Performance evaluation of a multi-core system using Systems development method utilizing Reverse modeling and Model-based Simulation

14th International Forum on Embedded MPSoC and Multicore
July 7 - 11, 2014

Yoshifumi Sakamoto, Ph.D. , PMP
Digital Front Office - Industrial Services, Global Business Services
IBM Japan, Ltd.,

OUTLINE

- Introduction
- Proposed Method
- Subject to be solved
- Objectives
- Modeling
- Simulation
- Conclusions

Introduction

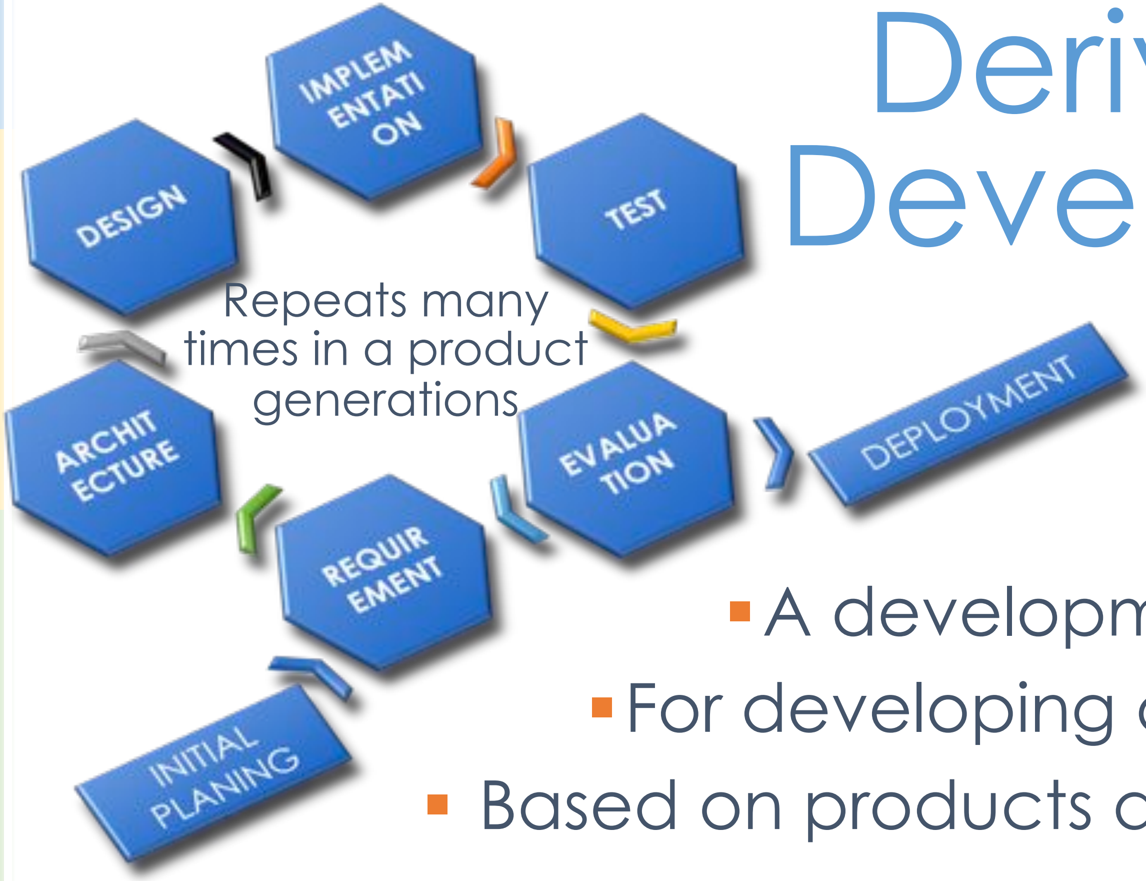
Embedded systems

Play an important role in configuring and maintaining Social Infrastructure

Available for groups of 3 or 4 travelling to or from Tsing Yi or Kowloon Station

Specially,
Ensuring **dependability** is highly prioritized issues

Introduction



Repeats many times in a product generations

What is Derivational Development?

- A development method
- For developing derived products
- Based on products already been released

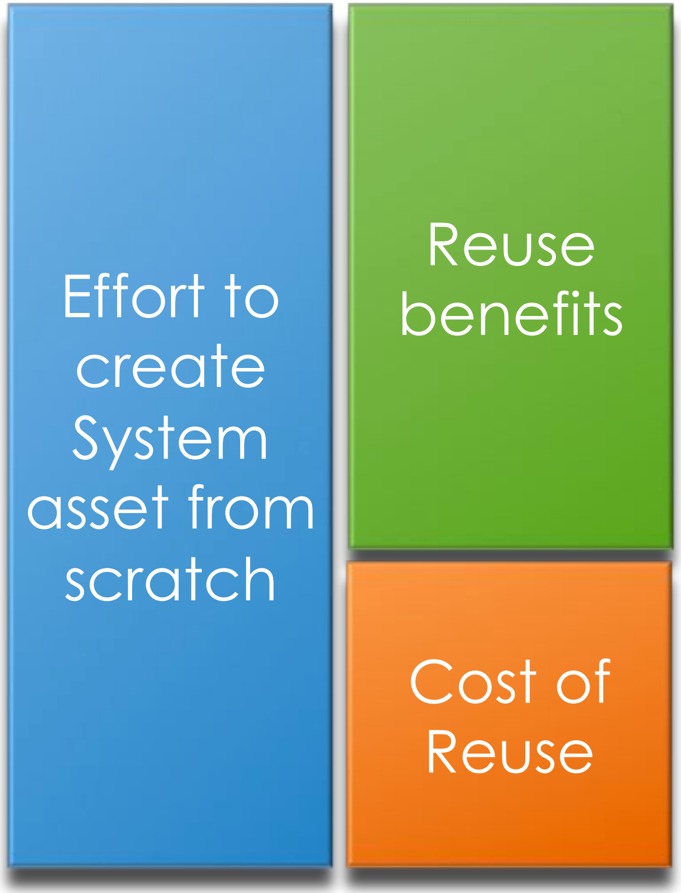
Introduction

- Why many embedded system development organizations applied derivational development?

To improve

- Development Efficiency
- Time-to-market
- Product Quality

Introduction



Effort to create System asset from scratch

Reuse benefits

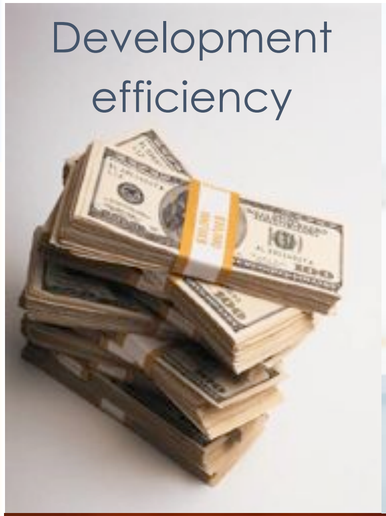
Cost of Reuse

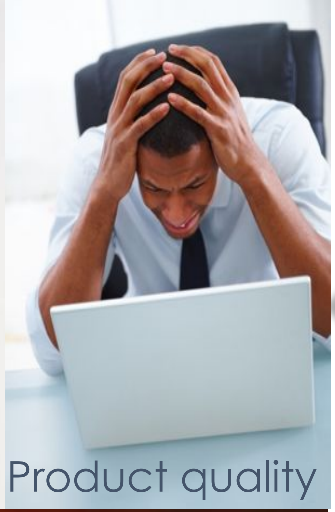
What is **Biggest advantage** of applying derivational development?

Reuse of existing design assets

The smaller the Reuse cost, the more the benefits for reuse

Problems to be solved





Derivational development was applied, **but**

Could **NOT** improved

what's happened?

- Development scale is Large?
- Systems become complicated?

Investigations were **only few cases**

Target - current situations Survey

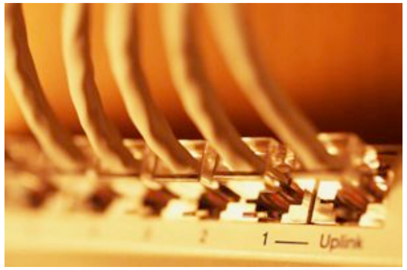
20 teams

6 campanies

- Multi Function Printer/Peripheral

7 teams

- Network equipment



1 team

- Factory Automation

2 teams

- Automotive

10 teams

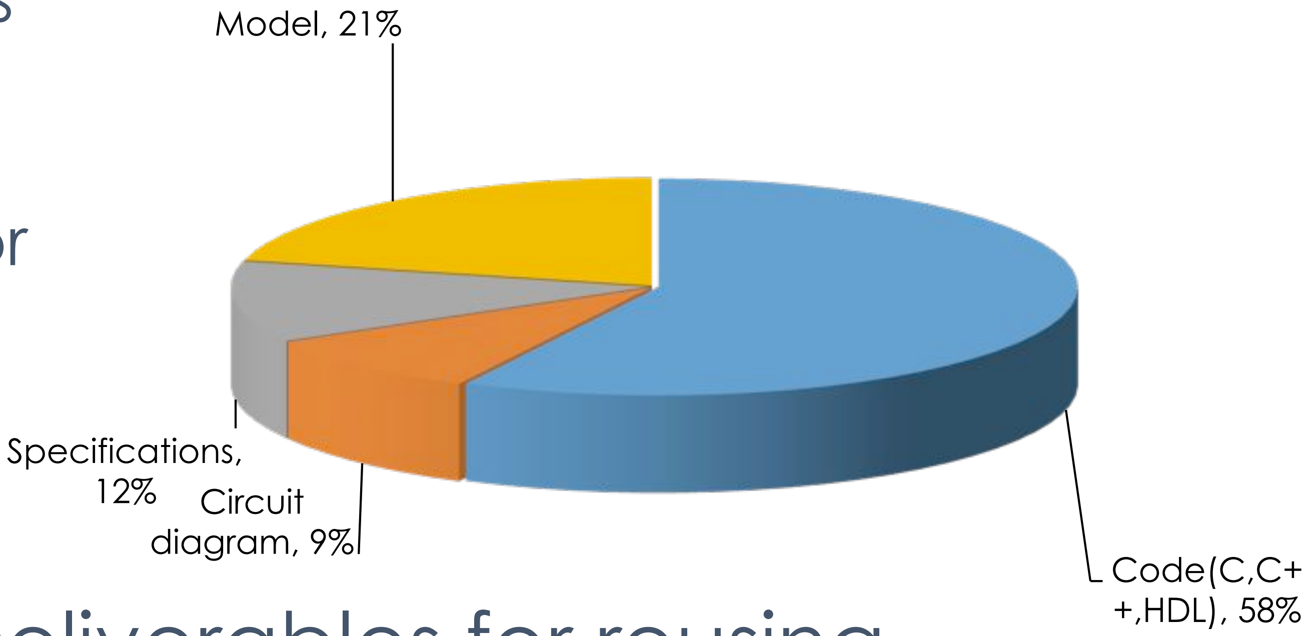
All **20** teams are applying derivational development

Survey result

- Reuse rate for specifications was only **12%**
- Reuse rate for codes and models were much higher than for specifications

rarely used

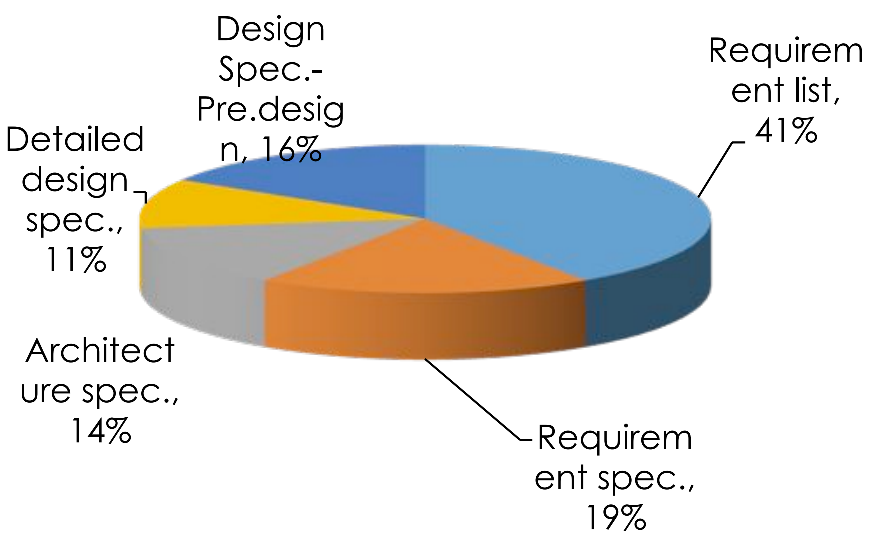
To create codes or models after creating specifications



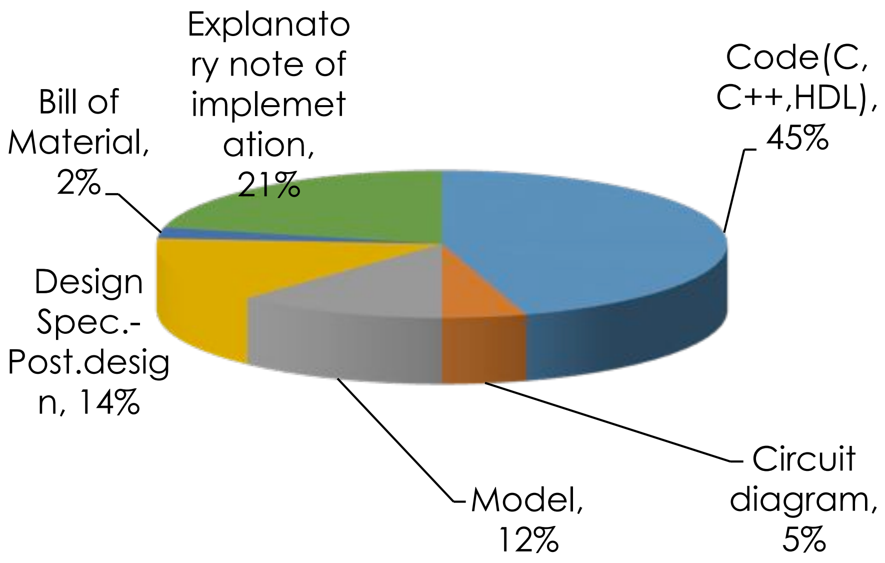
Deliverables for reusing

Survey result

- **Requirements are used as an input** for architectural design, detailed design and implementation
- **Creating Codes & Models first,** then create documents



Input to design activity

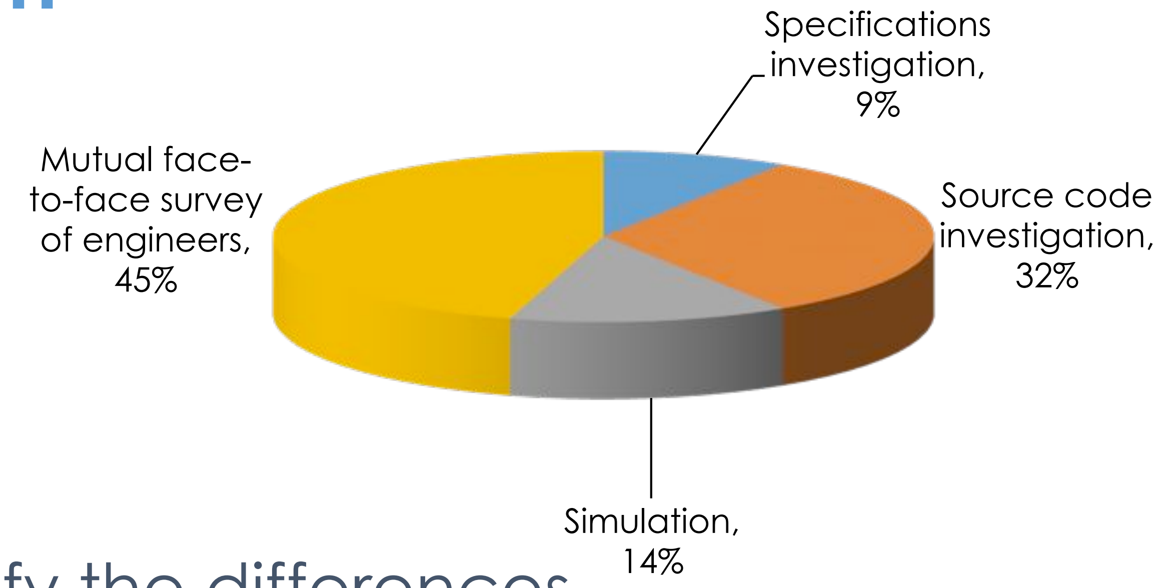


Output from design activity

Survey result

- To specify design addition or modifications, the most popular method is **Face to Face communication**

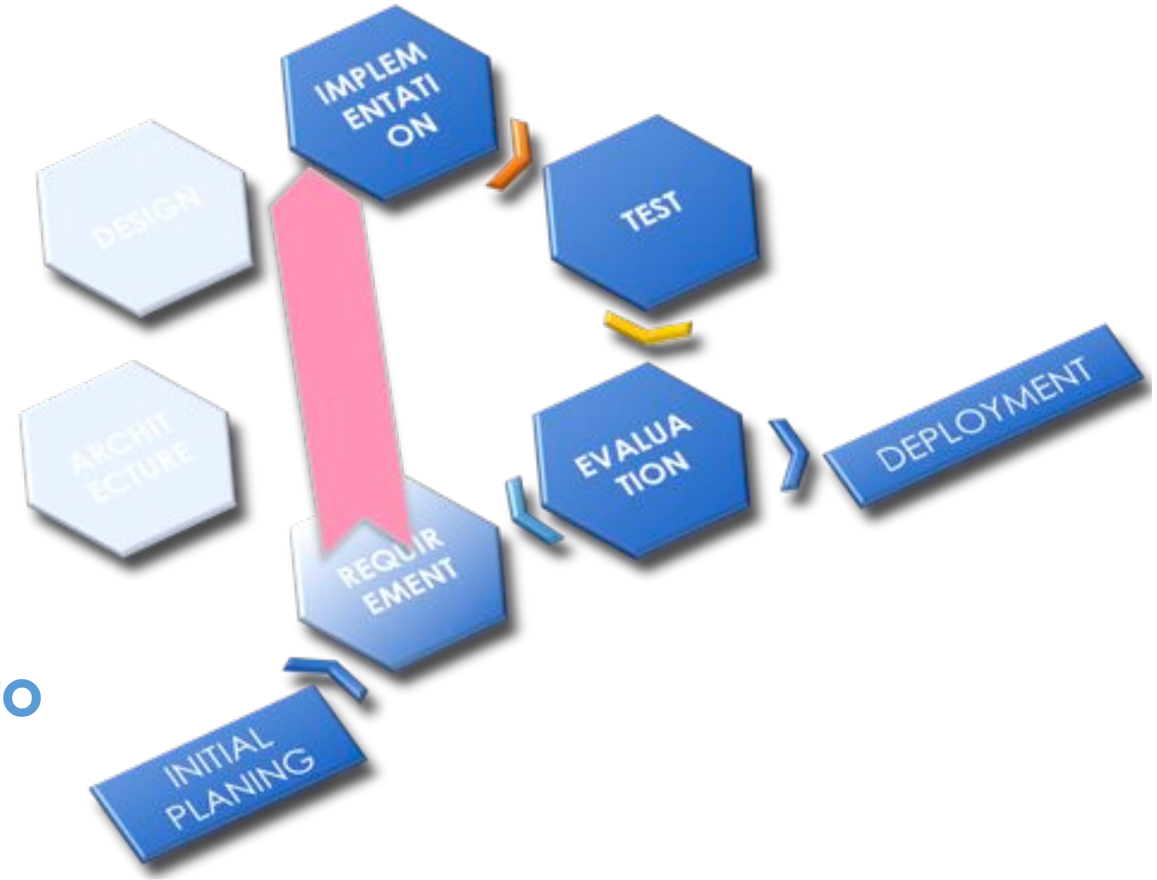
Specifications **aren't** popularly used, also



How to identify the differences

Issue measurement - what's happened?

- Architecture design and System design activity are **skipped**
- **Requirements** used as an input for **detailed design** and **implementation**
- Main development deliverable **is limited to source code**



Issue measurement - Examine

Why derivational development tends to take place in such a manner?

- Documentation is **eliminated** in order **to reduce development cost** and **to achieve development schedule**
- **Customers don't require** documents, because source code delivered and Integration activity need only them
- **There isn't a useful process** that suits derivational development

Objectives

- **Resolve** Specified unique issues of derivational development

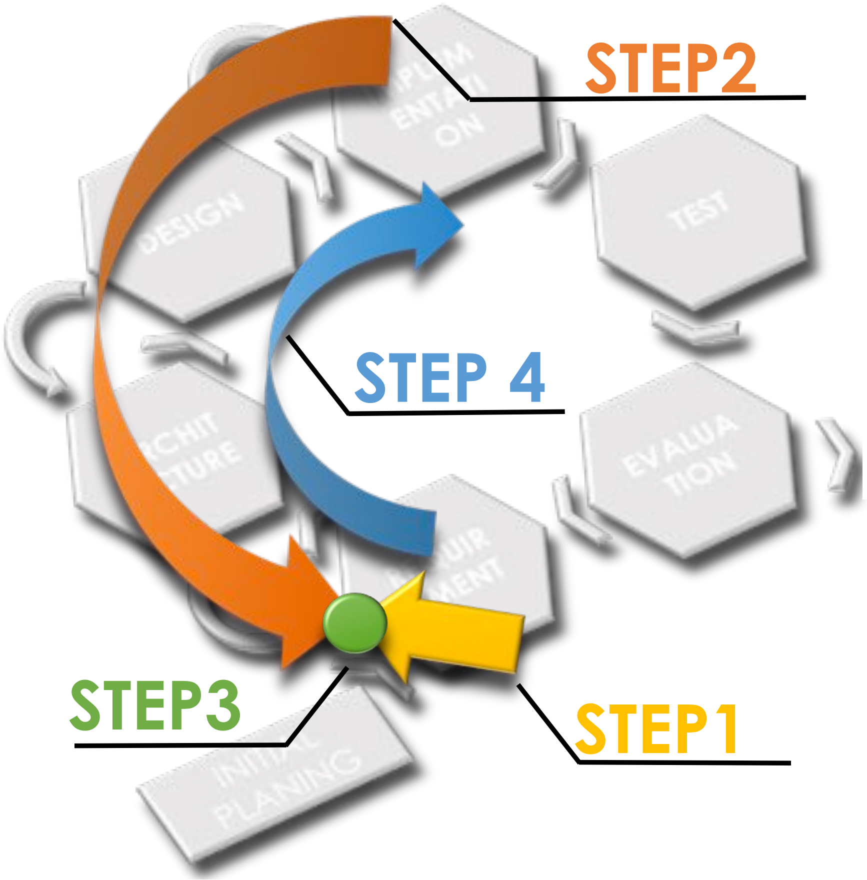
- **Propose** a methodology to solve issues of derivational development

What is SRMS ?

Systems development method utilizing
Reverse modeling and
Model-based
Simulation

Method to verify the design quality of the embedded system in upper reaches of process of the system development.

SRMS - Proposed Method



STEP1- Requirement Definition

Clarify the functions, users of the system, system operations and boundaries

STEP2- Reverse Modeling

Create system configurations, logical sequence, parameters , constraints and preconditions

STEP3- Model-based Simulation

Model-based performance evaluation

STEP4- New System Development

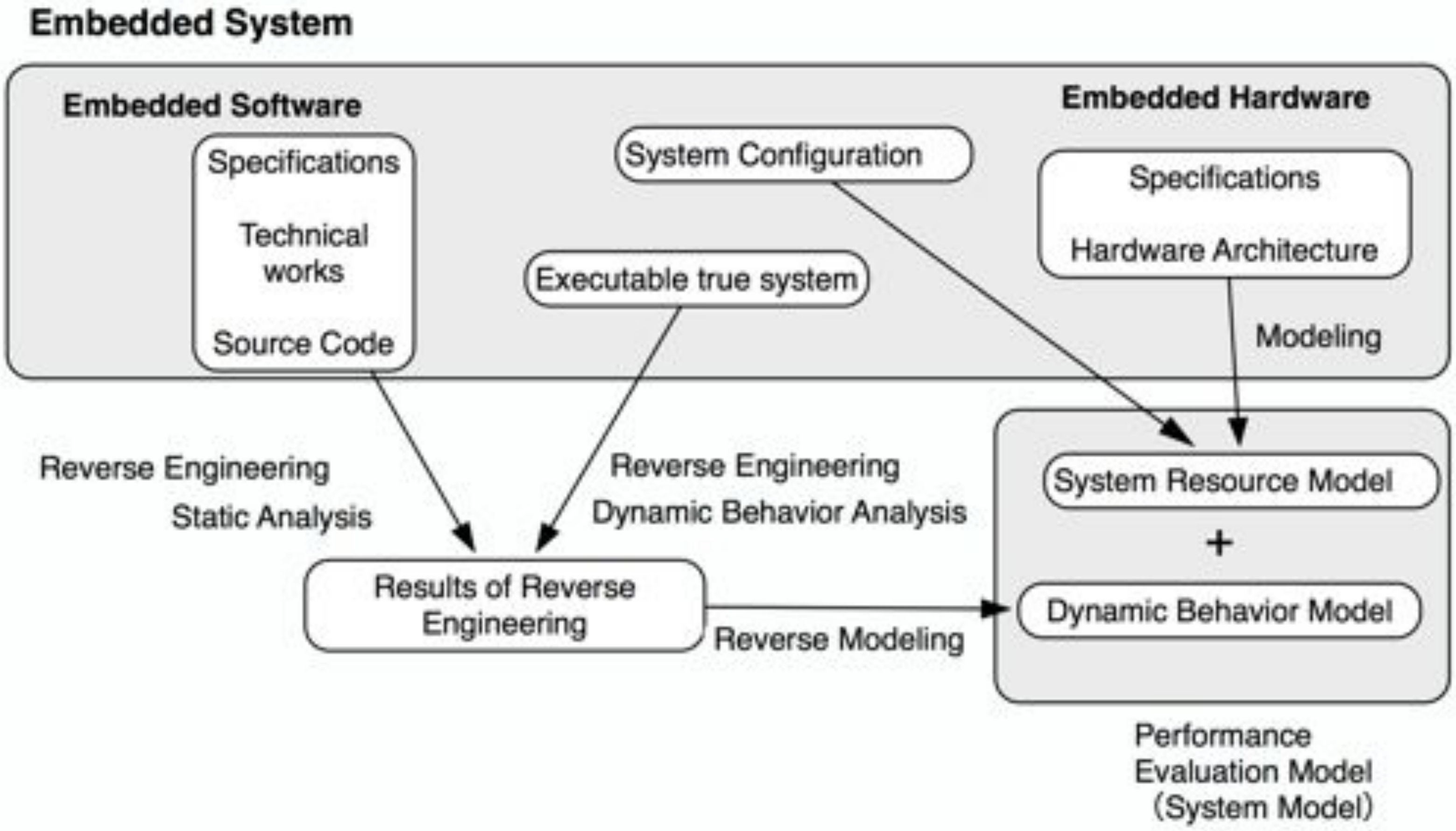
Gradual development from upper to lower activities of the process

Performance Evaluation for SoC

- Appropriate to estimate the performance of the SoC **in the early stages** of SoC development.

- **Higher accuracy** than conventional estimation methods.

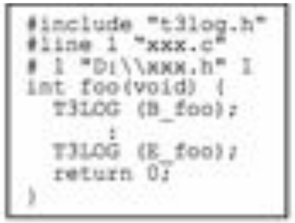
SRMS - Outline of the Reverse Modeling



Observation technology that I used for dynamic behavior analysis

Low invasiveness

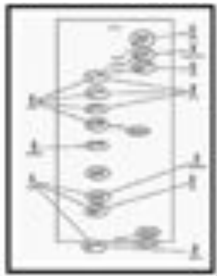
Prove Code



```
#include "t3log.h"
#line 1 "xxx.c"
# 1 "D:\\xxx.h" 1
int foo(void) {
  T3LOG (B_foo);
      :
  T3LOG (E_foo);
  return 0;
}
```

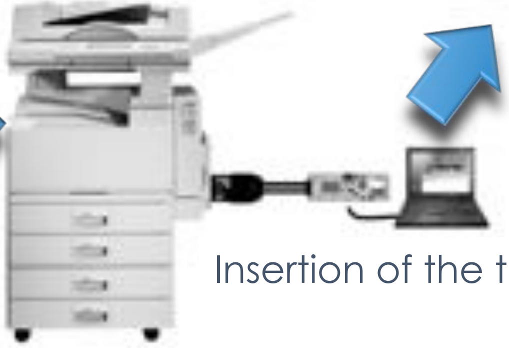
Prove Code embedding

Test Scenario



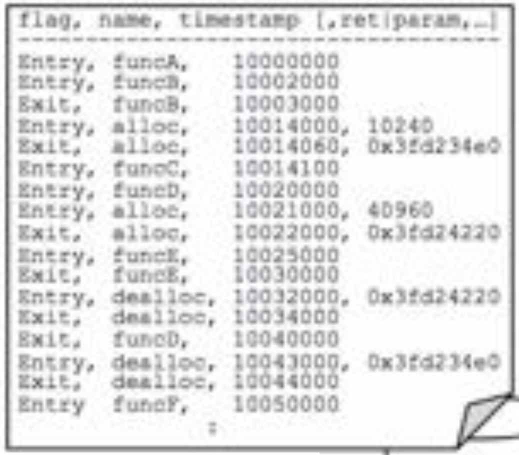
Source Code

Executable true System



Insertion of the time-stamp

Execution Trace Data



```
flag, name, timestamp [,ret|param,…]
--------------------------------------------
Entry, funcA,    10000000
Entry, funcB,    10002000
Exit,  funcB,    10003000
Entry, alloc,    10014000, 10240
Exit,  alloc,    10014060, 0x3fd234e0
Entry, funcC,    10014100
Entry, funcD,    10020000
Entry, alloc,    10021000, 40960
Exit,  alloc,    10022000, 0x3fd24220
Entry, funcE,    10025000
Exit,  funcE,    10030000
Entry, dealloc, 10032000, 0x3fd24220
Exit,  dealloc, 10034000
Exit,  funcD,    10040000
Entry, dealloc, 10043000, 0x3fd234e0
Exit,  dealloc, 10044000
Entry  funcF,    10050000
                 :
```

Visualizer

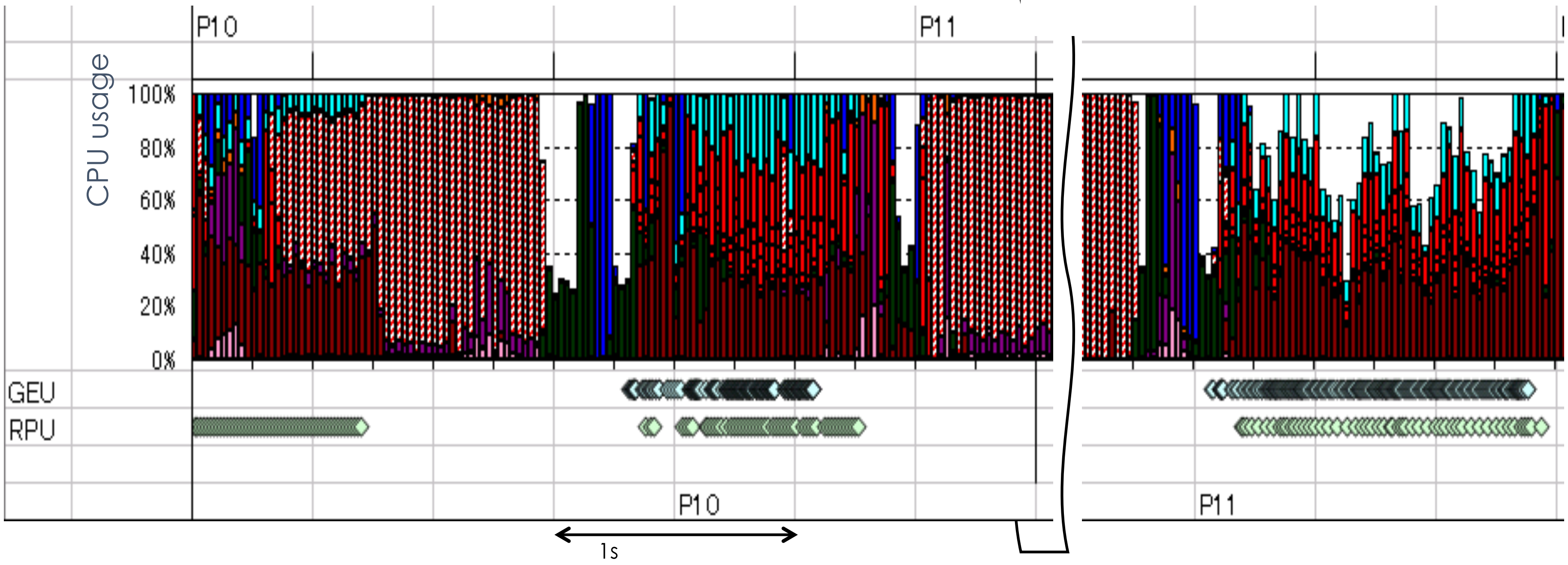
Dynamic Behavior Analysis

- Observe a **Dynamic behavior** of the embedded system and to collect the **execution trace data**.
- The execution trace data includes :
  the **timestamps** for function calls and returns, **processor usage** per unit time, and the **sizes of the data transfers** via the buses.

Observation Environment

Sample of the Execution Trace Data

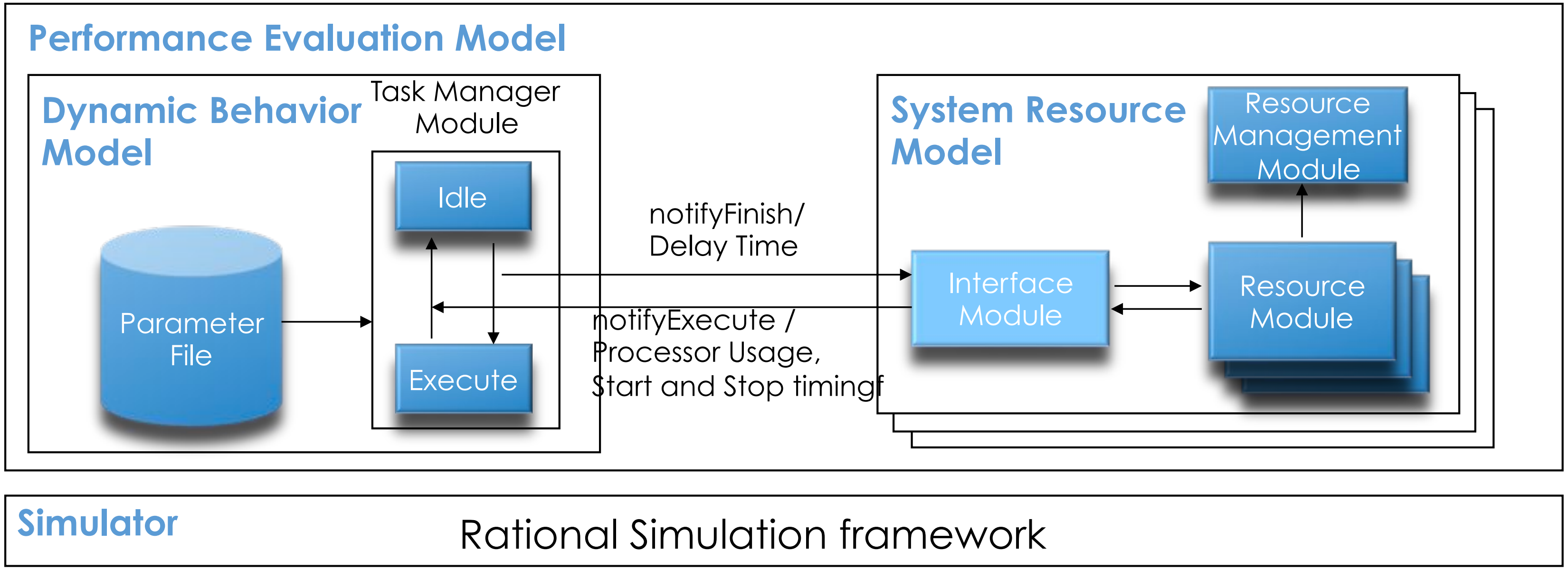
- CPU share graph



Each color represents a different task

Constitution of the performance evaluation model

- Change easiness of the model - analyze performance for **some architecture candidates**
- Adopt **the structure that isolated** a Dynamic Behavior Model and System Resource Model
- Execution trace data are stored away by a Parameter File



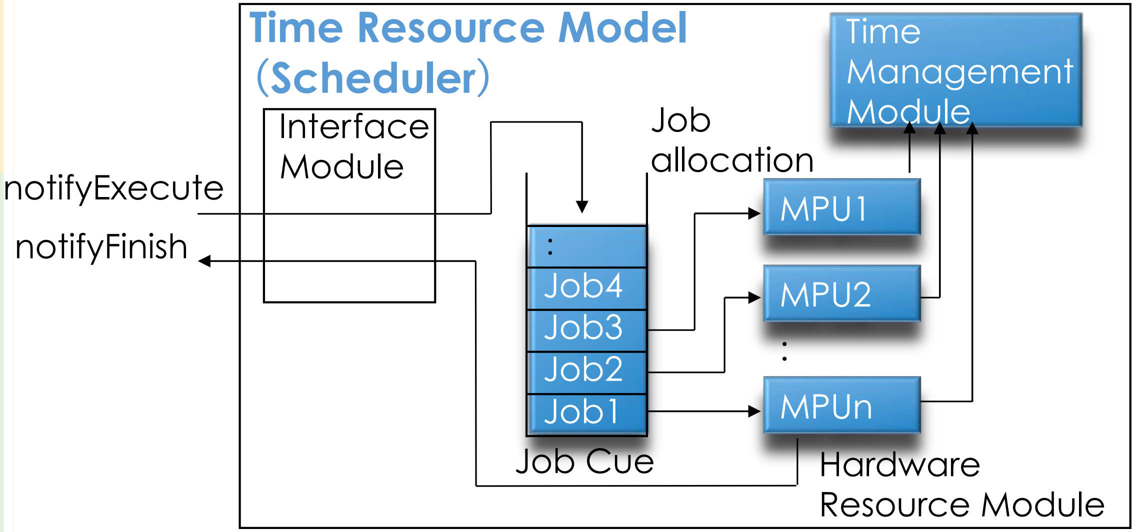
Variation of the System Resource Model

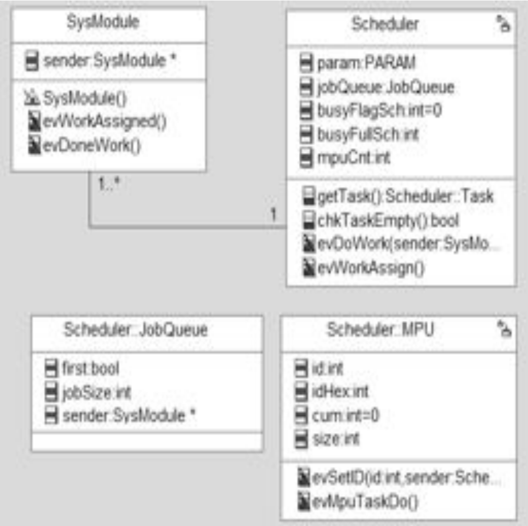
Time Resource Model

- The model describes progress of the **time**
- Job allocation

  AMP: Job cue allocates a peculiar job to a Hardware Resource Module **statically**

  SMP: Job cue allocates a job to a Hardware Resource Module **dynamically**

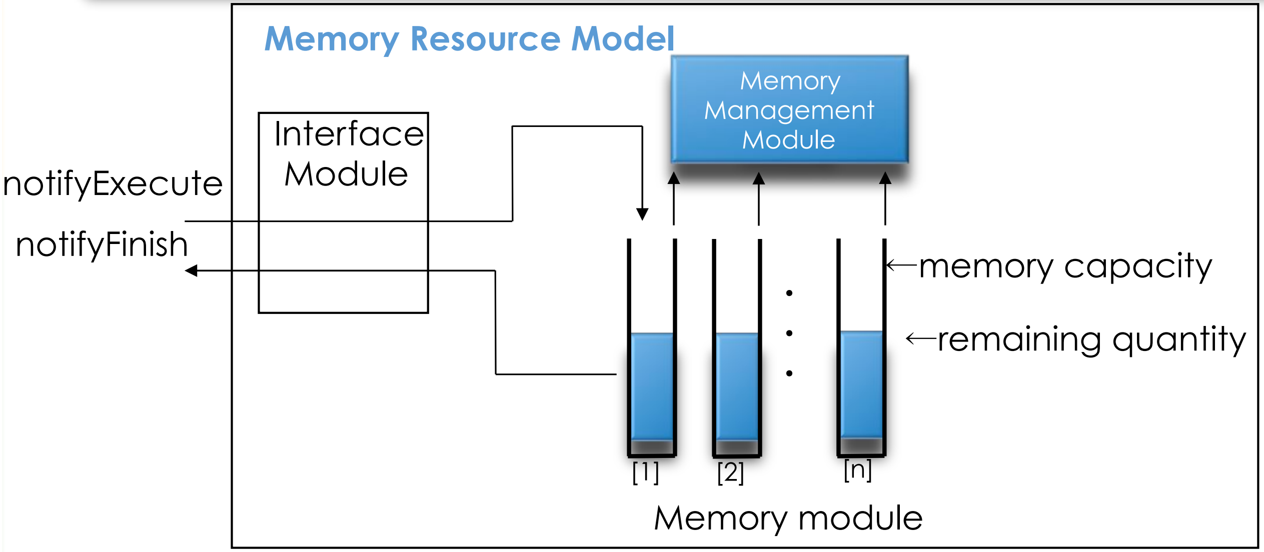


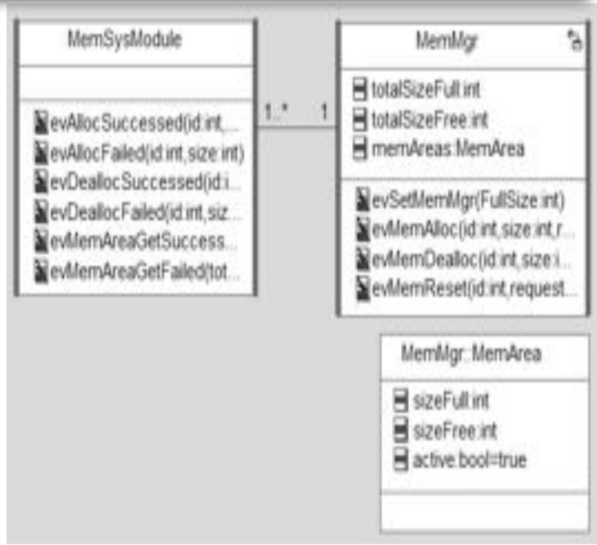


Variation of the System Resource Model

Memory Resource Model

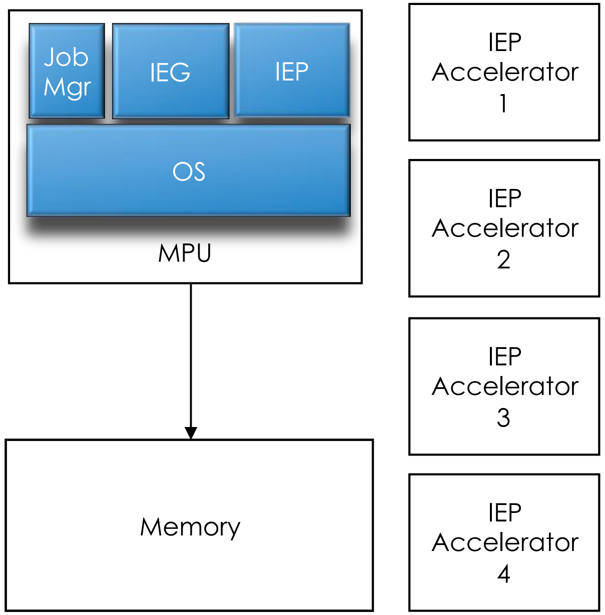
- This model expresses the consumption of the memory resource to assign to each processing.
- The dynamic behavior model shifts to the next processing step depending on a notice of the success or failure.



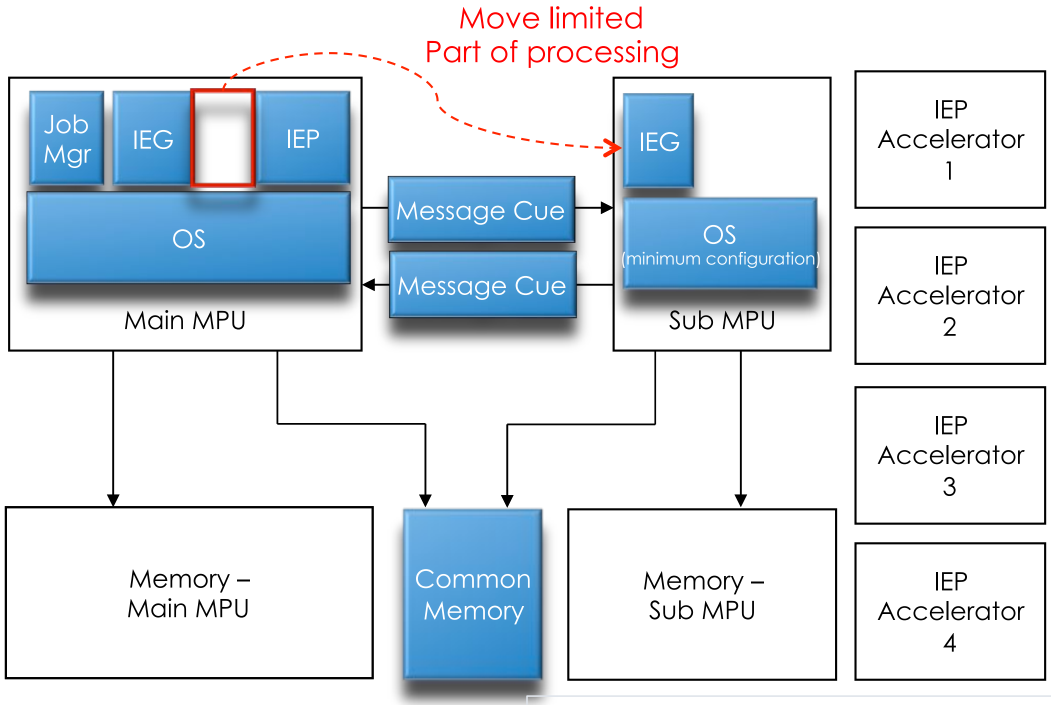

MemSysModule
evAllocSuccessed(id:int,...
evAllocFailed(id:int,size:int)
evDeallocSuccessed(id:i...
evDeallocFailed(id:int,siz...
evMemAreaGetSuccess...
evMemAreaGetFailed(tot...
1..*
1
MemMgr
totalSizeFull:int
totalSizeFree:int
memAreas:MemArea
evSetMemMgr(FullSize:int)
evMemAlloc(id:int,size:int,r...
evMemDealloc(id:int,size:i...
evMemReset(id:int,request...
MemMgr::MemArea
sizeFull:int
sizeFree:int
active:bool=true

System Architectures

Single-Core Architecture



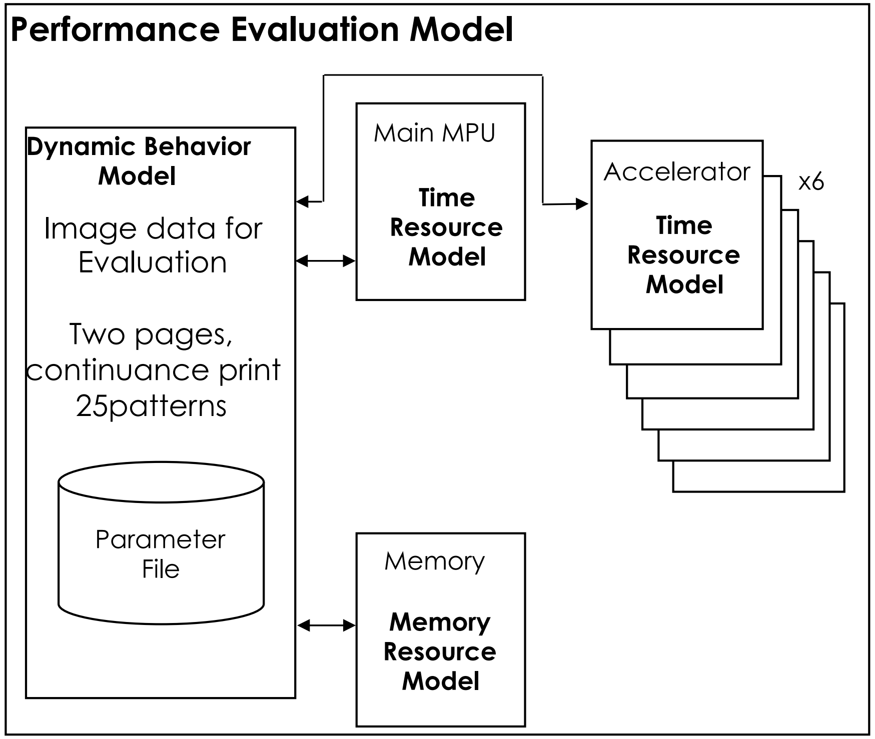
Multi-Core Architecture



IEG : Internal Expression Generation
IEP : Internal Expression Process

Performance Evaluation Model

Single-Core Architecture



Performance Evaluation Model

Dynamic Behavior Model

Image data for Evaluation

Two pages, continuance print 25patterns

Parameter File

Main MPU

Time Resource Model

Accelerator

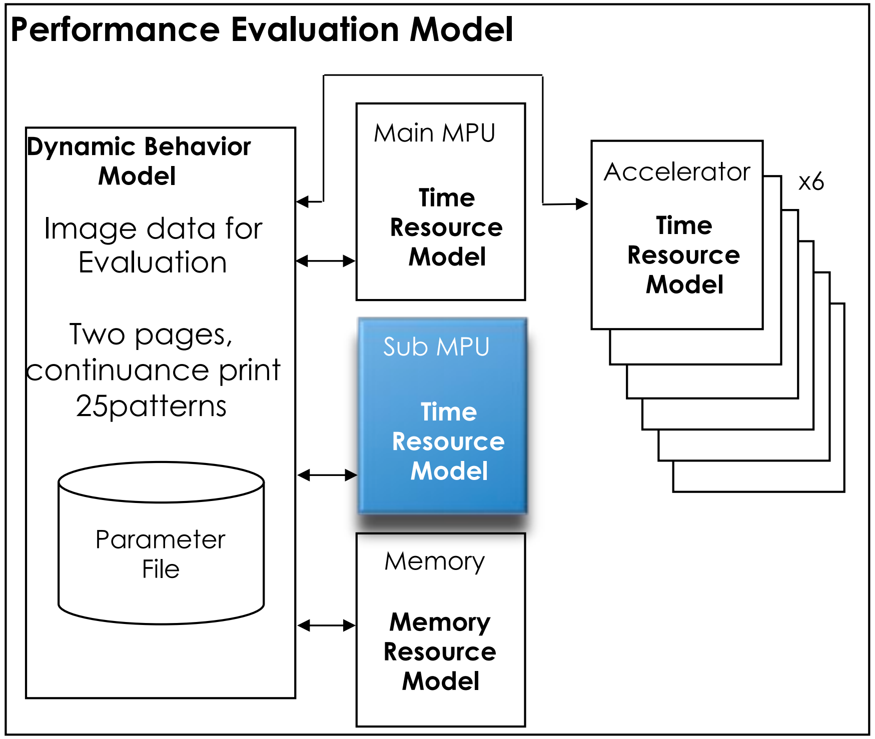
Time Resource Model

x6

Memory

Memory Resource Model

Multi-Core Architecture



Performance Evaluation Model

Dynamic Behavior Model

Image data for Evaluation

Two pages, continuance print 25patterns

Parameter File

Main MPU

Time Resource Model

Sub MPU

Time Resource Model

Accelerator

Time Resource Model

x6

Memory

Memory Resource Model

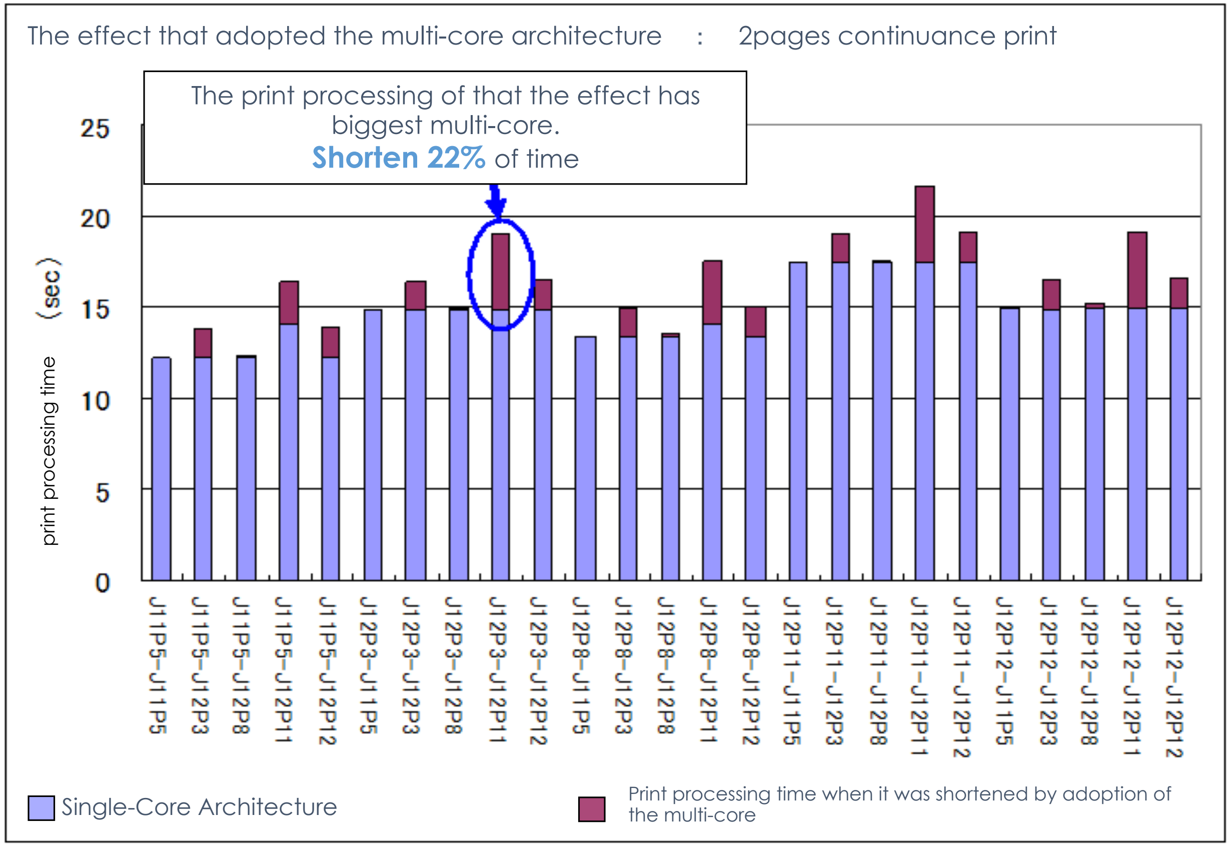
Execution Scenario

Print quality evaluation image JEITA J12-P11

Print quality evaluation image JEITA J12-P15

JEITA : Japan Electronics and Information Technology

Performance Evaluation – Model Simulations



The effect that adopted the multi-core architecture : 2pages continuance print

The print processing of that the effect has biggest multi-core.
Shorten 22% of time

▨ Single-Core Architecture

▨ Print processing time when it was shortened by adoption of the multi-core

Relative comparison with the single core architecture

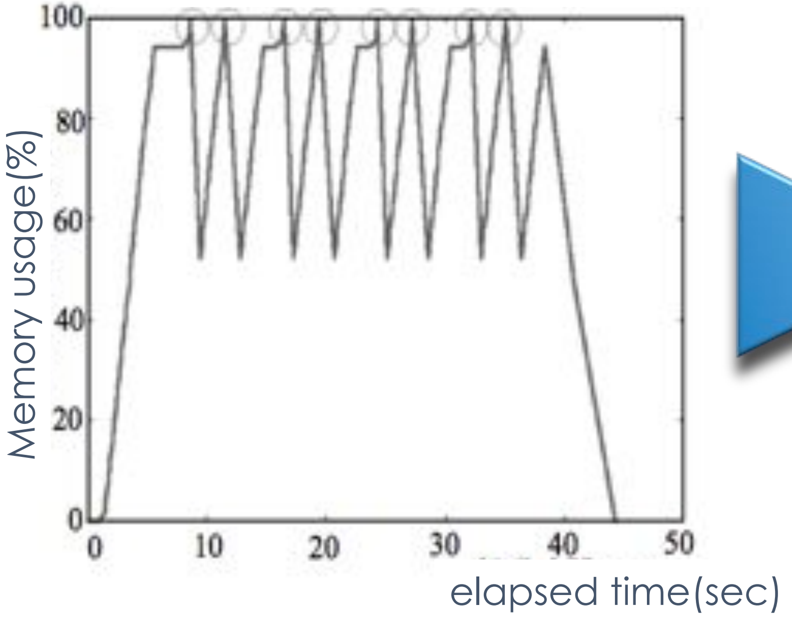
Short reduction ratio of the print processing time

Max. 22.0%

Mean. 8.3%

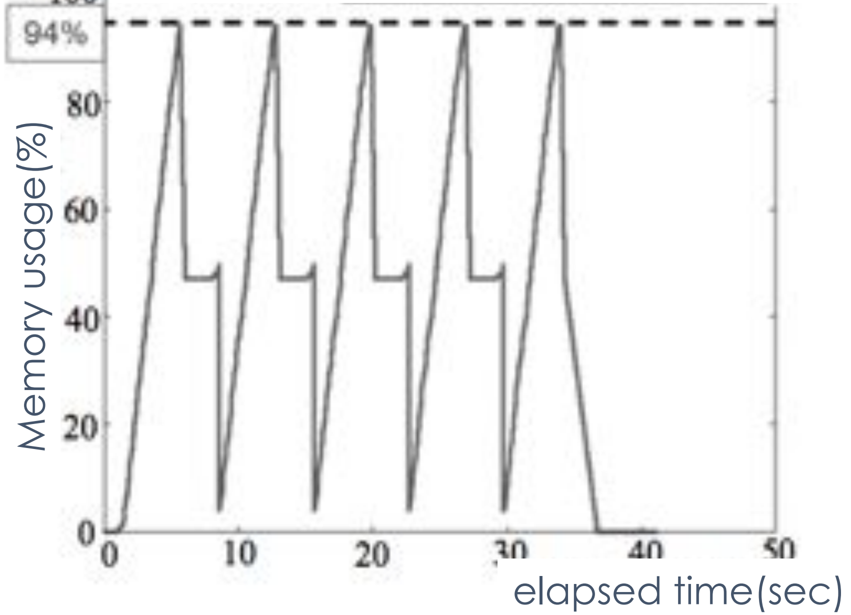
Memory Usage – Model Simulations

8 times of stalls with the lack of memory occur



Change of the memory usage

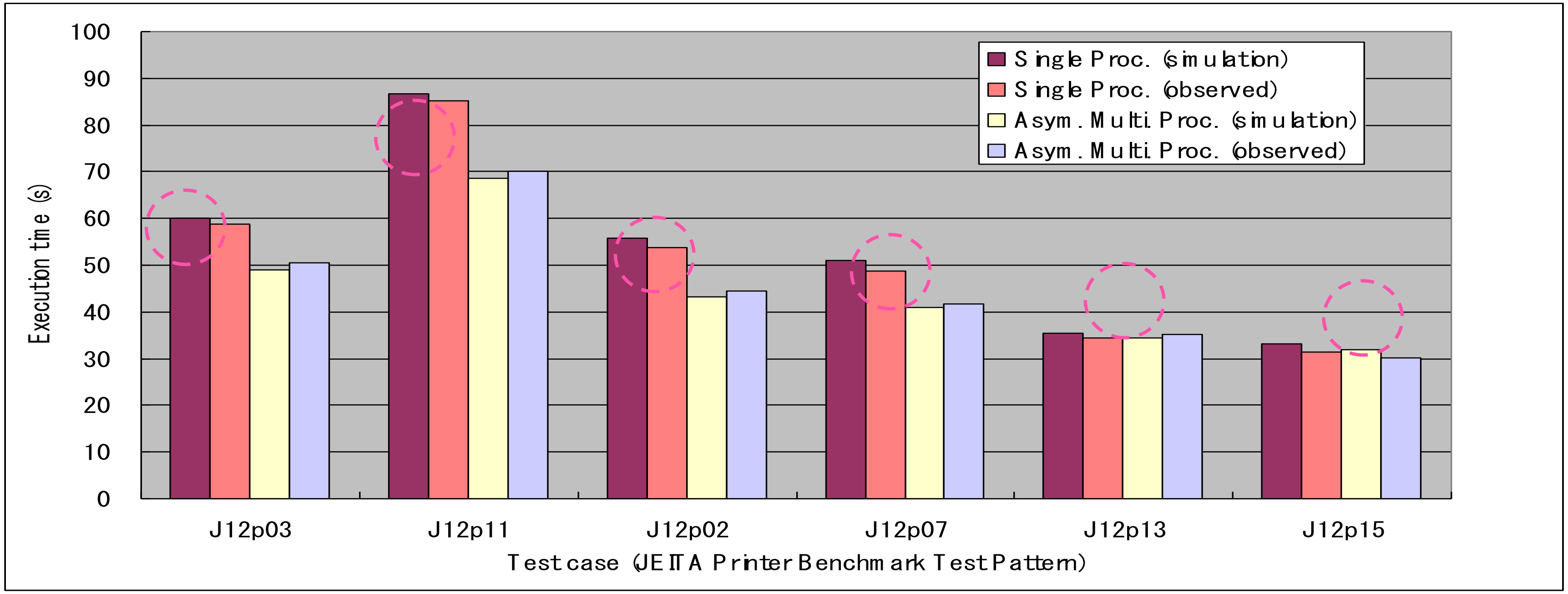
stall with the lack of memory does not occur



memory free timing modified

Validity Verification

- To verify the whole processes.
- Used an FPGA-based evaluation platform.
- Model Simulation vs Evaluation Platform.
- Difference ： 1.1% ~ 6.0%



100
90
80
70
60
50
40
30
20
10
0
Execution time (s)
Single Proc. (simulation)
Single Proc. (observed)
Asym. Multi Proc. (simulation)
Asym. Multi Proc. (observed)
J12p03
J12p11
J12p02
J12p07
J12p13
J12p15
Test case (JEITA Printer Benchmark Test Pattern)

Energy Estimation for SoC

- Appropriate to estimate the energy consumption of the SoC **in the early stages** of SoC development.

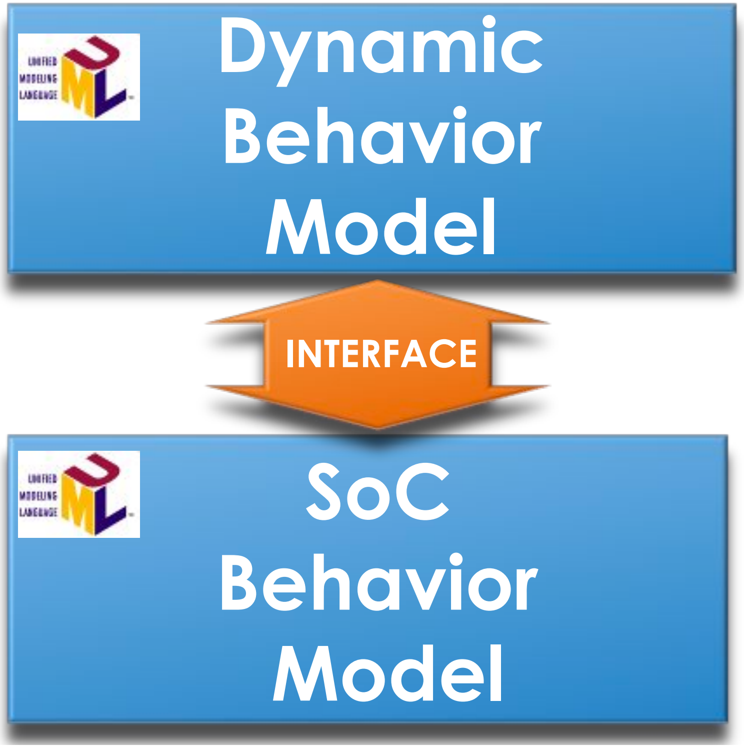
- **Higher accuracy** than conventional estimation methods.

Model for Energy Estimation

The energy consumption of an SoC used in embedded systems **is strongly affected by the dynamic behavior** of the software.

- Describe the **dynamic behavior of the software**.
- executable UML model

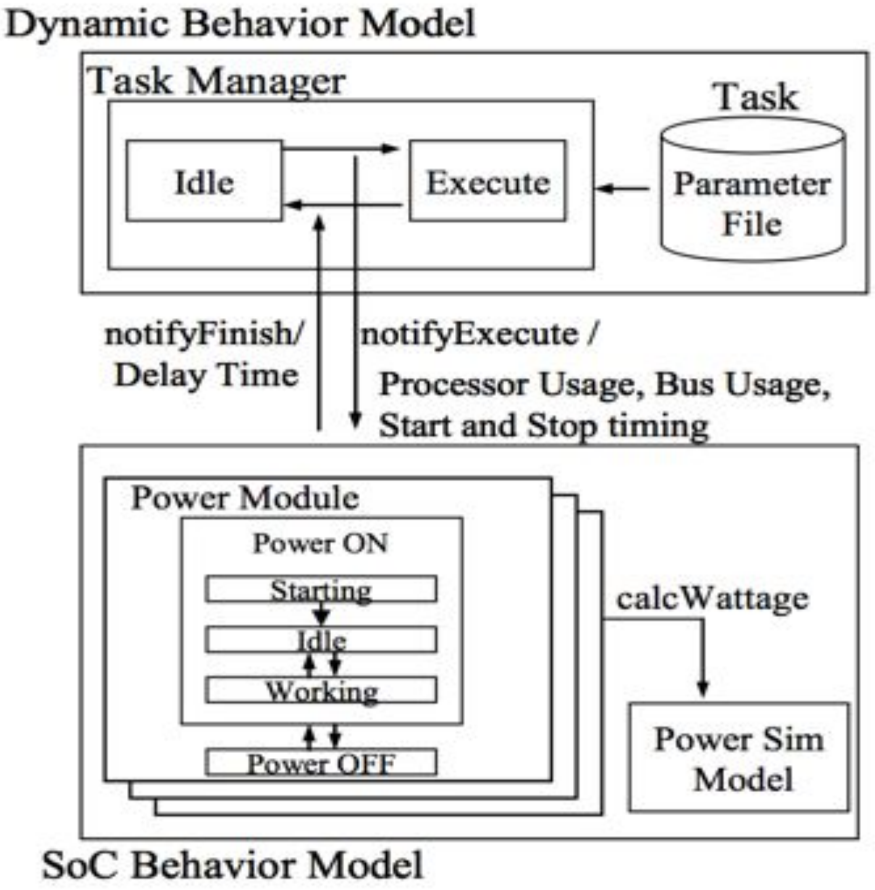
- Describe the **energy consumption and the delay time** of the SoC.
- executable UML model



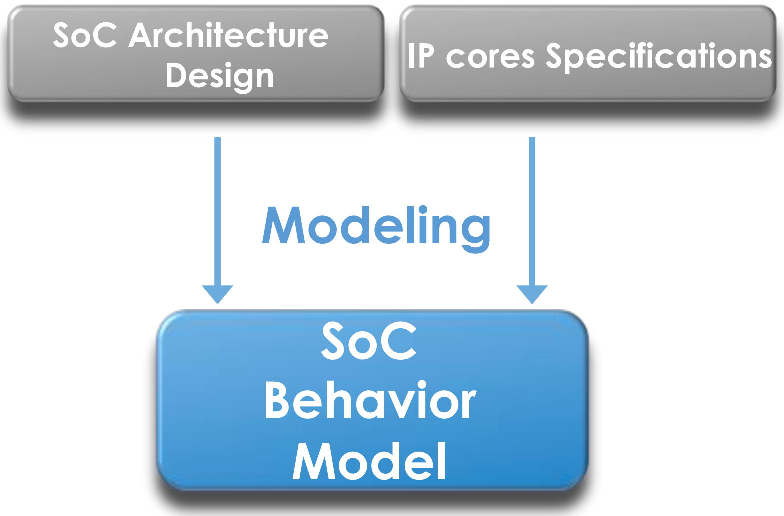
Model for Energy Estimation - Inside

Dynamic Behavior Model

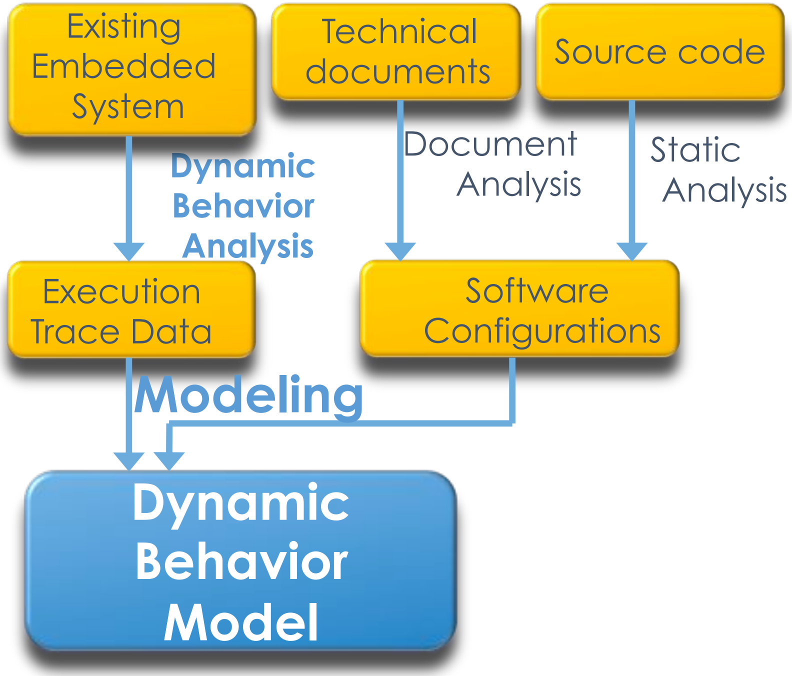
SoC Behavior Model



Modeling Flow



Created form the energy consumption **specification** of the SoC and IP Core.



Created from the existing embedded system utilizing a **Dynamic Behavior Analysis**.

Dynamic behavior model

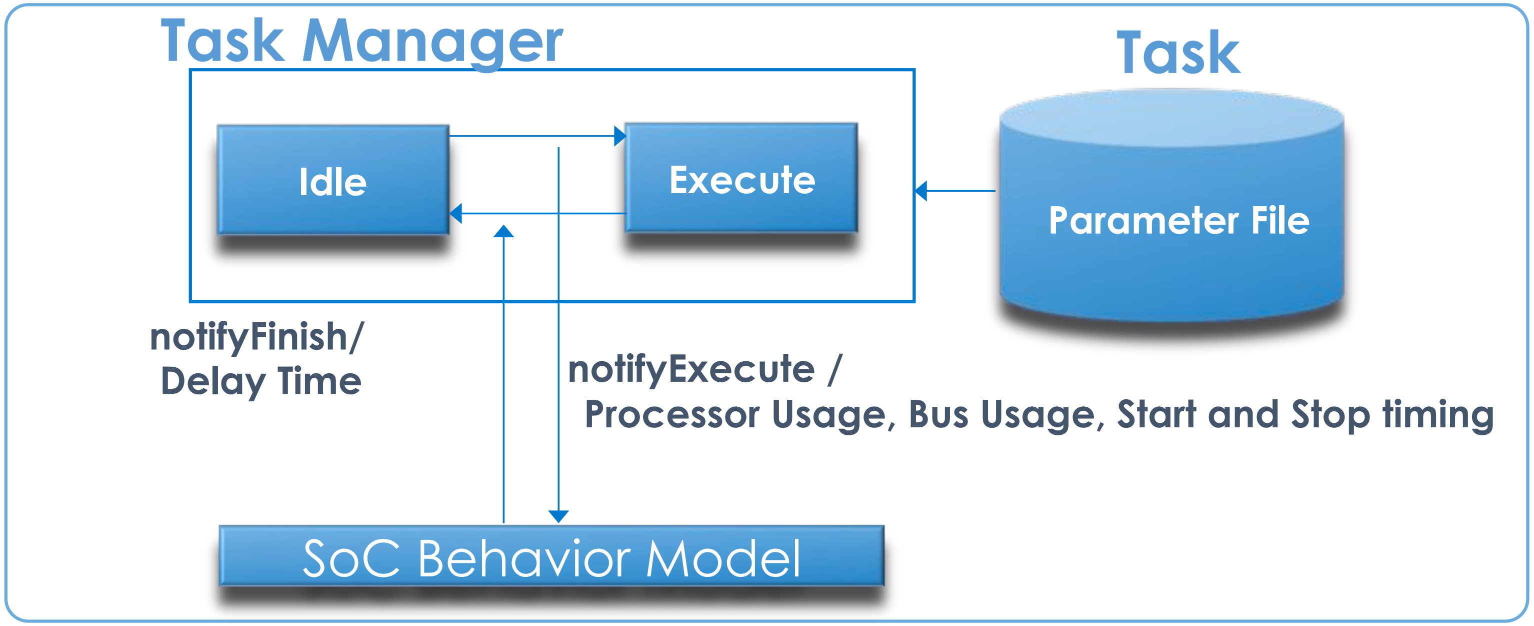
Task Manager Module

- **Control the SoC behavior model**
- described by a sequence diagram

Task Module

- Parameter file
- **Created from the execution trace data**
- loaded for a simulation scenario

Describe the dynamic behavior of the software



SoC behavior Model

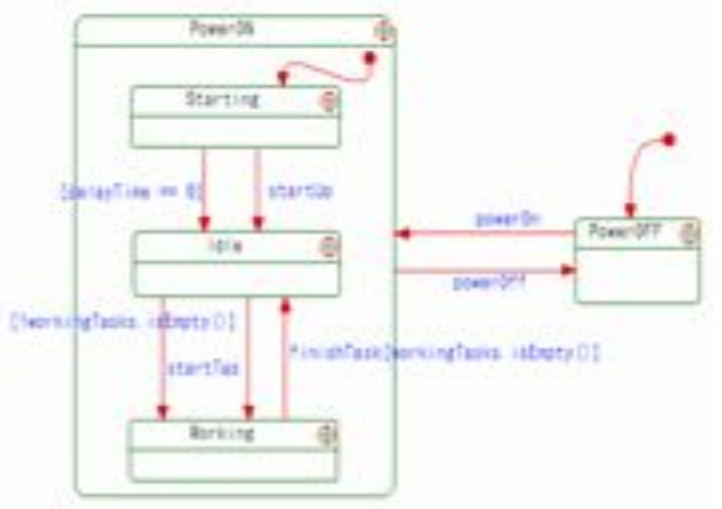
Power Module

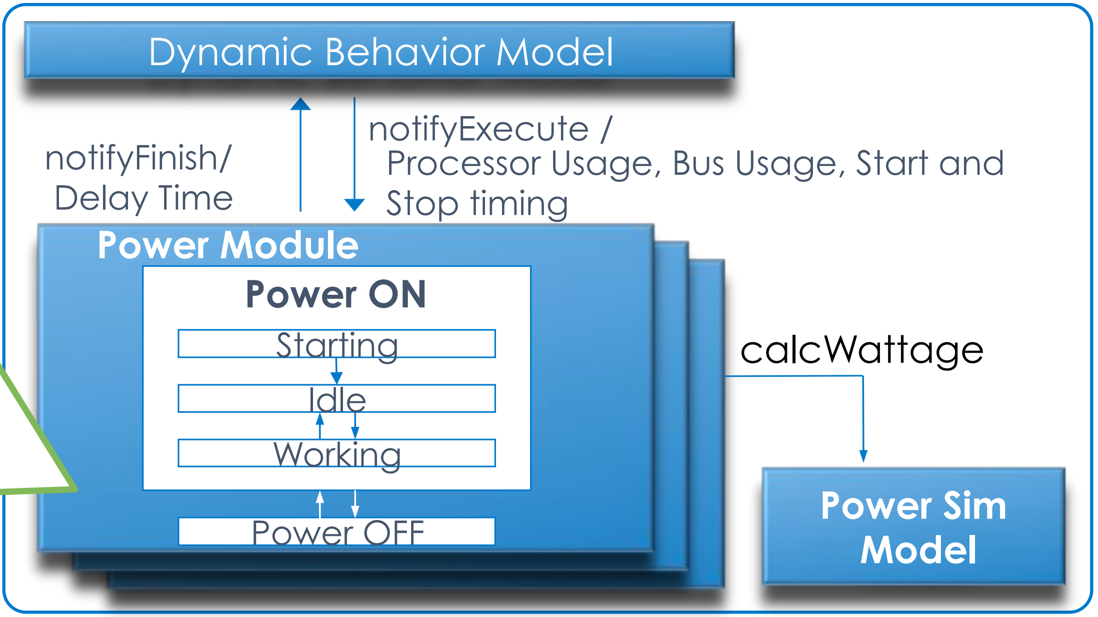
- **Controls the state of each IP core.**
- Calculate IP core energy consumption
- **Track the delay time of each behavior** for the dynamic behavior model.

Power Sim Model

- Accumulates the **energy consumption** of each power module
- Calculates the **total power consumption**

State Machine diagram





Dynamic Behavior Model

notifyFinish/
Delay Time

notifyExecute /
Processor Usage, Bus Usage, Start and
Stop timing

Power Module

Power ON

Starting

Idle

Working

Power OFF

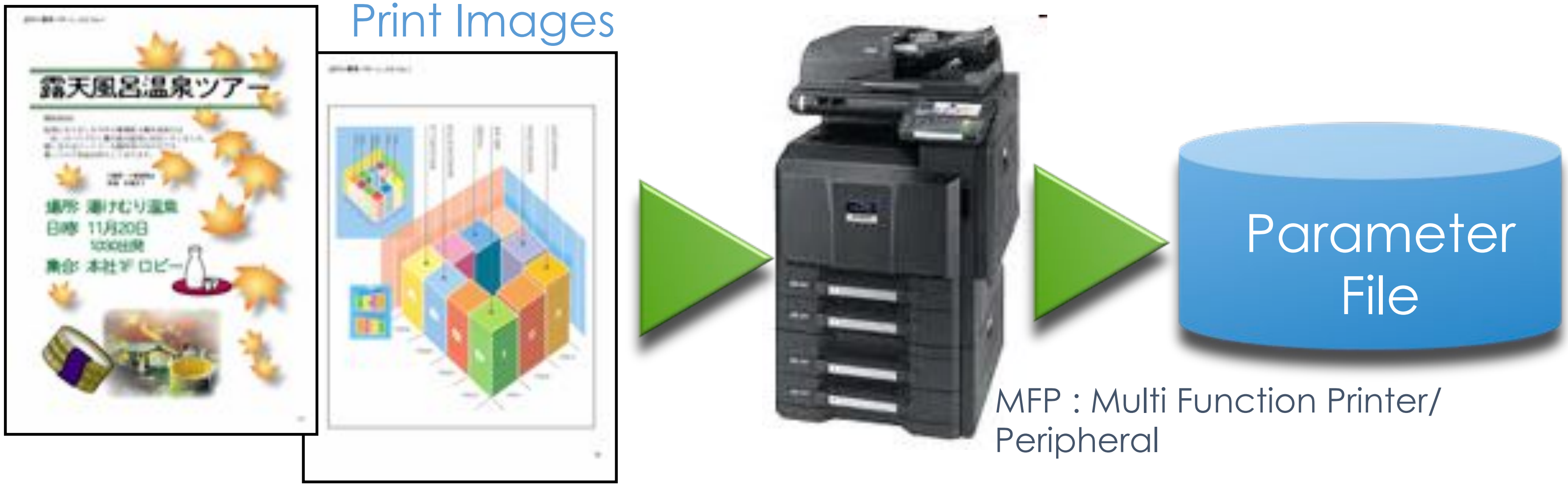
calcWattage

Power Sim Model

Dynamic behavior Model of MFP

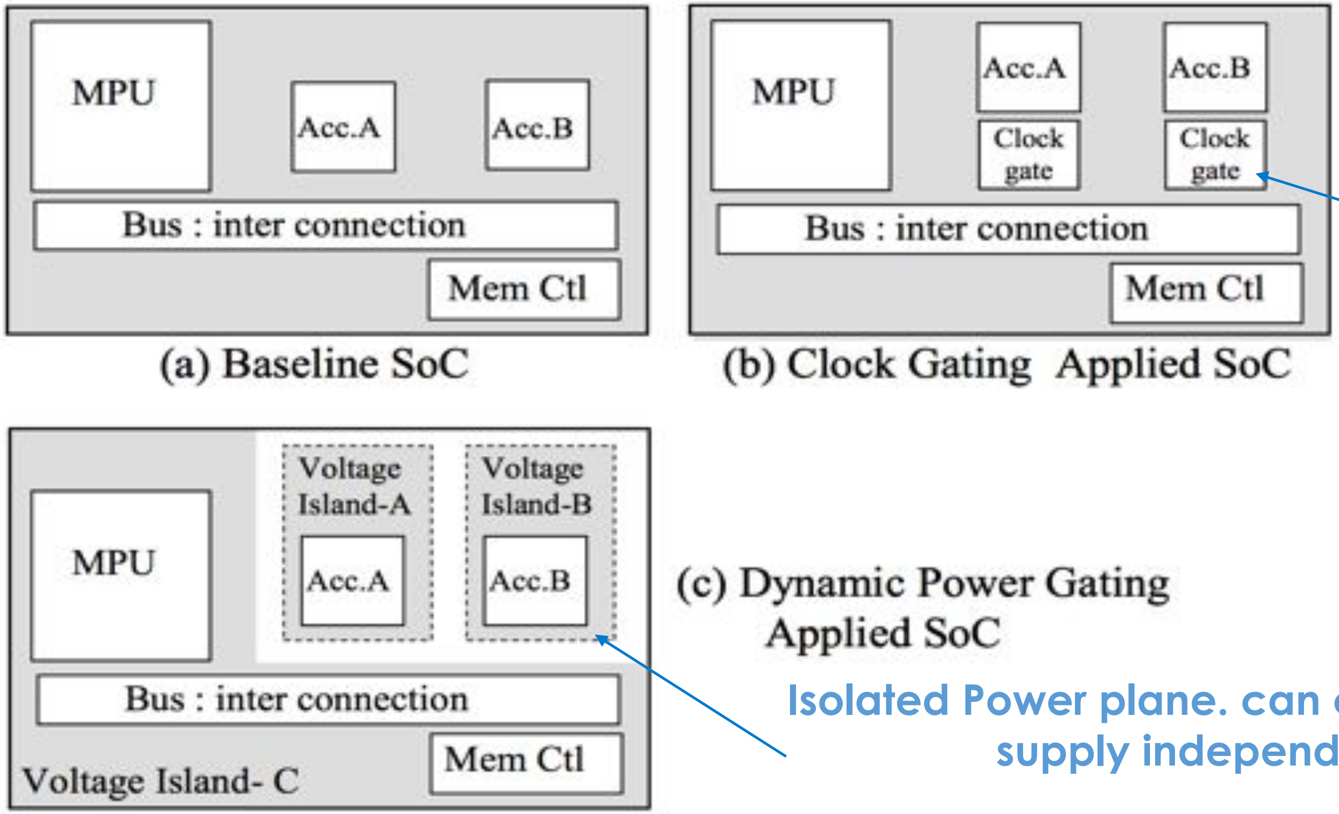
The proposed method is applied for an actual embedded system in an MFP.

- Execution trace data from the actual MFP is collected.
- Execution scenario calls for 4-pages continuous printing.



MFP : Multi Function Printer/ Peripheral

Internal structure of the SoC and energy-saving technology to be applied



(a) Baseline SoC

(b) Clock Gating Applied SoC

(c) Dynamic Power Gating Applied SoC

Suspend the Clock

Isolated Power plane. can cut its power supply independently

Equations of Energy Consumption

SoC Total

$$P = P_{\text{mpu\_rate}} + P_{\text{memc}\cdot\text{bus\_rate}} + P_{\text{acc\_A}} + P_{\text{acc\_B}}$$

MPU

$$P_{\text{mpu\_rate}} = (P_{\text{mpu\_max}} - P_{\text{mpu\_min}}) \cdot \text{MPU usage}(\%) + P_{\text{mpu\_min}}$$

Bus and Memory Controller

$$P_{\text{memc}\cdot\text{bus\_rate}} = \{(P_{\text{mem\_max}} - P_{\text{mem\_min}}) + (P_{\text{bus\_max}} - P_{\text{bus\_min}})\} \cdot U + P_{\text{mem\_min}} + P_{\text{bus\_min}}$$

$$U = \frac{\text{MemoryTransferSize}}{\text{EffectiveMemoryBandwidth}}$$

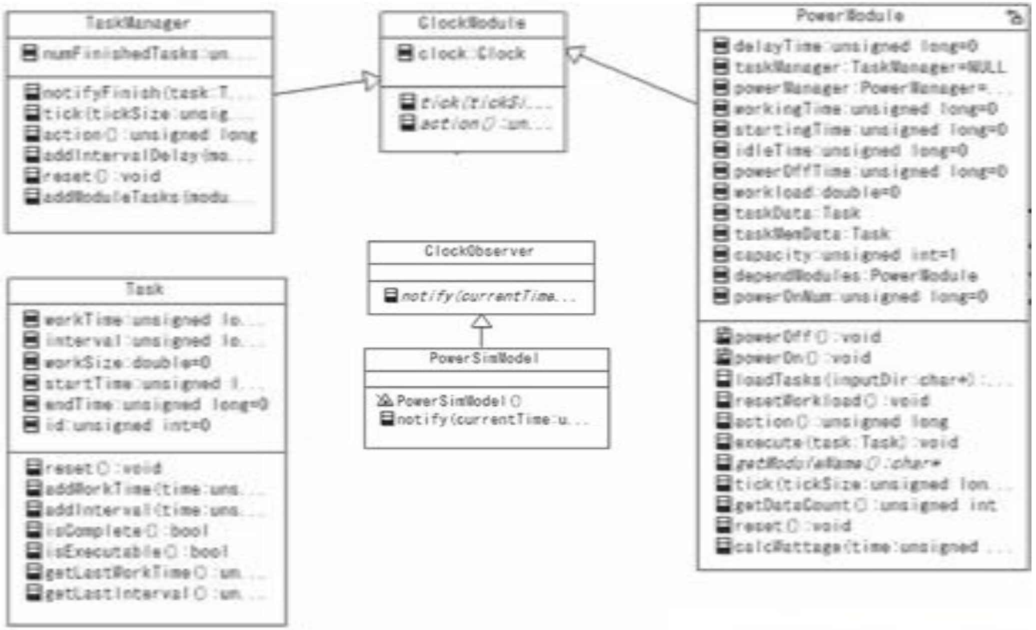
Accelerator A and B

$$P_{\text{acc\_}X} = \begin{cases} \sum_{t\in T} P_{\text{acc\_op}}(t) + \sum_{t\in V} P_{\text{acc\_fr}}(t) & \text{(BaselineSoC)} \\ \sum_{t\in T} P_{\text{acc\_op}}(t) + \sum_{t\in V} P_{\text{acc\_cg}}(t) & \text{(ClockGating)} \\ \sum_{t\in T} P_{\text{acc\_op}}(t) + \sum_{t\in V} P_{\text{acc\_pg}}(t) & \text{(DynamicPowerGating)} \end{cases}$$

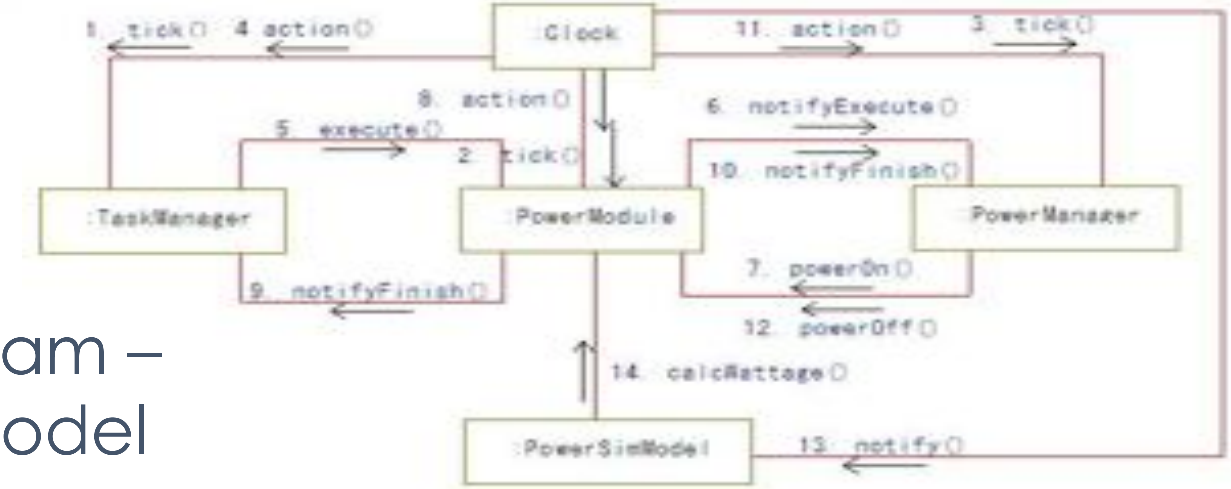
$$(X = \text{A}, \text{B})$$

TABLE I. ENERGY CONSUMPTION FOR EACH IP CORE IN THE SOC

IP Core	Condition	Name	Energy Consumption (mJ)
Processor	Working(Max.)	Pmpu_max	615
	Idle	Pmpu_min	120
Memory Controller	Working(Max.)	Pmem_max	53
	Idle	Pmem_min	35
Bus and Inter connections	Working(Max.)	Pbus_max	37
	Idle	Pbus_min	20
Accelerator –A And Accelerator-B	Working(Max.)	Pacc_op	165
	Idle	Pacc_fr	140
	Clock Gating	Pacc_cg	84
	Dynamic Power Gating	Pacc_pg	9

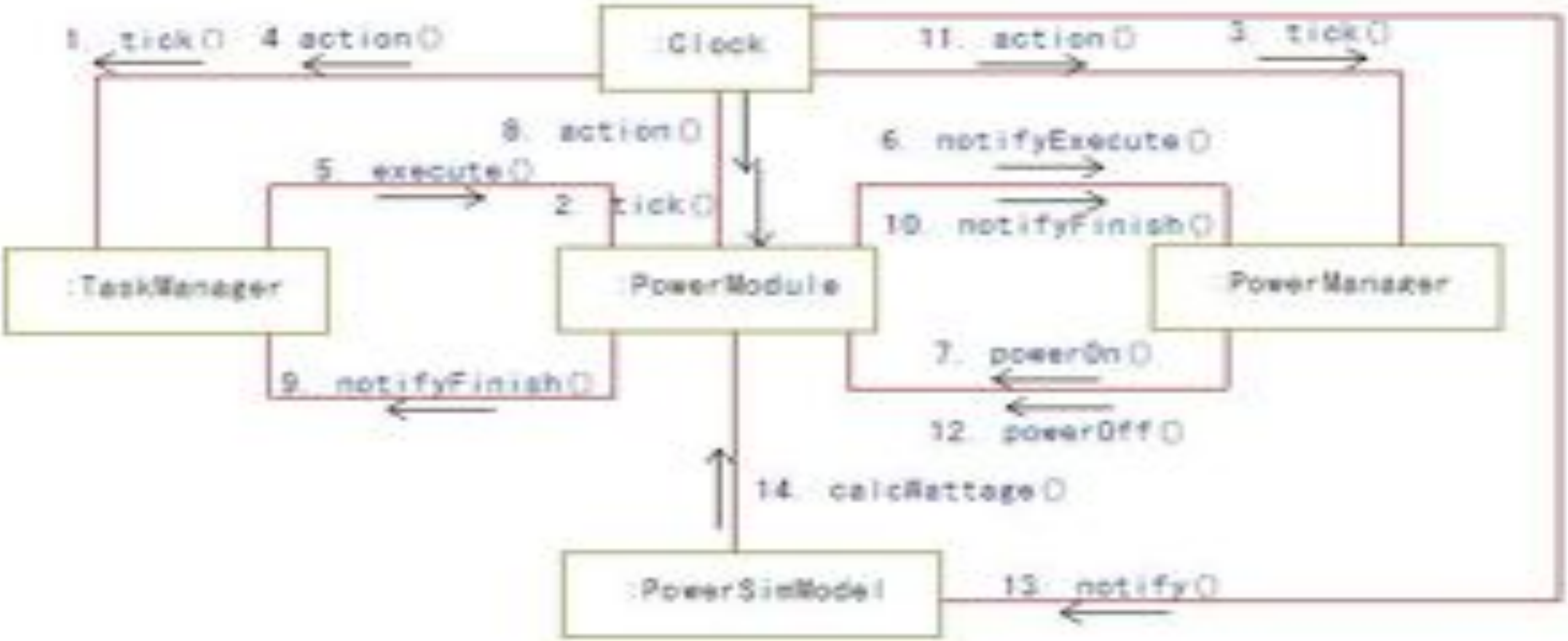


Class Diagram – Entire Energy Evaluation Model



Collaboration Diagram – Energy Evaluation Model

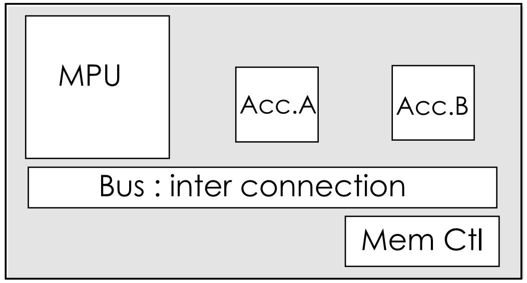
Collaboration Diagram – Energy Evaluation Model

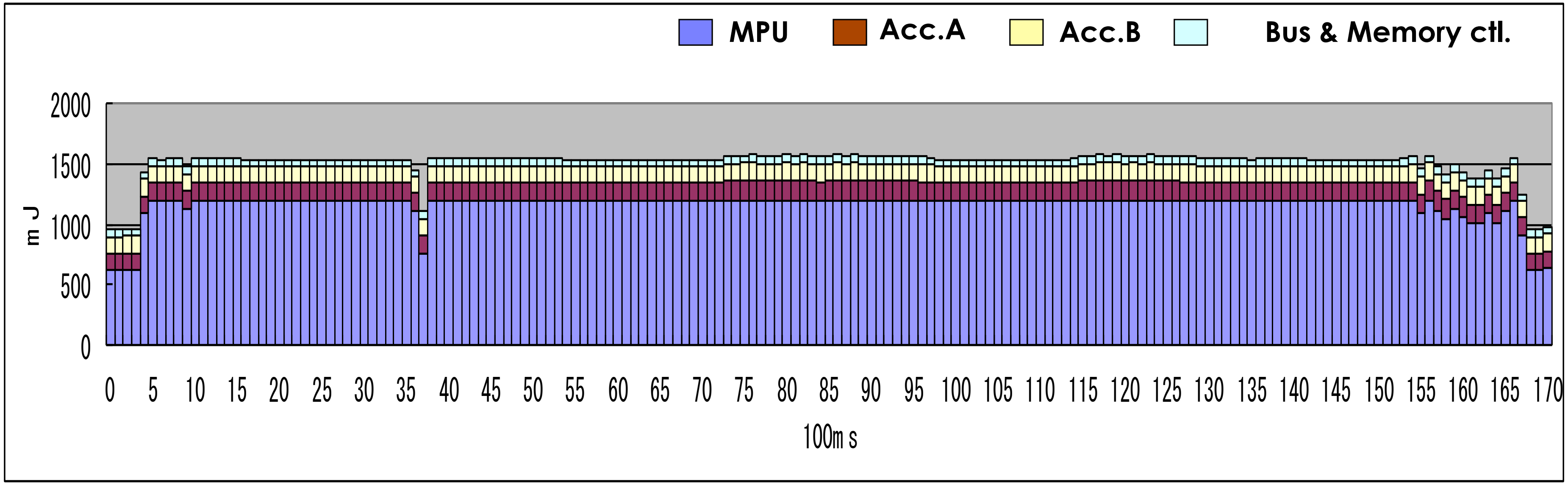


Simulation Result - Baseline

Simulation results showing the changes in the energy consumption over time.

- Mean 1,509mJ
- Peak 1,574mJ
- Whole 25,807mJ
- Execution time 17.10s

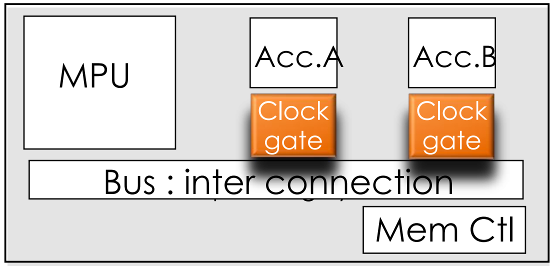


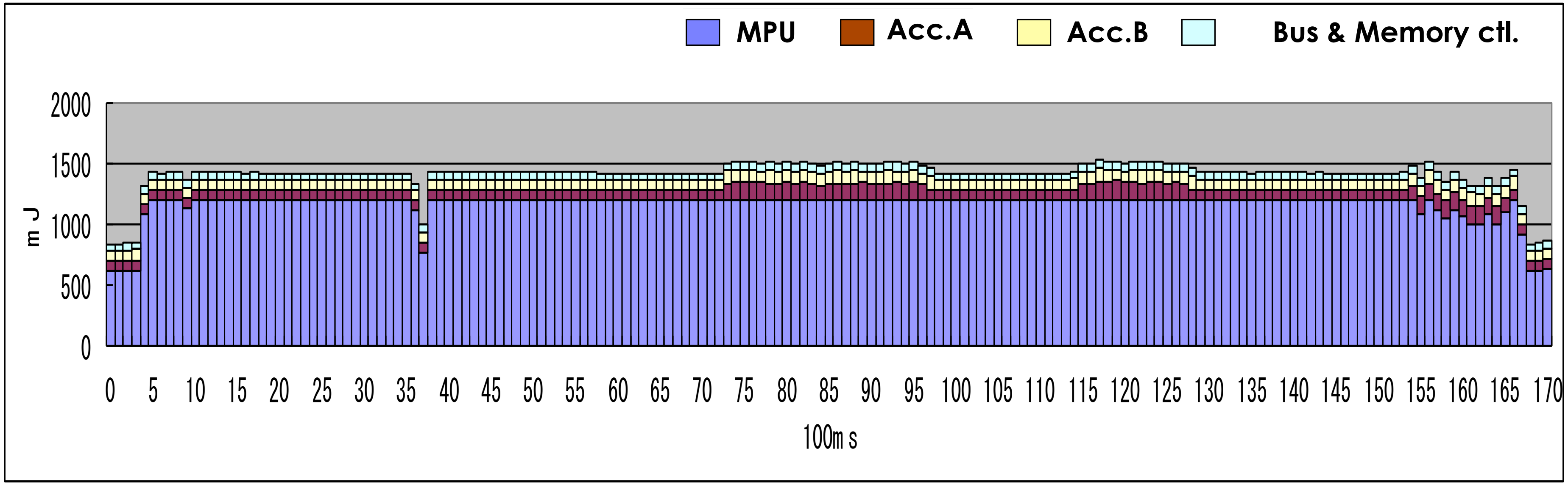


Simulation Result – Clock Gating

Reduces the energy consumption compared to the baseline SoC model.

- Mean 1,412mJ
- Peak 1,565mJ
- Whole 24,147mJ
- Execution time 17.10s

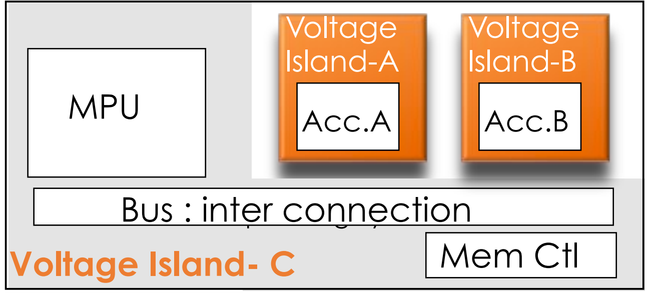


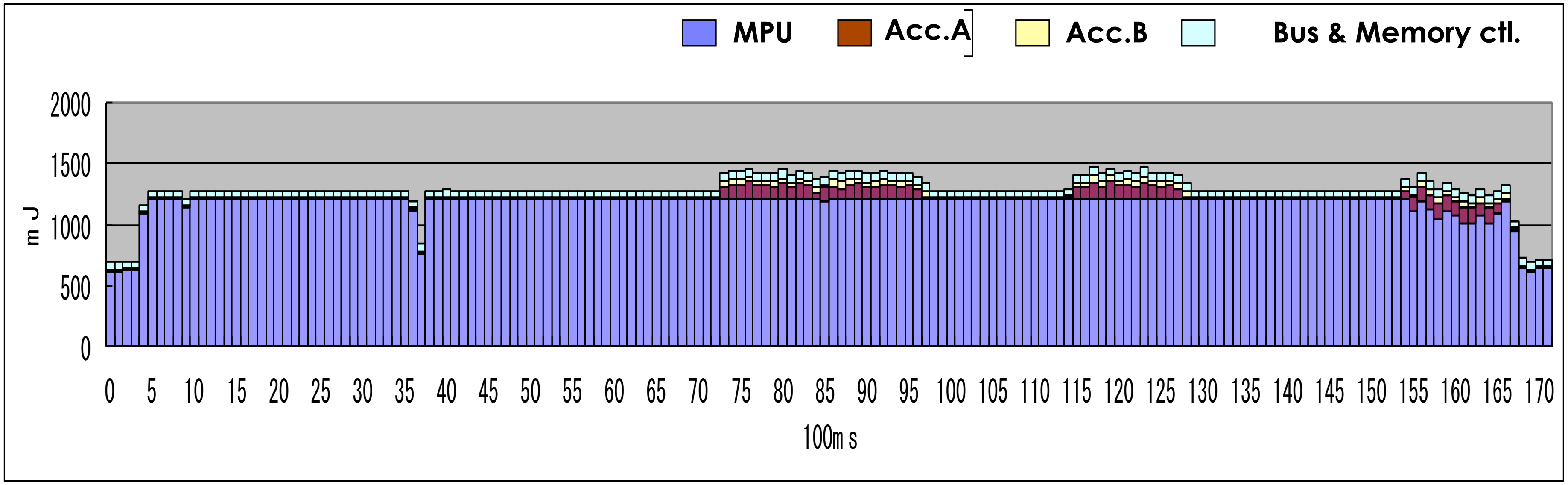


Simulation Result - Dynamic Power Gating

Reduces the energy consumption compared to the Clock Gating applied SoC model.

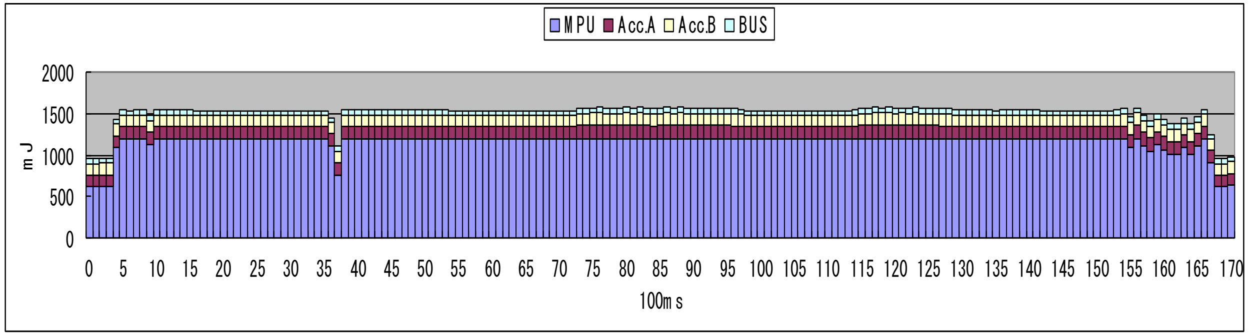
- Mean 1,282mJ
- Peak 1,468mJ
- Whole 21,931mJ
- Execution time 17.11s





Comparison of Simulation Results

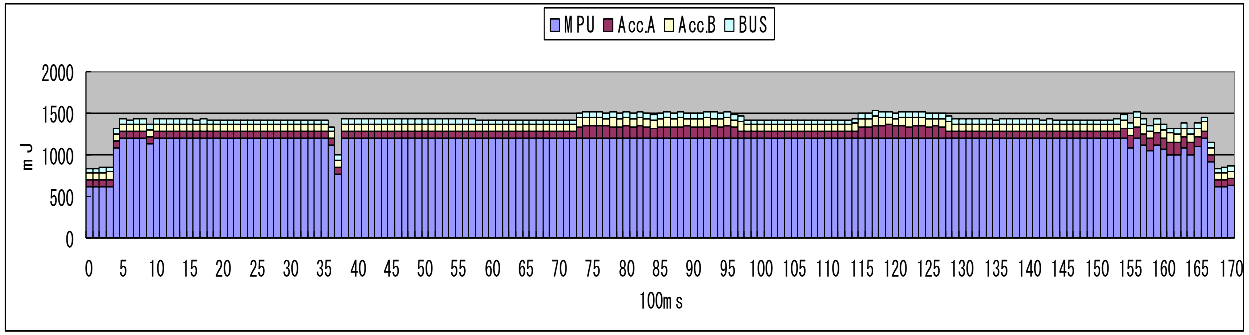
a) Baseline SoC



- Mean 1,509mJ
- Peak 1,574mJ
- **Whole 25,807mJ**
- Execution time 17.10s

6.5% Reduced

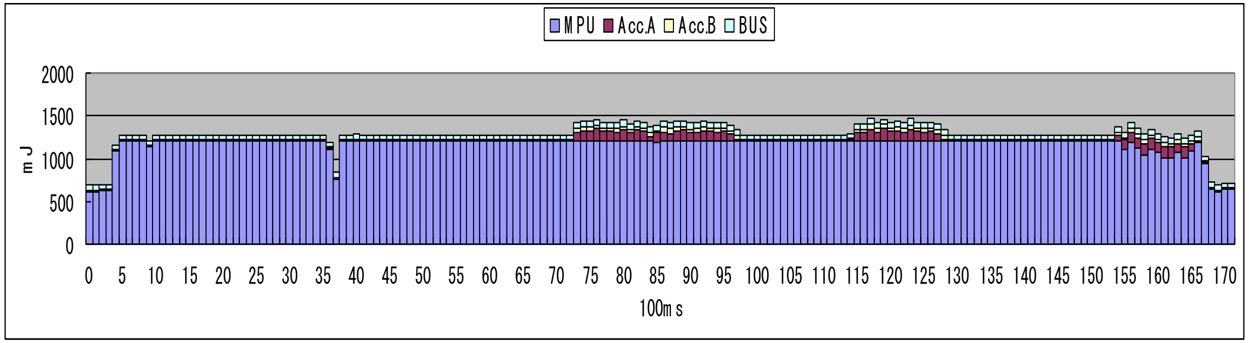
b) Clock Gating applied SoC



- Mean 1,412mJ
- Peak 1,565mJ
- **Whole 24,147mJ**
- Execution time 17.10s

9.1% Reduced

c) Dynamic Power Gating applied SoC



- Mean 1,282mJ
- Peak 1,468mJ
- **Whole 21,931mJ**
- **Execution time 17.11s**

10msec longer

Accumulate the switching time of each power plane.

Validity Verifications

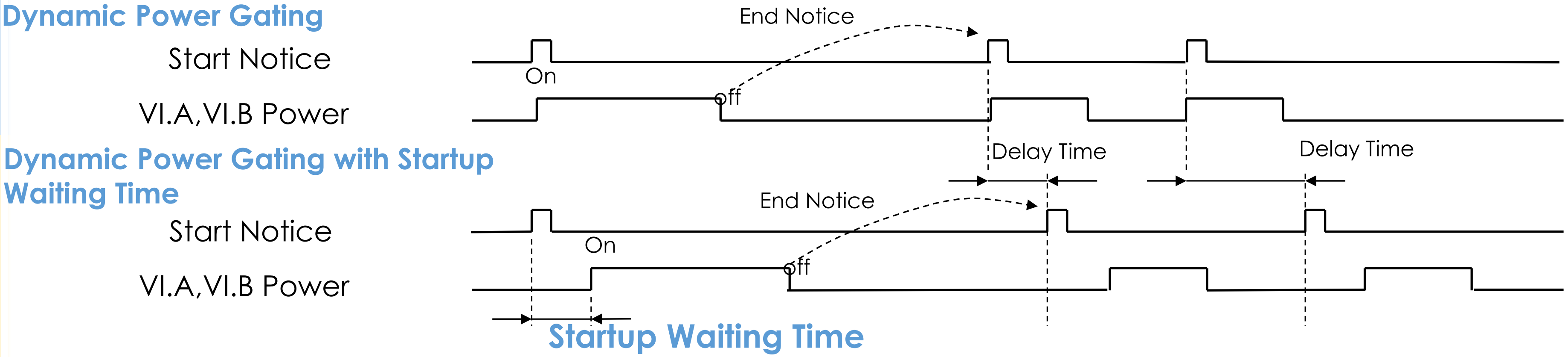
Comparison : **Proposed method**, **Spreadsheet** method and **Actual SoC**.

- Proposed method is **appropriate to estimate** the energy consumption of the SoC in the early stages of SoC development
- Proposed method had **higher accuracy** than the standard spreadsheet-based method

SoC type	(a) Proposed Method (mJ)	(b) Spread sheet (mJ)	(c) Actual SoC (mJ)	Error (a) vs (c) (%)	Error (b) vs (c) (%)
Baseline	1,509	1,620	1,361	10.9	19.0
Clock Gating Applied	1,412	1,488	1,250	13.0	19.0
Dynamic Power Gating Applied	1,282	1,377	1,113	15.2	23.7
Average	-	-	-	**13.0**	**20.6**

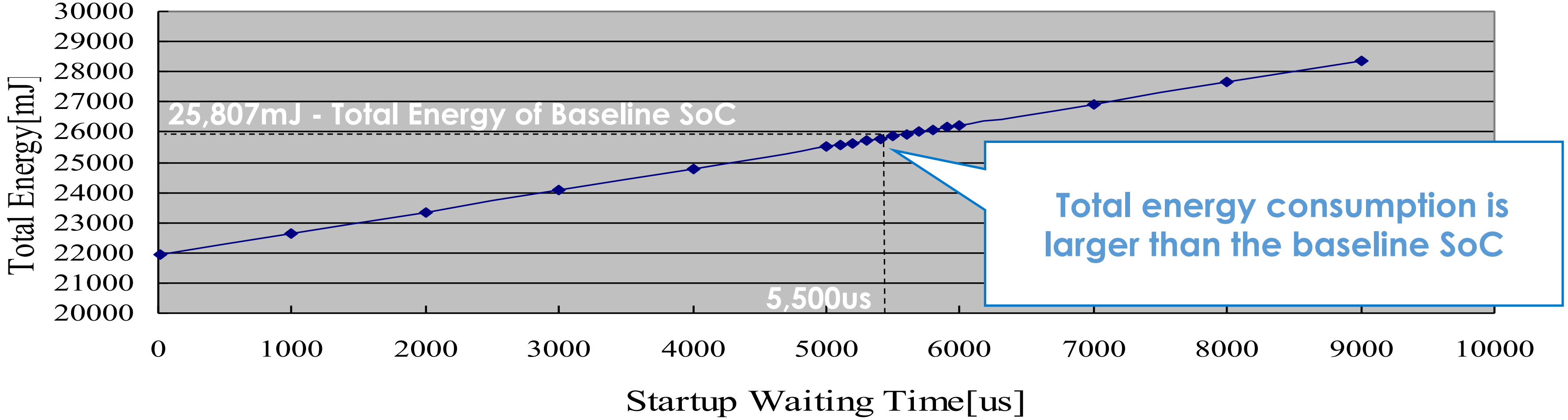
Simulation – Tradeoff of Dynamic Power Gating

One tradeoff is between the startup waiting time and the energy consumption.



Dynamic Power Gating

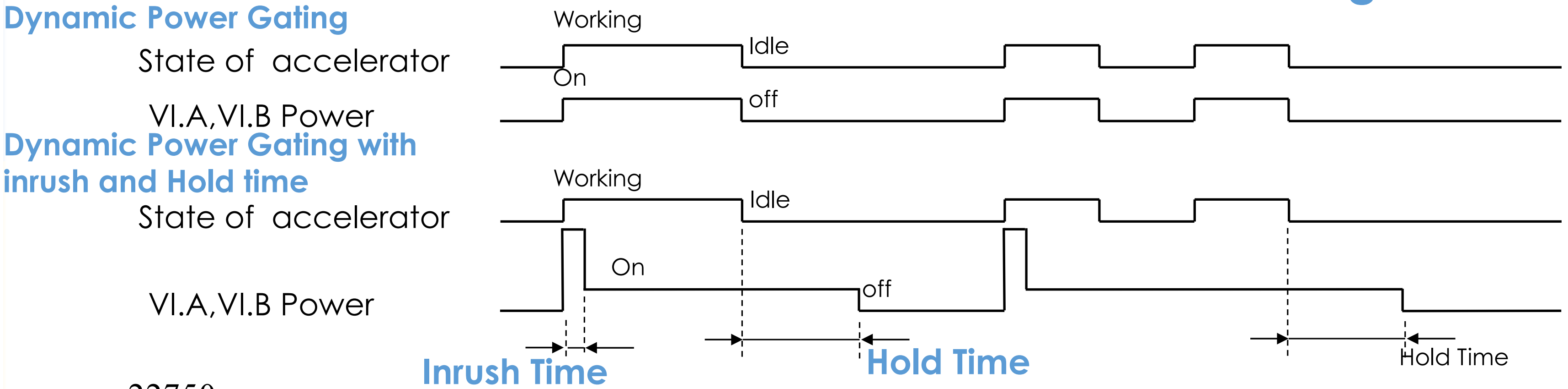
Dynamic Power Gating with Startup Waiting Time



Total energy consumption is larger than the baseline SoC

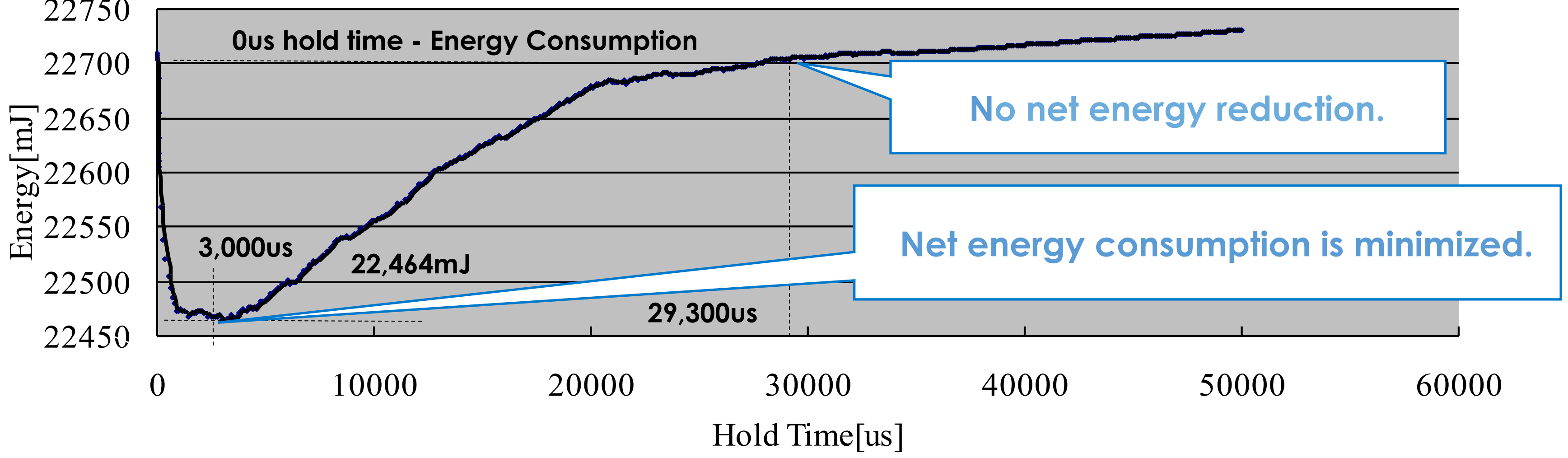
Simulation –Dynamic Power Gating with Inrush

- Surge current - at the time of power supply startup.
- Hold time – extend time to Cut off a Voltage-Island power



Dynamic Power Gating

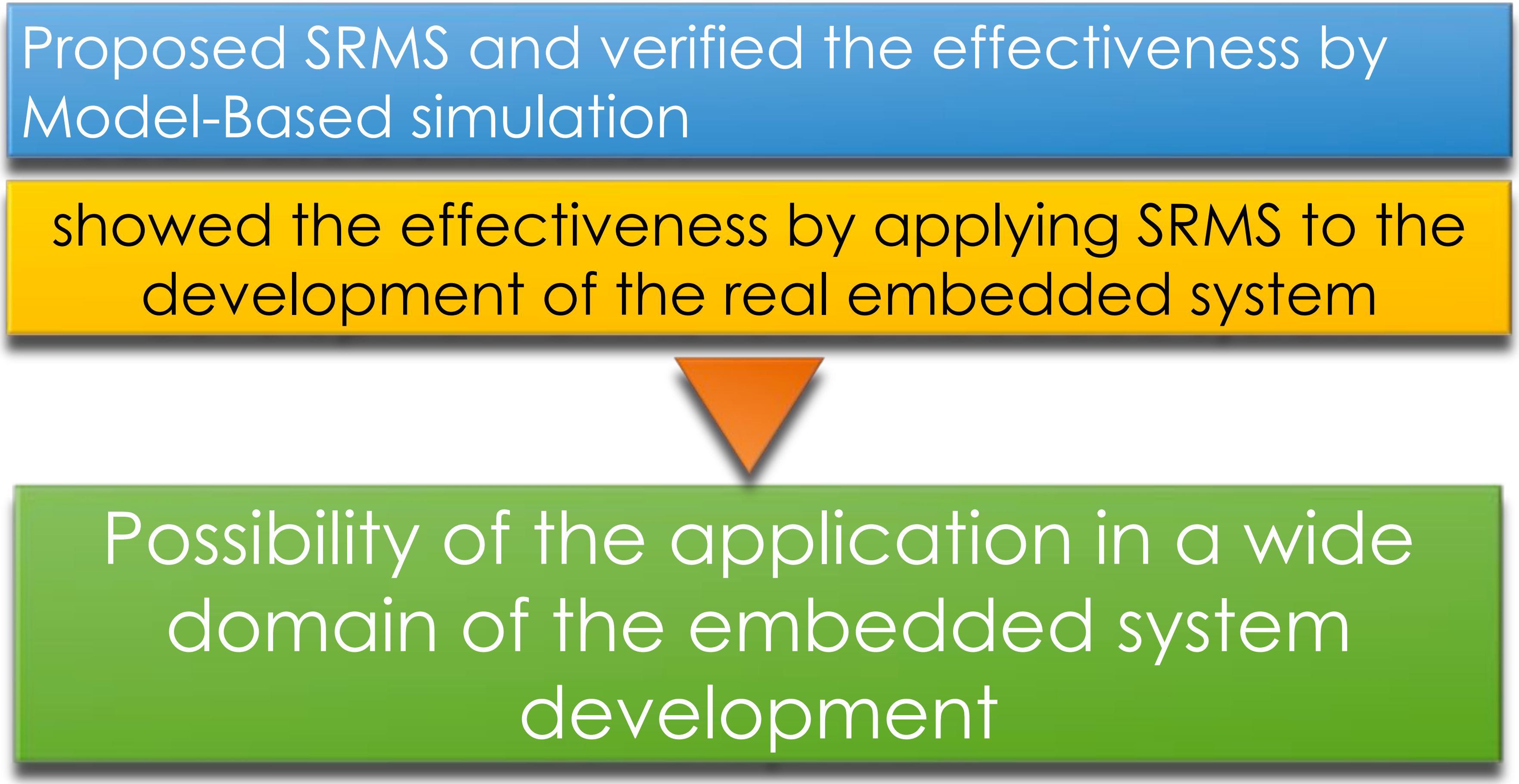
Dynamic Power Gating with inrush and Hold time



No net energy reduction.

Net energy consumption is minimized.

Conclusions



Proposed SRMS and verified the effectiveness by Model-Based simulation

showed the effectiveness by applying SRMS to the development of the real embedded system

Possibility of the application in a wide domain of the embedded system development

Thank you for your attention