 (CSR_CRC16) before starting the CRC counting. If programmer need to track values written to external memory, he must also write corresponding address to CSR_CRC8_ADDR (CSR_CRC16_ADDR).

Further, after starting the CRC count, every time a value is written to the register (memory cell) specified in the CRC control instruction, the current CRC value is counted. The fact of a (new) value writing is tracked by hardware. The next CRC value is available in CSR_CRC8 (CSR_CRC16) on the next clock cycle. Then it is stored in this register until a new (next) result is received or reset. The CRC count control instruction with the "stop the CRC count" command allows us to stop the CRC counting without resetting to save the current CRC value in the corresponding CSR register for future use.

The implementation of this variant is represented in Fig. 12. A striped background marks the additional units.

The timing characteristics obtained for this variant are shown in Fig. 13, estimate of overhead costs by area - in Fig. 14, 15.

When using this variant, the execution time of handlers that form data packets was reduced by about 2 times compared to the first and second variants. Since the execution time of these handlers is the longest, Trhmax was reduced by 2 times compared to the second variant (and 4 times compared to the first variant).

At the same time, hardware implementation costs increased very slightly, by about 10%.

D. The fourt proposed variant

In the fourth variant, the mode and status registers of SpaceFibre controllers and other units, that the *Event Transmitting Manager* can interact with are mapped to CSR registers. This allows us to reduce the access time to these registers. (It also allows us to eliminate the jitter of the access time to the registers in cases where there are other units in the SoC that can access these register units through the communication system.)

An example of such mapping for a terminal node is shown in Fig. 11.

For a router, the mapping is performed in a similar way.

This allows us to reduce the execution time of all actions by reducing the time of accessing registers. For routers and terminal nodes that have other units that can access these registers, this also reduces the jitter of access time due to waiting for communication system resources.

Fig. 12 shows a modified structure of processor core with additional components that allow us to implement the second, the third, and the fourth variants. Additional components are highlighted in bold outline.

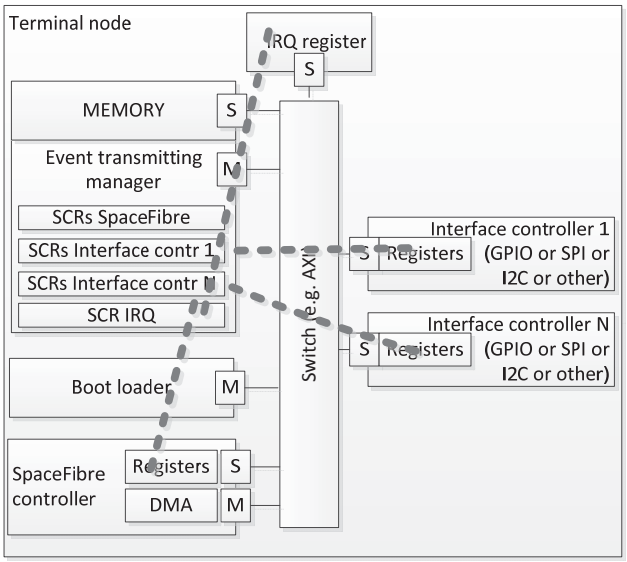


Fig. 11. Example of registers mapping for Terminal Node structure

The component with a gray background is used to implement the third variant. The component with a striped background is used to implement the additional functionality of the third variant. The component with a gray striped background is used to implement the additional functionality of option four.

For the examples we have considered, the gain was not very significant.

This variant is advisable to use for cases when the fraction of accesses to the external registers and memory significantly exceeds the fraction of rest of data processing instructions.

IV. EVALUATION OF CHARACTERISTICS FOR PROPOSED VARIANTS

Fig. 13 shows graphs of time characteristics (the minimum time between the occurrence of an event and the transmitting of information about it (Trmin) for handlers, that generate and transmit SpaceWire control codes, for handlers, that generate and transmit Broadcast messages and for handlers, that generate and transmit data packets; the maximum time between the occurrence of a high-priority event and the call of the event handler (Trhmax)). We evaluated these parameters for two values of access time to external registers and memory (2 clock cycles and 6 clock cycles). The graphs that marked with name "Best and worst processing time (access to external regs, mem=2)" (represented on the left part of figure) correspond to access time equal two clock cycles. The graphs represented on the right part of figure correspond to access time equal six clock cycles.

Each group of graphs has one main ordinate axis, on which the time is represented in number of clock cycles, and three additional ones, on which the time is represented in nanoseconds (correspond to the clock frequencies of 62.5 MHz, 125 MHz, 250 MHz).

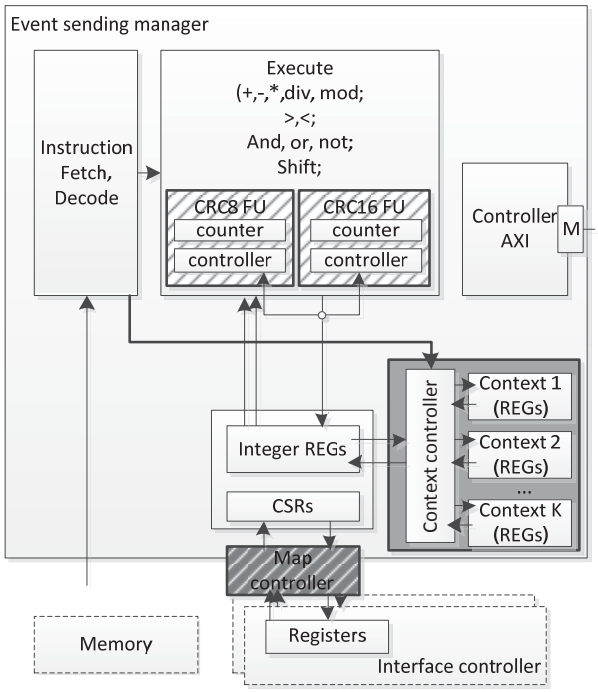


Fig. 12. Event Transmitting Manager structure (implementation of the second, the third and the fourth variants)

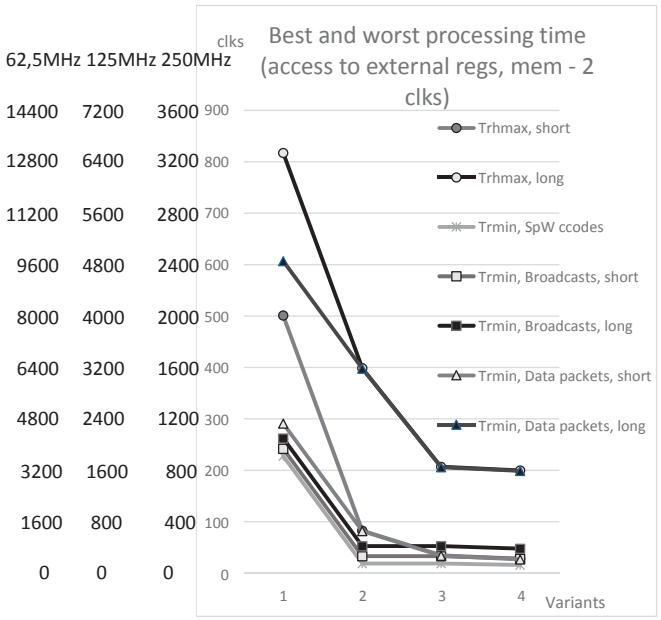


Fig. 13. Time characteristics of proposed variants

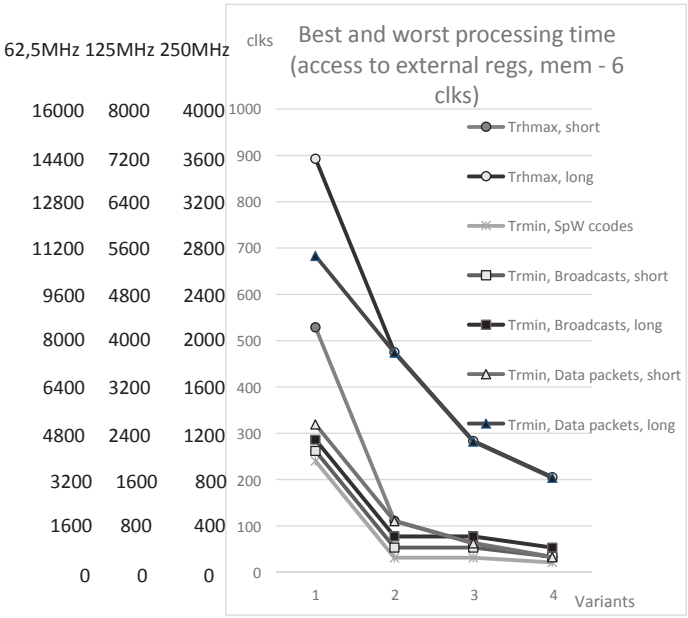
Let's consider the overhead costs for the area for the proposed implementation of the *Event Transmitting Manager*. We performed estimates for the proposed implementation variants with different instruction memory sizes (128, 256, 512 32-bit words). The memory size used determines the amount of

Graphs labeled “Trmin, short”, “Trhmax, short” correspond to Trmin and Trhmax for short processing algorithms (with minimal number of instructions). Graphs labelled “Trmin, long”, “Trhmax, … long” correspond to Trmin, Trhmax for long processing algorithms (with bigger number of instructions). Graphs, labelled “Trmin, SpW ccodes” correspond to Trmin when using SpaceWire control codes. Graphs, labelled “Trmin, Broadcasts …” correspond to Trmin when using Broadcast messages. Graphs, labelled “Trmin, Data packets …” correspond to Trmin when using data packets.

Trmin and Trhmax obtained using the formulas (1), (2) and using cycle accurate models of routers and terminal nodes are almost equal.

As you can see from these graphs, the hardware implementation of saving and restoring register values allows you to reduce Trhmax by 2 times, the hardware implementation of CRC calculation allows you to reduce Trhmax by 2 times and Trmin for handlers, that generate and transmit data packets by 2 times.

The fourth variant allow to reduce Trmin and Trhmax but no so essentially (by 30 – 50 %). For other systems, where the latency of accessing external registers and memory can be tens of clock cycles, the gain may be more noticeable.



.

software and, accordingly, the functionality of the *Event Transmitting Manager*.

For the second, third, and fourth variants, we performed estimates for two and four shadow register sets. For the estimates, we performed a synthesis using various technology libraries (design rules from 180 nm to 65 nm). The ratio of the

areas of different implementation variants does not depend much on the design rule. Fig. 14 shows the area ratios of different implementations of the *Event Transmitting Manager*. The area of the first implementation variant (with the instruction memory size of 128 words) is selected per unit area. As you can see from this figure, the area of instruction memory largely determines the area of the *Event Transmitting Manager*. (With a memory size of 512 words, the area is 1.5-2 times larger than with a memory size of 128 words.) The implementation of the mechanism for hardware saving and restoring register values has a rather noticeable effect on the area (an increase in the area by 20 – 40%). Hardware implementations of the other mechanisms considered have little effect on the implementation area.

Fig. 15 shows how much of the area of the terminal node, router is the area of the *Event Transmitting Manager*. For this evaluation we used the terminal node implementation with 2 SpaceFibre ports, and the router implementation with 8 SpaceFibre ports. As you can see from these histograms, the overhead cost of implementing all proposed variants does not exceed 23% of the area of the terminal node and 10% for router. The 10-15% limit is mostly exceeded by implementations with a memory size of 256, 512 words for Terminal nodes. Thus, in terms of hardware implementation costs, the main limitation is the amount of instruction memory used.

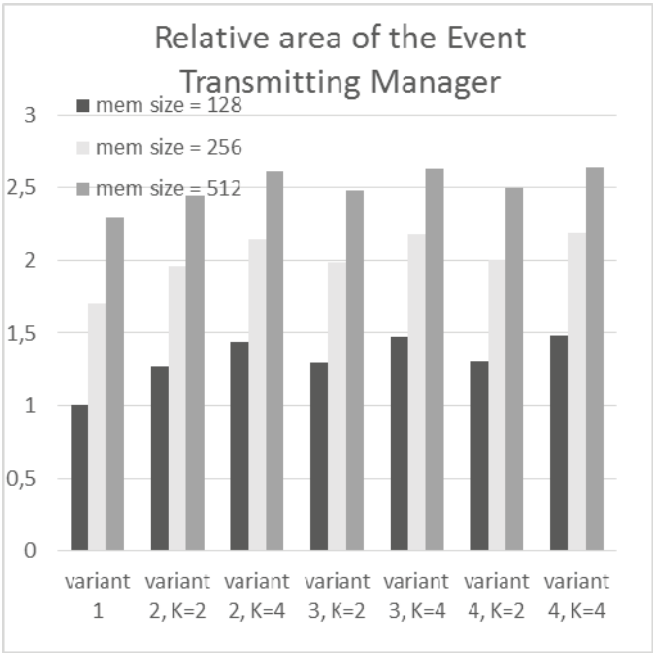


Fig. 14. Area of the Event Transmitting Manager

According to the estimates obtained, in most applications, the third and fourth implementation variants are the most suitable. The first implementation variant can be used in cases where large Trimax values are allowed and there are very significant area restrictions.

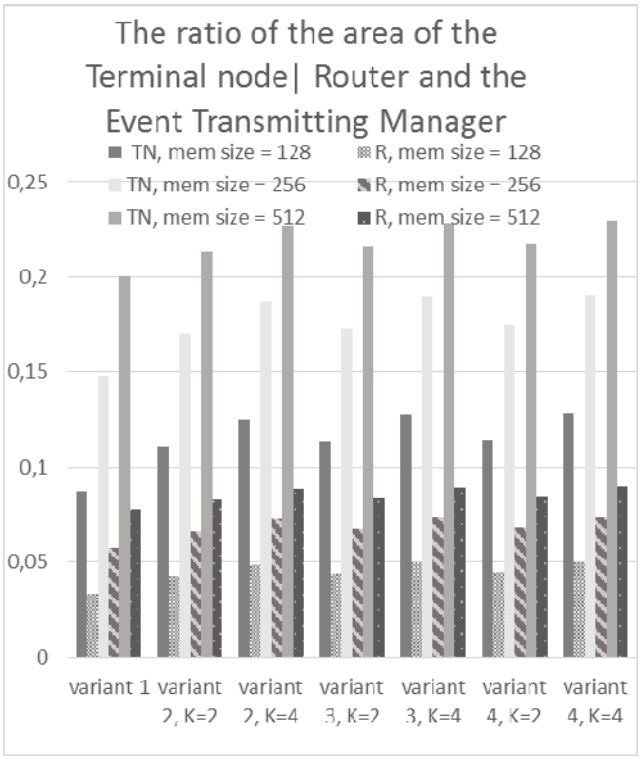


Fig. 15. The ratio of the area of the Terminal node|router and the Event Transmitting Manager

V. Conclusion

In this paper, we considered what requirements the mechanism for transmitting information about events occurring in terminal nodes and routers of the SpaceFibre network to a remote handler node must meet.

We considered the capabilities of RISC V ISA for implementing the *Event Transmitting Manager*.

The evaluation of achievable characteristics for a fully software-based implementation of the *Event Transmitting Manager* functions for the RISC V core was performed. The functions and actions whose program execution leads to the greatest time delays are defined. Variants of their hardware implementation have been developed, and instructions for controlling the corresponding hardware automata have been added to the instruction set.

For the proposed variants, estimates of time characteristics and overhead costs for implementation were made. In accordance with this, recommendations for their use are given.

Acknowledgment

The paper was prepared with the financial support of the Ministry of Science and Higher Education and of the Russian Federation, grant agreement No. FSRF-2020-0004 “Scientific basis for architectures and communication systems development of the onboard information and computer systems new generation in aviation, space systems and unmanned vehicles”.

REFERENCES

[1] SpaceFibre - Very high-speed serial link. ECSS-E-ST-50-11C.ESA-ESTEC.Noordwijk, The Netherlands. 15 May 2019. 233 p

[2] Suvorova E. Time synchronization in SpaceFibre Networks. FRUCT 28. 2021. 12 p.

[3] Waterman A., Lee Y., Patterson D. The RICS-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1. 2016. 133 p.

[4] Waterman A., Lee Y., Patterson D. The RICS-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.9. 2016. 91 p.

[5] Armv8-A Instruction Set Architecture. 2020. 39 p.

[6] Xtensa Instruction Set Architecture (ISA). Reference Manual. 2010. 662 p.

[7] Sumalan T., Lupu E., Arsinte R. Real Time Operating System Options in Connected Embedded Equipment for Distributed Data Acquisition. Carpatian Journal of Electronic and Computer Engineering 11/2. 2018. Pp 35 – 38

[8] Abdulganiyu A., Rabiu I. Comparative Analysis of Real-Time Operating System (RTOS) of Some Selected OS Using External Signal Generator and Oscilloscope. Int. Journal of Science and Engineering Investigations, Vol. 6, Issue 63. 2017. Pp. 47-53

[9] Huang X. Construction on Embedded Real-Time operating system of computer. 2nd international Conference on Electrical, Computer Engineering and Electronics. 2015

[10] Stangaciu C. S., Micea M.V., Cretu V.I. Hard real-time execution environment extensions for FreeRTOS. IEEE International Symposium on Robotic and Sensors Environment (ROSE). 2014

[11] Liebowitz K., Spies M., Rynardt C. VMware vSphere Performance: Designing CPU, Memory, Storage, and networking for Performance-Intensive Workloads. 2014

[12] Bloom G., Sherrill J. Scheduling and thread management with RTEMS. SIGBED Rev.11.1. 2014. Pp 20 – 25