Abstract — The paper presents one implementation possibility of simple control network using event driven software gate. The paper first reviews the basics of event-driven software gates and the components of event-driven control network. After introduction basics of software gates and event-driven control network the paper presents the dynamic behavior of simple control network during initialization phase and normal operation.

I. INTRODUCTION

Distributed control systems are widely used in home or building automation or in telemetry systems. The distribution systems communicate mostly with messages. The communication between nodes in distribution control networks can be implemented with wired or wireless solution.

My research area is the communication solution model in distributed control network. In this paper the reader can read about one type of implemented model usable in distributed networks. In this model the message contents are restricted as possible. For example in represented model the message do not contains data about identification of sender. It means the receiver object do not now have any information about sender, only about new value of changed output. The mathematical model of this massage base control system is can be found in [7] and in [8] literature.

II. EVENT-DRIVEN PROGRAMMING MODEL

Event-driven systems require a distinctly different way of thinking than traditional sequential programs [9]. The event-driver program does not wait for an external event, the program react to events [5]. The program must be designed in if-then conditions.[6] The event driven model based programs can be work as parallel systems. The event driven system can be works as data-flow model base systems. [1],[4],[12]

Before analyzing the event-driven programming model let's look at the concept of the events. The events are signals from the outside of world in generally. Events describe dynamic changes of inputs. Events can be external – hardware – events such as a button press or a tank becomes full or internal – software – events such as time expired, result of mathematical calculation, changing state of software resources. [6]

In event-driven model based program contains active and passive objects in term of behavior.

The passive objects are running only when they receive some messages. The passive objects can generate messages only, when they have been received some message. [6]

The active objects are running always. The active objects can generate messages without receiving messages. The active objects are usually hardware supported interrupt service routines. [6]

Programs written along traditional event-driven model include automatic control of processor power consumption. If the message comes when the event queue is empty the response time of program is the message processing plus the answer generation time.[6]

III. CONSTRUCTION OF EVENT DRIVEN COMBINATIONAL LOGIC NETWORKS

Some distributed systems are not easy to control with programs based on traditional control models. A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal. There are many alternatives for the message passing mechanism, including RPC-like connectors and message queues. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components. [3]

Wireless or wired automated productions systems [11] or wireless home automation solutions can be implemented with mash network [10]. This type of network contains active and passive end device members. Usually active member are actuators passive members are sensors. [2]

The presented event driven combinational logic model in this article is based on active object computing model. The active object computing model addresses most problems of the traditional event-driven architecture, retaining its good characteristics.[9] The active objects are individual task functions with own event queues. Each task starts running with initialization after tries to get event message from its event-queue. [6] Only newly received message cause activity of object. If object does not have message to process the object is in sleep mode. The flowchart of active object base task functions can be seen on Error! Reference source not found.
The objects are connected together with event message queues. Some tasks responsibility to generates messages to the queues. These tasks are inputs handling tasks. The control tasks receive and generate messages. The outputs handling tasks read they own message queues and sets the value of hardware outputs based on information in received message.

Messages used in represented model are composed from two parts:

- **Type of message** – The message type can be initialization or change notification message. Initialization messages: The object sends initialization messages as first message. The object can send initialization message only once. Change notification messages: The object sends change notification messages when its output value has been changed compared to the last sent message.[7]

- **Value of message** – The value of message can be true or false. The true means the object's output has been changed to true. The false means the object's output had been changed to false.[7]

IV. COMPONENTS OF EVENT-DRIVEN CONTROL NETWORK

The presented event-driven control model is composed from four types of objects:

- Input objects
- Control objects
- Output objects
- Message queues (to connect input, control, output objects)

The input and output objects are connected directly to the hardware. The input and output objects are connecting the real hardware detection and intervention with the software model behavior.

- **The input objects** are active objects in terms of behavior. The objects are able to send messages only. They can not receive messages.

  The input objects can send message in two cases:
  
  - During initialization process – the object reads the value of the corresponding input and sends initialization message to the corresponding receiver objects.
  
  - During normal operation – if the value of corresponding input has been changed, then the object will send change notification messages to the receiver objects.

- **The output objects** are passive objects in terms of behavior. The objects are able to receive messages only. They can not send messages. The objects are setting the value of the controlled outputs based on incoming messages. The output object does not distinguish between initialization and change notification messages. The object does not have initialization phase, so the first message will set up the value of output.

- **The control objects** are passive objects in terms of behavior. The objects are able to receive and send messages. The description and implementation of control objects, like event-driven software AND, OR, XOR and universal gates are not part of this article. You can get more information about the distance value based event-driven software gate in [7],[8] literatures.

The message queues links input, output and control objects together. The message queues are inseparable part of output and control objects. The sizes of the message queues are determining the immunity to message congestion of implemented control network.

The Figure 2. represents the architecture of event-driven control network. The input objects are placed on left, the output objects are placed on right side. In the middle section of the figure are the control objects. The control objects and output objects have input message queues, but do not have output message queues. Directions of the messages are illustrated by the dashed arrows. The generator of message knows the destination of the messages. The receiver does not know anything about the identity of the sender object. It knows only the content of received message: the value and the type of the received message.

V. CREATING SIMPLE DISTRIBUTED CONTROL NETWORK

The practical implementation of theoretical control network model shows the current chapter. The structure and practical realization of event-driven software gates and the whole event-driven control network is implemented in C language.

Software and hardware devices used for the testing of the event-driven control network model:

- Microchip Explorer development board with PIC24FJ128GA010 microcontroller
- Microchip MPLAB C Compiler for PIC24 and dsPIC version 3.31
- FreeRTOS operating system version 7.0.0 to implement active object model with tasks in microcontroller environment
- MPLAB Integrated Development Environment version 8.43 to simulate hardware and create logic analyzing graphs.
To demonstrate behavior of event-driven software gate based distributed control network I created a simulation environment with AND gate. The implemented control network with AND gate behavior is implemented with four independent task.

- **Two input object task** – The tasks monitor the corresponding input pin. If the pin value has been changed the object will send message to control object.
- **One control object task** – The task is passive object, it will run only when they got a message. The task is releasing AND gate based on [7],[8] literatures.
- **One output control task** – The task is passive object. The object is responsible for the adjustment of output value based on received messages.

Figure 6. represents the implemented control network with behavior of AND logical gate. Framing denotes that which message queue belongs to witch object. The dashed arrows illustrate the direction of the flow of messages.

**Event structure**

The objects use massages for communication. The messages are implemented with data structure. Source code presented in Figure 3. demonstrates the implementation of message structure written in C language.

```c
typedef struct
{
    BOOL bValue;
    BOOL bIsInitValue;
} BOOL_EVENT_TYPE;
```

**Input object**

Source code presented in Figure 4. shows the implementation of input object task-function. The task implements active object, so is working always. In the case when the monitored input is changed, it sends a message to the corresponding object or objects.

```c
void vInputTask( void *pvParameters )
{
    INPUT_PARAMETER xInputParameter = *(INPUT_PARAMETER*)pvParameters;
    BOOL_EVENT_TYPE xEventToSend, xLastEvent;
    portBASE_TYPE xInputMask = 1 << xInputParameter.xInputNum;
    portTickType xLastWakeTime = xTaskGetTickCount();
    xEventToSend.bIsInitValue = true;
    for( ;; )
    {
        xEventToSend.bValue = (mReadButtons() & xInputMask) ? true : false;
        /* Sending intialization or changing message*/
        if ((xEventToSend.bIsInitValue == true) ||
            (xEventToSend.bValue != xLastEvent.bValue))
        {
            /* Send Event to MessageQueue */
            vSendEventMsg( &xInputParameter.xGateParameter,
                           &xEventToSend);
            xLastEvent = xEventToSend;
            xEventToSend.bIsInitValue = false;
        }
        taskYIELD();
    }
}
```

**Output object**

Source code presented in Figure 5. shows the implementation of output object task-function. The task implements passive object, so is running only when the task has been received a message. When the task receives some message the task refresh the value of its corresponding output pin.

```c
void vOutputTask( void *pvParameters )
{
    portBASE_TYPE xOutputNum = ((OUTPUT_PARAMETER*)pvParameters)->xOutputNum;
    GATE_PARAMETER xGateParameter = ((OUTPUT_PARAMETER*)pvParameters)->xGateParameter;
    BOOL_EVENT_TYPE xReceivedEvent;
    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Receive Event from MessageQueue */
        vReceiveEventMsg( &xGateParameter, &xReceivedEvent );
        if( xReceivedEvent.bValue )
        {
            mTurnOnLED( xOutputNum );
        }
        else
        {
            mTurnOffLED( xOutputNum );
        }
    }
}
```

**Control object**

In the implemented control network the control object is an event-driven software gate. The theoretical description of event-driven software and the implementation of event-driven software gate task-function can be found in [7],[8] literature.

The task implements passive object, so is running only when the task has been received a message. In case when the task receives message refresh the value of its output. When the output value of the gates has been changed or the received message is a firs message, the task will send message to the corresponding objects.

**Compilation of control network**

Source code presented in Figure 7. shows the creation of task and their connecting message queues. Parameters of object function describe the behavior of objects and the connection between individual objects.

---

**Figure 3.** Implementation of message structure

**Figure 4.** Implementation of input object task-function

**Figure 5.** Implementation of output object task-function

**Figure 6.** Structure of event driven control network with AND gate behavior

**Figure 7.** Implementation of control network task-function
It can be observed in the figure the initialization message wave of the control network. The network initialized itself with the initialization messages first. On figure can be found in rectangle the object initialization message process wave. On the figure can be seen that the two input object is generating messages on system startup that does not caused change on input lines. The two initialization message was sent to the AND gate process. The AND gate process sends initialization message to the output object as a result of the first incoming message to itself. By sending a message to the output object the initialization process is complete. On figure it can be also observed the message flow of change notification messages.

Simultaneous changes of more then one input variable

Processing of events occurring at same time is implementation dependent and not specified. In preemptive multitasking environment order of the processing of the objects depends from the priority levels of the tasks.

In case when the input objects are in same priority level, then the processing of the messages coming in same time are random. The processing of simultaneously changing inputs is visible on Figure 11. The symbols of the lines are same like in Figure 9. In the figure can be seen that the control network react differently for the same input situation.
VII. SUMMARY

With programming model presented in this paper is possible to create event-driven distributed combinational logic based control networks. The main advantage of model is the software gates do not need to know the values of all input at given time moment to calculate they output values. The software model could be a good solution for controlling wireless distributed sensor networks. Using presented model the sensors and control unit do not need all time communication connection, enough if the sensors send initialization a change notification messages to control unit.

REFERENCES


[5] Dr. Kondorosi Károly, Dr. László Zoltán, Dr. Szirmay-Kalos László, Objektum-Orientált Szoftverfejlesztés, 2003


