

R. P. Self, M. Fleury, A. C. Downton  
Department of Electronic Systems Engineering,  
University of Essex, Colchester CO4 3SQ, UK.  
Email: { rpsself; fleum; acd }@essex.ac.uk

## Abstract

The codesign of reconfigurable systems, comprising multiple hardware-software blocks, is hampered by the lack of structured methods and high-level design support. An asynchronously-timed design model has been adopted, and on-chip Task Manager runtime environment implemented, to facilitate codesign. An exemplar implementation of the KLT algorithm demonstrates the runtime to have a minimal footprint and little impact on clock speed.

## 1. Introduction

Complex codesign systems are now feasible with the arrival of multi-million gate FPGAs, such as the Xilinx Virtex series. Furthermore, the design and development of such systems is greatly simplified by the availability of C-language hardware compilers, such as Handel-C. However, complex systems need structured methods and high-level design support to guide development, which an implementation language cannot provide. This poster presents a high-level design model, based on CSP [1], and an on-chip Task Manager runtime environment as a means of filling this design gap. The impact of Task Manager overhead on an application is investigated by examining a pipelined implementation of the Karhunen-Loève Transform (KLT) algorithm [2].

## 2. System-Level Design

For the design and development of complex systems loosely-coupled architectures are desirable, as this allows systems to be constructed in incremental fashion from existing modules. However, existing digital systems are usually based on a synchronous timing model, which tightly binds modules as a consequence of the timing dependencies necessary for lock-step operation. A solution in the form of a two-level, asynchronously-timed system model, shown in Figure 1, has been used to address this issue. The model comprises two views:

- **System-Level View:** At the system-level, a design is decomposed into functionally-independent tasks communicating asynchronously over channel objects.
- **Task-Level View:** At the task-level, each task performs a well-defined operation and can be decomposed into further sequential and parallel units.

No constraints are placed on task implementation. Tasks can be built from synchronous logic in order to maximise performance and achieve deterministic operation, or employ channels to form loosely-coupled networks.

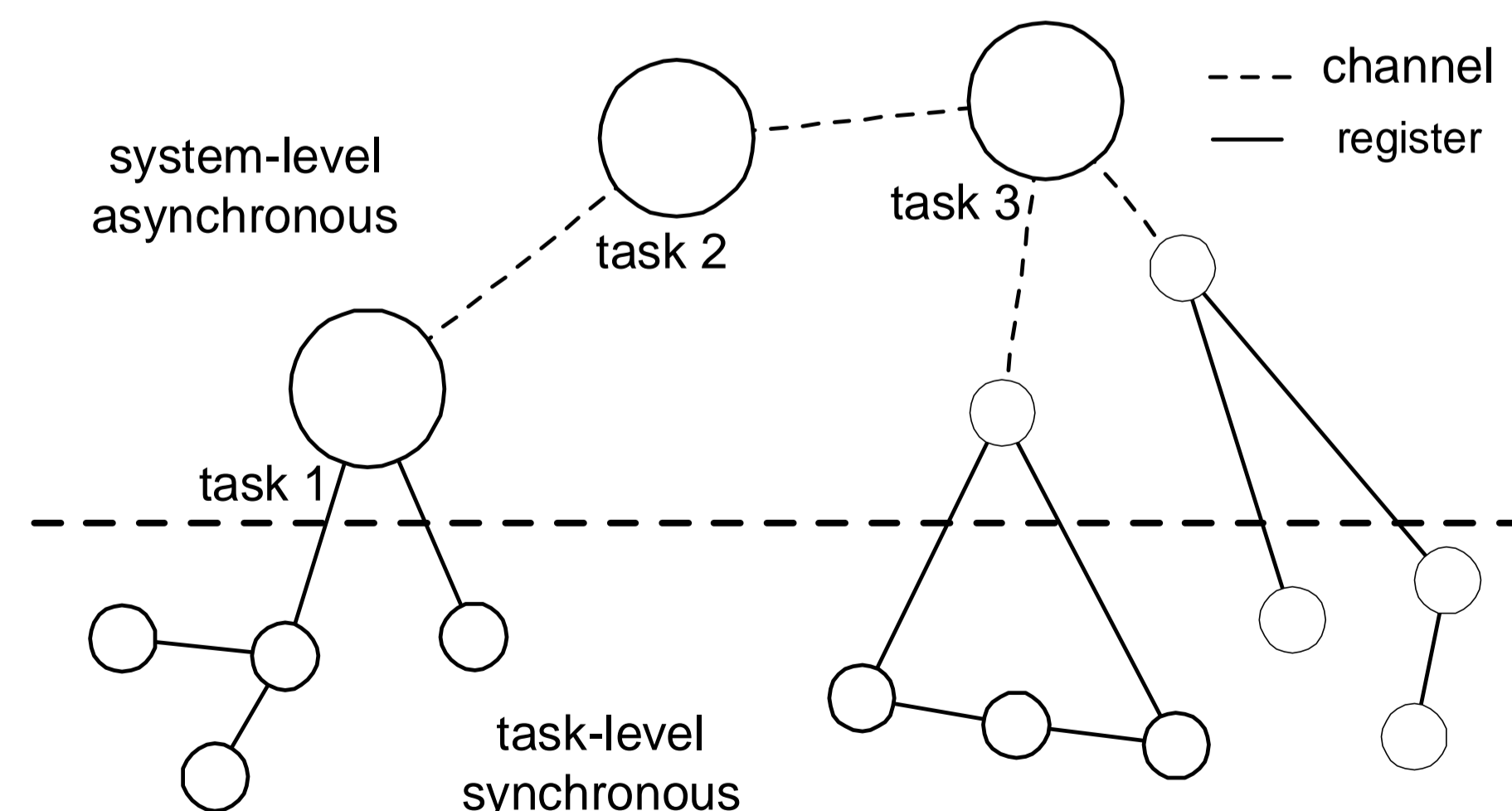


Figure 1: Task-Based Design

## 3. Tasks

Tasks are the means by which static FPGA processing resources are exposed to host software. Each task is uniquely identified by a 'taskid' and managed by host software through a command-response messaging protocol. A high-level protocol has been adopted in order to:

- Expose task execution to host software in a manner that improves task control and ensures robust operation.
- Support error handling and simplify application debugging by the provision of runtime status messages.

The messaging protocol is shown in Figure 2. The host executes a task by sending a 'start' message; the task then responds with 'taskrunning' and 'taskcomplete' messages that the host tracks for error and debug handling purposes.

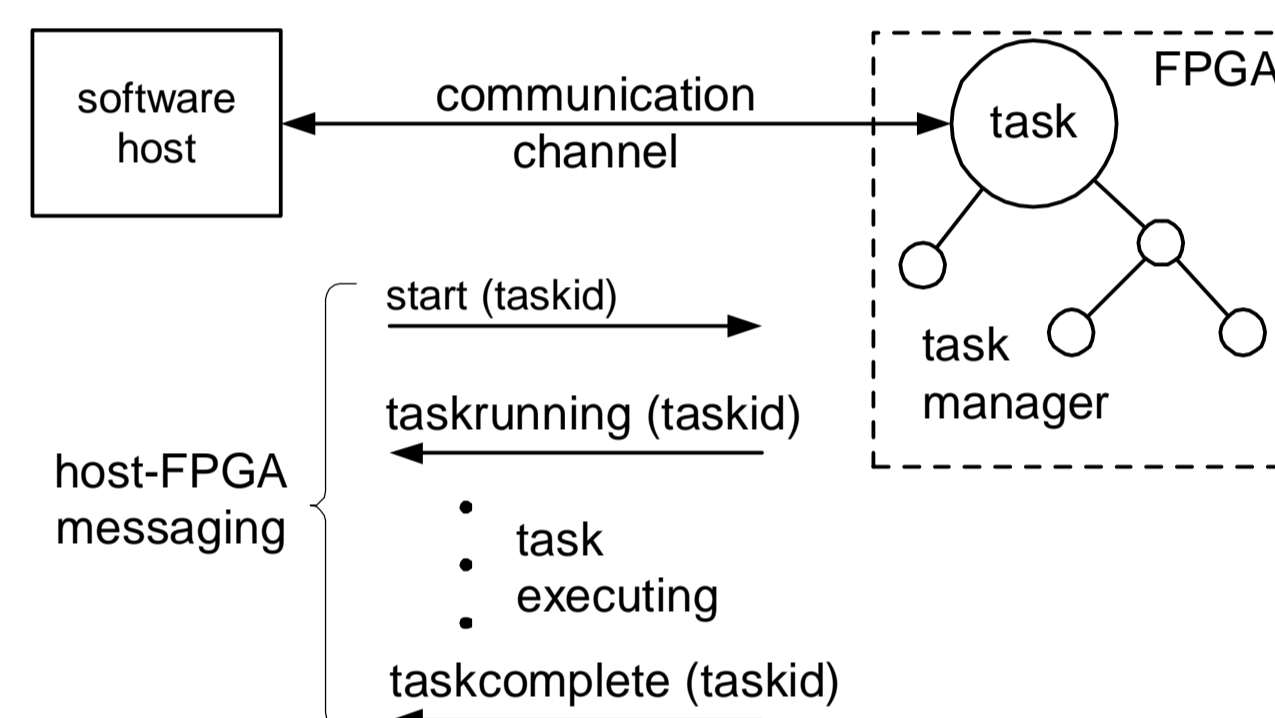


Figure 2: Host-FPGA Task Messaging

## 4. Task Manager

The Task Manager, shown in Figure 3, is responsible for controlling the interaction between the software host and individual tasks, and updating Task State administration, as a task transits from idle to started, and started to running state. Messages are mapped to low-level 'events' (Handel-C channels) for efficiency reasons. Each task comprises:

- **DoTask Unit:** The DoTask unit is a container for application code.
- **Task Controller:** The Task Controller acts as an intercept for Task Manager start events. This allows 'taskrunning' events to be generated independently of application code.

Tasks have been split in this manner to improve task modularity and ensure separation between application and Task Manager runtime functionality.

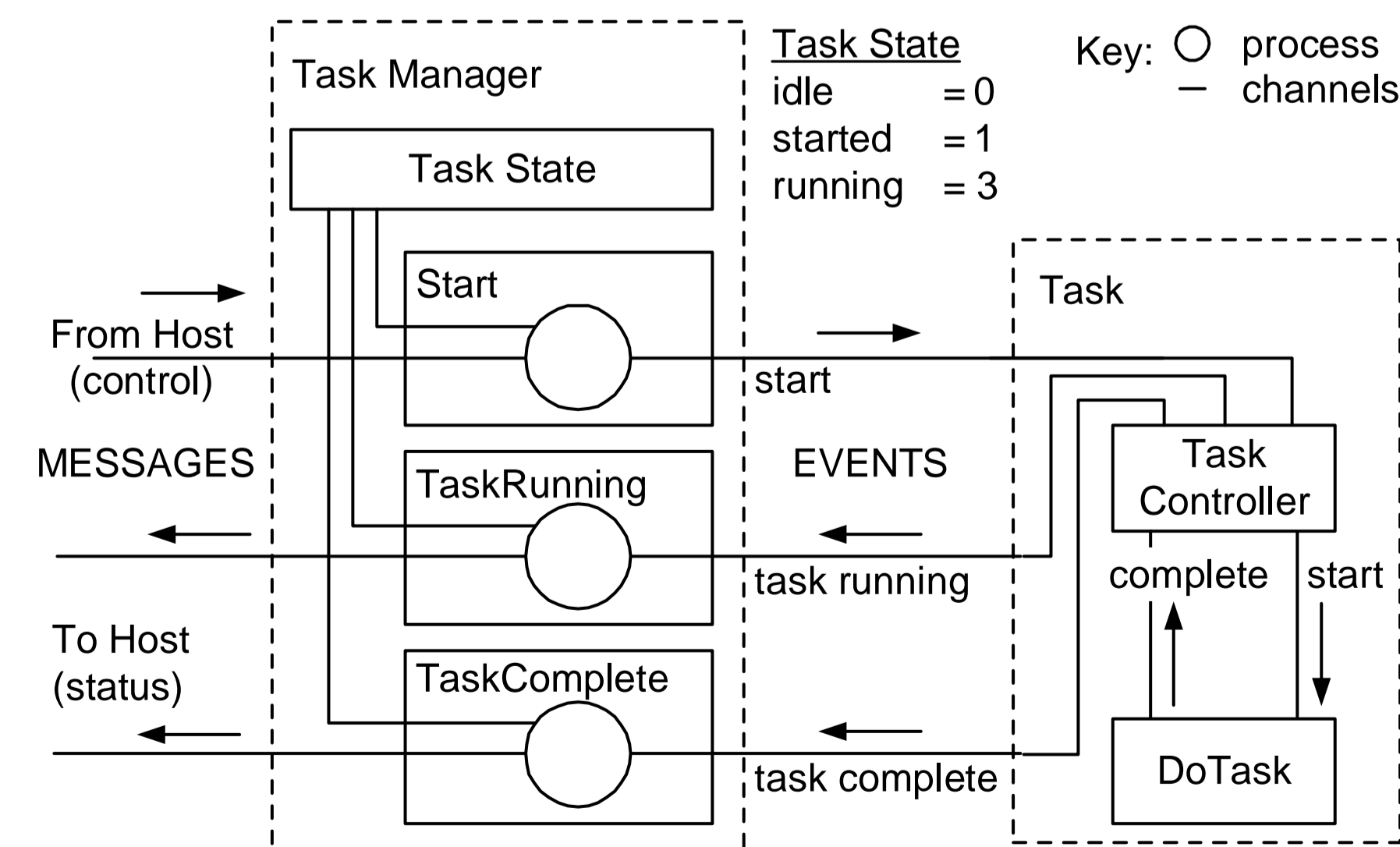


Figure 3: The Task Manager Runtime Environment

## 5. Results

The purpose of this work is to identify baseline overhead, and to demonstrate the impact of overhead on an exemplar design. Tables 1 and 2 show the area and clock speed figures relating to Task Manager implementations of 'Minimal' tasks (no application code), and the KLT exemplar. From Table 1 it can be seen that the worst-case area usage is 153 slices, while the worst-case clock speed is approximately 51 MHz. Table 2 shows that for the KLT application relative slice overhead drops with increased design complexity and clock speed remains largely unaffected.

In practice, neither area nor clock speed overhead are likely to be a significant limiting factor for large-scale codesign, as:

- Worst-case area usage is less than 2% of Xilinx Virtex XCV1000-4.
- Many Handel-C designs run at less than 40 MHz.

For these reasons, it is believed that the Task Manager runtime environment can be successfully applied to a broad range of application scenarios.

No of Tasks	Slices (n)	Speed (MHz)	No of Tasks	Slices (n)	Speed (MHz)
1	80	71.958	3	115	51.438
2	98	65.308	4	153	54.702

Table 1: 'Minimal' Task Area and Performance

Design	Baseline		Task Manager		Difference	
	Slices (n)	Speed (MHz)	Slices (n)	Speed (MHz)	Slices (% n)	Speed (% MHz)
Stage 1	555	32.936	633	32.347	14.0 (78)	1.80 (0.589)
Stage 3	402	24.704	483	25.897	20.0 (81)	4.80 (1.193)
Stage 1, 3	970	23.449	1,064	23.493	9.7 (94)	0.18 (0.044)

Table 2: Comparison of KLT Area and Performance

## 6. Conclusions

This poster has presented an asynchronously-timed, system-level design model and on-chip Task Manager runtime environment as a means of simplifying the design and development of codesign applications. The benefits of this approach include:

- A high-level design model resulting in a flexible system architecture that facilitates design modularity and code reuse.
- Simplified application development by replacing *ad hoc* methods with a structured approach to codesign engineering.
- Improved application robustness, because systems are constructed from an already proven runtime environment.
- Applicability to a broad range of design scenarios, due to the minimal overhead and resource requirements of the Task Manager runtime environment.

## References

- [1] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666-677, 1978.
- [2] M. Fleury and A. C. Downton. *Pipelined Processor Farms*. Wiley, 2001.