Optimal Allocation of I/O Device Parameters in Hardware and Software Codesign Methodology

Kuan Jen Lin, Shih Hao Huang and Shih Wen Chen

Dept. of Electronic Engineering, Fu Jen Catholic University,
242 Taipei County, Taiwan
kjlin@mail.fju.edu.tw, {a9250627, a9150628}@st2.fju.edu.tw

Abstract. For a programmable I/O device controller, the allocation of device parameters on I/O registers affects the code size and execution time of its associated I/O device driver. In traditional design flow, the development of device drivers can not begin until the allocation is fixed. This paper presents a new design methodology that allows a designer to seek an allocation that reduces the software or hardware cost concurrently with developing device drivers. The software cost means the code size or execution time and the hardware cost the number of I/O registers. The exact allocation with the minimum cost under constraints is formulated as zero-one integer linear programming problem. Heuristic algorithms based on iterative refinement are also proposed. The proposed design methodology was implemented in C language. Compared with current industrial designs, the approach can obtain design alternatives that reduce both software and hardware costs. Furthermore, the experimental results also investigate design spaces for various application features. It turns out that the HW/SW codesign approach is favorable in development of embedded systems.

Keywords: Hardware/Software Codesign, I/O interface, Device Driver, Programmable controller.

1 Introduction

A programmable 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 can configure 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 parameter* [3] (or device variable as defined in [8]). For example, the length of stop bits is a device parameter of a UART controller. To configure a data frame for a UART transfer, besides of the stop bits, one should also set parity mode, word length and baud rate. These parameters associated with a common purpose form a *parameter group.* To minimize the number

of registers and the number of register accesses, hardware designers often allocate device parameters 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 parameters. Let us see examples in the Fig. 2, that shows various allocations for two parameters A and B. Fig. 3 shows C codes of several access functions in the device driver (or HAL), including *set_B( ), get_A( ),* s*et_A_B( )* (modifying A and B simultaneously) and *get_A_B( ),* for allocation (2). Fig. 4 shows C codes of the same access functions for allocation (1). As shown in these codes, the allocation of device parameters on I/O registers affects the code size and execution time of its associated I/O device driver. As reported in [8], these 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 embedded systems.

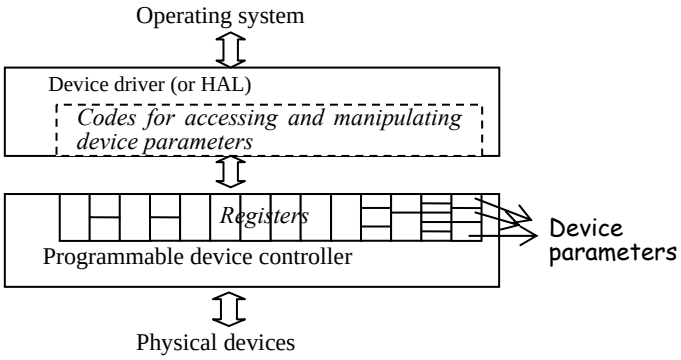

Fig. 1. Interface hardware and software for an I/O device.

In traditional design flow, the development of device drivers can not begin until the device parameter allocation is fixed. In the proposed HW/SW codesign flow, a software programmer can write the device driver using pre-defined parameter-access functions such as *set_B( )*. However, the real C codes of these access functions depend on the physical allocation of parameters. This paper presents a novel design methodology that allows a designer to seek an allocation that reduces the codes of parameter-access functions or the number of I/O registers when developing device drivers. The exact allocation with the minimum cost under constraints is formulated as a zero-one integer linear programming problem. The HW/SW codesign approach is favorable in development of embedded systems. The formulations of an allocation with minimum software or hardware costs are derived. Heuristic algorithms based on iterative refinement are proposed to explore the optimization. The proposed design methodology were implemented in C language and evaluated with a set of real devices. Compared with current industrial designs, we can obtain design alternatives that reduce both software and hardware costs. Furthermore, the results also indicate design spaces for various application features.

To the best of our knowledge, the paper [5] should be the first work to investigate how the parameter allocation can affect the performance of drivers and to try finding an optimal allocation. However, it does not address hardware optimization.

Furthermore, it does not allow parameters belonging to different groups to share a register. Other related works handling the synthesis of device drivers or interface codesign all assume the allocation of device parameters are pre-fixed. F. Merillon et al. [8] address the automatic synthesis of such low-level codes from a higher level specification, called as Devil language. Recent works [1, 9, 10] extend the language's descriptive capability and try to automatically synthesize more dedicated parts of a device driver. P. Chou et al. [2] propose an interface HW/SW cosynthesis work, which synthesizes driver codes as well as glue hardware logic to connect I/O controllers. G. Gognoit et al. proposed communication synthesis and HW/SW integration for Embedded System Design in [4].

The next section defines the software and hardware cost models. Section 3 formulates the exact solution of a parameter allocation with minimum costs. The whole design methodology and tools are presented in Section 4. Experimental results are shown in Section 5. The final section draws conclusions.

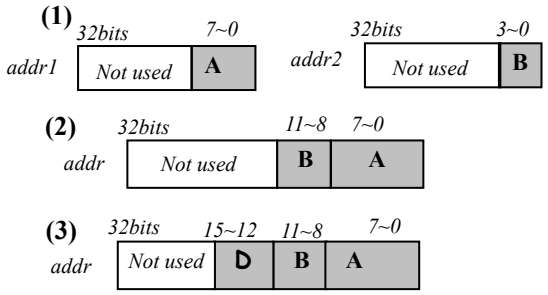

Fig. 2. Three different allocations for a device-parameter group {A, B}, where in case (3) the parameter D belongs to another group.

```
void set_B(int val) {
      int temp;
      temp = io_read (addr) ;
      temp = temp & 0xfffff0ff;
      temp = temp | (val << 8) ;
      io_write(temp, addr);
}//Cost=2C1+2C2

void set_A_B(int val1, int val2) {
      int temp;
      temp= val1;
      temp = temp | (val2 << 8);
      io_write (temp, addr) ;
}// Cost= C1+2C2
```

```
void get_B(int *val) {
      int tmp;
      temp = io_read(addr);
      *val = (temp>>8) & 0x0f ;
}//Cost=C1+C2

void get_A_B(int *val1, int *val2) {
      int temp;
      temp = io_read(addr);
      *val1 = temp & 0xff ;
      *val2 = (temp>>8) & 0x0f ;
}//Cost=C1+2C2
```

Fig. 3. The C codes *set_B ( ), set_A_B( ), get_B( )* and *get_A_B( )* for device parameters A and B which are allocated in the same register, i.e. allocation (2) in Fig. 2.

```
void set_B(int val) {
      io_write (val, addr2) ;
}//Cost=C1

void get_B(int *val) {
      *val=io_read (addr2) ;
}//Cost=C1

void set_A_B(int val1, val2) {
      io_write (val1, addr1) ;
      io_write (val2, addr2) ;
}//Cost=2C1

void get_A_B(int *val1, int *val2) {
    *val1 = io_read(addr1);
    *val2 = io_read (addr2) ;
}//Cost=2C1
```

Fig. 4. The C codes *set_B ( )*, *set_A_B( )*, *get_B( )* and *get_A_B( )* for device parameters A and B which are allocated in different registers, i.e. allocation (1) in Fig. 2.

2 Cost Models

The proposed system allows users to trade off the HW cost for an I/O controller and the SW cost for its associated device driver. The HW and SW cost modes are defined in this section. Based on the observations on the C codes in Fig. 3 and Fig. 4, we summarize the software costs for different accesses and allocations in Table 1. A parameter can be allocated in three types of registers: (1) *single*: the register contains the parameter only; (2) *shared*: the register contains more than one parameter and all parameters in it belong to the same group; (3) *group-shared*: the register contains more than one parameter and the parameters in it are from different groups. The allocation (3) in Fig. 2 is such an example. The I/O access functions include reading (writing) an individual parameter and reading (writing) *K* parameters of the same group simultaneously. Most access functions for shard register and group-shared register are the same, except writing data to some parameters of a group, in that accessing a group-shared register needs one more I/O read access for retaining the values of parameters belonging to other groups.

Table 1. Software cost of various I/O access for different allocations of device parameters.

Allocations / Accesses	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	$2 C_1+ 2C_2$	$2 C_1+ 2C_2$
Read *K* parameters of a group	KC_1	$C_1+ KC_2$	$C_1+ KC_2$
Write *K* parameters of a group	KC_1	$C_1+ KC_2$	$2C_1+(K+1)C_2$

C_1: Execution time (or instruction length) of an I/O access instruction
C_2: Execution time (or instruction length) of a bit-manipulation instruction

The total software cost under our consideration also depends on the execution numbers of these I/O accesses. These numbers are application-specific. The number of times a driver reads (writes) an individual parameter X_i in a period is given as *read*

frequency, f_i^r (*write frequency,* f_i^w). The number of times a driver reads (writes) the whole group is given as *group-read frequency* f_g^r, *(group-write frequency,* f_g^w). Given these access profiles, the software cost can be determined by the allocation of device parameters. In this work, the software cost can be code size or execution time. If code size is the main concern of software cost, the execution number means the number of parameter-access functions used as in-line functions.

Register index	**Variables allocated in this register**
Register *1*	$....X_2X_1$
….	….
Register *p*	$X_{m_p}X_{m_p-1}....$
Register *p+1*	$....X_{m_p+2}X_{m_p+1}$
….	….
Register *p+q*	$X_{m_p+m_q}X_{m_p+m_q-1}....$
Register *p+q+1*	$X_{m_p+m_q+1}$
….	….
Register $n-m_p-m_q$	X_n

Fig. 5. The physical allocation of $\Phi(V_g, m_p, m_q, p, q)$.

The following defines an allocation for a parameter group, $V_g = \{ X_1, X_2, ..., X_n \}$:

Definition 1 (Allocation $\Phi(V_g, m_p, m_q, p, q)$) : For a device parameter group, $V_g = \{ X_1, X_2, ..., X_n \}$, an allocation $\Phi(V_g, m_p, m_q, p, q)$ denotes that there are m_p parameters of V_g allocated in p shared registers and m_q parameters of V_g allocated in q group-shared registers, and the remaining $|V_g|-m_p-m_q$, parameters are allocated in single registers. Without loss of the generality, let $X_1X_2...X_{m_p}$ be in the shared registers, $X_{m_p+1}X_{m_p+2}...X_{m_p+m_q}$ *be in the group-shared registers,* and $X_{m_p+m_q+1}...X_n$ be in single registers.

Figure 5 illustrates the physical layout for $\Phi(V_g, m_p, m_q, p, q)$. Given access profiles, we can calculate the software cost as follows:

Definition 2: **(Software Cost Function $C_s(\Phi)$):** Given f_g^r, f_g^w, f_i^r and f_i^w, $i= 1, ..., n$, for a device parameter group $V_g = \{ X_1, X_2, ..., X_n \}$, the software cost $C_s(\Phi)$ of an allocation $\Phi(V_g, m_p, m_q, p, q)$ equals

$$
\begin{aligned}
C_s =& (f^r_{m_p+m_q+1}+....+f^r_n+f^w_{m_p+m_q+1}+....+f^w_n)C_1\\
&+(f^r_g+f^w_g)(n-m_p-m_q)C_1\\
&+(f^r_1+f^r_2....+f^r_{m_p})(C_1+C_2)+(f^w_1+f^w_2+...+f^w_{m_p})(2C_1+2C_2)\\
&+(f^r_g+f^w_g)(pC_1+m_pC_2)\\
&+(f^r_{m_p+1}+f^r_{m_p+2}....+f^r_{m_p+m_q})(C_1+C_2)+(f^w_{m_p+1}+f^w_{m_p+2}+...+f^w_{m_p+m_q})(2C_1+2C_2)\\
&+f^r_g(qC_1+m_qC_2)+f^w_g(q(2C_1+C_2)+m_qC_2)
\end{aligned}
$$

The first two rows in the above equation calculate the costs in the first column of Table 2. The third and fourth rows calculate the costs in the second column. The final two rows calculate the costs in the third column. Generally, the proposed system handles each parameter group separately since the register sharing between different groups always increases software cost. The following shows the software cost function for the case that a group does not share any register with other groups.

Definition 3: **($C_s(\Phi)$ for $\Phi(V_g, m, 0, p, 0)$):** Given f_g ($=f_g^r + f_g^w$), f_i^r and f_i^w, $i = 1,..., n$, for a device parameter group $V_g = \{ X_1, X_2,..., X_n \}$, the software cost $C_s(\Phi)$ for an allocation $\Phi(V_g, m, 0, p, 0)$ equals

$$
\begin{aligned}
&(f_{m+1}^r + ... + f_n^r + f_{m+1}^w + ... + f_n^w)C_1 \\
&+ (f_1^r + f_2^r .... + f_m^r)(C_1 + C_2) + (f_1^w + f_2^w + ... + f_m^w)(2C_1 + 2C_2) \\
&+ f_g(n-m)C_1 \\
&+ f_g(pC_1 + mC_2)
\end{aligned}
$$

The hardware cost is defined as the number of registers used to allocate device parameters, as described in the following definitions.

Definition 4: **(Hardware Cost Function $C_h(\Phi)$):** For a device parameter group $V_g = \{ X_1, X_2,..., X_n \}$, the hardware cost $C_h(\Phi)$ of an allocation $\Phi(V_g, m_p, m_q, p, q)$ equals $p + q + n - m_p + m_q$.

Definition 5: **($C_h(\Phi)$ for $\Phi(V_g, m, 0, p, 0)$):** For a device parameter group $V_g = \{ X_1, X_2,..., X_n \}$, the hardware cost $C_h(\Phi)$ of an allocation $\Phi(V_g, m, 0, p,0)$ equals $p + n - m$.

3 Formulation of Exact Minimization

This work seeks an allocation with the minimal SW or HW cost. The exact solution assuming no group-shared register being used is formulated. The software cost can be formulated as a problem of zero-one integer linear programming by using binary decision variables with two indices : $Y=\{y_{i,j};\ i=1, 2,..., n;\ j=1, 2,..., \lambda\}$, where n is the number of parameters, and λ is the upper bound of the register used. The $y_{i,j}$ = 1 (0) means that the device parameter X_i is (is not) allocated in the register j. First, the number of shared registers, p, is assumed to be fixed. In other words, at most $\lambda - p$ single registers are used. General cases are discussed later. Let B_i be the bit-length of device parameter X_i, and RL the width of register. The registers numbered 1, 2,…, p denote shared registers. The problem now can be stated as follows:

Minimize software:

$$
\sum_{j=1}^{p} f_g C_1 + \sum_{i=1}^{p} \sum_{i=1}^{n} \left[(C_1 + C_2)\left(f_i^r + 2f_i^w\right) Y_{i,j} + f_g C_2 Y_{i,j} \right] + \sum_{j=p+1}^{\lambda} \sum_{i=1}^{n} C_1 \left(f_i^r + f_i^w + f_g\right) Y_{i,j}
$$

Subject to:

(*a*) $\sum_{j=1}^{\lambda} Y_{1,j} = 1$, $\sum_{j=1}^{\lambda} Y_{2,j} = 1$, ... , $\sum_{j=1}^{\lambda} Y_{n,j} = 1$

(*b*) $\sum_{i=1}^{n} Y_{i,1} \geq 2$, $\sum_{i=1}^{n} Y_{i,2} \geq 2$, ... , $\sum_{i=1}^{n} Y_{i,p} \geq 2$

(*c*) $\sum_{i=1}^{n} Y_{i,p+1} \leq 1$, $\sum_{i=1}^{n} Y_{i,p+2} \leq 1$, ... , $\sum_{i=1}^{n} Y_{i+\lambda} \leq 1$

(*d*) $\sum_{i=1}^{n} B_i X_{i,1} \leq RL$, $\sum_{i=1}^{n} B_i X_{i,1} \leq RL$, ... , $\sum_{i=1}^{n} B_i X_{i,\lambda} \leq RL$

Condition (a) means that any parameter must be allocated in exactly one register. Condition (b) means that the number of parameters allocated in a shared register must be at least 2. Condition (c) means that the number of parameters allocated in a single register must not be greater than 1 (but could be zero). Condition (d) means that the sum of the bit lengths of parameters in a register must not be larger than the length of a register.

In the hardware side, the exact cost minimization under no constraint is just a bin packing problem [7]. Under a software constraint, the exact minimization can also be formulated as a zero-one ILP problem. We still need the decision variable *Y*. Furthermore, we define a binary decision variable Z_j, $j=\{1, 2,..., \lambda\}$, where $\lambda=n-p$, is the maximum number of registers probably used. The z_j = 1 (0) means that the j^{th} register is (is not) used. Let *SC* be the given software-cost constraint. The symbols B_i, X_i, and *RL* are used as for software minimization formulation. The problem now can be stated as follows:

Minimize hardware: $\sum_{j=1}^{p} Z_j + \sum_{j=p+1}^{\lambda} Z_j$

Subject to:

(*a*), (*b*), (*c*) (the same as ones for the software part)

(*e*) $\sum_{i=1}^{n} B_i X_{i,1} \leq RL \cdot Z_1$, $\sum_{i=1}^{n} B_i X_{i,1} \leq RL \cdot Z_1$,., $\sum_{i=1}^{n} B_i X_{i,\lambda} \leq RL \cdot Z_\lambda$

(*f*) $\sum_{j=1}^{p} f_g C_1 + \sum_{i=1}^{p} \sum_{i=1}^{n} \left[(C_1 + C_2)\left(f_i^r + 2f_i^w\right) Y_{i,j} + f_g C_2 Y_{i,j} \right] + \sum_{j=p+1}^{\lambda} \sum_{i=1}^{n} C_1 \left(f_i^r + f_i^w + f_g\right) Y_{i,j} \leq SC$

The former three conditions are the same as those for software minimization. Condition (e) constrains the total bit length of parameters in a register. Condition (f) is the software constraint. Based on the above formulations, an ILP-solver can be used to obtain an exact minimization if *p* shared registers are used. To obtain a global exact minimization, the ILP-solver must be run *n/2* times with $p = 1, 2,....n/2,$ and the best result then chosen.

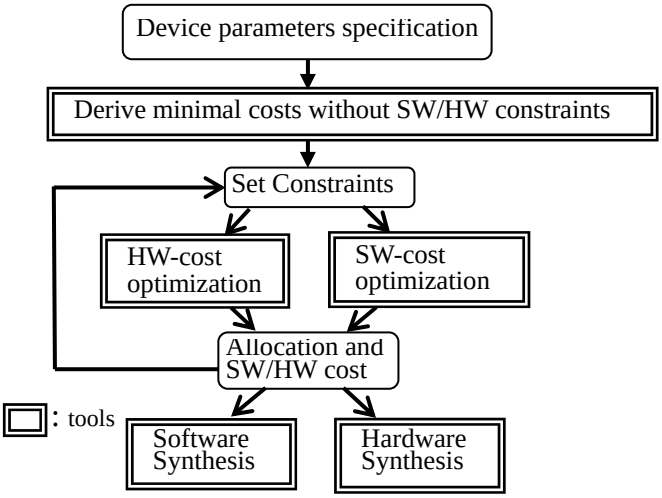

Fig. 6. An HW/SW codesign flow for I/O device controllers.

4 Design Methodology

The flow of the proposed design methodology is shown in Fig. 6. The device-parameters specification, defines the bit length of each device parameter, the members of each parameter group and access profiles f_g^r, f_g^r, f_i^r and f_i^w. The bit length of a parameter and groups' members are fixed, while the access profiles are determined by a dedicated application. Given the device-parameter specification, the system first derives two allocations on the assumption that the software (hardware) constraint is unlimited when minimizing the hardware (software) cost. One has a minimal software cost and the other minimal hardware cost. They are two extreme solutions of the design space. Then a user can give a hardware constraint (or a software constraint) and obtain an allocation having a minimal software cost (or hardware). The inner loop allows a user to iteratively refine the solution to meet a certain purpose. The acceptable allocation for all device parameters can then be fed into a software synthesis tool [6, 8], which generates the low level codes of parameter-access functions. Furthermore, the allocation will be used by a hardware synthesis tool to generate I/O registers.

In both HW and SW optimizations, we first handle each device parameter group separately. Then we allow parameters in different groups to share registers if the following situations occur: (1) if 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 solution with exact minimum cost, it is time-consuming for handling large cases. Heuristic approaches are presented here. The heuristic algorithm starts from an initial allocation with the smallest number of registers, which corresponds to an exact solution of a bin-packing problem. Then, the

allocation is iteratively refined to obtain a lower cost until the constraint is violated or no further improvement can be obtained. The strategy is applied to both software and hardware optimization. Each refinement process relies on proven lemmas, which have been presented in [5]. To show these lemmas, a new notation is defined as follows:

Definition 6 ***Φ'(V_g, V', m-k, 0, p-s, 0)***: An allocation $\Phi'(V_g, V', m-k, 0, p-s, 0)$ is derived from $\Phi(V_g, m, 0, p, 0)$ by reallocating a set of parameters V', $V'=\{X_j \mid j \in 1,\ldots, m\}$ and $|V'|= k$, to single registers. The remaining parameters are repacked into $p-s$ shared registers, where $k \geq 1$, $s \geq 0$ and $k \geq s$.

The lemmas exploited in our approach are described in the followings, whose proofs can be directly derived by subtracting $C_s(\Phi)$ from $C_s(\Phi')$.

Lemma 1: Given f_g, f_i^r and f_i^w, $i= 1,\ldots,n$, for a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$ and an allocation $\Phi(V_g, m, 0, p, 0)$, if $\Phi'(V_g, V', m-1,0, p-1,0))$ exists, then $C_s(\Phi')$ is smaller than $C_s(\Phi)$.

Lemma 2: Given f_g, f_i^r and f_i^w, $i= 1,\ldots, n$, for a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$ and an allocation $\Phi(V_g, m, 0, p, 0)$, $C_s(\Phi'(V_g, \{X_j\}, m-1, 0, p, 0)) < C_s(\Phi(V_g, m, 0, p, 0))$ if and only if $\frac{C_2 f_j^r + (C_1+2C_2) f_j^w}{C_1 - C_2} > f_g$.

The SW-optimization algorithm can be found in [5]. The HW-optimization part is shown in Fig. 7. An initial allocation with the minimum number of registers is derived by an exact bin-packing procedure, which uses a branch-and-bound approach as proposed by Martello and Toth [7]. Starting from the initial allocation, the solution is iteratively refined. If the software cost of the initial allocation is larger than the given software constraint (SC), the algorithm runs *bin_packing_maximize_single()*, which tries to derive a better allocation according to Lemma 1. Then, if the constraint is still not met, the Lemma 2 is applied.

```
MinmizeHCostUnderSC (VarList, f_g , f^r List, f^w List, SC) {
//SC : Software Constraint
// Let  Q(j) = (C2 f_j^r + (C1+2C2) f_j^w) / (C1 - C2) > f_g
//Aloc: An allocation of device variables
// Aloc-> ShVarlist: The variables packed in shared registers
// Aloc->RN : The number of registers used in the allocation
bin_packing(Varlist, & Aloc ); //initial
if ( SCost(Aloc) > SC )
   bin_packing_maxmize_single(Varlist, &Aloc);
while ( SCost(Aloc) > SC) {
   if Q(j) is the largest in Aloc->ShVarlist and Q(j) >1
      move the Var_j in   Aloc->ShVarlist to a single register;
   if no such j exists, return no-solution;
    }
 return Aloc;
}
```

Fig.7. The proposed algorithm for HW-Cost minimization under an SW constraint.

The solution obtained by the algorithm in Fig. 7 can be further reduced by using group-shared registers. However, as shown in Table 1, the use of group-shared registers increases the software cost if $f_g^w > 0$ for some group to be allocated in the group-shared register. Hence, the proposed algorithm firstly sorts groups according to the f_g^w. Then, two registers used by two different groups which have smaller f_g^w are selected to be merged. The merging is tried repeatedly until the software constraint is violated. Fig. 8 shows the procedure.

```
UseGroup-SharedRegisters( ) {
//Assume there is K different device-parameter groups. Without loss of
generality, let groups GP1 , GP2,..., GPk, be the increasing list of fg^w.
// T = GP1
for i=2 to K {
   try to merge any pair of registers (x, y) to be a register, where x∈ T, y∈ GPi,
        and both are a shared or group-shared register ;
   calculate SW-cost;
   if the software constraint is violated, discard the merging and return;
   T = T ∪ GPi ;
   }
}
```

Fig. 8. An algorithm to further improve HW-cost by using group-shared registers.

5 Experimental Results

The proposed design methodology was implemented in C. Experimental results were obtained for several industrial devices. These devices are originally used to show driver synthesis in [8], and specified in Devil language. Their specifications define the bit length of each device parameter and the members of a group according to industrial IC documents. The access profiles (i.e. f_g^r, f_g^w, f_i^r and f_i^w) are given by ourselves. The value of the SW cost depends on the given values of f_g^r, f_g^w, f_i^r and f_i^w. To illustrate the influence of access profiles, two cases were evaluated: (a) $f_g^r + f_g^w >> f_i^r + f_i^w$ and (b) $f_g^r + f_g^w << f_i^r + f_i^w$. For simplicity, in both cases, we let all groups have the same values for f_g^r and f_g^w. And all parameters have the same values for f_i^r and f_i^w.

Table 2 shows the experimental results. The “industrial” column shows the hardware and software costs of the allocation in original Devil specifications [8]. The two costs are used as the hardware and software constraints. Under the two constraints, software and hardware optimizations were derived by the proposed heuristics. The column SOP and HOP indicate the results assuming no group-shared register being used. The results show that an allocation with minimal software (hardware) cost is not necessarily to use maximal hardware (software cost). Comparing the experimental results with industrial designs, the proposed approach can obtain design alternatives that reduce both software and hardware costs. The “GHOP” column shows the HW costs which are derived by applying

UseGroup-SharedRegisters() procedure. In several cases, the HW-cost can be further reduced. As for the program time, the results for all examples can be obtained within 30 seconds on a 2.4 GHz P4-based PC.

To investigate the design spaces for various application features, we derived the allocations with the minimal SW cost under various HW constraints. Figure 9 shows such an investigation using IDE as an example. The three curves show the results for three different frequency ratios of $f_g^r+f_g^w$ to $f_i^r+f_i^w$. Some observations are obtained. In case of $f_g/f_i=10$, the solution with the minimal SW-cost generally is also the one with the minimal HW-cost. In case of $f_g/f_i=0.1$, the SW-cost is almost inversely proportional to the HW-cost. In case of $f_g/f_i=2$, the result shows that even more hardware can be used, the SW-cost is not further reduced.

Table 2. Experimental results

Devices	G/N(S)[2]	$f_g^r=f_g^w=10;\ \forall i, f^r_i=f^w_i=1$				$f_g^r=f_g^w=1;\ \forall i, f^r_i=f^w_i=10$			
		Industrial	SOP	HOP	GHOP	Industrial	SOP	HOP	GHOP
		HW cost (SW cost)[1]				HW cost (SW cost)			
BusMr	1/8(8)	2 (496)	2 (342)	2 (342)	2 (342)	5 (802)	5 (678)	5 (790)	5 (790)
IDE	3/38(22)	11 (2506)	5 (1490)	5 (1516)	3 (2116)	11 (4468)	11 (4206)	9 (4380)	7 (4416)
X11	5/47(23)	13 (3534)	11 (2094)	11 (2146)	11 (2146)	13 (6234)	13 (5916)	11 (6214)	11 (2146)
8290	7/107(26)	48 (11344)	17 (4242)	17 (4249)	15 (5874)	48 (10678)	48 (9698)	37 (10384)	35 (10440)
83905	7/103(32)	45 (13398)	17 (4214)	17 (4244)	14 (4404)	45 (11022)	45 (9920)	46 (10196)	43 (10212)
Total		122 (31278)	52 (12382)	52 (12497)	43 (14540)	122 (33204)	122(31818)	108(31964)	96(27214)
Compared to the industrial		1 (1)	0.43 (0..40)	0.43 (0.40)	0.35 (0.46)	1 (1)	1 (0.96)	0.89 (0.96)	0.79 (0.82)

1. *The SW-cost is calculated by assuming $C_1=3$ and $C_2=1$.*
2. *G/N(S) : # of group / # of all parameters (# of parameters in the maximal group).*
3. *Industrial: The cost of the original allocation in the Devil specification.*
4. *SOP: SW minimization using the industrial HW cost as the constraint.*
5. *HOP: HW minimization using the industrial SW cost as the constraint.*
6. *GHOP: HW minimization by allowing different groups to share a register.*

6 Conclusion

We have proposed a HW/SW codesign methodology that allows a designer to seek a parameter allocation that reduces the software or hardware cost during the development of a programmable I/O controller and its associated device driver. The approach is favorable in development of embedded systems. The exact allocation problems 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 current industrial designs, the system can obtain design alternatives that reduce both software and hardware costs. Furthermore, the results also indicate design spaces for various application features.

Acknowledgements. The authors would like to thank Taiwan NSC for financially supporting this research under Contract No. NSC 94-2215-E-030-007.

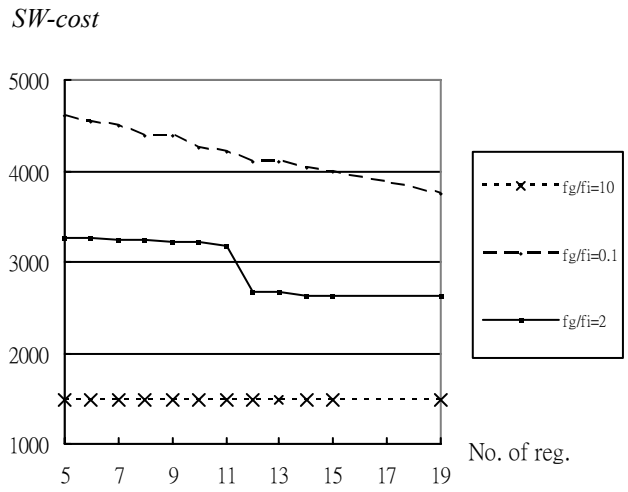

Fig. 9. SW-optimizations under various HW constraints. (IDE example)

References

1. C. L. Conway and S. A. Edwards, "NDL: A Domain -Specific Language for Device Drivers," Proceeding of ACM LCTES, pp. 30-36, 2004.
2. P. Chou, B. R. Ortega and G. Borriello, "Interface Co -synthesis Techniques for Embedded Systems," Proceedings of IEEE/ACM ICCAD, pp. 280-287, 1995.
3. T. Givargis, and F. Vahid, "Parameterized System Design," IEEE/ACM International Workshop on Hardware/Software Codesign, CODES, pp. 98–102, 2000.
4. G. Gognoit, M. Auguin and L. Bianco, "Communication synthesis and HW/SW integration for Embedded System Design," Proceeding of 6th international workshop on HW/SW codesing, pp. 49-53, 1998.
5. K. J. Lin, S. S. Huang and S. W. Chen, "A hardware/ software codesign approach for programmable IO devices," Proceedings of the 15th ACM Great Lakes Symposium on VLSI, pp. 323-327, 2005.
6. K. J. Lin and J. L. Chen "An Extension of C Preprocessor Directives for Device Programming", International Computer Symposium, pp. 1279-1284, Taiwan, 2004.
7. S. Martello and P. Toth, "Knapsack problems," Wiley, 1990.
8. F. Mérillon and G. Muller, "Dealing with hardware in embedded software: A general framework based on the Devil language," Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems, pp. 121 – 127, 2001.
9. S. Wang, S. Malik and R. A. Bergamaschi "Modeling and Integration of Peripheral Devices in Embedded Systems," DATE, pp. 136-141, 2003.
10. Q. L. Zhang, M. Y. Zhu and S. Y. Chen, "Automatic Generation of Device Drivers," ACM SIGPLAN Notices, pp. 60-69, June 2003.