September 2002

Phys. Chem. News 7 (2002) 63-70 PCN

A HIERARCHICAL REGISTER OPTIMIZATION APPROACH

M. Fettach[1]*, A. Hamdoun[1], O. Sentieys[2]

[1] *Laboratoire de Traitement de l'Information, Faculté des Sciences Ben M'Sick, B.P. 7955 , Sidi Othmane, Casablanca, Morocco*

[2] *ENSSAT, LASTI, Université de Rennes I, 6 Rue de Kérampont, 22305 Lannion, France*

** Corresponding author. Email: fettachm@yahoo.fr*

Received : 14 November 2001; revised version accepted : 06 June 2002

Abstract

A hierarchical register allocation approach in high-level synthesis is presented. First, we accomplish the trivial register allocation and then we attempt to optimize the number of required registers. In this work, we extend conventional register allocation algorithms to handle behavioral descriptions containing conditional branches and loops. However, in our approach the register optimization will carried out with explicit consideration of interconnection cost. Results show that our approach is more efficient for data flow graphs that contain nested conditional blocks and loops.

Keywords: Hierarchical register allocation; Register merging; Interconnection cost.

1. Introduction

High-Level Synthesis (HLS) is the design process which transforms a behavioral description of a digital design into its description of Register Transfer Level (RTL) structure [1]. Two major tasks are usually distinguished in HLS process: Scheduling and Hardware allocation. Hardware allocation involves assigning operations to functional units, allocating variables to registers and providing interconnections between functional units and registers. It's usually subdivided into three interdependent subtasks: operation assignment, register allocation and data transfer allocation.

This paper is concerned with the register allocation. Variables, whose source and destination operations are scheduled in different states, have to be stored in registers during the intermediate state transitions. Register allocation is the problem of mapping variables onto a minimum set of registers according to their lifetime analysis. The lifetime of a variable is the time of a period in which the value of the variable must be saved in a register. Behavioral descriptions usually contain control structures such as conditional branches and loops. In order to synthesize efficient circuits, it is important to deal with these structures especially in the register allocation. Since the conventional register allocation algorithms (clique partitioning algorithm [2, 3], left edge algorithm [4], bipartite weighted matching algorithm [5]) are based on data dependencies analysis in basic blocks, they do not deal with conditional branches and loops. However, having less registers does not necessarily guarantee that the final design will be optimal. Register allocation will have a major impact on interconnection cost. The interconnections constitute a substantial part of register transfer level design cost. Omitting them results in implementations that are far from optimal. The multiplexers and buses that have to be added have a large impact on both the area and the delay of implementations. In order to have significant savings in interconnect, some informations on source and destination operations of the registers are needed. These informations are deducted from functional units binding that we assume has been performed. If two operations have been mapped to a same functional unit, it will be advantageous to merge registers at their inputs and outputs. Kim et al [6] transform a data-flow graph with conditional branches into one without conditional branches by doing some pre-merging of variables. The research of the sets of variables that can be merged is modelled by a 0/1 Linear Programming (ZOLP) formulation. Since the ZOLP formulation increases rapidly with the number of local mutually exclusive variables, this approach is applicable only to very small problems. Although this approache can find near-optimal allocations in the existence of conditional branches, it does not consider the effect on interconnection cost. Next, Park et al [7] extend the previous transformational technique to handle cyclic data-flow graphs with conditional branches. In such cases, two ZOLP formulations are used, that proves the large increase in the CPU time. The first one is for transforming mutually exclusive variables into non-mutually exclusive variables while maximizing the sum of overlaps. The second formulation is used for allocating registers to the transformed variables while minimizing the number of register transfer operations. As was pointed out above, the interconnect cost is not considered.

In this work, we propose a hierarchical approach that consists in extending conventional allocation techniques to handle Data Flow Graphs (DFG) that contain nested conditional branches and loops, by considering data dependencies and control structures in the behavioral specification. However, our approach takes into account the effect of register optimization on interconnection cost. In our approach, we first perform the trivial register allocation, and then the problem of register allocation can be seen as a register optimization problem.

This paper is structured as follows. Some definitions and notations are given in section 2. Our hierarchical register optimization algorithm is described in section 3, followed by experimental results in section 4. Finally, section 5 concludes the paper.

2. Definitions and notations

In the following we give some definitions of terms used in the remainder of this paper. Since, we first perform the trivial register allocation and then we try to optimize the number of required registers, registers and variables will be used in an interchangeable manner.

A state graph SG = (S, E_S) is a directed graph possibly cyclic. Any node $S_i \in S$ represents a state and it is annotated by the set of operations scheduled in this state, and any unidirectional edge $e_{ij} = (S_i, S_j) \in E_S$ represents a state transition from the state S_i to the state S_j. We represent a scheduled Data Flow Graph (DFG) by a State Graph according to Moore's model [8], where any state of a conditional block is divided in some states, said coupled states, so that the mutually exclusive operations will be executed in distinct states. Generally, the number of states in the mutually exclusive branches is not the same. In such cases, some dummy states will be added to the shortest branches, so that all branches have the same length that the longest branch. Even though some supplementary states have been added, the behavioral of the digital design must not be modify. The Fig. 1(b) shows a SG derived from the scheduled DFG (Fig. 1(a)) given by Park et al in [7]. We assume that the scheduling and the functional unit binding have been done previously. The design can be described by a SG. The SG includes informations on both control and data flows, on the schedule and on functional unit binding. Each state of the SG is annoted by operations scheduled in this state, and by the functional units bound to these operations.

A register is said to be defined in a state if there exists an operation scheduled in this state that can possibly modify its content.

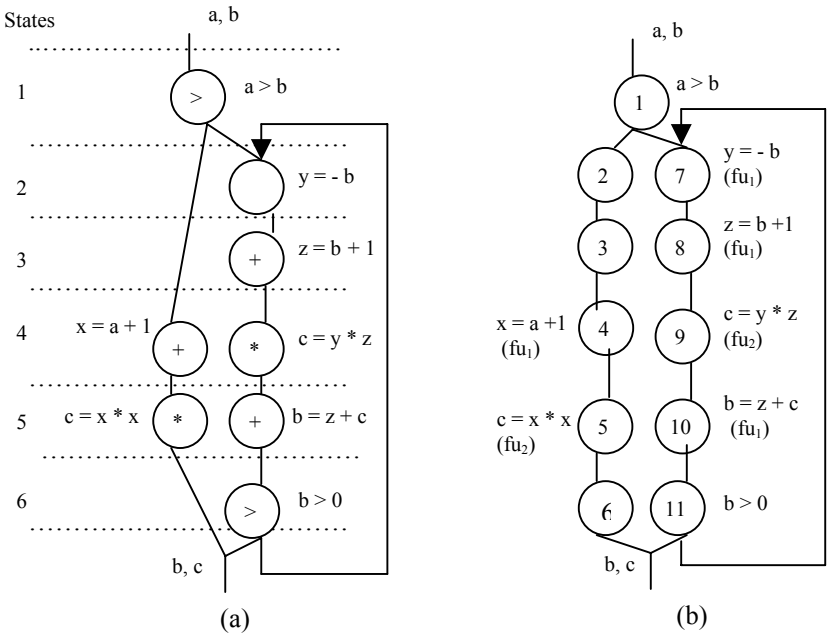


Figure 1: (a) A scheduled DFG and (b) The associated State Graph.

A register is said to be used in a state if it appears as operand in the expression of an operation scheduled in this state.

A register is said to be useful in a state, if it contains the value of a variable that might be used later. A register is useful from the time where it is first written until the time that its content is last read.

The utility phase of a register is the time of period (subset of states, not necessarily contiguous) during which the register is useful. The determination of utility phases of registers is based on the analysis of data flow in the description program. We can represent the utility phase of a register R_i by an interval $<SS(R_i), ES(R_i)>$, where the Starting State of the register R_i ($SS(R_i)$) is the state at which the register R_i is defined and the Ending State of the register R_i ($ES(R_i)$) is the state at which the register R_i is used for the last time [8]. Utility phases of registers can be regrouped in a table said the utility phases table.

Two registers are said to be compatible if they are not useful simultaneously, i.e. if their utility phases are not overlapping.

Two registers are said to be mutually exclusive if they are defined and used in different branches of a conditional block.

A register is said to be local with respect to a conditional block if it is useful only inside of the conditional block.

A register is said to be global with respect to a conditional block if it is useful both inside and outside of the conditional block.

A source operation of a register is the operation whose output operand should be mapped into this register.

A destination operation of a register is an operation whose one of its input operands has been bound to this register.

Two registers have source operations (or destination operations) in common[1] if their source (or destination) operations are mapped into a same functional unit.

Two registers have different source operations (or destination operations) if their source (or destination) operations are bound to different functional units.

To facilitate our representation, we assume that any conditional block has only two branches (b^l: the left branch, and b^r: the right branch) and we define the following notations:

- R_i^l: a register useful in b^l,
- R_j^r: a register useful in b^r,
- G_m^l: a group of compatible registers useful in b^l,
- G_n^r: a group of compatible registers useful in b^r.

3. The hierarchical register optimization algorithm

Our approach employs a hierarchical bottom-up method. That is why, a Flow Graph FG (B, E_B) is generated from the SG. The nodes $b_i \in B$ are basic blocks and the edges define a precedence relationship in the global control flow: there is an edge $e_b = (b_i,b_j) \in E_B$ if and only if control flow can be transfered directly from basic block b_i to basic block b_j. The Fig. 2(b) shows an example of a FG constructed from the SG given in Fig. 2(a). Any node of the FG has some attributes. In the Fig. 2(c), we have indicated in particular List of States (LS), List of Used Registers (LUR) and Priority Index (PI) for basic blocks b_1, b_2 and b_3. The priority indexes are affected to different blocks of a FG so that the innermost blocks should have the highest priority index. Blocks representing branches of a same conditional block must have a same priority index.

3.1 Impact on interconnection cost

The register merging can have a direct impact on interconnection cost. We will focus our discussion on interconnections between functional units and registers. The typical situations that occur when two registers are merged into a single register are (see Fig. 3):

(a) Merging registers that have different source and different destination operations,

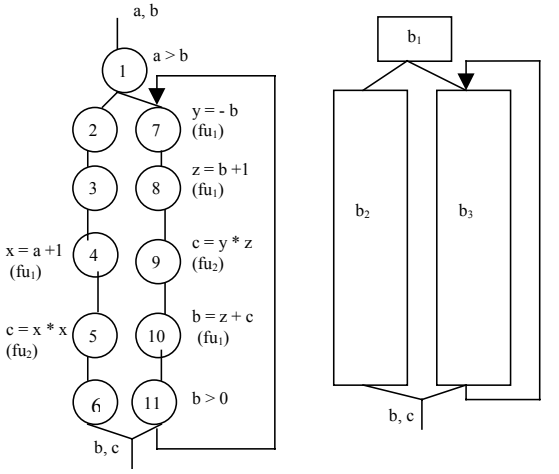
(b) Merging registers with the source operation of one register is the destination operation of the other register,

(c) Merging registers that have a common destination operation but different source operations,

(d) Merging registers that have a common source operation but different destination operations,

(e) Merging registers that have both a common source operation and a common destination operation.

The corresponding increase or decrease in multiplexers is also shown in the Fig. 3. Some situations show that there is a local interconnect reduction of merging registers associated to a same functional unit. Consequently, the register optimization can be done much better when the functional units binding has been done previously. Since the source and destination operations of any register are known, only the possibilities of register merging which result in a reduction in the interconnect cost should be selected.



$LS(b_1) = \{1\}$; $LRU(b_1) = \{a, b\}$; $PI(b_1) = 1$; …

$LS(b_2) = \{2, 3, 4, 5, 6\}$; $LRU(b_2) = \{a, b, c, x\}$; $PI(b_2) = 2$; …

$LS(b_3) = \{7, 8, 9, 10, 11\}$; $LRU(b_3) = \{b, c, y, z\}$; $PI(b_3) = 2$; …

(c)

Figure 2: (a) A State Graph, (b) A corresponding Flow Graph and (c) Somes attributes of basic blokcs.

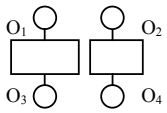
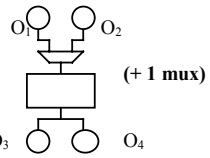
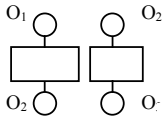
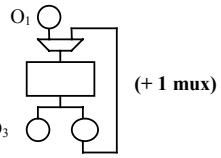
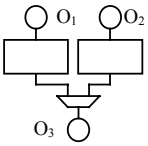
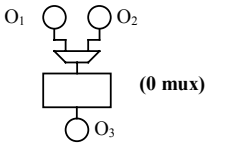
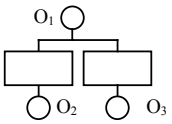
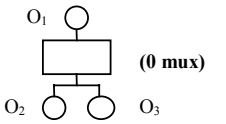
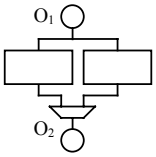
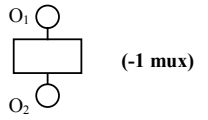
Case	Before merging	After merging
(a)		 **(+ 1 mux)**
(b)		 **(+ 1 mux)**
(c)		 **(0 mux)**
(d)		 **(0 mux)**
(e)		 **(-1 mux)**

Figure 3: Effect of register merging on interconnect.

3.2 Register merging

After that the trivial register allocation has been done, some registers can be merged into a single register, if their utility phases do not overlap (unconditional register merging), or if they are mutually exclusive (conditional register merging).

3.2.1 Unconditional register merging

In each branch b_i of a conditional block, we can define the unconditionally register merging in terms of a bipartite weighted matching between the set of registers useful in this conditional branch, $R = \{R_1, R_2, \ldots, R_r\}$, and the set of compatible registers groups $G = \{G_1, G_2, \ldots, G_{rmin}\}$, where any group G_j represents a set of registers to be merged, and r_{min} is the minimal number of registers required beforehand determined. There is a weighted edge, e_{ij}, between R_i and G_j if and only if the register R_i can be merged with registers in the group G_j. The weight w_{ij} on the edge e_{ij} reflects the impact on interconnection cost due to assigning the register R_i to the group G_j:

$$W_{ij} = C_1(X_{ij} + Y_{ij}) + C_2 Z_{ij} \quad (1)$$

where, C_1, C_2: are parameters to tune the interconnection cost,

X_{ij} = 1, if none of the registers that have been assigned to G_j has a source operation in common with R_i,
0, otherwise.

Y_{ij} = 1, if none of the registers that have been assigned to G_j has a destination operation in common with R_i,
0, otherwise.

Z_{ij} = number of different source operations of all registers belong to group G_j when register R_i is attributed to G_j.

Each register R_i in R is attributed to a group G_j in G according to a minimal weighted matching for the bipartite weighted graph WBG = ($R \cup G$, E).

3.2.2 Conditional register merging

Two mutually exclusive registers R_1^l and R_2^r in a conditional block can be merged into one register $R_{1,2}$ since one of them will be used during an execution instance. To maximize the use of $R_{1,2}$, registers R_1^l and R_2^r must have the maximal overlap. The overlap between registers R_1^l and R_2^r is defined to be the total number of pairs of coupled states in which R_1^l and R_2^r are useful simultaneously. In the case of merging between two groups of registers G_i^l and G_j^r, the overlap is computed as the sum of the overlaps between registers in groups G_i^l and G_j^r. The Fig. 4 illustrates overlaps between some registers and groups of registers. The registers will can be merged separately or by groups. There are four possibilities of merging any register R_i^l useful in the left branch (b^l) with registers useful in the right branch (b^r):

a. Merging R_i^l with a single register R_j^r,
b. Merging R_i^l with a group of registers G_n^r (if $|G_n^r(R_j^r)|>1$)[2,3],
c. Merging a group G_m^l that contains the register R_i^l (if $|G_m^l(R_i^l)|>1$), with a single register R_j^r,
d. Merging a group G_m^l that contains the register R_i^l (if $|G_m^l(R_i^l)|>1$) with a group of registers G_n^r(if $|G_n^r(R_j^r)|>1$).

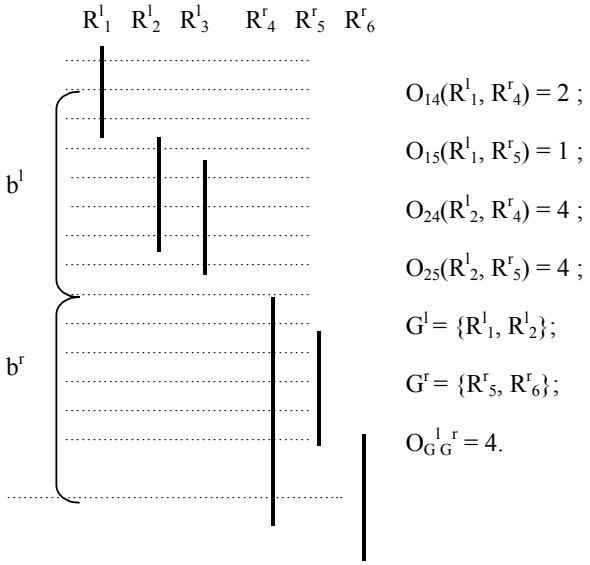


Figure 4: An example of overlaps between Registers.

3.3 Objective function

In order to maximize the use of merged registers and to reduce the multiplexers cost (number of inputs) required in certain cases, we introduce an objective function F. The possibility that presents more advantages is the one that maximizes the value of the objective function F. The expression of the objective function F depends upon the case under consideration:

3.3.1 Merging of two registers R_i^l and R_j^r

$$F(R_i^l, R_j^r) = O_{ij} SD(R_i^l, R_j^r) \quad (2a)$$

where, O_{ij}: Overlap between registers R_i^l and R_j^r,

$$SD(R_i^l, R_j^r) = \frac{\min(P(R_i^l), P(R_j^r))}{\max(P(R_i^l), P(R_j^r))}$$: the sharing degree between registers R_i^l and R_j^r.

$P(R_i^l)$: the utility phase of the register R_i^l,

$P(R_j^r)$: the utility phase of the register R_j^r,

Although in Fig. 4 registers R_2^l and R_4^r on the one hand, registers R_2^l and R_5^r on the other hand have the same number of overlaps ($O_{24} = O_{25} = 4$), their respective merging does not present the same advantage. In fact, since $P(R_2^l) = P(R_5^r)$ then SD $(R_2^l, R_5^r) = 1$ this means that the merged register $R_{2,5}$ is used during the same interval for each execution instance. Whereas, since $P(R_2^l) < P(R_4^r)$ then $SD(R_2^l, R_4^r) = 0.66$ this means that the merged register $R_{2,4}$ is not used during the same interval for each execution instance. In other words, in the left branch the merged register $R_{2,4}$ is used only 66 % of the interval if the merged register $R_{2,4}$ is used in the right branch.

3.3.2 Merging of a register R_i^l (or R_i^r) with a group of registers G_j^r (or G_j^l)

$$F(R_i^l, G_j^r) = \frac{1}{|G_j^r|} O_{ij} SD(R_i^l, G_j^r) \quad (2b)$$

where, $|G_j^r|$: the cardinal of the group G_j^r,

O_{ij}: the sum of overlaps between register R_i^l and all registers in G_j^r,

$$SD(R_i^l, G_j^r) = \frac{\min(P(R_i^l), P(G_j^r))}{\max(P(R_i^l), P(G_j^r))}$$: the sharing degree between register R_i^l and registers in G_j^r.

$P(R_i^l)$: the utility phase of the register R_i^l,

$P(G_j^r)$: the entire utility phase of all registers in G_j^r,

Note that the objective function $F(G_i^l, R_j^r)$ of merging of a group G_j^l with a register R_j^r has the same expression that the objective function $F(R_i^l, G_j^r)$ enough for replacing R_i^l by G_i^l and G_j^r by R_j^r.

3.3.3 Merging two groups of registers G_i^l and G_j^r

$$F(G_i^l, G_j^r) = \frac{1}{|G_i^l||G_j^r|} O_{ij} SD(G_i^l, G_j^r) \quad (2c)$$

where, $|G_i^l|$: the cardinal of the group G_j^l, $|G_j^r|$: the cardinal of the group G_j^r,

O_{ij}: the sum of overlaps between registers in G_i^l and in G_j^r,

$$SD(G_i^l, G_j^r) = \frac{\min(P(G_i^l), P(G_j^r))}{\max(P(G_i^l), P(G_j^r))}$$: the sharing degree between registers in G_i^l and registers in G_j^r,

$P(G_i^l)$: the entire utility phase of all registers in G_i^r,

$P(G_j^r)$: the entire utility phase of all registers in G_j^r,

However, since two global registers with respect to a conditional block can not be merged during the processing of this conditional block, then it is needless to compute overlaps between global registers and/or groups that contain global registers. In addition, note that a register can be merged only one time with some registers during the processing of the current conditional block. It follows that our approach does not compute values of the objective function for all possible pairs of registers and/or groups of registers and hence a gain in both memory and time.

3.4 Implementation

Our hierarchical register optimization approach is described by the **algorithm 1**. First, we determine the minimal number r_{min} of registers required, r_{min} is the number of registers useful in the state S_{max} in which the density of the utility phases table is maximal. The conditional and unconditional register merging are carried out in innermost conditional block first (block with the highest priority index), and they are carried out in the outermost conditional block last (block with the lowest priority index). For each step, the conditional block to handle is the one that has the highest value of the priority index. Since the global registers with respect to any conditional block can not be merged in this conditional block, we must identify them. For each iteration, we accomplish first the unconditional register merging in any conditional branch and next the conditional merging between registers useful in all conditional branches. As stated earlier, the unconditional register merging in each conditional branch is modelled as a matching on a weighted

bipartite graph WBG = ((R∪G), E). The registers useful in each conditional branch b_i^x (x=l for the left branch or r for the right branch) are divided into some clusters, such as registers in a cluster are mutually incompatible. Then, each register in a cluster C_k^x is attributed to a distinct group G_i^x according a minimal weighted matching found by using the Hungarian method [9]. Finally, each group G_i^x contains a set of registers to be unconditionally merged into one register. Then, we perform the conditional merging of registers useful in each conditional branch with registers useful in the others conditional branches. Next, the unconditional register merging between registers – not yet merged- in any group determined in step (b), is then performed. Then, the utility phases table is folded at levels of states corresponding to the current conditional block. The folding operation consists of minimizing the size of the utility phases table (reduction of number of lines) by merging pairs of coupled states. This process is continued until all conditional blocks are handled. In the last step we perform the unconditional register merging between registers useful in the principal block of priority index 1 (the utility phases table and the FG completely reduced). The complexity of our hierarchical register optimisation algorithm is $O(r_{min}.r^2)$, r is the number of the registers assigned by the trivial allocation and r_{min} is the minimal number of registers required ($r_{min} < r$).

4. Experimental results

The proposed algorithm has been implemented in the C language, executed on PC Pentium III running at 700Mhz, and tested on some benchmarks, available in the literature [gcd, 10]; [ex1 and kim1, 7] and [maha, 10], that comprise nested conditional blocks and loops. The experimental results are summarised in the table 1. We have compared our results with results obtained by Park [7]. Our results show that our approach has sufficient performance in terms of run time and quality of solutions. Although we do not compare run time under the same computer condition , the run time of our approach is short enough, since the solution of 0-1 integer linear programming problems is in general very time consuming. Although we have found the same optimal number of registers for each example, our results differ from results in [7] by the fact that we have not the same sets of registers (or variables) to merge. Our approach selectively merges registers that reduce the interconnection cost.

	Ours					Park		
Examples	No. of cb(1)	No. of fu(2)	No. of reg.	No. of mux.(3)	CPU time (secondes)	No. of reg.	No. of mux.	CPU time (secondes)
gcd	1	1	2	2	0.04	2	-	0.17
ex1	1	2	2	4	0.12	2	-	0.66
kim1	2	3	8	20	0.72	8	-	9.55
maha	6	2	7	19	0.18	7	-	1.06

Table 1: Experimental results.

(1) cb: conditional branches,
(2) fu: functional units,
(3) We use a multiplexer with two input ports and one output port as an interconnection unit.

5. Conclusion

We have proposed a hierarchical register optimization approach that is more efficient for data flow graphs which contain nested conditional blocks and loops. In addition, we have taken into account the interconnection cost. This constraint is very important in current designs using deep submicron technology. Indeed, interconnections

have a large impact on both the area and the delay of implementations. Experimental results show that our approach is more efficient for data flow graphs that contain nested conditional blocks and loops.

References

[1] M. C. McFarland, A. C. Parker, R. Camposano, Tutorial on High-Level Synthesis, Proc. of the 25th Design Automation Conference, July 1988, pp. 330-336.
[2] C. -J. Tseng, D. Siewiorek, IEEE Transaction on CAD, 5, N°3 (1986) 379-395.
[3] P. G. Paulin, J. P. Knight, IEEE Transaction CAD, 8, N°6 (1989) 661-679.
[4] K. Kurdahi, A. Parker, REAL: A program for register allocation, Proc.24th design Automation Conference (1987) 210-215.
[5] C. -Y. Huang, Y. -S. Chen, Y. -L. Lin, Y. -C. Hsu, Data path allocation based on bipartite weighted matching, Proc. 27th Design Automation Conference (1990) 499-503.
[6] T. Kim, C. L. Liu, Journal of VLSI Signal Processing, 12 (1996) 265-285.
[7] C. Park, T. Kim,, C. L. Liu, Journal of VLSI Signal Processing, 9 (1998) 269-285.
[8] M. Fettach, A. Hamdoun, O. Sentieys, An Efficient Register Allocation for Data path Synthesis systems, Proc. 6th Maghrebian Conference on Computer Sciences, MCSEAI'2000, Fès, 1-3 November, 2000.
[9] C. H. Papdimitriou, K. Steiglitz, Combinatorial optimization, Prentice-Hall (1982).
[10] Benchmarks for the 6th international workshop on high-level synthesis (1991).

Algorithm 1

1. Determine the minimal number of registers required (r_{min}).
2. Identify the maximal priority index (PI_{max}).
3. while (PI_{max} >1), do
For each conditional block of priority index PI_{max}, do
a. Identify global registers with respect to the current conditional block,
b. Apply the improved Bipartite Weighted Matching Algorithm (BWMA) to have groups of registers to be unconditionally merged in each conditional branch,
c. Identify, merge registers and/or groups of registers that maximize the objective function F,
d. Merge registers in groups (determine in step (b)) not yet merged.
e. Fold the utility phases table at levels of mutually exclusive states correspondent to the current conditional block.
f. Update the Flow Graph.
g. Decrease the value of the priority index ($PI_{max} = PI_{max} -1$) and return to step 3.
4. Apply the improved BWMA to merge some registers useful in the principal basic block (PI = 1).
5. End.

Notes

1. Source (or destination) operations in common with respect two registers do not means necessarily that operations are of the same type, but means that operations are mapped to a same functional unit.
2. $G_j^x(R_i^x)$: a group of compatible registers, that contains the register R_i^x. The groups $G_j^x (R_i^x)$, with j = 1, 2, …, r_{min}, are determined in the step of unconditional register merging.
3. $|G_j^x (R_i^x)| >1$ means that the group of registers $G_m^x(R_i^x)$ contains the register R_i^x and at least an other register useful in the conditional branch b^x.