Deadlock-Free Routing in SpaceWire Onboard Network

Lev Kurbanov, Ksenia Rozhdestvenskaya, Elena Suvorova
Saint-Petersburg State University of Aerospace Instrumentation
Saint-Petersburg, Russia
{lev.kurbanov, ksenia.khramenkova}@guap.ru, suvorova@aanet.ru

***Abstract*–In this article we consider the deadlock-free routing problem for onboard SpaceWire network with redundant devices. We work with static routing, it means that, routing tables are calculated and uploaded into the switches before launch the SpaceWire network. We solve deadlock-free problem with Up/Down routing approach, which is based on acyclic directed network graph. To build acyclic directed graph we modify the original algorithm of DFS based creating spanning tree. To find the routes of data transmission in the network we convert created directed graph to channel dependency graph. Also in this article we provide an example and explanation of our algorithms for deadlock-free routing in SpaceWire network.**

I. INTRODUCTION

This article is a continuation of the work [1], which tells about new computer-aided design system (CAD) for onboard SpaceWire networks, consisting of four components:

1) Onboard network topology design and evaluation of its structural characteristics;
2) Data transmission routing in a network;
3) Generation of the scheduling table for the STP-ISS transport protocol for the transmission of the data with Scheduled quality of service;
4) Simulation of the network operation with all the data current component got from other 3 components. Simulation performs with graphical user interface. Visualization and graphical user interface is used from VIPE project [2].

The article [1] is focused on the first component of CAD and it considers fault-tolerance analysis algorithm for SpaceWire onboard networks, which is based on k-connectivity search on a graph.

In the current work we consider the second component of CAD. Our task is data transmission routing in a network without deadlocks. This is one of the most important tasks in network design; it implies finding sequence of switches between source and target devices and creating routing tables for these switches. As we said, we work with static routing according to CAD's development specifications. Static routing differs from dynamic routing in that routing tables are adjusted in switches only once before network launch and cannot be automatically changed during network operation. Only manually changing the routing tables is possible, without special algorithms in contrast to dynamic routing. In other words, network administrator has to take a hand in each case when changes appear in onboard network. This is main disadvantage of static routing, but on other hand static routing is more stable and does not require large hardware and software switch resources for routing table service.

In dynamic routing, routes and routing tables are calculated on basis of the analysis of incoming messages during network operation. Switches or special devices, that are responsible for network state, can change routing tables taking into account incoming messages with information about network state. For instance, mechanism Plug-and-Play in SpaceWire network. In Plug-and-Play there is the special device (manager), which is responsible for routing tables state and their dynamic adjusting in case of network reconfiguration during network regular operation. In case of dynamic routing additional routes are used for service packets transfer, also switches should have processors and should be ready to execute special commands or analyze service packets by itself (without Plug-and-Play manager in network). The necessity of additional resources is disadvantage of dynamic routing. Increasing of network persistence is an advantage of dynamic routing. In case of dynamic routing designer of network does not have to analyze all possible network situations and find all possible routes to avoid any fault and save the traffic.

Static routing from this point of view is more restricted, but it has undeniable advantage – predictability. Network administrator using static routing knows packet routes at any moment, since they are known and do not change during network operation. The drawback of static routing is that scaling possibility is minimal or it is absent at all. When new device is added to N switches in network, it requires executing N write-commands. To put it otherwise, it is necessary to make special command about new device into each switch to adjust their communication with new device.

Therefore we can select the advantages of static routing:

- Easy checkout and adjusting in small networks. In small networks administrator can fast and with high quality check state of switches and determine fault in case of packet loss;
- Does not require additional software and hardware resources, since routing protocols are absent. Switches

ISSN 2305-7254

do not have to have processors, opportunity analyze commands and provide QoS of protocols;
- It is not necessary to create time out in switches operation. After executing the command, switch is immediately ready for operation. Time out period appears when switch rewrites his routing table. During this period all incoming packets have to wait for when switch will be able to transmit them to output ports;
- At any moment of network operation it is easy to predict switch operation. Administrator knows every routing tables.

The drawbacks of static routing are following:

- Weak scaling. In large networks it is impossible to add new device. Switches cannot determine and add new device in operation;
- Low stability to unpredicted network situations. Predict all possible network situations even in small network is difficult task for designer, and with increasing number of devices in network this task becomes impossible;
- Full checking route is necessary in case of packet loss. It is not obviously what the reason of packet loss. Therefore network administrator has to check each node in the route to determine the reason of fault and make manually corresponding changes to avoid subsequent packet losses.

In the current work we focus on onboard networks, which are based on SpaceWire standard. SpaceWire technology provides opportunity of creating onboard computing network with flexible network architecture and high throughput. Adjusting routing tables and adaptive routing tables in switches makes possible of using static routing in SpaceWire network.

The deadlock-free routing in static routing provide guarantee that packet transfer will be successfully finished in the target device, if following conditions are observed:

- There is the path in the network between source and target nodes, that can be used for data transfer;
- All physical channels and switches are stable and work correct.

The onboard network architecture is changed from mission to mission according to actual tasks. Therefore designers cannot use solutions that were achieved in previous projects. It means that for each new mission it is necessary to repeat whole process of network design including deadlock-free routing and creating routing table. This is very difficult and high consumption task. In this article we provide solution of this task. We describe a method that helps find deadlock-free routes and choose more suitable routes, taking into account criteria.

In section II we describe SpaceWire onboard network and its features and restrictions. Statement of the problem and its analysis are in section III. Explanation of deadlocks and description of deadlock-free routing you can find in section IV. Section V includes description of Up/Down routing that was chosen as base approach of deadlock-free routing. Section VI is devoted to search of routes in channel dependency graph that we obtain after applying Up/Down rules. Section VII describes the criteria which is used for second component of CAD. Conclusion is section VIII.

II. The network description

The network consists of main and redundant elements, because in case of failure of any element in the network, the only possible solution to support the maintain network operation is to replace the failed network element with workable network element that is similar to previous element. The network element can be a switch, a physical channel, a port or a unit in the device. The main network element we call element that is immediately ready to transmit the data. The redundant network element we call element that is workable but which does not take part in regular network operation. Other word it is disabled element, but which can be enabled in case of failure of main element.

Fig 1. shows an example of network, where you can see redundant and main network elements. Nodes with units “A”, “B” and “C” correspond to devices in the network. Remain nodes are switches. The main elements in this example are dark grey colored switches and all units “A”. The redundant elements are light grey colored switches and all units “B” and “C”. The redundant channels are channels that are connected between redundant elements or between redundant and main elements.

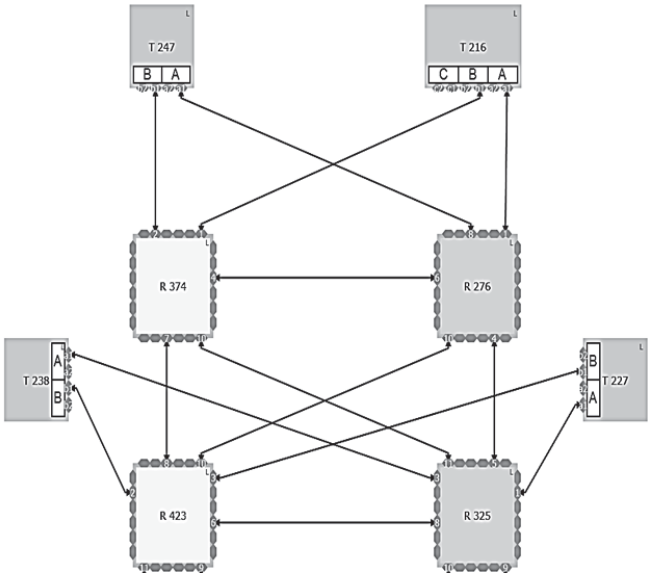

Fig. 1. An example of SpaceWire onboard network

III. Statement of the problem. Restrictions of SpaceWire standard

The task of the second CAD component is to find the specified by network designer number of routes, which do not lead to deadlocks and satisfy to specified criteria (the maximum data transmission delay and distance between connected devices). It is advisable to solve this task consistently, separating it into subtasks:

- Find all possible routes that do not lead to deadlocks;

- Choose routes that satisfy to criteria.

We have to note the additional requirements for data transmission routing:

- At least one of routes should to use only main network elements;
- Route that uses redundant elements should also use main elements.

SpaceWire standard has restrictions that we have to take into account:

1) Device in the network can generate packets with any rate and send these packets to any devices.
2) The packet that reaches its destination is deleted from the network. A packet arriving at the destination address is not forwarded to the network.
3) Wormhole routing is used in a network.

We also observe the following rules for packet transmission that are used in the network:

The wormhole routing rule. When a packet enters the input port on switch, the packet header is read, and according to routing table header is sent to output port. Packet tail is transmitted through switch "on the fly", taking up channel for transmission. Only one packet can be transmitted on the channel at a time.

The blocking rule. If the required channel is busy, packet is stored to the input buffer of switch and waits until the busy channel becomes free. If size of packet is larger than buffer space, the rest of packet is stored into input buffers of previous switches according to its route, occupying all channels between these switches. The packet cannot be separated and partially transmitted or transmitted through different routes. If the tail of blocked packet does not occupy full space in the buffer, the next incoming packet at this port will fill the rest of buffer space and will be blocked according to the blocking rule.

The rule of packet removing. This rule is related to device operation. The packet is completely unloaded from the network when the device receives the header byte.

These rules are basic for SpaceWire standard and it is necessary to take them into account.

IV. DEADLOCK-FREE ROUTING

In the network transfer process some packets may be locked and these packets do not reach their destination nodes. Usually such situations are results of deadlocks, when packet can be locked by other packets that use necessary resources for this packet. In other words, the main reason of deadlocks is the limited set of resources in the network. The physical channel is the main network resource.

Deadlocks appear as a result of cyclical dependences of resources in the network. When a packet is transmitted over a network, it takes up a physical channel on each hop between switches. As the packet travels through the network, the tail of the packet releases previously occupied channels.

Packet in the network has two states:

- "Exclusive ownership of channel" in case of packet transmitting;
- "Resource wait-for" state, when header of packet is blocked.

The "exclusive ownership" and "resource wait-for" conditions makes cyclic dependencies and deadlock possible [3].

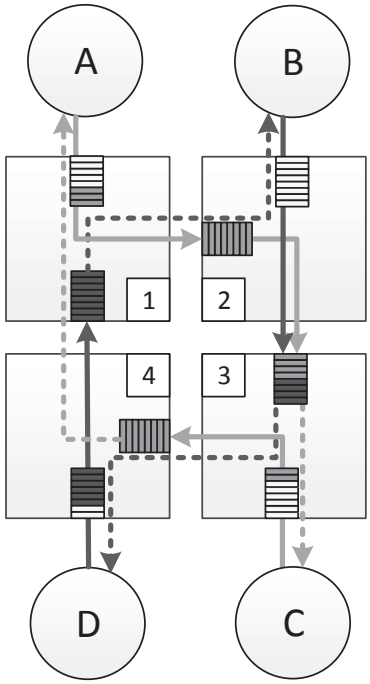

Fig. 2. Example of deadlock

In the Fig.2 you can see an example of deadlock. Circles represent the source and target nodes, squares represent switches. There are four routes in the network. Packet 1 has the route D-4-1-2-B. Packet 2 is transmitted through B-2-3-4-D. Packet 3 is transmitted through A-1-2-3-C. Packet 4 has the route C-3-4-1-A. Lines with arrows represent direction and route of data transition from source node to target node. Solid lines represent packet's "exclusive ownership of channel". The dash lines correspond to packet's "Resource wait-for" state.

In Fig. 2 deadlock includes all packet flows. The packet from node A is blocked on switch 3 because of packet from node B. The packet from node B is blocked on switch 3 by packet from node C. The packet from node C is waiting for free channel between switch 4 and switch 1, which is occupied by packet from node D. Packet from node D is blocked on switch 1 because of packet from node A.

Such deadlock stops whole network, and data transmission is impossible.

To describe this situation we create the channel dependency graph (CDG). An example you can see in Fig. 3.

CDG describes dependency of channels, which are used by packets. The vertices represent channels of network. Arcs correspond to sequence of using the channels by packets.

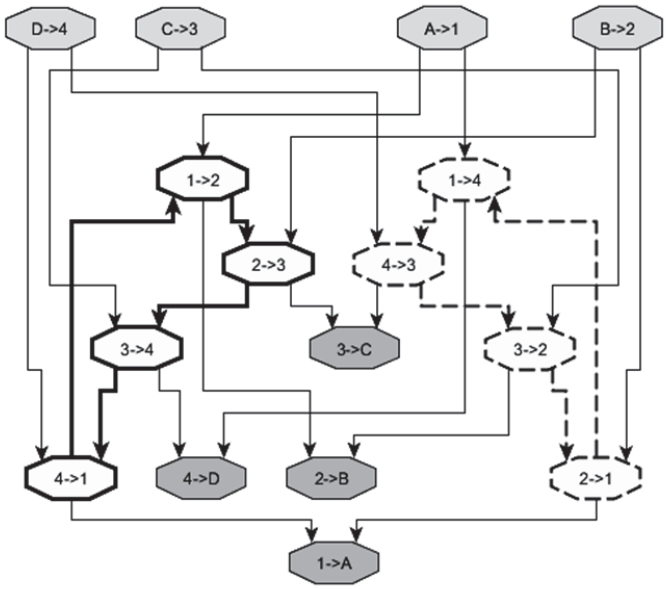

Fig. 3. Channel Dependency Graph

Dally W. J. and his colleagues in [4] formulated the following theorem: The routing function in the network is deadlock-free, if there are no cycles in the Channel Dependency Graph. This theorem is fundamental in deadlock-free routing problem.

In Fig. 3 you can see two cycles. The cycle with the bold line shows the situation from example in Fig. 2. The second cycle with dash line represent other deadlock, which can appear in the network.

There is a solution for deadlock-free routing problem [5]. Authors present an algorithm of breaking cycles in CDG. If deadlock exists, it removes by virtual channel insert. The presented algorithm [5] does not restrict designer in choosing the routing, but it requires some resources and technologies that cannot be applied in some networks. For example, it is impossible to solve deadlock-free routing problem by virtual channel insert for SpaceWire network, since this standard does not support virtual channel technology. Such strategy can be used only if deadlocks are rare and it is possible to remove them by introducing additional resources [6].

V. Up/Down routing

To solve deadlock-free routing problem authors [7] present an approach that is based on Up/Down routing for regular and irregular topologies. The Up/Down routing is the popular solution for deadlock-free routing in the modern commercial networks.

The classic Up/Down routing approach consists of the following steps:

1) Graph representation of the network;
2) Spanning tree construction with breadth-first-search algorithm (BFS);
3) Numbering of vertices in the spanning tree;
4) Direction assignment of the edges in spanning tree according to the numbering of vertices (direction from vertex with low number to vertex with higher number). The resulting graph should be acyclic;
5) Applying Up/Down rules in the directed acyclic graph based on network structure.

The Up/Down rules can be formulated in following points that are consistent in their execution:

a) A route can be laid through zero or more channels in "up" (forward) direction;
b) A route can be laid through zero or more channels in "down" (backward) direction.

The sequence of applying of these rules is strictly regulated and requires consistently execution: *a* at first and then *b*. Applying *a* after *b* is forbidden.

In this way we can avoid channel dependencies, because of packet cannot be transmitted through physical channel in "up" direction after transmitting through physical channel in "down" direction.

For instance, in Fig. 6 according to Up/Down rules the route A-1-4-3-C is allowed. The route A-1-2-3-C is forbidden.

In [7] authors in detail describe method that creates the spanning tree based on deep-first-search algorithm (DFS). The provided algorithm is more flexible in criteria adjusting to making the route, than classic method with BFS based spanning tree.

When we create DFS based spanning tree, it is necessary to choose the root vertex and choose the next vertex. In [8] authors present heuristic rules to choose root vertex for creating DFS based spanning tree. Also authors formulate heuristic rule to choose next vertex to add it to spanning tree.

If we create DFS based spanning tree from each vertex in the graph we get different solutions. In each directed graph that we acquire from spanning tree we can find at least one route between any couple of devices. It means that to get full set of solutions we have to build spanning tree from each vertex in the network graph.

According to our statement of problem we have to find all possible routes, therefore we consider full set of solutions.

When working with the SpaceWire network, the proposed methodology should be clarified. To choose the root vertex we should consider only those that are representation of the devices. Otherwise, if we create spanning tree from switch, we

can lose this switch as functional network element, because of transmitting through this switch becomes impossible according to Up/down rules.

If we create the DFS based spanning tree from vertex that is a representation of device in the network (Fig. 4.a), we get solution, where all switches can transmit data and are functional elements in the network. In case when we create spanning tree from vertex that is a representation of switch (Fig. 4.b), we get solution, where switch 5 cannot transmit data.

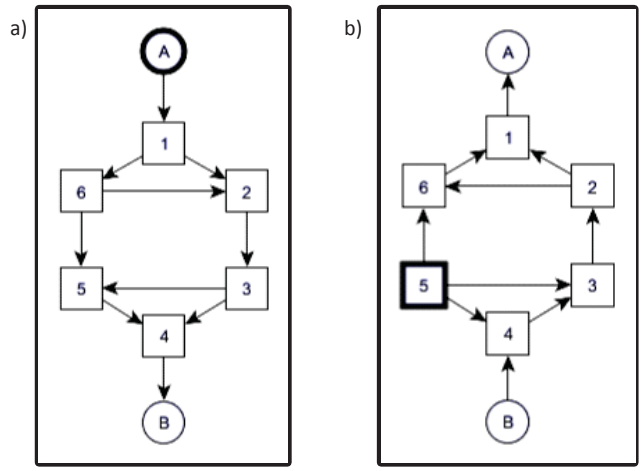

Fig. 4. An example of solutions with different root vertices: a) root vertex is device A; b) root vertex is switch 5

That is because of all channels that are connected to switch 5 are outgoing. According to Up/Down rules packet can income to this switch, but it cannot be transmitted forward, since after transmitting through channels in "down" direction transmitting through "up" direction is forbidden. Therefore, we build spanning trees only from those vertices that are representation of devices in the network.

To build DFS based spanning tree it is necessary know what the next vertex will be added to spanning tree. In [8] authors suggest choosing as next vertex that has with maximum number of connections with vertices in the spanning tree. In case of tie, next vertex is vertex that has maximum value of average topology distance. Authors argue that applying of this heuristic leads to reducing number of routing restrictions in the switches. You can find more detail information in [8]. An example you can see in Fig. 5.

Fig.5 shows an example of DFS based spanning tree. The boldest arcs (between vertices A,1,4,D) correspond to main branch, i.e. branch that is created until first return from recursive function. The less bold arcs correspond to secondary branches that are created after first return from recursive function. The thinnest arc (between vertices C and 3) corresponds to last secondary branch. The set of solid arcs and vertices forms the spanning tree. The dash arc is the connected arc between branches in the spanning tree.

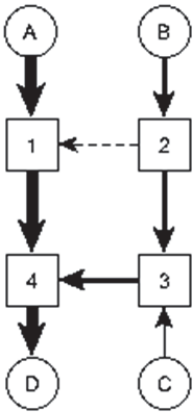

Fig. 5. DFS based spanning tree

Let's consider with algorithms that we use. For comfortable reading the algorithms we offer use Table 1 with notations and their descriptions.

We represent network as non-directed graph G (V, E). The set of vertices V corresponds to set of all devices and switches. This set includes main as well as redundant nodes in the network. Set of edges E corresponds to set of all channels, including channels between redundant elements. This network representation is most comfortable for finding deadlock-free routes and take into account redundant network elements.

An algorithm that we use does not require the vertices numbering to assign the edge direction. We assign the direction during the DFS creating spanning tree. This algorithm is shown in the *Algorithm 1*. The original algorithm is presented in [8].

Textual accompaniment for direction assignment algorithm:

- As input data we use root vertex v, flag MainBranch is equal to TRUE;
- The main cycle of algorithm is the deep-first-search of the graph G. This cycle executes until set ST is equal to set V;
- Vertex v is added to set ST, then we search all its neighbors (str. 3-4);
- If vertex v has no any neighbors, it means that current branch is finished. We set flag MainBranch to FALSE and make return from recursive procedure (str. 5-8);
- If set of neighbors NV is not empty, we find all edges between each neighbor vertex and vertices in the spanning tree, except current vertex v. For each edge we assign the direction according to MainBranch flag (str. 9-19);

TABLE I. INSTRUCTIONS

Notification	Description
$G(\mathbb{V}, \mathbb{E})$	Network's graph
$G(\mathbb{V}, \mathbb{A})$	Network's directed graph after direction assignment algorithm
$D(\mathbb{V}, \mathbb{A}')$	Network's digraph
$CDG(\mathbb{V}', \mathbb{A}'')$	Channel Dependency Graph
$\mathbb{V} = \{v_i : 1 \leq i \leq CN\}$	Set of the vertices in the network's graph G, where CN - quantity of nodes in network
$\mathbb{E} = \{edge : edge = \{v_i, v_j\} \wedge v_i, v_j \in \mathbb{V} \wedge i \neq j\}$	Set of edges in the network's graph G, where $edge = \{v_i, v_j\}$ - edge between vertex v_i and vertex v_j
$\mathbb{A} = \{arc : arc = (v_i, v_j) \wedge v_i, v_j \in \mathbb{V} \wedge i \neq j\}$ $\mid \mathbb{A} \mid = \mid \mathbb{E} \mid$	Set of arcs in the network's graph G, where $arc = (v_i, v_j)$ - arc from v_i to v_j
$\mathbb{A}' = \{arc : arc = (v_i, v_j) \wedge v_i, v_j \in \mathbb{V}' \wedge i \neq j\}$ $\mid \mathbb{A}' \mid = 2 \cdot \mid \mathbb{A} \mid$	Set of arcs in the network's digraph D
$\mathbb{V}' = \{v_i : 1 \leq i \leq \mid \mathbb{A}' \mid \wedge v_i \in \mathbb{A}'\}$	Set of vertices in the Channel Dependency Graph
$\mathbb{A}'' = \{arc : arc = (v_i, v_j) \wedge v_i, v_j \in \mathbb{V}' \wedge i \neq j\}$	Set of arcs in the Channel Dependency Graph
$\mathbb{ST} \subseteq \mathbb{V}$	Set of vertices in the spanning tree
$\mathbb{NV} = \{w : w \in \mathbb{V} \backslash \mathbb{ST} \wedge \exists\ edge = \{v, w\} \in \mathbb{E}\}$	Set of neighbours for current vertex v
$\mathbb{NC} = \{edge : edge = \{x, w\} \wedge edge \in \mathbb{E} \wedge w \in \mathbb{ST} \backslash \{v\}\}$	Set of edges between neighbour and vertices (except v) in the spanning tree
$PNC(x)$	Returns power of set $\mathbb{NC}$ for vertex x
$AVGD(x)$	Returns average topology distance for vertex x
$next \in \mathbb{V}$	Vertex that is chosen to add to spanning tree
$MainBranch$	Boolean flag that can be: TRUE - main branch is creating; FALSE - secondary branch is creating

- If main branch is creating (MainBranch=TRUE), then arc direction is from vertex in the spanning tree to neighbor vertex. If secondary branch is creating (MainBranch=FALSE), then arc direction is from neighbor vertex to vertex in the spanning tree (str. 12-16);
- We add arc to set A (str. 13,15);
- We store for each neighbor vertex number of connections with the spanning tree (str. 18);
- We filter set of neighbors NV and save next vertex that has maximum number of connections with the spanning tree (str. 20). If there are several such vertices, we filter NV again and save only one next vertex that has maximum value of the average topology distance (str. 21-25);
- We assign the edge direction between current vertex v and chosen next vertex according to MainBranch flag. If main branch is creating (MainBranch=TRUE), then arc direction is from current vertex v to neighbor vertex. If secondary branch is creating (MainBranch=FALSE), then arc direction is from neighbor vertex to current vertex v (str. 26-30);
- We add arc to set A (str. 27, 29);
- Algorithm is repeated for *next* vertex (str. 31).

This algorithm does not lead to cyclic directed graph, since cycles are possible if we create in one branch both arcs with forward and backward directions. According to our algorithm we can create only one type of direction in one branch.

Since we build the spanning trees from vertices that are the representation of devices in the network, there are several solutions. In Fig, 6 you can see one of the solutions of direction assignment algorithm. This solution is for root vertex A.

Algorithm 1 Direction assignment algorithm

```
1:  procedure DirectionAssignment(v)
2:      while ST ≠ V do
3:          ST ← ST ∪ {v}
4:          NV ← {w : w ∈ V\ST ∧ ∃ edge = {v, w} ∈ E}
5:          if NV = ∅ then
6:              MainBranch = FALSE
7:              return
8:          end if
9:          for all x ∈ NV do
10:             NC ← {edge : edge = {x, w} ∧ edge ∈ E ∧ w ∈ ST\{v}}
11:             for all edge ∈ NC do
12:                 if MainBranch = TRUE then
13:                     A ← A ∪ {arc : arc = (w, x)}
14:                 else
15:                     A ← A ∪ {arc : arc = (x, w)}
16:                 end if
17:             end for
18:             PNC(x) ← | NC |
19:         end for
20:         NV ← {w : w ∈ NV ∧ PNC(w) = max_{1≤i≤|NV|} (PNC(w_i))}
21:         if | NV |> 1 then
22:             next ← w, where w ∈ NV ∧ AVGD(w) = max_{1≤i≤|NV|} (AVGD(w_i))
23:         else
24:             next ← w, where w ∈ NV
25:         end if
26:         if MainBranch = TRUE then
27:             A ← A ∪ {arc : arc = (v, next)}
28:         else
29:             A ← A ∪ {arc : arc = (next, v)}
30:         end if
31:         DirectionAssignment(next)
32:     end while
33: end procedure
```

In Fig. 6 the bold arrows correspond to arcs and their direction after applying direction assignment algorithm. The grey arrows in the switches 1, 2, 3 and 4 show allowing direction for transmitting data according to Up/Down routing. As a result of algorithm in switch 2 data transfer is forbidden from channel 2 to channel 3 and in opposite direction as well. In this way we avoid the channel dependencies and the deadlocks.

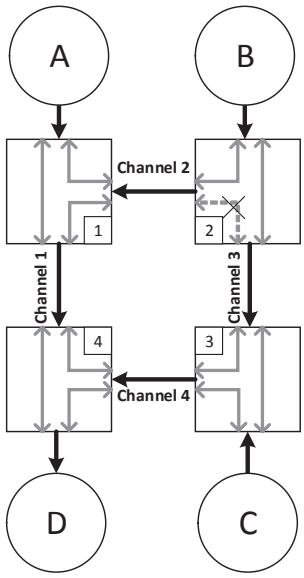

Fig. 6 The network directed graph after direction assignment algorithm

VI. Conversion of directed graph to CDG. The routes search .

Since we get the directed graph we suggest to convert it to CDG for more comfortable routes search. Conversion should be with respect to Up/Down rules and it can be separated by two subtasks:

- Creating the additional arcs in opposite direction;
- Conversion of arcs to vertices of CDG and connecting them according to Up/Down rules.

Creating the additional arcs. Since in the network full-duplex channels are used, we represent each channel as two arcs with opposite directions. As we work with directed graph at this stage, we create arc with opposite direction for each arc in the graph, marking the original arc. The marking arc is necessary for connecting the vertices in CDG. Graph with additional opposite arcs we call digraph D(V, A'), where A' is the set of original and opposite arcs. The algorithm of conversion of directed graph G to digraph D is shown in the Algorithm 2.

Algorithm 2 Conversion of directed graph to digraph

```
1: function MakeDiGraph(G(𝕍, 𝔸))
2:     for all arc = (a, b) ∈ 𝔸 do
3:         e1 ← arc, where arc = (a, b)
4:         IsMark(e1) ← TRUE
5:         e2 ← arc, where arc = (b, a)
6:         IsMark(e2) ← FALSE
7:         𝔸' ← 𝔸' ∪ {e1, e2}
8:     end for
9:     return D(𝕍, 𝔸')
10: end function
```

Conversion of arcs to the vertices of CDG. The vertices in CDG correspond to arcs in digraph D. The key moment creating CDG is the connecting its vertices. Connections in CDG correspond to possible sequence of using channels by packet. We create CDG with respect to Up/Down rules and some sequences of using channels are forbidden. It means that arcs that create cycles in the CDG will be absent, therefore we avoid channel dependencies and deadlocks are excluded.

Algorithm 3 Conversion of directed graph to channel dependency graph with respect to Up/Down rules

```
1: function ConvertToCDG(G(𝕍, 𝔸))
2:     D(𝕍, 𝔸') ← MakeDiGraph(G)
3:     for all arc = (x, y) ∈ 𝔸' do
4:         v1 ← arc
5:         𝕍' ← 𝕍' ∪ {v1}
6:         if IsMark(arc) = TRUE then
7:             ℕ𝕍' ← {arc' : arc' = (a, b) : arc' ∈ 𝔸' ∧ a, b ∈ 𝕍 ∧ y = a ∧ x ≠ b}
8:         else
9:             ℕ𝕍' ← {arc' : arc' = (a, b) : arc' ∈ 𝔸' ∧ a, b ∈ 𝕍 ∧ y = a ∧ x ≠
   b ∧ IsMark(arc') = FALSE}
10:        end if
11:        for all v2 ∈ ℕ𝕍' do
12:            𝕍' ← 𝕍' ∪ {v2}
13:            𝔸'' ← 𝔸'' ∪ {arc'' : arc'' = (v1, v2)}
14:        end for
15:    end for
16:    return CDG(𝕍', 𝔸'')
17: end function
```

The graphical representation of conversion of directed graph G to CDG is shown in Fig. 7 for descriptive reasons.

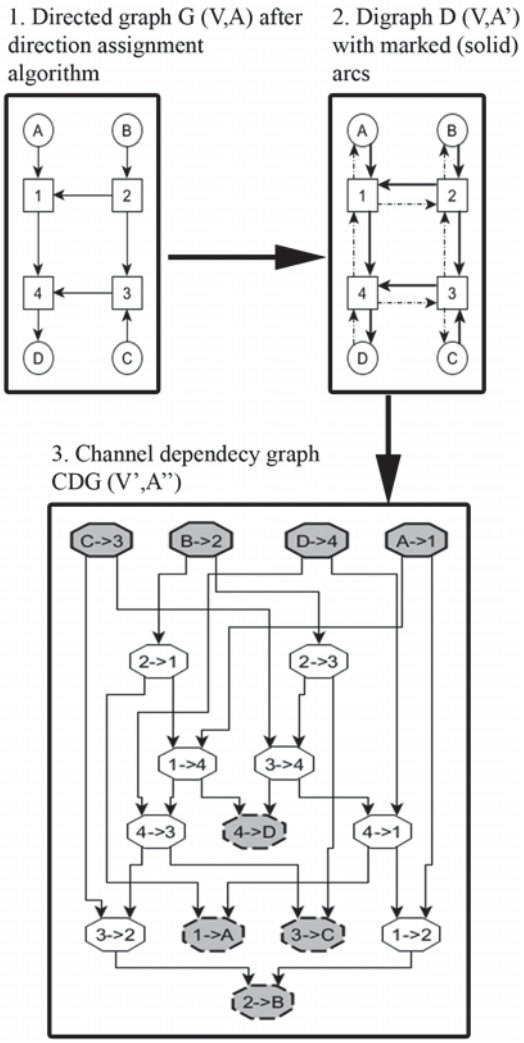

Fig. 7 An example of conversion of directed graph G to CDG

Now it is easy to find all routes between couple devices using BFS based algorithm in CDG. We suggest apply the BFS algorithm starting with leafs of CDG (incoming channels to devices). When we use BFS starting with leafs, we can create routing tables for all devices on each step of BFS algorithm.

The BFS algorithm allows us to find all routes between couple devices, including shortest routes, and write into all intermediated nodes corresponding routing tables.

Without filtering the gained routes by selection criteria, the result of algorithms is an adaptive routing for each switch in the network. The drawbacks of the adaptive routing you can find in [1]. In the current work there are selection criteria, which are described in the next section.

VII. Routes selection by criteria

In case of existing of several routes between devices, selection of optimal routes is applied. To define the degree of route optimality (A), we suggest to use formula of weighted average for two criteria that takes the following form:

$$A = \frac{h * x_1 + l * x_2}{2}, where\ x_1 + x_2 = 1 \quad (1)$$

, where

- x_1 – coefficient that is responsible for number of hops between devices. The reduction x1 reduces the distance influence on selection the optimal route.
- x_2 – coefficient that is responsible for delay in data transmission between devices. The reduction x_2 reduces the delay influence on selection the optimal route.

Network designer should set the coefficients x_1 and x_2 for each information flow.

To bring the parameters to the general range, we propose to normalize the parameters using the following formulas

$$h = \frac{MinDist}{Dist} \quad (2)$$

$$l = \frac{MinDelay}{Delay} \quad (3)$$

where MinDist – minimal distance (hops) among all routes between considered couple of devices, Dist – number of hopes in the current route; MinDelay – minimal delay among all routes between considered couple of devices, Delay – delay for the current route.

Since coefficients depend from each other $(x_1+x_2=1)$, we can express it with y:

$$x_1 = y, \quad x_2 = 1-y. \quad (4)$$

Therefore to select the optimal route by distance and delay criteria it is enough to set y.

VIII. Conclusion

In the current work we have researched the deadlock-free routing problem in SpaceWire onboard networks. We have considered the reasons of deadlocks and presented an algorithm for deadlock-free routing. This algorithm is based on Up/Down routing approach and the DFS spanning tree. Our algorithm does not require the numbering of vertices to assign the edge direction and guarantees acyclic directed graph as result. To find routes according to Up/Down rules we have built the CDG, where it is more comfortable find all routes using BFS algorithm. Also we suggested distance and delay criteria to choose optimal route among gained routes between each couple of devices.

Acknowledgment

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation under the contract RFMEFI57816X0214.

References

[1] Lavrovskaya I., Olenev V., Korobkov I., “Fault-Tolerance Analysis Algorithm for SpaceWire Onboard Networks”, *Proceedings of the 21st Conference of Open Innovations Association FRUCT. – FRUCT Oy*, Oct. 2017, pp. 28.

[2] Syschikov, Y. Sheynin, B. Sedov, V. Ivanova, “Domain-specific programming environment for heterogeneous multicore embedded systems”, *International Journal of Embedded and Real-Time Communication Systems*, vol 4. 2014, pp. 1-23.

[3] S. Warnakulasuriya, T. M. Pinkston, “Characterization of deadlocks in interconnection networks”, *Parallel Processing Symposium, 1997. Proceedings., 11th International. – IEEE*, 1997, pp. 80-86.

[4] W. J. Dally, C. L. Seitz, *Deadlock-free message routing in multiprocessor interconnection networks*, 1988.

[5] Seiculescu, S. Murali, L. Benini, G. De Micheli, “A method to remove deadlocks in networks-on-chips with wormhole flow control”, *Proceedings of the Conference on Design, Automation and Test in Europe,* March 2010, pp. 1625-1628.

[6] J. Duato, S. Yalamanchili and L. M. Ni, *Interconnection networks: an engineering approach*, 2003.

[7] J. C. Sancho, A. Robles, J. Duato, “A new methodology to compute deadlock-free routing tables for irregular networks”, *International Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing*, Jan. 2000, pp. 45-60.

[8] J. C. Sancho, A. Robles, “Improving the Up/Down routing scheme for networks of workstations”, *European Conference on Parallel Processing*, 2000, pp. 882-889.