

Module 57 - Lab A: Cost Modeling for System Design

Applications of Cost Modeling to Embedded Digital Systems Design

Copyright 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained, may be duplicated for non-commercial educational use only provided this copyright notice and the copyright acknowledgements herein are included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds "Unlimited Rights" in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this notice .

See the [RASSP Disclaimer file](#) for additional RASSP Disclaimer, Warranty and Limitation of Liability Information concerning the material, VHDL code and software developed under the RASSP programs or incorporated in RASSP material.

1. Overview

The objective of this lab is to demonstrate applications of cost modeling to the embedded-system design. The lab discussion starts by describing the use of cost models for estimating software life cycle costs. Then, the impact of time-to-market on architecture design is described. The effect of these cost elements on system design is demonstrated in the design of a simple compute-intensive application.

2. Cost Modeling

A parametric cost model is a mathematical representation of parametric cost estimating relationships (CERs) that provides a logical and predictable correlation between the physical or functional characteristics of a system and the resultant cost of the system. Such parametric cost estimating relationships are derived from the statistical correlation of historical system costs with performance and/or physical attributes of the system. When performing cost analysis, the primary purpose is not necessarily to use these models to produce precisely accurate point estimates, but rather to provide estimates of the comparative and relative costs of competing systems. Hence, the consistency of the model is perhaps more important than the absolute accuracy of its estimates. Parametric cost models are particularly useful because they are objective, repeatable, efficient, and they facilitate sensitivity analysis. If derived from a sound representative database, these parametric models can be relied upon to produce high quality, consistent estimates.

Parametric techniques focus on major cost drivers, and not minute details. The cost drivers are the controllable system design or planning characteristics that have a predominant effect on the system's costs. This allows parametric techniques to be used with limited system information, thereby making them a viable option in the conceptual stages of design. The two dominant costs elements which are traditionally neglected in COTS-based embedded systems design are hardware/software life cycle costs and time-to-market costs. For this reason, we will focus on the estimation of these two cost factors.

2.1. Software Cost Modeling

Many parametric software cost estimators have been developed since the late 1970's. Current commercial software cost estimation tools include COCOMO 2.0, PRICE-S, SEER-SEM, and REVIC. Most of these tools use algorithms, and some of the more advanced tools such as SEER are rule-based or knowledge-based as well as interactive. We refer to these tools interchangeably in this paper to emphasize the flexibility of our methodology.

Software cost estimators primarily consist of a core or nominal effort equation which relates the labor effort for developing software to the size of the software system. This nominal

effort equation represents the cost of developing a software system 'in a perfect world'. Effort adjustment factors are applied to the nominal estimate to adjust for organization and project specific economic factors. For example, the PRICE-S Model can be viewed as an onion, with a core surrounded by several layers. At the core of PRICE-S is a central equation that relates the labor effort for developing software to the volume of the software system, metered by the productivity of the organization performing the task. The core labor estimate represents a 'normalized' cost. When moving from the model's inner core to the reality of the outside world, CERs may be applied that adjust the normalized labor estimate based on the economics and complicating factors of the specific project.

Although many of these tools use very different parametric CERs, they all make similar claims about how the design can affect the software development cost. In all cases, historical data has shown that increased development cost and time can occur as a result of squeezing more and more functionality in smaller and smaller space and time intervals. REVIC, COCOMO 2.0, and PRICE-S assume that resource requirements of less than 50 percent capacity have no cost impact. But as the utilization of system resources increase above this level, the models assume that there will be more detailed design, less reliance on tools and high level languages, and increased integration and test requirements. As resource utilization approaches 100 percent, the cost impact is extreme.

To quantify this software prototyping principle, the embedded mode REVIC software development cost model is presented. In the REVIC model, the development cycle includes the contract award through hardware/software integration and testing. The development effort and cost equations can be written as follows:

$$s_C = C^S s_E$$

s_E is the software development effort in person-months. The software development

$$s_E = A(KSLOC)^B f_E f_M \prod_{i=1}^{17} f_i$$

cost, s_C , is the product of the software development labor cost per person-month, C^S , and the software development effort. The f_i 's represent cost drivers which model the effect of personnel, computer, product, and project attributes on software cost. The constant, A , captures the

multiplicative effects on effort with projects of increasing size. The default value used in REVIC for A is 4.44. The scale factor, B , accounts for the relative economies or diseconomies of scale encountered for software projects of different sizes. The default value used in REVIC for B is 1.2, representing a diseconomy of scale. The effort adjustment factors, fE and fM , denote the effect of the execution time margin and storage margin on development cost. The relation between these effort adjustment factors and the CPU and memory utilizations is shown in Figure 1. As the hardware utilization increases, so do the values of the execution time and main storage constraint multipliers. This increase causes a corresponding increase in the development cost and time.

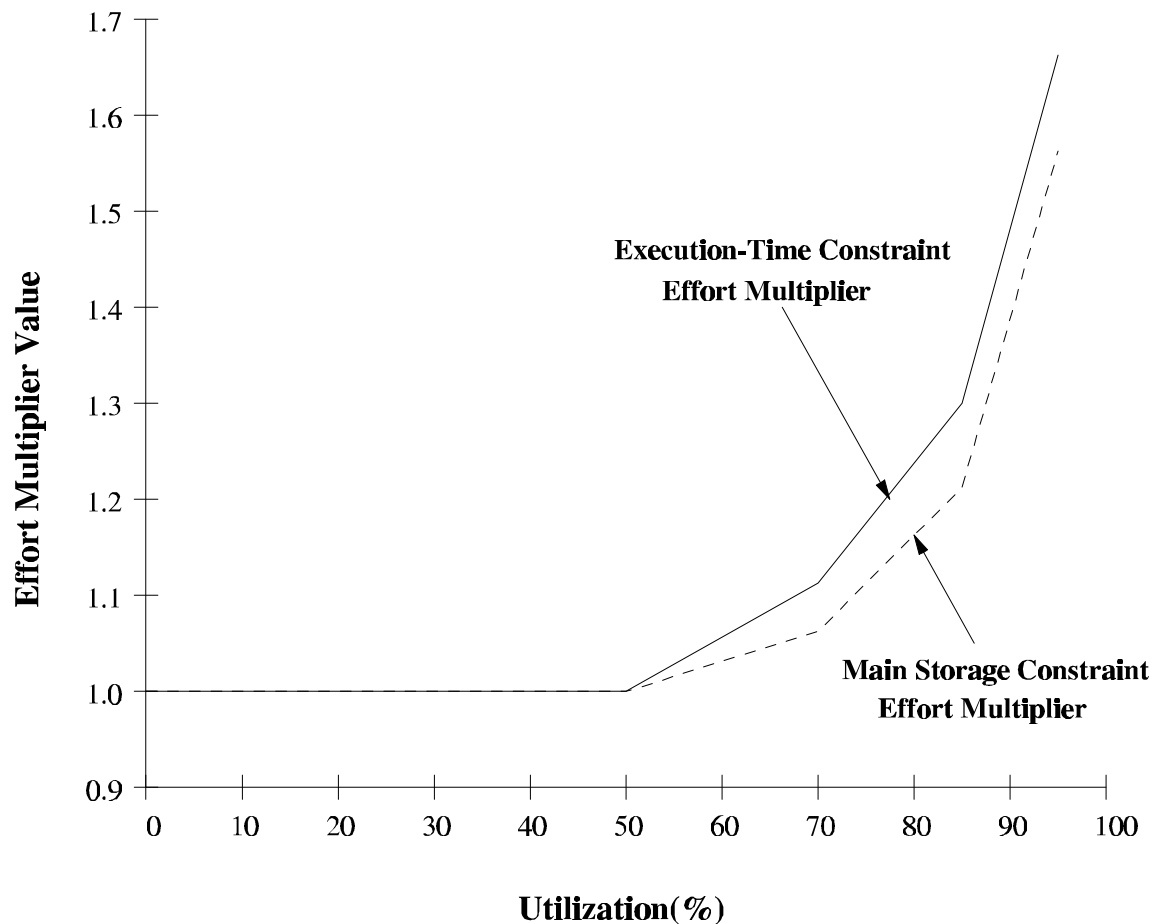


Figure 1. Execution-time and main storage constraint effort multipliers

The primary input to the software development (since hardware is COTS and is not developed within the design cycle) cost equation is the software size estimate, $KSLOC$. $KSLOC$

denotes the equivalent number of source lines of code (thousands) including application code, OS kernel services, control and diagnostics, and support software. To compute the effective software size of a product, the size estimate must be adjusted for reuse of COTS software. Software size estimates are composed of two parts, the number of new source lines of code and the number of adapted source lines of code. COCOMO 2.0 uses a variation of the following model to estimate the equivalent number of new lines of code:

$$KSLOC = KNSLOC + KASLOC \cdot \frac{(AA + SU + 0.4 \cdot DM + 0.3 \cdot CM + 0.3 \cdot IM)}{100}$$

The symbols in the equations are defined in Table 1. Also, additional refinement to the size estimate is necessary to account for the costs of software re-engineering and conversion. Methods for modeling these costs are described in the *COCOMO 2.0 Reference Manual*. Some overall software sizing techniques include the Pert Sizing, the Wideband Delphi technique, function point sizing, and sizing by analogy. Software size serves as the dominant cost driver for parametric software cost estimators.

Table 1 . Software Sizing Model Symbol Definitions

Symbol	Description
KNSLOC	Size of component expressed in thousands of new source lines of code
KASLOC	Size of the adapted COTS software component expressed in thousands of adapted source lines of code
AA	Degree of assessment and assimilation of COTS software
SU	Software understanding penalty and interface checking penalty
DM	Percentage of design modified
CM	Percentage of code modified
IM	Percentage of integration and test modified

Tight resource constraints have a similar effect on software maintenance cost. Many of the models assume that software maintenance costs are determined by substantially the same cost driver attributes that determine software development costs. For example, REVIC models the annual software maintenance effort as:

$$m_E = ACT \left(A(KSLOC)^B f_E f_M \prod_{i=1}^{16} mf_i \right)$$

The maintenance costs are determined by multiplying the maintenance effort by the software maintenance labor cost per person-month. The mf_i 's refer to the maintenance effort adjustment factors for which are primarily the same as those used in the software development cost equations. ACT is the annual change traffic. This metric corresponds to the fraction of the software product's source instructions which undergo change during a typical year, either through addition or modification. $KSLOC$, A , B , f_E , and f_M are defined as before, and thereby, have the same effect on maintenance costs as on development costs.

When faced with the prospect of long development schedules which will cause a product to be delivered to market late, many managers attempt to compress the schedule by throwing more people (e.g., person-hours) at the problem. Most of the parametric software cost estimation models show that schedule compression sometimes has the adverse effect of greatly increasing development cost. This increase in cost is usually due to the larger labor force, which in turn increases communication problems, thereby adding errors and inefficiencies within the team. REVIC models this effect of schedule compression by using a schedule constraint effort multiplier to scale the software development effort as shown in Figure 2. The REVIC development time equation can be written as follows:

$$s_T = C(s_{Enom})^D \bullet \frac{SCED}{100}$$

where s_T denotes the software development time in calendar months. s_{Enom} signifies the estimated development effort under nominal schedule constraints (no schedule compression/expansion). The constant C captures the multiplicative effects on development time for projects of increasing

effort. REVIC's default value for C is 6.2. The scale factor D accounts for the relative economies or diseconomies of scale encountered for software projects of different required efforts. REVIC's default value for D is 0.32, representing an economy of scale. $SCED$ is the percent compression/expansion to the nominal deployment schedule. Figure 2 illustrates that the REVIC model assumes that the nominal schedule cannot be compressed by more than 25 percent. Most parametric software cost models agree that there is a limit on the amount a schedule can be compressed. Furthermore, the amount of schedule compression is also constrained by the number of available full-time software personnel, which can be quantitatively described as follows:

$$F_{sp} \geq \frac{s_E}{s_T}$$

When the hardware platform consists of mostly COTS components, long software development times will dominate the overall system development time. Thus, software development time will be the major factor determining time-to-market for the HW/SW product.

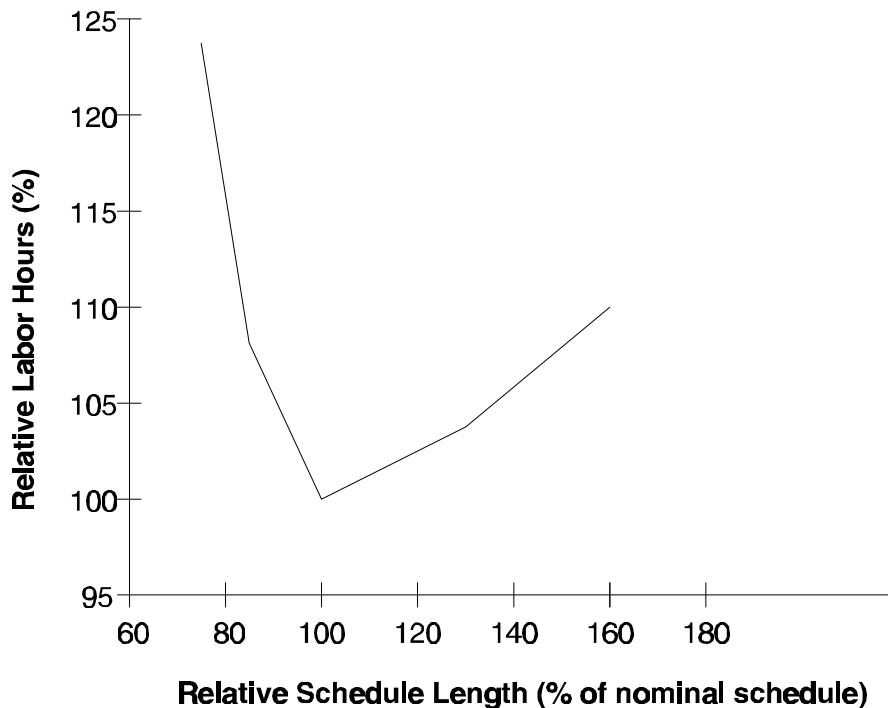


Figure 2. The Cost Impact of Schedule Constraints

2.2. Time-to-Market Cost Modeling

While the implications of constraining the HW/SW architecture can be very detrimental to software development cost, the corresponding effect on development time can be even more devastating. For commercial products, time-to-market costs can often outweigh design, prototyping, and production costs. A recent survey showed that being six months late to market resulted in an average of 33% profit loss. Engineering managers stated that they would rather have a 100% overrun in design and prototyping costs than be three months late to market with a product. Early market entry allows for increased brand name recognition, market share, and product yields.

There have been numerous attempts at quantitatively modeling the effect of delivering a product to market late. As previously mentioned, market research performed by Logic Automation (now owned by Synopsys) has shown that the demand and potential profits for a new HW/SW product can be modeled by a triangular window of opportunity as shown in Figure 3. The non-shaded region of the triangle signifies the lost of revenue due to late entry in the market. This loss in revenue can be mathematically stated as follows:

$$r_L = R_0 \cdot d^+ ((3W - d^+) / (2W^2))$$

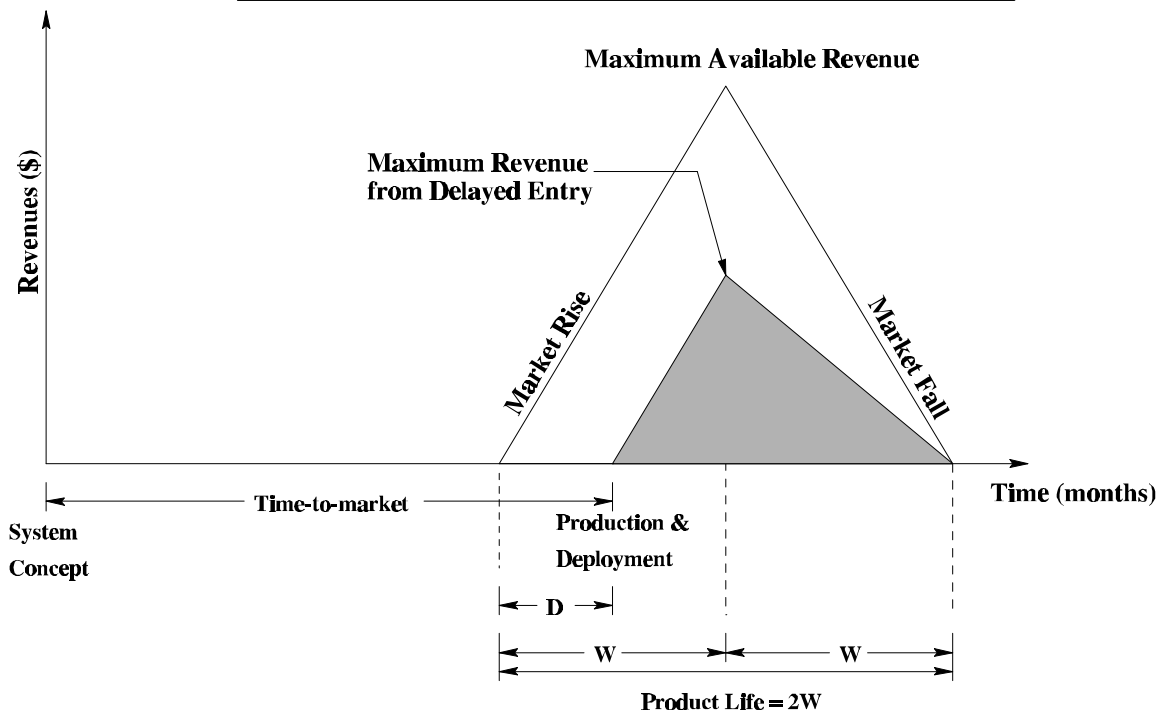


Figure 3. A typical time-to-market cost model

R_0 refers to the expected product revenue. d^+ is the delay (months) in delivering a product to market, and W is half the product life cycle (months). If the product life cycle is short, being late to market can spell disaster.

3. Design Experiment

Our goal for this laboratory exercise is to demonstrate the use of cost modeling in the architectural design an embedded platform. Figure 4 shows the data flow graph (DFG) for the simple application. Each node in the DFG is annotated with its execution time on the generic processor used in this lab. Also, each edge in the DFG is annotated with the communication volume which is transferred between the communicating nodes during each iteration. The combination of the application code and support software is estimated to account for 5000 lines of source code (KSLOC = 5). Data is input to the algorithm at 50 samples/sec. Hence, the real-time constraint on each task (node) is 20 ms.

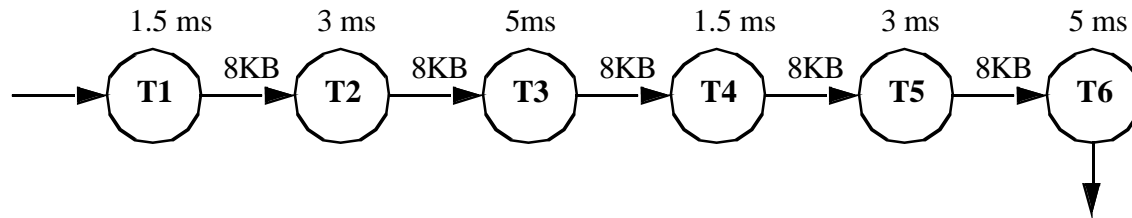


Figure 4. Data flow graph for simple embedded application

Case 1: In this case, the entire application is implemented on one processor. We estimate the costs for this simple assignment. If the resource utilization is very high, then the cost of such a design in terms of person months and schedule can be very high.

Case 2: In this case, we subdivide the application into two portions, each of which runs on a separate processor, and the processors are interconnected via a communication pathway (in this case a crossbar chip).

One concern in Case 2 is that if the communication subsystem were not fast enough for the application, then the real-time guarantees required by the application may not be satisfied. In addition, the overhead caused by interprocessor communication can increase the processor utilization, thereby increasing the software cost and schedule. Therefore, this close coupling between cost and performance modeling is needed to complete the laboratory.

3.1. Nominal Software Cost Estimation

The REVIC model will be used to estimate the cost and schedule for the various HW/SW architectural candidates. The application requires 5,000 lines of source code. Thus, the nominal software development (assuming all effort multipliers equal 1.0) effort can be computed as follows:

$$\text{Effort} = 4.44 (5)^{1.2} =$$

We wish to adjust now for the execution-time resource constraint (f_E), which can be calculated from a description of the real-time performance of the algorithm.

3.2. Adjusted Software Cost Estimation

3.2.1. Case 1

For Case 1, we assume that the execution-time for one iteration of the algorithm can be computed by summing the computation times for each task in the DFG. As previously mentioned, the real-time constraint for each task is 20 ms. Therefore, the processor utilization is:

$$(1.5 + 3 + 5 + 1.5 + 3 + 5) / 20 = 0.95$$

or 95% processor utilization

From the REVIC table (in the Cost Modeling Module) we obtain the execution-time resource constraint effort multiplier to be:

$$f_E =$$

Cost (adjusted) for the software development is $= 30.6 * 1.66 = 50.8$ person months.

Cost in dollars (assuming \$15,000 per person month) =

The development time (assuming a nominal schedule) is:

$$s_T = 6.2 (50.8)^{0.32} = 21.8 \text{ months}$$

3.2.2. Case 2

For Case 2, let us assume that we want to reduce the cost of the design, so we try to reduce the execution-time effort adjustment factor by using two processors interconnected by a communication pathway (a crossbar). If communication time is assumed to be zero, the aggregate utilization is halved. However, the interprocessor communication overhead can have a dominant effect on performance when tasks are poorly allocated to processors or poorly scheduled.

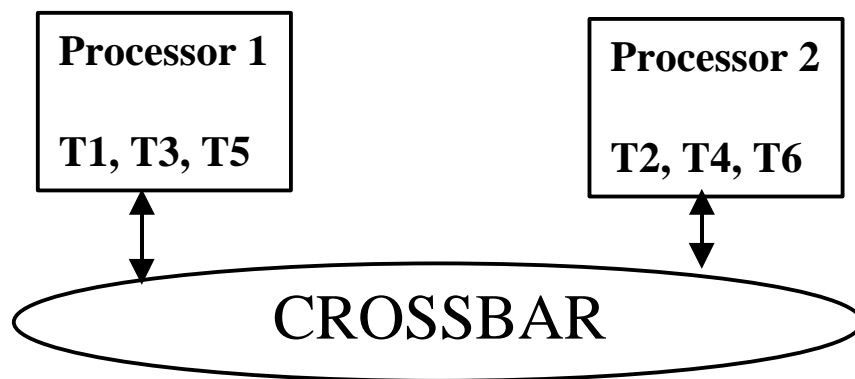


Figure 5. Multiprocessor architecture with poorly allocated tasks

Verification of the Cost-Driven Architecture Design (Case 2a)

The single processor implementation uses benchmarked execution times for performance estimates and perform no interprocessor communication. Therefore, simulation-based performance modeling is not needed for Case 1.

However, Case 2 is a multiprocessor implementation which requires interprocessor communication. Figure 5 shows a HW/SW architecture implementation for the example application. VHDL performance modeling will be used to estimate the overhead associated with communication and to ensure that the processor implementation meets real-time constraints.

The makefiles and libraries used in this exercise were developed for use with the Mentor Graphics QuickVHDL tools on a UNIX-based platform. To get started, create a directory for the performance modeling section of the lab in your home directory as follows

```
>> mkdir PF_LABS  
>> cd PF_LABS
```

The file "m57_lab_a.tar" will be made available with your labkit and it should be copied to this performance modeling directory. To uncompress and untar the "m57_lab_a.tar" file type:

```
>> tar -xvf m57_lab_a.tar
```

The performance modeling code for our example embedded application is now stored in your directory. The "m57_lab_a" directory has a number of subdirectories and files. They are described as follows:

- m57_lab_a/COMPONENTS/ - contains the RACEway crossbar and compute element processor models, xbar.vhdl and ce.vhdl.
- m57_lab_a/PACKAGES/ - contains the packages with define the communication token structure and FIFO elements.
- m57_lab_a/PROGRAM?/ - contains programs files used to configure the processor architectures. For example, p1_cost.dat is used to program processor 1.
- m57_lab_a/ROUTES/ - contains the route files used for routing of communications between each processor in the system.
- m57_lab_a/SYSTEM/ - contains the top-level structural VHDL performance modeling code for the two processor architecture.
- m57_lab_a/RESULTS?/ - contains output files of pre-run VHDL simulations for the experiments described in this lab.

To simulate the architecture depicted in Figure 5, the performance model must first be compiled and the route and program files must be copied into the system simulation directory. This is done automatically with the following commands:

```
>> cd m57_lab_a
>> make1.com
>> cd SYSTEM
>> qhsim toyl &
QHSIM>> run 25 ms
# ** Note: Real-Time Deadline Met!!
#   Time: 20000001 ns Iteration: 0   Instance:/pe1
# ** Note: WARNING: Real-Time Deadline Missed!!
#   Time: 23288721 ns  Iteration: 0   Instance:/pe2
```

During simulation, you should notice a message noting that the real-time deadlines some real-time deadlines have been missed. To see detailed information, type the following commands:

```
>> grep Missed pe*
pe2_B1_TREE1_timeline.dat: RT DEADLINE : Missed by
1788 us
>> more pe2*
```

The GANTT diagram describing the execution of the application on the two processors in shown in Figure 6. Processor 2 is activated after task 1 on processor 1 completes execution. The excessive communication overhead due to the poor task assignment causes the architectural implementation to miss the real-time deadline. Processor 2 takes 21.8 ms to complete one iteration. However, the real-time deadline is 20 ms. Thus, the processor misses the constraint.

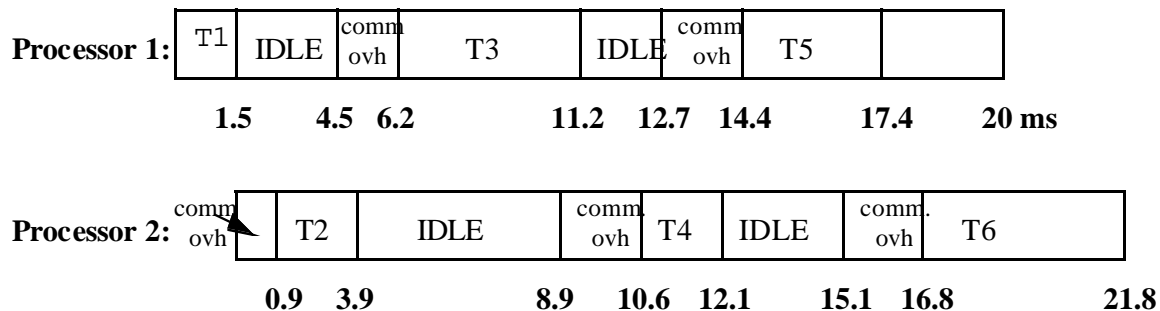


Figure 6. Gantt diagram for architectural implementation with poor task assignment

In order to meet the real-time constraint, we reassign the tasks to processors as shown in Figure 7 and pipeline the computation. This case can be simulated as follows:

```
>> cp ../PROGRAMS_2/* .
```

Click on FILE in the QHSIM window and select RESTART DESIGN.

```
QHSIM>> run 30 ms
# ** Note: Real-Time Deadline Met!!
# Time: 20000001 ns Iteration: 0 Instance:/pe1
```

```
# ** Note: Real-Time Deadline Met!!
#   Time: 29500001 ns   Iteration: 0   Instance:/pe2
```

We now observe that the architecture implementation meets the real-time constraints.

To calculate the communication overhead due to interprocessor communication, examine the output file produced by the performance modeling simulation. To do this, type:

```
>> grep overhead pe*
pe1_B1_TREE1_timeline.dat: Communication overhead for
time interval: 1 us
```

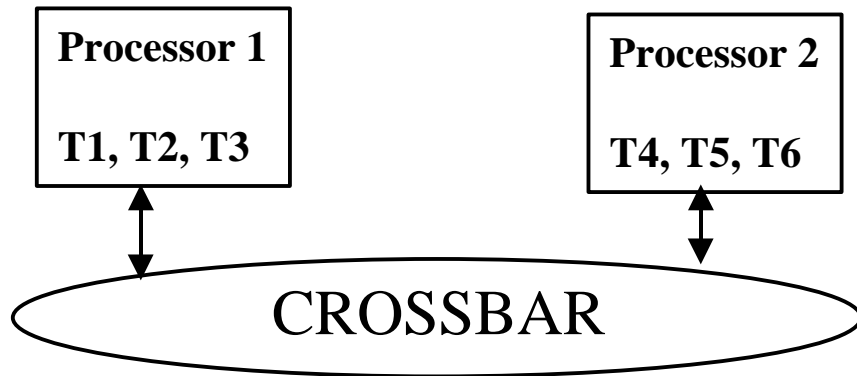


Figure 7. New task assignment for the multiprocessor implementation

```
pe1_B1_TREE1_timeline.dat: Communication overhead for
time interval: 1 us
pe2_B1_TREE1_timeline.dat: Communication overhead for
time interval: 858 us
```

This communication overhead data can be used obtain an accurate processor utilization estimate. The Gantt chart describing the processor execution are shown in Figure 8.

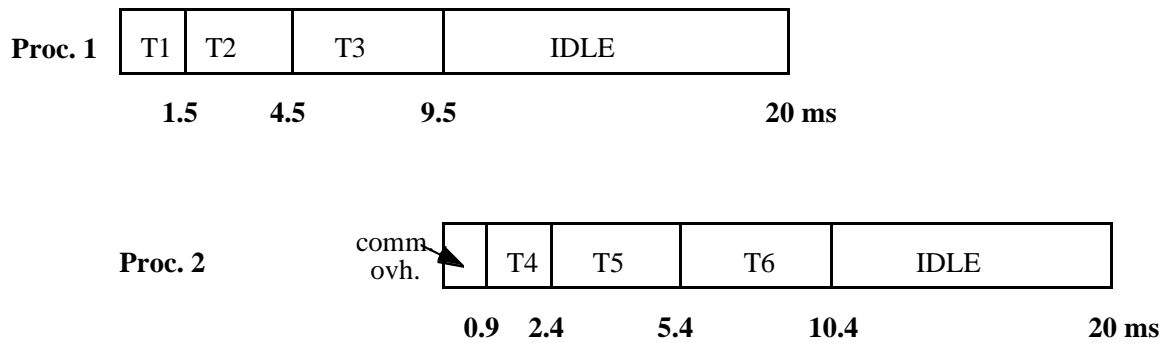


Figure 8. Gantt Chart for implementation with improved task assignment

Software Cost Estimation

The processor utilization is computed as:

$$\text{(Total Computation time + Total Comm. Overhead)} / (\text{No. of Processors} * 20 \text{ ms})$$

$$\text{Hence, Proc. Util} = (19 + 0.9)/40 = .4975 \text{ or } 49.75\%$$

From the REVIC table (in the Cost Modeling Module) we obtain the execution-time resource constraint effort multiplier to be:

$$f_E =$$

Cost (adjusted) for the software development is = $30.6 * 1.0 = 30.6$ person months.

Cost in dollars (assuming \$15,000 per person month) =

Compute the development time (assuming a nominal schedule) is:

$$s_T =$$

3.3. Hardware Procurement Cost

We are assuming that the hardware cost can be added to the software cost as follows:

Production Volume = 1000 units.

Design 1 (hardware cost) = $3000 * 1000 = \$3\text{M}$, at \$3000 a board.

Design 2 (hardware cost) = $5000 * 1000 = \$5\text{M}$ at \$5000 a board.

Compute the total hardware plus software costs.

Case 1:

Case 2:

3.4. Time-to-Market Cost Calculation

The product lifetime $2W = 2$ years.

Maximum available revenue $R_0 = \$10\text{M}$.

For a product deployment goal of 19 months and 22 months, use the time-to-market cost model described in Section 2.2 to compute the revenue will be lost due to schedule delays for both architectures for both goals.

Table 2

	19 months	22 months
Case 1		
Case 2		

3.5. Total System Cost Calculation

Now, compute the total system costs (software development, hardware procurement, and time-to-market costs) for the candidate architectures.

Table 3

	19 months	22 months
Case 1		
Case 2		

For this application, does adding for hardware (an extra processor) actually reduce overall system costs?