A User-oriented Development Method in FSM supported Multiprocessor Embeded PLCs

***Abstract*—Programmable logic controllers (PLCs) are a base in automation, however applications become complex on logic and motion control mixed scenarios while the PC-based PLC has high price and complex system which can not meet the customized requirement of large equipments. The development of PLC has encountered bottlenecks. Hence, this paper presents a user-oriented development method. We pose a customized mulitprocessor $ePLC$ to enhance the performance, a multi-language supported uniform development platform to improve the adaptability of developers, an optimized system structure (reasonable memory allocation, user-oriented thread structure, proposed LPM data interaction, modular software design, finite state machines) to reduce the development complexity. Ultimately, we adopt the proposed method to implement the distributed control system on a 200 ton injection molding machine. By comparison with TECHMATION and KEBA system, the startup time of the implemented system has been increased by more than 20 times while the key performance is almost identical. In addition, the implemented system adopts the customized multiprocessor $ePLC$ and detached human machine interface (HMI).**

***Index Terms*—Multiprocessor, motion control, Injection Molding Machine, embedded PLC, User-oriented**

I. Introduction

Some concepts, such as smart factory, intelligent manufacturing [1], [2], and some technologies, such as Internet of Things, 5G, augment reality [3], [4] are paving the road of the fourth industrial revolution. Normally, a typical plant is full of large equipments, for instance, cranes, CNC machining centers, injection molding machines (IMMs), air pumps, chillers, automatic guided vehicles and types of robots and most of them are controlled by the PLC. PLCs have become the main control system. Numerous researchers are focusing on PLC technologies which extremely extends its application fields. [5]–[7] guarantee the reliability by verifying the program of PLCs, [8]–[10] improve the performance of PLCs using advanced algorithms, [11] alleviates the development complexity of PLCs with a special software structure, [12], [13] pose methods to update PLC programs dynamically. However, with the rapidly ever-growing demands and the tend of applications to be user-oriented and complex logic and motion control mixed [14], [15], PLCs still encountered bottlenecks, specially on large equipments (CNC machining center, IMM, etc.).

A. Motivations

To date, the hardware architecture of PLCs has two directions: $ePLC$s (embedded PLCs) and PC-based PLCs. PC-based PLCs are increasingly used on the applications of complex logic and motion control mixed scenarios on account of its high performance and lots of user-oriented tools [15]. Considering the IMM industry, Table I lists the composition of IMM, including parts(described as modules for programming), DI\Os and AI\Os. Normally, a simplest IMM system consists of 10 modules, 20 DIs, 30 DOs, 3 AIS and 7 AOs. Complex relations among them and high performance requirement of algorithms tremendously increase the difficulty of programming. Hence, as listed in Table II, the comparison of KEBA, BECKHOFF, GAFRAN and TECKMATION system which are the main brand of IMM controller illustrates that almost all of them are using PC-based PLCs. According to the complexity of the IMM system and the software architecture of PC-based PLC, the HMI is banded to PLC which leads to little independence of IMM manufacturers.

$ePLC$s have a wide area of applications in automation due to its easy programming and high reliability, however some disadvantages still limit its further development [15], especially coping with complex logic and motion control mixed applications. On the other hand, advances in fields of wireless communication, Internet of Things, etc. are accelerated requirement for low power consumption [16], which is an advantage of embedded PLC. How to integrate the low power consumption, easy programming and high reliability of $ePLC$ with high performance and usr-oriented development induces our research.

B. Related Works

Various researchers present methods to integrate motion control algorithms (e.g. linear interpolation, position control, arc interpolation, etc.) into PLCs [17]–[19]. Logic control and motion control become inseparable. Two ways exist to realize the integration: individual motion control module collaborated with PLC [20] and PLCs directly integrated with motion control functions [17], [21]. For the way of individual module, different kinds of modules and PLCs which are coded by different languages and developed in their own platforms tremendously increase the complexity to implement applications. [20], [22] and [23] all describe these modules. The second way simplifies the development method. Nevertheless, it is hard to guarantee high reliability logic control and high accuracy motion control simultaneously, since they run in the same thread.

We propose the concept of multi-processor $ePLC$ (multiple processor chips and multiple cores in one chip are all called multi-processor here). In this $ePLC$, extra processors (e.g. DSP, FPGA, etc.) are introduced to enhance the performance. Various technologies contribute to the improvement of multiple processor. [24], [25] pose data interaction methods among multiple processors, [26] presents a method to balance computing ability of the processors, [27] proposes a thread scheduling method in multiprocessors. All of these works are not implemented in $ePLC$ but inspire us to build the architecture of multi-processor $ePLC$. Moreover, some researches [28], [29] are be done to introduce additional high performance processors into $ePLC$s, though no improvement of development method is proposed for complex logic and motion control mixed applications.

In terms of development methods of the individual module, which is much convoluted, users should take a lot of time on selecting the platform and take more time on learning the particular software and its supported language. For instance, the PMAC is using C++ [20], [22], the MC421/221 of OMRON CS1 series is supported by G-Code [30], the FP series PLC of Panasonic is adopted special instructions embedded in LD (Ladder Diagram). Therefore, the uniform developing methodology in PLC platform attracts us. Since the 90's of the last century, IEC-61131 has been focused on the standardization of PLC [31]. In 2005, PLCopen organization has released a related standard [32] which standardizes the motion control in PLC and then papers, such as [33], made some interaction on it and companies, such as 3S [34], provide some tools. Howbeit, regarding some complex applications, programmers are occasionally prefer to use more popular or object-oriented languages (e.g. C, C++, etc.) [35]–[37] except the specified ones in IEC-61131-3. Recently, some methods, such as model-based software, component-based [38], [39] are also researched to reduce the complexity of PLC program. However, facing the complex control and motion control scenario, a more comprehensively improved development method still should be proposed. Hence, considering the popular concept of user oriented [40], [41], we present a user-oriented development method.

C. Our Contributions

To the best of our knowledge, the user-oriented development method should improve every aspect of the PLC system (development method, program, processor, RAM, thread) and propose a comprehensive optimization approach. Hence, we pose a flexible solution to enhance performance by adding sufficient processors, a multi-language supported graphical component to improve the adaptability of developers, an optimized system structure (reasonable memory allocation, user-oriented thread structure, LPM data interaction, modular software design, finite state machines) to reduce the development complexity. Ultimately, we adopt the proposed method to implement a distributed IMM system which is considered as a kind of complex logic and motion control mixed application.

This remaining paper is organized as follows. Section II introduces the system architecture, multi-language supported graphical component, memory allocation, usr-oriented multi-threading and modular design. In section III, we present the compilation of graphical component, LPM data interaction mechanism, the execution of multithreading and finite state machines. At last, in section IV, we implement the IMM distributed system with the posed method and compare it with the TECHMATION and KEBA system from aspects of system condition, system structure and key performance.

TABLE I
MODULES, DI/DO, AI/AO OF IMM

No.	Module	DI	DO	AI	AO
1	Mold	Safety valve	Mold close	Mold position	System pressure
2	Injection	Heating detection	Mold open	Injection position	System flow
3	Core	Servo alarm	Inject	Nozzle position	Back pressure
4	Nozzle	Motor overload	Charging	Temperature 1	
5	Heating	Emergency button	Grean light	Temperature 2	
6	Ejector	Injection shield	Red light	Temperature 3	
7	AirValve	Detection switch	Yellow light	Temperature 4	
...	....	...	...	...	...
50		Screw speed			

TABLE II
SYSTEM COMPARISON OF PC-BASED PLC IN IMM

Brand	CPU	ROM	Language	Distributed	HMI
TECHMATION	DSP	Built-in	Assembly language	No	Irreplaceable
KEBA	Intel	1G	IEC61131-3	Yes	Irreplaceable
BECKHOFF	Intel	1G	IEC61131-3	Yes	Irreplaceable
GEFRAN	Intel	1G	IEC61131-3	Yes	Irreplaceable

II. SYSTEM ARCHITECTURE

A. Hardware Structure of $ePLC$

Fig.1 shows a type of hardware structure of multi-processor $ePLC$ which contains a master processor and two slave processors. The master processor is responsible for logic control, communication, etc. The slave processor is designed for complex algorithms which could be customized on demand (the number of processors, DI\Os, AI\Os and controlled servo motors).

B. Multi-language supported graphical component

In order to develop the logic program and algorithm program in the uniform platform. We package the algorithm into

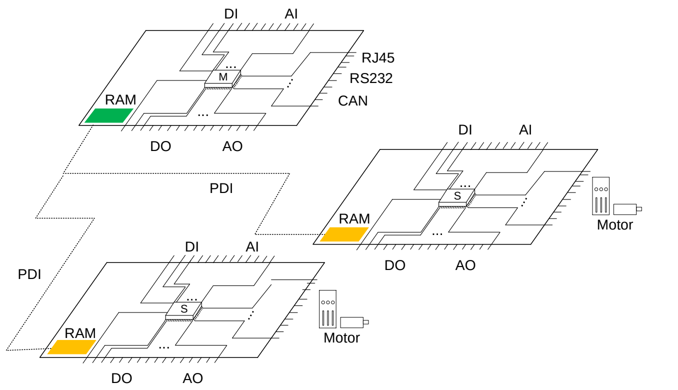
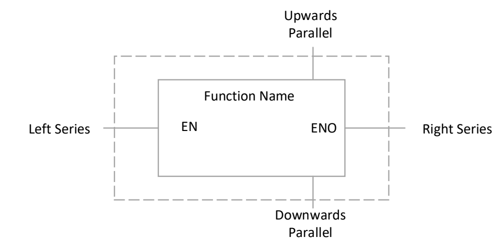


Fig. 1. A type of hardware structure of multi-processor $ePLC$ which contains a master processor and two slave processors.



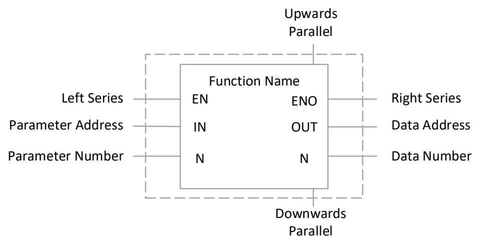


Fig. 2. Two typical design: single component and component with input and output.

graphical component which is multi-language supported. The component is defined below.

$$LDC < Name, ID, PI, RI, PT, SF > \tag{1}$$

where:

Name is the name of component used to describe the function.

ID is the unique identifier of the component in the graphical program.

PI is a collection of service interfaces, including output data interfaces, right serial connection, downwards parallel connection and some auxiliary interfaces.

RI is a collection of requirement interfaces, including input data interfaces, left serial connection, downwards parallel connection and some auxiliary interfaces.

PT is an attribute collection of the component, including position, size, comment, etc.

SF is the function description explained by specific text, formula or frame template. The graphical basic component are divided into contact components, functional block components, coil components, cross line vertical components, change lines, comments, etc. The multi-language component specially includes the development language, the supported compiler and the executing processor.

Fig. 2 illustrates two component design: single component and component with input and output. The single component has function name, left series, right series, upwards parallel and downwards parallel. In component with input and output, it contains function name, left series, right series, upwards parallel, downwards parallel, parameter address, parameter number, data address and data number.

As showed in Fig. 3, from the user's point of view, after inducing the multi-language component, the algorithm and logic program could be developed in a uniform PLC platform. Multi-language components are supported by multiple languages such as IL instructions, ST language, C language, C++ language, etc. Algorithms contained in components are mainly motion control algorithms.

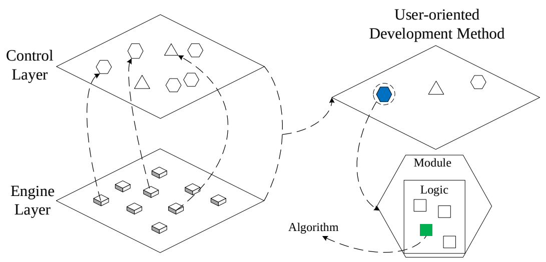


Fig. 3. From the user's point of view, after inducing the multi-language component, the algorithm and logic program could be developed in a uniform PLC platform.

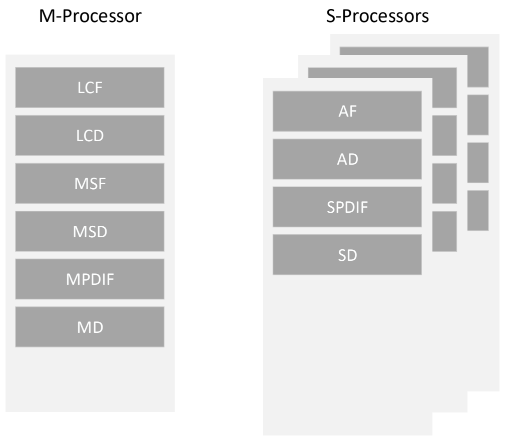


Fig. 4. Memory allocation in master and slave processors.

C. *Memory Allocation*

The dedicated storage area of PLC in memory is made up of bit data area (M area) and byte data area (D area). Meanwhile, we regard M area and D area as set M of bit and set D of byte. Furthermore, in this remaining paper, if $\exists$ set T, we describe its subscripted lowercase letter t_i as an element of T and the subscripted i is used to distinguish the elements. Henceforth, two definitions are illustrated below.

Definition 1 If $T \subseteq M$, the value saved in $t_i \in \{0, 1\}$ and each element t_i has four operators: $\mathcal{S}_0(t_i)$ denotes that t_i is set to zero, $\mathcal{S}_1(t_i)$ denotes that t_i is set to one, $\mathcal{J}_0(t_i)$ represents that the value of t_i is judged as 0, $\mathcal{J}_1(t_i)$ represents that the value of t_i is judged as 1. Then we define the set T has $\mathcal{B}$ attribute.

Definition 2 If $T \subseteq D$ and $\forall t_i \in T$ has 4 bytes. We define the set T has $\mathcal{D}$ attribute.

Fig. 4 shows the memory allocation of master and slave processors. All slave processors have the same storage structure.

LCF (Logic Control Flag Area): the flag are used to start the modules. It has $\mathcal{B}$ attribute.

LCD (Logic Control Data Area): these data will be used to delivery to algorithm. It has $\mathcal{D}$ attribute.

AF (Algorithm Flag Area): it includes algorithm flag of execution (AFE) and algorithm flag of state (AFS). Both of them have $\mathcal{B}$ attribute.

AD (Algorithm Data Area): these data help specified algorithm executing. It has $\mathcal{D}$ attribute.

MF (Message Flag Area): it includes defined message flag (DMF) and user customized message flag (UMF). DMF is the necessary message for system execution, e.g. start module flag, alarm flag, etc. UMF could be defined by users. Both of them have $\mathcal{B}$ attribute.

MD (Message Data Area): it is used to transfer message information which includes system message data area (DMD) and usr message data area (UMD). It is defined in D area.

MPDIF (Master Processor Data Interaction Flag Area): it contains begin data transfer flag from master to slave (MSB), transfer state of master from master to slave (MSF), acknowledge flag of master from master to slave (MSA) and transfer state of master from slave to master (MSS). All of them have $\mathcal{B}$ attribute.

MSD (Master Processor Data Interaction Data Area): an area stores the data delivered from slave processors and it has $\mathcal{D}$ attribute.

SPDIF (Slave Processor Data Interaction Flag Area): this area includes the begin data transfer flag from slave to master (SMB), transfer state of slave from slave to master (SMF), acknowledge flag of slave from slave to master (SMA) and transfer state of slave from master to slave (SMS). All of them have $\mathcal{B}$ attribute.

SMD (Slave Processor Data Interaction Data Area): an area stores the data delivered from master processor and it has $\mathcal{D}$ attribute.

D. User-oriented Thread Design

From the user's point of view, in most cases, the logic control program (LCP) and algorithm program (AP) could be developed independently [11], hence we have logic thread and algorithm thread. On the other hand, in order to satisfying ever-growing performance requirement of users, we proposed the customized multi-processor $ePLC$. Correspondingly an individual motion thread is designed into every slave processor. The user-oriented thread structure can be seen in Fig 5. Every processor is a four level preemptive scheduling thread structure. **Emergent Thread**, **Communication Thread**, **Diagnose Thread** and **APC Thread** see in [11]. Two special threads are explained below.

Control Thread: it is running in master processor and has functions including dealing with DI\O, executing logic program, exchanging data with slave processors, etc.

Algorithm Thread: it is running in slave processor and contains functions including interacting data with master, executing algorithm program, controlling actuators, etc.

E. Modular Design

In applications, modular design will reduce the complexity of program [39]. Therefore, we provide a system level frame for modular design. In Fig. 3, we can have a look on the modular design. The program is composed by a lot of modules and a module consists of logic program and several related algorithms. Modules will work under the reasonable memory allocation, data interaction mechanism and running multithreading. Hence, we could define a program of $ePLC$ as follows.

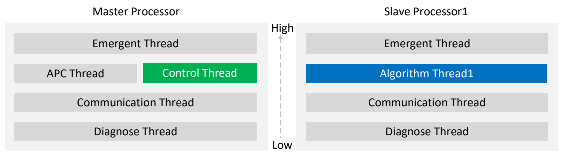


Fig. 5. User-oriented thread design in master and slave processors.

$$\begin{cases} PS = \{MS, MA, LPM, TD\} \\ ms_i \in MS = \{lcp_i, \bigcup_{j=g}^{h} ap_j\} \\ lcp_i \in LCP = \{ip_i, lcf_i, lcd_i, lpb_i\} \\ ap_j \in AP = \{afe_j, afs_j, ad_j, ab_j\} \end{cases} \tag{2}$$

The programming structure (PS) consists of modules (MS), memory allocation (MA), LPM data interaction and threads (TD). Each ms_i has two parts: logic control program lcp_i and several algorithm programs ($ap_g, ap_{g+1}, ..., ap_j, ..., ap_h$). Each lcp_i includes initial program (ip_i), logic control flag (lcf_i), logic control data (lcd_i) and logic program body (lpb_i). Each ap_j contains afe_j, afs_j, ad_j and algorithm body (ab_j).

III. System Implementation

A. Compilation of the graphic program

The compilation contains two parts: compiling graphic language into instruction list and compiling the multi-language components.

In the first part, since we see the multi-language components as an common component, the compilation of graphic language embedded multi-language components is almost the same with the process of [42] in which you can find the detailed explanation. As an example shown in Fig. 6, we adopt three steps to implement it:

Step 1: convert topology structure to directed graph according to ladder diagram syntax library and analyze the errors of the topology.

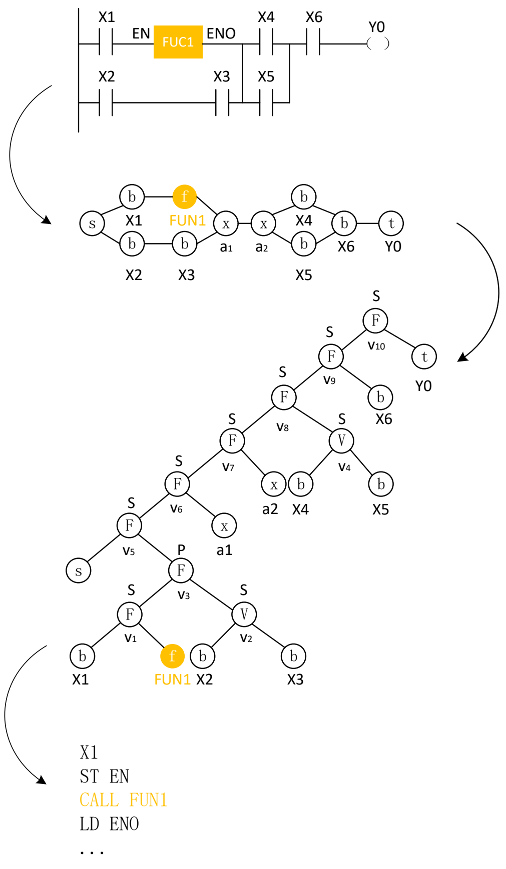
Step 2: generate a binary decomposition tree according to series and parallel rules.

Step 3: generate IL instructions according to the IL grammar library. For multi-language components, it is described as a program entry.

In the second part, the multi-language components will be compiled. For the convenience of users, they can still use the same grammar to program the $ePLC$ dedicated storage area inside the multi-language component, such as $M2000 = 1$ which represents to give 1 to the bit area $M2000$, whereas it is illegal in other languages. Hence, we should compile the component to the identifiable code which contains two steps.

Step 1: address mapping. Every type of processor has its own address mapping rules (AMR).

$$APR = \{CID, MAS, DAS, \mathcal{A}_m, \mathcal{A}_d\} \tag{3}$$



```
X1
ST EN
CALL FUN1
LD ENO
...
```

Fig. 6. Three steps of compilation of graphic program which contains multi-language component: convert topology structure to directed graph, generating a binary decomposition tree and generating IL instructions.

Where CID is the compiler identity, MAS and DAS are the start address of M and D area, respectively. $\mathcal{A}_m$ and $\mathcal{A}_d$ are the rules to map the M and D to the address of processor, respectively, see Algorithm 1 and Algorithm 2. Algorithm 1 translates the four operators of each element m_i, which are $\mathcal{S}_0(m_i)$, $\mathcal{S}_1(m_i)$, $\mathcal{J}_0(m_i)$, $\mathcal{J}_1(m_i)$, to recognizable form of CID compiler. In addition, in PLC platform, we adopt the octal system so it is necessary to translate the octal number to decimal number. Algorithm 1 and Algorithm 2 both contain this process.

Step 2: call the corresponding compiler to compile the component.

B. LPM data interaction

As shown in Fig. 7, we define the LPM data interaction with three parts: $\mathcal{L}$ (layer data interaction), $\mathcal{P}$ (processor data interaction) and $\mathcal{M}$ (module data interaction). $\mathcal{L}$ seen in [11] is the process to exchange the data between application customized layer and control layer.

$\mathcal{P}$ is used to interact data between master processor and slave processors, hence it has the process of transferring data from master to slave ($\mathcal{P}_{mts}$) and the process of transferring data from slave to master ($\mathcal{P}_{stm}$) which are defined below:

Algorithm 1: $\mathcal{A}_m$

```
Input: string oStr of $m_i$ contained its operator
Output: converted string cStr
Get the number $i$ from oStr;
Get operator $opt$ from oStr;
remainder r = $i\%10$;
Octal oI = $i/10$;
Convert oI to decimal dI;
for $opt$ do
    if $opt$==$\mathcal{S}_0$ then
        cStr = "CassMen[$MAS$+dI] $\gg$ r == 0";
    end
    if $opt$==$\mathcal{S}_1$ then
        cStr = "CassMen[$MAS$+dI] $\gg$ r == 1";
    end
    if $opt$==$\mathcal{J}_0$ then
        cStr = "CassMen[$MAS$+dI] |~(1 $\ll$ r)";
    end
    if $opt$==$\mathcal{J}_1$ then
        cStr = "CassMen[$MAS$+dI] & (1 $\ll$ r)";
    end
end
```

Algorithm 2: $\mathcal{A}_d$

```
Input: string oStr of $d_i$
Output: converted string cStr
Get the number $i$ from oStr;
Convert $i$ to decimal dI;
cStr = "CassMen[$DAS$+dI]";
```

$$\begin{cases} \mathcal{P}_{mts} &= \mathcal{U}(msb_i, msf_i, sma_i, sms_i, smd_i) \\ \mathcal{P}_{stm} &= \mathcal{U}(smb_i, smf_i, msa_i, mss_i, msd_i) \end{cases} \tag{4}$$

Where $\mathcal{U}$ is the function to implement the process of data interaction between master and slave processors. $\mathcal{P}_{mts}$ and $\mathcal{P}_{mts}$ use the same function $\mathcal{U}$.

The process flow of $\mathcal{P}_{mts}$ is denoted below: $\mathcal{S}_1(msb_i) \to \mathcal{S}_1(msf_i) \to send(smd_i) \to \mathcal{S}_0(msb_i) \to \mathcal{S}_1(sms_i) \to check(smd_i) \to \mathcal{S}_1(sma_i) \to \mathcal{S}_0(sms_i) \to \mathcal{S}_0(sma_i) \to \mathcal{S}_0(msf_i)$.

Here $send(msd_i)$ denotes sending data to msd_i in slave processor. $check(msd_i)$ denotes to check data of msd_i. $\to$ denotes the transition to the next step, e.g. $\mathcal{S}_1(msb_i) \to \mathcal{S}_1(msf_i)$ denotes msb_i is set one and then msf_i is set one.

$\mathcal{M}$ is used to interact data among modules. It includes two type messages: system defined message interaction $\mathcal{M}_d$ and user message interaction $\mathcal{M}_u$. The process is defined below:

$$\begin{cases} \mathcal{M}_d = \mathcal{V}(dmf_i, dmd_i, \mathcal{E}) \\ \mathcal{M}_u = \mathcal{V}(umf_i, umd_i\, \mathcal{E}) \end{cases} \tag{5}$$

Where $\mathcal{V}$ is the function to broadcast message and transfer data. $\mathcal{E}$ is the collection of all execution functions after getting related message, $\mathcal{M}_s$ and $\mathcal{M}_u$ have the same function $\mathcal{V}$.

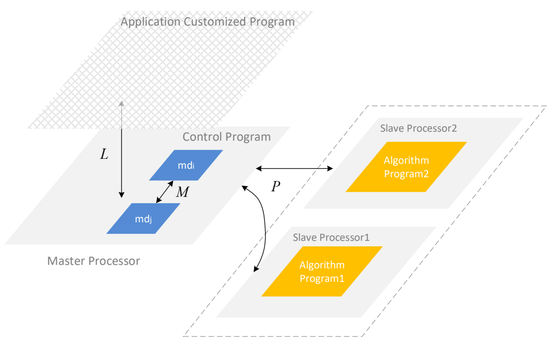


Fig. 7. LPM data interaction is defined with three parts: $\mathcal{L}$ (layer data interaction), $\mathcal{P}$ (processor data interaction) and $\mathcal{M}$ (module data interaction).

One module receives a system defined message shown below: $\mathcal{J}_1(smf_i) \rightarrow GetMessage_j(dmf_i) \rightarrow GetData(dmd_i) \rightarrow \mathcal{E}_j$.

Here $GetMessage_j(dmf_i)$ represents the ith module getting a message dmf_i and $GetData(dmd_i)$ represents the ith module getting the message information.

C. Execution of Threads

Commonly, threads in master processor and in slave processors execute separately according to their priority and the interaction between control thread and algorithm threads occurs when using $\mathcal{P}_{mts}$ and $\mathcal{P}_{mts}$. Basic execution units of control thread are shown as follows:

C_1: start a module.

C_2: transfer data to slave processor by $\mathcal{P}_{mts}$.

C_3: deal with the feedback data.

C_4: broadcast a message.

C_5: handle the message.

Motion thread contains the following basic execution units:

M_1: start the algorithm.

M_2: execute the algorithm.

M_3: feedback the data to master processor by $\mathcal{P}_{stm}$.

M_4: end algorithm.

Two cases shown in Fig. 8 are explained below:

Case one: the execution of control thread and two algorithm threads among three processors. The control thread (CT) traverses LCF, finds ms_i to be executed , runs lcp_i, finds ap_j, executes C_1 unit, executes C_2 unit and then transfers data from LCD to SMD of processor 1. Algorithm thread (AT) 1 executes M_1 unit, executes M_2 after transferring data from SMD to AD, runs M_3 unit, feedbacks data to CT. When the ap_j finishes, AT 1 executes M_4 and informs CT the end of ap_j. CT executes C_3 to end the process and then finds ap_{j+1}, executes C_1 and C_2, transfers the data from LCD to SMD of processor 2, AT 2 executes M_1 unit, executes M_2 after transferring data from SMD to AD. When the ap_{j+1} finishes, AT 2 executes M_4 unit and informs CT the end of ap_{j+1}. CT executes C_3 to finish the process.

Case two: the execution of control thread and two algorithm threads among three processors together with message mechanism. The CT traverses LCF, finds ms_i to be executed , runs lcp_i, finds ap_j, executes C_1 unit, executes C_2 unit and

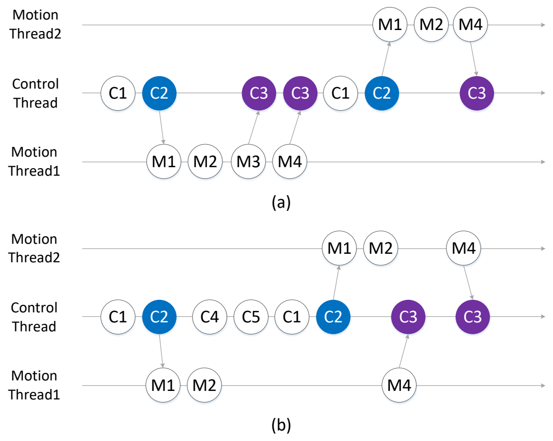


Fig. 8. Execution of control thread and two algorithm threads among three processors with and without message.

then transfers data from LCD to SMD of processor 1. AT 1 executes M_1 unit, executes M_2 after transferring data from SMD to AD. During the execution of ms_i, CT executes C_4 to broadcast the message dmf_x to inform ms_k to run. After executing of C_5, ms_k gets data and start, then CT finds ap_{j+1} in ms_k, executes C_1 and C_2, transfers the data from LCD to SMD of processor 2, AT 2 executes M_1 unit, executes M_2 after transferring data from SMD to AD. During the execution of ap_{j+1}, AT 1 executes M_4 and informs CT to execute C_3. After that, ap_{j+1} finishes, then CT executes C_3 to finish the process.

D. Finite State Machines

The finite state machines adopt the 5-tuple which is similar with [43]:

$$\mathcal{F} = (Q, X, Y, \delta, \lambda) \tag{6}$$

Where $Q = \{q_0, q_1, ..., q_i\}$ is the collection of states, $q_0 \in Q$ is the initial state. X is the finite set of inputs. Y is the finite set of outputs. δ is the state transition function $\delta : Q \times X \rightarrow Q$. λ is the output function $\lambda : Q \times X \rightarrow Y$. If F is in state q and x is occurred, then the F transits to state $q' = \delta(q, x)$ and outputs $y = \lambda(q, x)$. This transition could be denoted $\tau = (s, x/o, s')$.

Hence, we have the following finite state machines of master processor:

$$\begin{cases} \mathcal{F}_m = \{Q_m, X_m, Y_m, \delta_m, \lambda_m\} \\ Q_m = \{mstop, mrun, mbm, pdi\} \\ X_m = \{x_{m1}, x_{m2}, x_{m3}, x_{m4}, x_{m5}, x_{m6}, x_{m7}\} \\ Y_m = \{y_{m1}, y_{m2}, y_{m3}\} \end{cases} \tag{7}$$

Where, $mstop$ is the stop state and the initial state of master processor, $mrun$ is the run state, mbm is the broadcast state and pdi is the data interaction state. The inputs are defined below:

$$x_{m1} \Leftrightarrow \exists lcf_i : \mathcal{J}_1(lcf_i)$$
$$x_{m2} \Leftrightarrow \forall lcf_i : \mathcal{J}_0(lcf_i)$$
$$x_{m3} \Leftrightarrow \exists mf_i : \mathcal{J}_1(mf_i)$$
$$x_{m4} \Leftrightarrow \forall mf_i : \mathcal{J}_0(mf_i)$$
$$x_{m5} \Leftrightarrow \exists msf_i, mss_j : \mathcal{J}_1(msf_i) \vee \mathcal{J}_1(mss_j)$$
$$x_{m6} \Leftrightarrow \forall msf_i, mss_j, mf_k : \mathcal{J}_0(msf_i) \wedge \mathcal{J}_0(mss_j) \wedge \mathcal{J}_0(mf_k)$$
$$x_{m7} \Leftrightarrow \forall msf_i, mss_j, \exists mf_k : \mathcal{J}_0(msf_i) \wedge \mathcal{J}_0(mss_j) \wedge \mathcal{J}_1(mf_k)$$
$$y_{m1} \Leftrightarrow C_1$$
$$y_{m2} \Leftrightarrow C_4$$
$$y_{m3} \Leftrightarrow C_2 \quad or \quad C_3$$

Here, e.g. $x_{m1} \Leftrightarrow \exists lcf_i : \mathcal{J}_1(lcf_i)$ denotes that x_{m1} is an input of the X_m and this input equivalent to the existence of an logic control flag, lcf_i whose value is 1.

Then we can get the state transitions of master processor:

$$\begin{cases} \tau_m 1 = (mstop, x_{m1}/y_{m1}, mrun) \\ \tau_m 2 = (mrun, x_{m2}, mstop) \\ \tau_m 3 = (mrun, x_{m3}/y_{m2}, mbm) \\ \tau_m 4 = (mbm, x_{m4}, mrun) \\ \tau_m 5 = (mrun, x_{m5}/y_{m3}, pdi) \\ \tau_m 6 = (pdi, x_{m6}, mrun) \\ \tau_m 7 = (mbm, x_{m5}/y_{m3}, pdi) \\ \tau_m 8 = (pdi, x_{m7}, mbm) \end{cases} \tag{8}$$

The finite state machines of every slave processor are illustrated below:

$$\begin{cases} \mathcal{F}_s = \{Q_s, X_s, Y_s, \delta_s, \lambda_s\} \\ Q_s = \{sstop, sready, srun, pdi\} \\ X_s = \{x_{s1}, x_{s2}, x_{s3}, x_{s4}, x_{s5}\} \\ Y_s = \{y_{s1}, y_{s2}, y_{s3}\} \end{cases} \tag{9}$$

where $sstop$ is the stop state and the initial state, $srun$ is the run state, $sready$ is the ready state and pdi is the data interaction state. The events are defined below:

$$x_{s1} \Leftrightarrow \exists sms_i : \mathcal{J}_1(sms_i)$$
$$x_{s2} \Leftrightarrow \exists afe_i : \mathcal{J}_1(afe_i)$$
$$x_{s3} \Leftrightarrow \exists afs_i : \mathcal{J}_1(afs_i)$$
$$x_{s4} \Leftrightarrow \forall afs_i : \mathcal{J}_0(afs_i)$$
$$x_{s5} \Leftrightarrow \exists smb_i, sms_j : \mathcal{J}_1(smb_i) \vee \mathcal{J}_1(sms_j)$$
$$y_{s1} \Leftrightarrow M_1$$
$$y_{s2} \Leftrightarrow M_2$$
$$y_{s3} \Leftrightarrow M_3$$

Then we can get the state transitions of slave processor:

$$\begin{cases} \tau_{s1} = (sstop, x_{s1}/y_{s1}, pdi) \\ \tau_{s2} = (pdi, x_{s2}, sready) \\ \tau_{s3} = (sready, x_{s3}/y_{s2}, srun) \\ \tau_{s4} = (srun, x_{s4}, sstop) \\ \tau_{s5} = (srun, x_{s5}/y_{s3}, pdi) \end{cases} \tag{10}$$

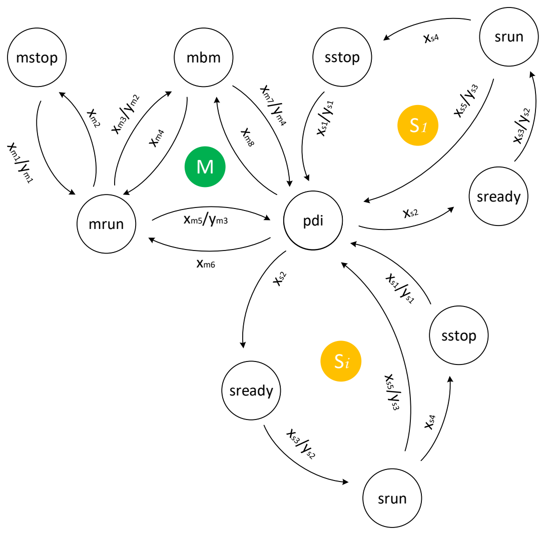


Fig. 9. Finite state machines of $ePlC$ which contains master processor M, slave processor S_1 and slave processor S_i.

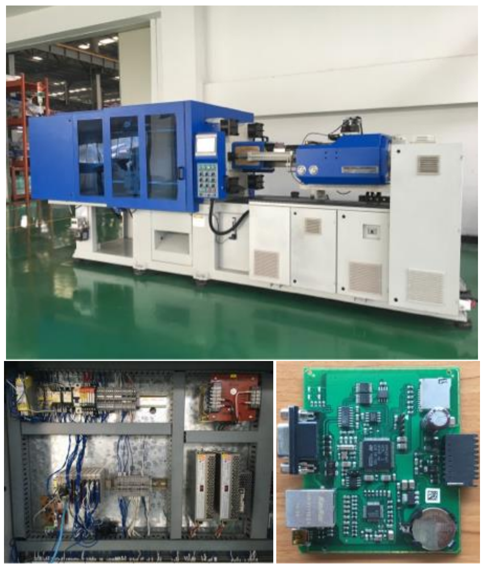


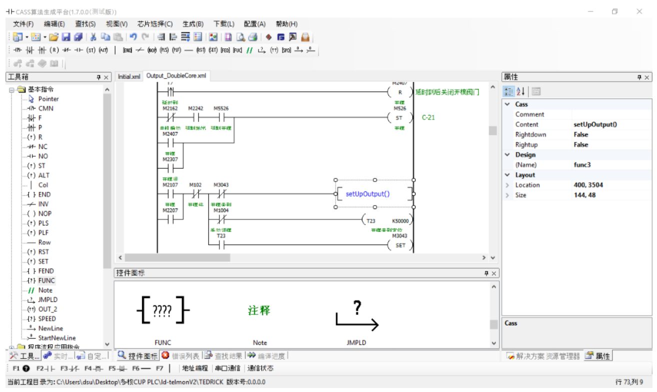
Fig. 10. 200 ton injection molding machine and its control distributed system.

Fig. 9 illustrates the whole finite state machines of $ePLC$ which contains master processor M, slave processor S_1 and slave processor S_i. This is the figure form of Eq. 7 and Eq. 9.

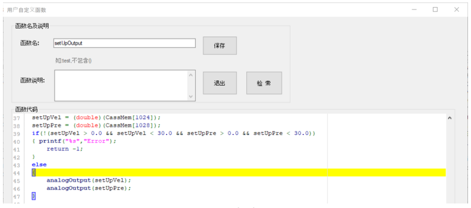
IV. Experiment

A. Distributed Control System

As show in Fig. 10, we verify the proposal development method on a 200 ton IMM. The TI F28M35 chip is chosen as the main chip of $ePLC$. It has two cores: a TI C28x and an ARM Cortex M3. Considering the DSP is more suitable for motion control, Cortex M3 is chosen as master processor and



(a)



函数名: setUpOutput

```
setUpVel = (double)(CassMem[1024]);
setUpPre = (double)(CassMem[1028]);
if(!(setUpVel > 0.0 && setUpVel < 30.0 && setUpPre > 0.0 && setUpPre < 30.0))
{ printf("%s","Error");
    return -1;
}
else
{
    analogOutput(setUpVel);
    analogOutput(setUpPre);
}
```

(b)

Fig. 11. (a) is the uniform development platform. In the dotted line box, it is the C language component and (b) is its partial code. This component represents to output a small velocity and pressure in setup mode.

C28x is chosen as slave processor. The $ePLC$ has a RJ45, a RS232 and a CAN. RJ45 is used to download program, RS232 is for connecting with HMI and CAN is designed to extend the DI\O and AI\O. The HMI could be customized by users.

B. Software Structure

Fig. 11 shows uniform development platform developed by ourselves. In the dotted line box of Fig. 11 (a), it is the C language component and Fig. 11 (b) is its partial code. This component represents to output a small velocity and pressure in setup mode.

After the design of every used components, we design the modules according to Table I. Additionally, a special module is designed to control the execution flow of all modules with the DMF and DMD.

Three typical requirements using the proposed user-oriented development method are discussed below:

1) Using multi-processors: adding S-curve acceleration and deceleration algorithm. For this case, we can run the S-curve in the DSP (slave processor).
2) Design in the uniform development platform: adding an ejector module contained a T-curve. We can design the LCP with LD and the T-curve packaged as a component in the same platform.
3) Modular Design: inject before high pressure of mold close. Customized a message flag in MSF, broadcasting the message when the beginning of high pressure and then the module of injection will get the message and the CT will start it.

TABLE III
COMPARISON OF SYSTEM PERFORMANCE

Brand	ST	DP	mean EoCP	mean EoCM	mean EoCEP	mean EoMOEP
Techmation	120s	0.27%	0.094	0.2	0.13	0.17
KEBA	150s	0.24%	0.064	0.1	0.1	0.12
Implemented	6s	0.25%	0.05	0.1	0.11	0.11

C. Analysis

We compared the system information, development method and key performance among the Techmation system, the KEBA system and the proposed system.

1) System information. TECHMATION and KEBA system account for the main market. Due to the reduced complexity, our system can have customized PLC and separated HMI. To some extend, it extremely decrease the cost. Our system adopts the widely used distributed structure which decrease wire usage and increase immunity to interference.
2) Development method. Our system adopted a usr-oriented development method, including customized multiprocessor $ePLC$, component based uniform development platform and comprehensive optimization of the system. Other systems are hard to do these and can not support C language component. Specially, Techmation system is developed by assembly instruction and even do not support IEC61131-3.
3) Key performance. To our best knowledge, we adopted defective percentage (DP), error of change-over position ($EoCP$), error of cushion minimum ($EoCM$), error of charging end position ($EoCEP$) and error of mold open end position ($EoMOEP$) as the key performance. All the systems were adjusted to use T-curve and the key parameters were set the same value. The cycle time, mold close time, mold open time, injection time, charging time and cooling time, ejector forward time and ejector backward time were controlled at about 8 s, 2 s, 2 s, 1 s, 1 s, 1 s, 0.5 s, 0.5 s respectively. Fig. 12 shows the 100 times error line graph of the key performance. The proposed system has almost identical performance with KEBA system which is better than TECHMATION system. Talbe I are the comparison of startup time (ST), DP and the mean of the key performance. Our system startup time has been increased by more than 20 times in the case of almost identical key performance.

V. Conclusion

This paper presents a user-oriented development method. We pose the customized mulitprocessor $ePLC$ to enhance the performance, the multi-language supported graphical component to improve the adaptability of developers, the optimized system structure (reasonable memory allocation, user-oriented thread structure, LPM data interaction, modular software design, finite state machines) to reduce the development complexity. Ultimately, we adopt the proposed method to

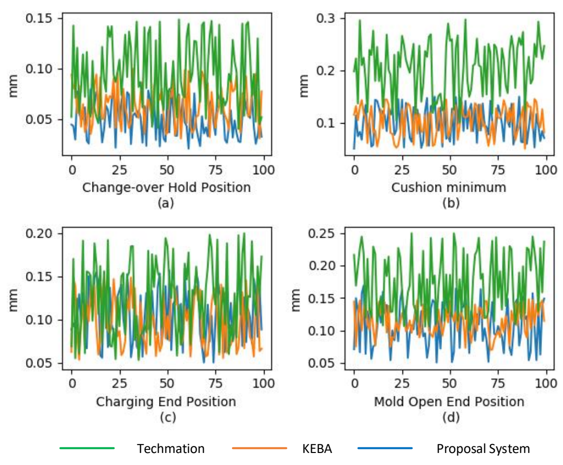


Fig. 12. (a) is EoCP, (b) is EoCP, (c) is EoCM, (d) is EoMOEP.

implement the distributed IMM system. By comparison with TECHMATION and KEBA system, our system startup time has been increased by more than 20 times in the case of almost identical key performance and our system support customized multiprocessor $ePLC$ and detached HMI.

References

[1] A. G. C. Gonzalez, M. V. S. Alves, G. S. Viana, L. K. Carvalho, and J. C. Basilio, "Supervisory control-based navigation architecture: A new framework for autonomous robots in industry 4.0 environments," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2017.

[2] D. A. Chekired, L. Khoukhi, and H. T. Mouftah, "Industrial iot data scheduling based on hierarchical fog computing: A key for enabling smart factory," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2018.

[3] S. Li, Q. Ni, Y. Sun, G. Min, and S. Al-Rubaye, "Energy-efficient resource allocation for industrial cyber-physical iot systems in 5g era," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2018.

[4] L. Ling, C. Chen, S. Zhu, and X. Guan, "5g enabled co-design of energy-efficient transmission and estimation for industrial iot systems," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2018.

[5] Y. Jiang, H. Zhang, H. Liu, X. Song, W. N. Hung, M. Gu, and J. Sun, "System reliability calculation based on the run-time analysis of ladder program," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 2013, pp. 695–698.

[6] Y. Jiang, H. Zhang, X. Song, X. Jiao, W. N. N. Hung, M. Gu, and J. Sun, "Bayesian-network-based reliability analysis of plc systems," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 11, pp. 5325–5336, 2013.

[7] B. F. Adiego, D. Darvas, E. B. Viñuela, J. C. Tournier, S. Bliudze, J. O. Blech, and V. M. G. Suárez, "Applying model checking to industrial-sized plc programs," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[8] S. Gerkšič, G. Dolanc, D. Vrančić, J. Kocijan, S. Strmčnik, S. Blažič, I. Škrjanc, Z. Marinšek, M. Božiček, and A. Stathaki, "Advanced control algorithms embedded in a programmable logic controller," *Control Engineering Practice*, vol. 14, no. 8, pp. 935–948, 2006.

[9] C. Y. Chang, "Adaptive fuzzy controller of the overhead cranes with nonlinear disturbance," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 2, pp. 164–172, 2007.

[10] S. Dominic, Y. Lohr, A. Schwung, and S. X. Ding, "Plc-based real-time realization of flatness-based feedforward control for industrial compression systems," *IEEE Transactions on Industrial Electronics*, vol. PP, no. 99, pp. 1–1, 2016.

[11] H. Wu, Y. Yan, D. Sun, and S. Rene, "A customized real-time compilation for motion control in embedded plcs," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1.

[12] D. Schütz, A. Wannagat, C. Legat, and B. Vogel-Heuser, "Development of plc-based software for increasing the dependability of production automation systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2397–2406, 2013.

[13] J. D. L. Morenas, P. G. Ansola, and A. García, "Shop floor control: A physical agents approach for plc-controlled systems," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2017.

[14] M. F. Zaeh, C. Poernbacher, and J. Milberg, "A model-based method to develop plc software for machine tools," *CIRP Annals - Manufacturing Technology*, vol. 54, no. 1, pp. 371–374, 2005.

[15] S. Hossain, M. A. Hussain, and R. B. Omar, "Advanced control software framework for process control applications," *International Journal of Computational Intelligence Systems*, vol. 7, no. 1, pp. 37–49, 2014.

[16] R. Arshad, S. Zahoor, M. A. Shah, A. Wahid, and H. Yu, "Green iot: An investigation on energy saving practices for 2020 and beyond," *IEEE Access*, vol. 5, no. 99, pp. 15 667–15 681, 2017.

[17] M. G. Ioannides, "Design and implementation of plc-based monitoring control system for induction motor," *IEEE Transactions on Energy Conversion*, vol. 19, no. 3, pp. 469–476, 2004.

[18] X. M. Shi, W. J. Fei, and S. P. Deng, "The research of circular interpolation motion control based on rectangular coordinate robot," *Key Engineering Materials*, vol. 693, pp. 1792–1798, 2016.

[19] N. Fang, "Design and research of multi axis motion control system based on plc," *Academic Journal of Manufacturing Engineering*, vol. 15, no. 1, pp. 17–23, 2017.

[20] W. F. Peng, G. H. Li, P. Wu, and G. Y. Tan, "Linear motor velocity and acceleration motion control study based on pid+velocity and acceleration feedforward parameters adjustment," *Materials Science Forum*, vol. 697-698, pp. 239–243, 2011.

[21] A. Syaichu-Rohman and R. Sirius, "Model predictive control implementation on a programmable logic controller for dc motor speed control," in *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. IEEE, 2011, pp. 1–4.

[22] J. Qian, H. B. Zhu, S. W. Wang, and Y. S. Zeng, "A 5-dof combined robot platform for automatic 3d measurement," *Key Engineering Materials*, vol. 579-580, pp. 641–644, 2014.

[23] P. Co.Ltd, *Programmable controller FP2 Positioning Unit Mannual*, 2011.

[24] M. Dubois and C. Scheurich, "Memory access dependencies in shared-memory multiprocessors," *IEEE Transactions on Software Engineering*, vol. 16, no. 6, pp. 660–673, 2002.

[25] J. H. Patel, "Processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, vol. C-30, no. 10, pp. 771–780, 2006.

[26] D. Zhu, L. Chen, S. Yue, T. Pinkston, and M. Pedram, "Providing balanced mapping for multiple applications in many-core chip multiprocessors," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 3122–3135, 2016.

[27] L. M. Albarakat, P. V. Gratz, and D. A. Jiménez, "Mtb-fetch: Multi-threading aware hardware prefetching for chip multiprocessors," *IEEE Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2017.

[28] Z. Hajduk, B. Trybus, and J. Sadolewski, "Architecture of fpga embedded multiprocessor programmable controller," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 5, pp. 2952–2961, 2015.

[29] M. Chmiel, J. Kulisz, R. Czerwinski, A. Krzyzyk, M. Rosol, and P. Smolarek, "An iec 61131-3-based plc implemented by means of an fpga," in *IEEE/IFAC PDES Conference*, 2016, pp. 28–37.

[30] O. Co.Ltd, "Cs1w-mc221(-v1)/mc421(-v1) motion control units." *Operation Mannual*, 2004.

[31] I. 61131-3, *Programmable controllers - part 3: Programming languages.*, 1993.

[32] P. T. C. 2., *Function blocks for motion control version 1.1*, 2005.

[33] C. Sünder, A. Zoitl, F. Mehofer, and B. Favre-Bulle, "Advanced use of plcopen motion control library for autonomous servo drives in iec 61499 based automation and control systems," *E & I Elektrotechnik Und Informationstechnik*, vol. 123, no. 5, pp. 191–196, 2006.

[34] S. S. S. GmbH, "Logic and motion control integrated in one iec 61131-3 system:development kit for convenient engineering of motion, cnc and robot applications." 2017.

[35] M. Bonfe and C. Fantuzzi, "Object-oriented approach to plc software design for a manufacture machinery using iec 61131-3 norm languages," in *Ieee/asme International Conference on Advanced Intelligent Mechatronics, 2001. Proceedings*, 2001, pp. 787–792 vol.2.

[36] B. Werner, "Object-oriented extensions for iec 61131-3," *Industrial Electronics Magazine IEEE*, vol. 3, no. 4, pp. 36–39, 2009.

[37] F. Basile, P. Chiacchio, and D. Gerbasio, "On the implementation of industrial automation systems based on plc," *IEEE Transactions on Automation Science & Engineering*, vol. 10, no. 4, pp. 990–1003, 2013.

[38] M. Bonfè, C. Fantuzzi, and C. Secchi, "Design patterns for model-based automation software design and implementation," *Control Engineering Practice*, vol. 21, no. 11, pp. 1608–1619, 2013.

[39] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[40] O. Verscheure, X. Garcia, G. Karlsson, and J. P. Hubaux, "User-oriented qos in packet video delivery," *IEEE Network*, vol. 12, no. 6, pp. 12–21, 2016.

[41] N. Choi, D. Kim, S. J. Lee, and Y. Yi, "A fog operating system for user-oriented iot services: Challenges and research directions," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 44–51, 2017.

[42] Y. Yan and H. Zhang, "Compiling ladder diagram into instruction list to comply with iec 61131-3," *Computers in Industry*, vol. 61, no. 5, pp. 448–462, 2010.

[43] R. M. Hierons and U. C. Turker, "Parallel algorithms for testing finite state machines: Generating uio sequences." *IEEE Transactions on Software Engineering*, vol. 42, no. 11, pp. 1077–1091, 2016.