Microprocessors and Microsystems 000 (2016) 1–9

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Modern methods in railway interlocking algorithms design

Piotr Kawalec[a], Marcin Rżysko[b,*]

[a] Warsaw University of Technology, Faculty of Transport, Koszykowa 75, Warszawa, 00-662, Poland
[b] Bombardier Transportation (Rail Engineering) Polska, Ogrodowa 58, Warszawa, 00-876, Poland

ARTICLE INFO

Article history:
Received 26 October 2015
Accepted 22 November 2015
Available online xxx

Keywords:
Algorithms
Interlocking
Railways
Train control

ABSTRACT

Despite years of railway control and signalling development, modern formal description methods are still not widely used. Lack of standards in the interlocking logic construction method causes the development of the railway control systems to be more and more expensive. Moreover, the microprocessor technology used nowadays reaches its limits regarding signal processing time in decentralised systems. This forces the industry to seek for new solutions. This paper presents an algorithmic approach to interlocking logic development, together with a modern implementation methods using hardware description languages and programmable devices.

1. Introduction

Railway has become the most complex mean of transportation in terms of traffic control and safety. Currently developed solutions allow nearly autonomous train operation at very high speeds, providing continuous control on each level, from the track occupancy check, through real-time speed profile calculation and execution, to automatic routing of multiple trains in a control area. Ensuring safety of these operations requires redundant and safety-critical appliances.

Since the beginnings of the railway interlocking development the train speeds and number of required functionalities increase. Starting with the relay technology introduction, railway interlocking systems began to be decentralised and remotely controlled, so as previously one large station could have around ten manned signal boxes, since the relay era one dispatcher could fully control large area including lines and stations. It was possible also due to the fact, that the relay technology is relatively fast – the signal delay time depends mostly on electric wave propagation. Major disadvantages of the relay systems were the cost of safety-related relays and high space consumption in signal boxes.

Today most of the railway interlocking systems are based on microprocessor technology. These became more and more popular along with increasing availability of large scale of integration devices and *Commercial Off-The-Shelf* hardware. The software part of these solutions is often based on the preceding relay systems. Although it was an easy way to transfer the existing functions to a new technology platform, vulnerabilities of the predecessors are sometimes preserved. The calculation time in software solutions is also a lot higher, causing that implementing new functions becomes more problematic.

Nowadays trends also begin to point at minimising the costs of railway interlocking equipment, but keeping the safety standards at equally high level. It is therefore necessary to look for a new, complete and efficient solution for railway interlocking systems. A possible next step is to go back to the hardware, or hybrid solutions, taking all new achievements in specialised electronic devices into consideration.

Regardless of the technology used, every modern railway interlocking system has to conform to the high safety parameters. The following documents issued by the European Committee for Electrotechnical Standardization (CENELEC) describe the requirements for the new systems:

- EN50126 – Railway applications. The specification and demonstration of reliability, availability, maintainability and safety (RAMS);
- EN50128 – Railway applications. Communications, signalling and processing systems. Software for railway control and protection systems;
- EN50129 – Railway applications. Communications, signalling and processing systems. Safety related electronic systems for signalling.

These documents describe both hardware (EN50126 and EN50129) and software (EN50128) part of the system. However there are no strict guidelines for hardware logic implementation such as FPGAs. Nevertheless, due to the rapid development of these solutions, this approach needs to be revised.

* Corresponding author. Tel.: +48 698648063.
E-mail addresses: pka@wt.pw.edu.pl (P. Kawalec), marcin.rzysko@pl.transport.bombardier.com, m.rzysko@gmail.com (M. Rżysko).

http://dx.doi.org/10.1016/j.micpro.2015.11.018
0141-9331/© 2015 Elsevier B.V. All rights reserved.

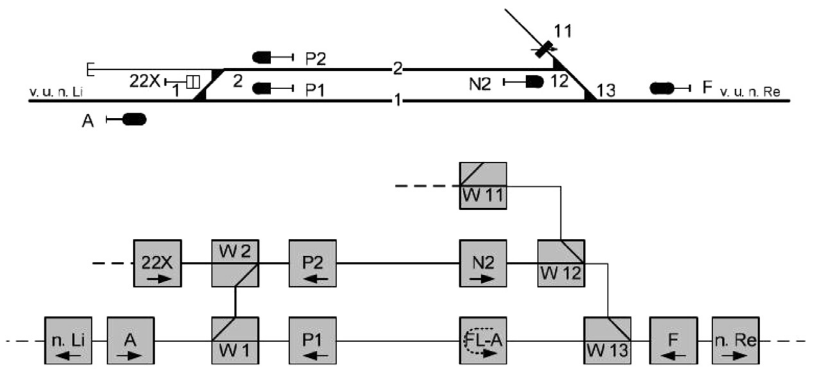


Fig. 1. Track layout and the corresponding logical objects connection plan (from [7]).

In this paper an algorithmic approach to the interlocking logic design is proposed, followed by an implementation method using programmable devices.

2. Material and methods

Many various approaches to the interlocking logic design have been developed over the decades of railway history. Some of the railway administrations established their own unique solutions. Even though there are various formal models for describing the project requirements (*Geographic Data Language* for British Railways, *PlanPro* model for Deutsche Bahn or the *SUBSET112* for European Union railways), no such model exists for defining the interlocking logic specification itself. This matter however becomes more often a topic of the scientific discussions [2,6]. Because of the recent increase of executed functions, and various tries to unify the cooperation of interlocking systems developed by different suppliers, it is even more important to create such model.

In many of the currently developed interlocking systems the *topological principle* is used. The topological principle means that every logical element type (such as point, signal, level-crossing) is described in a generic way and then used for a specific site application. In relay interlocking this can be a relay module [8], in computer-based interlocking it is usually a separate, reusable part of the code.

A basic building block of the topological system is a *logical object*. Topological principle allows us to completely describe every logical object type once, and then use such generic specification for the designed station layout. It is then necessary that the objects are organised according to the topology of the station. An example of the track layout and a corresponding logical object layout is shown in Fig. 1.

Object communicate with each other using data channels and execute the designed functions. The connections between the logical objects are usually organised into geographical and non-geographical. The geographical connections take part in all complex functions like routes, signalling, passage control. Non-geographical connections allow implementation of additional, custom functions such as blocking all points in particular area.

Taking the topological model as a base, the system core can now be described. For creating the formal specification of such model it is necessary to organise all data processed by the system into relevant vectors.

2.1. Data analysis

Considering the railway interlocking system integrally, it has two main interfaces. One between the interlocking system and the dispatching system (Man-Machine Interface), allowing the dispatcher to issue commands and observe the results. The second is

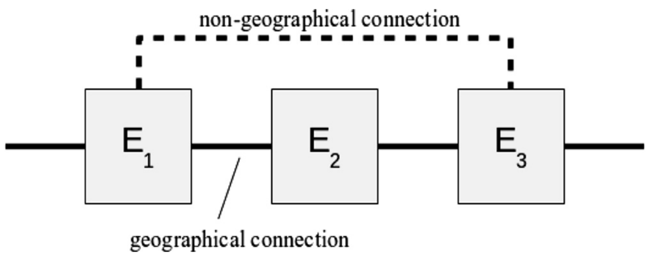


Fig. 2. Data flow between logical objects.

between the interlocking system and the object controllers, which are connected directly to the track-side equipment (points, signals, axle counters). This interface allows us to interact with the external environment.

According to the above, data flow can be organised into following alphabets:

- input data $X = \{X_P, X_K\}$,
- output data $Y = \{Y_M, Y_S\}$.

These sets consist of the following vectors:

- command vector $X_P = \{X_{P1}, X_{P2}, \ \ldots\ , X_{PN}\},\ X_{P\ldots} = (x_{p0}, x_{p1}, \ldots, x_{pn})$;
- check vector $X_K = \{X_{K1}, X_{K2}, \ldots, X_{KN}\},\ X_{K\ldots} = (x_{k0}, x_{k1}, \ldots, x_{kn})$;
- indication vector $Y_M = \{Y_{M1}, Y_{M2}, \ldots, Y_{MN}\},\ Y_{M\ldots} = (y_{m0}, y_{m1}, \ldots, y_{mn})$;
- manoeuvre vector $Y_S = \{Y_{S1}, Y_{S2}, \ldots, Y_{SN}\},\ Y_{S\ldots} = (y_{s0}, y_{s1}, \ldots, y_{sn})$.

where N is a number of logical objects and n – number of variables in the vector.

To be able to analyse the data flow inside the system, additional vectors have to be foreseen.

When describing a single object E_i the *interlocking vector* was introduced. It allows the logical objects to communicate with each other.

- $X_Z = \{X_{ZA}, X_{ZB}, \ldots, X_{ZG}\},\ X_{Z\ldots} = (x_{z0}, x_{z1}, \ldots, x_{zn})$;
- $Y_Z = \{Y_{ZA}, Y_{ZB}, \ldots, Y_{ZG}\},\ Y_{Z\ldots} = (y_{z0}, y_{z1}, \ldots, y_{zn})$;

where $A, B, \ldots$ are geographical connections to neighbouring logical objects and G – non-geographical connections, as shown in Fig. 2.

Having the data identified and organised, it is possible to begin construction of the algorithm. At first it is necessary to choose the notation method.

2.2. Notation

To choose the notation method for the constructed algorithm it is necessary to ensure compact form and unambiguity.

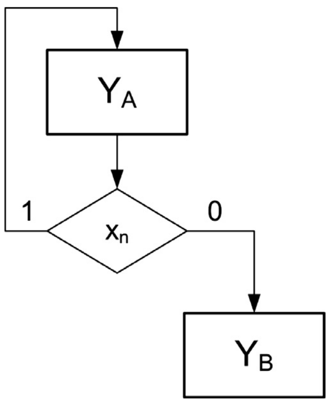


Fig. 3. Graphical algorithm scheme.

For manual algorithm processing the graphical form of the notation may be more user-friendly, however the modern design tools allow us to generate the specification automatically, so other solutions can be more profitable.

The *logical algorithm scheme* is a formal description method, which has a purely mathematical form, compared to the *graphical algorithm scheme*, which is basically a drawing as in Fig. 3.

As described in [10], states (elements of the Y set) represent values of the output variables according to chosen coding convention. The input variables (X set) condition the next step to be executed. If the examined condition equals ‘1’, the next step is the operation on the right. Alternatively, if the condition equals ’0’, then the next step is determined by the number over the arrow. Each arrow pointing down has a unique number and marks the transition end. The transition placed after the ω sign is executed unconditionally.

The available symbols are listed below:

- Y_k – output vector,
- x_k – input variable,
- X_k – input vector, $X_k = (x_1, x_2, \ldots, x_n)$,
- $\overset{i}{\uparrow}$ – condition check, if the preceding expression equals ‘1’ the next action is described by the element on the right side, if it equals ‘0’, a transition to the place marked with arrow number i is executed,
- $\overset{i}{\downarrow}$ – end point for transitions marked with i,
- $\omega\overset{i}{\uparrow}$ – unconditional transition (always-false condition).

The example shown in Fig. 3 can be expressed in logical algorithm scheme as $\overset{1}{\downarrow} Y_A \overline{x_n} \overset{1}{\uparrow} Y_B$.

The use of logical algorithm schemes was proposed during the early development of the description methods for railway in Poland (see [1]). Unfortunately at this time there was no design tool which would allow us to use this notation directly for the system implementation. Today such tools are available; therefore this approach can be again a profitable solution.

2.3. Algorithm construction method

To begin constructing the algorithm, several statements are proposed.

Proposition 1. *The railway interlocking logic model is a set of interconnected logical objects, each one concurrently executing a set of algorithms. Each logical object E_i belongs to a set of possible object types. Each algorithm A_i is designed to execute one functionality. Every functionality F_i belongs to a set of functionalities foreseen for the particular adaptation of the system.*

This means, that the interlocking system have to be split in two different perspectives, as shown in Fig. 4. One is the topological construction perspective, and the second is the functional approach.

Proposition 2. *Each functionality consists of steps. Each algorithm is constructed basing on executed function steps and amount of possible algorithm variants.*

This means that each algorithm is also decomposed in two perspectives. Steps of each functionality should be arranged according to the customer guidelines. Variants depend on functionality type and object properties and generic rules. Finally this results in the general algorithm structure given below.

$$\begin{aligned}
&A_i = \overset{0}{\downarrow} Y_A \overline{X_{ab1}} \overset{11}{\uparrow} \overline{X_{ab2}} \overset{12}{\uparrow} \ldots \overline{X_{abw}} \overset{1w}{\uparrow} \omega \overset{0}{\uparrow} \\
&\overset{11}{\downarrow} Y_{A1} X_{ab1} \overset{11}{\uparrow} \overset{21}{\downarrow} Y_{B1} X_{bs1} \overset{21}{\uparrow} \ldots \overset{s1}{\downarrow} Y_{S1} X_{sa1} \overset{s1}{\uparrow} \omega \overset{0}{\uparrow} \\
&\overset{12}{\downarrow} Y_{A2} X_{ab2} \overset{12}{\uparrow} \overset{22}{\downarrow} Y_{B2} X_{bs2} \overset{22}{\uparrow} \ldots \overset{s2}{\downarrow} Y_{S2} X_{sa2} \overset{s2}{\uparrow} \omega \overset{0}{\uparrow} \\
&\quad \vdots \\
&\overset{1w}{\downarrow} Y_{Aw} X_{abw} \overset{1w}{\uparrow} \overset{2w}{\downarrow} Y_{Bw} X_{bsw} \overset{2w}{\uparrow} \ldots \overset{sw}{\downarrow} Y_{Sw} X_{saw} \overset{sw}{\uparrow} \omega \overset{0}{\uparrow}
\end{aligned} \tag{1}$$

where $A, B, \ldots, S$ are the states, s is the number of states and w is the number of algorithm variants.

The above notation is equal to the graph shown in Fig. 5.

Proposition 3. *Functionalities can be local (involving one logical object) or complex (involving two or more logical objects)*

An example of local functionality is blocking point movement or preventing track from being entered by a train. Complex functionalities include route setting, signalling or level-crossing activation by an approaching train.

Using the method described above, the algorithm design process has been decomposed in a way that it is possible to create the complete specification only by identifying single dependencies. The example of such dependency is: *Condition for transition from setting to locking state in a route A–B in a point logical object.*

3. Results

To illustrate the proposed method, a set of logical objects was specified. By creating descriptions of basic objects and functions, a prototype interlocking logic was developed and successfully tested.

An algorithm representing one of the functionalities of point logical object is presented as an example. The algorithm A_{ZJ} is responsible for route execution functionality.

The typical route life cycle, based on [9], consists of Calling, Setting, Locking, and Releasing stages. After adding Rest state as the default stage, an example of transition requirements were described in Table 1.

The point modelled as a logical object has three geographical connections. Considering the two possible point positions, and two possible route directions, there are four available routes for this logical object, as presented in Fig. 6. This results in four variants of the algorithm.

Using the construction method given above it was possible to create the algorithm described in Eq. (2).

$$\begin{aligned}
&A_{ZJ} = \overset{0}{\downarrow} Y_S \overline{X_{sn1}} \overset{11}{\uparrow} \overline{X_{sn2}} \overset{12}{\uparrow} \overline{X_{sn3}} \overset{13}{\uparrow} \overline{X_{sn4}} \overset{14}{\uparrow} \omega \overset{0}{\uparrow} \\
&\overset{11}{\downarrow} Y_{W1} X_{wn1} \overset{11}{\uparrow} \overset{21}{\downarrow} Y_{N1} X_{nu1} \overset{21}{\uparrow} \overset{31}{\downarrow} Y_{U1} X_{uz1} \overset{31}{\uparrow} \overset{41}{\downarrow} Y_{Z1} X_{zs1} \overset{41}{\uparrow} \omega \overset{0}{\uparrow} \\
&\overset{12}{\downarrow} Y_{W2} X_{wn2} \overset{12}{\uparrow} \overset{22}{\downarrow} Y_{N2} X_{nu2} \overset{22}{\uparrow} \overset{32}{\downarrow} Y_{U2} X_{uz2} \overset{32}{\uparrow} \overset{42}{\downarrow} Y_{Z2} X_{zs2} \overset{42}{\uparrow} \omega \overset{0}{\uparrow} \\
&\overset{13}{\downarrow} Y_{W3} X_{wn3} \overset{13}{\uparrow} \overset{23}{\downarrow} Y_{N3} X_{nu3} \overset{23}{\uparrow} \overset{33}{\downarrow} Y_{U3} X_{uz3} \overset{33}{\uparrow} \overset{43}{\downarrow} Y_{Z3} X_{zs3} \overset{43}{\uparrow} \omega \overset{0}{\uparrow} \\
&\overset{14}{\downarrow} Y_{W4} X_{wn4} \overset{14}{\uparrow} \overset{24}{\downarrow} Y_{N4} X_{nu4} \overset{24}{\uparrow} \overset{34}{\downarrow} Y_{U4} X_{uz4} \overset{34}{\uparrow} \overset{44}{\downarrow} Y_{Z4} X_{zs4} \overset{44}{\uparrow} \omega \overset{0}{\uparrow}
\end{aligned} \tag{2}$$

Having prepared the general layout it is now necessary to identify conditions for each transition (Table 2).

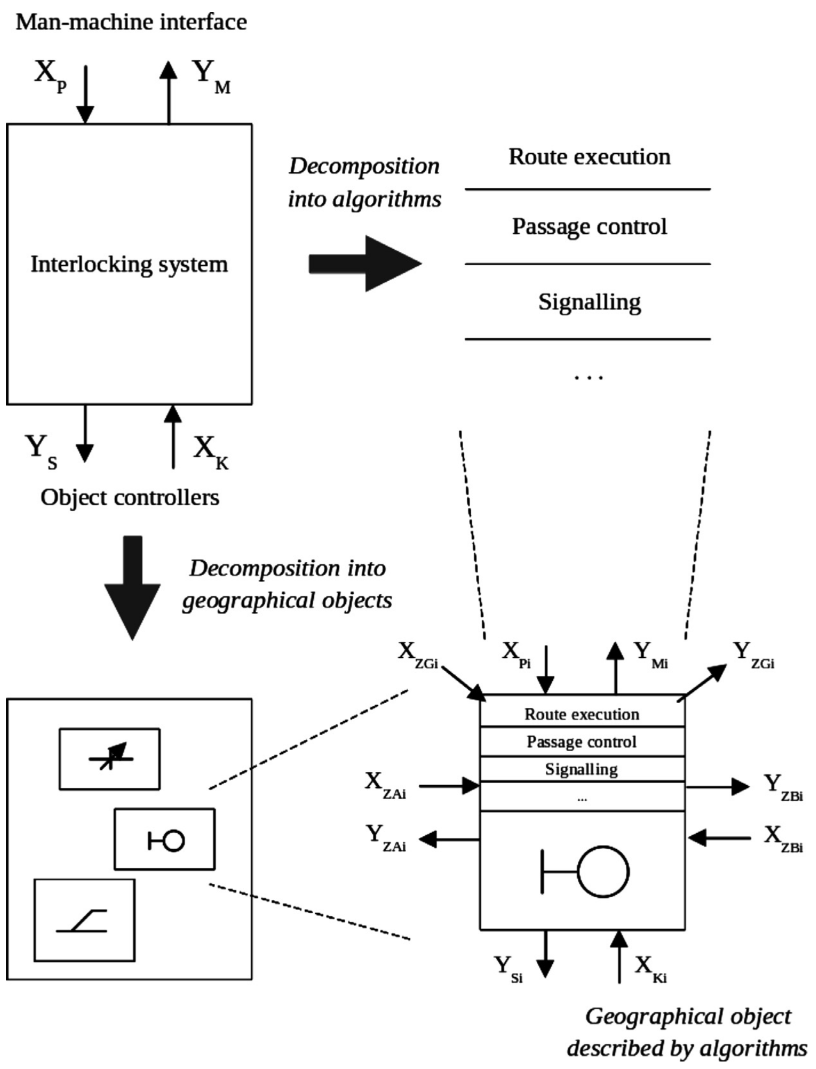


Fig. 4. Decomposition method.

Table 1
Route life cycle in point logical object.

Transition	Condition
Rest (S) to Calling (W)	Point position selected by the dispatching system
Calling (W) to Setting (N)	Information about neighbour objects' readiness (from the topological connections)
Setting (N) to Locking (U)	Required position achieved
Locking (U) to Releasing (Z)	Track circuit occupancy
Releasing (Z) to Rest (S)	Train passage in required direction detected

Table 2
Exemplary transition conditions.

Transition	Condition
Rest (S) to Calling (W1)	$X_{sw1} = x_{p0}$
Calling (W1) to Setting (N1)	$X_{wn1} = x_{za0}$
Setting (N1) to Locking (U1)	$X_{nu1} = x_{k0}$
Locking (U1) to Releasing (Z1)	$X_{uz1} = \overline{x_{k1}}$
Releasing (Z1) to Rest (S)	$X_{zs1} = x_{k1} \wedge \overline{x_{zb1}}$

Using this method to describe a basic set of functionalities in the most common logical object types it was possible to create a set of algorithms. The algorithms could be then specified in HDL, implemented and tested.

Today the railway industry looks for new solutions allowing us to increase the number of executed functions, but without changing the speed of the system. In software solutions this means that the amount of code lines should be kept on the same level. Because of the ongoing implementation of the European Rail Traffic Management System (ERTMS) and introducing local and regional control centres, new solutions, which could replace the microprocessor technology used now, are discussed. As proposed earlier in this paper, every logical object should execute required algorithms concurrently. The microprocessor implementation of the interlocking system does not allow concurrent data processing, instead providing calculation cycles of a fixed length. Every cycle processes the necessary data, for example logical objects one by one. Since the areas controlled by a single interlocking become larger, this becomes more problematic.

One of the discussed solutions is the use of hardware solutions, such as Field Programmable Gate Arrays (FPGAs), taking the latest industry achievements into account (see also [2]).

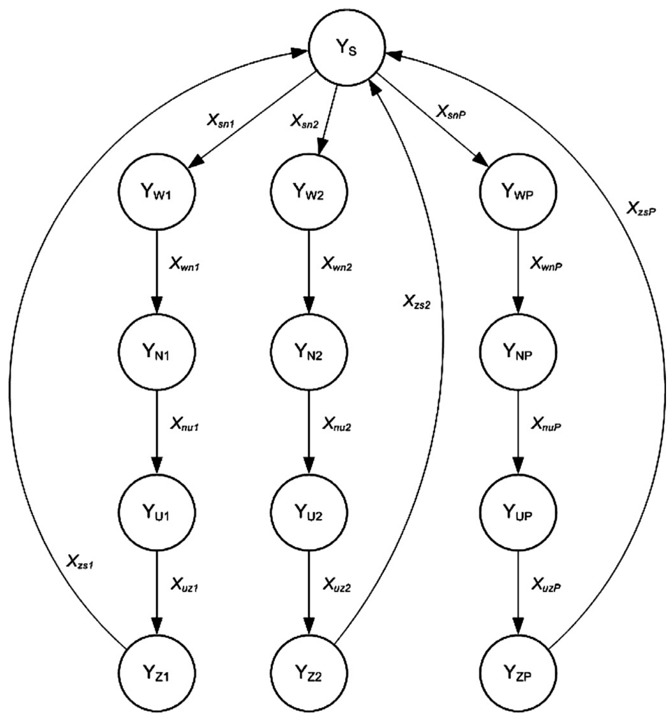


Fig. 5. Algorithm graph created from the logical algorithm scheme.

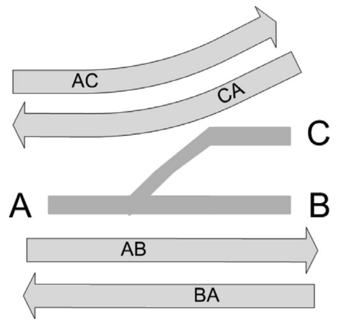


Fig. 6. Routes available for point logical object.

3.1. Specification

The algorithm construction method shown above can be easily used for FPGA implementation. To be sure that the selected specification method utilises all of the latest achievements available on the market, it is necessary to use the world standard solutions. Because of this, one of the most popular hardware description languages – VHDL, was used in the proposed method. The VHDL (*VHSIC Hardware Description Language*) is widely used in many fields of electronic devices market. Invented for military purposes in the 80’s, it is still being developed, with the latest revision issued in 2008.

When selecting tools for the design process, it is necessary to look for solutions that can be used on every stage of product development. For the method development the *Active-HDL* integrated design environment is used. This is one of the most popular environments for specification, implementation and verification of programmable devices.

The *Active-HDL* allows us to prepare the specification using three editors.

The *FSM* editor allows us to prepare the data in the graphical environment using finite-state machines. These are easily achievable from the logical algorithm schemes.

Considering every value of the output vector Y_i as a state, and every value of the condition check X_i as a transition, it is easy to create a graph. Inserting the graph into the FSM editor (as shown in Fig. 7) allows us to automatically generate HDL code for this algorithm and use it further in the design process. Additionally it is possible to use features like default states, trap states or transition priorities, which allow us to enhance the specification.

The *BDE* editor allows us to create multi-level hierarchy using blocks. Every part of the VHDL code can be presented as a single block and interconnected with the others. A group of blocks can also be presented as one, creating a higher hierarchy level. This function fits perfectly to the topological principle intended to use in the proposed method. It enables to easily design logical object connections in *WYSIWYG* technique.

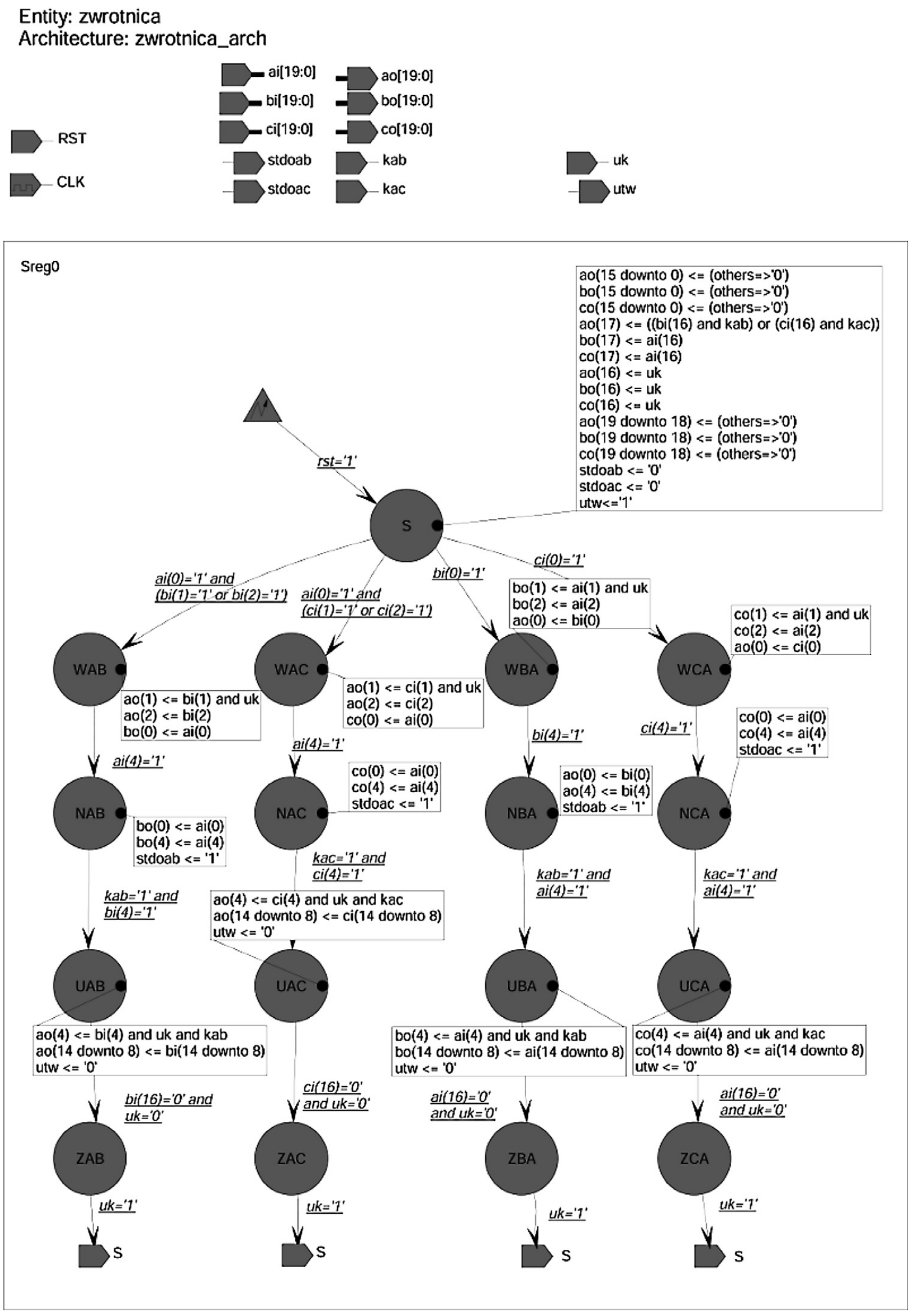
In the proposed method this editor is used for merging the algorithms into logical objects, and for creating the topology of the station. As presented in Fig. 8 this allows us to create a station topology specification which is easy to design and verify.

Each of the graphical editors allow automatic HDL code generation, as shown below for the A_{ZJ} algorithm.

```
-----------------------------------
-- Machine: A_ZJ
-----------------------------------
A_ZJ_machine: process (CLK, rst)
begin
 if RST=’1’ then
  A_ZJ <= INIT;
  -- Set default values for outputs,
  -- signals and variables
 elsif CLK’event and CLK = ’1’ then
  case A_ZJ is
   when INIT =>
    A_ZJ <= S;
   when S =>
    if Xp(0)=’1’ then
     A_ZJ <= W1;
    end if;
   when W1 =>
    if XZa(0)=’1’ then
     A_ZJ <= N1;
    end if;
   when N1 =>
    if Xk(0)=’1’ then
     A_ZJ <= U1;
    end if;
   when U1 =>
    if Xk(1)=’0’ then
     A_ZJ <= Z1;
    end if;
   when Z1 =>
    if Xk(1)=’1’ and XZb(1)=’o’ then
     A_ZJ <= N;
    end if;
--vhdl_cover_off
   when others =>
    null;
--vhdl_cover_on
  end case;
 end if;
end process;
```

The *HDL* text editor can be then used to review and adjust the code. It provides all necessary functions like code completion, syntax checking and code assistant. The designer can then focus on legal and functional aspects of creating the particular algorithm instead of creating the code itself.

To ensure the correctness of the specification, testing is crucial at every stage of the development. Testing of the created specification is one of the most difficult and time-consuming parts of the



```
ao(15 downto 0) <= (others=>'0')
bo(15 downto 0) <= (others=>'0')
co(15 downto 0) <= (others=>'0')
ao(17) <= ((bi(16) and kab) or (ci(16) and kac))
bo(17) <= ai(16)
co(17) <= ai(16)
ao(16) <= uk
bo(16) <= uk
co(16) <= uk
ao(19 downto 18) <= (others=>'0')
bo(19 downto 18) <= (others=>'0')
co(19 downto 18) <= (others=>'0')
stdoab <= '0'
stdoac <= '0'
utw<='1'
```

```
bo(1) <= ai(1) and uk
bo(2) <= ai(2)
ao(0) <= bi(0)
```

```
co(1) <= ai(1) and uk
co(2) <= ai(2)
ao(0) <= ci(0)
```

```
ao(1) <= bi(1) and uk
ao(2) <= bi(2)
bo(0) <= ai(0)
```

```
ao(1) <= ci(1) and uk
ao(2) <= ci(2)
co(0) <= ai(0)
```

```
co(0) <= ai(0)
co(4) <= ai(4)
stdoac <= '1'
```

```
co(0) <= ai(0)
co(4) <= ai(4)
stdoac <= '1'
```

```
ao(0) <= bi(0)
ao(4) <= bi(4)
stdoab <= '1'
```

```
bo(0) <= ai(0)
bo(4) <= ai(4)
stdoab <= '1'
```

```
ao(4) <= ci(4) and uk and kac
ao(14 downto 8) <= ci(14 downto 8)
utw <= '0'
```

```
ao(4) <= bi(4) and uk and kab
ao(14 downto 8) <= bi(14 downto 8)
utw <= '0'
```

```
bo(4) <= ai(4) and uk and kab
bo(14 downto 8) <= ai(14 downto 8)
utw <= '0'
```

```
co(4) <= ai(4) and uk and kac
co(14 downto 8) <= ai(14 downto 8)
utw <= '0'
```

Fig. 7. Specification of the algorithm in the FSM editor.

project lifetime. This is why it is most efficient to use tools that allow us to have full control of the simulation process, having ability to automate the process in the same time. Many companies develop proprietary tools to achieve this, but it is also an expensive solution.

However, it is possible to use the tools available on the market, which ensures that we can use the latest available technology. In environment like the *Active-HDL* the simulation and testing process can be performed and automated in various ways.

The set of internal logic simulators allow us to have full control of the process. Not only it is possible to run functional simulations, but also time simulations, where the speed parameters of the selected device type are taken into consideration. In hardware solution like FPGAs, the time required to execute a function depends entirely on signal propagation time. Therefore, unlike in the software solutions, it is possible to calculate exact device response time.

The simulation process can be documented as waveforms, as in Fig. 9, which allow us to observe value of every variable in a specified moment in time. Creating waveform documentation of the time simulation is crucial for identifying possible hazards and other time dependencies. It is also possible to observe the simulation in the graphical editors, as for example the FSM editor in Fig. 10 or BDE editor in Fig. 11.

Automatic testing is beneficial for project costs and process reliability. The simulation scenarios can be prepared in the form of

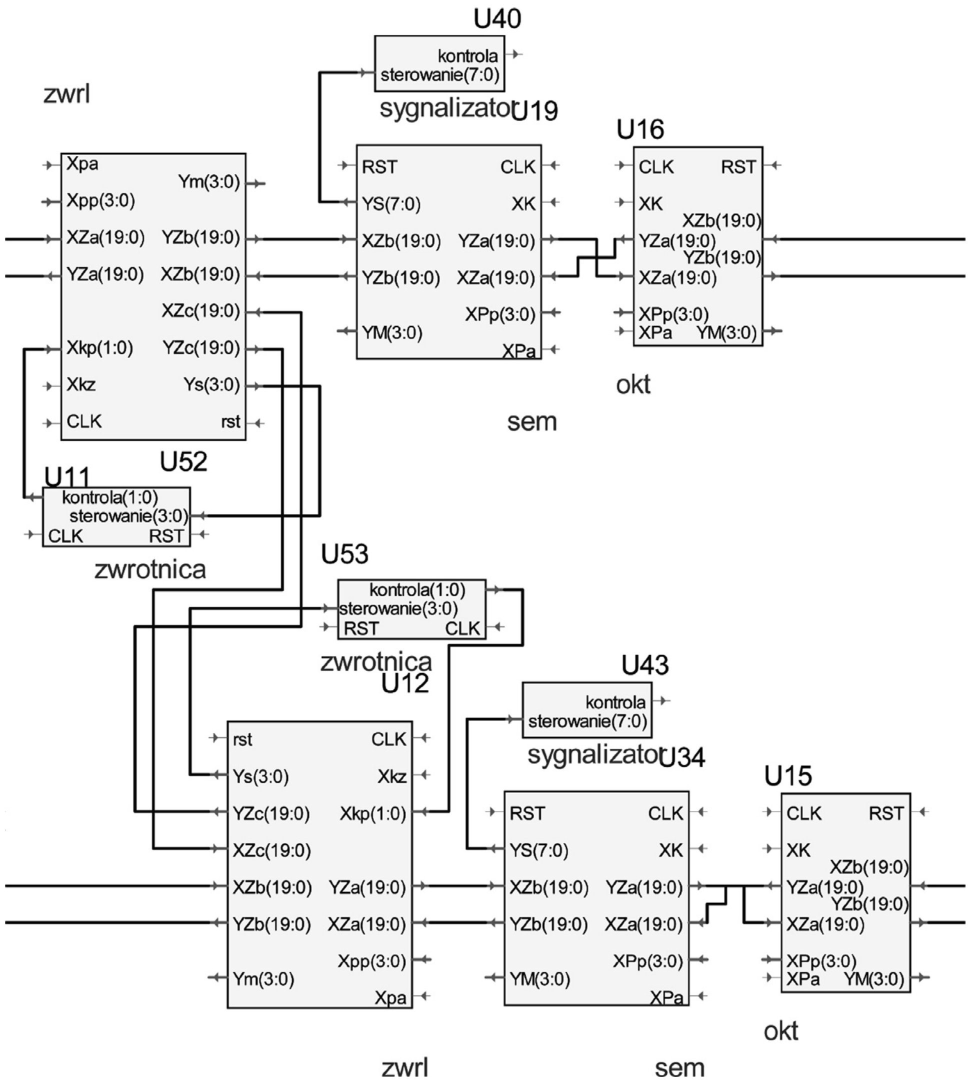
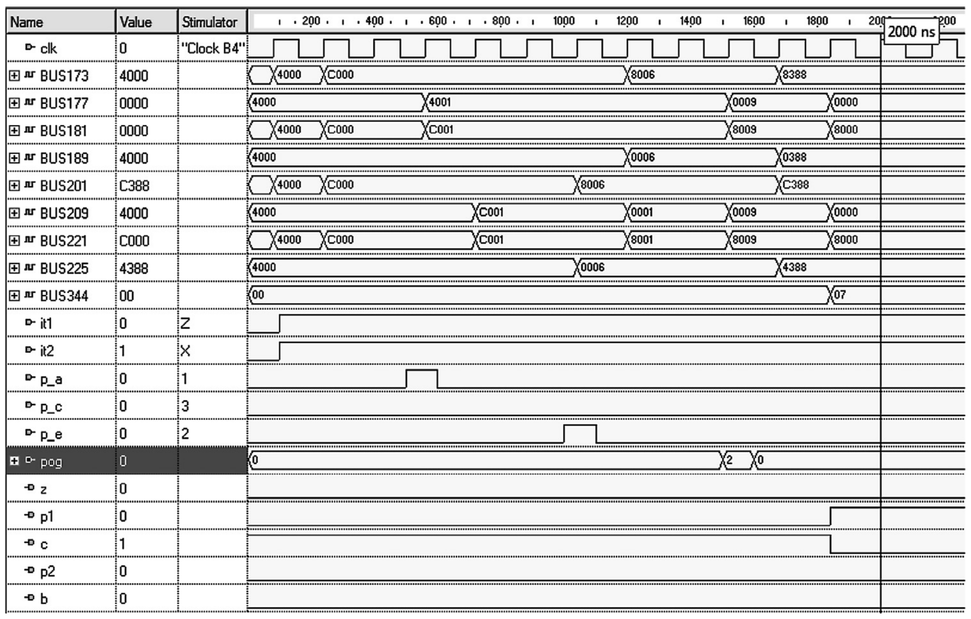


Fig. 8. Logical objects in BDE editor.



Name	Value	Stimulator
clk	0	"Clock B4"
BUS173	4000	
BUS177	0000	
BUS181	0000	
BUS189	4000	
BUS201	C388	
BUS209	4000	
BUS221	C000	
BUS225	4388	
BUS344	00	
it1	0	Z
it2	1	X
p_a	0	1
p_c	0	3
p_e	0	2
pog	0	
z	0	
p1	0	
c	1	
p2	0	
b	0	

Fig. 9. Waveform.

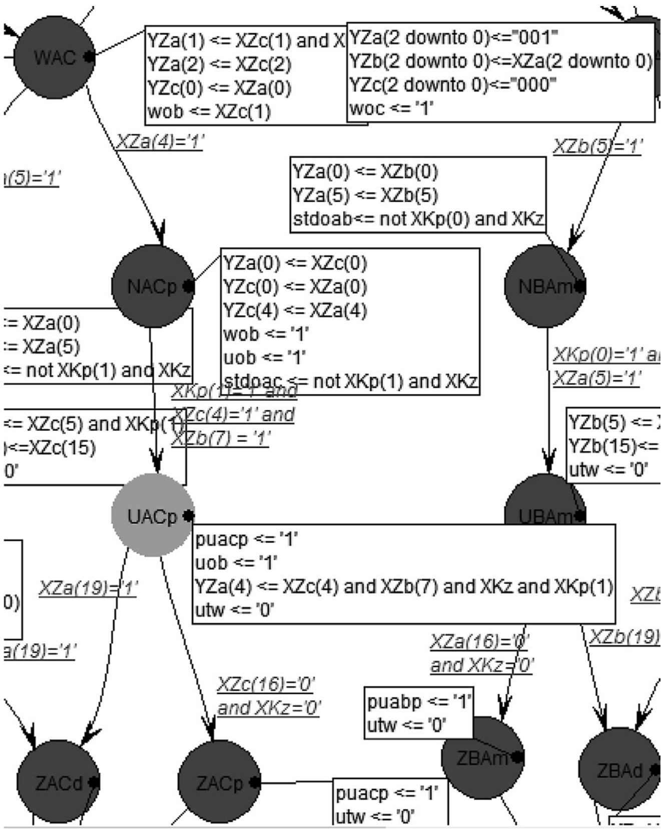


Fig. 10. Simulation preview in FSM editor. The active state is marked with a lighter colour.

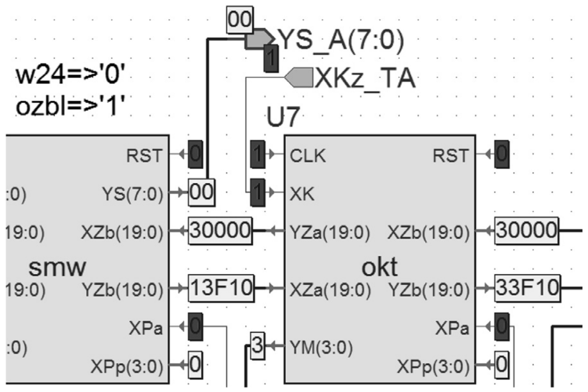


Fig. 11. Simulation preview in BDE editor. Values of variables and vectors are displayed.

Active-HDL Macro Language files. These simple script files allow us to execute most of the *Active-HDL* functions in the command-line environment, for instance changing the input variable values, running simulation for a specified time, compiling. It is therefore possible to automate the testing process by generating scripts based on the customer requirements, as described in our previous papers (see [5]).

It is also possible to generate test scenarios for the FSM designs automatically. These *Testbenches* can be prepared using different scenarios. For example the test can automatically cover every transition in a graph. This makes the testing process simple and efficient.

Table 3
Safety levels according to IRSE (see [3]).

Safety level	λ	Description	Subsystem
4	10^{-11}	Very high	Interlocking
3	10^{-9}	High	Propulsion
2	10^{-7}	Mid	Dispatching
1	10^{-5}	Low	Information
0		Not safety-related	Management

3.2. Implementation

For every new solution in railway interlocking market it is necessary to calculate the achieved reliability and safety levels and to check if the results conform to the norms. The safety of the HDL code can be compared to the code used in the integrated circuits and ASIC design process. In both cases the code is used to describe and implement the device, rather than to be executed on the machine.

Since the mentioned documents does not provide guidelines regarding the programmable devices, it can be assumed that such appliances have to confirm to the hardware requirements of the mentioned documents. Nevertheless, the VHDL language used for the specification is derived from Ada - language widely used in safety-critical solutions.

When publishing reliability reports, FPGA manufacturers use the FIT (Failures In Time) parameter to describe the reliability of the devices. One of the top FPGA suppliers, Xilinx, calculates the parameter using Eq. (3) [11].

$$Failure\ Rate = \frac{\chi^2 10^9}{2NTA} \tag{3}$$

where:
χ^2 = Chi-squared value at a desired confidence level and (2f+2) degrees of freedom, where f is the number of failures,
N = number of devices,
T = number of hours,
A = acceleration factor.

Capacity of modern FPGA devices allow us to implement object controllers or interlocking systems as a single SoC design. Comparing the requirements proposed by IRSE – Institution of Railway Signal Engineers (see Table 3) with the FIT parameters of modern devices (see [11]) it can be assumed, that a single programmable device can conform to safety level 3. Only for level 4 the hardware redundancy has to be applied.

Using hardware redundancy for one-chip devices it is possible to create $k > n$ (where $(n > 2)$) fault-tolerant solutions. One other quality of specialised hardware solutions is their speed, which allows us to implement self-diagnostic algorithms without a significant impact on system reaction times. It will be therefore possible to detect persistent or spontaneous failures even without hardware redundancy [3].

4. Conclusions

The proposed algorithm construction method allows us to decompose the system in a unified manner, and prepare the specification using elementary conditions. Positive verification of the created model proves that the proposed specification and verification method succeeded in creating the interlocking logic. Having this model as a base, it will be possible to create an implementation in programmable devices, which will allow us to prove safety and reliability of this approach. Earlier analysis show that the results can be promising [4].

The wide range of programmable devices provide a solution well adjusted to the customer needs. The speed of hardware solutions could allow us to conform to the highly demanding needs of modern interlocking. Using the commercial off-the-shelf design environment like the presented Active-HDL suite allows us to minimise costs of the internal design tools development.

The presented approach to railway interlocking logic design, together with the use of modern hardware solutions, can be a profitable solution for the future development of railway equipment.

References

[1] S. Apuniewicz, Układy przekaźnikowe w automatyce zabezpieczenia ruchu kolejowego, WPW, Warszawa, 1969.
[2] J. Borecký, P. Kubalík, H. Kubátová, Reliable Railway Station System based on regular structure implemented in FPGA, in: Proceedings of 12th EUROMICRO Conference on Digital System Design, Patras, 2009, pp. 348–354.
[3] P. Kawalec, Analiza i synteza specjalizowanych układów modelowania i sterowania ruchem w transporcie (Analysis and synthesis of specialised modelling and traffic control systems in transport, in Polish), Prace Naukowe – Transp. 68 (2009).
[4] P. Kawalec, D. Koliński, J. Mocki, Zastosowanie programowalnych struktur logicznych w urządzeniach sterowania ruchem kolejowym, Probl. Kolej. z. 140 (2005) 66–88.
[5] P. Kawalec, M. Rżysko, Weryfikacja równań zależnościowych z wykorzystaniem symulatorów logicznych na przykładzie zastosowania pakietu Active-HDL (Verification of interlocking equations using logic simulators in Active-HDL environment, in Polish), Tech. Transp. Szyn. 10 (2013) 1587–1594.
[6] P. Kawalec, M. Rżysko, Algorytmiczne podejście do projektowania logiki zależnościowej w systemach sterowania ruchem kolejowym (The algorithmic approach to railway interlocking logic design, in Polish), Pomiar., Autom., Kontrola 10 (2014) 826–828.
[7] U. Maschek, Sicherung des Schienenverkhers, Springer Vieweg, Wiesbaden, 2012.
[8] E. Miksza, W. Olendrzyński, A. Zubkow, Zblokowany system sterowania ruchem kolejowymna stacjach typu IZH 111, WKiŁ, Warszawa, 1979.
[9] G. Theeg, S. Vlasenko, Railway signalling & interlocking, Eurailpress, Hamburg, 2009.
[10] W. Traczyk, Układy cyfrowe. Podstawy teoretyczne i metody syntezy, Wydawnictwa Naukowo-Techniczne, Warszawa, 1982.
[11] Xilinx., Device reliability report. First half, 2015, http://www.xilinx.com/support/documentation (accessed 28.12.15).

Piotr Kawalec M.Sc.Eng. in electronics, Dr.Eng. in elements and devices for calculation technology and control systems. Currently works at Warsaw University of Technology, Faculty of Transport. Professional and scientific interests include modelling and designing traffic control devices using specialised electronic colutions.

Marcin Rżysko In 2013 became M.Sc.Eng. in railway traffic control at the Warsaw University of Technology, Faculty of Transport. Currently works as an application engineer at Bombardier Transportation (Rail Engineering) Poland. Professional and scientific activities include designing and verification of railway interlocking logic and site-specific applications.