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, developers can describe AUTOSAR software components. Then, developers 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

These characteristics make low-cost and speedy development 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 as shown Figure 1. The application layer has several software components (SWCs). Each SWC implements a function of the Engine Control Unit (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 layer uses the virtual functional bus (VFB), and realizes the communication between SWCs and between SWCs and the basic software modules. The RTE layer provides API for communication between SWCs and execution control. When we develop SWCs, we do not have to accommodate lower 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. These structured layers make reusability of basic software advance.

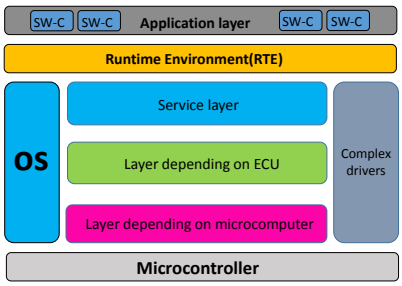

Figure 1. Architecture of AUTOSAR

Contribution

Organization: The rest of this paper is organized as follows. Section 2 provides a fundamental knowledge for mapping method from block diagrams to runnables. Section 3 gives an existing mapping algorithm and problem of it. Section 4 proposes an extended method algorithm. Section 5 measures and prepare with existing method and new one. Section 6 discusses related work. In the end, Section 7 concludes this paper.

2. A FUNDAMENTAL KNOWLEDGE

2.1. SR model

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

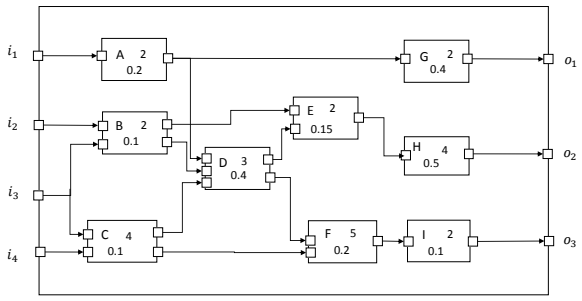

Figure 2. Example of a block diagram

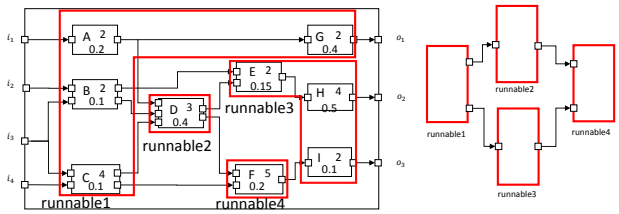

Figure 3. generated runnables from Figure 2

$Y^j = \{{x_1}^j, {x_2}^j, ..., {x_{p^j}}^j\}$. Each block ${b_i}^j$ has a period and the worst-case execution time (WCET) ${\gamma_i}^j$. Each block is classified as either a combinational or a sequential block. Its outputs depend on only its input because a combinational block does not have state. A sequential block has state, and it is classified as either a Moore-sequential or a non-Moore-sequential block [5]. 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.

Figure 2 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

Timed block diagrams are a kind of diagrams with triggers [5]. Boolean signal x is called the trigger of a block A when the block A is fired by signal x. If x is the trigger of block A, block 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 such as 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. An 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. An FTA A is a pair $A = (\theta, S)$. θ is a base period, and it is the time that it takes for transition. $S = (V, v_0, F, \delta)$. $V = v_0, v_1, ..., v_n$ 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 an FTA, we want to capacitate for describing an FTS with an automaton. A runnable is unit of blocks.

Therefore, a runnable has a property similar to blocks. A runnable r_k has blocks $B^k = \{b_1^k, b_2^k, ..., b_{n^k}^k\}$, period

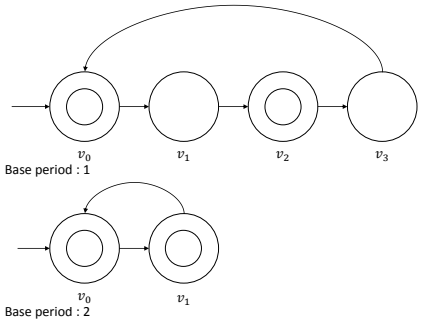

Figure 4. An FTA sample

t_k, WCET γ_k. $Pre(r_k)$ is a set of directly r_k depending runnables. $Suc(r_k)$ is a set of directly depended runnables by r_k.

The left of Figure 3 is the example of block mapping from SR model to runnables. The right of Figure 3 shows dependency of resulting runnables. Now, I would like to express the runnable1(r_1) of Figure 3 in an FTA. $B_1 = \{blockA, blockB, blockC, blockG\}$. In the case that base period is one, the FTA is shown upper in the Figure 4. By the way, the greatest common divisor (GCD) of each blocks in B_1 is two. The base period of FTA θ is able to be 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 several 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 the period divided by new runnable period, which periods of block A, B, C, G are 2/2, 2/2, 4/2, 2/2 respectively. In the example of the lower Figure 4, block A, B, C, and G are fired at $vertex_0$, and block A, B, and G are fired $vertex_1$. The WCET is defined for each vertex, which it becomes the sum of WCETs of blocks fired at the vertex, that is, the WCET of v_0 and v_1 is 0.8, 0.7 respectively.

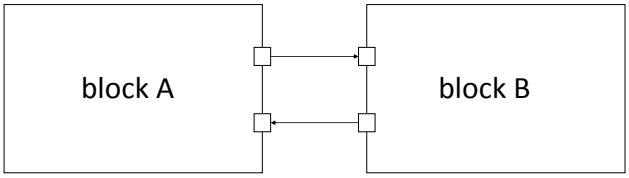

Figure 5. Example of cycle dependency

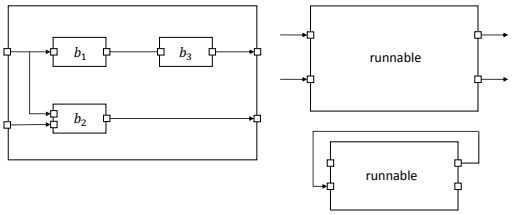

Figure 6. Result of runnable generation with false dependency

2.3. Cycle dependency

Cycle dependency is that exist cycle in execution order constraints. 5 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 are not able to be calculated. When a block diagram has a cycle dependency, several 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. 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 becomes difficult to grasp it.

2.4.2. Reusability. Reusability of generated runnable is measured whether it does not have any false input-output dependency. The component of left side of Figure 6 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 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 vertexes v_i in FETA [5] 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. Note that c_{i0} of not firing nodes is 0. The sum of maximum local utilization of each runnable is denoted as U_l. U_l is able to be computed as:

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

The component utilization U_c is computed as:

$$U_c = \sum_{b_i \in C} \frac{C_i}{T_i} \tag{2}$$

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, the Alfa ratio is able to be computed, and schedulability can be measured by Alfa ratio. There is following theorem in Alfa.

Theorem The Alfa ratio $\alpha_u = \frac{U_l}{U_c}$ for any runnable 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 [2]. 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 achieving maximum reusability with a MILP formulation [3]. Next step is to decompose the runnables to improve schedulability until obtaining desire

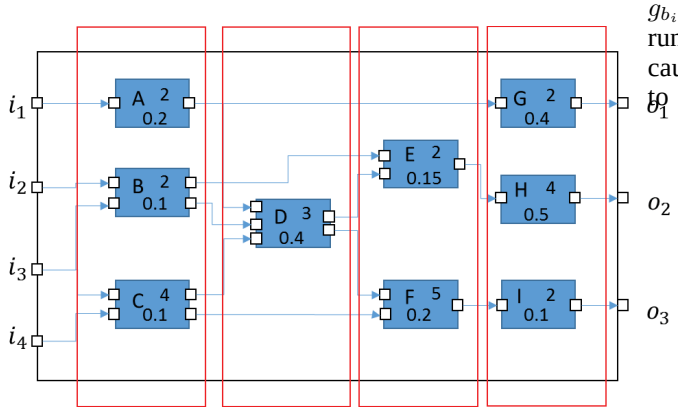

Figure 7. figure8

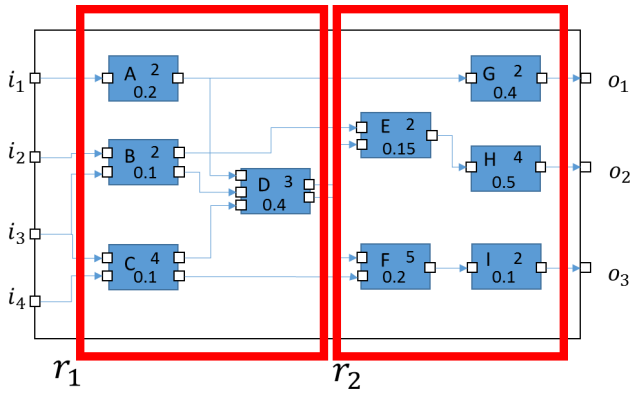

Figure 8. figure1

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 to. C_{b_i} denotes the component that b_i belongs to. Similarly, C_{r_m} denotes the component that r_m belongs to. 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} preserves 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 binary value. Therefore, $\sum_m Z_{r_m}$ is the 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) denote definitions of the causality relationship among runnables. (12) is condition not to make any false input-output dependency.

$$min \sum_m Z_{r_m} \tag{3}$$

$$Z_{r_m} \geq g_{b_i,r_m} \forall b_i, r_m \in B, R \tag{4}$$

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

$$g_{b_i,r_m} = 0 \forall C_{b_i} \neq C_{r_m} \tag{6}$$

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

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

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

$$h_{r_m,r_n} \geq h_{r_m,r_l} + h_{r_l,r_n} - 1 \tag{10}$$

$$h_{r_m,r_n} + h_{r_n,r_m} \leq 1 \tag{11}$$

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

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

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

Let each block b_i in S be initially mapped to each task γ_i.
Let $\Gamma = \gamma_i$, and change = true.
while change **do**
  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**
    if schedulable_after_merge(b_i, b_j) **then**
      merge_blocks(b_i,b_j); merge_task(γ_i, γ_j).
    end if
  end while
  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**
    if schedulable_after_merge(b_i, b_j) **then**
      merge_blocks(b_i, b_j); merge_task(γ_i, γ_j).
    end if
  end while
  change = false
  if $\exists b_i \in G_k, b_i \in G_k, s.t.$ max_reusability_after_merge(b_i, b_j) $\wedge$ schedulable_after_merge(b_i, b_j) **then**
    merge_blocks(b_i, b_j); merge_task(γ_i, γ_j); change = true.
  end if
end while
return Γ

This alternative heuristic achieves solution quickly, however this solution is not optimal. Therefore, if you want to get optimal modularity solution of small 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 $\acute{N}$.

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: α_u = ComputeAlpha()
4: while α_u > ά_u ∨ N < Ń 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 ∈ r_m, order b_i based on descending T_{b_i}
7:    r_n = φ, found = false, T_n = 0
8:    for b_i ∈ r_m in order do
9:       if found = false then
10:         r_n = r_n ∪ b_i
11:         p_t = 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 ∪ b_i
20:            p_t = 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, α_u = ComputeAlpha(), N = N + 1
27: end while
```

This sequential of method has several problems. First, we discuss the heuristic algorithm that make the initial set of runnables. Making the set of runnables, the heuristic must satisfy constraints that the set of runnables has no false dependency and does not introduce 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 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 utilization 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 algorithm. As above, it is a problem that the number of initial runnables is large. First step, the method obtains an appropriate runnables, which the number of it is the same to the number of outputs of its component. Note that each block is mapped only one runnable for minimizing code size. Due to this initial mapping, the number of runnables will decrease compared with existing method, however the generated runnables may introduce false dependency. Accordingly, extended *decompose()* can decompose a runnable if the generated set of runnables do not have false dependency.

The algorithm is as below.

5. RERATED WORK

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

In multitask software implementation of SBDs, blocks to task mapping method is commonly proposed. In [3], as 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 an SBD, all blocks are fired all the time in previous. [4] discusses triggered and timed block diagrams, and proposes FTA that is useful for describing firing time of a unit of synchronous system. Due to FTA, it become possible to express derails of firing time, block should fire only in designated time.

A block diagram is a useful graphical notation. It is used in commercial products such as Simulink and SCADE [5]. With SAT solver [5], 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 they 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. CONCLUTION

7. test

As shown in [2]. As shown in [3] As shown in [4] As shown in [5] As shown in [6]

References

[1] AUTOSAR. http://www.autosar.org/.

[2] 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.

[3] 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.

[4] R. Lublinerman 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.

[5] Roberto Lublinerman, 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.

[6] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.