DOI: 10.48011/asba.v2i1.1267

A CLOSED DESIGN CYCLE TO EMBEDDED LOW-COST AUTOMATION BASED IN MICRO-CONTROLLERS

Victor G. Cardoso*, Jessica T. Barros*, Renata T. Tavares*, Pedro M. G. del Foyo*

*Departamento de Engenharia Mecânica, Universidade Federal de Pernambuco
Av. Acadêmico Helio Ramos s/n, CEP 50740-550, Recife/PE, Brasil

Emails: victor.cardoso@ufpe.br, jessica.tbarros@ufpe.br, renata.ttavares@ufpe.br, pedro.foyo@ufpe.br

Abstract— In this paper a Discrete Event Systems approach is used to design embedded systems, implemented using a commercial low-cost platform based on a single-board micro-controller. The approach uses Time Petri Nets (TPN) to build the system model and a model-checking tool to perform the formal verification and to support the embedded system code generation. The method was applied to design an automated sorting machine using the *Arduino* platform as programming device.

Keywords— Embedded systems, Discrete Event Systems, Model checking

Resumo— Neste artigo, uma abordagem de sistemas de eventos discretos é usada para projetar sistemas embarcados, implementados utilizando uma plataforma comercial de baixo custo baseada em micro-controlador de placa única. A abordagem usa Redes de Petri Temporizadas para construir o modelo do sistema e uma ferramenta para verificação de modelos para executar a verificação formal do sistema e dar suporte à geração de código do sistema embarcado. O método foi aplicado ao projeto de uma máquina de classificação automatizada usando a plataforma *Arduino* como dispositivo de programação.

Palavras-chave— Sistemas embarcados, Sistemas de Eventos Discretos, Verificação de modelos

1 Introduction

Embedded systems are present in almost all aspects on today's society such as industrial machines, agricultural and process industry devices, medical equipment, household appliances, as well as mobile devices. Embedded systems demand is expected to continue rapidly growing, driven in large part by the internet of things (IoT).

Embedded systems consist of hardware, software and an environment (Henzinger and Sifakis, 2006). As stated in (Liu et al., 2018), traditional simulation-based verification could not cover all possible test cases due to a large number of potential interleaving between the hardware and software components. Then, formal verification techniques are needed in order to verify that design models work correctly for all allowed inputs by using rigorous mathematical methods.

Recent trends have focused on combining both language-based and synthesis-based approaches (hardware/software co-design) and on gaining, during the early design process, maximal independence from a specific implementation platform (Henzinger and Sifakis, 2006). Methodologies like SystemC (Panda, 2001) and UML (Rumbaugh et al., 2004) go that way but the lack of analytical tools for computational models to deal with physical constraints; and the difficulty to automatically transform non-computational models into efficient computational ones limits their application. Other techniques basically propose unified properties specification languages and a co-verification framework used to reason about the relationship between specification and corresponding implementation. A survey of those techniques can be found in (Liu et al., 2018). Most of the success of those techniques are in the verification of low-level software on firmware and drivers which can be used in compositional design approaches like in (Sifakis, 2005).

In (Hajjar, 2013) a technique based on Commercial Off The Self (COTS) components was proposed. In such method, formal verification is used to discover design errors and a Discrete Control Synthesis (DCS) (Ramadge and Wonham, 1989) method is used to automatically enforce some desired behaviors on a given system. The control is implemented using a sample-driven approach instead the event-driven approach. The DCS approach was modified to deal with a sample-driven approach which is more appropriated for electronic design techniques (Xie and Liu, 2007).

According to (Henzinger and Sifakis, 2006), the solution for the embedded system design problem cannot be achieved by simple juxtaposition of analytical and computational techniques, but requires their tight integration within a new mathematical foundation that spans both perspectives. Still, there is a class of embedded systems that can be implemented in commercial low cost platforms based on micro-controllers (Wilmshurst, 2007). The choice for an open source, commercial low-cost platform sometimes is justified by the need to speed up the development of the application. There is also a tested hardware, which provides enough computational power to execute the desired functionality. The *Arduino* platform has been successfully used in several minor projects like (Syazlina Mohd Soleh et al., 2018; Mukherjee et al., 2019; Lapshina et al., 2019).

The design, validation and maintenance processes make software, paradoxically, the most costly and least reliable part of systems in automotive, aerospace, medical, and other critical applications (Henzinger and Sifakis, 2006). Then, it is desirable that even for minor projects a formal verification can be done before implementation. In this paper, a methodology based on the Discrete Event System approach is used to design and implement an M&M's sorting machine. The paper main contribution is the scheduler synthesis from the automaton built after the system verification. Such scheduler will be the core of the embedded system running over a commercial low-cost platform.

The paper is organized as follows. Section 2 presents the M&M's sorting machine mechanical project and describes the system specification. In Section 3 the design methodology is presented. Section 4 shows the prototype implementation in two versions: preemptive and non-preemptive with its respective verifications procedures. Section 5 brings the implementations results. Finally, Section 6 presents the paper conclusion.

2 The M&M's sorting machine

The M&M's sorting machine was designed to separate M&Ms according with their color, putting them into specific cups for each color. The M&Ms come in packs with six different colors mixed. The mechanical project was adapted from (Wiki, 2017) and it is shown in Figure 1.

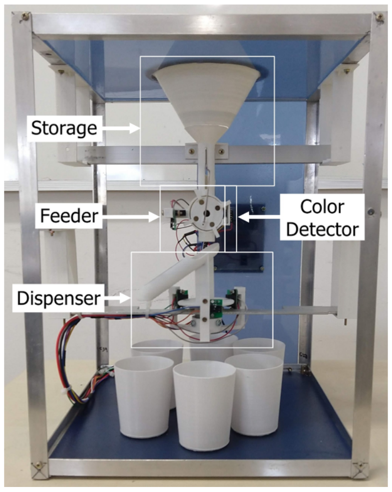


Figure 1: M&M's sorting machine

The machine prototype was built using 3D printed parts, aluminum angle connectors and electromechanical components available in the laboratory. As shown in Figure 1 the machine consists of four modules, three of which have electrical components necessary for their operation. The Feeder is driven by a continuous rotation servomotor (model SM-S4306R). Its function is to conduce the M&Ms from the storage to the color detection module and, at the same time, to deliver the M&M in the color detection position to the dispenser. An optical sensor (model HC 020k) detects when the Feeder reaches the desired position.

The Color Detection module has a color sensor (model TCS34725). When required, the sensor illuminates an M&M piece and collects the RGB intensities that will be used to detect the correct color by using appropriated algorithm. The Dispenser is the mechanism projected to deliver the M&M piece to the desired cup, which is attached to a step motor (model 28YJ-48). An encoder disk and a set of three optical sensors (model HC 020k) were used to identify the dispenser's six different positions.

The machine was designed to be fully automated and to support up to six different colors. It must process up to 20 M&M pieces by minute. The system must ensure that once the color was detected, the M&M will be dispensed to the appropriated cup. There must be also guaranteed that the dispenser will only move when the current position is different from the corresponding position to the color been detected.

3 Methodology

The proposed approach starts by choosing the basic features that each module must perform. It must be ensured that such functionalities are the simplest and its compositions will satisfy all required system functionalities. Once those basic tasks were defined, the system automation is reduced to synthesize a scheduler to guarantee the desired behavior. The idea is to use an existing commercial low cost platform based on single-board micro-controller that provides the necessary resources to interface with the proposed modules.

The advantage of working with a commercial low cost platform, besides the cost, is that there are tested hardware devices with access to IDE platform to program the device. The software/hardware co-verification is not needed since the basic tasks must be directly programmed and tested over the same devices in it will be operating. The challenge will be to show that the integration of those simple, reliable and tested tasks ensure the system specification.

Then, the system required behavior will be modeled using TPNs. TPN (Merlin and Faber, 1976) is a formal model suitable to deal with real-time systems because of their ability to represent varying time durations of activities, by using timing intervals attached to transitions (del Foyo and Silva, 2011). There are also several tools available that can be used to verify the system specification. TINA (Berthomieu and Vernadat, 2006) is one of them, and it has features that can support the system implementation.

The system model must be built using the state-transition paradigm where transitions represent system function executions and states represent system or environment variables. Once the system model was completed, the specification is translated in temporal formulae and a model checking procedure will answer if the desired behavior was achieved or not.

Besides safety and liveness properties verification, performance issues are usually part of the system specifications. Depending on the performance criteria established, TPNs properties can be used to assess those metrics. There are also other kinds of properties linked to software development like deadlock avoidance or termination proof. Such properties also need to be checked.

To avoid introducing errors in the translation from the model to the programming language, the scheduler is synthesized directly from the automaton built from the system model once the verification process is concluded.

3.1 Implementing the execution flow control function (scheduler)

In this proposal, a TPN verified model is used to automatically generate the C code to be used in the platform based on single-board microcontroller. In order to do this, the Labeling TPN must be used, in which the labels in transitions are related to the functions being executed and labels in places are associated to variables of two types: boolean or enumerated.

The algorithm proposed to synthesize the execution flow control function needs two basic entries: a file with the automaton that describes the desired scheduler behavior and a file with the Place and Transitions labels.

A Time Petri Net is a tuple $(P, T, B, F, M_o, SIM, \mathcal{L})$ where:

- P is a finite non-empty set of places $\{p_i\}$;
- T is a finite non-empty set of transitions $\{t_i\}$;
- $B : T \times P \rightarrow \mathbb{N}$ and $F : T \times P \rightarrow \mathbb{N}$ are the backward and forward incidence functions;
- $M_o : P \rightarrow \mathbb{N}$ is the initial marking
- SIM: $T \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$ is the static interval for each transition
- $\mathcal{L} : T \rightarrow L$ is a labeling function

When a software system is modeled with a TPN, several transitions could represent the same function. Such relaxation allows to build nets in an easy way. A labeling function is needed to define which transitions represent the same software function.

The labeling function $\mathcal{L}$ can be organized as a partition of T. Such partition must contain not only the labels (function names) but also all functions possible exit results. Lets define $O(L_k)$ as an operator that returns the set of transitions which represents all results of function L_k. If function L_k, represented by transition t_k returns only one result, $O(L_k) = \emptyset$. Formally:

$$O(L_k) : \forall t \in p^\bullet | t_k \in \mathcal{L}(L_k) \wedge t_k \in {}^\bullet p \wedge |p^\bullet| > 1$$

where ${}^\bullet p : \forall t \in T | B(t, p) \geq 1$ and $p^\bullet : \forall t \in T | F(t, p) \geq 1$.

Let $L_1, L_2, \ldots L_n$ be the functions names in TPN. The partition must satisfy the following condition to be considered valid:

$$L_1 \cap O(L_1) \cap \ldots \cap L_n \cap O(L_n) \quad = \quad \emptyset \qquad (1)$$

An automaton that preserves the linear temporal properties can be built using the state class approach and partial order techniques (Berthomieu et al., 2004).

That automaton obtained from the TPN model is a tuple $G = (S, \mu, s_0, T, \delta)$, where:

- S is a finite set of nodes, representing all reachable marking in the TPN;
- $\mu : S \rightarrow 2^P$ is a labeling function assigning to each node a set of marking places in TPN;
- $s_0 \in S$ is the start node;
- T is a set of transitions in the TPN;
- $\delta : S \times T \rightarrow S$ is transition function;

The scheduler is then synthetized from the automaton and TPN labeling information. Such operation is based on Algorithm 1:

```
1  **Algorithm 1:** Scheduler generation
   **Input:** File **.adr**: $S, \mu, s_0, T, \delta$
              File **.yaml**: $\mathcal{L}$
   **Output:** OutFile **.ino**
2  write '*int Scheduler(int s){*'
3  write '*switch(s){*'
4  **for** $s$ **do**
5      write '*case (s):*'
6      If $|s^\bullet|$=1 write $\mathcal{L}(s^\bullet)$;
7      write '$s = \delta(s, s^\bullet)$';
8      write 'break;'
9      **else**
10         write '*switch(VarName($\mu(s)$)){*'
11         **foreach** $t \in s^\bullet$ **do**
12             write 'case($\mathcal{L}(t)$)';
13             write '$s = \delta(s, t)$';
14             write 'break;'
15         **end**
16         write '} break;'
17     **end**
18     write '} break;'
19 **end**
20 write '}'
21 write 'return(s)'
```

The notation $s^\bullet$ was used to define the set of transitions that may occur to reach the s next states.

$$\forall t \in T | t \in s^\bullet \text{ iff } \delta(s,t) = s' \wedge s, s' \in S$$

$VarName(\mu(s))$ gets from the partition, the variable name associated to marking that holds in s. $\mathcal{L}(t)$ gets the function name associated to transition t. Those operations are conducted over the partition info file.

The proposed methodology will be applied to the M&M's sorting machine design.

4 Prototype Development

4.1 System Architecture

In order to implement the system automation, the architecture shown in Figure 2 was proposed. All necessary modules, described in previous section, must be controlled using a micro-controller based platform.

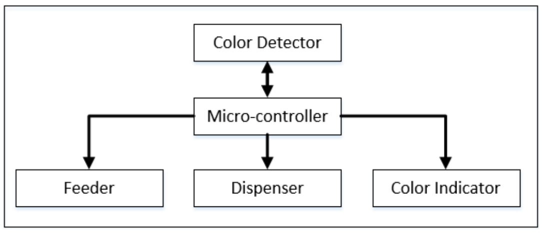


Figure 2: M&M's sorting machine architecture

An Arduino Nano board was selected as the micro-controller based platform in this project due to several reasons: the amount of available I/O pins, compatibility to I^2C communication protocol, facility of integration with other peripherals and its low cost. Such single-board micro-controller is based on the ATmega328P microprocessor and the system performance will be validated or not in the methodology next steps.

4.2 Software Functions Implementation

First of all, a proper basic set of functions must be found and implemented in the platform programming device. The fewer functions to be implemented and the simpler they are, the easier will be to maintain the code. There is also the claim that all "system intelligence" will be provided by the combination of that minimum set of simple functions, so there is no need to implement complex functions which are more difficult to verify.

The formal verification will answer if the desired functionality was achieved by those basic functions. Another set of functions must be implemented if the desired functionality is not achieved.

In order to achieve the desired system functionality the next functions were implemented:

- **MoveForward()** - This function sends the command to shift the feeder 90° clockwise. The wheel need to be stopped by software when the (inPosition) boolean variable is set to true by a hardware interrupt activated by the optical sensor.
- **GetColor()** - Performs the detection of the M&M's color by getting the RGB intensities and applying some calculation based on a calibration procedure. It returns an integer that represents one of six colors or that spot is empty.
- **MoveCW()** - Moves the dispenser 60° clockwise, only considering the number of steps given by the step motor.
- **MoveCCW()** - Moves the dispenser 60° counterclockwise, only considering the number of steps given by the step motor.

A **ShowColor()** function was develop to signalize the detected color in a RBG LED located behind the machine. Such function can be use inside the **GetColor()** function if desired but it does not interfere in system functionality. There is also the **Initialize()** function which instantiates variables, sensors and actuators according to the TPN initial marking.

4.2.1 Time measurements for determining function execution times

As stated before, TPN uses time interval in transitions to define the earliest and latest firing time. Such approach is appropriate to software functions since its executions times tends to be affected by many factors. Experiments were conducted in order to determine the minimum and maximum execution times (in milliseconds) for all proposed functions. Each function was executed 30 times and their duration were measured. The results are presented in Table 1.

Function Name	Min.	Mean	Max
MoveForward()*	730	754.93	785
GetColor()	1701	1702.23	1703
MoveCW()	478	479.10	480
MoveCCW()	479	479.70	481

Table 1: Time measurements results (in ms)

The execution time of the **MoveForward()** function was measured considering the time when the hardware interruption signaled that the Feeder reached 90° and the wheel was stopped.

As an embedded system design, environment behavior must be included in the system model. Then all system events are subject to time measurement experiments. In the case of the M&M's sorting machine there were also carried out some experiments to determine the time elapsed from the moment in which the M&M piece is released and the moment in which exits the dispenser to fall in the corresponding cup. After 40 experiments, a minimum time of 180ms and a maximum time of 630ms was determined.

4.3 The non preemptive solution

Once the set of basic functions were defined and implemented, a TPN model is built using *Net Draw* (nd) tool in TINA package.

Figure 3 shows the TPN model of the non preemptive solutions, that is, when function **MoveForward()** only exits when the feeder reaches the desired position.

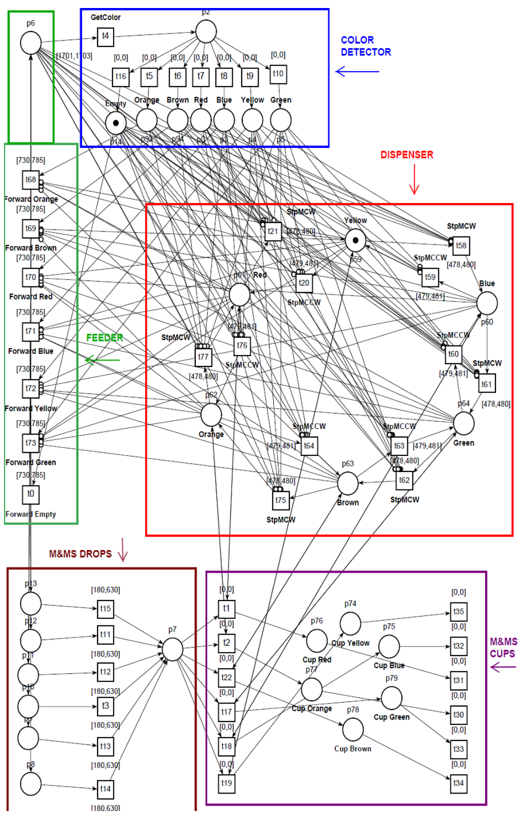


Figure 3: TPN model of the non preemptive solution

For verification purposes, the state space which represents the system behavior is computed using the *reachability analysis* tool with (-W) option. The TPN was identified as bounded and the computed state space had 78 states and 114 transitions.

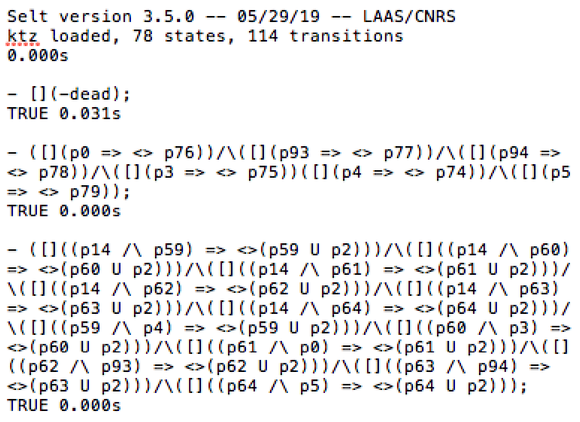
The bounded property in a TPN ensures the program termination. The absence of deadlocks was determined by model checking the property $\Box(\neg dead)$.

The temporal formula $(p0 \rightarrow \Diamond p76)$ states that once $p0$ is marked, signalizing that a yellow M&M was detected in the color analyzing position, all paths will lead to a state in which $p76$ holds, meaning that the M&M was dispensed to the yellow cup.

A temporal formulae to check the specification "The system must ensure that once the color was detected, the M&M will be dispensed to the appropriate cup", was built by the conjunctive form including the six available colors and cups.

The specification "The dispenser will only move when the current position is different from the corresponding position to the color been detected" was checked by a combination of twelve temporal formulas. Six of the sub-formulas were related to the case in which the spot is empty (signalized by $p14$) and the dispenser is in one of the six valid positions ($p59$ to $p64$ respectively), the marking with the same position will hold until the function **GetColor** be run again, signalized by marking $p2$. For instance: $(p15 \wedge p59) \rightarrow \Diamond(p59 \bigcup p2)$. Other six sub-formulas check that when the dispenser is in the position corresponding to the detected color it will not move until a new color had been detected. It is checked by a temporal formula like $(p59 \wedge p4) \rightarrow \Diamond(p59 \bigcup p2)$.

Figure 4 shows the verification results of temporal formulae designed to verify all three specifications.


```
Selt version 3.5.0 -- 05/29/19 -- LAAS/CNRS
ktz loaded, 78 states, 114 transitions
0.000s

- [](-dead);
TRUE 0.031s

- ([](p0 => <> p76))/\([](p93 => <> p77))/\([](p94 =>
<> p78))/\([](p3 => <> p75))([](p4 => <> p74))/\([](p5
=> <> p79));
TRUE 0.000s

- ([]((p14 /\ p59) => <>(p59 U p2)))/\([]((p14 /\ p60)
=> <>(p60 U p2)))/\([]((p14 /\ p61) => <>(p61 U p2)))/
\([]((p14 /\ p62) => <>(p62 U p2)))/\([]((p14 /\ p63)
=> <>(p63 U p2)))/\([]((p14 /\ p64) => <>(p64 U p2)))/
\([]((p59 /\ p4) => <>(p59 U p2)))/\([]((p60 /\ p3) =>
<>(p60 U p2)))/\([]((p61 /\ p0) => <>(p61 U p2)))/\([]
((p62 /\ p93) => <>(p62 U p2)))/\([]((p63 /\ p94) =>
<>(p63 U p2)))/\([]((p64 /\ p5) => <>(p64 U p2)));
TRUE 0.000s
```
Figure 4: Verification results for TPN model of the non preemptive solution

In order to evaluate the performance criteria, an M&M test package composed by 90 pieces, 15 of each color, was used in simulations. The TPN model in Figure 3 was modified to accept fire transitions that represent color detection only 15 times. Using the *stepper simulator* tool from TINA package with the random execution option the time elapsed to process the entire 90 pieces package can be obtained. According to the performance criteria established, such a package must be processed in at most 04 : 30s.

The simulation was repeated 40 times and the average time elapsed to process the test package was 286443ms, which represent approximately 04 : 46s above the desired rate of 20 M&M pieces by minute.

4.4 The preemptive solution

Considering that the **MoveForward()** function can be executed in a preemptive way, that means, when the function is called, the servomotor starts its movement and the function exits. Such operation takes less than 1ms. Then another action, for example a **MoveCW()** or **MoveCCW()**, can be executed before waiting for the interruption to stop the servomotor. That can be done since the worst execution time for CW or CCW operations are less than the best execution time waiting for the interruption.

Now the TPN model was updated and the control logics checks whether the dispenser is one movement away from reach the desired position. In those cases the **MoveForward()** function is executed before the last **MoveCW()** or **MoveCCW()** operation. Figure 5 shows the TPN that model the described behavior.

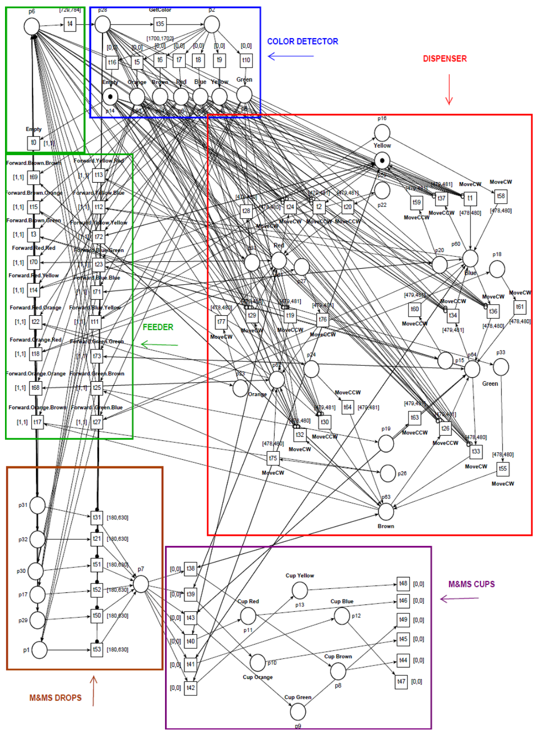


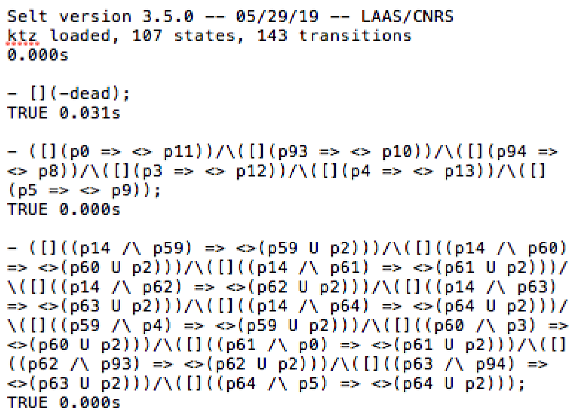
Figure 5: TPN model of the preemptive solution

Despite the fact that the same properties must be checked, the temporal formulae can be slightly different due to the place or transition increased in that version. Figure 6 shows the verification results of all temporal formulae.

The performance was evaluated using the same procedure and the average time elapsed was 250910ms, which represent approximately 04 : 11s bellow the desired rate of 20 M&M pieces by minute.

Once all system specification were achieved the execution flow control function must be synthesized. To yield the automaton that describes the scheduler, all net elements (places, transitions and arcs attached to them) representing the environment must be removed. So, elements modeling the M&M drop and M&M cups in Figure 5 are all removed. It is obvious that the environment behavior will be manifested in real execution.

By default, the Arduino sketch relies on

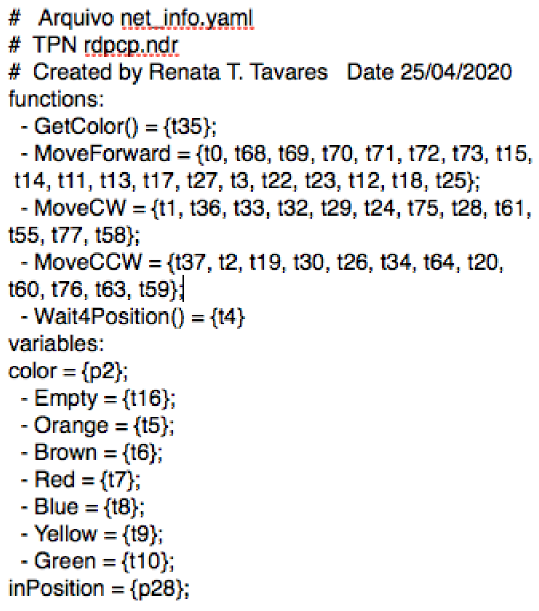


```
Selt version 3.5.0 -- 05/29/19 -- LAAS/CNRS
ktz loaded, 107 states, 143 transitions
0.000s

- [](-dead);
TRUE 0.031s

- ([](p0 => <> p11))/\([](p93 => <> p10))/\([](p94 =>
<> p8))/\([](p3 => <> p12))/\([](p4 => <> p13))/\([]
(p5 => <> p9));
TRUE 0.000s

- ([]((p14 /\ p59) => <>(p59 U p2)))/\([]((p14 /\ p60)
=> <>(p60 U p2)))/\([]((p14 /\ p61) => <>(p61 U p2)))/
\([]((p14 /\ p62) => <>(p62 U p2)))/\([]((p14 /\ p63)
=> <>(p63 U p2)))/\([]((p14 /\ p64) => <>(p64 U p2)))/
\([]((p59 /\ p4) => <>(p59 U p2)))/\([]((p60 /\ p3) =>
<>(p60 U p2)))/\([]((p61 /\ p0) => <>(p61 U p2)))/\([]
((p62 /\ p93) => <>(p62 U p2)))/\([]((p63 /\ p94) =>
<>(p63 U p2)))/\([]((p64 /\ p5) => <>(p64 U p2)));
TRUE 0.000s
```

Figure 6: Verification results for TPN model of the preemptive solution

two main functions: **setup()** and **loop()**. The **setup()** function only runs once, at the Arduino startup and generally it is used to initialize system variables and peripherals. The **loop()** function runs the code contained in its scope as an infinite loop. The system's control logic must be inserted in this context in order to operate continuously. The programmer is responsible by developing the code for both functions.

A program was developed in Python programming language to yield the scheduler following the algorithm explained in section 3.1. The automaton with 78 states and 114 edges was obtained from the *.ktz* file yield using TINA. The file *net_info.yaml* with function names and variables information is shown in Figure 7.



```
# Arquivo net_info.yaml
# TPN rdpcp.ndr
# Created by Renata T. Tavares   Date 25/04/2020
functions:
  - GetColor() = {t35};
  - MoveForward = {t0, t68, t69, t70, t71, t72, t73, t15,
 t14, t11, t13, t17, t27, t3, t22, t23, t12, t18, t25};
  - MoveCW = {t1, t36, t33, t32, t29, t24, t75, t28, t61,
 t55, t77, t58};
  - MoveCCW = {t37, t2, t19, t30, t26, t34, t64, t20,
 t60, t76, t63, t59}
  - Wait4Position() = {t4}
variables:
color = {p2};
  - Empty = {t16};
  - Orange = {t5};
  - Brown = {t6};
  - Red = {t7};
  - Blue = {t8};
  - Yellow = {t9};
  - Green = {t10};
inPosition = {p28};
```

Figure 7: net_info.yaml file corresponding to the TPN in Figure 5

The file describes the function labelling and variable declarations used to yield the scheduler function in C programming language based on *switch case sentence*. The function names: **GetColor()**, **MoveForward()**, **MoveCW()** and

MoveCCW() are the $L_1, \ldots, L_4$ labels described in section 3.1 with its respective transitions and variables *Color* - that can take one of next values: Empty, Orange, Brown Red, Blue, Yellow or Green, each of them represented by a transition - and *InPosition* - which is a boolean variable.

Operator $O(L_1)$ returns the transitions $t16, t5, t6, t7, t8, t9$ and $t10$ which represent the **GetColor()** possible results and $O(L_2), O(L_3)$ and $O(L_4)$ returns the empty set. So the partition condition 1 is satisfied.

The function **setup()** calls the **Initialize()** function and do $s = s_0$. The function **loop()** is based on the scheduler obtained by Algorithm 1:

```
void loop(){
  s = Scheduler(s);
}
```

The **Initialize()** function follows the TPN initial marking which was the same in both versions. The *color* variable was initialized in 0, corresponding to the empty spot, the *inPosition* variable was set to false and the dispenser was moved to the Yellow cup position.

5 Results

Following the proposed method, the **Scheduler()** function was obtained both for the non-preemptive -called here as version 1- and the preemptive -called version 2. It was also developed a version 3, following the approach in (Wiki, 2017). That last version was designed to follow logic operations using condition clauses and repetition loops along the process. At every iteration, the controller considers the dispenser position and the color detected to decide the next action. When a movement of the dispenser is necessary, the controller chose the closest path to take and preempts the feed wheel shift before the last dispenser movement.

The dispenser rotation functions utilized in version 3 differ considerably from that implemented for version 1 and 2. Instead of turning 60° continuously, these functions realize micro steps and check the optical sensors to detect the arrival at the next position. Furthermore, other minor functions were created to give support to logic operations. The dispenser motor speed remains the same in all versions in order to do a fair performance comparison.

Using a M&M package as defined in the tests, ten experiments were carried out for each version and the time required for processing was computed. It is worth mentioning that each experiment represents a different color sequence due to the randomness of this process. Thus, the results presented in Table 2 for a given experiment do not imply that the order in which the M&M pieces were processed was the same for the three evaluated versions.

Exp. No.	Version 1	Version 2	Version 3
1	04:44	04:17	04:10
2	04:54	04:08	04:02
3	04:40	04:09	04:07
4	04:49	04:11	04:20
5	04:52	04:08	04:11
6	05:02	04:09	04:15
7	04:50	04:12	04:13
8	04:44	04:10	04:08
9	04:59	04:09	04:06
10	04:52	04:11	04:08

Table 2: Processing time (in mm:ss) for standard test package (90 pieces, 15 of each color)

From the results shown in Table 2, it can be seen that the TPN performance evaluation was quite precise. For version 1 the expected average time was $04:46$s while in the experiments the average time was $04:50$s. For version 2 the expected average time was $04:11$s while in the experiments the average time was $04:10$s, almost the same of the version 3 considering randomness nature of M&M pieces in the storage module.

Despite the different design approaches and implementations in version 2 and 3 there was no significant differences in the system performance. Both versions met all the specifications provided for the system.

Considering all the materials and devices, 3D printed parts, plastic and metallic materials, electromechanical devices, sensors and the microcontroller board, the project cost was US$ 65.32.

6 Conclusion

The proposed approach led to an implementation using a low-cost commercial *Arduino* board in which all system specification were met. Even though there was no performance gain between the version adapted from (Wiki, 2017) and the one proposed here, both of them were successful attending the systems requirements.

The solution proposed here used fewer and simpler functions to implement the solution, which can be seen as an indicator of how easy will be to maintain the system. Basic functions like the ones proposed here tend to have similar execution flows regardless of the context because they do not need to identify the state of the device to determine the action that will be performed. Such characteristic implies in a more predictable execution time, as well as less difficulty in verifying its operation on a given hardware.

Considering that, the class of problems addressed here are linked to the execution of specific tasks generally connected to higher level devices in charge of coordinating and integrating the functioning of smaller systems, the solution of the automation problem tends to adopt the centralized paradigm.

The method proposed here proved to be satisfactory to solve the problem of automating low-cost systems, ensuring the proper functioning through the use of formal verification techniques.

Although when a case study in which there was no need to perform periodic tasks - which is common in embedded systems - was used to show the proposed method, it does not mean that the proposed approach would not work for that kind of systems. In (Salmon et al., 2014) it was shown how to model periodic tasks using TPN, verify quantitative temporal formulas using the observer approach in order to synthesize an scheduler based on task priorities. Such approach can be used in the context here proposed.

Acknowledgements

The authors want to thank Prof. José Reinaldo Silva for valuable comments on a preliminary draft of this manuscript.

References

Berthomieu, B., Ribet, P. O. and Vernadat, F. (2004). The tool tina – construction of abstract state spaces for petri nets and time petri nets, *International Journal of Production Research* **42**(14).

Berthomieu, B. and Vernadat, F. (2006). Time petri nets analysis with tina, tool paper, *Proceedings of 3rd Int. Conf. on The Quantitative Evaluation of Systems*, IEEE Computer Society.

del Foyo, P. M. G. and Silva, J. R. (2011). Some issues in real-time systems verification using time petri nets, *Journal of the Braz. Soc. of Mech. Sci. and Eng.* **33**(4): 467–474.

Hajjar, S. (2013). *Safe Design Method of COTS based embedded systems based on COTS*, PhD thesis, INSA de Lyon.

Henzinger, T. A. and Sifakis, J. (2006). The embedded systems design challenge, *Proceedings of the 14th International Symposium on Formal Methods (FM)*, Springer, pp. 1–15.

Lapshina, P. D., Kurilova, S. P. and Belitsky, A. A. (2019). Development of an arduino-based co2 monitoring device, *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 595–597.

Liu, K., Kong, W., Hou, G. and Fukuda, A. (2018). A survey of formal techniques for hardware/software co-verification, *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pp. 125–128.

Merlin, P. and Faber, D. (1976). Recoverability of communication protocols– implications of a theoretical study, *IEEE Transactions on Communications, [legacy, pre-1988]* **24**(9): 1036–1043.

Mukherjee, S., Ghosh, A. and Sarkar, S. K. (2019). Arduino based wireless heart-rate monitoring system with automatic sos message and/or call facility using sim900a gsm module, *International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, pp. 1–5.

Panda, P. (2001). Systemc: A modeling platform supporting multiple design abstractions, *Proceedings of the International Symposium on Systems Synthesis (ISSS)*, ACM, pp. 75–80.

Ramadge, P. J. and Wonham, W. M. (1989). The control of discrete event systems, *Proceedings of the IEEE* **77**(1): 81–98.

Rumbaugh, J., Jacobson, I. and Booch, G. (2004). *The Unified Modeling Language Reference Manual*, second edition edn, Addison-Wesley.

Salmon, A. Z. O., del Foyo, P. M. G. and Silva, J. R. (2014). Scheduling real-time systems with periodic tasks using a model-checking approach, *12th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 73–78.

Sifakis, J. (2005). A framework for component-based construction, *Proceedings of the Third International Conference on Software Engineering and Formal Methods (SEFM)*, IEEE Computer Society, pp. 293–300.

Syazlina Mohd Soleh, S. S., Som, M. M., Abd Wahab, M. H., Mustapha, A., Othman, N. A. and Saringat, M. Z. (2018). Arduino-based wireless motion detecting system, *2018 IEEE Conference on Open Systems (ICOS)*, pp. 71–75.

Wiki, I. (2017). Skittles m&m's sorting machine, `https://beta.ivc.no/wiki/index.php/Skittles_M&M's_Sorting_Machine`. Accessed: 2020-04-15.

Wilmshurst, T. (2007). *Designing Embedded Systems with PIC Microcontrollers*, second edn, Elsevier.

Xie, F. and Liu, H. (2007). Unified property specification for hard- ware/software co-verification, *Computer Software and Ap- plications Conference*, pp. 483–490.