

Cooptimization of Interface Hardware and Software for I/O Controllers

Kuan Jen Lin⁺, Shih Hao Huang and Shan Chien Fang
Department of Electronic Engineering, Fu Jen Catholic University, Taiwan
⁺kjlin@mails.fju.edu.tw

Abstract

The allocation of device variables on I/O registers affects the code size and performance of an I/O device driver. This work seeks the allocation with the minimal software or hardware cost in a hardware/software codesign environment. The problems of exact minimization under constraints are formulated as zero-one integer linear programming problems. Heuristic algorithms based on iterative refinement are also proposed. The proposed design methodology was implemented in C language. Compared with industrial designs, the system can obtain design alternatives that reduce both software and hardware costs.

1. Introduction

An I/O controller is used to manage a peripheral physical I/O device. A device driver is a software layer that lies between an operation system and the I/O controller. It generally configures the operation modes of the device, observe its statuses, and transfer the data via accessing registers in the I/O controller. Figure 1 shows such a hardware and software interface. Usually, a register contains several fields (each occupying several consecutive bits), each of which represents an operation mode, a status or a datum. Each field is referred as a *device variable* [1]. A set of variables associated with a common purpose form a *variable group*. To minimize the number of registers and the number of register access, hardware designers often allocate device variables in the same group into the same register. However, this may increase the number of I/O accesses and bit-operations (such as shift instruction and logic instruction) when a driver wants to manipulate individual variables. Figure 2 shows such a code example, which modifies variable *B* in an I/O register while keeping *A* and *C* unchanged. The operation totally needs four instructions. If we use a register to contain only variable *B*, the operation just needs one instruction *io_write()*. Specifically, the software cost is affected by the allocation of device variable. As reported in [1], low-level codes have been found to represent up to 30% of a device driver. Their size and performance inevitably become important issues for I/O

intensive systems.

Previous works addressing I/O interface synthesis were based on already defined I/O controller [1, 2, 3]. Namely, the allocation of device variables is fixed. In this paper, we propose a novel design methodology that attempts to co-optimize the number of registers inside an I/O controller and the driver codes to access and manipulate these registers by tuning the allocation of device variables. The approach is favorable in an SOC HW/SW codesign environment.

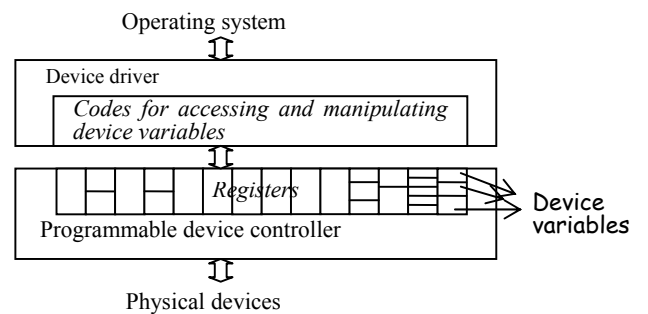


Figure 1: Interface hardware and software of an I/O device.

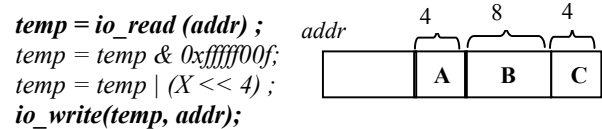


Figure 2: Write *X* (in a driver code) to device variable *B* in a 32-bit register.

2. Cost Model

The software costs for different accesses and allocations are summarized in Table 1. A variable can be allocated in three types of registers: (1) *single*: a register contains only one variable; (2) *shared*: a register contains more than one variable and they belong to the same group; (3) *group-shared*: a register contains more than one variable and they are from different groups. There are four types of I/O access, including reading (writing) an individual variable and reading (writing) *K* variables of a group simultaneously, i.e. in one instruction.

The total software cost also depends on the execution numbers of various I/O accesses. These numbers are application-specific. Given these access profiles and an allocation, the software cost can be calculated by adding up all the twelve entries in the Table 1. The hardware cost under our consideration is the number of registers used to allocate device variables. Notably, the software cost is not necessarily inversely proportional to hardware cost. This work seeks the allocation with the minimal software or hardware cost under constraints. These problems of exact minimization are formulated as zero-one integer linear programming problems [4].

Table 1: Software cost

Allocation Access	Single register	Shared register	Group-shared register
Read an individual	C_1	$C_1 + C_2$	$C_1 + C_2$
Write an individual	C_1	$2C_1 + 2C_2$	$2C_1 + 2C_2$
Read K variables of a group	KC_1	$C_1 + KC_2$	$C_1 + KC_2$
Write K variables of a group	KC_1	$C_1 + KC_2$	$2C_1 + (K+1)C_2$

C_1 : Execution cost of an I/O access instruction

C_2 : Execution cost of a bit-manipulation instruction

3. Design Methodology

The flow of the proposed design methodology is shown in Fig. 3. Given the device-variable specification, the system derives two initial allocations without any HW or SW constraint. One has the minimal SW cost and the other minimal HW cost. Then given a hardware constraint (or a software constraint), the system can obtain an allocation having a minimal software (or hardware) cost under the constraint. The inner loop allows a user to iteratively refine the solution to meet a certain purpose. In both HW and SW optimizations, we first handle each device variable group separately. Then we allow variables in different groups to share registers if the following situations occur: (1) no solution exists to meet the hardware constraint and (2) the hardware cost can be further reduced while still meeting the software constraint.

Although an ILP solver can derive the exact solution with the minimum cost, it is time-consuming for handling large cases. We propose a heuristic approach that starts from an initial allocation and iteratively refines the solution unless the constraint is violated. The iterative refinement techniques are based on a set of proven lemmas [4].

4. Experimental Results and Conclusion

The proposed design methodology was implemented in C. We used a set of devices from the Devil system [1], including BusMr, IDE, X11 8290 and 83905 devices. The

bit length of each device variable and the members of a group in the original Devil's specification were used in our work. The access profiles were assigned by ourselves to evaluate three cases: (a) $f_g^r + f_g^w \gg f_i^r + f_i^w$ (10: 1), (b) $f_g^r + f_g^w \cong 2(f_i^r + f_i^w)$ and (c) $f_g^r + f_g^w \ll f_i^r + f_i^w$ (1: 10), where f_i^r (f_i^w) denotes the profile to read (write) an variable i and f_g^r (f_g^w) the whole group. Table 3 shows the experimental results. Compared with industrial designs, we can obtain design alternatives that reduce both software and hardware costs.

We will evaluate the proposed tools with the access profiles derived from real applications and explore design space for various profiles.

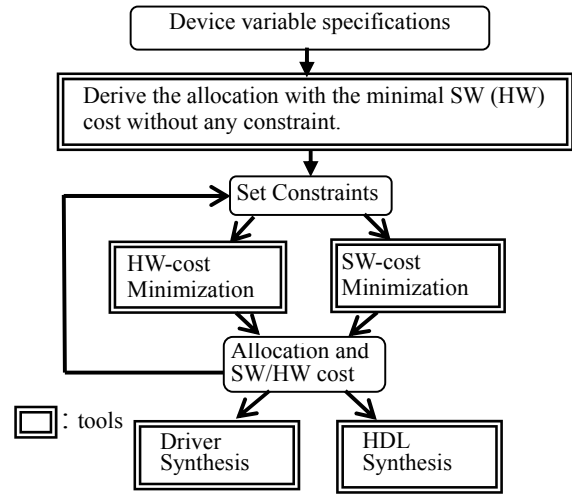


Figure 3: HW/SW cooptimization flow for an I/O device controller.

5. References

- [1] F. Mérillon and G. Muller, "Dealing with hardware in embedded software: A general framework based on the Devil language," *ACM LCTES*, pp. 121 – 127, 2001.
- [2] P. Chou, B. R. Ortega and G. Borriello, "Interface Co-Synthesis Techniques for Embedded Systems," *ICCAD*, pp. 280-287, 1995.
- [3] M. O'Niels, J. Oberg and J. Jantsch, "Grammar Based Modeling and Synthesis of Device Drivers and Bus Interfaces," *Euromicro Workshop*, pp. 55-58, 1998.
- [4] S. H. Huang, "Optimize HW/SW Interface Design programmable IO devices," *MS Thesis*, Fu Jen Catholic University, Taiwan, 2005.

Table 3: Experimental results

Access profiles	Industrial	SW OP	HW OP	GHW OP
	Hardware cost / Software Cost			
$f_g^r + f_g^w \gg (f_i^r + f_i^w)$	100%/100%	43%/40%	43%/40%	35%/46%
$f_g^r + f_g^w \cong 2(f_i^r + f_i^w)$	100%/100%	67%/56%	43%/60%	35%/62%
$f_g^r + f_g^w \ll (f_i^r + f_i^w)$	100%/100%	100%/96%	89%/96%	79%/82%

Notes: Industrial: the cost of the original allocation in the Devil specification.

SW OP: SW minimization using the industrial HW cost as the constraint.

HW OP: HW minimization using the industrial SW cost as the constraint.

GHW OP: HW minimization by allowing different groups to share a register.