



**Center for Embedded Computer Systems  
University of California, Irvine**

---

## **Embedded System Environment : Overview**

Kyoungwon Kim

Center for Embedded Computer Systems  
University of California, Irvine  
Irvine, CA 92697-2620, USA

{kyoungk1}@uci.edu

CECS Technical Report 11-11  
December, 29, 2011

# Embedded System Environment : Overview

Technical Report CECS-11-11

Kyoungwon Kim

Center for Embedded Computer Systems,  
University of California, Irvine  
Irvine, CA, 92697, USA  
+1 (949) 824-8919  
kyoungk1@uci.edu

December, 29, 2011

## Abstract

*This report describes ESE(Embedded System Environment), a tool set for modeling, synthesis and validation of MPSoC design. It is an answer to the modern design challenge, widening productivity gap. In the ESE design flow, the final implementation is obtained from the universal specification of a given system by automatic or semi-automatic refinement. Application, platform and their mapping serve as the specification for the system. The level of abstraction is raised to transaction level. Verification of functionalities as well as validation of user requirements such as cost, power consumption and performance should be done by any design process. Transaction-Level Model(TLM) makes the tasks much easier so that the design cycles are shortened and wide design space exploration is available. TLM estimation is more than 90% accurate as Pin-Cycle Accurate Model(PCAM) based estimation while the simulation speed is a couple of thousand times faster. Compared to prototype board, execution on which is faster than TLM simulation in addition to that it has high accuracy, TLM is available at much earlier design stages. Depending on TLM that is even automatically generated, designers benefits from high design productivity.*

# Contents

<b>Table of Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>ii</b>
<b>List of Algorithms</b>	<b>ii</b>
<b>List of Tables</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 ESE Idea</b>	<b>2</b>
<b>3 TLM Estimation</b>	<b>4</b>
<b>4 Application Estimation</b>	<b>6</b>
4.1 Processing Unit Model(PUM) . . . . .	6
4.2 Algorithms for Application Estimation . . . . .	7
4.3 Experiment . . . . .	10
4.4 Summary . . . . .	11
<b>5 Timed RTOS Model</b>	<b>12</b>
5.1 Overview . . . . .	12
5.2 Experiment . . . . .	12
5.3 Summary . . . . .	14
<b>6 Communication Estimation</b>	<b>14</b>
6.1 Synchronization Delay and Arbitration Delay . . . . .	15
6.2 Data Transfer Delay . . . . .	16
6.3 Bus Delay Database . . . . .	16
6.4 Experiment . . . . .	16
6.5 Summary . . . . .	18
<b>7 Conclusion</b>	<b>18</b>
<b>8 Acknowledgments</b>	<b>19</b>

## List of Figures

1	ESE Design Flow ( source [1] ) . . . . .	2
2	Time and Accuracy Trade-off among Different Models ( source [3] ) . . . . .	3
3	Design Time Saving ( source [1] ) . . . . .	4
4	Design Quality ( source [1] ) . . . . .	4
5	TLM Estimation Framework . . . . .	5
6	Application Estimation( source [4] ) . . . . .	6
7	Example of Processing Unit Model - Microblaze Processor( source [4] ) . . . . .	7
8	Control Flow of AdvClock . . . . .	9
9	MP3 Application ( source [3] ) . . . . .	10
10	Four Designs for MP3 Decoder ( source [3] ) . . . . .	10
11	Three Computation Delay Contributors( source [3] ) . . . . .	12
12	Test Architecture for JPEG+MP3 ( source [3] ) . . . . .	13
13	Communication Estimation ( source [2] ) . . . . .	15
14	JPEG Application ( source [6] ) . . . . .	16
15	JPEG Platforms ( source [6] ) . . . . .	17

## List of Algorithms

1	Optimistic Scheduling Algorithm( source [4] ) . . . . .	8
2	Computation of Total Basic Block Delay( source [4] ) . . . . .	9

## List of Tables

1	Annotation and Simulation Time for Timed TLM ( source [4] ) . . . . .	11
2	Accuracy Result ( SW only, source [4] ) . . . . .	11
3	Error Ratio Compared to Board Measurement[%]( Platforms w/ Custom HW, source [4] ) . . . . .	11
4	Accuracy of Abstract Models[%] ( source [4] ) . . . . .	13
5	RTOS Generation Time[sec] (source [5] ) . . . . .	13
6	Simulation Time[sec] ( source [5] ) . . . . .	14
7	Error Ratio of TLM Compared to Board Measurement ( source [6] ) . . . . .	17

# 1 Introduction

Design complexity in modern embedded systems has been rapidly increasing. Market is demanding new applications more than it was ever. Moreover, due to both of advance in technology and bottle necks in power supply and cooling systems, complete many-processor systems are being putting together on a single chip(MPSoC). On the contrary, constraints including time-to-market are becoming even more tighter. Those are why productivity is no more secondary issue in modern system designs.

Traditional system design methodologies have been failing to answer the challenges in the system designs. In those methodologies, designers have different models in different levels of abstractions and the models are not compatible with each other. For example, marketing department, application developers, system engineers and HW designers all have their own specification of the system, which falls into waste of time and labor. In addition to having different but inconsistent models in different levels of abstraction, the level of abstraction that the traditional design methods work on is too low to cope with the design complexity in contemporary system designs. Systems are directly designed at low HW or SW level in the design methods. However, verification of system functionalities and validation of user requirements are hard and too time-consuming, then. The tasks are done by Pin-Cycle Accurate Model(PCAM) simulation or by execution on the prototype board. The former is too slow and may not available for the whole system. The latter is accurate and fast but ready only at too late design stages.

Based on the observation, first of all, we can infer that an alternative system design methodology is necessary to be introduced and the methodology must come up with an universal specification of the system. In addition to that, the level of abstraction where the specification is should be raised to a reasonably high level. The problem that follows the raise of the level of abstraction is to bridge the semantic gap between the specification of a given system and its final implementation. To improve productivity, the process needs to be automated as possible. However, for automation, the methodology should clearly define abstraction levels, models in each abstraction level, the semantics of system design languages, design rules, modeling styles and etc.

ESE has emerged as an answer. It takes application, platform and their mapping as the universal specification of the system. ESE raised the abstraction level to transaction level model. That allows a couple of thousand times faster estimation compared to PCAM while more than 90% of accuracy compared to board measurement is being kept. Algorithmic level is faster but not accurate while PCAM is too slow. To close the semantic gap between the specification model and the final implementation, ESE also clearly defines PCAM and provides an automated or semi-automated design process that drives the system specification to the final implementation. By all of those, ESE drastically saves design times while keeping or even improving the quality of design, based on the experiments [4], [5], [3], [6], in which the design time to obtain PCAM for three different platforms is measured and the design quality is also measured in terms of chip utilization, computation time for the given, same task and memory usages. ESE reduces the design time by 76 – 79%. The design quality is not that different from the manual design since ESE uses more BRAMs but less FPGA slices and the performance is almost same.

This report covers only half of ESE, its front-end, the tool for the half of the ESE design process. With a given specification of the system, TLM is automatically generated to be simulated to provide TLM estimation. Based on the estimation, design

optimization can be done so that the design that all constraints are met can be passed as an input to produce PCAM.

Therefore, the following sections are organized as follows. In section 2, an general overview of ESE and its front-end is given. Section 3 is an overview of TLM estimation. TLM estimation, however, consists of three parts, application estimation, timed RTOS modeling and communication estimation. Section 4, 5 and 6 explain each of them, respectively, with experiments. Section 7 is the conclusion and future works.

## 2 ESE Idea

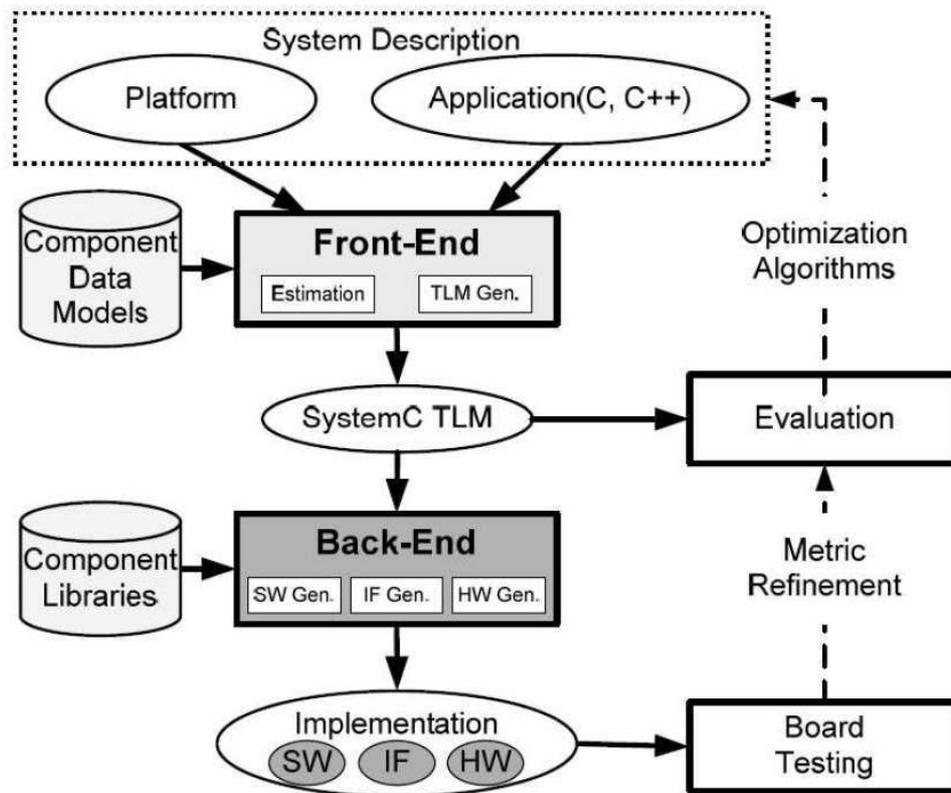


Figure 1: ESE Design Flow ( source [1] )

Figure 1 shows the ESE design flow. This design flow should present how the goal of ESE is or supposed to be achieved. System Description is the universal specification of the system. It consists of platform, application and their mapping. Platform is a net-list of platform components such as PEs, buses and transducers. The components are in transaction level. Application is a set of processes. Each process is one C/C++ function, respectively. Those processes are assumed to be executed concurrently. There is neither hierarchy nor state transition in the processes. A process may be communicating with other processes via channels. These communications put an execution order in the application model.

ESE front-end takes it as an input to produce TLM. For that, it needs static informations. For example, the delays due to execution of application functions may differ depending on the type of PE on which the functions are executed. That is, we need timing database for the PE. That kind of information is stored in the library of

component data models to be used by ESE front-end.

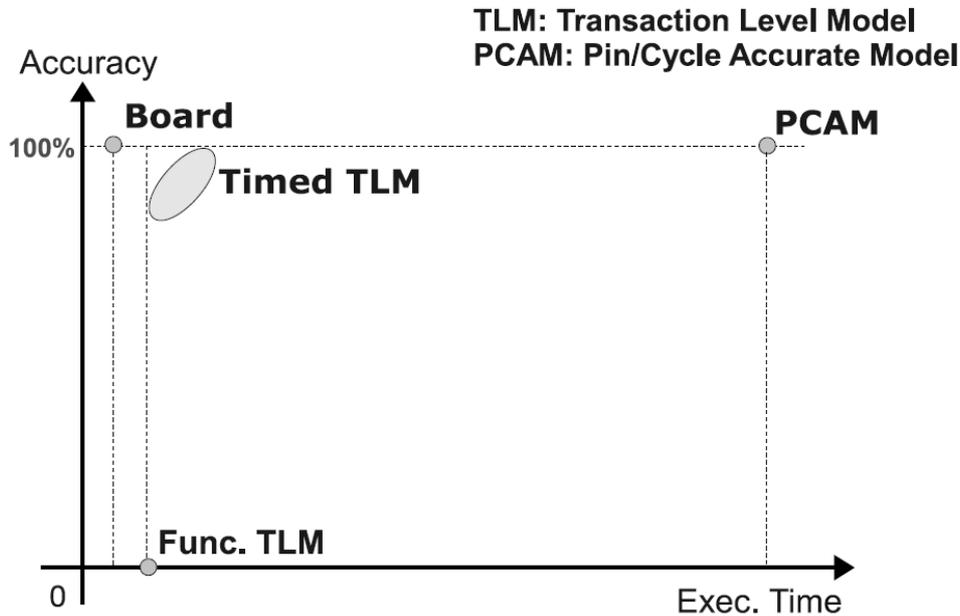


Figure 2: Time and Accuracy Trade-off among Different Models ( source [3] )

The generated TLM is mainly intended to help designers to evaluate the system without having PCAM or other models in any lower level of abstraction. As depicted in Figure 2, TLM gives accuracy and speed at the same time. It runs several thousand times faster than PCAM while the accuracy is almost as good as that of board measurement.

Designers can optimize the design based on the estimation. That is manually done but introducing automatic optimization is not impossible. Note that, with having fairly high accuracy, an optimal design at transaction level may probably be a design good enough when it comes to the final implementation.

ESE back-end will not be covered in this report in great detail. We will just mention the followings. To answer the design challenge, widening productivity gap, an Electronic System-Level(ESL) design tool should automate the design process as possible. ESE back-end automates production of the final implementation, with having TLM as an input. The final implementation should include all RTLs of all the HW platform components. For that, some RTLs are given in the component libraries while custom HW, transducers and etc are generated by ESE back-end. RTOS and device drivers must be included in the final implementation as well. Some of them are stored in the libraries, while others missing in any of the library will be generated by ESE back-end.

An experiment has been done to measure design time saving of ESE and the quality of design obtained with ESE. Four different platforms for a MP3 decoder have been developed for the experiment. Every of them has one processor. Each of them has 0, 1, 2 and 4 custom HW, respectively. Every platform has been developed in two ways; manually and with ESE.

The results are depicted in Figure 3 and 4. As in the Figure 3, The design time to obtain PCAM was from 30 days to 60 days in case of manual development. On the contrary, ESE helps it to be reduced to about 10 days. Figure 4 two different things; utilization of chips and execution time to decode one frame. The design quality of ESE

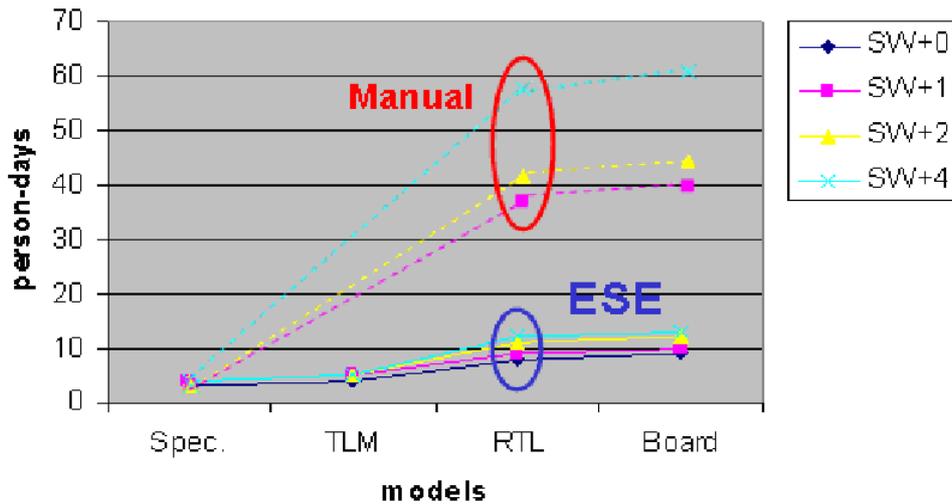


Figure 3: Design Time Saving ( source [1] )

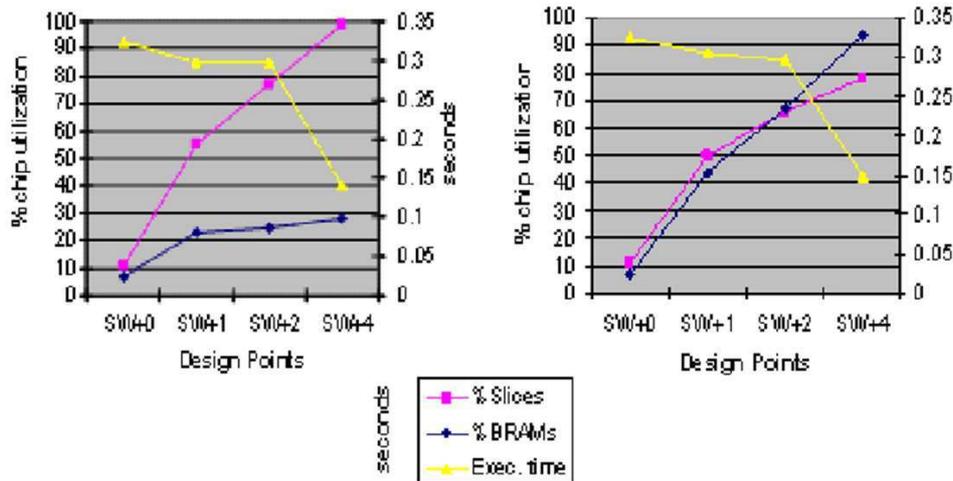


Figure 4: Design Quality ( source [1] )

is almost the same as that of manual development. In addition to that, ESE uses less FPGA slices and more FPGA BRAMs so that we can say ESE uses approximately same amount of HW resources to realize the platforms.

One of the core part of ESE, which the achievement comes true, is TLM estimation and its accuracy and speed. We will overview the TLM estimation technology of ESE in the coming sections.

### 3 TLM Estimation

This section presents a performance estimation technique for cycle-approximate TLMs, which is automatically generated to model heterogeneous MPSoC designs. Inputs are application C processes, platform and mapping between them. The output is timing-annotated TLM. By simulating and profiling the TLM, we obtain a performance

estimation in transaction level on the given design.

The delays can be grouped into three.

1. Delays from execution of the application
2. Delays due to RTOS activities
3. Communication delays

The first is due to execution of instructions in the applications. The second is caused by RTOS activities such as scheduling or interrupt services. The last one depends on communication such as data transfer or bus arbitration.

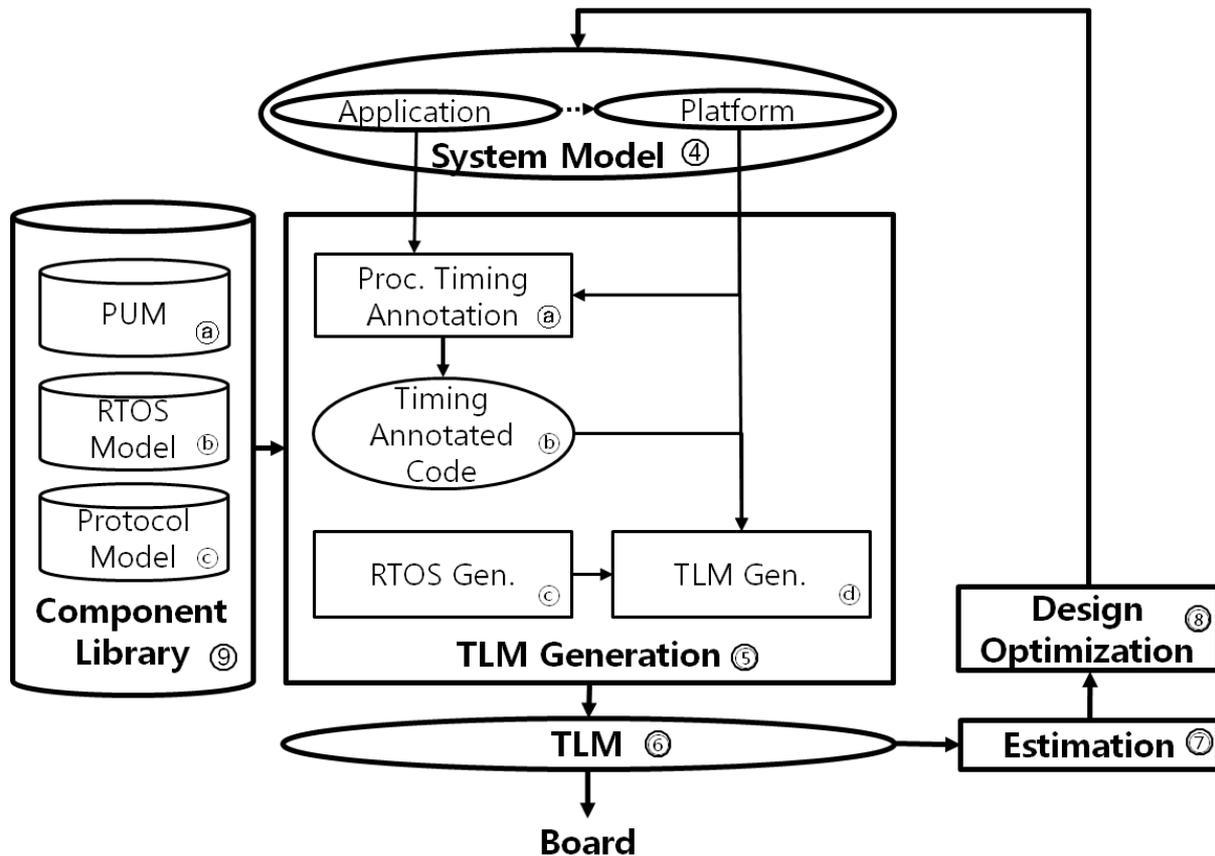


Figure 5: TLM Estimation Framework

Different techniques or tools depending on the categories of the delays are needed. TLM estimation framework depicted in Figure 5 shows that. For the delays due to application execution, the application in system model(4) is passed to the process timing annotator(5a) in TLM generator. Based on processing unit models(PUMs, 9a), which are abstract model of PEs, it produces the timing annotated application(5b), which is given as an input to TLM generator(5d). For delays caused by RTOS activities, RTOS models are introduced. They are inserted to TLM(6) by RTOS generator(5c) and TLM generator so that they do scheduling one or more processes on a single PE. In addition to computational delays, communication delays should be considered. For that, TLM generator inserts communicational delays into universal bus channel(UBC) models in TLM(6) based on protocol models(9c).

We will discuss every of them, in the following sections, respectively.

## 4 Application Estimation

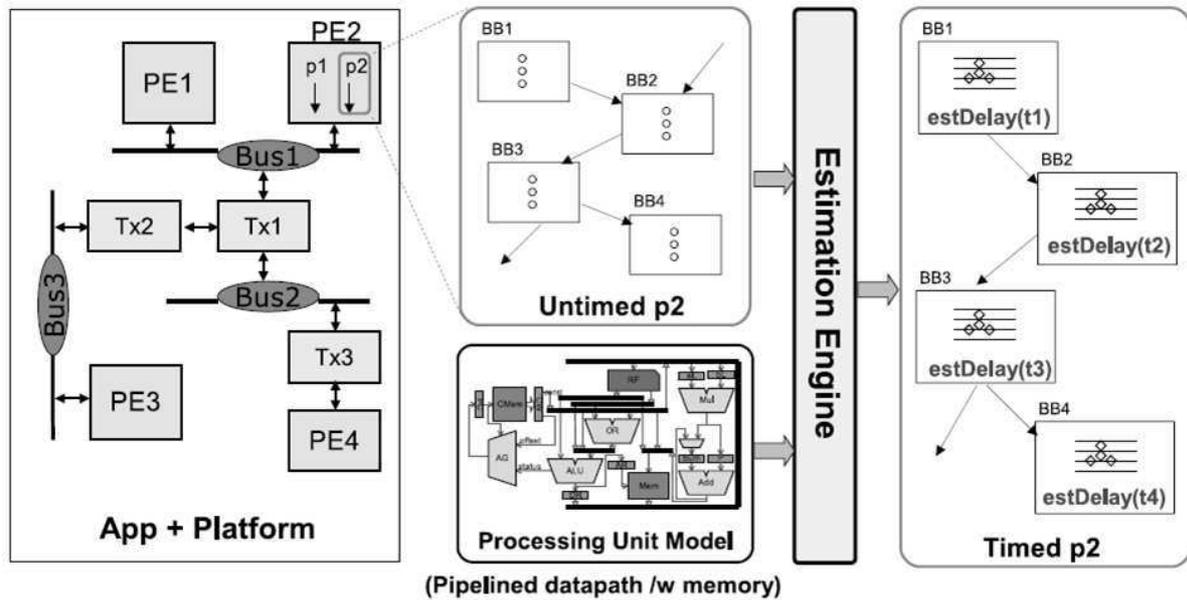


Figure 6: Application Estimation( source [4] )

Application estimation is described in the Figure 6. Each process is changed into a control-flow data graph. A node in the graph is a basic block. The process is mapped to a PE and a PUM for the PE is given. All instructions in each basic block are scheduled based on the PUM to get the estimated delay, which will be annotated at the end of the basic block.

The way in which the annotation process works will be discussed in the following two sections.

### 4.1 Processing Unit Model(PUM)

A PUM defines abstracted datapath and how instructions are executed in the datapath. For the purpose, a PUM consists of four models.

1. Datapath model
2. Execution model
3. Memory model
4. Branch delay model

Datapath model is a set of functional units, pipelines and a mapping table. Execution model consists of instructions and instruction mapping table. For example, execution model of microblaze defines which pipeline stages the add instruction passes, which functional units are accessed due to the add instruction during each pipeline stage and etc. Memory model specifies memory access latency. Branch delay model defines the cost of branch instructions.

A typical example of PUM is depicted in Figure 7. In the memory model, instruction/data cache hit ratio and miss penalty are present. Branch is predicted to be

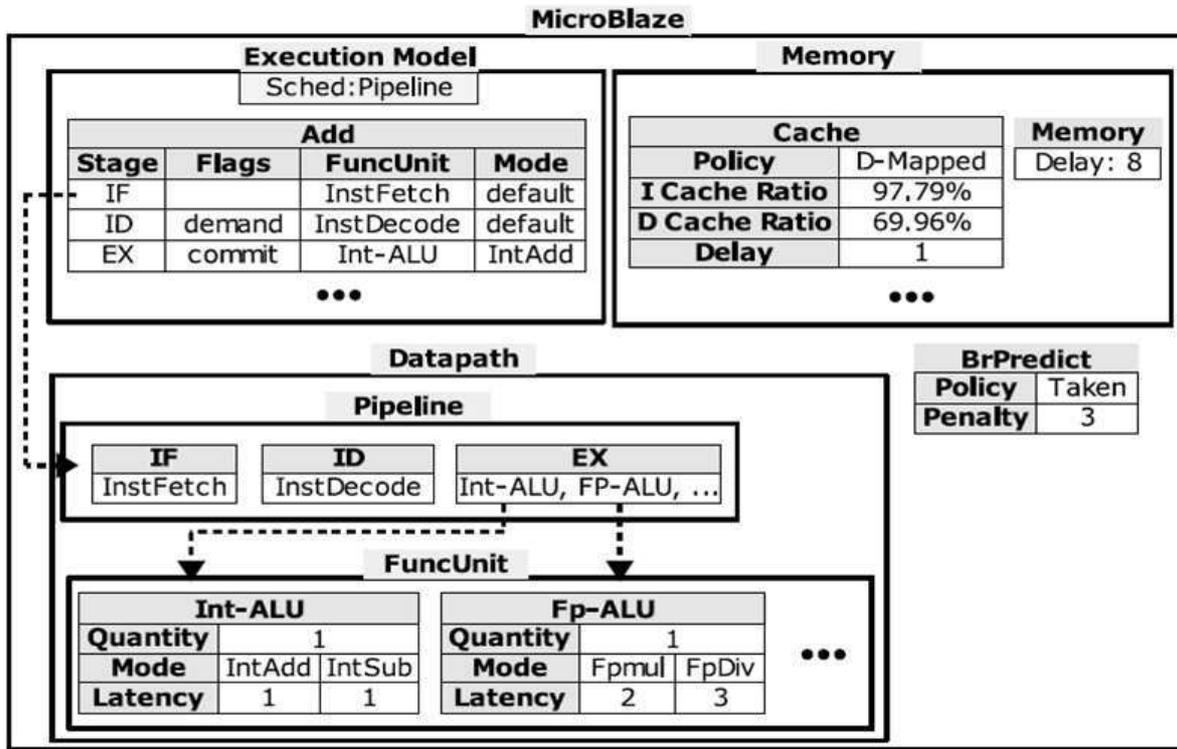


Figure 7: Example of Processing Unit Model - Microblaze Processor( source [4] )

always taken and the miss penalty is 3 cycles. Microblaze has 3 stage pipeline; IF, ID and EX. In addition to that, functional units and mapping between the units and pipeline stages are specified in the datapath model. For example, Fp-ALU unit for floating point arithmetic operations are mapped to the EX stage. In the example, the execution model defines instructions such as add. An add passes three pipeline stages, IF, ID and EX. On each stage, the functional units, InstFetch, InstDecode and Int-ALU are accessed respectively.

In the execution model, there is flags. In Figure 7, ID stage of add is marked as 'demand' and EX is marked as 'commit'. 'Demand' means, the stage has data dependency and might need stall. The 'commit' flag shows the data is committed so that the data is now available for different instructions depend on the data.

#### 4.2 Algorithms for Application Estimation

The goal of this section is to show the way that the estimated delay of a basic block is computed based on a PUM.

The delay is sum of the following two.

1. Delays when optimistic scheduling is assumed
2. Delays caused by branches and memory accesses

As addressed, the instructions in a basic block should be scheduled.

First, we schedule them optimistically. In other words, we schedule them without considering the costs of branches and memory accesses. Following the optimistic scheduling, we add the costs of branch instructions and memory accesses.

Algorithm 1 shows the optimistic scheduling algorithm.

---

**Algorithm 1** Optimistic Scheduling Algorithm( source [4] )

---

```

1: delay = 0
2: c_set =  $\emptyset$  // operations in pipeline
3: d_set =  $\emptyset$  // operations done
4: r_set = x | alloperationsinBB // remaining operations
5: while |d_set|  $\neq$  #ofBB do
6:   for all pipeline do
7:     d_set = d_set  $\cup$  advClock(pipeline)
8:     c_set = d_set
9:   end for
10:  for all pipeline do
11:    c_set = c_set  $\cup$  AssignOps(pipeline.stg1, r_set)
12:    r_set = r_set - c_set
13:  end for
14:  delay = delay + 1
15: end while
16: return delay

```

---

In brief, as far as all instructions are not done(line 5), we advance pipelines(line 6-9) as well as bring new instructions(line 10-13). However, how the pipeline is advanced, which is marked as *advClock*(line 7), is not clear in the algorithm 1.

That is explained in the Figure 8. An instruction can be moved to the next stage once it spends all of the designated cycles. Even in this case, there may be a data dependency in the next stage of the instruction. Before all necessary data are committed, the instruction cannot be moved toward. A data is ready when the instruction producing it has done its commit stage.

Unlike instruction-based estimations that do not care about data paths, the algorithm 1 does take datapath into consideration and is more accurate. In addition to that, with commit/demand flags, it is able to consider out-of-order execution that is popular in modern embedded processors.

However, the algorithm 1 has a missing part. The cost of branch and memory accesses are not considered yet. Therefore, the total estimated delay for a basic block is computed as in algorithm 2. In addition to the delay from optimistic scheduling, the costs of branches and memory accesses are considered.

As we have seen, the algorithm 2 uses a statistic model of memory access and can not take complex branch prediction mechanisms into consideration. Cache miss rate is given in a table based on statistics. That is a fairly accurate way with minimal cost.

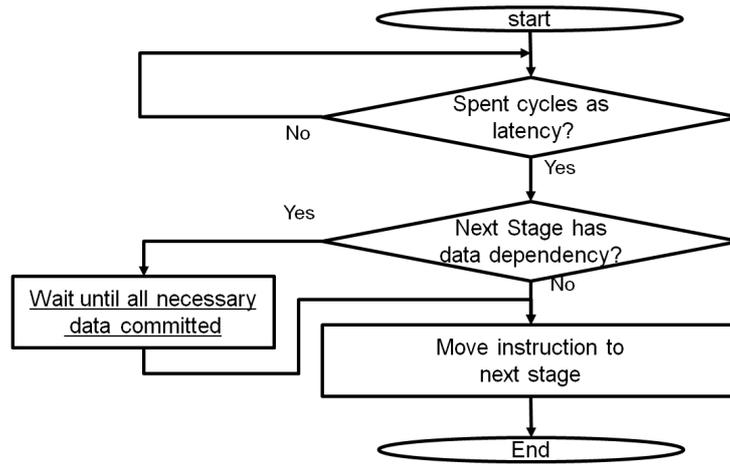


Figure 8: Control Flow of AdvClock

---

**Algorithm 2** Computation of Total Basic Block Delay( source [4] )

---

```

1:  $BB\_delay = OptimisticSchedule()$  //delay from optimistic scheduling
2: // add the costs of branches if pipelined
3: if is pipelined then
4:    $BB\_delay += BP\_miss\_rate * Br\_penalty$ 
5: end if
6: // add the costs memory accesses
7: if has i - cache then
8:    $BB\_delay += \# \text{ of } BB \text{ Ops} * (i\_cache\_miss\_rate * i\_cache\_miss\_penalty +$ 
       $i\_cache\_hit\_rate * i\_cache\_delay)$ 
9: end if
10: if has d - cache then
11:    $BB\_delay += \# \text{ of } BB \text{ Ops} * (d\_cache\_miss\_rate * d\_cache\_miss\_penalty +$ 
       $d\_cache\_hit\_rate * d\_cache\_delay)$ 
12: end if
  
```

---

However, cache miss rate may differ from application to application, from platform to platform and so on. Therefore, there is still a room for improvement. Revising branch model in PUM and providing more complex algorithms to deal with complex branch predictions in modern processors may be a good future work.

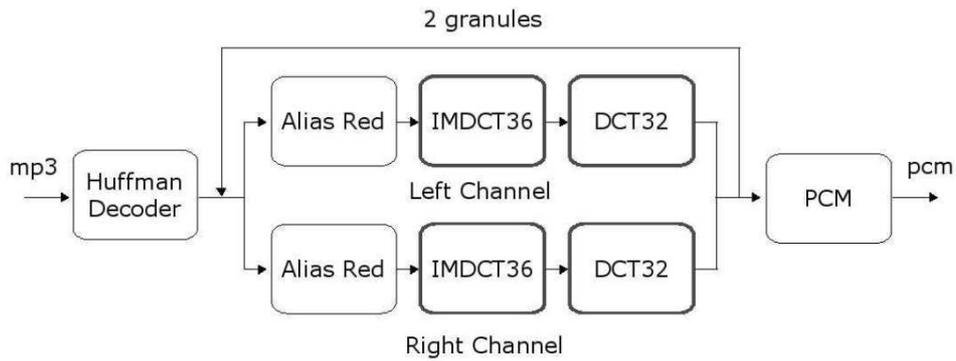


Figure 9: MP3 Application ( source [3] )

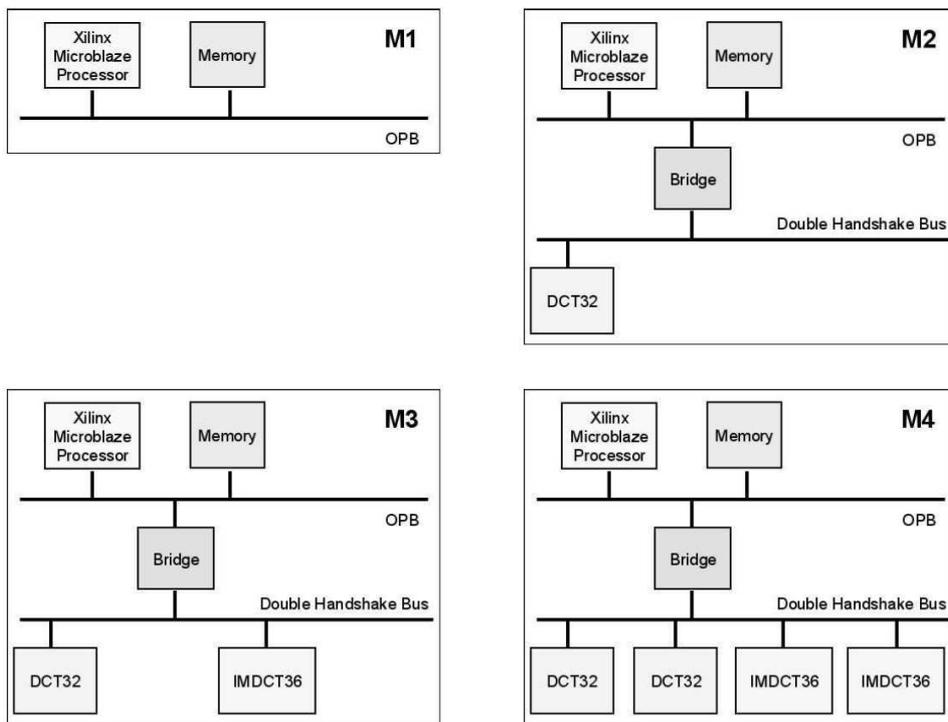


Figure 10: Four Designs for MP3 Decoder ( source [3] )

### 4.3 Experiment

First of all, all experiments in this section are cited from [4] and [3]. The application that was used is a mp3 decoder in Figure 9. The application is mapped to four different platforms depicted in 10. Every of M1, M2, M3 and M4 has a Microblaze processor. In addition to that, they has 0, 1, 2 and 4 custom HW respectively.

The experiments aim at a proof that application estimation is accurate while TLM generation and simulation is fast enough. The experiments come from [3] and [4]. In the Table 1, annotation and simulation time for the four different platforms are enumerated and compared to that of PCAM. According to the table, TLM simulation is thousand times faster than PCAM simulation.

Design	Anno.	Functional TLM	Timed TLM	PCAM
M1	31.181s	0.002s	<b>0.004s</b>	<b>15.93h</b>
M2	49.841s	0.006s	<b>0.216s</b>	<b>17.56h</b>
M3	47.364s	0.010s	<b>0.253s</b>	<b>17.71h</b>
M4	71.108s	0.012s	<b>0.355s</b>	<b>18.06h</b>

Table 1: Annotation and Simulation Time for Timed TLM ( source [4] )

I cache /D cache	Board Cycles	ISS		TLM	
		Cycles	Error	Cycles	Error
0k/0k	27.22M	16.47M	39.48%	25.51M	6.27%
2k/2k	8.91M	7.28M	18.38%	8.32M	6.68%
8k/4k	5.83M	5.62M	3.55%	5.55M	4.74%
16k/16k	4.41M	5.13M	-16.32%	5.02M	-13.83%
32k/16k	4.38M	5.11M	-16.60%	4.99M	-13.89%
<b>Average</b>	N/A		<b>18.86%</b>	N/A	<b>9.08%</b>

Table 2: Accuracy Result ( SW only, source [4] )

To compare accuracy of TLM application estimation to ISS estimation, SW only implementation, which is M1, is tested with five different memory configurations. The results are shown in the Table 2. While error ratio of ISS estimation on average is 18.86%, that of TLM estimation is only 9.08%. Even though ISS is inaccurate due to its inaccurate memory model, TLM is even more accurate than the ISS and the error rate is less than 10% on average.

I/D Cache Size	M2	M3	M4
0k/0k	9.00	18.18	18.61
2k/2k	-7.16	-15.79	-9.35
8k/4k	9.13	9.00	-0.18
16k/16k	4.66	2.63	3.65
32k/16k	-8.29	1.57	2.29
<b>Average</b>	<b>7.65</b>	<b>7.97</b>	<b>6.82</b>

Table 3: Error Ratio Compared to Board Measurement[%]( Platforms w/ Custom HW, source [4] )

Since the ISS is not the best reference and not available for the whole platforms including custom HW, in table 3, the results of TLM estimation on the platforms are compared only to those from board execution. The error ratio is 7.48%, which is less than 10%.

#### 4.4 Summary

In summary, application estimation provides more than 90% of accurate estimation at the expense of fairly small increase in simulation time, which is only a couple

of ten seconds depending on platforms. However, it is not yet completely perfect due to the followings. First of all, the delays due to memory accesses are estimated based on statistical information. The delays, though, vary depending on applications, platforms and mappings. Providing good statistics working for all the applications all over the world is impossible. Next, it does not consider optimization of compilers, which implies the improvement of optimizing compilers will be ignored. They can be future works.

## 5 Timed RTOS Model

### 5.1 Overview

Basically, concurrent processes cannot be executed at the same time on a single PE. There are limited resources on the PE so that it should be sequentialized by RTOS. Therefore, if we assume that all processes run in parallel on the same PE at the same time, then, it will be reduced to inaccuracy in estimation. However, the activities of RTOS are pretty common so that modeling RTOS is a sound choice. In addition to that, we put some estimation on every activities of each RTOS. That is the way RTOS estimator works.

Aside from execution of application processes themselves, there are two more delay contributors which RTOS has an impact on.

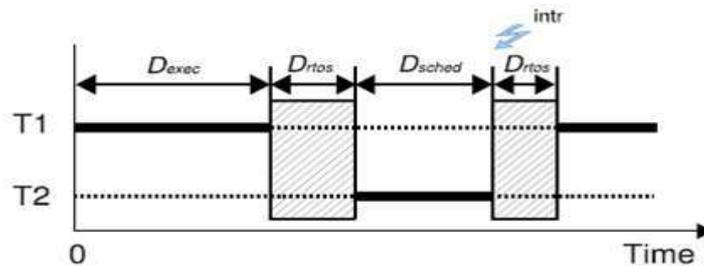


Figure 11: Three Computation Delay Contributors( source [3] )

Figure 11 enumerates the all of three delay contributors.  $D_{exec}$  is delay due to application processes' execution, which is not related with RTOS and has been explained in the previous sections.  $D_{sched}$  is a time to wait for the execution of different processes and caused by dynamic scheduling of RTOS.  $D_{rtos}$  is an overhead of RTOS tasks. Context switching of RTOS takes time and that is one example of  $D_{rtos}$ .

$D_{sched}$  depends on the scheduling policy of RTOS. RTOS models(9b, Figure 5) implement those scheduling policies and designers choose one of them during platform generation.

$D_{rtos}$  is different from RTOS to RTOS so that it should be stored in the database(9, Figure 5).

### 5.2 Experiment

In this subsection, an experiment is introduced to claim that timed RTOS modeling drastically enhances accuracy at the expense of a small increase in TLM simulation time. The experiment was done in the research that has been published in [5].

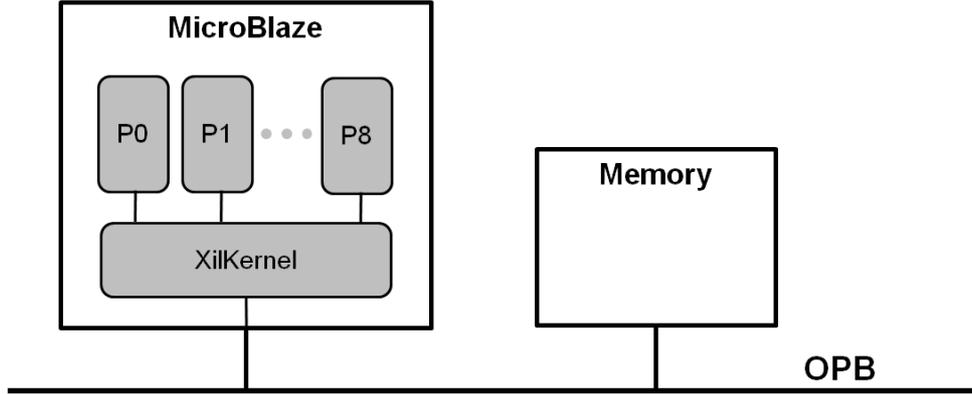


Figure 12: Test Architecture for JPEG+MP3 ( source [3] )

JPEG encoder and MP3 decoder are captured in nine concurrent processes and mapped to a single processor, Microblaze, as depicted in the Figure 12.

	Timed TLM	TLM w Untimed RTOS	TLM w/ Timed RTOS	XVP
<b>JPEG</b>	-75.00	-35.56	-9.98	50.00
<b>MP3</b>	-41.00	-25.95	5.29	7.00
<b>MP3+JPEG</b>	-83.00	-33.25	-6.20	37.00

Table 4: Accuracy of Abstract Models[%] ( source [4] )

Table 4 enumerates the error ratio. Three TLM models for the application and platform are generated; timed TLM without RTOS, timed TLM with untimed RTOS and timed TLM with timed RTOS. The first assumes that nine processes run in parallel. The second one has modeling of RTOS activities such as scheduling but, assumes that the activities themselves take no time. The last one is considering overhead of the RTOS activities in addition to that RTOS activity itself. They are compared to board measurement and Xilinks Virtual Platform(XVP), which is a commercial virtual platform. Without RTOS model, the error ratio was up to 83%. However, with untimed RTOS, estimation is improved a lot. Error rate is now from 25.95% to 35.56% but there still is a room for improvement. By considering RTOS overhead as well as RTOS activities, namely, by introducing timed RTOS, the error rate becomes less than 10%. TLM with timed RTOS is clearly better than XVP in accuracy.

	Timed TLM	TLM w/ Timed RTOS
<b>JPEG</b>	1.2	1.2
<b>MP3</b>	32.1	32.1
<b>MP3+JPEG</b>	33.3	33.4

Table 5: RTOS Generation Time[sec] (source [5] )

Introducing RTOS does not increase TLM generation time so much. RTOS models are stored in the library and TLM generator just brings them to put the models into TLM together. Table 5 exemplifies that. Compared to TLM without RTOS, the extra time that addition of timed RTOS needs is less than 0.1s.

	Timed TLM	TLM w/ Timed RTOS	XVP	Board
<b>JPEG</b>	0.02	0.27	168.00	0.83
<b>MP3</b>	0.01	0.08	60.00	0.34
<b>MP3+JPEG</b>	0.04	0.33	213.00	1.17

Table 6: Simulation Time[sec] ( source [5] )

There is a trade-off between improvement in accuracy and simulation time. The simulation time increases mainly since RTOS modeling adds more Inter Process Communication(IPC) on the simulation host. However, the increased simulation time, which is from 0.07s to 0.29s, is not a big deal as we can see in the Table 6. Compared to XVP, even the order of magnitude is different. The simulation time itself is faster than prototype board maybe due to the simulation host is much faster than the board although we need to consider that TLM needs additional generation time as well.

### 5.3 Summary

The increase in simulation time due to addition of timed RTOS is small. However, the improvement in accuracy is so large that the error ratio can be reduced from 83% to 6.20%. We can conclude that timed RTOS modeling gives a good accuracy at the expense of small overhead in TLM generation and simulation time. However, we can address the following drawbacks. It does not fit preemptive RTOS perfect although a number of important commercial operating systems for embedded systems are preemptive. ESE RTOS model does not allow any context switching inside basic block unless application processes yield by explicit call of RTOS services. In addition to that, for a certain but common development environment, the level of abstraction might be too high. For example, chip vendors such as Samsung, LG, Qualcomm and etc, need a fast, virtual platform, on which they can develop system SW for a specific processor(such as ARM's) and specific operating systems. ESE RTOS model does not seem to be the fittest.

## 6 Communication Estimation

In this section, we will present how communication delays are estimated in transaction level. Figure 13 is explaining that.

Communication delays are not considered during application estimation or RTOS modeling. The goal of this section is to take them into consideration.

Fortunately, in transaction level, buses are described as common bus functions. Each of these functions are timing-annotated depending on protocol model of the bus in concern.

For example, bus1, bus2 and bus3 in the figure provides a set of common bus functions such as Get\_Bus, Transfer and Release\_Bus. We will annotate timing each

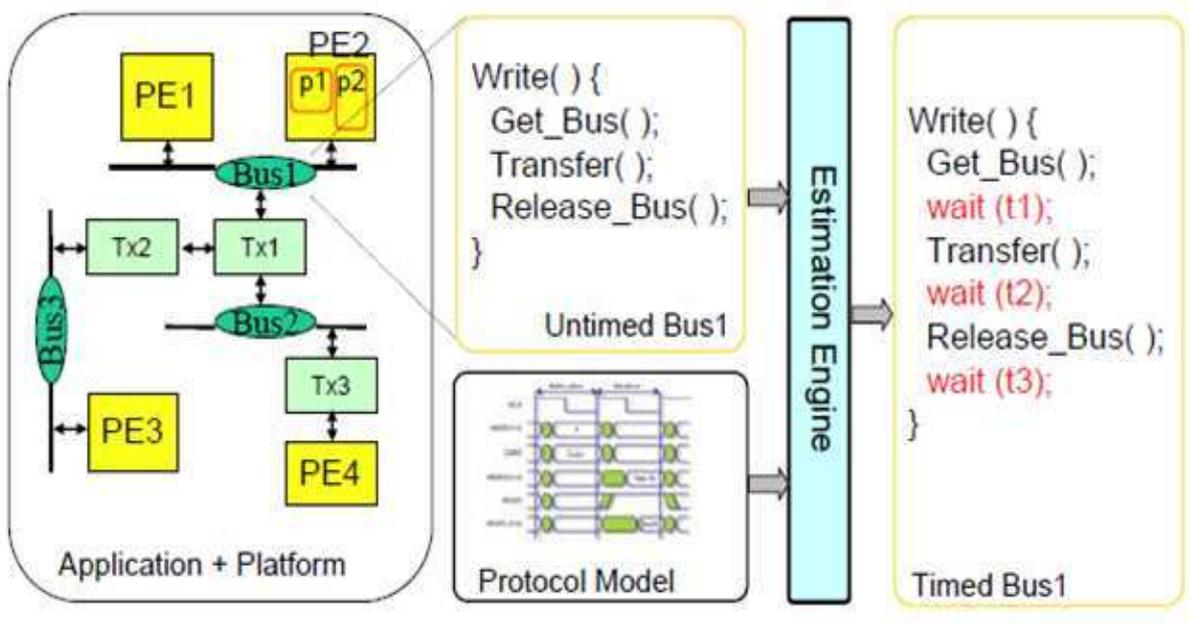


Figure 13: Communication Estimation ( source [2] )

of these functions. However, the delay value is different from bus to bus. That's what protocol model is for. For example, depending on the protocol model of bus1, we get  $t_1$ ,  $t_2$  and  $t_3$  as the estimated delays of the three functions, respectively.

In addition to that, in ESE, all buses are modeled as Universal Bus Channels(UBCs). It implies that the bus functions are not different between buses. The only difference is the values of the estimated delays.

These estimated delays are inserted at the end of each UBC functions. During simulation, the communication delays can be profiled.

An individual transaction delay,  $T_{total}$ , is the sum of the following three [6].

1. Synchronization delay,  $T_{sync}$
2. Arbitration delay,  $T_{arb}$
3. Data Transfer Delay,  $T_{dt}$

Synchronization delay is time to wait for the communication partner ready. For example, to make a synchronized transaction with p1, process p0 might have to wait for p1 ready. That is synchronization delay. Arbitration delay is caused by bus contention. Data transfer delay includes cycles of control/address phases as well as data transfer phases.

In the following sections, we will discuss how each of the estimated delays is computed.

### 6.1 Synchronization Delay and Arbitration Delay

Communication estimator does nothing to compute  $T_{sync}$ . SystemC simulation kernel takes care of it.

Arbitration delays depend on the bus arbitration policy. UBC implements one of the arbitration policies in the component library(9, Figure 5).

## 6.2 Data Transfer Delay

$$T_{dt} = CA \cdot S + \lceil \frac{S \cdot 8}{B_{width}} \rceil \cdot D_{bus} \quad (1)$$

Data transfer delay is defined as equation 1, where the unit is number of cycles [6].  $CA$  is control/access phase length in cycle.  $S$  is the size of transaction in byte. If the bus supports burst mode, this  $S$  is always 1 regardless of the size of transactions in byte.  $B_{width}$  is data bandwidth of the bus in bits. That is why 8 is present in the formula.  $D_{bus}$  is a bus transaction delay in cycle per single transaction.

The question is how those values are obtained. That will be answered in the next section, section 6.3.

## 6.3 Bus Delay Database

Protocol models should include database to give hints for the values such as  $CA$ ,  $S$ ,  $B_{width}$  and  $D_{bus}$  introduced in section 6.2 to be obtained. What the database should have can be grouped into two categories.

1. Bus properties from data sheet
2. List of bus delay values in multiple setups

Bus properties are exemplified by data bandwidth, bus address space, arbitration options and data transfer options such as availability of burst mode.  $CA$  and  $B_{width}$  are determined depending on the bus properties.

The database must include bus delay values in multiple setups. The values are different in bus protocol, memory location(external or internal), type of memory accessed and etc.  $D_{bus}$  is a delay for a single transaction. It is determined by considering bus delay values in the setup.

$S$  is the amount of the data and depends on the application. All of the values in equation 1 can be obtained by referring bus delay database.

## 6.4 Experiment

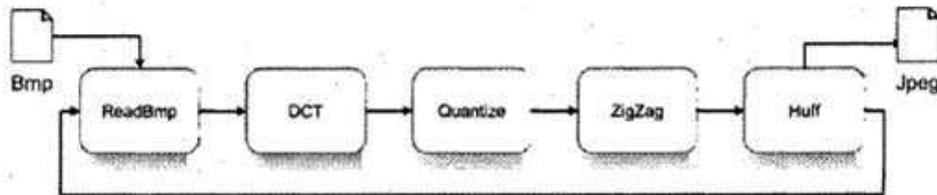


Figure 14: JPEG Application ( source [6] )

The experiment comes from [6]. The application is JPEG encoder depicted in Figure 14. Seven processes are communicating to encode a bmp to a JPEG.

The application is mapped to four different platforms. The results are depicted in Figure 15. Communication delays are estimated and also measured on the board.

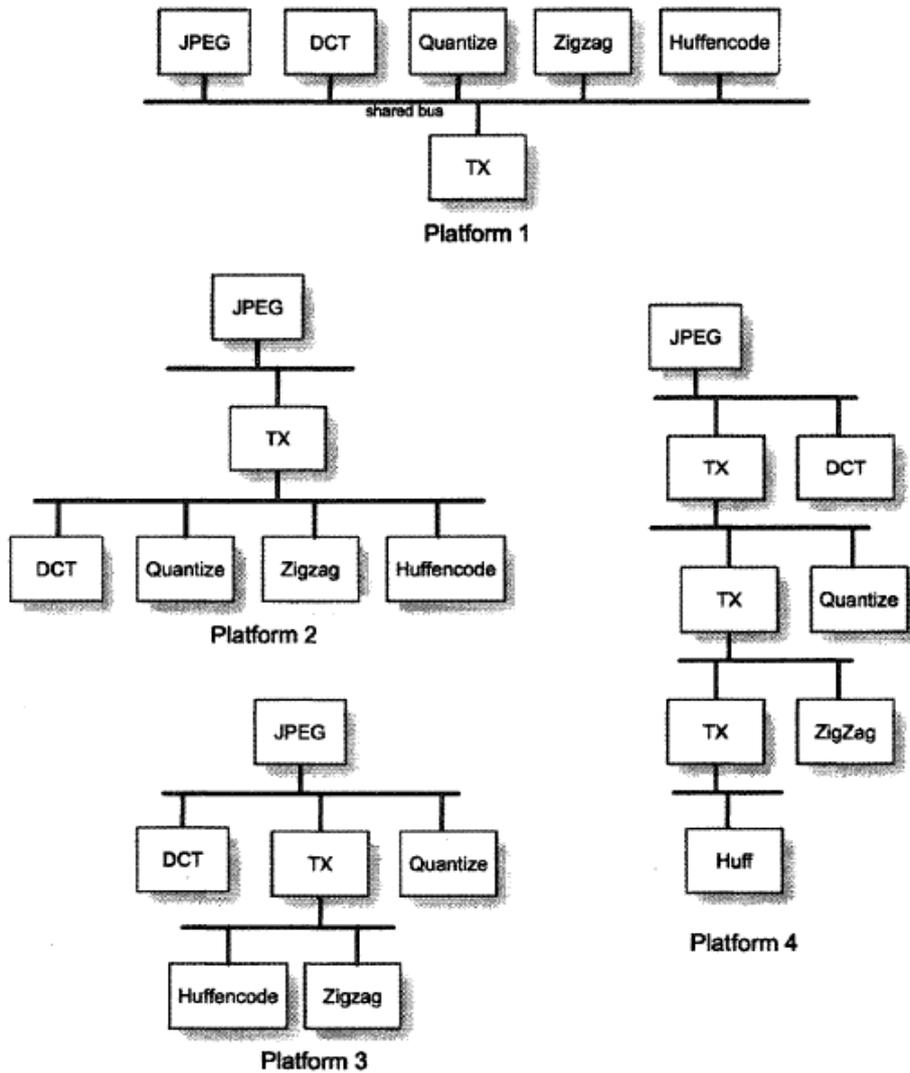


Figure 15: JPEG Platforms ( source [6] )

Communication Channel	Error Ratio
BMP to DCT	1%
DCT to QUANTIZE	2%
QUANTIZE to ZIGZAG	5%
ZIGZAG to HUFFMANCODE	6%

Table 7: Error Ratio of TLM Compared to Board Measurement ( source [6] )

Table 7 shows the results of the experiment. Compared to board measurement, the error ratio of the estimation on the delay for each of the four communication channels is from 1% to 6%, which is relatively small. [6] does not mention about simulation time and generation time. However, it is included those of the experiments in section 4.

## 6.5 Summary

As far as the experimental results are true, categorization of bus functions and delay models for them seem to be sound. The error ratio, which is equal or less than 6%, is low enough and the cost is not that large.

## 7 Conclusion

For a system design methodology to answer to the challenge in the modern system designs, widening productivity gap, the design process needs to have an universal specification of a given system. Further, based on the system specification, verification of functionalities and validation of user requirements should be able to be done and the final implementation must be obtained by almost automated processes.

ESE is an example. The universal specification of the system is application, platform and mapping. The level of abstraction are raised to transaction level. Since PCAM is automatically generated from TLM and TLM generation is also automatic, the design process in ESE is highly automated. In addition to that, Designs can be evaluated and optimized at transaction level not at pin-cycle accurate level or on board. TLM estimation is fast and accurate at the same time. Different results shows that the error ratio is less than 10% while the simulation speed is a couple of thousand times better than PCAM and TLM generation time is just a couple of ten seconds.

From those reasons, ESE helps designer to save design times more than 70% while keeping the design quality almost the same or even improving. That is why ESE, as far as its implementation is sound, can be an answer to the most demanding design challenge in our times, productivity gap.

In section 4.4, 5.3 and 6.5, we described what is missing in what ESE claims to have. Application estimation does not have accurate cache model and branch delay model. Timed RTOS model is inaccurate when a preemptive kernel is in concern. Application model is claimed to be general enough [1] but it is not since there is neither hierarchy nor state transition.

Those are what are still needed even if implementation of ESE were perfect. However, the implementation is not perfect so that there is a room for improvement as well.

First, TLM components should be re-implemented to work correctly in their functionalities. All of the components except PE and RTOS works only with limited range of platforms and setups. For example, it usually works if the platform has two buses and one transducers but does not when three or more buses are introduced in the platform.

Second, the common parts in instances in each TLM modules should be moved to a library and shared. For example, two PE instances, PEO and PE1 have lots of common codes. It is because they are different only in processes mapped to each of them and the number of bus ports. However, current implementation duplicates every single lines of code. Therefore, we have to extract the common parts and put them to a SW library.

Third, TLM generator needs to be modularized. A single file has 8K lines. A single function has 3K lines of code. That is not a good SW practice.

Fourth, ESE Data Structure(EDS) should be revised to make design space exploration easier. Application, platform and mapping are messed up in a single data structure. As a result, for example, we cannot specify any process without a PE. Also,

removal of a PE needs too many updates in the structure.

All of those needs to be done to improve ESE implementation aside from the new features that we may want to add to ESE.

## 8 Acknowledgments

First of all, I would like to thank my advisor, professor Daniel Gajski. This report would not even be able to be written without his guidance. I also thank Quoc-Viet Dang. Without his help on my writing, it would not be in a good shape.

## References

- [1] Embedded system environment. <http://www.cecs.uci.edu/~ese>, Dec., 23 2011.
- [2] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design: Modeling, Synthesis, Verification*. Springer, 2009.
- [3] Y. Hwang. *Transaction Level Model Based Performance Estimation and System Generation*. Ph. D. Dissertation, University of California Irvine, 2009.
- [4] Y. Hwang, S. Abdi, and D. D. Gajski. Cycle-approximate retargetable performance estimation at the transaction level. In *DATE*, Munich, Germany, April 2008.
- [5] Y. Hwang, S. A. G. Shirner, and D. D. Gajski. Accurate timed rtos model for transaction level modeling. In *DATE*, 2008.
- [6] L. Yu. *Automatic Generation and Verification of Transaction Level Modeling*. Ph. D. Dissertation, University of California Irvine, 2009.