Proceedings in Manufacturing Systems, Volume 19, Issue 1, 2024, 17-24 ISSN 2067-9238

# ON FLEXIBLE MANUFACTURING SYSTEM CONTROLLER DESIGN AND PROTOTYPING USING PETRI NETS AND MULTIPLE MICRO-CONTROLLERS

**Konstantinos PETROPOULOS[1], George-Christopher VOSNIAKOS[2,*], Emmanuel STATHATOS[3]**

[1)] MSc graduate student, School of Mechanical Engineering, National Technical University of Athens, Athens, Greece
[2)] Prof., Manufacturing Technology Laboratory, School of Mechanical Engineering, National Technical University of Athens, Athens, Greece.
[3)] Dr, Senior postgraduate researcher, School of Mechanical Engineering, National Technical University of Athens, Athens, Greece.

***Abstract:*** *This work aims to design the central controller of a Flexible Manufacturing System and test it beyond simulation. The FMS controller is modelled through standard PN formalism tested by simulation before subsequent controller prototyping. Then, prototypes are implemented in two alternative schemes. First, by a central controller and local micro-controllers of Arduino™-type, in a master-slave configuration, communication of micro-controllers with each other materializing through asynchronous data transmission over the I2C bus. Second, by replacing local controllers by light emitting diodes (leds) and switches for 'receive' and 'send' signals respectively. The central controller was implemented in microcontroller language by direct conversion from the respective simulated and analyzed Petri Net. Execution of the developed control programs was performed and evaluation proved that the developed prototyping method is efficient, low cost and scalable in a system-commissioning context.*

***Key words:*** *Flexible Manufacturing System; Petri Nets; Microcontroller; I2C protocol; serial communication; Discrete events; Hierarchical controller.*

## 1. INTRODUCTION

There are several approaches in designing the controller of a Flexible Manufacturing System (FMS), all linked to the discrete event paradigm according to a formalism that may range from Petri Nets (PNs) to rule-based and fuzzy sets [1].

PNs have been the prominent approach to FMS controller design due to their theoretically founded ability to model complex discrete event systems effectively. They consist of places representing conditions or states of machines, buffers, and parts within the system, transitions representing events or actions pertaining typically to manufacturing processes or part handling processes, and tokens representing the presence of a resource or the activation of a state, thereby modelling the dynamic state changes of the system. PNs enable various types of analysis to ensure the system's correctness and performance: reachability, liveness, boundedness and deadlock detection. In capturing dynamics of real-time control of FMS elements characterizing queuing network configuration and scheduling policy depend on the current state of the FMS [2]. Furthermore, a system model can be enriched with elements collected from the context, which optimizes the design of constraints that can then be integrated to control frameworks for synthesis and code generation[3].

Controller design using PNs encompasses different research approaches. Several typical methods for structural simplification techniques of FMS supervisory controllers have been developed in [4]. Optimal PN controllers for a special class of systems with simple sequential processes with so-called ξ-resources were developed in [5]. A single control place for each concurrent process of a PN model was designed in [6] optimally enforcing liveness and ensuring high resource utilization. Place-Transition controller is designed for each concurrent process of the system in [7] creating extra states in the reachability graph so that the resulting supervisory structure controls deadlock occurrence without restricting system operation.

Design is followed by physically implementation of the controller, e.g. making use of Programable Logic Controllers (PLCs), industrial PCs or other types of computing equipment and then, commissioning and testing it. Commissioning an FMS controller involves several steps to ensure that the system operates correctly and meets the required performance standards. Within commissioning, testing takes prominent place, involving functional, end-to-end, performance, safety / reliability and Final Acceptance testing.

Physical commissioning is a costly process, thus it makes sense to perform most of the testing in simulation mode or even in a hybrid 'hardware-in-the-loop' mode, as a step of Virtual Commissioning (VC) [8]. A brief State-of-the-Art (SoA) of FMS controller testing within VC is provided in Section 2. In this work, an FMS that is small enough to be comprehended and large enough to justify non-trivial control is used as a testbed, see Section 3. Its controller is developed using classical PNs and dedicated simulation software, see Section 4. The FMS controller is implemented on microcontroller hardware to be physically tested and the FMS station controllers are replaced by simple microcontrollers or a switch /

[*] Corresponding author: Heroon Polytehniou 9, Athens 15773, Greece
Tel.: +30 210 7721457.
E-mail address: *vosniak@central.ntua.gr* (G.-C. Vosniakos).

photodiode pair. These schemes are presented in Section 5. Conclusions and outlook are articulated in Section 6.

## 2. SoA FOR FMS CONTROLLER TESTING IN VC

A real-time event-based digital platform is presented in [9] allowing all physical objects, virtual models, and industrial systems to communicate and integrate with each other. As another example, a real-time co-simulation platform for VC included the integration of powerful technology-specific simulation solutions based on integration interfaces and a real-time co-simulation architecture based on partitioning, parallelization, synchronization and data exchange mechanisms [10]. A methodology is presented in [11] for the development and implementation of virtual and hybrid scenarios by using highly integrated, digital manufacturing tools and was implemented successfully by virtual and hybrid commissioning scenarios to develop smart factories.

A recent VC trend in has been the use of Digital Twins (DTs) of equipment or subsystems. DTs can enhance the system reactivity to uncertain events by getting data from the field and triggering actions on the physical asset.

The concept of a DT-based virtual factory is presented in [12] and its architecture to support modelling, simulation and evaluation of manufacturing systems while employing virtual reality (VR) learning/training scenarios. Verification of developed DT architecture and integration in the manufacturing system is implemented in [13] in a virtual environment, thus supporting VC. In the same context, a DT-based remote semi-physical commissioning approach for flow-type smart manufacturing systems is proposed and validated through a case study of a smartphone assembly line [14].

The interconnection between a physical production system and its DT is examined [15], focusing on required sensors and the number of synchronization points to realize a dedicated control by the digital twin.

Four commercial industrial simulation software systems supporting VC function are compared in [16] in the context of constructing the FMS DT with a commercial master control system for VC.

## 3. THE FMS STUDIED

The FMS that was studied is shown schematically in Fig 1. Note that this layout is the intended one, since only 3 processing stations (CNC lathe, CNC mill, robot) are currently present whilst auxiliary stations (transport, identification-ID, palletizing, buffers, storage) are in the implementation stage. There are 24 stations in total, belonging to 4 groups: common, pallets, parts and tools highlighted in red, blue, green and brown, respectively

The processing stations employed are: a two-axis EMCO™ COMPACT 5 CNC lathe, a 3-axis EMCO™ F1 CNC mill and a 5-axis Mitsubishi™ RM501 robot. These stations have undergone radical upgrading of motors, electronics and controllers, now employing LinuxCNC™ [17–19].

Storage_start (station 13, 6 for parts and pallets respectively), Storage_finish (station 19, 12 for parts and pallets respectively) and Tools_storage (stations 1, 2 for milling and turning, respectively) are hosted in a 15-slot Automatic Storage and Retrieval System (AS/RS) which is served by a cartesian robotic mechanism [20].

The mill (station 23) and the lathe (station 24) possess intermediate buffers for part and pallet storage before and after processing (stations 14, 15, 7, 8 and 17, 18, 10, 11 respectively). They also possess a tool buffer each (station 3, 4, respectively).

The robot (station 21) is responsible for loading / unloading the lathe and mill with parts and pallets. It also moves parts and pallets between the automatic (vision-based) identification station (ID – station 22) and an intermediate buffer for storage of unidentified parts (station 22) and pallets (station 9). Its reach is not enough to cover the aforementioned stations thus it seats on a linearly moving platform.

Manual labour is required for feeding unprocessed parts, empty pallets, milling tools and turning tools (stations 13, 6, 1, 2 respectively) into the system or taking finished parts, full pallets, used milling and tuning tools away (stations 19, 12, 1, 2 respectively), for palletizing and de-palletizing parts (station 5), for loading and unloading milling and turning tools from/to the tool buffers (stations 3 and 4, respectively).

The movement of parts, pallets and tools between the AS/RS and the intermediate buffers is undertaken by a transport system (station 20), which is envisaged to be a mobile robot.

Aspects of the available equipment of the FMS are shown in Fig. 2.

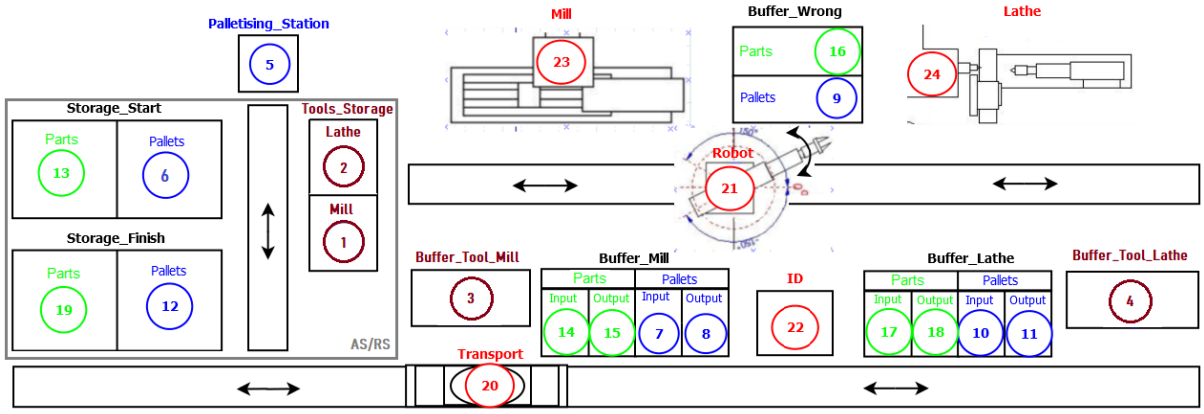



**Fig. 1.** General layout of the FMS studied.

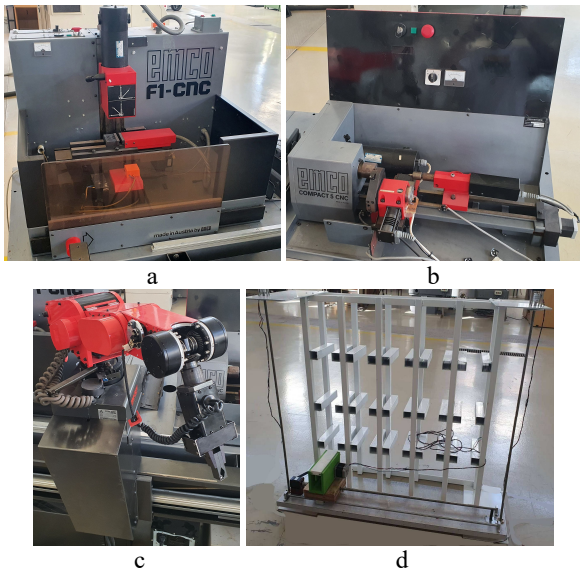



**Fig. 2.** FMS stations: *a* – mill; *b* – lathe; *c* – robot; *d* – AS/RS.

Note that the capacity of buffers, storage areas, palletizing station and transport are parametric and thus user-defined. Buffer capacities are by default equal to 1. The robot, by default, moves one part or one tool or one pallet. Generally, movement or transporting priority is highest for tools, followed by pallets and last by parts, since (a) no machine can work without tools and (b) pallets generally contain more than one parts or tools, promoting productivity.

Six process plans are supported, involving one to three sequential operations that can be of type turning (T) or milling (M). Operations in each process plan have user-defined duration. The process plans are: T, T-M, T-M-T , M, M-T, M-T-M.

The production plan consists of the process plan and the number of parts or pallets, the number of turning and milling tools required.

## 4. FMS CONTROLLER DEVELOPMENT

The PN corresponding to one of the 6 process plans, in fact a simple one of the type T-M for reasons of comprehensibility, is shown in Fig. 3. Different number and/or sequence of operations, as in the rest 5 process plans, would necessitate construction of a different PN. However, the principles of constructing any PN remain the same. The PN was designed using the open software Platform Independent Petri Net Editor v4.3.0 [21].

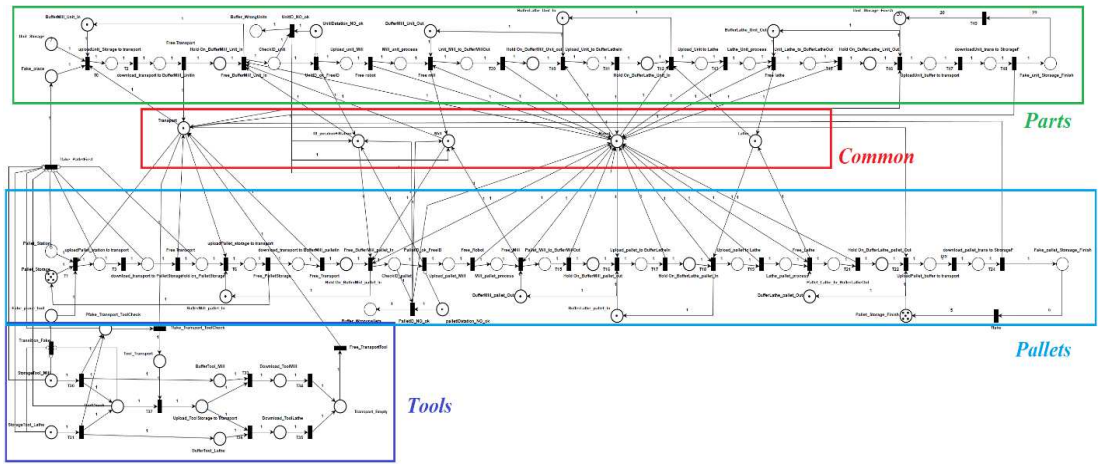
### 4.1. PN structure

This particular PN consists of 72 Places and 52 Transitions. The Places are grouped following the same coding (Common, Parts, Pallets and Tools) used in defining the FMS, compare Fig. 3,*a* and Fig. 1.

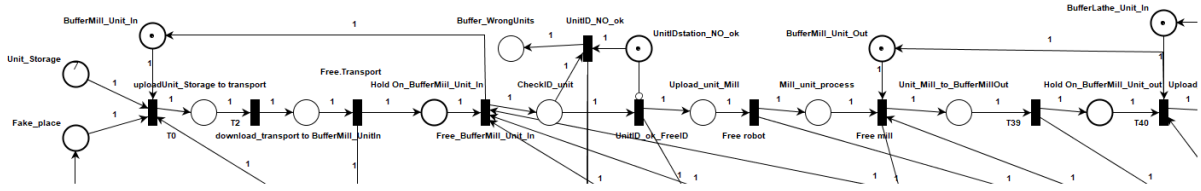
Each one of the groups of Places of the PN is shown in magnification in Fig. 3,*b*–*g*.

In their overwhelming majority, Places represent states of the FMS but, in addition, some "fake" Places have been used as they serve token flows or impose priorities. Further to normal arcs forming the input to Transitions, some inhibitor arcs have also been used as negative conditions of Transition firing. Examples will be shown in Section 4.2.

Places may represent availability of a resource, e.g. Robot, Mill, Lathe, Transport, ID_Position+Station in Fig. 3,*d*, availability of a buffer slots up to maximum capacity as denoted by the number of tokens, e.g. BufferMill_Unit_In (1), Pallet_Storage_Finish (5), BufferMill_pallet_Out (1) in Figs. 3,*b*, *c* and *e* respectively).




a




b

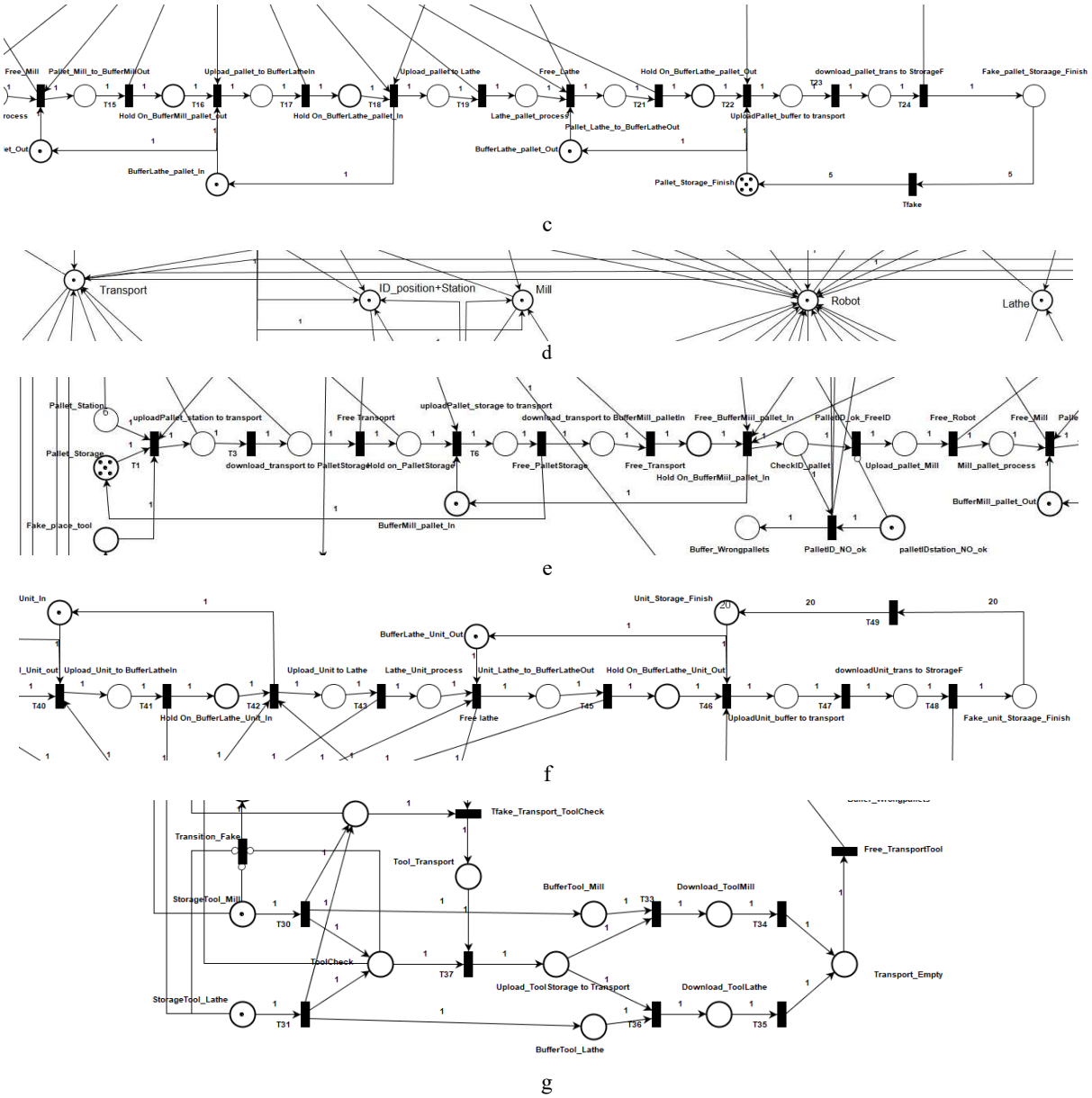



**Fig. 3.** Petri Net developed: *a* – General view; *b–c* – Parts detail; *d* – Common detail; *e–f* – Pallets detail; *g* – Tools detail.

Places may represent available parts to be processed, e.g. Unit_storage in Fig. 3,*b* (max capacity 20 parts in this case), or available tools, e.g. StorageTool_Mill in Fig. 3,*g* (max capacity 5 tools in this case). The number of tokens is set by the user, typically as an initial condition of the simulation.

Some Places may have a variable (infinite) capacity, e.g. Buffer_Wrongpallets, in Fig. 3,*e* as the number of pallets that are misidentified is not known beforehand.

Places may represent a "waiting" type activity of unknown duration, e.g. HoldOn_BufferMill_Unit_In, HoldOn_BufferLathe_Unit_In, in Fig. 3, and *f*, respectively.

Places may represent an activity of known duration, generally loading and unloading parts, pallets and tools and processing parts and pallets e.g. Upload_Pallet_Mill, Mill_Pallet_Process in Fig. 3,*e*, Download_ToolMill in Fig. 3,*g*.

In some cases, auxiliary Places (termed "fake") have been used to impose priorities, e.g. Fake_place_tool in Fig. 3,*e* has been defined with capacity 1 signalling completion of tool transport so that pallet transport can take over in connection to inhibitor arcs in Transition 'Fake' in Fig. 3,*g*. Similarly, auxiliary Place 'palletIDstation_ NO_ok' in Fig. 3,*e* has been defined with capacity 1 to signal that the pallet has been misidentified. Thus, by use of an inhibitor arc in the antagonist Transitions PalletID_No_OK, PalletID_OK_Free_ID the pallet (token) is directed to Buffer_Wrongpallets.

As for the Transitions involved in the PN model, in general these signify start / end events or triggers for changes of state, i.e. tokens appearing in / disappearing from the respective Places.

In fact, for each one of the 52 Transitions defined in the system the set of the firing rules (conditions and results) has been defined taking into account the input Places, the output Places, the type of arcs (normal or

inhibitor) and the arc weights. This is the backbone of the implementation of the PN into a controller program.

### 4.2. PN simulation

The PN was simulated in the PIPE environment [21]. The initial marking comprises, in addition to default values for availability of resources, storage and buffers, of 6 pallets and 8 parts in the 20-slot store, 2 milling tools and 3 turning tools in the 10-slot respective storages. The PN is 20-bounded since no place acquires more than 20 tokens irrespective of the initial state-marking. This is largely due to the capacity restrictions imposed in the Places.

In numerous initial and intermediate markings a number of transitions can fire mostly due to their independence (concurrency) or in some cases as a result of conflict. i.e. being fed by the same Place(s). Random selection of the Transition to fire within that set is adopted in all such cases. The simulation ends when all parts and pallets have been processed with the following overall statistics:

According to PIPE's mathematical analysis, the PN is not covered by positive T-Invariants, therefore we do not know if it is bounded and live. In addition, the PN is not covered by positive P-Invariants, therefore we do not know if it is bounded. Minimal siphons are: palletIDstation_NO_ok and Minimal traps: Buffer_Wrongpallets. Simulation stopped after 164 transition firings, 10 replications and average number of tokens was 95%. Two characteristic snapshots of the token flow are shown in Fig 4.

In particular, Fig. 4,*a* depicts movement of a pallet from Pallet_Station to Pallet_Storage by the Transport. The respective three Places have to contain at least one token and so does Fake_place_tool, to confirm that all processes related to tools have been completed.

Figure 4,*b* depicts firing of the Transition that loads a pallet from the buffer of the Mill to the Mill itself by means of the robot. It has to pass first through the identifications station ID_position+station. Upon firing of this Transition, the identity of the pallet has to be checked in order to be allowed processing on the Mill. In addition, the Mill's buffer is emptied.

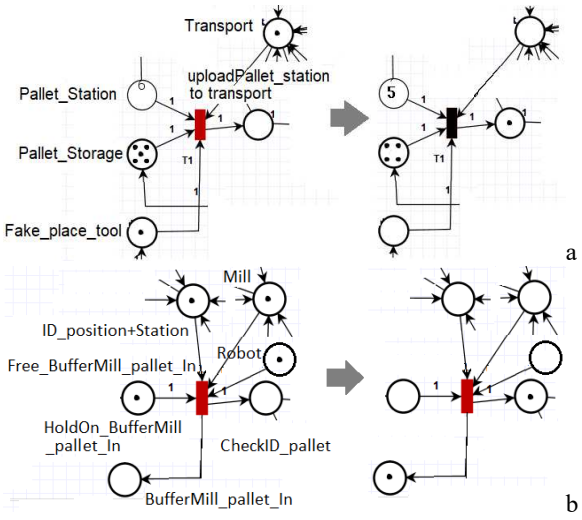



**Fig. 4.** Characteristic Transition firings: *a* – pallet movement to Pallet_Storage; *b* – pallet movement to Mill.

## 5. FMS CONTROLLER PROTOTYPING

Each FMS station has specific inputs - outputs (signals) and a local controller, which hierarchically activates further software to perform specialized functions, such as LinuxCNC™ or other software based on which each FMS station operates. The central controller executes a control program resulting from the conversion of the corresponding PN. In this context it must communicate with the respective local controllers of the individual stations through appropriate signals.

It is straightforward to translate each transition of the PN into a rule containing preconditions for firing and token transactions upon firing. Thus, the PN is translated into a control program running on the central µ-controller. This is executed in the same way as in the simulator, e.g. PIPE, the main difference being that token addition to or deduction from a Place that corresponds to an FMS station (slave) is regarded as a change in the value of a respective variable in the slave controller. Similarly, checking for presence or absence of a token in a place representing an FMS station is equivalent to reading the value of the respective variable in the FMS station slave controller concerned.

The control logic follows a loop, until the end of the functional scenario, consisting of the following steps:

- The master controller asks the slave controllers to report whether their availability or not. Slaves do so after confirming with their respective FMS station, or in the absence of a real or computer-controlled station with the human user.
- The master controller executes the control program serially, i.e. it checks for each transition of the Petri net if all the relevant selection conditions (if conditions) constituting input branches, are satisfied and creates a list of possible activations. If there are conflicts, one transition is randomly selected.
- Once a transition is selected, the commands that make up its output branches are executed. These include engagement or release signals of the FMS stations, which are sent by the master to the slaves, which act accordingly.

Engagement can mean running a CNC program on the Lathe, Mill or Robot stations, which lasts for a specified period of time. The duration of execution of programs or actions, given that there are no real stations but only their simplified slave controllers, is determined by timing commands of the type: delay (duration). In this way, the controller acquires a 'sense' of time, even though the concept of time does not exist in the classic PN, where the controller is based. In the case of storage, the engagement is implemented by converting a storage location into a reserved one, that is, by simply changing the value of the corresponding variable. Similarly, station release is interpreted and implemented. Generally, two values denoted as LOW and HIGH are used: LOW means the station is available (released) or has a token (as interpreted in PNs) and HIGH means reserved.

### 5.1. Master-slave scheme

To implement the FMS control prototype in this way, microcontrollers as opposed to microprocessors or PLCs are sufficient for the following reasons: simplicity of use

and ease of programming, connectivity with peripheral circuits and with each other, low cost, access to individual add-on components (shields) and libraries.

**5.1.1. Hardware.** An Arduino™-Mega 2560 Rev3 microcontroller was used as the central controller and Arduino™ Uno Rev3 microcontrollers as local controllers. These do support connectivity and simple control of many peripheral devices together, which involves mere switching of output-input signals between High and Low.

The main features of the Arduino™ Mega are: an ATMega328P processor, 14 digital I/O pins, of which 6 can be PWM (Pulse Width Modulation) outputs and the 6 analog I/O pins. In addition, when using the serial monitor for communication (Serial Monitoring) with the computer, pins 0 and 1 act as RX (Receiver) and TX (Transmitter). There are also 2 small leds with the same name (RX and TX), which light up accordingly for serial data reception or transmission. Most of the functions of the Arduino™Mega are the same as those of the Arduino™Uno. Their main differences are: the microcontroller (ATmega2560), RAM (8 KB), Flash Memory (256 KB), EEPROM (4 KB), higher number of I/O ports (54 digital and 16 analog) as well as additional pins for 5 V output power supply and ground (GND). It also contains 4 UART serial ports (0-1,14-15,16-17,18-19) and a 16 MHz crystal oscillator.

Microcontrollers communicate asynchronously with each other in a multimaster-slave scheme according to the UART standard. In particular, the I2C protocol followed is a multi-master, serial, single-ended bus, referred to as a two-wire interface. It can connect up to 127 nodes at low speed with two lines, namely SCL, the clock line used for synchronization, and SDA, the data line. There is, of course, a ground (GND) connection and there may be a 5V supply line, see Fig.5.

When the master (central) controller wishes to communicate with a slave (local) controller, it sends an appropriate sequence of pulses on the SDA and SCL lines. The data includes the address of the slave (7 bits + 1 bit that determines whether it is send or receive. Via the analog ports SDA (A4/20) and SCL (A5/21) it is possible to connect the microcontrollers (Arduino™Uno and Arduino™Mega, respectively).

Indicatively, the implementation using one master controller – central and two slave controllers corresponding to the robot and the Mill is presented, see Fig. 6. Further to the I2C connections through a breadboard, three light emitting diodes (leds) were used, namely yellow for the central controller, green for the robot and red for the Mill slave together with 220 Ω protection resistors. They served for visual monitoring of serial data transmission between master and slaves, as well as for loop cycle monitoring.

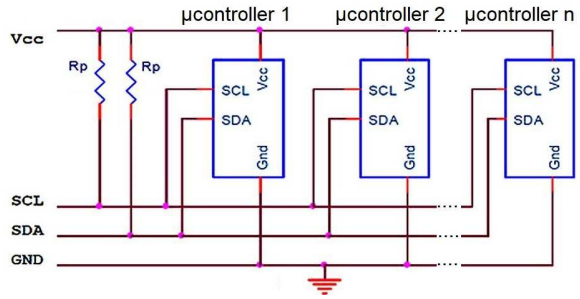



**Fig. 5.** I2C Connectivity.

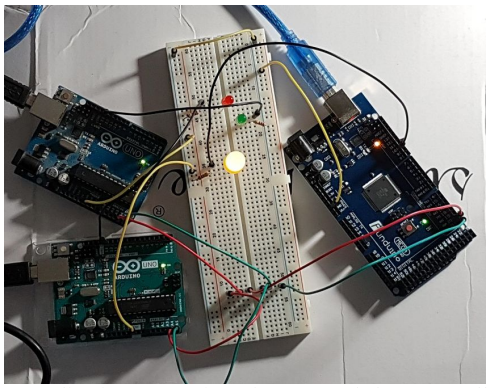


**Fig. 6.** Indicative connection of master and slave controllers.

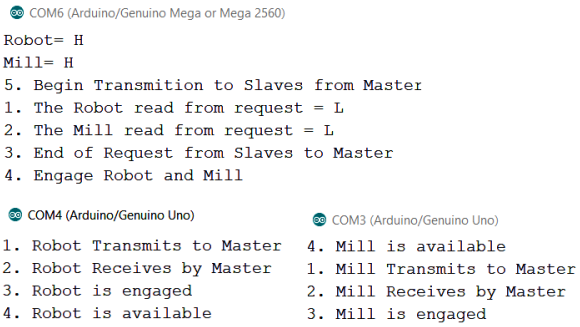


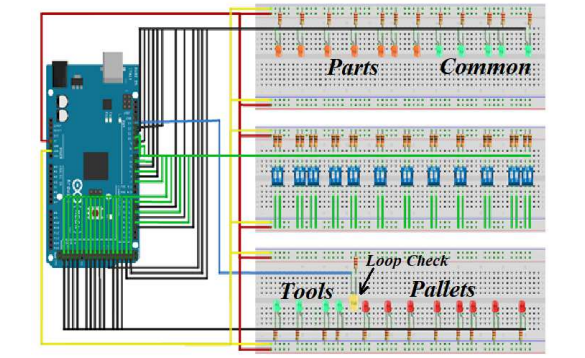

**Fig. 7.** Indicative master-slave controllers communication flow.

**5.1.2. Control program prototypes.** The controller programs were written in Arduino IDE v.1.8.5, the master controller running the PN-based control logic whilst the slave controller programs were exactly the same, consisting of 67 lines each. The programs made used of the ready-made I2C library, the Wire library, in particular the following basic commands: begin(address), requestFrom(address,quantity,stop), beginTransmission(address), endTransmission(stop), write(val), available(), read(), setClock(), onReceive(handler), onRequest(handler). Serial communication events among the controllers were printed out by the respective controller initiating or responding to the particular requests, for purposes of documentation, see Fig. 7.

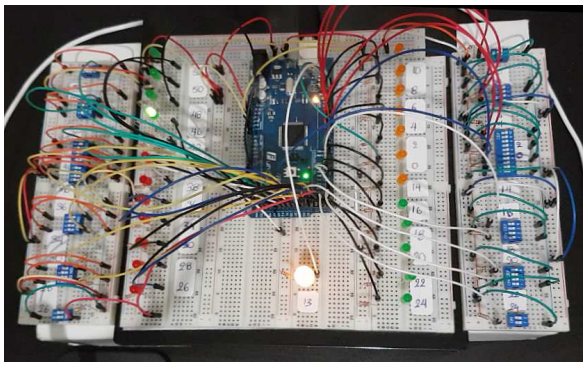
### 5.2. Photodiode-switch pairs as local controllers

As an alternative, the Arduino™Uno local controllers were replaced by pairs of a switch and a led. The led provides information on the state (HIGH/LOW = ON/OFF= token presence/absence) of the FMS station that the local controller is associated with, whereas the switch is the means to change that state.

Thus, state changes do not result from a command by the master controller as in the I2C connection, but happen interactively. First, visual output is created to inform the user, using both leds and textual communication, so that he/she can then manually change the state of the switch following additional commands he/she receives.




a



b

**Fig. 8.** Connection of central controller to led – switch pairs representing slave controllers: *a* – schema; *b* – hardware.

**5.2.1. Hardware.** The master controller directly reads an input (from respective Input pins) from its (+5V) supply. A two-position (On/Off) switch is inserted between the power supply and the input. The Arduino™Mega central controller was combined with 24 switches and 24 LEDs that correspond to the 24 positions of the PN representing assets (machines, robots, storage, buffers), see Fig. 8,*a*.

Note that the leds have different colors according to the grouping of the PN Places, see Fig. 3,*a* and Fig. 8,*a*. Also, 220 Ω resistors are used to protect each led and 10 KΩ pull-down resistors for the switches, so they do not 'float' when open. One large yellow led is on as soon as the central controller enters the control loop and off at the end of it.

**5.2.2. Controller prototype programming.** First, the following are initialized:

- The switches/stations that make up the inputs and correspond to specific input Pins of the microcontroller.
- The variables that read the switches of the FMS stations. They have the station/input name prefixed by "switch".
- The leds that make up the outputs and correspond to specific output Pins. They have the station/input name prefixed by 'Led'. Each Led is right next to the corresponding switch, see Fig. 8,*b*.

Note the following in the logic of the control loop:

- First all inputs/switches are read and registered in the corresponding switch-variables, through the digitalRead() command.

```
        --------The contoller read all the Inputs--------
I       BufferTool_Mill (Token=1 - switch50=OFF) is available
II      Tool_Mill is uploaded to Transport
III     Tool_Mill is empty (Token=0 - switch52=ON)
IV      Transport is engaged (Token=0 - switch24=ON)
V       Tool_Mill is downloaded to BufferTool_Mill
VI      BufferTool_Mill (Token=0 - switch50=ON) is engaged
VII     Transport is available (Token=1 - switch24=OFF)
```

**Fig. 9.** Indicative interaction flow in serial port.

- The following general principle applies to the state of switches (ON/OFF) and leds (HIGH/LOW): "If the position has token Token = 1, then Led = LOW and switch = OFF, regardless of whether the token means 'available' or 'engaged' or any number of some kind.'
- The result of the activation rules is a command to the user to change within 15 secs (delay(15000)), the state of the switch (ON/OFF), based on the state of led (HIGH/LOW) and a corresponding instruction in the interaction window.

The corresponding control program consists of 976 lines of code. The following interaction flow is shown as an example, see Fig. 9 for textual output on the serial port:

I. Display Led BufferTool_Mill = LOW, Message: switch50 = OFF (Release).
II. Message: Tool_Mill is loaded into Transport
III. Display LedTool_Mill = HIGH, Message: switch52 = ON, because Token = 0 (Token Allocation = 0 now). So, the switch will remain ON and the Led will remain HIGH.
IV. Display LedTransport = HIGH, Message: switch24 = ON, because Token = 0 (Commitment).
The inputs are re-read due to changed state.
V. Message: Tool_Mill is unloaded to BufferTool_Mill
VI. Display Led BufferTool_Mill = HIGH, Message: switch50 = ON, (Commit) and it stays like this, as the Mill tool buffer is not released.
VII. Display LedTransport = LOW, Message: switch24 = OFF, because Token = 0 (Release).

## 6. CONCLUSIONS

Control of an FMS is systematically and reliably designed and tested based on PN. In oθρ case, classical simple, non-timed networks were used, but they do not provide the flexibility required for the introduction of new process plans. As a result, a different control program must be run on the central controller for each product mix. Therefore, to cover all possible product combinations the corresponding PN should be very large. Alternatively, individual PNs can be created for individual product combinations that must be foreseen from the outset. However, the conversion of such PNs into a control program is simple and direct. Alternatively, a more elegant solution that provides flexibility and small PN size is colored networks. Their transformation into a control program is possible, but not as direct as in the case of classical PNs [22].

Given the PN and the corresponding program of the central microcontroller, a viable solution with its fully automated communication with the FMS stations is to connect each station to the central micocontroller through

a corresponding slave microcontroller. In this context, exploitation of the I2C communication bus has the following positive characteristics:

- The central controller is defined directly as master and the others as slaves, without the need to write code defining the relationships of these devices.
- The connection to each device is made through 3 cables only (SDA, SCL, GND).
- The speed of information transmission, through the I2C serial bus, is satisfactory because although response is not instantaneous, only simple High/Low signals are transferred.
- The control program executed on the slave microcontrollers is exactly the same, i.e. standard, if only switching between two states (on / off) is required. Otherwise, if further functions are required that the central controller must know of, which, in the general case, are different for each station, the standard program has to be augmented / modified accordingly. However, it should be noted that in most cases the central controller is covered by only these two states, and all the rest possible states of the FMS station are internal to the local controller.

The alternative implementation that was tried by replacing the peripheral slave microcontrollers with led-switch pairs is oriented towards the involvement of the human factor, essentially as an operator of the assets. The central controller communicates with human on one hand and the latter, in turn, communicates with the individual local controllers of the FMS stations, in a 'human-in-the-loop' mode. Therefore, such a model makes sense either if FMS stations are manual, or for testing and debugging purposes before replacement by an automated FMS control model.

## REFERENCES


[1] A. Florescu, S.A. Barabas, *Modeling and simulation of a flexible manufacturing system—A basic component of industry 4.0*, Applied Sciences, 10, 2020, p. 8300 (20).

[2] M.P. Fanti, B. Maione, G. Piscitelli, B. Turchiano, *System Approach to the Design of Generic Software for Real-Time Control of Flexible Manufacturing Systems (FMS)*, Chapter 1 in: Computer-Aided Design, Engineering, and Manufacturing, CRC Press, 2019, p. 29.

[3] A.L. Silva, R. Ribeiro, M. Teixeira, *Modeling and control of flexible context-dependent manufacturing systems*, Inf Sci (N Y), 421, 2017, pp. 1–14.

[4] H. Hu, Y. Liu, L. Yuan, *Supervisor simplification in FMSs: Comparative studies and new results using PNs*, IEEE Transactions on Control Systems Technology, 24, 2015, pp. 81–95.

[5] H. Liu, W. Wu, H. Su, Z. Zhang, *Design of optimal Petri-net controlers for a class of flexible manufacturing systems with key resources*, Inf Sci (N Y), 363, 2016, pp. 221–234.

[6] M. Bashir, Z. Li, M. Uzam, A. Al-Ahmari, N. Wu, D. Liu, T. Qu, *A minimal supervisory structure to optimally enforce liveness on PN models for flexible manufacturing systems*, IEEE Access, 5, 2017, pp. 15731–15749.

[7] M. Bashir, J. Zhou, B.B. Muhammad, *Optimal supervisory control for flexible manufacturing systems model with Petri Nets: A place-transition control*, IEEE Access, 9, 2021, pp. 58566–58578.

[8] N. Striffler, T. Voigt, *Concepts and trends of virtual commissioning–A comprehensive review*, J Manuf Syst, 71, 2023, pp. 664–680.

[9] C.E.B. López, *Real-time event-based platform for the development of digital twin applications*, The International Journal of Advanced Manufacturing Technology, 116, 2021, pp. 835–845.

[10] C. Scheifele, A. Verl, O. Riedel, *Real-time co-simulation for the virtual commissioning of production systems*, Procedia CIRP, 79, 2019, pp. 397–402.

[11] M. Hincapié, A. Valdez, D. Güemes-Castorena, M. Ramirez, *Use of laboratory scenarios as a strategy to develop smart factories for Industry 4.0*, International Journal on Interactive Design and Manufacturing (IJIDeM), 14, 2020, pp. 1285–1304.

[12] E. Yildiz, C. Møller, A. Bilberg, *Virtual factory: digital twin based integrated factory simulations*, Procedia CIRP, 93, 2020, pp. 216–221.

[13] G. Barbieri, A. Bertuzzi, A. Capriotti, L. Ragazzini, D. Gutierrez, E. Negri, L. Fumagalli, *A virtual commissioning based methodology to integrate digital twins into manufacturing systems*, Production Engineering, 15, 2021, pp. 397–412.

[14] J. Leng, M. Zhou, Y. Xiao, H. Zhang, Q. Liu, W. Shen, Q. Su, L. Li, *Digital twins-based remote semi-physical commissioning of flow-type smart manufacturing systems*, J Clean Prod, 306, 2021, p. 127278 (15).

[15] A. Ait-Alla, M. Kreutz, D. Rippel, M. Lütjen, M. Freitag, *Simulation-based analysis of the interaction of a physical and a digital twin in a cyber-physical production system*, IFAC-PapersOnLine, 52, 2019, pp 1331–1336.

[16] W. Sun, J. Wu, G. Xiao, Z. Jin, *Research on selection of commercial industrial simulation software oriented to virtual commissioning*, in: J Phys Conf Ser, 2021, p. 12052 (6).

[17] G.-C. Vosniakos, N. Zourtsanos, N. Kontogiannis, *Appreciation of CNC Technology Through Machine Tool Upgrading by an Open Controller*, in: Manufacturing Engineering Education, Elsevier, 2019, pp. 105–130.

[18] A. Tzanis, *Control of a machining center based on LinuxCNC*, Dissertation, Interdepartmental Postgraduate Course: Automation Systems, National Technical University of Athens, 2019, p.111. doi: 10.26240/heal.ntua.17629.

[19] D. Tsoumpas, *Control of an industrial robot based on open CNC software*, Final Year Dissertation, National Technical University of Athens, 2015, p.157, doi: 10.26240/heal.ntua.10091.

[20] E. Vavylousakis, *Design and control of a smart automatic storage and retrieval system serving a flexible manufacturing system*, Final Year Dissertation, National Technical University of Athens, 2022, p.156. doi: 10.26240/heal.ntua.24528.

[21] S. Tattersall, *Platform Independent Petri Net Editor*, https://sarahtattersall.github.io/PIPE/index.html (accessed August 15, 2024).

[22] A. Papazoglou, *Modelling of a flexible manufacturing cell by coloured petri nets and conversion to PLC program*, Dissertation, Interdepartmental Postgraduate Course: Automation Systems, National Technical University of Athens, 2017, p.158. doi: 10.26240/heal.ntua.5884.