Model-Based Design of Time-Triggered Real-time Embedded Systems for Industrial Automation

Jiang Wan[1], Arquimedes Canedo[2], and Mohammad Abdullah Al Faruque[1]

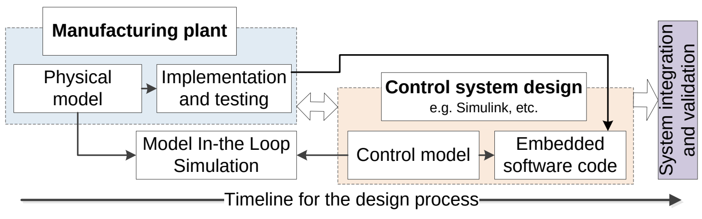
[1]Department of Electrical Engineering & Computer Science, University of California, Irvine, USA
[2]Siemens Corporation, Princeton, USA

Abstract*— This paper presents a novel Model-Based Design (MBD) approach and associated tool-chain for the Time-Triggered Real-time Embedded (TTRE)[1] systems. Our tool-chain automatically synthesizes software for manufacturing TTRE systems. A Software-In-the-Loop Simulation (SILS) framework integrated into our tool-chain helps to reduce the design iterations. Using a manufacturing robot-arm use-case, we validate our tool-chain and demonstrate a 39× improvement in the Quality-of-Control (QoC) when compared to the state-of-the-art approach [4]. Our auto-generated scheduler meets all the hard real-time constraints (zero deadline misses) for a given TTRE system when compared to the scheduler (e.g., 145 deadline misses for a CPU utilization of 95%) presented in [1], [26]. Moreover, compared to the traditional MBD approach for the TTRE systems where many design iterations are required [5], our approach and tool-chain can generate high quality and accurate tasks with an associated scheduler in a single design iteration.

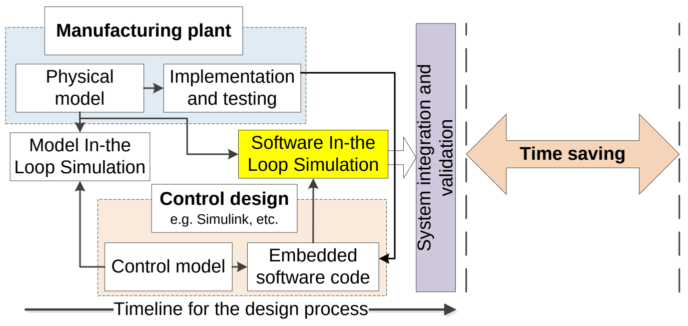
I. Introduction and Related Work

TTRE systems are real-time and typically form networked embedded systems that interact with the physical world using the time-triggered architecture [16], [20]. Today, manufacturing TTRE system is experiencing a major paradigm shift thanks to the innovations in the semiconductor and software industries that make the manufacturing faster, more energy efficient, and reliable [7], [8], [12]. Unfortunately, one of the major problems with a TTRE system is that once it is deployed, it becomes challenging to extend it. Once in the field, designers may not have the freedom to influence the timing behavior without reducing the *Quality-of-Control* (QoC)[2]. For example, adding extra software to a component in a TTRE system may increase its execution time, therefore requiring a re-evaluation of the whole scheduling policy. Extending a TTRE system will, in most cases, involve a complete re-design of the control system. To solve this problem, MBD [21], [22], [24] is considered to be a very promising solution for the design and verification of the TTRE systems because any modification to the system may be rapidly addressed and verified in a higher level of abstraction. Moreover, hardware/software may be quickly and automatically generated from the model using *Electronic Design Automation* (EDA) tools.

Traditional MBD for TTRE systems involves a sequential and independent development of the physical plant, and then the control system [5], [22] (see Figure 1 (a)). This is because a control algorithm is typically highly tuned for a specific dynamic behavior of the physical plant. *Model-In-the-Loop Simulation* (MILS) is performed to verify the control algorithm in the model level. Therefore, an accurate physical plant model is constructed prior the development of the control algorithms. The major problem of the sequential design process is that it requires a longer design time–including both the design time of the physical plant and the control system. Moreover, another important drawback is that the whole system can only be tested or verified *after* the integration of both the real physical plant and the embedded control code, any unexpected change to the physical plant or the control system requires re-execution of all the simulation/testing-processes of the entire system. To solve this problem, this paper proposes **a novel SILS-integrated MBD approach that allows the concurrent design and verification of the physical plant and the control process** (see Figure 1 (b)) for digital manufacturing applications. The proposed SILS process validates the embedded control code without the need of a completed physical plant; as a result, it reduces the number of design iterations and saves the design time.


Manufacturing plant
Physical model
Implementation and testing
Control system design
e.g. Simulink, etc.
Model In-the Loop Simulation
Control model
Embedded software code
System integration and validation
Timeline for the design process

(a) *Traditional MBD methodology for industrial TTRE systems*


Manufacturing plant
Physical model
Implementation and testing
Model In-the Loop Simulation
Software In-the Loop Simulation
Control design
e.g. Simulink, etc.
Control model
Embedded software code
System integration and validation
Time saving
Timeline for the design process

(b) *Proposed SILS-integrated MBD methodology for industrial TTRE systems*

Figure 1: Comparison of the traditional and proposed design approach for the manufacturing TTRE systems

The manufacturing TTRE systems are typically implemented in *Programmable Logical Controllers* (PLCs) and programmed with IEC 61131-3 languages [13], [19] (see Section III). In [18], [25], MBD methods for PLC application development are discussed. Similarly, Simulink [3] is a commercial MBD tool capable of modeling and simulating

[1]IEC 61131-3 specific *Programmable Logical Controllers* (PLC) are used as the de-facto standard to build manufacturing TTRE systems [8].

[2]QoC refers to the quantitative value expressing control performance in control theory (e.g. the rise-time, the peak-overshoot and the settling-time etc.)

TTRE systems according to a synchronous reactive *Model of Computation* (MoC). The *Simulink PLC Coder* [4] can be used to automatically generate IEC 61131-3 code for Simulink subsystems. However, all the above mentioned academic and commercial MBD methodologies share the following limitations: 1) the state-of-the-art *Simulink PLC Coder* [4] tool can only generate non-executable *Function Blocks* (FBs)– An FB is an IEC 61131-3 specific semantic (for details see Section III) for PLCs; 2) minimum support for the design-space exploration of concepts and control strategies [10]; 3) the lack of a tool-chain to perform rapid SILS or *Hardware-In-the-Loop Simulation* (HILS) (see details of SILS/HILS in Section V) for the developed TTRE systems on a target Soft-PLC[3]. Although co-simulation solutions have been proposed in [9], [14], [23], these tool-chains do not support simulation in real-time.

To overcome these limitations, this paper proposes a complete tool-chain integrating off-the-shelf domain-specific tools and our proposed interfaces and algorithms. Using our tool-chain, designers may automatically generate, from a high-level model, executable IEC 61131-3 programs with timing (*Worst Case Execution Time* (WCET) [27]) and schedulability analysis for a target Soft-PLC platform.

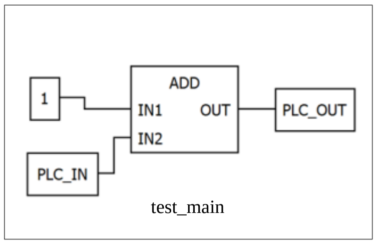
The rest of the paper is organized as follows. Section II presents our novel contributions. Section III provides the key concepts and definitions for PLCs. Section IV describes our novel MBD approach for TTRE systems targeting Soft-PLCs. Section V describes our proposed SILS framework. Section VI provides our experimental evaluation. Section VII provides our concluding remarks.

II. Our Contributions

(1) A novel MBD approach for the manufacturing TTRE systems and a complete tool-chain consisting of off-the-shelf domain-specific tools and newly developed interfaces and algorithms for automatically generating executable Soft-PLC programs.
- An algorithm for estimating and back annotating the timing information of the control system in order to synthesize time-accurate tasks and to reduce the design iterations.
- A tool for the automatic synthesis of Simulink-generated FBs in IEC 61131-3 into the Soft-PLC-based TTRE systems composed of tasks and a real-time scheduler.
(2) A novel SILS framework where the physical plant is modeled in Simulink, the control program is executed in a Soft-PLC [1], [26], and the two communicate using an industrial communication protocol [6]. In this framework, various configuration knobs can be tested for the design space exploration of alternatives that meet the timing constraints and improve the QoC.

III. IEC-61131-3 based Soft-PLC

PLCs are cyclic TTRE systems heavily used in the digital manufacturing applications that satisfy various requirements such as hard real-time, operations in harsh environments and long life cycles [12]. Soft-PLCs, or PC-based controllers, are soft real-time controllers that run on commercial off-the-shelf general purpose computers. Soft-PLCs provide a more flexible alternative to the traditional *hardware* PLCs because they can communicate with other applications running in a general purpose PC. PLCs/Soft-PLCs are typically programmed using the IEC 61131-3 [8] language standard. There are 2 graphical languages (LD and FBD) 2 textual languages (ST and IL) and 1 additional *Sequential function chart* (SFC) supported in IEC 61131-3 language.



(a) *FBD program example*

```
PROGRAM test_main
 VAR
   PLC_OUT : INT;
   PLC_IN : INT;
 END_VAR

 PLC_OUT := ADD(1, PLC_IN);
END_PROGRAM
```

(b) *ST program example*

```
CONFIGURATION conf

  RESOURCE res ON PLC
    TASK task0(INTERVAL := T#50ms,PRIORITY := 0);
    PROGRAM prog0 WITH task0 : test_main;
  END_RESOURCE

END_CONFIGURATION
```

(c) *Configuration example*

Figure 2: A simple example of the IEC 61131-3 language-based program

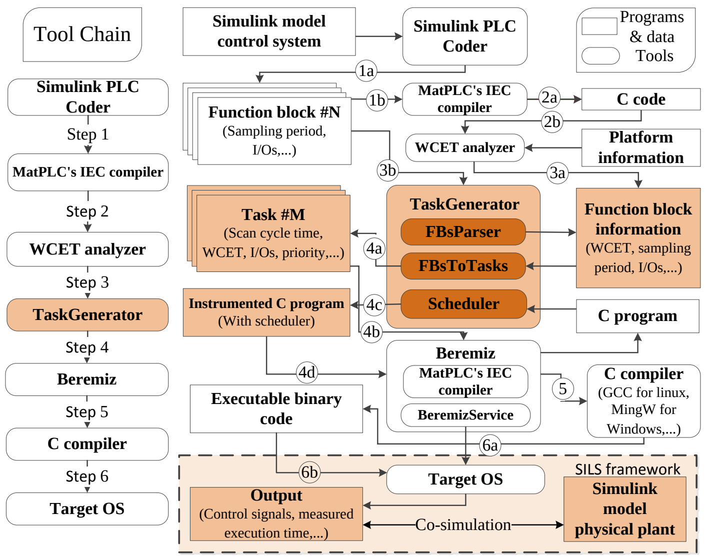


Figure 3: Our proposed MBD approach and the complete tool-chain (the highlighted parts are our contributions and rest of the parts are off-the-shelf tools/processes/functionality)

Figure 2(a) and (b) show an example of IEC 61131-3-based program in FBD and ST languages. An example of configuration is shown in Figure 2(c). Details about PLC, Soft-PLC, and the IEC 61131-3 language may be found in [7], [8], [12]. In the scope of this paper, we target Soft-PLC systems because they are becoming a popular choice in the next-generation manufacturing TTRE systems (also known as digital manufacturing systems) for applications like building

[3]Soft-PLCs or PC-based controllers are control programs running on personal computers.

energy management, home automation, etc.

IV. MBD of TTRE for Soft-PLC Platforms

Figure 3 shows our proposed 6 steps MBD approach and associated tools.

Step 1: The first step of the proposed design approach is to automatically generate non-executable FBs written in the IEC 61131-3 ST language using the *Simulink PLC Coder* [4].

Step 2: Using an IEC 61131-3 compiler [26] to translate the FBs into ANSI C-based *Function Codes* (FCs).

Step 3: Calculating the FCs WCET using C language as the intermediate representation.

Step 4: Our *TaskGenerator* generates the executable tasks with the timing information (see Section IV-A2) and synthesizes a real-time preemptive scheduler (see Section IV-B) for the run-time system. Moreover, the timing and scheduling information is fed back to the system designers. This will help them to achieve reduced iterations and shorter design-time.

Step 5: When tasks are automatically generated, this step implements the control program in a Soft-PLC.

Step 6: Soft-PLC programs are compiled for a target *Operating System* (OS) and made executable together with the provided Soft-PLC services.

A. Executable Task Generation

Our *FBsToTasks* (see Figure 3) tool generates a set of executable tasks from a set of non-executable FBs. The problem formulation for the tool development is given as follows: A resource R is defined as: $R(PLM, CM, OS, CHC)$, where $R.PLM$ is the hardware platform, $R.CM$ is the compiler, $R.OS$ is the OS of the Soft-PLC, and $R.CHC$ is the communication channel capacity. A function block FB is defined as: $FB(e, s, p, i)$, where $FB.e$ is the WCET on a particular resource R, $FB.s$ is the sampling period of the current FB defined in the control algorithm[4], $FB.p$ is the priority of the FB, and $FB.i$ is the total bits of I/Os of the FB. Moreover, a set of FBs is defined as $\Phi = \{FB_1, FB_2, ..., FB_n\}$. A task τ is defined as: $\tau(e, s, d, p, i)$, where $\tau.e$ is the WCET of the task, $\tau.s$ is the *Scan Cycle Time* (SCT) [8] defined in the PLC program, $\tau.d$ is the deadline, $\tau.p$ is the priority, and $\tau.i$ is the total bits of I/Os of the task. In PLCs, the deadline of a task is the time of beginning the next scan cycle, so we can simplify the task as $\tau(e, s, p, i)$ by removing the deadline. Moreover, a set of tasks T may be defined as $\{\tau_1, \tau_2, ..., \tau_m\}$.

Therefore, *FBsToTasks* tool is to find a mapping $\Phi \rightarrow T$ defined by a function $f(\Phi, R)$, where $m \leq n$. In this paper, we are only considering a Soft-PLC having a single resource R. Therefore, we divide Φ onto subsets $\varphi_1, \varphi_2, ..., \varphi_m$, such that inside each subset, all $FBs \in \varphi_j(s, n, i)$ have the same sampling rate. In a subset φ_j, $\varphi_j.s$ is the sampling rate of any $FB \in \varphi_j$. Now, we create a set T of m tasks $\tau_1, \tau_2, ..., \tau_m$, map each task τ_j to φ_j by assigning $\tau_j.s$ to $\varphi_j.s$ and instantiate all $FB^j s$ inside τ_j.

[4]In our case, this is the sampling rate configured and simulated in Simulink.

1) Intermediate Representation of IEC FBs: In order to generate executable tasks, the capability of WCET analysis of the IEC 61131-3 based FBs needs to be achieved. For conventional PLCs, PLC vendors use proprietary hardware platforms ($R.PLM$) and compilers ($R.CM$), therefore, the existing WCET analyzers which need R as an input may not be used. However, for a Soft-PLC, R can be easily obtained since the hardware is a general purpose computer. In our work, we have used the state-of-the-art WCET analyzer tool *AbsInt* [17] which takes R and a sequentially executed code segment as inputs, and generates the WCET information an output. However, existing WCET analysis methods are limited to inputs such as ANSI C or executable binary code. Therefore, we translate the set of tasks T in the IEC 61131-3 language onto ANSI C code using an IEC compiler [26].

2) Timing information Analysis of IEC FBs: After generating the WCET of all the FBs in Φ, the total execution time for a task τ_i may be calculated as:

$$\forall FB^i \in \varphi_i : \tau_i.e = \sum_{1}^{\varphi_i.n} FB^i.e \quad (1)$$

Besides the WCET, for any task τ_i, with $\tau_i.i$ running on $R.CHC$, we estimate the communication delay as $(\tau_i.i)/(R.CHC)$. Finally, for any task τ_i, the processor utilization U_i may be calculated as:

$$U_i = ((\tau_i.e + (\tau_i.i)/(R.CHC)))/(\tau_i.s) \quad (2)$$

Based on the derived U_i of all the tasks, our *FBsToTasks* tool reports the schedulability of these tasks on the targeted resources. Therefore, the schedulability constraint will be:

$$\sum_{i=1}^{m} U_i \leq 100\% \quad (3)$$

Here, m is the total number of generated tasks.

In summary, the *FBsToTasks* tool creates a report of the WCETs, the communication delays, and the result of the schedulability analysis of all the generated tasks and provides them back to the model designers. The designers could apply configuration knobs (such as: (1) reduce the I/O delay; (2) reduce the WCET of each task; and (3) increase the SCT) to reduce the processor utilization and increase the QoC of the system. With the help of our tool-chain, these configuration knobs may be applied concurrently for reducing the number of design iterations.

B. Automatic Synthesis of Real-time Preemptive Scheduler for the Executable Tasks

A real-time scheduler is a fundamental run-time mechanism used to guarantee hard real-time constraints for the generated executable tasks. However, an IEC 61131-3 language specific PLC-based TTRE system does not have the required language semantics to support a scheduler for the generated tasks [8]. Typically, PLC vendors manage the scheduler manually, it is highly implementation-specific, and requires manual modification whenever there is a change in the system specification. Moreover, *Simulink PLC Coder* [4] does not support the generation of any real-time scheduler for its non-executable FBs. To solve this problem, a real-time scheduler (see Algorithm 1) for the target Soft-PLC-based TTRE system is automatically generated and the necessary code instrumentation (see Algorithm 2) is performed to support the scheduler.

Algorithm 1 Real-time preemptive scheduler for the executable tasks

```
Input:      Original C code with a set of tasks
            $T\{\tau_1, \tau_2, ..., \tau_n\}$
Output:     Instrumented C code with a set of
            scheduled threads $THR\{thr_1, thr_2, ..., thr_n\}$
            mapped from $T$
Variables:  $scp$: schedule cycle period
            $scn$: schedule cycle number
            $THR_{rdy}$: a list of threads ready to be executed
1:  Set highest OS priority for current thread of scheduler;
2:  $THR = thr_1, thr_2, ..., thr_n; THR_{rdy} = \emptyset$;
3:  for all $thr \in THR$ do
4:      $thr.P_{OS} = NORMAL$; $thr_i.\tau_i = \tau_i$;
5:  end for
    // Initialize all threads
6:  $scp = GCD(\tau_1.s, \tau_2, s, ...\tau_n.s)$;
7:  while TRUE do
8:      for all $thr_j \in ((scn \times scp)\%thr_j.\tau.s == 0)$ do
9:          $THR_{rdy}.add(thr_j)$
10:         if $\tau_j.s$ is the maximum of all $s$ in $T$ then
11:             $scn = 0$;
12:         end if
13:     end for
        // Check and add new thread to be executed,
        reset the scn if necessary
14:     for all $thr_k \in THR_{rdy}$ do
15:         $thr_k.P_{OS} = NORMAL$;
            $thr_k.d = thr_k.\tau.s - (scn \times scp)\%thr_k.\tau.s$;
16:     end for
        // Update all threads in the list
17:     $\exists! thr_l \in THR_{rdy} : thr_l.d$ is the smallest in $THR_{rdy}$;
18:     $thr_l.P_{OS} = HIGH$;
        // Prompt the thread/task with earliest
        deadline in list to HIGH priority
19:     $scn++$;
20:     Wait($scp$);
        // Release the resource to run a cycle period
21: end while
```

Algorithm 2 Instrumentation of the Soft-PLC tasks

```
Input:   A set of tasks.
Output:  A set of instrumented tasks.
1: Execution of the current task;
   // Finish the running of task instructions
2: $THR_{rdy}$.remove(current thread);
   // Remove the current thread from the $THR_{rdy}$
3: $\exists! thr_l \in THR_{rdy} : thr_l.d$ is the smallest in $THR_{rdy}$;
4: $thr_l.P_{OS} = HIGH$;
   // Pickup a new thread to execute
```

In our current work, we automatically synthesize an *Earliest Deadline First Scheduler* (EDFS), which is a dynamic priority scheduling algorithm for single core real-time systems. We consider the following assumptions to generate our scheduler for the executable tasks : (1) $\forall \tau_i, \tau_j \in T, \tau_i, \tau_j$ are independent of each other; (2) $\forall \tau_i \in T$, the deadline is deterministic and equal to the sampling rate; (3) $\forall \tau_i \in T$, priority p is assigned dynamically based on the earliest-deadline-first conventions; (4) $\forall \tau_i, \tau_j \in T$, if $\tau_i.p > \tau_j.p$, then τ_i will be able to preempt τ_j immediately with a negligible preemption overhead.

Soft-PLCs are capapble of utilizing the standard thread services (which support preemption capability) provided by the host OS (e.g. Linux, Windows) to implement the run-time system. Therefore, during the synthesis of the scheduling algorithm, we have implemented the EDFS by assigning each task to a thread with the priority decided by its deadline. Algorithm 1 presents the pseudo-code of the automatically synthesized EDFS scheduler. We define a thread as $thr(P_{OS}, \tau, d)$, where P_{OS} is the current priority of the thread in the host OS, τ is the task mapped to this thread, and d is the deadline of the current thread. We define the value of P_{OS}, to be either $NORMAL$ or $HIGH$. scp is the *Greatest Common Divider* (GCD) of all the sampling period of tasks. Either at the end of execution of a task (see Algorithm 2) or in every time stamp of scp, the scheduler will preempt the current task, renew the list of tasks ready to be executed, and choose the task with the earliest deadline by setting a $HIGH$ priority to it. The complexity of our synthesized algorithm is $O(n)$, where n is the number of tasks.

V. Validation Framework

A. Our Framework

Traditional MILS provides functional correctness but lacks accurate timing information of the control system and the physical plant. Therefore, industry has adopted HILS techniques where real controllers and communication hardware are integrated in the control loop for accurate timing constraint validation. However, HILS requires a longer design cycle and is typically used during the final phase of the validation. Therefore, an emulation-based SILS approach where the software-based control can be executed in a real control environment by keeping the plant and the communication as software models has been widely adopted [15]. In this paper, we present a SILS framework for our proposed MBD approach of the Soft-PLC-based TTRE systems (see Figure 4).

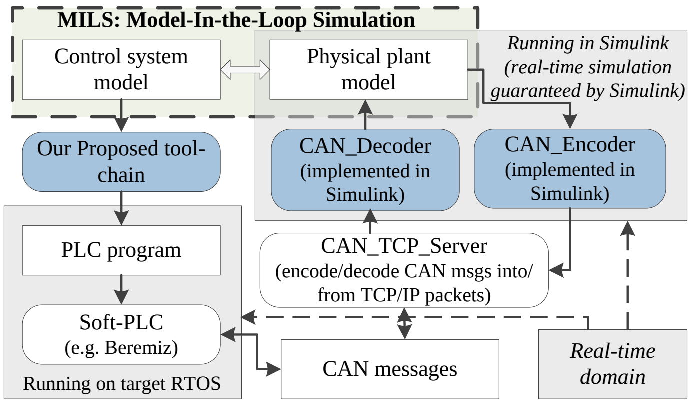

MILS: Model-In-the-Loop Simulation
Control system model
Physical plant model
Running in Simulink (real-time simulation guaranteed by Simulink)
Our Proposed tool-chain
CAN_Decoder (implemented in Simulink)
CAN_Encoder (implemented in Simulink)
PLC program
CAN_TCP_Server (encode/decode CAN msgs into/ from TCP/IP packets)
Soft-PLC (e.g. Beremiz)
CAN messages
Real-time domain
Running on target RTOS

Figure 4: Our proposed SILS framework

In our SILS framework, the physical plant is modeled using the physical modeling language (e.g. Simscape [2]). Embedded control software (see Section IV) is executed on a Soft-PLC platform. As indicated by the dotted line in Figure 4, both the physical plant model and the embedded software are executed in the real time domain connected through the TCP/IP channel. The TCP/IP packets mimic the messages according to the CANOpen [6], or PROFINET [11] protocols. The control program in the Soft-PLC uses the real-time service from the target *Real-Time Operating System* (RTOS) to guarantee the real-time execution and the plant model (Simulink model in current work) uses the real-time synchronization tool-box.

B. Communication and Synchronization

In our SILS framework, one CAN message (containing 64 bits of data) is encoded into one TCP/IP packet. From the plant side, TCP/IP packets may be received every fixed period of time defined in the TCP/IP blocks in the Simulink. Therefore, the communication throughput of our SILS framework may be calculated as:

$$CommThroughput = N_{message} \times 64bits/st \tag{4}$$

Here, $N_{message}$ is the number of messages transferred per sample time st of the TCP/IP channel.

VI. Results and Evaluation

In our experiments, we study a real-world factory robot-arm application to validate our approach. In this case study, four "robot-arm joints" are controlled by the Soft-PLC. The control tasks are shown in Figure 5. For our experimental purposes, we implemented the "Soft-PLC" and "Robot-Arm Model" in two separated PCs communicated through ethernet to mimic the real scenario in manufacturing factory.

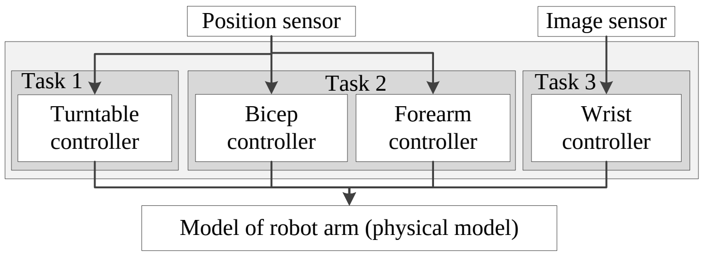


Figure 5: The factory robot-arm use-case

We have compared our experimental results with state-of-the-art methods and tools using two *Key Performance Indicators* (KPIs). Table I shows various control/platform-related parameters for the three tasks shown in Figure 5.

To compare the improvements of QoC using our tool, we use the *Sum of Absolute Differences* (SAD) between MILS and SILS results as the KPI. Figure 7 shows that compared to the Simulink PLC Coder, our tool decreases SAD exponentially to 2.58%. This indicates a 39x improvement.

Table I: Timing information of robot-arm control system

	Task 1	Task 2	Task 3	Processor utilization
WCET on initial processor	80ms	40ms	8ms	200.00%
I/O delay on initial channel	40ms	40ms	40ms	
Initial scan cycle time	200ms	100ms	80ms	
I/O delay on faster channel	32ms	32ms	32ms	178.00%
WCET on faster processor	20ms	10ms	2ms	110.50%
Redesigned scan cycle time	200ms	120ms	100ms	95.00%

Figure 8 (a) shows the MILS/SILS simulation results for the wrist controller without the proposed tool-chain. This graph represents the approach supported by Simulink for a target PLC. A significant angular difference (degree) is observed between the MILS and the SILS results. Initially, the wrist controller has been designed with a sampling rate (SCT) of 80ms. Using our tool-chain, the timing information of *Task 3* is estimated to have a WCET of 8ms and I/O delay of 40ms. With these timings, the processor utilization (Equation 2) is calculated to be 200% as shown in Table I. In order to achieve processor utilization lower than 100% (Equation 3) we explored alternative designs. Firstly, we decreased the block sample time of the physical plant (see Section IV-A) to reduce the I/O delay to 32ms. However, the processor utilization remains at 178% and therefore we have further increased the processor frequency to achieve a processor utilization of 110.55% (see Figure 8 (b)). Finally, we have slightly increase the scan cycle time of *Task 2* and *Task 3* (Figure 8 (b)). The result is shown in three steps; however our tool may use all three configuration knobs in a single design iteration. The processor utilization is reduced to 95.00% indicating a schedulable control program. Furthermore, Figure 8 (c) compares the result of the wrist controller using the Beremiz scheduler [1], [26] and our implemented EDFS scheduler. Figure 6 compares the number of deadline misses (KPI for scheduler in real-time system) using these two schedulers; our scheduler achieves the real-time requirement of zero deadlines misses compared to the Beremiz scheduler [26].

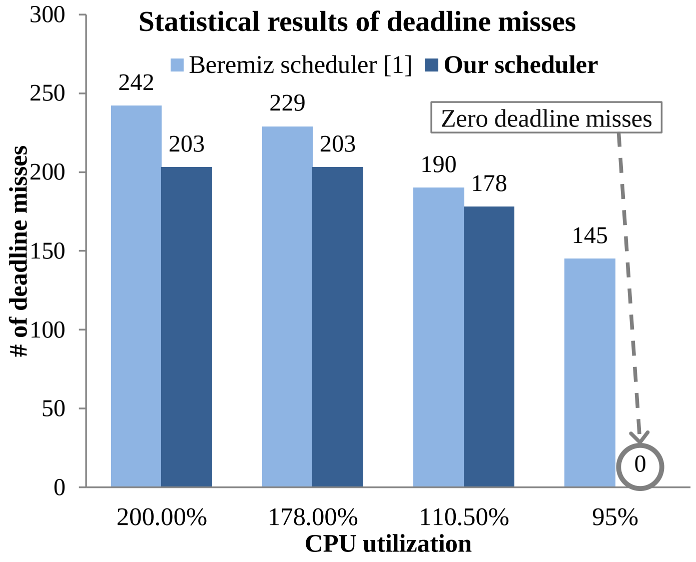


Figure 6: Comparison of our scheduler to Beremiz scheduler

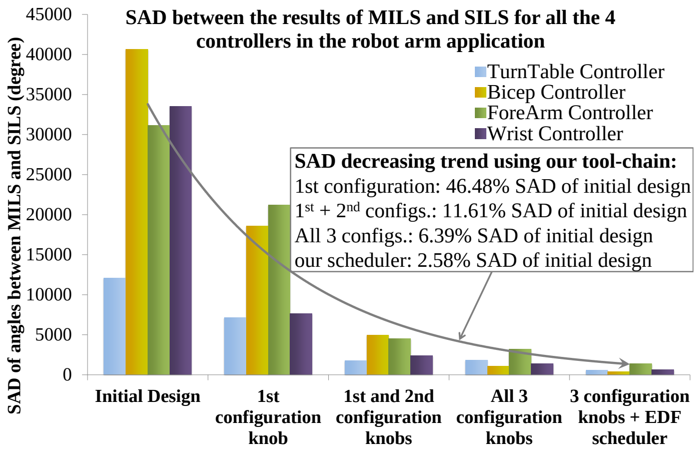
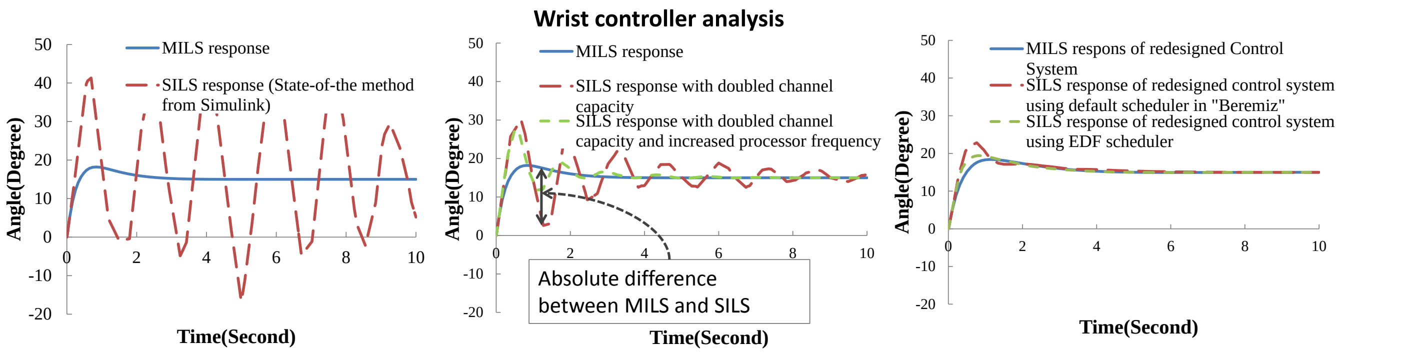


Figure 7: QoC analysis for the robot arm application

VII. Conclusion

This paper presents a MBD approach and associated tool-chain for the design of TTRE systems for digital manufac-



(a) Comparison of MILS result and SILS result without using our tool chain

(b) Comparison of MILS and improved SILS results using our tool chain: **two configuration knobs** are used

(c) Comparison of MILS and improved SILS result using our tool chain**: three configuration knobs** are used and **two different scheduler** are compared

Figure 8: Wrist controller analysis using our tool-chain

turing. Our system is integrated to a SILS framework for the real-time validation of the generated control software using a Soft-PLC. Experiments using our tool chain show a 39× improvement of QoC (2.58% SAD compared to [4]) with zero deadline misses that meets the hard real-time requirement (compared to [1], [26]). Using our configuration knobs, we have demonstrated that a real world manufacturing application can be redesigned in a single iteration. Compared to traditional approaches [3], our approach and tool chain compresses the overall design time.

References

[1] "Beremiz". http://www.beremiz.org/.

[2] "Simscape". http://www.mathworks.com/products/simscape/.

[3] "Simulink". http://www.mathworks.com/products/simulink/.

[4] "Simulink PLC Coder". http://www.mathworks.com/products/simulink-coder/.

[5] A. A. Cabrera, M. Foeken, O. Tekin, K. Woestenenk, M. Erden, B. D. Schutter, M. van Tooren, R. Babuka, F. van Houten, and T. Tomiyama. "Towards automation of control software: A review of challenges in mechatronic design". *IEEE Transactions on Mechatronics*, pages 876 – 886, 2010.

[6] CAN In Automation and International Users and Manufacturers Group. "CANopen Application Layer and Communication Profile, CiA/DS301, Version 4.2.0". 2000.

[7] A. Canedo, L. Dalloro, , and H. Ludwig. "Pipelining for cyclic control systems". *In Proceedings of the 16th ACM international conference on Hybrid systems computation and control, (HSCC'13)*, pages 223–232, 2013.

[8] A. Canedo and M. A. A. Faruque. "Towards parallel execution of IEC 61131 industrial cyber-physical systems applications". *in IEEE/ACM Design Automation and Test in Europe (DATE'12)*, pages 554–557, 2012.

[9] H. Carlsson, B. Svensson, F. Danielsson, and B. Lennartson. "Methods for Reliable Simulation-Based PLC Code Verification". *Industrial Informatics, IEEE Transactions on*, pages 267–278, 2012.

[10] D. Dumbacher and S. R. Davis. "Building operations efficiencies into NASAs Ares I crew launch vehicle design". *In 54th Joint JANNAF Propulsion Conference*, 2007.

[11] N. eV and PROFIBUS. Application layer protocol for decentralized periphery and distributed automation; specification for profinet; version 2.1, june 2006. *Order*, (2.722).

[12] M. A. A. Faruque and A. M. Canedo. "Intelligent and Collaborative Embedded Computing in Automation Engineering". *in IEEE/ACM Design Automation and Test in Europe (DATE'12)*, pages 344–355, 2012.

[13] G. Frey and L. Litz. "Formal methods in PLC programming". *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, pages 2431–2436, 2000.

[14] U. Gangoiti, M. Marcos, and E. Estevez. "Using cyclic executives for achieving closed loop co-simulation". *Decision and Control and European Control Conference (CDC-ECC '05)*, pages 4785–4790, 2005.

[15] D. Gao, C. Mi, and A. Emadi. "Modeling and Simulation of Electric and Hybrid Vehicles". *Proceedings of the IEEE*, pages 729 – 745, 2007.

[16] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. "Time-Triggered Implementations of Mixed-Criticality Automotive Software". *Proceedings of the 15th Conference for Design, Automation and Test in Europe (DATE'12)*, pages 1227–1232, 2012.

[17] R. Heckmann and C. Ferdinand. "Worst-case execution time prediction by static program analysis". *AbsInt Angewandte Informatik GmbH*, 2004.

[18] J. C. Jensen, D. Chang, and E. A. Lee. "A Model-Based Design Methodology for Cyber-Physical Systems". *Wireless Communications and Mobile Computing Conference (IWCMC'11)*, pages 1666–1671, 2011.

[19] K.-H. John and M. Tiegelkamp. *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. Springer, 2010.

[20] G. Madl and N. Dutt. "Model-based Analysis of Event-driven Distributed Real-time Embedded Systems". *Ph.D. Dissertation, University of California, Irvine*, 2009.

[21] M. Marcos, E. Estevez, N. Iriondo, and D. Orive. "Analysis and validation of IEC 61131-3 applications using a MDE approach". *Emerging Technologies and Factory Automation (ETFA'10)*, pages 1–8, 2010.

[22] D. Mendez-Fernandez, B. Penzenstadler, M. Kuhrmann, and M. Broy. "A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering". *In Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, pages 183 – 197, 2010.

[23] N. Mhleis, M. Gla, L. Zhang, and J. Teich. "A co-simulation approach for control performance analysis during design space exploration of cyber-physical systems". *SIGBED Rev.*, pages 23–26, 2011.

[24] A. Sangiovanni-Vincentelli and M. D. Natale. "Embedded System Design for Automotive Applications". *Computer*, pages 42–51, 2007.

[25] K. Thramboulidis and G. Frey. "Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation". *Journal of Software Engineering and Applications*, pages 217–226, 2011.

[26] E. Tisserant, L. Bessard, and M. de Sousa. "An Open Source IEC 61131-3 Integrated Development Environment". *IEEE Transactions on Industrial Informatics*, pages 183–187, 2007.

[27] R. Wilhelm, J.Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, and G. B. et al. "The worst-case execution-time problem overview of methods and survey of tools". *ACM Transactions on Embedded Computing Systems (TECS'08)*, pages 1–53, 2008.