12-2015

Platform-Specific Code Generation from Platform-Independent Timed Models

BaekGyu Kim
University of Pennsylvania, baekgyu@seas.upenn.edu

Lu Feng
University of Pennsylvania, lufeng@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Platform-Specific Code Generation from Platform-Independent Timed Models

Abstract
Many safety-critical real-time embedded systems need to meet stringent timing constraints such as preserving delay bounds between input and output events. In model-based development, a system is often implemented by using a code generator to automatically generate source code from system models, and integrating the generated source code with a platform. It is challenging to guarantee that the implemented systems preserve required timing constraints, because the timed behavior of the source code and the platform is closely intertwined. In this paper, we address this challenge by proposing a model transformation approach for the code generation. Our approach compensates the platform-processing delays by adjusting the timing parameters in system models, based on an Integer Linear Programming problem formulation. We demonstrate the usefulness of our approach via a case study of infusion pump systems. Experimental results show that the code generated using our approach can better preserve the timing constraints.

Platform-Specific Code Generation from Platform-Independent Timed Models

BaekGyu Kim[1,2] Lu Feng[1] Oleg Sokolsky[1] Insup Lee[1]

[1]University of Pennsylvania
[2]Toyota InfoTechnology Center, U.S.A.

***Abstract*—Many safety-critical real-time embedded systems need to meet stringent timing constraints such as preserving delay bounds between input and output events. In model-based development, a system is often implemented by using a code generator to automatically generate source code from system models, and integrating the generated source code with a platform. It is challenging to guarantee that the implemented systems preserve required timing constraints, because the timed behavior of the source code and the platform is closely intertwined. In this paper, we address this challenge by proposing a model transformation approach for the code generation. Our approach compensates the platform-processing delays by adjusting the timing parameters in system models, based on an Integer Linear Programming problem formulation. We demonstrate the usefulness of our approach via a case study of infusion pump systems. Experimental results show that the code generated using our approach can better preserve the timing constraints.**

I. Introduction

Many safety-critical real-time embedded systems need to meet stringent timing constraints, of which a common type is the bounded delay between input and output events. For example, a car should stop (*output event*) within three seconds when a driver hits a brake (*input event*); a pacemaker should generate electrical signals (*output event*) within eight milliseconds when an abnormal heart rhythm is detected (*input event*). The violation of such timing constraints may lead to catastrophic consequences. Thus, it is important to guarantee that the developed real-time embedded systems satisfy the timing constraints. One promising approach is via model-based development. First, a system model is built to capture abstractions of the system's timed behavior. Then, if the model is verified to satisfy the timing constraints, source code is automatically generated from the model using a code generator. Finally, the generated code is implemented on a platform. By platform, we refer to a collection of hardware (*e.g.,* sensors/actuators) and software (*e.g.,* real-time operating systems and I/O device drivers) that are required for the code to read input from or write output to the environment.

System models are typically constructed *independently* of a specific platform for a few benefits: (1) the generated source code can be implemented on many different platforms, and (2) the modeling process can be initiated without knowledge of specific platform in the early system development stage. State-of-the-art code generators, such as Real-Time Workshop [13] and TIMES [4], produce source code using timing parameters specified in a system model directly, with the assumption that the input/output (I/O) communication between the code and the environment can be processed infinitely fast by a platform. However, it is difficult, if not impossible, for any platform to meet this assumption in practice, due to processing delays caused by, *e.g.,* I/O device drivers or communication jitter.

In our previous work, we proposed to overcome this obstacle by modeling architectural aspects of platforms [12] [10] and generating test cases [11]. However, one limitation is that, the code generated from models that incorporate platform-processing delays, when being implemented on a specific platform, may yield a system that does not respect timing constraints verified at the modeling level. Because platform-processing delays are added to the code-level delays. It is challenging to guarantee that the implemented systems preserve required timing constraints, because timing aspects of the system model, the code and the platform are closely intertwined.

In this paper, we aim to address this challenge by proposing a model transformation approach for the code generation. We argue that system models are not appropriate representations of the code-level timed behavior. We transform a system model into a software model by explicitly characterizing (non-zero) platform-processing delays and appropriately compensating those delays. The software model is then used to generate code, which would be implemented on the platform whose processing delays are compensated during the model transformation process. The resulting system implementation is guaranteed to meet the timing constraints that have been verified in the system model. We restrict our attention to timing constraints in the type of bounded delay between I/O events.

Our model transformation approach involves two steps. First, check whether it is feasible to compensate the processing delays of a given platform while preserving bounds on delays between observable events in the system model and implementation. Second, adjust timing parameters in the system model to obtain a software model that compensates the platform-processing delays. We formulate and solve the problem using Integer Linear Programming (ILP). We define the objective function of ILP based on the goal of obtaining a software model with the *least* timed behavior perturbation from the system model. The linear constraints of ILP are given to quantify the delay-bound differences between the system model and implementation, taking into account the model's I/O delays accumulated over paths (between pairs of I/O transitions) and the platform-processing delays. By solving the ILP problem, we obtain a set of timing parameter assignments that can be used to transform the system model into a software model.

We demonstrate the usefulness of our approach via a case study of infusion pump systems. Experimental results show that systems implemented with the code generated from our transformed software models have (significantly) better performance in terms of timing constraints preservation.

The rest of this paper is organized as follows. Section II

*This research was supported in part by NSF CNS-1035715 and the DGIST Research and Development Program of the Ministry of Science, ICT and Future Planning of Korea (CPS Global Center).

*This work has been performed while the first author was at the University of Pennsylvania.

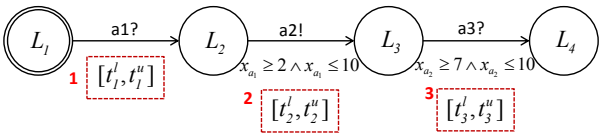

Fig. 1. Model 1 with variable assignment (Sequential Pattern)

introduces a motivating example, the problem statement and an overview of our approach. Section III describes how to compute the minimum and maximum delays of I/O events in the system model and implementation. Section IV develops the ILP problem formulation for the model transformation. Section V presents a case study of infusion pump systems. Finally, Section VI discusses related work and Section VII summarizes our findings.

II. Problem Statement and Approach Overview

A. Motivating Example

We first explain, through the following example, why we need to transform system models for representing the code-level timed behavior. Consider a system model represented as an event-clock automaton [3] (cf. Definition 2) shown in Figure 1. Each transition in the model is labelled with either an input ($a_1?$ and $a_3?$) or an output ($a_2!$) event. Each event is associated with a clock, which is automatically reset to zero whenever a transition associated with the corresponding event is taken. For example, the clock x_{a_1} is reset to zero when the transition labelled with the input event $a_1?$ is taken. Transitions are also annotated with clock guard conditions. For example, the guard condition $x_{a_1} \geq 2 \wedge x_{a_1} \leq 10$ means that the transition can be taken anytime non-deterministically in between 2 and 10 time-unit after the previous transition. It is straightforward to verify that the example model satisfies the following timing constraint (**REQ0**): *"a system shall produce the output* $a_2!$ *within 2 and 10 time-unit since the input* $a_1?$ *event occurs from the environment"*.

The timing parameters in the clock guard conditions do not distinguish code-level delays and platform-processing delays. Suppose the platform-processing delay is 2 time units. Then a system implemented using the code generated from the model shown in Figure 1 would not preserve REQ0, because the delay of output $a_2!$ is now bounded by 4 and 12 time units. This implies that the system model shown in Figure 1 is not an appropriate representation for the code generation. Thus, there is a need for model transformation. In practice, the given platform-processing delays are often in a range (*i.e.,* minimum and maximum bounds) rather than a single constant, which makes the model transformation challenging. Because we need to consider the intermix of min/max platform-processing delays and code-level delays, while the latter may also be affected by the model structure (*e.g.,* loops). Once we know what the ranges of platform delays are, timing guards in the model can be adjusted so that the generated code, running on the platform, would exhibit correct system-level behavior.

B. Problem Statement

Figure 2 shows an overview of relations between system model, software model, code, platform and implementation. We adopt Parnas' four-variable model [14] to define the boundaries of an implemented system (shown in the right side of the figure). In the four-variable model, *monitored* (m) and *controlled* (c) variables characterize changes of physical environmental quantities; on the other hand, *input* (i) and *output* (o) variables characterize software behavior that interact with the physical environment through input/output devices (*i.e.,* platforms in our work). Based on this variable mapping, we define the boundaries of an implemented system to formalize the problem: *io*-boundary that separates a platform and the code, and *mc*-boundary that separates an environment and a platform. Suppose that, for a given platform, the minimum and maximum delays of processing each input/output event is known (cf. Definition 3). The goal is to transform a system model ($\mathsf{M_s}$) into a software model ($\mathsf{M_c}$) by compensating the platform-processing delays (P), in order to preserve the timing constraints in the system implementation.

We first define three functions: $f_{\mathsf{M_s}}$, f_{SOF}, and f_{IMP}, which quantify the min/max delay-bounds of the occurrence of an I/O event succeeding another event in the system model, *io*-boundary (software model), and *mc*-boundary, respectively. Formally, we define $f_{\mathsf{M_s}}(i,j)$ as the min/max delay-bounds of *simple paths* (*i.e.,* those without cycles) starting from the transition i and ending with the transition j in the system model $\mathsf{M_s}$. For example, the timing constraint REQ0 can be formally denoted by $f_{\mathsf{M_s}}(1,2)$=[2,10], representing that the output transition 2 shall be taken in between 2 and 10 time-unit after taking the input transition 1 of Model 1 shown in Figure 1. Given a simple path p, we can also write $f_{\mathsf{M_s}}(p)$. We define f_{SOF} and f_{IMP} in a similar fashion as for $f_{\mathsf{M_s}}$. In the model transformation from $\mathsf{M_s}$ to $\mathsf{M_c}$, we only adjust the timing parameters (*i.e.,* clock guards) and do not change the model structure. Therefore, there are one-to-one mappings between transitions i and j for $f_{\mathsf{M_s}}(i,j)$, $f_{\mathsf{SOF}}(i,j)$ and $f_{\mathsf{IMP}}(i,j)$.

Definition 1 (*Delay-Bound Inclusion Constraint*)**.** A system implementation preserves the delay-bound inclusion constraint with respect to the corresponding system model iff:

- the minimum delay bound of f_{IMP} for any pair of I/O events at the *mc*-boundary is *no less* than that of $f_{\mathsf{M_s}}$,
- the maximum delay bound of f_{IMP} for any pair of I/O events at the *mc*-boundary is *no greater* than that of $f_{\mathsf{M_s}}$.

Formally, we denote the constraint satisfaction by $f_{\mathsf{IMP}}(i,j) \in f_{\mathsf{M_s}}(i,j)$ iff $f^{\mathsf{min}}_{\mathsf{IMP}}(i,j) \geq f^{\mathsf{min}}_{\mathsf{M_s}}(i,j)$ and $f^{\mathsf{max}}_{\mathsf{IMP}}(i,j) \leq f^{\mathsf{max}}_{\mathsf{M_s}}(i,j)$ for any pair of I/O transitions (events) i and j.

Example 1. Model 1 shown in Figure 1 has three pairs of I/O transitions, where $f_{\mathsf{M_s}}(1,2)$=[2,10], $f_{\mathsf{M_s}}(2,3)$=[7,10], $f_{\mathsf{M_s}}(1,3)$=[9,20]. The delay-bound inclusion constraint holds for a system implementation with $f_{\mathsf{IMP}}(1,2)$=[4,6], $f_{\mathsf{IMP}}(2,3)$=[8,9], $f_{\mathsf{IMP}}(1,3)$=[15,19]}, because $f^{\mathsf{min}}_{\mathsf{IMP}} \geq f^{\mathsf{min}}_{\mathsf{M_s}}$ and $f^{\mathsf{max}}_{\mathsf{IMP}} \leq f^{\mathsf{max}}_{\mathsf{M_s}}$ for all possible pairs of I/O transitions. □

The key of model transformation from a system model $\mathsf{M_s}$ into a software model $\mathsf{M_c}$ lies in solving the research problem of finding a suitable function f_{SOF} such that the induced function f_{IMP} (based on the known platform-processing delay P) preserves the delay-bound inclusion constraint with respect to the function $f_{\mathsf{M_s}}$ (determined by $\mathsf{M_s}$). We also need to show that any system implementation meeting the delay-bound inclusion constraint is guaranteed to satisfy the timing requirements that are verified in the system model $\mathsf{M_s}$.

C. Approach Overview

Finding a suitable function f_{SOF} is a challenging problem. On the one hand, we need to consider its dependency to the function f_{IMP} and the platform-processing delay P. On the other hand, we need to make sure that the derived function f_{IMP} and the system model function $f_{\mathsf{M_s}}$ satisfy the delay-bound inclusion constraint. The simple shrinking or expanding timing guards of $f_{\mathsf{M_s}}$ would not give us a satisfying f_{SOF}, because the

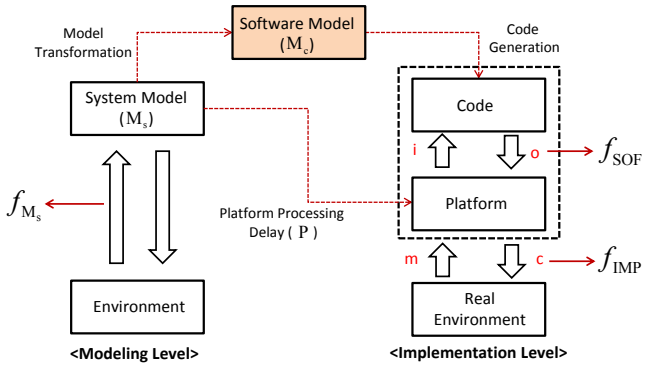

Fig. 2. Mapping from system models to implementations

change of timing guards in one transition (*e.g.,* with the intent of decreasing f_{SOF}) may actually increase f_{SOF} for another transition/path, due to the complex dependency among various I/O transitions and the mixture of minimum/maximum delays.

Our approach is to formalize those dependencies in terms of a set of linear constraints to automatically find timing parameter assignments for the function f_{SOF} using the integer linear programming (ILP). First, we propose algorithmic procedures to compute $f_{\mathsf{M_s}}$ and f_{IMP}. The computation of $f_{\mathsf{M_s}}$ is based on the timing parameters (*i.e.,* clock guard constants) in the system model $\mathsf{M_s}$, while the computation of f_{IMP} is based on the timing parameters of the software model $\mathsf{M_c}$ (represented as variables in f_{SOF}) and the platform-processing delays P. Then, we formalize the delay-bound inclusion constraint between $f_{\mathsf{M_s}}$ and f_{IMP} as a set of linear inequality constraints for ILP. A satisfying solution of the ILP problem gives us a set of variable assignments for f_{SOF}, which can be used to parameterize the timing guards of the software model $\mathsf{M_c}$. Finally, we show by Theorem 1 that the system implemented using the code generated from the software model $\mathsf{M_c}$ is guaranteed to satisfy the timing requirements (*i.e.,* bounded delays between a pair of I/O events).

III. Computing $f_{\mathsf{M_s}}$ and f_{IMP}

In this section, we develop algorithmic procedures for computing $f_{\mathsf{M_s}}$ and f_{IMP}, which are functions that quantify the min/max delay-bound of any pair of I/O transitions and events in the system model and the implementation, respectively. In this paper, we consider system models that are represented as *event-clock automata* [3].

Definition 2 (*Event-Clock Automata*)**.** An event-clock automaton is a tuple $\mathcal{M}$= ($\mathcal{L}$, L_0, L_f, Σ, E), where $\mathcal{L}$ is a set of locations; $L_0 \in \mathcal{L}$ and $L_f \in \mathcal{L}$ are an initial and a final location, respectively; $\Sigma = \Sigma_{in} \cup \Sigma_{out}$ is an alphabet with a set of input (*resp.* output) events Σ_{in} (*resp.* Σ_{out}); $E = \{(L, L', a, \varphi)\}$ is a fine set of transitions with each transition connecting a starting location L and an ending location L', labelled with an input/output event $a \in \Sigma$, and associated with a clock constraint φ over the clocks C_Σ.

We refer to [3] for the formal operational semantics of event-clock automata. An informal semantic description for an example model could be found in Section II-A. In this paper, we consider three different model structure patterns, namely, *sequential* (*e.g.,* Model 1 shown in Figure 1), *alternative* (*e.g.,* Model 2 in Figure 3), and *cyclic* (*e.g.,* Model 3 in Figure 4).

A. Computing $f_{\mathsf{M_s}}$

The computation of the function $f_{\mathsf{M_s}}$, which represents the minimum and maximum delay-bounds between two I/O transitions in the system model, is non-trivial. Firstly, there can be many different paths in between two transition occurrences. In this case, the path that has the minimum delay can be different from the one that has the maximum delay. Therefore, we need to examine all possible paths between the two transitions in order to compute $f_{\mathsf{M_s}}$. Secondly, paths connecting two transitions may include cycles. Since we assume non-negative guard conditions, the minimum delay bound is obtained by considering only a simple path. However, when it comes to the maximum delay bound, it differs depending on how many cycles are taken between the two transitions. For example, in Model 3 (cf. Figure 4), the maximum delay-bound between transitions 1 and 3 increases as more cycles are taken.

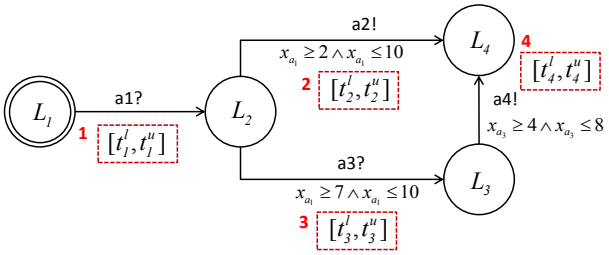

Fig. 3. Model 2 with variable assignment (Alternative Pattern)

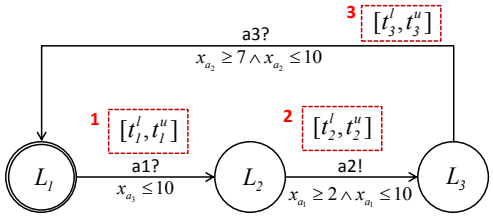

Fig. 4. Model 3 with variable assignment (Cyclic Pattern)

We first show how to compute $f_{\mathsf{M_s}}$ for a pair of transitions which are connected by *simple paths* (*i.e.,* paths that do not contain any repeating locations) only. To this end, we adapt the Floyd-Warshall algorithm. The Floyd-Warshall algorithm computes the minimum and maximum timing intervals between the entering of locations L_i and L_j in a model with n locations, denoted by $D^{\mathsf{min}}(L_i, L_j, n)$ and $D^{\mathsf{max}}(L_i, L_j, n)$, respectively. We cannot use the Floyd-Warshall algorithm to compute $f_{\mathsf{M_s}}$ directly, because the function represents the delay-bound between two transitions rather than locations. Instead, we define:

$$f^{\mathsf{min}}_{\mathsf{M_s}}(i,j) = D^{\mathsf{min}}(L_i^{\mathsf{post}}, L_j^{\mathsf{pre}}, n) + \nu_j^l \quad (1a)$$

$$f^{\mathsf{max}}_{\mathsf{M_s}}(i,j) = D^{\mathsf{max}}(L_i^{\mathsf{post}}, L_j^{\mathsf{pre}}, n) + \nu_j^u \quad (1b)$$

where L_i^{post} is the ending location of transition i, L_j^{pre} is the starting location of transition j, ν_j^l and ν_j^u are the lower and upper bounds of clock valuations associated with transition j. For example, Figures 5(a)-(b) show the results of applying the Floyd-Warshall algorithm to Model 3, while Figures 5(c)-(d) show the computation results of $f_{\mathsf{M_s}}$ for Model 3. Note that the values of the main diagonal entries are undefined in Figures 5(c)-(d), because the method described above is only applicable for transitions connected with simple paths.

Now, we generalize the computation of $f_{\mathsf{M_s}}$ for paths with cycles (*i.e.,* non-simple paths).

Lemma 1. *The min/max delay-bound over a (non-simple) path p in a system model $\mathsf{M_s}$, denoted by $f_{\mathsf{M_s}}(p)$, is the summation*

(a)

$D^{\min}$	L_1	L_2	L_3
L_1	0	0	2
L_2	9	0	2
L_3	7	7	0

(b)

$D^{\max}$	L_1	L_2	L_3
L_1	0	10	20
L_2	20	0	10
L_3	10	20	0

(c)

$f^{\min}_{M_s}$	1	2	3
1	-	2	9
2	7	-	7
3	0	2	-

(d)

$f^{\max}_{M_s}$	1	2	3
1	-	10	20
2	20	-	10
3	10	20	-

Fig. 5. The results of applying the Floyd-Warshall algorithm and computing f_{M_s} for Model 3

of the min/max delay-bounds of all simple paths that comprise p.

Proof: See the Appendix. ■

Note that Lemma 1 is only applicable to event-clock automata where clocks reset on every transition, which is sufficient for this paper. The computation for more general cases (*e.g.,* clocks reset in arbitrary transitions) was studied in [8] using the reachability graph, but that algorithm may not terminate in the presence of cycles.

Example 2. Consider a path p in Model 3 that starts with the transition t_1 and ends with t_3 after repeating two cycles; that is, p=t_1,t_2,t_3,t_1,t_2,t_3. The path p consists of the following simple paths: p_1=t_1,t_2,t_3; p_2=t_3,t_1,t_2; and p_3=t_2,t_3. From Figures 5(c)-(d), we know that $f_{M_s}(1,3)$=[9,20], $f_{M_s}(3,2)$=[2,20], and $f_{M_s}(2,3)$=[7,10]. Based on Lemma 1, we obtain that $f^{\min}_{M_s}(p)$=$f^{\min}_{M_s}(1,3)$+$f^{\min}_{M_s}(3,2)$+$f^{\min}_{M_s}(2,3)$ and $f^{\max}_{M_s}(p)$=$f^{\max}_{M_s}(1,3)$+$f^{\max}_{M_s}(3,2)$+$f^{\max}_{M_s}(2,3)$. Thus, we have $f_{M_s}(p)$=[18,50]. □

B. Computing f_{IMP}

In the following, we describe how to compute the function f_{IMP} based on the platform-processing delay P and the code-level delay f_{SOF}.

Definition 3 (*Platform-Processing Delay*)**.** For any I/O event $a_k \in \Sigma$, the platform-processing delay $P(a_k) = [\delta^{\min}_k, \delta^{\max}_k]$, where $\delta^{\min}_k$ and $\delta^{\max}_k$ are the minimum and maximum times needed for the platform to process the event a_k, respectively.

The platform-processing delay characterizes the min/max delays consumed by a given platform (independently of the code-level delays) to process each I/O event. In Figure 2, this delay is considered as the input delay consumed over the information flow from m to i, and the output delay consumed over the information flow from o to c. These delays occurring over the information flows originate from several delay segments that are necessary for the code to interact with its environment via a platform, such as the timing overhead for processing physical input/output signals incurred by device drivers and scheduling delays. More details of such delay segments can be found in our previous work [11] [10], and this paper assumes that the platform-processing delays are bounded as minimum and maximum parameters.

Example 3. There are three I/O events $\{a_1?, a_2!, a_3?\}$ in Model 1 shown in Figure 1. Suppose the platform-processing delay is given by P=$\{P(a_1)$=[1,2],$P(a_2)$=[3,4],$P(a_3)$=[2,5]$\}$ That is, it takes the platform at least 1 and at most 2 time-unit from the moment it reads the input event a_1 (mapped to m variable in Figure 2) from the environment at the *mc*-boundary until the moment the code reads the processed input event (mapped to i variable) at the *io*-boundary. Similarly, it takes the platform at least 3 and at most 4 time-unit from the moment the code writes the output event a_2 (mapped to o variable) at the *io*-boundary until the moment the platform writes the processed output event (mapped to c variable) to the environment at the *mc*-boundary.

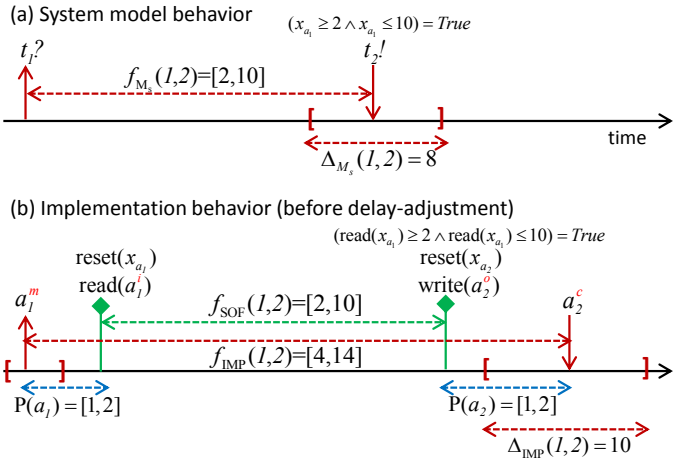

Fig. 6. The timed behavior comparison between the system model (M_s) and the implementation ($P(a_1)$=[1,2] and $P(a_2)$=[1,2]); the arrows imply the events of the *mc*-boundary, while the diamond polls imply the events of the *io*-boundary

Now, we explain the relation between the system implementation delay f_{IMP}, the code-level delay f_{SOF}, and the platform-processing delay P. For example, Figure 6 illustrates how the system model and the implementation behave differently when processing a pair of the *input* transition 1 (t_1) and the *output* transition 2 (t_2) of Model 1. In the system model, t_2 is taken (lower-direction arrow) in between 2 and 10 time-unit since t_1 has been taken (upper-direction arrow); therefore, $f_{M_s}(1,2)$=[2,10]. In the implementation, the delay-bound $f_{IMP}(1,2)$ may differ from $f_{M_s}(1,2)$ if the same timing parameters (T_s) of M_s is used to implement the code-level delay (f_{SOF}). Suppose the code is generated using T_s (*i.e.,* 2 and 10), and this code is to be integrated with a platform that has the processing delay: P=[1,2] (assuming that the same min/max bound is applied to all I/O events). Assume that the code interacts with the platform through a set of primitives: *read* primitive to read the processed input values from the platform or to read current clock values; *reset* primitive to set the clock values to zero; *write* primitive to write the output values to the platform. Note that the time instances when these primitives are called by the code are different from the times when the corresponding I/O occurs in the environment, due to the platform delays.

This implemented system behaves as follows: when the input (denoted as a_1^m) associated with the transition t_1 of M_s is generated from the environment, (1) the platform reads a_1^m first at the *mc*-boundary (red upper arrow); (2) the code reads the processed input (denoted as a_1^i) at the *io*-boundary sometime after, in between 1 and 2 time-unit (first green diamond poll) due to $P(a_1)$, and reset the associated clock (x_{a_1}); (3) the code

produces the output (denoted as a_2^o) at the *io*-boundary (second diamond poll) in between 2 and 10 time-unit after reading the previous input (a_1^i); (4) the platform processes and writes the output (denoted as a_2^c) to the environment at the *mc*-boundary (red lower arrow) in between 1 and 2 time-unit due to $\mathsf{P}(a_2)$. Intuitively, $f_{\mathsf{IMP}}(1,2)$ will be larger than $f_{\mathsf{M_s}}(1,2)$ (i.e., $f_{\mathsf{IMP}} \notin f_{\mathsf{M_s}}$) in this case; this deviation occurs since the I/O information flow and P have not been compensated in implementing the code-level delay (f_{SOF}) (i.e., the code-level delay should have been *shrinked* for the compensation in this case).

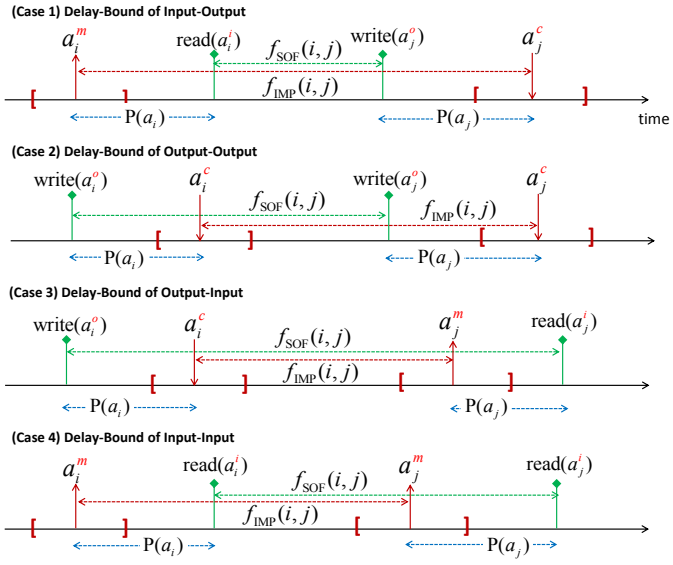

Fig. 7. The four cases of the delay bound computation at the implementation level

In general, f_{IMP} can be larger or smaller than $f_{\mathsf{M_s}}$ depending on the event type and the amount of the platform-processing delay and the dependency among different transitions; the four possible I/O patterns and their implementation behavior are illustrated in Figure 7. Therefore, it is problematic to generate the code using the same parameters (T_{s}) of $\mathsf{M_s}$. Instead, we want to find a new timing parameter assignment (T_{c}) that can be used for the code so that the delay-bound inclusion constraint holds.

In order to find T_{c}, we derive the equation of f_{IMP} by separating two parts: we consider the part that constitutes the platform-processing delay (P) as *known* parameters, while we leave the part that constitutes the code-level delay (f_{SOF}) as *unknown* variables; now, f_{IMP} for any pair of I/O events can be calculated as follows:

Each event (i, j) occurring at the *mc*-boundary is either input or output, hence there are four combinations for a pair of events that can be calculated as follows:

- Case 1: i is an input event and j is an output event:
$[f^{\min}_{\mathsf{SOF}}(i,j)+(\mathsf{P}^{\min}(a_j)+\mathsf{P}^{\min}(a_i)),\ f^{\max}_{\mathsf{SOF}}(i,j)+(\mathsf{P}^{\max}(a_j)+\mathsf{P}^{\max}(a_i))]$
- Case 2: i is an output event and j is an output event:
$[f^{\min}_{\mathsf{SOF}}(i,j)+(\mathsf{P}^{\min}(a_j)-\mathsf{P}^{\max}(a_i)),\ f^{\max}_{\mathsf{SOF}}(i,j)+(\mathsf{P}^{\max}(a_j)-\mathsf{P}^{\min}(a_i))]$
- Case 3: i is an output event and j is an input event:
$[f^{\min}_{\mathsf{SOF}}(i,j)-(\mathsf{P}^{\max}(a_j)+\mathsf{P}^{\max}(a_i)),\ f^{\max}_{\mathsf{SOF}}(i,j)-(\mathsf{P}^{\min}(a_j)+\mathsf{P}^{\min}(a_i))]$
- Case 4: i is an input event and j is an input event:
$[f^{\min}_{\mathsf{SOF}}(i,j)-(\mathsf{P}^{\max}(a_j)-\mathsf{P}^{\min}(a_i)),\ f^{\max}_{\mathsf{SOF}}(i,j)-(\mathsf{P}^{\min}(a_j)-\mathsf{P}^{\max}(a_i))]$

These equations are obtained by the straightforward case analysis illustrated in Figure 7. A detailed derivation in given in Appendix.

IV. Delay-Bound Adjustment using Integer Linear Programming

In this section, we explain how to find the unknown variables (T_{c}) that will be used to implement the code-level delay (f_{SOF}) using the integer linear programming (ILP).

A. Intuition of the Delay-Bound Adjustment

We formalize the ILP problem to find T_{c} to generate the code that can be integrated with a platform preserving the delay-bound inclusion constraint with respect to the system model ($\mathsf{M_s}$). Before explaining the details, the intuition behind this formalization is given.

Consider the timed behavior of $\mathsf{M_s}$ and the implemented system in Figure 6. In case the code is generated using T_{s}, two orthogonal aspects result in the deviation of the timed behavior between $\mathsf{M_s}$ and the implementation.

(1) the deviation of the uncertainty range: the uncertainty range of a pair of I/O transition t_i and t_j of $\mathsf{M_s}$ and the uncertainty range of a pair of the corresponding I/O events of an implemented system is defined as follows:

$$\Delta_{\mathsf{M_s}}(i,j) = f^{\max}_{\mathsf{M_s}}(i,j) - f^{\min}_{\mathsf{M_s}}(i,j) \quad (2a)$$

$$\Delta_{\mathsf{IMP}}(i,j) = f^{\max}_{\mathsf{IMP}}(i,j) - f^{\min}_{\mathsf{IMP}}(i,j) \quad (2b)$$

This range implies that the time interval where the second transition (t_j) is allowed to occur following the first transition (t_i). However, if the code is generated using the original timing parameter assignment (T_{s}), the system model uncertainty ($\Delta_{\mathsf{M_s}}$) is directly implemented as a code as well. The issue is that the chosen platform will add another uncertainty that comes from the platform-processing delay (P). For example, in Fig. 6-(b), the uncertainty range of the implementation (Δ_{IMP}) becomes 10 that is larger than the system model uncertainty ($\Delta_{\mathsf{M_s}}$) by 2; this additional amount comes from P that results in violation of the delay-bound inclusion constraint. To remedy this, we introduce the *Shrinking* operation to adjust *either* upper-bound or lower-bound of the guard condition of the code to compensate P; this will make Δ_{IMP} fit into that of $\Delta_{\mathsf{M_s}}$ by either increasing the lower-bound or decreasing the upper-bound (i.e., $\Delta_{\mathsf{IMP}}(i,j) \leq \Delta_{\mathsf{M_s}}(i,j)$). In Figure 8-(c), for example, the uncertainty range of the implementation is reduced by changing the guard condition associated with t_2 from [2,10] to [4,10] (i.e., the lower bound of the original guard condition is increased by 2).

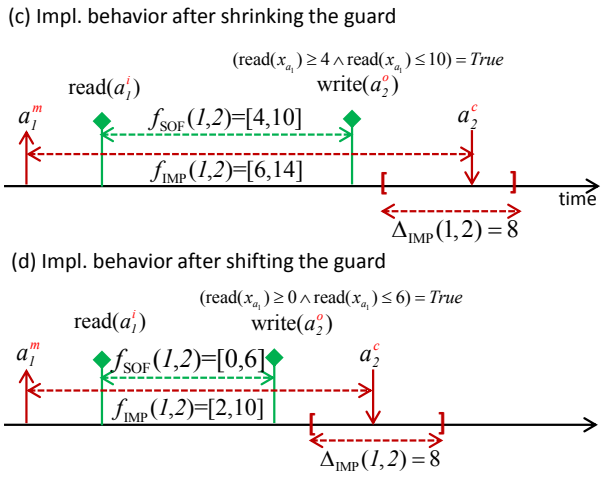

Fig. 8. The illustration of the delay adjustment algorithm

(2) I/O information flow directions: In spite of the shrinking operation in Figure 8-(c), the resulting $f_{\mathsf{IMP}}(1,2)$ is [6,14] that is not included in $f_{\mathsf{M_s}}(1,2)$=[2,10] (i.e. $f_{\mathsf{IMP}}(1,2) \notin f_{\mathsf{M_s}}(1,2)$). This implies that it is not sufficient to perform the *Shrinking* operation only in order to obtain T_{c}. The main reason is that the shrinking operation only makes Δ_{IMP} fit into $\Delta_{\mathsf{M_s}}$, but it does not consider the aspects originating from different combinations of I/O information flows.

Depending on the types of I/O event pairs, the delay-bound of an implementation is deviated from $\mathsf{M_s}$ in four different ways as illustrated in Figure 7. To remedy this, we introduce the *Shift* operation that moves the *relative* timing of the second event occurrence back and forth by adjusting *both* the upper and lower guard condition of the code (c.f., the shrinking operation only adjusts either guard condition). In Figure 8-(d), for example, both the lower and upper guard condition of the code are decreased by 4; the result of this shift operation is that the code should now produce the output in between 0 and 6 time-unit; then the implementation preserves the delay-bound inclusion constraint with respect to $\mathsf{M_s}$ (i.e., $f_{\mathsf{IMP}}(1,2)$=[2,10] $\in f_{\mathsf{M_s}}(1,2)$=[2,10]).

Applying the shrinking/shift operations for all possible pairs of I/O transitions is challenging since there are many different aspects intertwined with each other in a complicated pattern, such as the platform-processing delay, the I/O information flows and the dependency among different I/O transitions. Furthermore, these operations may be impossible under certain circumstances (e.g., Δ_{IMP} is too small relative to the amount of the delay that has to be shrinked). We explain how these conditions are formalized in terms of the ILP so that the two operations (i.e., shrinking and shifting) can be automatically performed for all possible I/O transitions to adjust the code-level delay.

B. Formalization of the ILP Problem for Acyclic Models

We first explain how the problem is formalized for acyclic models, and then explain how it can be extended to cyclic models.

Integer Variable Mapping to the Model: To formalize the ILP problem, we first map integer variables to the system model ($\mathsf{M_s}$) that express the *unknown* min/max code-level delay (f_{SOF}). Here is our mapping: two integer variables, t_k^l and t_k^u, are mapped to each transition of $\mathsf{M_s}$; these variables represent the lower and upper bounds, respectively, of the guard condition associated with the transition. In general, this mapping requires $2n$ variables if $\mathsf{M_s}$ has n transitions. For example, six variables (t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u) are used to represent Model 1. This variable mapping enables the min/max code-level delay (f_{SOF}) for any pair of I/O transitions to be represented in terms of a linear combination of variables. We will use this representation to define constraints as follows:

Defining Linear Constraints: For all pairs of I/O transitions in $\mathsf{M_s}$, we formalize the following constraints using the linear combination of the integer variables:

- (Type 1) The minimum delay-bound of the implemented system should be equal or greater than that of $\mathsf{M_s}$ (i.e., $f_{\mathsf{IMP}}^{\mathsf{min}} \geq f_{\mathsf{M_s}}^{\mathsf{min}}$);
- (Type 2) The maximum delay-bound of the implemented system should be equal or less than that of $\mathsf{M_s}$ (i.e., $f_{\mathsf{IMP}}^{\mathsf{max}} \leq f_{\mathsf{M_s}}^{\mathsf{max}}$);
- (Type 3) Each min/max delay-bound of the code should be non-negative;
- (Type 4) The minimum delay-bound of the code should be equal or less than the maximum delay bound.

Type 1 and type 2 characterize the delay-bound inclusion constraints of Definition 1; we need the constraints of type 3 and type 4 to obtain only non-negative guard conditions whose minimum delay bound is less than the maximum bound.

Example 4 (Linear Constraints for Model 1)**.** Model 1 has three pairs of I/O transitions: (1,2), (2,3), (1,3), and here are the linear constraints:

- (C1a) $t_2^l+\mathsf{P}^{\mathsf{min}}(a_2)+\mathsf{P}^{\mathsf{min}}(a_1)\geq 2$
- (C1b) $t_2^u+\mathsf{P}^{\mathsf{max}}(a_2)+\mathsf{P}^{\mathsf{max}}(a_1)\leq 10$
- (C2a) $t_3^l-(\mathsf{P}^{\mathsf{max}}(a_3)+\mathsf{P}^{\mathsf{max}}(a_2))\geq 7$
- (C2b) $t_3^u-(\mathsf{P}^{\mathsf{min}}(a_3)+\mathsf{P}^{\mathsf{min}}(a_2))\leq 10$
- (C3a) $t_2^l+t_3^l-(\mathsf{P}^{\mathsf{max}}(a_3)-\mathsf{P}^{\mathsf{min}}(a_1))\geq 9$
- (C3b) $t_2^u+t_3^u-(\mathsf{P}^{\mathsf{min}}(a_3)-\mathsf{P}^{\mathsf{max}}(a_1))\leq 20$
- (C4) $t_1^l\geq 0 \wedge t_1^u\geq 0 \wedge t_2^l\geq 0 \wedge t_2^u\geq 0 \wedge t_3^l\geq 0 \wedge t_3^u\geq 0$
- (C5) $t_1^l\leq t_1^u \wedge t_2^l\leq t_2^u \wedge t_3^l\leq t_3^u$

Suppose a platform is characterized as P=[1,2], and the code is assumed to be generated from Model 1. In this case, the linear constraints are obtained by plugging in P as follows: (C1a) $t_2^l\geq 0$; (C1b) $t_2^u\leq 6$; (C2a) $t_3^l\geq 11$; (C2b) $t_3^u\leq 12$; (C3a) $t_2^l\geq 10$; (C3b) $t_2^u+t_3^u\leq 19$; (C4) $t_1^l\geq 0 \wedge t_1^u\geq 0 \wedge t_2^l\geq 0 \wedge t_2^u\geq 0 \wedge t_3^l\geq 0 \wedge t_3^u\geq 0$; (C5) $t_1^l\leq t_1^u \wedge t_2^l\leq t_2^u \wedge t_3^l\leq t_3^u$ □

The constraints (C1a,b), (C2a,b), (C3a,b) are relevant to the pairs of transitions (1,2), (2,3), (1,3), respectively. The constraints (C1a)-(C3a) and (C1b)-(C3b) are the type 1 and type 2 constraints, respectively; the right hand side of these constraints come from the calculated $f_{\mathsf{M_s}}$ according to Lemma 1, while the left hand side are obtained based on the calculations of f_{IMP} in Section III by replacing $f_{\mathsf{SOF}}^{\mathsf{min}}(i,j)$ and $f_{\mathsf{SOF}}^{\mathsf{max}}(i,j)$ with the linear combination of (1) the relevant integer variables that express the *unknown* code-level delay, and (2) the *known* platform-processing delay. On the other hand, the constraint (C4) and (C5) are the type 3 and type 4 constraints needed independently of the platform-processing delay (P).

The linear constraints for Model 2 are also similarly defined, and those constraints and the example are given in Appendix.

Defining Objective Functions for Optimization: Our goal is to find the parameter assignments (T_{c}) for the *unknown* variables that satisfy these linear constraints. Note that there can be many possible parameter assignments that satisfy these linear constraints. For example, in Figure 8, the guard condition (lower bound: 0 and upper bound: 6) for the pair of I/O transition allows the code to write an output (a_2^o) in between 0 and 6 time-unit since it has read the input (a_1^i) with the result of $f_{\mathsf{IMP}}(1,2)$=[2,10]. Another possible assignment is (lower bound: 0, upper bound: 1) that results in $f_{\mathsf{IMP}}(1,2)$=[2,5], which still preserves the delay-bound inclusion property with respect to Model 1.

To this end, we define optimality in choosing T_{c} by considering the following aspect: apart from the platform-processing delay (P), the code also requires some computation time for its internal processing; for example, the code computes the output based on the input according to the control law that can be a complex function. Therefore, a better solution is to find T_{c} that *maximizes* the room for the internal computation of the code. In the above example, the assignment (lower bound: 0, upper bound: 6) is a better solution than the other assignment (lower bound: 0, upper bound: 1) since the code can have more computation time before producing the second event (i.e, 6 time-unit versus 1 time-unit). Such an optimal assignment can be obtained by defining the objective function that maximizes the uncertainty range of the implementation (Δ_{IMP}) as close as to that of the system model ($\Delta_{\mathsf{M_s}}$); this will allow us to find

the largest room for the code computation while preserving the linear constraints.

Suppose a system model ($\mathsf{M_s}$) has n pairs of I/O transitions. Then, we also have n uncertainty ranges for the system model ($\Delta_{\mathsf{M_s}}$) and an implemented system (Δ_{IMP}) that corresponds to each pair of I/O transitions. $\Delta_{\mathsf{M_s}}$ is known since it is calculated from the known T_s, but Δ_{IMP} is unknown since it is calculated from the unknown T_c. Let $\Delta_{\mathsf{IMP}}(k)$ be the uncertainty range of an implemented system for a particular pair (k) of I/O transitions. Then, the general form of our objective function is as follows:

$$\textbf{maximize} \quad \sum_{i=1}^{n} \Delta_{\mathsf{IMP}}(i) \tag{3}$$

For example, Model 1 has three uncertainty ranges that correspond to the three pairs of I/O transitions.

$$\begin{aligned}
\Delta_{\mathsf{IMP}}(1,2) &= (t_2^u + P^{\mathsf{max}}(a_2) + P^{\mathsf{max}}(a_1)) - \\
&\qquad (t_2^l + P^{\mathsf{min}}(a_2) + P^{\mathsf{min}}(a_1)) \\
\Delta_{\mathsf{IMP}}(2,3) &= (t_3^u - (P^{\mathsf{min}}(a_3) + P^{\mathsf{min}}(a_2))) - \\
&\qquad (t_3^l - (P^{\mathsf{max}}(a_3) + P^{\mathsf{max}}(a_2))) \\
\Delta_{\mathsf{IMP}}(1,3) &= (t_2^u + t_3^u - (P^{\mathsf{min}}(a_3) - P^{\mathsf{max}}(a_1))) - \\
&\qquad (t_2^l + t_3^l - (P^{\mathsf{max}}(a_3) - P^{\mathsf{min}}(a_1)))
\end{aligned}$$

The following is the objective function for Model 1:

$$\textbf{maximize} \quad \Delta_{\mathsf{IMP}}(1,2) + \Delta_{\mathsf{IMP}}(2,3) + \Delta_{\mathsf{IMP}}(1,3)$$

Our solver will find the parameter assignment for the six variables (t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u) by maximizing the summation of the uncertainty range of the implementation.

Example 5 (Optimal parameter assignment of Model 1)**.** Suppose the code is to be generated from Model 1, and integrated with a platform characterized as P=[1,2]; then the optimal parameter assignment for the code is (t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u) = (0, INF, 0, 6, 11, 12), where INF implies the absence of the upper bound. Consider another P=[1,3]; then there is no possible parameter assignment for the code generation. □

In Example 5, the solver finds *no* feasible parameter assignment for the code in case the platform with P=[1,3] has to be used. This implies that no code can be generated from Model 1 for this platform by preserving the delay-bound inclusion constraint. However, suppose another platform P=[1,2] is chosen whose maximum I/O processing delay is 1 time-unit faster than the previous platform. In this case, the solver can find the optimal parameter assignment that can be used to generate the code running on this platform. The objective function for Model 2 is also similarly defined, and it is given in Appendix with the example of the optimal assignment.

We believe that one can define more refined notions of optimality in terms of the internal computation time for the code. One possible notion is to give more internal computation time for a particular I/O transition than the other in case adjusting both Δ_{IMP} equally conforms linear constraints. Accommodating such refined optimality will result in more complex objective functions than Equation 3. Exploring all possible notions of optimality and comparing them to each other is out of the scope of this work.

C. Handling Cyclic Models

Note that, if a model contains cycles, there can be an infinite number of paths between a pair of transitions. For example, Model 3 contains a cycle closed by transition 3. Consider a pair of transitions (t_1, t_3); there are an infinite number of paths, corresponding to the number of times the cycle has been followed. Yet, we want to preserve the delay-bound inclusion property for each of these paths. To this end, we show that it is sufficient to consider only the simple paths through the model to guarantee that the delay-bound inclusion property for arbitrary paths, as follows:

Theorem 1. *Given a platform processing delay* P *and a system model* $\mathsf{M_s}$*, if* $f_{\mathsf{IMP}}(i,j) \in f_{\mathsf{M_s}}(i,j)$ *for every pair of transitions* t_i *and* t_j*, then* $f_{\mathsf{IMP}}(p) \in f_{\mathsf{M_s}}(p)$ *for any path* p *that starts with* t_i *and ends with* t_j*.*

Proof: See the Appendix. ■

Example 6 (Linear Constraints for Model 3)**.** According to Theorem 1, we only need to consider a finite number of simple paths in Model 3, and the corresponding linear constraints are listed below:

- (C1a) t_2^l+$\mathsf{P}^{\mathsf{min}}(a_2)$+$\mathsf{P}^{\mathsf{min}}(a_1)\geq 2$
- (C1b) t_2^u+$\mathsf{P}^{\mathsf{max}}(a_2)$+$\mathsf{P}^{\mathsf{max}}(a_1)\leq 10$
- (C2a) t_2^l+t_3^l-($\mathsf{P}^{\mathsf{max}}(a_3)$-$\mathsf{P}^{\mathsf{min}}(a_1)$)$\geq 9$
- (C2b) t_2^u+t_3^u-($\mathsf{P}^{\mathsf{min}}(a_3)$-$\mathsf{P}^{\mathsf{max}}(a_1)$)$\leq 20$
- (C3a) t_3^l+t_1^l-($\mathsf{P}^{\mathsf{max}}(a_2)$+$\mathsf{P}^{\mathsf{max}}(a_1)$)$\geq 7$
- (C3b) t_3^u+t_1^u-($\mathsf{P}^{\mathsf{min}}(a_2)$+$\mathsf{P}^{\mathsf{min}}(a_1)$)$\leq 20$
- (C4a) t_3^l-($\mathsf{P}^{\mathsf{max}}(a_3)$+$\mathsf{P}^{\mathsf{max}}(a_2)$)$\geq 7$
- (C4b) t_3^u-($\mathsf{P}^{\mathsf{min}}(a_3)$+$\mathsf{P}^{\mathsf{min}}(a_2)$)$\leq 10$
- (C5a) t_3^l-($\mathsf{P}^{\mathsf{max}}(a_1)$-$\mathsf{P}^{\mathsf{min}}(a_3)$)$\geq 0$
- (C5b) t_3^u-($\mathsf{P}^{\mathsf{min}}(a_1)$-$\mathsf{P}^{\mathsf{min}}(a_3)$)$\leq 10$
- (C6a) t_1^l+t_2^l+($\mathsf{P}^{\mathsf{min}}(a_3)$+$\mathsf{P}^{\mathsf{min}}(a_2)$)$\geq 2$
- (C6b) t_1^u+t_2^u+($\mathsf{P}^{\mathsf{max}}(a_3)$+$\mathsf{P}^{\mathsf{max}}(a_2)$)$\leq 20$
- (C7) $t_1^l \geq 0 \wedge t_1^u \geq 0 \wedge t_2^l \geq 0 \wedge t_2^u \geq 0 \wedge t_3^l \geq 0 \wedge t_3^u \geq 0$
- (C8) $t_1^l \leq t_1^u \wedge t_2^l \leq t_2^u \wedge t_3^l \leq t_3^u$

□

Now, the objective function can be similarly defined as Model 1 and Model 2; the following is the list of uncertainty ranges that correspond to the pairs of I/O transitions in Model 3:

$$\begin{aligned}
\Delta_{\mathsf{IMP}}(1,2) &= (t_2^u + P^{\mathsf{max}}(a_2) + P^{\mathsf{max}}(a_1)) - \\
&\qquad (t_2^l + P^{\mathsf{min}}(a_2) + P^{\mathsf{min}}(a_1)) \\
\Delta_{\mathsf{IMP}}(1,3) &= (t_2^u + t_3^u - (P^{\mathsf{min}}(a_3) - P^{\mathsf{max}}(a_1))) - \\
&\qquad (t_2^l + t_3^l - (P^{\mathsf{max}}(a_3) - P^{\mathsf{min}}(a_1))) \\
\Delta_{\mathsf{IMP}}(2,1) &= (t_3^u + t_1^u - (P^{\mathsf{min}}(a_2) + P^{\mathsf{min}}(a_1))) - \\
&\qquad (t_3^l + t_1^l - (P^{\mathsf{max}}(a_2) + P^{\mathsf{max}}(a_1))) \\
\Delta_{\mathsf{IMP}}(2,3) &= (t_3^u - (P^{\mathsf{min}}(a_3) + P^{\mathsf{min}}(a_2))) - \\
&\qquad (t_3^l - (P^{\mathsf{max}}(a_3) + P^{\mathsf{max}}(a_2))) \\
\Delta_{\mathsf{IMP}}(3,1) &= (t_3^u - (P^{\mathsf{min}}(a_1) - P^{\mathsf{max}}(a_3))) - \\
&\qquad (t_3^l - (P^{\mathsf{max}}(a_1) - P^{\mathsf{min}}(a_3))) \\
\Delta_{\mathsf{IMP}}(3,2) &= (t_1^u + t_2^u + (P^{\mathsf{max}}(a_3) + P^{\mathsf{max}}(a_2))) - \\
&\qquad (t_1^l + t_2^l + (P^{\mathsf{min}}(a_3) + P^{\mathsf{min}}(a_2)))
\end{aligned}$$

The following is the objective function for Model 3:

$$\begin{aligned}
\textbf{maximize} \quad & \Delta_{\mathsf{IMP}}(1,2) + \Delta_{\mathsf{IMP}}(1,3) + \Delta_{\mathsf{IMP}}(2,1) + \\
& \Delta_{\mathsf{IMP}}(2,3) + \Delta_{\mathsf{IMP}}(3,1) + \Delta_{\mathsf{IMP}}(3,2)
\end{aligned}$$

Example 7 (Optimal parameter assignment of Model 3)**.** Suppose the code is to be generated from Model 3, and integrated with a platform characterized as P=[0,1]; then the optimal parameter assignment for the code is $(t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u)$ = (0, 10, 2, 8, 9, 9). Consider another P=[1,2]; then there is no possible parameter assignment for the code generation. □

V. Case Study: Infusion Pump Systems

An infusion pump is a safety-critical medical device that injects drugs to a patient's body in a controlled manner for medical purposes, such as diabetes treatments or anesthesia. Its hardware platform is equipped with sensors (e.g., a bolus request button, empty reservoir switch) and actuators (e.g., a pump motor, alarm buzzer) to interact with its physical environment (e.g., patient). For example, a patient presses a bolus request button to request an additional amount of drugs; a pump rotates a pump-motor that generates physical forces to move a loaded syringe so that drug can flow from the syringe to the patient. In addition, the pump should detect various alarming conditions using sensor input (e.g., empty/low reservoir condition) to generate alarm signals to caregivers in a timely manner.

In order to show the applicability of the proposed approach, we generate the code for the Baxter II Syringe PCA infusion pump platform used for the GPCA reference implementation project [9], and validated several delay-bound inclusion constraints (the equipment setup is shown in Appendix). Here are the details of the case study:

(1) Modeling and verification for the infusion pump systems: Model 4 in Figure 9 is the system model ($\mathsf{M_s}$) used for this case study, and its informal semantics and the implication in the actual PCA pump implementation are as follows: in the initial location (L_1), the pump waits for the patient's bolus request (in the implementation, the input $mBolusReq$ is generated by pressing a button). However, the pump shall not accept a bolus request occurring earlier than 5000 ms [1] since the previous infusion has been either normally finished by providing the expected amount of drugs or abnormally finished due to the alarming condition (i.e, those premature bolus requests should be ignored). Otherwise, the pump shall process any valid bolus request by taking the transition from L_1 to L_2. In L_2, the pump shall initiate the bolus infusion in between 150 ms and 470 ms by taking the transition from L_2 to L_3 (in the implementation, the output $cStartInfusion$ is produced by rotating the pump motor at a calculated speed). Any bolus infusion will finish in between 300 ms and 750 ms by taking the transition from L_3 to L_1 (in the implementation, the output $cStopInfusion$ is produced by stopping the pump motor rotation), unless it encounters the empty syringe condition. During the infusion, if the empty syringe is detected (in the implementation, the input $mEmptySyringe$ is detected from the switch sensor that is pressed when the syringe hits the bottom), the pump takes a transition from L_3 to L_4 to prepare alarms. In L_4, the pump raises an alarm in between 200 ms and 500 ms by taking the transition from L_4 to L_1 (in the implementation, the output $cAlarm$ is produced by providing signals to a buzzer and alarm LEDs).

Here are three timing constraints considered in this case study:

- (REQ1) *When a patient requests a bolus, the bolus infusion should start in between 150 ms and 470 ms*;
- (REQ2) *The bolus infusion should be active at least 300 ms, and at most 750 ms*;
- (REQ3) *When a patient requests a bolus, the bolus infusion should finish in between 450 ms and 1220 ms in the absense of the empty reservoir alarm.*

[1] We assume 1 time unit in the model is mapped to 1 ms in the implementation

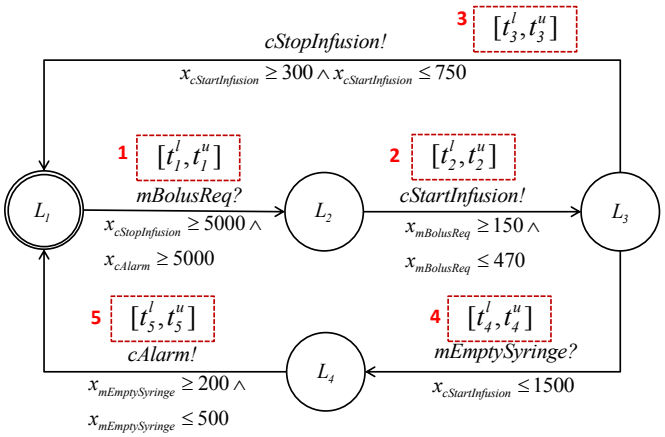

Fig. 9. Model 4 with variable assignment (Simplified GPCA model)

(2) Obtaining the platform-processing delay (P): We measured the platform-processing delay (P) of each input and output of the Baxter PCA pump according to the measurement method introduced in [11], and here is the brief explanation:

To measure the input processing delay, we obtained the timestamp (t_m) when the input is generated from the environment (*mc*-boundary), and obtained another timestamp (t_i) when the code reads the processed input (*io*-boundary); the input processing delay is calculated by subtracting t_m from t_i. To measure the output processing delay, we obtained the timestamp (t_o) when the output is generated from the code (*io*-boundary), and another timestamp (t_c) when the platform actually writes the processed output to the environment (*mc*-boundary); the output processing delay is calculated by subtracting t_o from t_c.

We obtained each timestamp using the oscilloscope that captures the signal changes of the sensors and actuators of the PCA pump (the example of this oscilloscope measurement is given in Appendix). The measurement has been performed 20 times to measure each I/O delay (two inputs and three outputs), and the min/max bound and the average of the measured platform processing delay are summarized in Table. I.

I/O Event	Type	P^{Δ}_{min} (ms)	P^{Δ}_{max} (ms)	Avg. (ms)
mBolusReq?	Input	50	151	105.27
cStartInfusion!	Output	100	303	225.97
cStopInfusion!	Output	98	302	213.64
mEmptySyringe?	Input	104	200	142.8
cAlarm!	Output	298	303	300.05

TABLE I. The measured platform processing delay of the Baxter II syringe pump platform

From this measurement, we characterize the platform-processing delay of the Baxter PCA pump platform as P={P($mBolusReq$)=[50,151],P($cStartInfusion$)=[100,303], P($cStopInfusion$)=[98,302],P($mEmptySyringe$)=[104,200], P($cAlarm$)=[298,303]}.

(3) Formalizing the ILP problem: According to the procedure explained in Section IV, we formalized the ILP problem to find the optimal parameter assignment to generate the code. Firstly, two integer variables are mapped to each I/O transition of the

model, which are unknown variables that need to be solved to implement the code-level delay; a total of 10 integer variables are mapped to the model (t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u,t_4^l,t_4^u,t_5^l,t_5^u). Secondly, $f_{\mathsf{M_s}}$ has been calculated for the simple paths between all pairs of I/O transitions; a total of 20 pairs of $f_{\mathsf{M_s}}$ are calculated. Thirdly, based on the calculated $f_{\mathsf{M_s}}$, 42 linear constraints have been formalized that need to be satisfied for delay-bound inclusion constraint. Finally, the objective function is defined with the given P, and the solver finds the optimal parameter assignment as summarized in Table II. This parameter assignment is used to implement the guard conditions in generating the code that will be executing on the given platform.

	t_1^l	t_1^u	t_2^l	t_2^u	t_3^l	t_3^u	t_4^l	t_4^u	t_5^l	t_5^u
T^{M_s}	5000	INF	150	470	300	750	0	1500	200	500
T^{M_c}	5454	INF	0	16	505	548	503	1704	0	202

TABLE II. THE PARAMETER ASSIGNMENT (T_s) OF $\mathsf{M_s}$ AND THE PARAMETER ASSIGNMENT (T_c) OF $\mathsf{M_c}$

(4) Validating the optimal solution: The goal of the validation is to compare how well the two different codes generated from the system model ($\mathsf{M_s}$) and the software model ($\mathsf{M_c}$) preserve the delay-bound inclusion constraint. We generated the two different codes from each model using the TIMES tool [4]. In order to obtain the implemented system, the generated codes are interfaced with the same platform-specific I/O processing routines whose min/max processing delay-bound is measured as shown in Table I.

In order to validate the delay-bound inclusion constraint, we measured the delay between a pair of I/O events associated with each timing constraint at the *mc*-boundary. For example, in order to validate REQ1, we obtained the timestamp ($t_{mBolusReq}$) for the signal change occurring at the bolus request button (its signal changes from 5 volts to 0 volts if the button is pressed by the patient); and we obtained another timestamp ($t_{cStartInfusion}$) for the signal change occurring at the pump motor (its signal changes from 0 volts to 5 volts if the motor starts rotating); the delay between the two signal changes are measured by subtracting $t_{mBolusReq}$ from $t_{cStartInfusion}$.

For each code, the delay measurement has been performed 20 times for each timing constraint; the trends of the delay-bound for each constraint throughout the testing are shown in Figure 10 11 12; their min/max/average delays are summarized in the two separate f_{IMP} columns of Table III; at the same time, $f_{\mathsf{M_s}}$ is also shown in Table III as well for comparison.

REQ	f^{M_s}		$f^{IMP}(T^{M_s}, P^{\Delta})$			$f^{IMP}(T^{M_c}, P^{\Delta})$		
	Min (ms)	Max(ms)	Min(ms)	Max(ms)	Avg.(ms)	Min(ms)	Max(ms)	Avg.(ms)
REQ1	150	470	603	843	695.8	155	459	283.4
REQ2	300	750	603	907	845	595	619	606.6
REQ3	450	1220	1400	1710	1537	747	1070	887.55

TABLE III. THE VALIDATION RESULT OF THE DELAY-BOUND INCLUSION CONSTRAINT OF THE GPCA REFERENCE IMPLEMENTATION

Analysis: Meeting the delay-bound inclusion constraint means that all the measured implementation delays of each timing constraint should be within the time interval allowed by the system model ($\mathsf{M_s}$) (i.e., $f_{\mathsf{IMP}}^{\mathsf{min}} \geq f_{\mathsf{M_s}}^{\mathsf{min}}$ and $f_{\mathsf{IMP}}^{\mathsf{max}} \leq f_{\mathsf{M_s}}^{\mathsf{max}}$). If at least one delay measurement is out of the interval allowed by $\mathsf{M_s}$, the implementation does not conform to the delay-bound inclusion constraint.

Figure 10, 11, 12 show the validation result of REQ1, REQ2, REQ3, respectively. The purple line and and green line represent $f_{\mathsf{M_s}}^{\mathsf{min}}$ and $f_{\mathsf{M_s}}^{\mathsf{max}}$, respectively. Hence, the implementation

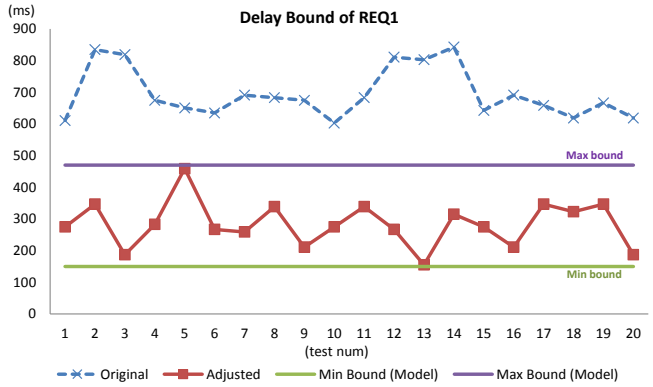

Fig. 10. Validation Result for REQ1

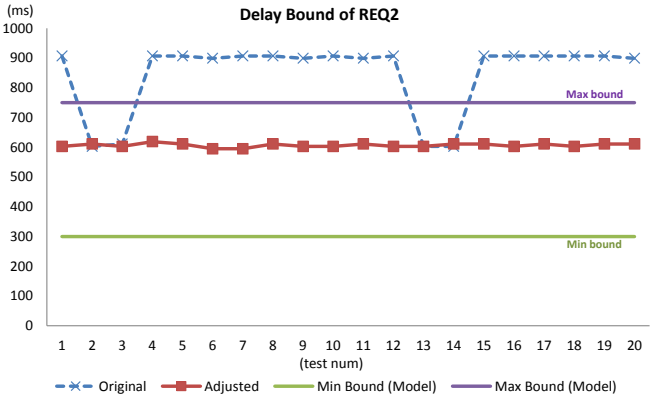

Fig. 11. Validation Result for REQ2

delay (f_{IMP}) is expected to stay within these min/max bounds. The blue line represents the actual delay of the implemented system that executes the code generated from the system model ($\mathsf{M_s}$) without any delay adjustment; on the other hand, the red line represents the actual delay of the implemented system that executes the code generated from the software model ($\mathsf{M_c}$) that has been constructed according to the proposed approach.

According to our testing results, all measured implementation delays of the system running the code generated from $\mathsf{M_c}$ are within the delay-bound allowed by $\mathsf{M_s}$ for the three timing constraints (i.e., the delay-bound inclusion constraint holds under this testing set). On the other hand, most measured implementation delays of the system running the code generated from $\mathsf{M_s}$ are out of the interval allowed by $\mathsf{M_s}$ for all three timing constraints (i.e., the delay-bound inclusion constraint does not hold under this testing set). This result shows that a system model ($\mathsf{M_s}$) is not an appropriate model to generate the code for the timing constraint conformance; instead, a software model ($\mathsf{M_c}$) has to be used to generate the code toward the timing constraint conformance.

VI. RELATED WORK

There have been a number of works to incorporate the platform processing delay in high level models in several different ways. [16] proposed a parametric semantics that allows the reaction time of a platform to be modeled, and an analytic method to check if a platform has sufficient processing power for property preservation. On the other hand, [2] proposed a way to explicitly model software and platforms together to abstract the timed behavior of an implementation. Their approach includes platform modeling such as, digital clock, program execution and I/O interfaces, so that the

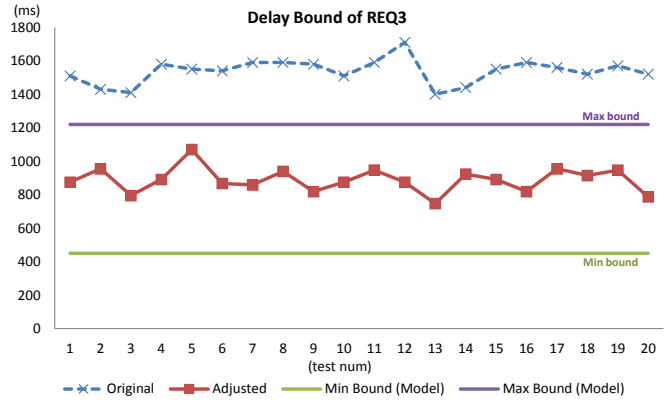

Fig. 12. Validation Result for REQ3

collection of these models can be checked for the property preservation of an implementation. [1] [15] also studied how platform-specific timing delays can be incorporated with the platform-independent model. Their approach is to transform the platform-independent model into a physical model by decomposing each action transition in two separate transitions to express the platform-overhead required for processing events of the platform-independent model. Even though these platform models proposed in each work [16] [2] [1] [15] are useful in analyzing the property preservation of an implementation, these works lack of consideration as to how software part can be generated and integrated with a platform in a way the implementation still holds the property. Furthermore, their platform models are rather too abstract by ignoring some aspects that result in the timed behavior deviation between a model and an implementation. For example, these works lack of consideration as to how I/O dependency and information flow that affect to the timed behavior of an implementation w.r.t. its model.

In comparison to the above works and our work that considered the timed automata semantics in characterizing timing aspects of platforms, there are other works that express the timing aspects in different ways. AADL [6] introduced several components that can express platform's timing aspects such as threads, port connections, and bus components; TrueTime [7] introduced kernel and network blocks that can be connected to the ordinary Simulink blocks so that the temporal behavior of a system that executes control algorithms on a particular platform can be simulated. Even though these modeling strategies do not discuss about the code generation, we believe that the way these works abstract the timing aspects of platforms can complement our work in refining the definition of the platform processing delays.

Regarding to the delay-bound computation, [5] defined three quantitative refinement metrics -(a) maximum time difference, (b) eventual maximum time difference, (c) average time difference.- that quantify the timing mismatches between a model and its implemented systems. Even though this work does not consider how to obtain code from the timed models, we believe that these are more refined metrics to reason about the relationship between a model and its implementations compared to our delay-bound inclusion constraint. We leave applying these metrics to extend our code generation framework as a future work.

VII. Conclusion and Future Work

We proposed a way to transform a system model ($\mathsf{M_s}$) into a software model ($\mathsf{M_c}$) in order to compensate the platform-processing delays (P) for code generation purpose. Our model transformation is performed by formalizing it using integer linear programming in order to preserve the delay-bound inclusion constraint at the implementation level. This approach provides two types of automation in the code generation process: (1) given a platform, it automatically checks whether there exists a code generation strategy that preserves the delay-bound inclusion constraint; (2) (if it is feasible), the software model ($\mathsf{M_c}$) is automatically obtained from the system model ($\mathsf{M_s}$), which guarantees to generate optimal code that can utilize the maximum computation time in between processing any pairs of I/O events. Our case study using the infusion pump system shows the code generated from $\mathsf{M_c}$ better preserves the delay-bound inclusion constraints compared to the code generated from $\mathsf{M_s}$.

As a future work, we plan to study a broader scope of modeling patterns that can be applicable for computation of $f_{\mathsf{M_s}}$ and f_{IMP}. We also plan to extend a notion of platforms to the distributed setting. That is, a system model abstracts timed behavior of distributed systems; and several different code should be generated running on each platform that are connected through network. This requires more refined definition of the platform-processing delays (e.g., incorporating network delays) and a new model transformation strategy for code generation.

References

[1] Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Model-based implementation of real-time applications. In *EMSOFT*. ACM, 2010.

[2] K. Altisen and S. Tripakis. Implementation of timed automata : an issue of semantics or modeling. *FORMATS*, 2005.

[3] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(12):253 – 273, 1999.

[4] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES: a tool for schedulability analysis and code generation of real-time systems. In *FORMATS*, 2003.

[5] Krishnendu Chatterjee and Vinayak S. Prabhu. Quantitative timed simulation functions and refinement metrics for real-time systems. HSCC '13, pages 273–282, New York, NY, USA, 2013. ACM.

[6] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.

[7] Dan Henriksson, Anton Cervin, and Karl erik rzn. Truetime: Real-time control system simulation with matlab/simulink. In *Proc. of the Nordic MATLAB Conference*, 2003.

[8] Inhye Kang and Insup Lee. An efficient state space generation for analysis of real-time systems. *SIGSOFT Softw. Eng. Notes*, 21(3):4–13, May 1996.

[9] Baek Gyu Kim, A. Ayoub, O. Sokolsky, Insup Lee, P. Jones, Yi Zhang, and R. Jetley. Safety-assured development of the gpca infusion pump software. In *EMSOFT '11*, pages 155–164, Oct 2011.

[10] BaekGyu Kim, Lu Feng, Linh T. X. Phan, Oleg Sokolsky, and Insup Lee. Platform-specific timing verification framework in model-based implementation. DATE '15, pages 235–240, 2015.

[11] BaekGyu Kim, Hyeon I Hwang, Taejoon Park, Sang H. Son, and Insup Lee. A layered approach for testing timing in the model-based implementation. In *DATE '14*, pages 1–4, March 2014.

[12] BaekGyu Kim, Linh T. X. Phan, Oleg Sokolsky, and Insup Lee. Platform-dependent code generation for embedded real-time software. In *CASES '13*. IEEE Press, 2013.

[13] MathWorks. Simulink coder - generate c and c++ code from simulink and stateflow models.

[14] David Lorge Parnas and Jan Madey. Functional documents for computer systems. *Science of Computer Programming*, 25:41–61, 1995.

[15] Ahlem Triki, Jacques Combaz, Saddek Bensalem, and Joseph Sifakis. Model-based implementation of parallel real-time systems. In *FASE '13*. 2013.

[16] Martin Wulf, Laurent Doyen, and Jean-Franois Raskin. Almost asap semantics: From timed models to timed implementations. In *HSCC*. 2004.

APPENDIX

A. Notations

Table IV list the notations used in the paper.

Notations	Definitions
M_s	The system model
M_c	The software model
T_s, T_c	The timing parameter assignments to the system model and the software model
t_k^l, t_k^u	The lower/upper bound of clock values associated with a transition k of the software model
v_k^l, v_k^u	The lower/upper bound of clock values associated with a transition k in the system model
$\mathsf{P}(a_k)$ $[\mathsf{P}^{\min}(a_k), \mathsf{P}^{\max}(a_k)]$	The min/max platform processing delay of the event a_k; By adding min and max superscript, each indicates minimum delay bound and maximum delay bound separately.
$f_{\mathsf{M}_s}(i,j)$ $[f_{\mathsf{M}_s}^{\min}(i,j), f_{\mathsf{M}_s}^{\max}(i,j)]$	The min/max delay bound of a transition j succeeding a transition i in the system model; By adding min and max superscript, each indicates minimum delay bound and maximum delay bound separately
$f_{\mathsf{M}_s}(p)$ $[f_{\mathsf{M}_s}^{\min}(p), f_{\mathsf{M}_s}^{\max}(p)]$	The min/max delay bound of a path p in the system model; By adding min and max superscript, each indicates minimum delay bound and maximum delay bound separately
$f_{\mathsf{IMP}}(i,j)$ $[f_{\mathsf{IMP}}^{\min}(i,j), f_{\mathsf{IMP}}^{\max}(i,j)]$	The min/max delay bound of an event j (corresponding to transition j in the system model) succeeding an event i (corresponding to transition i in the system model) at the *mc*-boundary; By adding min and max superscript, each indicates minimum delay bound and maximum delay bound separately
$f_{\mathsf{IMP}}(p)$ $[f_{\mathsf{IMP}}^{\min}(p), f_{\mathsf{IMP}}^{\max}(p)]$	The min/max delay bound of a final event (corresponding to the final transition of p in the system model) succeeding a start event (corresponding to the start transition of p in the system model) at the *mc*-boundary; By adding min and max superscript, each indicates minimum delay bound and maximum delay bound separately
$f_{\mathsf{SOF}}(i,j)$ $[f_{\mathsf{SOF}}^{\min}(i,j), f_{\mathsf{SOF}}^{\max}(i,j)]$	The min/max delay bound of an event j (corresponding to transition j in the system model) succeeding an event i (corresponding to transition i in the system model) at the *io*-boundary; By adding min and max superscript, each indicates minimum delay bound and maximum delay bound separately
$a_k?, a_k!$	The system model-level input/output event (identifier: k)
a_k^m, a_k^i	The implementation-level input event: the input (m) generated from the environment at the *mc*-boundary; the input (i) processed by the platform at the *io*-boundary (identifier: k)
a_k^o, a_k^c	The implementation-level output event: the output (o) generated by the code at the *io*-boundary; the output (c) generated by the platform at the *mc*-boundary (identifier: k)
x_{a_k}	The event clock associated with the event a_k
$\Delta_{\mathsf{M}_s}(i,j)$	The uncertainty range of the event j occurrence following the event i occurrence in the system model
$\Delta_{\mathsf{IMP}}(i,j)$	The uncertainty range of the event j occurrence following the event i occurrence in the implemented system

TABLE IV. NOTATIONS USED IN THE PAPER

B. Computation of $f_{\mathsf{IMP}}(i,j)$

Computation of $f_{\mathsf{IMP}}(i,j)$ follows the case analysis for the four cases illustrated in Figure 7. We only show Case 1 since the rest of the equations are similarly derived.

(Case 1: input t_i and output t_j) Refer to the information flow of Case 1 in Figure 7. Suppose the code reads the input i at τ_i and the code writes the output j at τ_j at the *io*-boundary. The possible range of the input occurrence at the *mc*-boundary is $[\tau_i - \mathsf{P}^{\max}(a_i),\ \tau_i - \mathsf{P}^{\min}(a_i)]$ because the input at the *mc*-boundary has to occur before reading the input at the *io*-boundary. The possible range of the output occurrence at the *mc*-boundary is $[\tau_j + \mathsf{P}^{\min}(a_j),\ \tau_j + \mathsf{P}^{\max}(a_j)]$ because the output at the *mc*-boundary has to occur after writing the output at the *io*-boundary. Then, the minimum interval of these two events is $(\tau_j + \mathsf{P}^{\min}(a_j)) - (\tau_i - \mathsf{P}^{\min}(a_i))$; the maximum interval of these two events is $(\tau_j + \mathsf{P}^{\max}(a_j)) - (\tau_i - \mathsf{P}^{\max}(a_i))$. By rewriting, the min/max interval of the two event is $[\tau_j - \tau_i + \mathsf{P}^{\min}(a_j) + \mathsf{P}^{\min}(a_i),\ \tau_j - \tau_i + \mathsf{P}^{\max}(a_j) + \mathsf{P}^{\max}(a_i)]$. Note that, the term $\tau_j - \tau_i$ is the code-level delay-bound that is *unknown*; the minimum interval of the two events can be obtained by having the minimum interval of the code-level delay; hence, the term $\tau_j - \tau_i$ of the minimum interval is rewritten as $f_{\mathsf{SOF}}^{\min}(i,j)$ that indicates the *unknown* minimum interval of the code-level delay. The maximum interval of the two events can be obtained by having the maximum interval of the code-level delay; hence, the term $\tau_j - \tau_i$ of the maximum interval is rewritten as $f_{\mathsf{SOF}}^{\max}(i,j)$ that indicates the *unknown* maximum interval of the code-level delay. As a result, we obtain the final equation of $f_{\mathsf{IMP}}(i,j)$ for Case 1 as follows:

$$[f_{\mathsf{SOF}}^{\min}(i,j) + (\mathsf{P}^{\min}(a_j) + \mathsf{P}^{\min}(a_i)),\ f_{\mathsf{SOF}}^{\max}(i,j) + (\mathsf{P}^{\max}(a_j) + \mathsf{P}^{\max}(a_i))]$$

C. Proofs of Lemma and Theorem

Lemma 1 The min/max delay-bound over an (non-simple) path p in a system model M_s, denoted by $f_{\mathsf{M}_s}(p)$, is the summation of the min/max delay-bounds of all simple paths that comprise p.

Proof: Suppose p consists of a series of transitions $t_i \ldots t_j$, where t_i is the starting transition of p, and t_j is the ending transition of p. Let $t_1^p \ldots t_k^p$ be the ending (and starting) transitions in between t_i and t_j that comprises all simple paths $\in p$; that is, $p = t_i \ldots t_1^p \ldots t_2^p \ldots t_k^p \ldots t_j$ with a set of simple paths $t_i \ldots t_1^p$, and $t_1^p \ldots t_2^p, \ldots$, and $t_{k-1}^p \ldots t_k^p$, and $t_k^p \ldots t_j$. In an event-clock automaton, the min/max interval of a particular simple path can be calculated independently of the min/max interval of its adjacent simple paths (i.e., the min/max interval of a prior simple path immediately followed by this simple path, and the post simple path immediately following this simple path) using an event-recording clock that is reset to zero upon taking a starting transition of a simple path. Therefore, the minimum interval of $f_{\mathsf{M}_s}(p)$ is calculated as follows: $f_{\mathsf{M}_s}^{\min}(p) = f_{\mathsf{M}_s}^{\min}(t_i, t_1^p) + f_{\mathsf{M}_s}^{\min}(t_1^p, t_2^p) + \ldots + f_{\mathsf{M}_s}^{\min}(t_{k-1}^p, t_k^p) + f_{\mathsf{M}_s}^{\min}(t_k^p, t_j)$; and the maximum interval of $f_{\mathsf{M}_s}(p)$ is also similarly calculated. ■

Theorem 1 Given a platform processing delay P and a system model M_s, if $f_{\mathsf{IMP}}(i,j) \in f_{\mathsf{M}_s}(i,j)$ for all possible pair of transitions t_i and t_j, then $f_{\mathsf{IMP}}(p) \in f_{\mathsf{M}_s}(p)$ for all paths p that starts with t_i and ends with t_j.

Proof: (Proof by contradiction) Suppose $\exists\ p$ such that $f_{\mathsf{IMP}}(p) \notin f_{\mathsf{M}_s}(p)$, where $p = t_i \ldots t_1^p \ldots t_2^p \ldots t_k^p \ldots t_j$ with a set of simple paths $t_i \ldots t_1^p$, and $t_1^p \ldots t_2^p, \ldots$, and $t_{k-1}^p \ldots t_k^p$, and $t_k^p \ldots t_j$. By Lemma 1, $f_{\mathsf{M}_s}(p)$ is calculated as follows:

$$f_{\mathsf{M}_s}^{\min}(p) = f_{\mathsf{M}_s}^{\min}(t_i, t_1^p) + f_{\mathsf{M}_s}^{\min}(t_1^p, t_2^p) + \ldots + f_{\mathsf{M}_s}^{\min}(t_{k-1}^p, t_k^p) + f_{\mathsf{M}_s}^{\min}(t_k^p, t_j)$$
$$f_{\mathsf{M}_s}^{\max}(p) = f_{\mathsf{M}_s}^{\max}(t_i, t_1^p) + f_{\mathsf{M}_s}^{\max}(t_1^p, t_2^p) + \ldots + f_{\mathsf{M}_s}^{\max}(t_{k-1}^p, t_k^p) + f_{\mathsf{M}_s}^{\max}(t_k^p, t_j)$$

Since $f_{\mathsf{IMP}}(p) \notin f_{\mathsf{M}_s}(p)$, either of the following cases must be satisfied:

$$\text{(Case 1) } f_{\mathsf{IMP}}^{\min}(p) < f_{\mathsf{M}_s}^{\min}(p)$$
$$\text{(Case 2) } f_{\mathsf{IMP}}^{\max}(p) > f_{\mathsf{M}_s}^{\max}(p)$$

Suppose (Case 1) holds; then,

$$f_{\mathsf{IMP}}^{\min}(t_i, t_1^p) + f_{\mathsf{IMP}}^{\min}(t_1^p, t_2^p) + \ldots + f_{\mathsf{IMP}}^{\min}(t_{k-1}^p, t_k^p) + f_{\mathsf{IMP}}^{\min}(t_k^p, t_j) < f_{\mathsf{M}_s}^{\min}(t_i, t_1^p) + f_{\mathsf{M}_s}^{\min}(t_1^p, t_2^p) + \ldots + f_{\mathsf{M}_s}^{\min}(t_{k-1}^p, t_k^p) + f_{\mathsf{M}_s}^{\min}(t_k^p, t_j).$$

However, this is not possible due to the assumption that $f_{\mathsf{IMP}}^{\min}(t_i, t_j) \geq f_{\mathsf{M}_s}^{\min}(t_i, t_j)$ for $\forall\ t_i,\ t_j$.

Suppose (Case 2) holds; then,

$$f_{\mathsf{IMP}}^{\max}(t_i, t_1^p) + f_{\mathsf{IMP}}^{\max}(t_1^p, t_2^p) + \ldots + f_{\mathsf{IMP}}^{\max}(t_{k-1}^p, t_k^p) + f_{\mathsf{IMP}}^{\max}(t_k^p, t_j) > f_{\mathsf{M}_s}^{\max}(t_i, t_1^p) + f_{\mathsf{M}_s}^{\max}(t_1^p, t_2^p) + \ldots + f_{\mathsf{M}_s}^{\max}(t_{k-1}^p, t_k^p) + f_{\mathsf{M}_s}^{\max}(t_k^p, t_j).$$

However, this is not possible due to the assumption that $f_{\mathsf{IMP}}^{\max}(t_i, t_j) \leq f_{\mathsf{M}_s}^{\max}(t_i, t_j)$ for $\forall\ t_i,\ t_j$.

This constradicts the fact that either (Case 1) or (Case 2) must be satisfied. ■

D. ILP Formalization of Model 2

Example 8 (Linear Constraints for Model 2)**.** Model 2 has four pairs of I/O transitions: (1,2), (1,3), (1,4), (3,4), and here are the linear constraints:

- (C1a) $t_2^l + \mathsf{P}^{\min}(a_2) + \mathsf{P}^{\min}(a_1) \geq 2$
- (C1b) $t_2^u + \mathsf{P}^{\max}(a_2) + \mathsf{P}^{\max}(a_1) \leq 10$
- (C2a) $t_3^l - (\mathsf{P}^{\max}(a_3) - \mathsf{P}^{\min}(a_1)) \geq 7$
- (C2b) $t_3^u - (\mathsf{P}^{\min}(a_3) - \mathsf{P}^{\max}(a_1)) \leq 10$
- (C3a) $t_3^l + t_4^l + (\mathsf{P}^{\min}(a_4) + \mathsf{P}^{\min}(a_1)) \geq 11$
- (C3b) $t_3^u + t_4^u + (\mathsf{P}^{\max}(a_4) + \mathsf{P}^{\max}(a_1)) \leq 18$
- (C4a) $t_4^l + (\mathsf{P}^{\min}(a_4) + \mathsf{P}^{\min}(a_3)) \geq 4$
- (C4b) $t_4^u + (\mathsf{P}^{\max}(a_4) + \mathsf{P}^{\max}(a_3)) \leq 8$
- (C5) $t_1^l \geq 0 \wedge t_1^u \geq 0 \wedge t_2^l \geq 0 \wedge t_2^u \geq 0 \wedge t_3^l \geq 0 \wedge t_3^u \geq 0 \wedge t_4^l \geq 0 \wedge t_4^u \geq 0$
- (C6) $t_1^l \leq t_1^u \wedge t_2^l \leq t_2^u \wedge t_3^l \leq t_3^u \wedge t_4^l \leq t_4^u$

Suppose a platform is given characterized as P=[2,4]; and the code is assumed to be generated from Model 2. The corresponding linear constraints are as follows: (C1a) $t_2^l \geq -2$; (C1b) $t_2^u \leq 2$; (C2a) $t_3^l \geq 9$; (C2b) $t_3^u \leq 8$; (C3a) $t_3^l \geq 7$; (C3b)

$t_3^u+t_4^u \leq 10$; (C4a) $t_4^l \geq 0$; (C4b) $t_4^u \leq 0$; (C5) $t_1^l \geq 0 \wedge t_1^u \geq 0 \wedge t_2^l \geq 0 \wedge t_2^u \geq 0 \wedge t_3^l \geq 0 \wedge t_3^u \geq 0 \wedge t_4^l \geq 0 \wedge t_4^u \geq 0$; (C6) $t_1^l \leq t_1^u \wedge t_2^l \leq t_2^u \wedge t_3^l \leq t_3^u \wedge t_4^l \leq t_4^u$ □

Model 2 has four uncertainty ranges that correspond to the four pairs of I/O transitions.

$$\Delta_{\mathsf{IMP}}(1,2) = (t_2^u + P^{\max}(a_2) + P^{\max}(a_1)) - (t_2^l + P^{\min}(a_2) + P^{\min}(a_1))$$

$$\Delta_{\mathsf{IMP}}(1,3) = (t_3^u - (P^{\min}(a_3) - P^{\max}(a_1))) - (t_3^l - (P^{\max}(a_3) - P^{\min}(a_1)))$$

$$\Delta_{\mathsf{IMP}}(1,4) = (t_3^u + t_4^u + (P^{\max}(a_4) + P^{\max}(a_1))) - (t_3^l + t_4^l + (P^{\min}(a_4) + P^{\min}(a_1)))$$

$$\Delta_{\mathsf{IMP}}(3,4) = (t_4^u + (P^{\max}(a_4) + P^{\max}(a_3))) - (t_4^l + (P^{\min}(a_4) + P^{\min}(a_3)))$$

The following is the objective function for Model 2:

maximize $\Delta_{\mathsf{IMP}}(1,2) + \Delta_{\mathsf{IMP}}(1,3) + \Delta_{\mathsf{IMP}}(1,4) + \Delta_{\mathsf{IMP}}(3,4)$

This objective function is obtained in a similar way with Model 1 case. Our solver will find the parameter assignment for the eight variables (t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u,t_4^l,t_4^u).

Example 9 (Optimal parameter assignment of Model 2)**.** Suppose the code is to be generated from Model 2, and integrated with a platform characterized as P=[2,4]; then there is no possible parameter assignment for the code. Suppose the code is to be generated from the same model, and integrated with another platform characterized as P=[2,3]; then the optimal parameter assignment for the code is (t_1^l,t_1^u,t_2^l,t_2^u,t_3^l,t_3^u,t_4^l,t_4^u) = (0, INF, 0, 4, 8, 9, 0, 2). □

In Example 9, the solver finds *no* fesible parameter assignment for the code in case the platform with P=[2,4] has to be used. This implies no code can be generated from Model 2 for this platform by preserving the delay-bound inclusion constraint. However, suppose another platform P=[2,3] is chosen whose maximum I/O processing delay is 1 time-unit faster than the previous platform. In this case, the solver can find the optimal paramter assignment that can be used to generate the code running on this platform.

E. ILP Formalization of Model 4

Model 4 has total 42 linear constraints as follows:

- (C1a) $t_2^l+\mathsf{P}^{\min}(a_2)+\mathsf{P}^{\min}(a_1) \geq 150$
- (C1b) $t_2^u+\mathsf{P}^{\max}(a_2)+\mathsf{P}^{\max}(a_1) \leq 600$
- (C2a) $t_2^l+t_3^l+\mathsf{P}^{\min}(a_3)+\mathsf{P}^{\min}(a_1) \geq 900$
- (C2b) $t_2^u+t_3^u+\mathsf{P}^{\max}(a_3)+\mathsf{P}^{\max}(a_1) \leq 2100$
- (C3a) $t_2^l+t_4^l-(\mathsf{P}^{\max}(a_4)-\mathsf{P}^{\min}(a_1)) \geq 150$
- (C3b) $t_2^u+t_4^u-(\mathsf{P}^{\min}(a_4)-\mathsf{P}^{\max}(a_1)) \leq 2100$
- (C4a) $t_2^l+t_4^l+t_5^l+\mathsf{P}^{\min}(a_5)+\mathsf{P}^{\min}(a_1) \geq 350$
- (C4b) $t_2^u+t_4^u+t_5^u+\mathsf{P}^{\max}(a_5)+\mathsf{P}^{\max}(a_1) \leq 2600$
- (C5a) $\min[t_3^l+t_1^l-(\mathsf{P}^{\max}(a_1)+\mathsf{P}^{\max}(a_2)), t_4^l+t_5^l+t_1^l-(\mathsf{P}^{\max}(a_1)+\mathsf{P}^{\max}(a_2))] \geq 5200$
- (C5b) $\max[t_3^u+t_1^u-(\mathsf{P}^{\min}(a_1)+\mathsf{P}^{\min}(a_2)), t_4^u+t_5^u+t_1^u-(\mathsf{P}^{\min}(a_1)+\mathsf{P}^{\min}(a_2))] \leq$ INF
- (C6a) $t_3^l+\mathsf{P}^{\min}(a_3)-\mathsf{P}^{\max}(a_2) \geq 750$
- (C6b) $t_3^u+\mathsf{P}^{\max}(a_3)-\mathsf{P}^{\min}(a_2) \leq 1500$
- (C7a) $t_4^l-(\mathsf{P}^{\max}(a_4)+\mathsf{P}^{\max}(a_2)) \geq 0$
- (C7b) $t_4^u-(\mathsf{P}^{\min}(a_4)+\mathsf{P}^{\min}(a_2)) \leq 1500$
- (C8a) $t_4^l+t_5^l+(\mathsf{P}^{\min}(a_5)-\mathsf{P}^{\max}(a_2)) \geq 200$
- (C8b) $t_4^u+t_5^u+(\mathsf{P}^{\max}(a_5)-\mathsf{P}^{\min}(a_2)) \leq 2000$
- (C9a) $t_1^l-(\mathsf{P}^{\max}(a_3)+\mathsf{P}^{\max}(a_1)) \geq 5000$
- (C9b) $t_1^u-(\mathsf{P}^{\min}(a_3)+\mathsf{P}^{\min}(a_1)) \leq$ INF
- (C10a) $t_1^l+t_2^l+(\mathsf{P}^{\min}(a_3)-\mathsf{P}^{\max}(a_2)) \geq 5150$
- (C10b) $t_1^u+t_2^u+(\mathsf{P}^{\max}(a_3)-\mathsf{P}^{\min}(a_2)) \leq$ INF
- (C11a) $t_1^l+t_2^l+t_4^l-(\mathsf{P}^{\max}(a_4)+\mathsf{P}^{\max}(a_3)) \geq 5150$
- (C11b) $t_1^u+t_2^u+t_4^u-(\mathsf{P}^{\min}(a_4)+\mathsf{P}^{\min}(a_3)) \leq$ INF
- (C12a) $t_1^l+t_2^l+t_4^l+t_5^l+(\mathsf{P}^{\min}(a_5)-\mathsf{P}^{\max}(a_3)) \geq 5350$
- (C12b) $t_1^u+t_2^u+t_4^u+t_5^u+(\mathsf{P}^{\max}(a_5)-\mathsf{P}^{\min}(a_3)) \leq$ INF
- (C13a) $t_5^l+t_1^l-(\mathsf{P}^{\max}(a_1)-\mathsf{P}^{\min}(a_4)) \geq 5200$
- (C13b) $t_5^u+t_1^u-(\mathsf{P}^{\min}(a_1)-\mathsf{P}^{\max}(a_4)) \leq$ INF
- (C14a) $t_5^l+t_1^l+t_2^l+\mathsf{P}^{\min}(a_2)+\mathsf{P}^{\min}(a_4) \geq 5350$
- (C14b) $t_5^u+t_1^u+t_2^u+\mathsf{P}^{\max}(a_2)+\mathsf{P}^{\max}(a_4) \leq$ INF
- (C15a) $t_5^l+t_1^l+t_2^l+t_3^l+\mathsf{P}^{\min}(a_3)+\mathsf{P}^{\min}(a_4) \geq 6100$
- (C15b) $t_5^u+t_1^u+t_2^u+t_3^u+\mathsf{P}^{\max}(a_3)+\mathsf{P}^{\max}(a_4) \leq$ INF
- (C16a) $t_5^l+\mathsf{P}^{\min}(a_5)+\mathsf{P}^{\min}(a_4) \geq 200$
- (C16b) $t_5^u+\mathsf{P}^{\max}(a_5)+\mathsf{P}^{\max}(a_4) \leq 500$
- (C17a) $t_1^l-(\mathsf{P}^{\max}(a_5)+\mathsf{P}^{\max}(a_1)) \geq 5000$
- (C17b) $t_1^u-(\mathsf{P}^{\min}(a_5)+\mathsf{P}^{\min}(a_1)) \leq$ INF
- (C18a) $t_1^l+t_2^l+(\mathsf{P}^{\min}(a_2)-\mathsf{P}^{\max}(a_5)) \geq 5150$
- (C18b) $t_1^u+t_2^u+(\mathsf{P}^{\max}(a_2)-\mathsf{P}^{\min}(a_5)) \leq$ INF
- (C19a) $t_1^l+t_2^l+t_3^l+(\mathsf{P}^{\min}(a_3)-P_{max}^5) \geq 5900$
- (C19b) $t_1^u+t_2^u+t_3^u+(\mathsf{P}^{\max}(a_3)-\mathsf{P}^{\min}(a_5)) \leq$ INF
- (C20a) $t_1^l+t_2^l+t_4^l-(\mathsf{P}^{\max}(a_4)-\mathsf{P}^{\max}(a_5)) \geq 5150$
- (C20b) $t_1^u+t_2^u+t_4^u-(\mathsf{P}^{\min}(a_4)+\mathsf{P}^{\min}(a_5)) \leq$ INF
- (C21) $t_1^l \geq 0 \wedge t_1^u \geq 0 \wedge t_2^l \geq 0 \wedge t_2^u \geq 0 \wedge t_3^l \geq 0 \wedge t_3^u \geq 0 \wedge t_4^l \geq 0 \wedge t_4^u \geq 0 \wedge t_5^l \geq 0 \wedge t_5^u \geq 0$
- (C22) $t_1^l \leq t_1^u \wedge t_2^l \leq t_2^u \wedge t_3^l \leq t_3^u \wedge t_4^l \leq t_4^u \wedge t_5^l \leq t_5^u$

The uncertainty ranges and the objective function for Model 4 are similary defined with Model 1, 2, 3.

F. Case Study Platform Setup

Figure 13 shows the case study platform setup and the measurement method of the delay-bounds using the oscilloscope. The infusion pump platform shown in (a) is equipped with sensors (bolus request button, empty/low reservoir detection switch, door open detection switch) and actuators (pump-motor, alarm LED, buzzer) that are necessary to implement the drug administration process and alarming condition detection. These sensors/actuators are interfaced with ARM7 microcontroller that is running FreeRTOS. The generated code from the software model ($\mathsf{M_c}$) executes as a periodic thread along with other threads that interact with sensors and actuators.

The two oscilloscope screen-shots compare the signal changes that are occurring at the sensor (bolus request button) and the actuator (pump-motor) implemented by the two codes generated from $\mathsf{M_s}$ and $\mathsf{M_c}$, respectively. The delay-bounds associated with REQ1, REQ2, REQ3 are measured by time-stamping these signal changes. These measurements are summarized in Figure 10 11 12 in Section V.

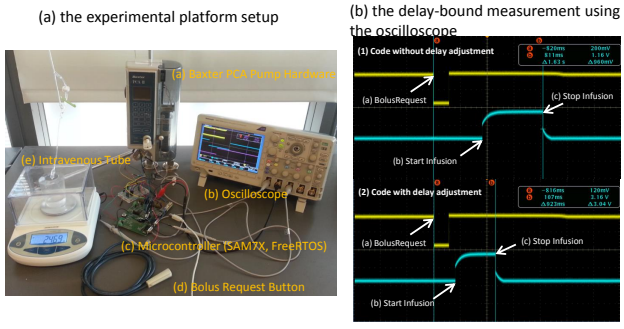

Fig. 13. (a) Case study platform setup (Baxter PCA infusion pump), (b) Delay measurement using the oscilloscope for validation