Monitoring Programmable Logic Controllers

Henrique Gonçalves, José Gaspar
Instituto Superior Técnico / Universidade de Lisboa, Portugal
henriquea.goncalves@gmail.com, jag@isr.tecnico.ulisboa.pt

Abstract—Discrete Event Systems (DES) are ubiquitous in industrial applications. Programmable Logic Controllers (PLC) are commonly used to implement DES. Designing a DES and implementing in a PLC is a lengthy process typically requiring multiple verifications and an end process validation.

Our approach to design a DES involves using Petri Nets (PN). Then we propose an automatic conversion of the PN to Structured Text (ST) code, a standard PLC programming language. With the objective of verifying that the PLC program fulfills design specifications we develop a PLC monitoring system.

Construction of a hardware PLC Signal Acquirer, able to monitor PLC inputs and outputs, is detailed in this work. Cost is kept low by using the affordable microcontroller board Arduino Mega. Development of additional hardware is also detailed, having the purpose of interconnecting the monitor with the PLC and the controlled plant.

With our hardware developed and DES implemented we show characteristics of the PLC behavior, as the variability of the scan cycle time, and validate the DES implementation of a keyboard reading case study.

Keywords: Signal Acquirer, Discrete Event Systems, Programmable Logic Controllers, Arduino, Petri Net, Structured Text.

I. INTRODUCTION

In industrial applications Discrete Event Systems (DES) are ubiquitous, from telecommunications to flexible manufacturing. Programmable Logic Controllers (PLC) are used to implement DES. With the objective of verifying that a PLC program fulfills design specifications we develop a PLC monitoring system. To achieve our objective we detail the use of Petri Nets (PN) to create a DES, and develop a program that from PN characteristics is able to create Structured Text (ST) code, a PLC programming language, to be implemented in a PLC. Construction of an hardware PLC Signal Acquirer, able to monitor PLC programs, will be detailed using the affordable microcontroller board Arduino Mega. Development of additional hardware will be explained, in order to make connections possible between the Arduino a PLC and a controlled plant. With our hardware developed and DES implemented we then are able to observe the PLC behavior, and try to validate the DES implementation.

A. Related Work

Petri net theory was continually developed since its beginnings with Dr. Petri’s 1962 Ph.D. Dissertation[13]. But it was not until 1981, that Peterson [12] wrote the book that presented all the development work scattered among many sources in a coherent and consistent manner. In 1967 Naylor and Finger [10], proposed a multistage process of validation. This validation method consists of (1) developing the model on theory, observations, and general knowledge; (2) validating the model’s assumptions where possible by empirically testing them; and (3) comparing (testing) the input-output relationships of the model to the real system.

In 1979 Schlesinger [14], defined Model validation to mean ”substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model”. In 1992 [2], Cutts and Rattigan addressed the fact that most PLC programmers develop programs using Ladder Logic, that makes a situation with a very large number of logical inputs and logical combinations, very hard to control. As a solution to this problem, Cutts and Rattigan [2], proposed to use software based on Petri nets. In 2004 conformance test, testing a PLC system seen as a black-box (its internal structure is unknown) with observable inputs/outputs, is advocated by certification bodies and standards [7]. In 2011 Sargent [8], made a study of the several validation techniques and tests commonly used in model validation, and purposed a model for several steps for validation of simulation models, in which it is necessary to make comparisons between the simulation model and system behavior data for at least a few sets of experimental conditions, and preferably for several sets. In 2013 Vargas, Mellado and Lesage [4] [3], developed a software tool for building automatically interpreted Petri net models from an observed system’s input/output sequence.

B. Objectives and Challenges

In this thesis we develop PLC monitoring with the objective of verifying that a PLC program fulfills the design specifications. In other words, we want a design validation procedure applicable to a discrete event system (DES) implemented in a PLC. We assume that the input and output events during a finite amount of time are enough to characterize the DES. Hence, our validation procedure will be based in observing input and output signals (events) of the DES.

Our approach for building and debugging a DES implemented with a PLC is the following: (i) Design one PN solving the same problem that the DES will solve. In this step we avoid some problems by design, e.g. not using unobservable transitions, preventing deadlocks, etc. (ii) Simulating the PN and saving input and output signals. (iii) Encode the PN into a PLC language. (iv) Compare simulation with real run.

In this work is proposed the construction of an hardware PLC Signal Acquirer, able to monitor analyze and validate PLC programs.
II. DISCRETE EVENT SYSTEMS

In this chapter we introduce Discrete Event Systems (DES). The structure of the introduction is based in the course Industrial Processes Automation by Professor Paulo Oliveira [1,11] and in books [12, 9, 1].

A DES can be characterized by a discrete state space whose state transitions happen at discrete points in time due to events [1]. In generic terms one PLC is composed of tree major parts, namely inputs, outputs and a control processor. When an input changes, the control processor responds by changing an output. This can be compared to the the state change that leads to an event occurrence in a DES.

Petri nets allow representing general DES, whose operation is a finite set of transitions, \( T \), and \( T \) is the incidence matrix that accounts the balance of tokens on \( p \) of marks on \( p \). The implementation of a transition \( t \) is enabled when the number \( \mu \) of tokens is at least as great as the weight arc connecting \( p \) to \( t \) for all paths \( p \) that are inputs to the transition \( t \).

The state evolution of a Petri net can be described as the following matrix function,

\[
\mu(k + 1) = \mu(k) + D \ast q(k).
\]

where \( \mu(k) \) represents the present marking, \( \mu(k + 1) \) is the marking we want to achieve, \( q(k) \) represents the triggering vector needed to go from the marking \( \mu(k) \) to \( \mu(k + 1) \) and \( D \) is the incidence matrix that accounts the balance of tokens (giving the transitions fired).

A transition \( t_j \in T \) is active if \( x(p_i) \geq w(p_i,t_j) \) for all \( p_i \in I(t_j) \). That is, \( t_j \) transition is enabled when the number of marks on \( p_i \) is at least as great as the weight arc connecting \( p_i \) to \( t_j \) for all paths \( p_i \) that are inputs to the transition \( t_j \).

The implementation of a transition \( t \) is accomplished by removing each place of assembly \( L, I(L,t) \) marks. After this, \( O(P,t) \) marks are set in each place of assembly \( P \).

C. Properties

Petri net properties provide us a way to characterize and study discrete event systems.

Property 1. Reachability, a configuration of the marks over places \( M \) (marking), given a Petri net \( N \), is reachable if \( M \in R(N) \), where \( R(N) \) is the group of all the possible markings of \( N \).

Property 2. Boundness, it is the property that verifies if the number of marks for all the places is limited. A place in a Petri net is \( k \)-bounded if it does not contain more than \( k \) marks in all reachable markings.

Property 3. Safeness, a Petri net is safe if for every possible marking, there isn’t more than one mark for each place. A Safe Petri net is \( I \)-bounded.

Property 4. Conservation, A Petri net is strictly conservative, when for every marking the total number of marks remains the same. A Petri net is strictly conservative if for every \( \mu \in R(C,\mu) \):

\[
\sum_{p_i \in P} \mu'(p_i) = \sum_{p_i \in P} \mu(p_i)
\]

Property 5. Liveness. This property is associated, with the possibility that a Petri net has of running continuously without reaching a deadlock. A Petri net is alive if for all the possible markings, there’s a trigger for another marking.

Property 6. Coverability. Given a Petri net with initial marking \( u_0 \), the state \( u' \in R(C,u) \) is covered if \( u'(i) \leq u(i) \), for all places \( p_i \in P \).

D. Supervision based on Linear Constraints

A supervisor is a discrete event system that is used to monitor and control another discrete event system, satisfying a specification. In a Petri net, a supervisor restricts transitions that may trigger at a given state of the Petri net, so that the specification is satisfied. We desire a supervisor to prevent the system from reaching markings which do not satisfy,

\[
L_{\mu p} \leq b
\]

Where \( L \) is an integer \( n_c \times n \) matrix (\( n_c \) is the number of constraints, \( n \) is the number of places of the given Petri net), \( b \) is an integer column vector, and \( \mu \) is the Petri net marking, \( \mu_c \) is a vector of \( n_c \) non-negative slack variables, defined as,

\[
\mu_c = b - L\mu
\]

After introduction of slack variables, \( L_{\mu p} \) becomes,

\[
L_{\mu p} + \mu_c \leq b
\]

With \( \mu_{c0} \) as the initial marking of the Petri net supervisor,

\[
\mu_{c0} = b - L\mu_{0}
\]

Let \( D_p \) be the Petri net incidence matrix, the supervisor with incidence matrix be \( D_c = -LD_p \),

\[
D = [D_p^T, D_c^T]^T,
\]
If,
\[ L_{\mu_0} - b \leq 0, \]
(7)
Then a Petri net supervisor enforces the constraint (2) when included in the closed-loop system. If (7) is not true, the constraints cannot be enforced.

III. PLC DATA ACQUISITION

This section details first the hardware already available for PLC interaction, and then the hardware used and developed for signal acquisition.

A. Hardware Setup

Figure 1 shows the system architecture in which we based ourselves to fulfill this thesis, composed of four elements: the controlled plant, the programmable logic controller (PLC), the signal acquirer and a personal computer (PC). The signal acquirer and PC form the monitoring hardware. The PLC provides all the information necessary for the plant to operate, the monitoring hardware is responsible for monitoring the system, while it is also able to control the plant (reference generator).

![Generic setup block diagram.](image1)

Fig. 1. Generic setup block diagram.

(a) Power supply.  
(b) PLC.  
(c) Terminal.

Fig. 2. 24 volt DC power supply (a). Modicon PLC (b). Terminal with twelve key numbered keyboard (c).

B. PLC Input and Output Connections

The used controlled plant, our terminal, is a twelve key keyboard, seen in Figure 2(a).

To facilitate connections between the difference hardware components, several printed circuit boards were designed (PCBs), that would allows us to forty pin IDC cables and cables that have a forty pin IDC connection on one end and two twenty pin IDC connection connections on the other, Figure 3.

![Assembled printed circuit boards, for PLC (a), and for terminal (b).](image2)

Fig. 3. Assembled printed circuit boards, for PLC (a), and for terminal (b).

![PLC to terminal connections using final PCBs manufactured at Eurocircuits.](image3)

Fig. 4. PLC to terminal connections using final PCBs manufactured at Eurocircuits.

C. PLC Input and Output Monitor

Monitoring the PLC will provide a way to keep a log of all signals that go from/to the PLC to/from the terminal, as well as communicate with the PLC (acting as a if it was the terminal), and communicate the acquired data with a PC.

![Arduino based pass through logic observer and reference generator.](image4)

Fig. 5. Arduino based pass through logic observer and reference generator.

In figure 5 a design for an Arduino based monitoring hardware is shown. The signal acquirer/monitor, composed by the Arduino and Arduino shield, is the center piece of the system, everything connects to it.
The design of our signal acquirer was done with an Arduino Mega as its main processing unit, because of its characteristics, ease of use, and low cost.

The designed shield provides three forty pin IDC cable connection, there are two possible connection configurations that can be used. The first one for use as a data logger, in the other configuration the Arduino can be configured so that its inputs/outputs mimics an hardwired connection and also, to act as a terminal, sending desired signals to the PLC.

The last problem that our Arduino shield solves is the problem of the Arduino and the PLC’s inputs/outputs working in different potential differences, using voltage dividers and hex buffer/driver integrated circuits.

IV. PLC Program Development and Monitoring

In this chapter we explain our approach to designing a discrete event system (DES) problem solution and implementation based in Petri net theory.

A. Petri net Simulation

With a 3x4 keyboard in mind, a Petri net, describing the events and state evolution of the key reading system, was developed to model the process of reading one keyboard key as a DES, this subject is focus of chapter [12]. Its analysis is done resorting to the Petri net simulator PM Editeur[6] and the Matlab Petri Net toolbox[13], that allows us to generate Petri net’s incidence matrix and initial marking matrix. Using Matlab functions, combined with the tools mentioned above, we are able to simulate the Petri net’s behavior.

The Matlab simulator works by receiving a structure for Petri net’s characteristics and running a loop, for which each iteration corresponds to ten milliseconds. The loop’s functions receive as inputs the Petri net’s characteristics, and has the following sequence: PN_s2act, for each loop iteration, this function returns an array with the active keyboard column for that iteration; PN_tfire, this function returns the Petri net’s transitions that can be activated at the current time that the function is called; filter_possible_firings, this function compares the result from PN_tfire with the marked places from the previous loop iteration or from initial marking, the function returns an array with the possible transition to be triggered, the program can now reassign the marked places array with the marked places result for this loop iteration; PN_s2yout, this function checks the marked places array and returns the key pressed for this loop iteration.

B. Petri net to PLC Conversion

The approach chosen is using PLC programming language Structured Text (ST), as the pure text content allows an easier transfer from the Petri net to the PLC integrated development environment. ST allows to describe the behavior of a control function according to the information it receives. Its understanding is simple for users familiar with high-level programming languages.

1) Petri net to Structured Text Compiler: The method chosen to transcribe Petri net to ST, creating a PLC program from a Petri Net and IO mapping, uses three auxiliary functions to that prepare a problem’s specific data, like timed transitions and physical inputs or outputs, to be used straightforward by the main script. After having the information provided by our auxiliary functions, we are able to create a PLC program from a Petri Net and IO mapping using the functions briefly described:

- plc_map_inputs(inp_bits_lst, trans_lst, options), input is a List of mapped transitions, output is ST code with list of transitions and their respective input. Allocates one memory position for each transition. Each transition is represented by memory position that is the sum of %MW100 plus its transition number (transition four is %MW104). Establishes causal

Fig. 6. Designed Arduino Mega shield PCB (a). Signal monitor composed by Arduino Mega and Arduino shield (b).

Fig. 7. Generic setup with signal monitor assembled.
effect relation between PLC inputs and memory positions that define transitions.

- **plc_timed_transitions(t_trans_place_lst)**
  inputs are a list of timed transitions, time out time, list of places; output is ST code with one timer for each timed transition. Creates one timer for each timed transition. The activation of the timer is controlled by a place being marked, after the timer reaches timeout time the transition is triggered. Each of this timed transitions are allocated to one memory position.

- **plc_encode_places(places_lst, mu0, tprio, options)**, input is loaded Petri net, output is ST code for memory initialization and logical conditions between memory positions. To set the initial condition of the program (initial marking in the Petri net) it is necessary to create a statement that will be executed only once by the PLC. The program represents each place by a memory position that is the sum of %MW200 plus its place number (place four is %MW204). It verifies if initialization command is received, and if so, all the variables associated to places are set to their initial marking value.

Defines cause and effect relations between memory positions that define places and memory positions that define transitions, by using if statements that check if the memory position for a transition has a value different than one, and that the memory positions for the places that need to be marked for the transition’s firing are marked, if these two conditions are met, the memory position values are decremented and the memory positions for the places that receive a mark from the referred transition are incremented.

- **plc_output_if_any_place(places_lst_main, outp_bit_main)**, input is list of mapped places, output is ST code with list of places and their respective output. The value of auxiliary variable associated with each place is assigned to its respective output variable, using if statements. Various places can be associated with the same PLC output, using or statements.

- **plc_encode_places(places_lst, outp_bits_lst)**, input is list of places, output is ST code to output debug / monitor info. Creates PLC code to output debug/monitor info. Receives a list of places. Using if, establishes relation between those places being marked and a variable (output). It also creates a flag to check if more then one of the places in the list is marked. Using bit logic bit logic, and the value from the variable output, a bit signal representing the marked place is sent using four outputs.

V. CASE STUDY: KEYBOARD READING

The physical system’s process consists of reading one keyboard key with a PLC implementing the DES. This section describes the synthesis of the DES for keyboard reading, taking into account modeling and analysis properties. Synthesis will be based on supervised control.

A. Proposed Petri Net Model

Modeling the process of reading one keyboard key as a DES, is done by developing a Petri Net that describes the events and the state evolution of the key reading system. The developed system can be divided into tree main blocks. Each block represents one of the tree keyboard columns. In each of the three blocks there is a wait state, waiting for a key in their column to be pressed. If a key is pressed the system evolves to a state waiting for the key to be released, and then the system evolves to the first state, of waiting for a key to be pressed (same column). The wait for a key to be pressed state is timed (fifty milliseconds), if a key is not pressed the system evolves to a wait state for keys from the second column. If again a key is not pressed, after fifty milliseconds the system evolves to a state of waiting for a key to be pressed on the third column, if another fifty milliseconds go by, the system evolves again to the first state. A timed transition (T1, T11 and T2) is used for each of the tree wait states. The activation of different columns is represented by the marking of one place connected that connect to one of these transitions (P1, P2 ad P3). If a key is pressed transitions are activated, depending on the key, that pass the mark to a new place representing the pressed key. A transition connected to each place representing a pressed key, sends the mark to the wait timed place when a key is released. From the event evolution the incident matrix can be obtained. The resulting Petri net is shown in Figure 8.

![Developed Petri net](image)

B. Properties

From equation it is known that for in order there needs to exist a triggering vector \( q(k) \) to go from the marking \( \mu(k) \) to \( \mu(k + 1) \), in order \( \mu(k + 1) \) be reachable. By analyzing...
the Reachability Tree, with initial state \( \mu_0 = (1, 0, \ldots, 0) \) (corresponding to the beginning column scanning in column 1), it is verified that the reachability set \( R(N) \) is finite and that all other system states are reachable. From the Reachability Tree we can observe that there are sixteen different possible markings. From a direct analysis, we observe that for each marking, it is always possible to define a triggering sequence that will enable any other transition. From this we can conclude all transitions are live, and from every state you can reach every state, no deadlock occurs. All the transitions are live of level four. A Petri net is k-bounded if all its places are k-bounded. A place is k-bounded if for all possible states, the number of marks is less than or equal to a integer k. The developed Petri net is conservative, the total number of marks remains constant, which ensures that all states have the same number of marks that the initial state. As the initial state has a mark in place one (scanning starts in column one), and zero marks in the other places, after each transition, only one of the places will have a mark and all other places will have zero marks. Therefore, all places are one bounded and as such, the Petri net is one bounded. It is observable that the Petri net is safe, since by definition, a Petri net is safe if all its places are one bounded.

C. Supervisor

In order to eliminate failure in the designed system, due to more than one key being pressed simultaneously, designing and implementation of a supervisor, that detects the multiple keys pressed error and resumes the normal operation of the system, is done by applying Supervisory Control theory.

Using only two transitions and two places, we can inform the system of triggering of multiple keys at the same time, taking the system to a error state. Place seventeen is used as f\( \text{lag} \) that when marked indicates multiple keys are being triggered. In order to trigger transition twenty eight, the triggering of multiple keys must be detected, leading to the marking of place seventeen. When a non-error state is again reached transition twenty nine is triggered and place sixteen is marked. On Figure 9 the Petri net way of working is shown. To integrate the f\( \text{lag} \) Petri net, with the Petri net described in section V-A and this way creating a controller to handle the multiple keys pressed error, linear constraints must be implemented. The linear constraints imposed are related to key validation places and the place that is used as a error f\( \text{lag} \) not being able to be marked simultaneously. This is represented by, if \( \mu_k \) is the number of marks in place k:

\[ \mu_{17} + \mu_k \leq 1, k \in \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\} \]  \( (8) \)

From the linear constraints and using controller synthesis described in section II-D, the relation \( (9) \) allows us to obtain the matrices for \( L \) and \( b \).

\[ L\mu_0 \leq b \]  \( (9) \)

The next step in controller synthesis based on linear constraints is verifying \( b - L\mu_0 \geq 0 \), which is true given that the result of \( b - L\mu_0 \) is \( b \). It is relevant to note that \( \mu_0 \) represents the initial markings of the combined Petri nets, it is an one column matrix with one row per Petri net place, seventeen in total, with all values zero except for the first and sixteenth values, which are one. The first value represents the first state of the keyboard Petri net, waiting for a key to be pressed for fifty milliseconds, and sixteenth value represents the f\( \text{lag} \) Petri net state of multiple keys not being pressed at the same time. The supervisor incidence matrix \( (D_c) \) and initial markings \( (\mu_{c0}) \) are determined by the relations \( (10) \) and \( (11) \) respectively. \( D_p \) represents the incidence matrix of the combined Petri nets.

\[ D_c = -LD_p \]  \( (10) \)

\[ \mu_{c0} = b - L\mu_0 \]  \( (11) \)

The finale incidence matrix of the supervised system \( (D) \) is constituted by the incidence matrix for the Petri net used for key detection from section V-A the incidence matrix for the Petri net used to detect pressing of multiple keys, the incidence matrix for the system that combines the last two referred Petri nets \( (D_p) \), and the supervisor controller incidence matrix synthesized based on linear constraints \( (D_c) \). The supervised system resulting Petri net is shown in Figure 10.

![Fig. 9. Flag Petri net method of operation.](image)

![Fig. 10. Petri net developed for keyboard control, with supervisor.](image)
D. Implementation

Using the ST compiler detailed in [V-B1] we are able to implement the Petri net for our case study in the PLC as ST code. With the results from the auxiliary functions, our compiler is then able to create ST code to run in loop in the PLC, that simulates the work-flow of the developed Petri Net, taking into account the inputs and outputs received by the PLC. The ST program generate is comprised of five cyclic operations: Petri net initialization To set the initial condition of the program (initial marking in our Petri net) it is necessary to create a statement that will be executed only when an specific input is received by the PLC. If this condition is met, the value HIGH must be assigned to all the auxiliary variables corresponding to the places initially marked, and the value LOW must be assigned to all the auxiliary variables corresponding to the places with no initial marking. This correlates to our case study keyboard being in the state of no keys pressed and with the column one being the first one to be checked. To map PLC inputs For each input (that represents a transition) there are two variables, one to represent the HIGH state of the input and one to represent the LOW state. The variables also takes into consideration if the output that allows state of the input and one to represent the transition) there are two variables, one to represent the LOW state. This part of the program checks for keys being pressed or released, and sets the appropriate value to a variable that represents those actions transition. Controlling of timed transitions is done with three timers which consecutively, for fifty milliseconds, set the value of one variable corresponding to a Petri net place to HIGH state. This are the same transitions in our Petri net to wait for a key to be pressed for fifty milliseconds for each column. The Petri net loop does PLC variables verification. For each scan cycle, checks which places should be marked or unmarked. For this, a logical combination (AND) of two conditions must be performed: (a) input variables associated with the transition are satisfied (HIGH); (b) existence of tokens in places (auxiliary variables are HIGH). If a transition to another PN state is possible, the value of place variables is decreased, and the value of the variable representing the next Place is increased. Setting PLC outputs is done by checking the values of the created variables and variables that represent PN Places directly dependent on the first referred variables. It then sets one of three PLC outputs to a HIGH or LOW level, making possible the interaction of the terminal with the PLC. Encoding of places checks the value of the PLC variables that represent PN Places are marked depending on the PLC input, and then outputs a four bit signal depending on which PN Place is marked or if more then one Place is marked (error case).

VI. EXPERIMENTS

This chapter describes the experiments performed to validate the methodologies presented and introduced in previous chapters.

A. PLC Characterization

PLC’s operate by continually scanning programs and reading inputs and outputs. One iteration of this process is called scan cycle. The scan cycle consists of three steps: determine status of inputs, interpret logic of programs, and update output status. Knowing the scan cycle allows us to know smallest time period possible for the PLC to detect two consecutive key triggers. In our experiments the PLC’s scan cycle was fixed to ten milliseconds, this value was chosen to have a more stable scan cycle, as experiments will show, and because in the implemented Petri net the minimum value of scan cycle needed for correct operation is fifty milliseconds. It is also an unrealistic to experiment terminal key pressing for time differences so low. The used experimental setup for this section is shown in Figure 11 (a). Figure 11 (b) shows the block diagram representation of the system used for the PLC’s scan cycle characterization. The structured text program implemented in the PLC in order to monitor the scan cycle is shown in Figure 11 (c). By running this program the PLC is setting its first output (%q0.3.16) to zero volts or twenty four volts, its output is a square wave. Figure 12 shows the results of monitoring the PLC’s square wave for eight seconds. In (a) and (b) we overlap the results of the monitored scan cycles rising edge’s shifted to time = 0s. In (a) the overlapping is
done in groups of five scan cycles, and in (b) all the monitored scan cycles are overlapped. In (c) each of the blue circles represents the time difference between monitoring a digital TRUE value and a digital FALSE value PLC output (half the period of the square wave the PLC sending as output) along the eight seconds of monitoring, each value represents a different time of scan cycle. (d) is a graphical representation of the distribution of the scan cycle time value, histogram. From the results shown in Figure 12 it is observable that the PLC does not have an exact time value between occurrences, this time varies roughly between nine point five milliseconds and ten point five milliseconds. In the context of our case study, keyboard reading, the interpretation of these results leads us to acknowledge that a key should not be pressed for less then ten point five milliseconds, or it might not be detected by the PLC. This is of course also valid for any input we wish the PLC to read. In this next experiment the PLC structured text

%q0.3.16 := %i0.3.0;

(a) PLC program.

(b) I/O signals.

(c) Input Detection.

Fig. 13. Observing the time for 100ms period square signal detection. PLC test code. (a) Connections diagram (b). Observed PLC time of signal detection along 8 seconds (c).

program in Figure 13 (a) was implemented in PLC. In this case, the monitoring system is not only monitoring the PLC’s inputs and outputs, but it is also generating a square wave signal with a period of one hundred milliseconds. The PLC should set its output according to the value it is receiving in its input (digital TRUE or FALSE), the received square wave value. In Figure 13(b) a block diagram for this experiment is shown. From the results shown in Figure 13(c) we observe that the monitoring system detected the PLC output change in the expected time instances, fifty milliseconds apart (half the period time). We can also observe that time of input detection is not exact. The results also show some increase in input detection, of almost ten milliseconds. This happens when the PLC fails to detect an input change, and only detects it in its next scan cycle, thus the increase of ten milliseconds. The forty milliseconds time change shown in the results, happens because an input was detected late by the PLC, so the time difference of the next correctly detected input should be detected ten milliseconds earlier. The fact that there are two input detections at sixty milliseconds since the last inputs, but only one forty milliseconds input detection since the last input, can be attributed to two straight input detection delays by the PLC. In the context of our case study, keyboard reading, the interpretation of previous results leads us to acknowledge that the PLC set to a ten millisecond time of

scan cycle has an accuracy with an error of plus and minus ten milliseconds. Figure 14 shows the minimum (a) and maximum (b), time difference between rising edges, of monitoring time of scan cycle, as it was done in the first experiment (results shown in Figure 12), but with the difference that this results were obtained without a PLC fixed scan cycle time of ten milliseconds. As shown in Figure 14 the PLC can have a scan cycle of less then one millisecond, but the variation of time of scan cycle increases greatly, leading to detected change in scan cycle time of around three milliseconds. Since there is no need in our case study, to have a scan cycle lower than ten milliseconds, and since not assigning a fixed value to the PLC’s time of scan cycle leads to a more unstable scan cycle time, the ten milliseconds time of scan cycle was sustained in experiments done in the next sections.

B. Key detection

This experiment carried out a sequential triggering and release of all the keys, without conflicts. The used experimental setup for this section is the same shown in Figure 7. The key

Fig. 14. Varying scan cycle time. Minimum scan cycle time observed experimentally (a). Maximum scan cycle time observed experimentally (b).

Fig. 15. PLC output captured by the Arduino based monitor, input data key sequence (a). Input data, sequence of keys (b). Expected output, MatLab Simulation (c). PLC output observed with the developed monitor (d).

sequence used, during a time period of eighteen seconds (input data), is shown in Figure 15 (a). In Table (b), besides the key sequence used, also includes the period of time in which the keys are triggered and the expected results the PLC should
output. Figure 15(c) shows the Matlab simulation results using the input data for this experiment. Figure 15(d) shows the PLC monitoring results for the same input data. The experimental results are consistent with the Matlab simulation results, as well as with the expected results. This experiment shows that the correct validation of key triggering and release of all the keys, without conflicts, is correctly done by the Petri net implemented in the PLC.

C. Initial Marking Effect

Initial marking of the implemented Petri net is expected to have an effect to our system in the way it validates one of multiple keys triggered at the same time from different terminal keyboard columns. The key sequence used for all the pressed keys, the key validated is the one that belongs to the initial marked column (third column), key three. Comparing these results with the ones on the last experiment we also observe that the rest of the validated keys does not match, this is also expected, since the fact that the initialization is done in different columns, with the passage of time, the column activation for each column will be done in different time instances.

D. Supervisory Control

In this section we present the results concerning the implementation, to the design Petri net, of a supervisor based in linear constraints, done in section V-C, in order to prevent failure in the designed system, due to more than one key being pressed simultaneously. Figure 15(a) shows the key combinations being pressed and released by the developed monitoring system, during a time period of eighteen seconds (input data). The table (b) in the same Figure shows the experiments in this section, during a time period of eighteen seconds (input data), is shown in Figure 16(a). In (b) is shown the first used initial marking matrix. Figure 16(c) shows the Matlab simulation results using the input data for this experiment. Figure 16(d) shows the PLC monitoring results for the same input data. The experimental results are consistent with the Matlab simulation results. As expected, the results show that in the first time instance of pressed keys, the key validated is the one that belongs to the initial marked column (third column), key three. Comparing these results with the ones on the last experiment we also observe that the rest of the validated keys does not match, this is also expected, since the fact that the initialization is done in different columns, with the passage of time, the column activation for each column will be done in different time instances.
expected results for each time instant. Figure 18 (c) shows the Matlab simulation results from the input data mentioned above. Figure 18 (d) shows the PLC monitoring results for the same input data. Both results are consistent with the expected results, and with each other. The supervisor detects triggering of multiple keys and does not validate any of them. In the Matlab simulation that single keys being pressed during the time of experiment are validated, but when the system receives multiple keys from the same column, eight seconds time mark, no key is validated. In the monitoring results, no key validated in the eight seconds time instance, and we can also observe that the PLC outputs the error bit sequence as mentioned in section [12] This error sequence is received multiple times during the referred time instance, this is due to the fact that the PLC keeps receiving, from the monitoring system, the same key triggering information for every scan cycle it does during the said time period.

VII. CONCLUSION AND FUTURE WORK

The work described in this thesis aimed at verifying that a PLC program fulfills design specifications, from the data acquired by a developed signal acquirer for PLC signal inputs and outputs. The main objective of developing this device is to be able obtain information about a DES PLC implementation that could lead to the validation of the implementation.

We first presented a form of DES representation, Petri nets, a intuitive method that captures structural information about the system and has mathematical analysis techniques well developed. This proved to be a way to represent our case study. We were able to create the intended system as a Petri net, and using matrix techniques, we were able to solve the problem of multiple keys being triggered at the same time.

We have developed a signal acquirer, based in the low-cost microcontroller-based board Arduino Mega, that can observe and save in memory signals going from and to a PLC connected to a terminal, and at the same time is able to connect to a PC for further data processing. The signal acquirer is also able to act as the terminal, mimicking its behavior by sending signals to the PLC. In order to achieve this we developed an Arduino shield that was able to ensure the connections between the Arduino, the PLC and the terminal.

To implement a DES in a PLC, in a the form of Petri net representation, we chose to do it building a program that produces structured text code, a form of PLC programming, based on receiving a Petri net characteristics. Implementing a Petri net in a device usually requires discrete events, but may also require timed events. To develop the conversion program we add to take into consideration not only the Petri net relationships between places and transitions, but we had to take into consideration the assurance of having a way to initialize the implemented Petri net in the desired configuration. We had to define connections between signal inputs and outputs with the various PLC memory positions attributed to the Petri net places and transitions. In the end, besides one function that creates code from a Petri net structure, four other functions had to be created. This points are also valid for the Petri net simulator used to obtain results about the Petri net development along a time period.

The case studied in chapter 14 showed that, although a Petri net being good way to represent a DES, it does not take into consideration some. One of those aspects is that a PLC is based in scan cycles, as shown in the experiments section, which constrain the activation speed of transitions. We also observed, in the experiments, that a PLC is also subject to delay of signal change detection, which is another aspect that a Petri net does not take into consideration.

The Petri net modeling proved to be effective. We were able to obtain a working system for the proposed case study. The supervisor implementation was also successful, in the sense that applied the desired constraints. However, even thought we were able to predict to a large extent the experimental results, the experiment consisting of changing the initial marking of a Petri net, proved that a DES implementation into a real world situation has more details that have to be taken into consideration than those presented by a Petri net.

Future work should focus on further development of the Arduino shield. Ways to not bind each of the Arduino's inputs and outputs to one signal can be implemented using IC’s that can send send signals to multiple lines at the same time, by only receiving signal from one source. This would make it possible to use a even lower-cost version of the Arduino board.

Another aspect that should be developed, is the DES representation. The use of a more advanced Petri net representation, I/O Petri net, can lead to a more detailed DES representation, with fewer aspects that affect its implementation.

REFERENCES