DOI: http://dx.doi.org/10.26483/ijarcs.v9i1.5407

ISSN No. 0976-5697

Volume 9, No. 1, January-February 2018

International Journal of Advanced Research in Computer Science

TECHNICAL NOTE

Available Online at www.ijarcs.info

IMPROVING PROTOTYPING EFFICIENCY BY USING SIMULATION IN INTERDEPENDENT HARDWARE AND SOFTWARE PROJECTS

Prashant Mohta
Information Technology
Vidyalankar Institute of Technology
Mumbai, India

Reshma Sherugar
Information Technology
Vidyalankar Institute of Technology
Mumbai, India

Prof. Santosh Tamboli
Information Technology
Vidyalankar Institute of Technology
Mumbai, India

Abstract: Software development on an interdependent Hardware and Software project depends largely on all the specific components used in the current project. It is therefore difficult to develop the system software before the exact details of the hardware are finalised. This forces the developer to first prioritize the hardware prototyping during which time, there is minimal to no software development.This paper proposes a solution that can be implemented at any stage of the project development and can parallelise the software and hardware development i.e. decouple the two by removing the dependency of software development on the availability of testing hardware.

Keywords: common interface, prototyping, simulation, software design, virtual prototype.

I. INTRODUCTION

Development of interdependent Hardware and Software projects usually requires the hardware to be developed and finalised before the development on software can even begin. This leads to an increase in project prototyping time, which leads to an overall increase in project completion time. The standard practice in the industry is the use of virtual prototypes in place of the actual hardware[1]. However, these virtual prototypes involve a lot of low level simulation that increases their complexity. By developing a high level simulator that shares the software interface with the hardware facing modules, it is possible to parallelise the software and hardware prototyping without the increase in complexity, thus reducing overall project completion time and using the available human resources in the project more efficiently.

II. PROBLEM STATEMENT

The traditional approach of designing hardware and software in separate phases, results in an unnecessary delay in beginning the software prototyping which may again take more time. Furthermore if a hardware change is required later into the project timeline, it may cause more delays in the software development till the hardware changes are finalised.
The dependency on hardware for software development requires multiple hardware prototypes for testing, in a project with more than a few software developers. This requires the fabrication of multiple test prototypes of the same hardware specification every time the hardware specifications are altered. This not only creates a delay in the project prototyping but also increases the prototyping cost.

III. PROPOSED SOLUTION

A. *Common Interface*

The solution that this paper looks at, is to create a common interface between the application code and the hardware facing code i.e. the code used to interface with the hardware. This is the only part of the software development that is truly dependendent on the hardware components and the circuit design. In doing so, we can successfully decouple the two development processes till the hardware design is finalised, after which the hardware facing components can be implemented in a short amount of time.

B. *Simulation*

Furthermore, the unavailability or failure of hardware need not affect the software development because the common interface can be used to simulate the same kind of input/output functionality on the machines being used to code, as will be on the prototype hardware. This also eliminates the need to transfer the code to the hardware for testing the impact of code changes, as these changes can be seen using the aforementioned simulated input/output functionality.

IV. COMMON INTERFACE AND SIMULATION BASED SOFTWARE DESIGN

The common interface also means that, a hardware/circuit design change need not necessarily impact the application code or development as only the hardware facing code would required to be changed in most cases.This is akin to the virtual prototyping model already being increasingly used[2]. However, what differentiates common interface and simulation based software design (CISD) is that the application code does not interact with a perfect replica simulation i.e. emulation of the hardware, rather it interacts with a “translator” that translates the input to a particular component into the expected output from it, within acceptable error boundaries. This means that CISD focuses less on the minute details of implementation but rather on that the more general expected results are achieved. This makes CISD viable for rapidly prototyping project ideas for feasibility or practicality reasons. Which is an important part in deciding whether or not a project will be dropped[3]. In case a project is dropped having used CISD would reduce the losses due to pursuing an infeasible project.

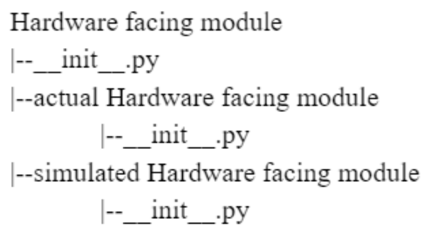
C. *Designing a common interface*

Refer Fig. 1, Inside actual hardware facing module we will import a hardware specific module that is only present on actual hardware and Inside the simulated hardware facing module we will import a simulation specific module that is only present on simulation machine.

Then, inside the parent module, we will import them using exception handling and only one of the two child modules will be imported, the logic is as follows:

```
if (actual hardware facing module is loaded correctly):
        The code is running on hardware so use actual hardware facing module
elif (simulated hardware facing module is loaded correctly):
        The code is running on simulation machine so use simulated Hardware facing module
else:
        Throw an exception that module cannot be loaded
```

Alternatively, the modules can simply be swapped-in during compile time for code size reduction and compilation purposes in languages such as C.



```
Hardware facing module
|--__init__.py
|--actual Hardware facing module
        |--__init__.py
|--simulated Hardware facing module
        |--__init__.py
```

Fig. 1 Python module directory structure for one hardware component

D. *Designing the simulation*

The simulated hardware facing modules have the same functions with the exact same signatures, global constants and variables as the actual hardware facing modules. But in addition to these, they may also have a few helper functions to map between the implemented simulation and the hardware device. Furthermore, the functions that are common in both the simulated and the actual hardware facing modules return the same values. However, the simulated hardware facing module functions have different code which with the help of the helper functions are able to simulate the hardware component.

The simulation for devices like screens, LEDs, keyboards, mouse, touchscreen, buttons, sensors, actuators, etc. can be implemented using pygame, in python, for creating the GUI that is used to display the output from the application code as well as take the input for the application code. The application code behaves as if it were running on the hardware itself and is unaware of whether it is being simulated or not as long as -

- The input is same as it would be from an actual hardware device.
- The output is in the same format as expected by the actual hardware device.
- The simulation code maps perfectly between the input and the output as the actual hardware would.

If the above 3 conditions are fulfilled by the simulation module, then, common interface and simulation based design can be used throughout the project timeline for testing of different versions of the application code on the simulated hardware i.e. the coding machine itself, without the need to be transferred to the hardware which will speed up the testing process.

In the case of a modification in the hardware design of the project, the changes must be reflected in both the actual hardware facing module and the simulated hardware facing module. Thus, preserving the above 3 conditions. This will ensure the ability to use CISD in testing.

V. IMPACT OF COMMON INTERFACE AND SIMULATION BASED SOFTWARE DESIGN ON THE TIME TAKEN FOR PROJECT PROTOTYPING

$$T = \sum_{0}^{n} t_i \text{ - time overlap between activities}$$

$t_i = i^{th}$activity in the project

T = Development time taken in typical timeline

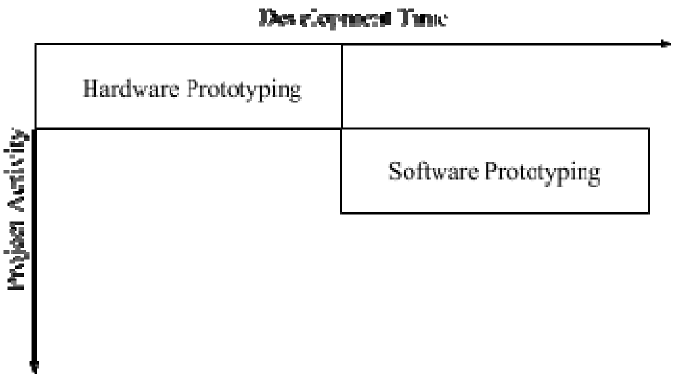


Fig. 2 Typical Timeline for Hardware and Software prototyping

Consider Fig. 2, a typical timeline for hardware and software prototyping.

Let,

t_h = time taken for hardware prototyping

t_s = time taken for software prototyping

Thus,

Development time taken in a typical timeline $= t_h + t_s$

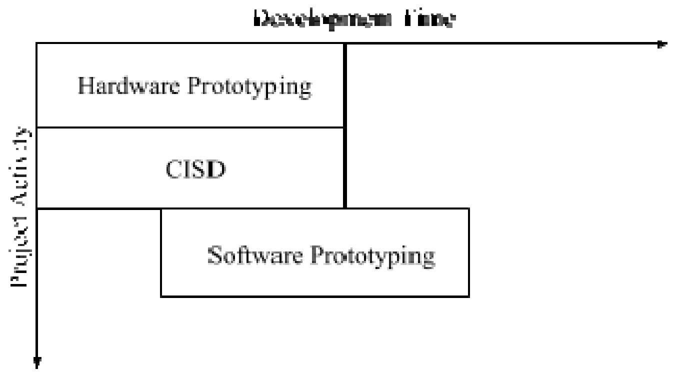


Fig. 3 Modified Timeline for Hardware and Software prototyping

Consider Fig. 3, modified timeline for hardware and software prototyping

Let,

t_h = time taken for hardware prototyping

t_s = time taken for software prototyping

t_{CISD} = time taken for CISD

Thus,

$$T_m = t_s + t_{CISD} - (t_s \cap t_{CISD})$$

T_m = Development time taken in modified timeline

Comparing T and T_m, we can infer that,

$$\text{if, } t_{CISD} - (t_s \cap t_{CISD}) < t_h$$
$$\text{then, } T_m < T$$

Where,

$t_{CISD} - (t_s \cap t_{CISD})$ is the time saved due to CISD.

VI. Impact of Increasing Tcisd on Modified Project Timeline

Consider a project with 50% of its time allocated to hardware prototyping and 50% to software prototyping .The above graph compares the time taken to complete the project (in percent) in the original timeline with the modified timeline using CISD, as the time taken for common interface and simulation design process varies from 0% to 200% of the original project time, it is assumed that software prototyping can begin 50% into CISD i.e. the time overlap between software and CISD is half the time taken to complete CISD.

From the graph it can be inferred that any project which takes less than 100% of its time for CISD can still gain some reduction in the time taken to complete the project by using CISD.

An interesting case is when $t_{CISD} = 0$ i.e. the common interface and simulation is designed in a previous project and is only reused. This allows for full parallelization of hardware and software prototyping leading to a 50% of savings in project time.

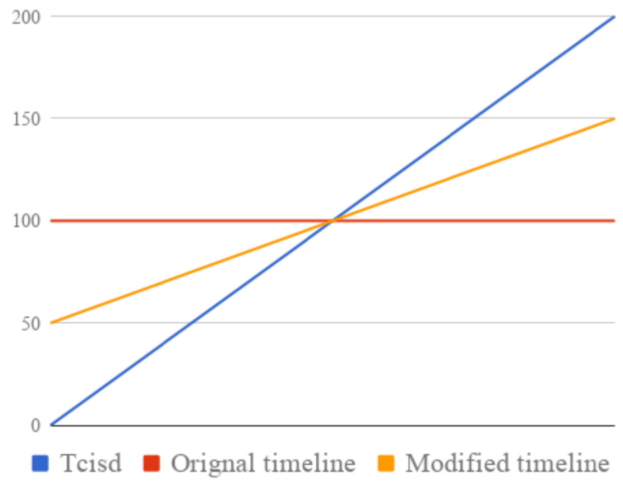


Fig. 4 Impact of increasing Tcisd on modified project timeline

VII. Merits of Using CISD

Thus the CISD methodology also benefits from code reuse, a standard practice in software development that speeds up the project and reduces the cost of development by saving a significant amount of time[4].

During the prototyping phase, the advantages of having a CISD based process should be viewed as a hybrid approach to system development where the prototyping takes full advantage of the simulation capability and the final hardware testing provides a absolute surety of correctness. Furthermore, it is possible to simulate only the peripherals and not the core hardware components of the system using CISD. Ideally, this hybrid approach should yield the best results by finding a middle ground between accuracy and speed of testing[5].

If the system being designed is a real world physics dependent system, then, it is also possible to ue CISD to speed up the prototyping process by simulating all the physics based components in a physics simulation engine and designing a common interface with the controller hardware which will again, comparatively be cheaper and safer

Another advantage of CISD, is that, because the hardware design is now happening parallely with the software design, it is possible to iteratively add or modify features in the hardware design based on changing software design requirements. Which in turn,ends up improving the end user experience as the entire project is more cohesively developed i.e. developed as a complete package rather than in distinct stages[6].

VIII. References

[1] J.C. Schaaf, F.L. Thompson, “System Concept Development With Virtual Prototyping,” Winter Simulation Conference Proceedings, Atlanta, Georgia, USA, 1997, pp. 941-947.

[2] Website: https://www.twi-global.com/technical-knowledge/faqs/faq-what-is-virtual-prototyping/

[3] Goodman L.J. (1988) “Feasibility Analysis and Appraisal of Projects.” In: Project Planning and Management. Springer, Boston, MA.

[4] R. Prieto-Diaz, P.Freeman, “Classifying Software for Reusability,” in *IEEE Software*, vol. 4, no. 1, pp. 6-16, Jan. 1987.

[5] Tom Borgstrom, Eshel Haritan, Ron Wilson, David Abada, Ramesh Chandra, Chuck Cruse, Andrew Dauman, Olivier Mielo, Achim Nohl “System prototypes: Virtual, hardware or hybrid?,” *2009 46th ACM/IEEE Design Automation Conference*, San Francisco, CA, 2009, pp. 1-3.

[6] Marcello Pellicciari, Alberto Vergnano, Giovanni Berselli “Hardware-in-the-Loop Mechatronic Virtual Prototyping of a high-speed capsule filling machine,” 2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA), Senigallia, 2014, pp. 1-6.