

Extended Top Down Method from SR model to Runnables in AUTOSAR

Shunsuke Hori

Abstract—Model based development has become important. In MATLAB/Simulink which is a model based development tool, synchronous reactive models (SR models) are used. Using SR models, **developpers** can describe AUTOSAR software components. Then, **developpers** must map blocks of SR models to AUTOSAR runnables that are units of processing in AUTOSAR. This paper proposes a top down mapping method from SR models to runnables considering optimal schedulability and modularity, while achieving schedulability and code size.

1. INTRODUCTION

Embedded system development has become large scale and complication innovation of process in embedded system development is needed. Conventionally, we develop it with paper specification, well we would like to use tools, for example MATLAB/Simulink, to realize low-cost and fast development. This is Model base development. Four characteristics of Model based development is as below.

- Representation or definition of a measure due to a model
- Verification due to simulation from a model
- Automatic code generation from a model
- Reuse of a model in test or verification

Due to these characteristics, low-cost and speedy development is made possible. AUTomotive Open System ARchitecture (AUTOSAR) is global development partnership in motor vehicle industry. The purpose of it is to develop an open industry standard for automotive software architecture. Conventionally, we develop software depending on hardware, but that is inefficient. In addition, due to having hierarchical structure, we can localize the parts depending on hardware. We can develop them reusable, and it is very efficient. AUTOSAR architecture consists of three layer, application layer, RTE layer, basic software layer. (fig1) The application layer has some software components (SWCs). Each SWC implements a function of the ECU in the unit called runnable. AUTOSAR architecture is modular structure, therefore we can develop an application in component units, engine control and body control, such as light and wiper control. The RTE layer is middle layer which provide communication services to application software. The RTE uses the virtual functional bus (VFB), and realizes the communication between SWCs and between SWCs and the basic software modules. The RTE provides API for communication between SWCs and **exclusion** control. When we develop SWCs, we do not have to accommodate lower

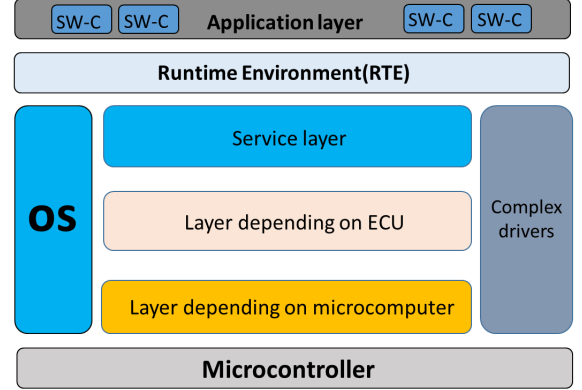


Figure 1. fig1

than LTE layer. Therefore, we can develop SWCs without depending on ECUs, and SWCs are reusable. The basic software layer is **divided** into three layers: a services layer, an ECU abstraction layer, a Microcontroller Abstraction Layer. Because of this, reusability of basic software advances.

organization: The rest of this paper is organized as follows. Section2 provides a fundamental knowledge for mapping method from block diagrams to runnables. Section3 gives an existing mapping **algorizm** and problem of it. Section4 proposes an extended method **algorizm**. Section5 measures and prepare with existing method and new one. Section6 discusses related work. Section7 concludes this paper.

2. A FUNDAMENTAL KNOWLEADGE

2.1. SR model

A SR model is used for modeling systems, which many things are **happining** concurrently. Whereas a dataflow model is good at managing currents of data, a SR model is good at managing sporadic data, where events do not have to exist. A SR model can detect or adapt faults by that events do not exist. Therefore, a SR model can implement precise control. SR model is a model expressed by a synchronous block diagram (SBD). SBD has a set of subsystems or components $S = \{C_j\}$. Each **Compo-**
nennt C_j has blocks $B^j = \{b_1^j, b_2^j, \dots, b_{m^j}^j\}$, a set of inputs $X^j = \{x_1^j, x_2^j, \dots, x_{p^j}^j\}$, and a set of outputs $Y^j = \{x_1^j, x_2^j, \dots, x_{p^j}^j\}$. Each block b_i^j has a period and

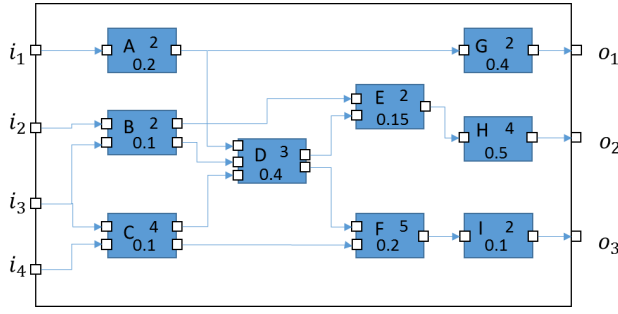


Figure 2. figure2

the worst case execution time (WCET) γ_i^j . Each block is classified as either a combinational or a sequential block. A combinational block does not have state, so its outputs depend on only its input. A sequential block has state, and it is classified as either a Moore-sequential or a non-Moore-sequential block [4]. Outputs of Moore-sequential block only depend on state, not inputs. Therefore, a combinational and non-Moore-sequential block has execution order constraint. For each b_i^j , $Pred(b_i^j)$ is defined, and is a set of directly connected with input ports of the block. Similarly, $Succ(b_i^j)$ is a set of directly connected with output ports of the block.

Figure2 shows an example of SR model with eight blocks labeled A to I, four inputs, and three outputs. Upper right numeral is period, and lower is the WCET.

2.2. Firing Timed Automaton

Time block diagrams are a kind of diagrams with triggers. Boolean signal x is called the trigger of a block A when the block A is fired by signal x . If x is trigger of block A , A can be fired only when x is true. (One block can have at most one trigger.) Time block diagrams have a trigger which is true when it becomes predetermined time. The trigger like this is called firing time specification (FTS). There are two notations for FTS. One notation is called a PPP. A PPP represents firing time with pair period and phase.(Figure) The other way is automaton. A FTA is an automaton to describe activation events that consist of union of periodic systems. In other words, a FTA is suitable to define runnable activation events which is periodic. A FTA A is a pair $A = (\theta, S)$. θ is a base period, and it is the time that it takes for transition. $S = (V, v_0, F, \delta)$. $V = v_0, v_1, \dots, v_n$ is a set of vertexes. $v_0 \in V$ is initial vertex. $F \subseteq V$ is the subset of firing vertexes. $\delta : v_{i-1} \rightarrow v_i$ is a link or transition. If $v_n \rightarrow v_0 \in \delta$, the FTA is a cycle. The cycle period of the FTA is $\Theta = \theta * (n + 1)$. Now, extending FTA, we want to capacitate for describing FTS with automaton. A runnable is unit of blocks. Therefore, a runnable has a property similar to blocks. A runnable r_k has blocks $B^k = \{b_1^k, b_2^k, \dots, b_{n_k}^k\}$, period t_k , WCET γ_k . $Pre(r_k)$ is set of directly r_k depending

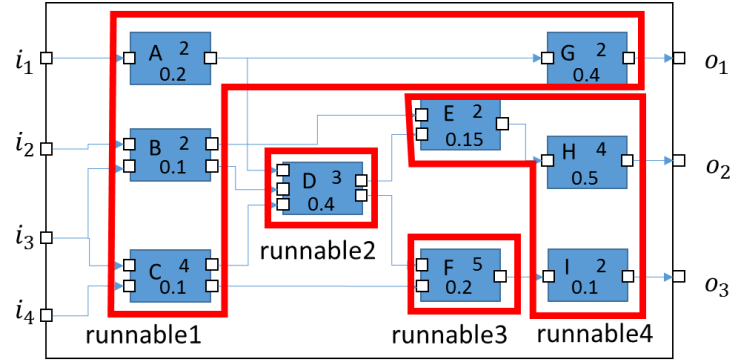


Figure 3. figure3

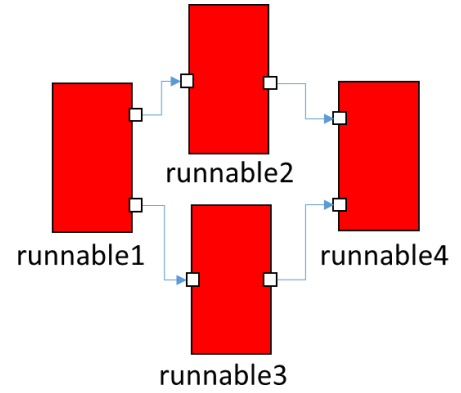


Figure 4. figure4

runnables. $Suc(r_k)$ is set of directly depended runnables by r_k .

Figure3 is the example of block mapping from SR model to runnables. Figure shows dependency of resulting runnables. Now, I would like to express the runnable1(r_1) of Figure3 in FTA. $B_1 = \{blockA, blockB, blockC, blockG\}$. In the case that base period is one, FTA is shown upper in the Figure5. By the way, greatest common divisor(GCD) of each blocks in B_1 is two. Therefore, the base period of FTA θ is changed to two. Actually, only when it is a multiple of GCD, blocks fires. When base period is one, the runnable fires all the time, but if we can change base period to GCD, the runnable firing frequency decreases, and the runnable will fire if and only if some of its blocks need to fire. Once the FTS of runnable is updated, blocks of the runnables must be updated. A period of blocks of the runnable changes period divided by new runnable period, which period of block A, B, C, G are 2/2, 2/2, 4/2, 2/2 respectively. Therefore, FTA of the runnable become below in Figure5. WCET is sum of each blocks in B_1 , and γ_{r_1} is 0.8.

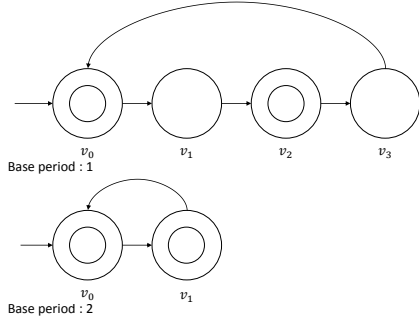


Figure 5. figure5

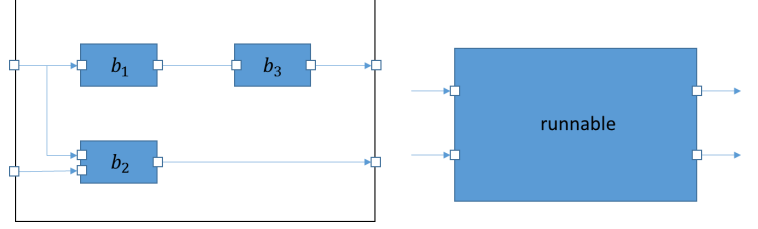


Figure 7. figure7

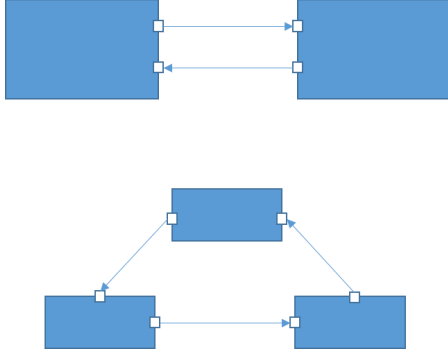


Figure 6. figure6

2.3. Cycle dependency

Cycle dependency is that exist cycle in execution order constraints. See the Figure 2. Figure4 shows an example of cycle dependency. The output of block A depends the output of block B, but the output of block B also depends on output of block A. Because of this, both outputs is not able to be calculated. When a block diagram has a cycle dependency, some blocks which compose cycle dependency are Moore-sequential, all output must be able to be calculated.

2.4. Three metrics

In mapping from blocks to runnables, there are three evaluation **criteria** in below.

2.4.1. Modularity. Modularity is measured by the number of generated runnables. The smaller the number is, the higher the degree of **modularity** is. Runnables hide the internal **information**, so when the degree of **modularity** is very high, that is, the number of runnables is considerably small compared with the number of blocks, the internal structure become difficult to grasp it.

2.4.2. Reusability. Reusability of generated runnable is measured whether it dose not have any false input-output dependencies. The component of left side of Figure2 consists of three blocks, two inputs, and two outputs. Now, let this component map one runnable. (right side of Figure2) Because runnable makes internal information black box, o_1 and o_2 depend on i_1 and i_2 from external **information**. In the other word, the link like Figure2 is refused. However, in practice o_1 does not depend i_2 , and the feedback link is safe. In this way, qualitatively when the number of runnable is as many as the number of blocks, false dependencies are difficult to **exist**. **Modularity** and Reusability are trade-off relationship.

2.4.3. Schedulability. Schedulability expresses index of whether the generated runnables are able to schedule. If generated runnables have scheduling bottlenecks, it becomes low. Scedulability is **measured** as follows. Local utilization $u_i = c_{i0}/\rho_i$ is defined for each vertex v_i in FETA of its runnable. c_{i0} is unconditional WCET of v_i , and ρ_i is defined as $\rho_i = \theta_k * d_i$. θ_k is the period of runnable γ_k , and d_i is **distance** of FETA path from v_i to the next firing vertex. R is the set of runnable for a component C . For each runnable γ_k , let the maximum local utilization be u_k^{max} , where u_k^{max} is the most among u_i in γ_k . **But**, let c_{i0} of not firing nodes be 0. The sum of maximum local utilization of each runnables is denoted as U_l . U_l is able to be computed as:

$$U_l = \sum_{r_k \in R} u_k^{max} \text{ where } u_k^{max} = \max_v(u_k^v) \quad (1)$$

The component utilization U_c is computed as:

$$U_c = \sum_{b_i \in C} \frac{C_i}{T_i} \quad (2)$$

where C_i is WCET of b_i and T_i is period of b_i . Alfa ratio $\alpha = U_l/U_c$ is defined. When we compose runnable from a component, we can compute alfa, and schedulability can be measured by alfa. There is following theorem in alfa. Theorem The alfa ratio $\alpha_u = \frac{U_l}{U_c}$ for any runnable

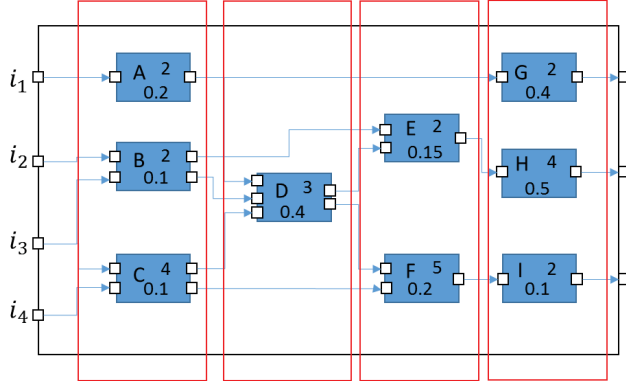


Figure 8. figure8

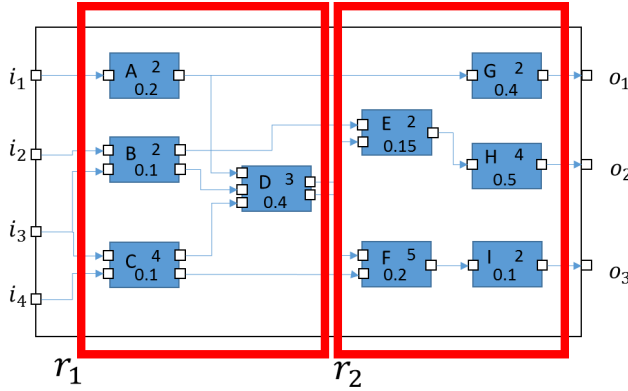


Figure 9. figure1

implementation R is greater than or equal to 1, i.e. $U_l \leq U_c$. The ratio is 1 if and only if the blocks with the same period are grouped in the same runnable.

3. TOP-DOWN METHOD

A top-down method is proposed in [1]. The top-down method is one of runnable synthesis algorithms to trade off schedulability and modularity while achieving maximum reusability and minimum code size, using FTA and alpha ratio. First step is to find a runnable generation that optimizes modularity, while achieves maximum reusability with a MILP formulation. Next step is to decompose the runnables to improve schedulability until obtaining desire alfa ratio and the number of runnables. During this process, maximum reusability and minimum code size are preserved, because decomposing runnables does not make false dependency. The initial MILP formulation is as follows. The important purpose of this step is obtaining the runnable generation with no false dependency. Maximum reusability is preserved if it is satisfied with constraints with respect to relationships among inputs and outputs. X and Y denote the set of input and output virtual block. B denotes the

set of original blocks. An input and output block must be mapped to the same runnable as the block that the input or output belongs. C_{b_i} denotes the component that b_i belongs. Similarly, C_{r_m} denotes the component that r_m belongs. g_{b_i, r_m} is a binary value, which is one if b_i is mapped to r_m , otherwise zero. A binary value h_{r_m, r_n} denotes a relationship between runnables, and r_m must be implemented before r_n when it is one. Q_{b_i, b_j} preserve a relationship between blocks and is computed by using Breath-first search (BFS). BFS is used before the MILP formulation. Z_{r_m} denotes whether there exists at least a block that is mapped to r_m , is a binary value. Therefore, $\sum_m Z_{r_m}$ is a number of runnables. (3) is the objective formulation, it optimizes modularity. (4) must be satisfied by definition of Z_{r_m} and g_{b_i, r_m} (5-8) is the constraints on mapping from blocks to runnables. The constraints (9-11) denotes definition of the causality relationship among runnables. (12) is condition not to make any false input-output dependencies.

$$\min \sum_m Z_{r_m} \quad (3)$$

$$Z_{r_m} \geq g_{b_i, r_m} \forall b_i, r_m \in B, R \quad (4)$$

$$\sum_m g_{b_i, r_m} = 1 \forall b_i, r_m \in B, R \quad (5)$$

$$g_{b_i, r_m} = 0 \forall C_{b_i} \neq C_{r_m} \quad (6)$$

$$g_{b_i, r_m} = 1 \forall b_i, r_m \in X, Y \wedge m = i \quad (7)$$

$$g_{b_i, r_m} = 0 \forall b_i, r_m \in X, Y \wedge m \neq i \quad (8)$$

$$h_{r_m, r_n} \geq h_{b_i, r_m} + g_{b_j, r_n} + Q_{b_i, b_j} - 2 \quad (9)$$

$$h_{r_m, r_n} \geq h_{r_m, r_l} + h_{r_l, r_n} - 1 \quad (10)$$

$$h_{r_m, r_n} + h_{r_n, r_m} \leq 1 \quad (11)$$

$$h_{r_m, r_n} = Q_{b_m, b_n} \forall b_m, r_m \in X \wedge b_n, r_n \in Y \quad (12)$$

However, a computational complexity of the MILP formulation is high. Therefore, by using the below method, we can get alternative solution for modularity optimization.

This alternative heuristic **acheives** solution quickly, but this solution is not optimal. Therefore, if you want to get optimal **modularity** solution of **smoal** systems, you should use the MILP formulation.

Next, Algorithm 2 decomposes obtained runnable generation to improve schedulability. The algorithm finds runnable r_m that has most maximum local utilization among runnables, and decomposes r_m to r_n . The algorithm sorts blocks of r_m in period descending order. It tries to decompose each block in order, and if there are no cycle dependencies, the block is decomposed, and mapped to r_n . Once it finds block that can be mapped to r_n , a variable found becomes true, and period T_n of r_n become the period of the block mapped. If found is true, candidate blocks mapped to r_n are blocks that has the same period as T_n . At the end, alpha ratio is computed, and N is added 1 to. This continues until obtaining desire alpha ratio or target \hat{N} .

This sequential of method has some problems. First, we discuss the heuristic **algorizm** that make the initial set of runnables. **Maiking** the set of runnables, the heuristic must satisfy constraints that the set of runnables has no false

Algorithm 1 modularity optimization solution to fill in for the MILP formulation

```

1: Let each block  $b_i$  in  $S$  be initially mapped to each task  $\gamma_i$ .
2: Let  $\Gamma = \gamma_i$ , and change = true.
3: while change do
4:   while  $\exists b_i \in G_k, b_i \in G_k, s.t. Pred(b_j)$  and there exists no input directly connected to  $b_j$  do
5:     if schedulable_after_merge( $b_i, b_j$ ) then
6:       merge_blocks( $b_i, b_j$ ); merge_task( $\gamma_i, \gamma_j$ ).
7:     end if
8:   end while
9:   while  $\exists b_i \in G_k, b_i \in G_k, s.t. Succ(b_j)$  and there exists no output directly connected to  $b_i$  do
10:    if schedulable_after_merge( $b_i, b_j$ ) then
11:      merge_blocks( $b_i, b_j$ ); merge_task( $\gamma_i, \gamma_j$ ).
12:    end if
13:   end while
14:   change = false
15:   if  $\exists b_i \in G_k, b_i \in G_k, s.t. \max\_reusability\_after\_merge(b_i, b_j)$  and schedulable_after_merge( $b_i, b_j$ ) then
16:     merge_blocks( $b_i, b_j$ ); merge_task( $\gamma_i, \gamma_j$ ); change = true.
17:   end if
18: end while
19: return  $\Gamma$ 

```

dependency and do not introduces false dependency as a result of *decompose()*. Because of this, the heuristic executes many runnables, that is, it is low modularity.

Next, we examine the top-down method. This method uses the set of runnables gotten by the heuristic, and finds the solution with *decompose()*. *decompose()* can decompose a runnable if the the set of runnables including the result runnable does not have cycle dependency. The decomposed runnable is determined according to the maximum local utilization, and the decomposed block is selected depending on maximum vertex utilization. At this time, if the block is not seccessed at be decomposed due to cycle dependency, more effective decomposing can not be implemented. If the block is not able to be decomposed in succession, total utilizatin α is not much improved. This becomes the bottle neck, the number of runnables increases as result.

4. EXTENDED MAPPING METHOD

For resolving the above problems, this section proposes the extended mapping algorizm. First step, the method obtains an appropriate runnables

5. RERATED WORK

This section discusses related work with respect to a model based development.

In multitask software implemeentation of SBDs, blocks to task mapping method is commonly proposed. In [2], as

Algorithm 2 Top-down method

```

1: In First step, we obtain a runnable generation p that is optimal modularity, while it introduces no false dependencies, using the MILP formulation.
2: In Second step, we improve schedulability as below until we obtain desire alfa ratio and target number of runnables.
3:  $\alpha_u = \text{ComputeAlpha}()$ 
4: while  $\alpha_u > \alpha'_u \vee N < \hat{N}$  do
5:   Let  $r_m$  be the runnable with maximum  $u_m^{max}$  among those that have different period blocks.
6:   For each  $b_i \in r_m$ , order  $b_i$  based on descending  $T_{b_i}$ 
7:    $r_n = \phi$ , found = false,  $T_n = 0$ 
8:   for  $b_i \in r_m$  in order do
9:     if found = false then
10:       $r_n = r_n \cup b_i$ 
11:       $p_t = \text{decompose}(r_m, r_n)$ 
12:      if existsCycle( $p_t$ ) then
13:         $r_n = r_n - b_i$ 
14:      else
15:         $p_b = p_t$ , found = true,  $T_n = T_{b_i}$ 
16:      end if
17:    else
18:      if  $T_{b_i} = T_n$  then
19:         $r_n = r_n \cup b_i$ 
20:         $p_t = \text{decompose}(r_m, r_n)$ 
21:      else
22:         $p_b = p_t$ 
23:      end if
24:    end if
25:  end for
26:   $p = p_b, \alpha_u = \text{ComputeAlpha}(), N = N + 1$ 
27: end while

```

evaluation objects, timing metrics is proposed, in addition to modularity, reusability, and code size. Moreover, a new mapping method (TRCM and MRCT) is proposed, considering optimal timing metrics and modularity, while achieving maximum reusability and minimum code size.

Automatical code generation is focused on. In notation FTS of blocks in SBD, all blocks are fired all the time in previous. [3] discusses triggered and timed block diagrams, and propose FTA that is useful for describing firing time of a unit of snchronous system. Due to FTA, it become possible to express derails of firing time, block should fire only in designated time.

Block diagrams are a usuful graphical notation. It is used in commercial products such as Simulink and SCADE [4]. With SAT solver [4], code generation that optimizes modularity while maintaining maximum reusability and minimum code size is proposed.

Robotic middlewares, such as Orocos-RTT [], is platform that executes component-based applications and abstract functionality, but are difficult to be integrated in a model-based flow with automatic code generation. Control applications are designed by synchronous models (Simulink), but the code generated is executed only by a

single core.

6. test

As shown in [1]. As shown in [2] As shown in [3]
As shown in [4] As shown in [5]

References

- [1] Peng Deng, Fabio Cremona, Qi Zhu, Marco Di Natale, and Haibo Zeng. A model-based synthesis flow for automotive cps. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems, ICCPS '15*, pages 198–207, New York, NY, USA, 2015. ACM.
- [2] Peng Deng, Qi Zhu, M. Di Natale, and Haibo Zeng. Task synthesis for latency-sensitive synchronous block diagram. In *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*, pages 112–121, June 2014.
- [3] R. Lubliner and S. Tripakis. Modular code generation from triggered and timed block diagrams. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE*, pages 147–158, April 2008.
- [4] Roberto Lubliner, Christian Szegedy, and Stavros Tripakis. Modular code generation from synchronous block diagrams: Modularity vs. code size. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '09*, pages 78–89, New York, NY, USA, 2009. ACM.
- [5] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

