Interconnect Optimization Techniques in Data Path Synthesis

C. A. Mandal P. P. Chakrabarti S. Ghose

Department of Computer Science & Engineering
Indian Institute of Technology
Kharagpur 721302, INDIA

Abstract

This work presents methods for interconnect optimization while performing register optimization and placement of registers in memory. It has been shown that the port assignment problem is similar to the register optimization problem that also considers interconnect optimization. The basic formulation of the register-interconnect optimization problem has been posed as that of finding a clique cover under some constraints. A measure of the interconnect area has been defined. This measure has been integrated with a traditional well known algorithm for performing clique cover to achieve both register and interconnect minimization. Both the techniques show favourable results.

Keywords: VLSI, Data Path Synthesis, Register Allocation, Port Assignment

1 Introduction

Data path synthesis often starts with a procedural description of the behaviour of the target system. This description is transformed into an intermediate representation that clearly shows the dataflow and the control flow embedded in the original description. The intermediate code is often organized as a graph of basic blocks called the flow graph [1]. The translation process introduces several temporary variables. Variables are mapped on registers in the final design. Some of the registers may be profitably clubbed into memories. Since registers take up considerable area and also contribute significantly to the interconnect cost, it is desirable to optimize the number of registers used in the final design.

Live variable analysis on the flow graph generates a profile of the times when the variables carry relevant values, ie. when they are live. Two variables that not live at the same time can be placed on the same register. On the other hand, two registers having overlapping life times but disjoint access times can be placed on the same single port memory, while a multiport memory is required to house registers that are accessed simultaneously.

Facet [4] and Real [3] represent two important techniques for register allocation. While Real purely performs register minimization for DAGs, Facet handles interconnect optimization in a separate phase. Pure register minimization produces designs with high interconnect complexity. Keeping this difficulty in view an algorithm was designed to perform register minimization with an interconnect cost lookahead. Register optimization alone is not sufficient to generate feasible designs, for even after minimization the absolute number of registers may still be large. Placement of registers on memories permits considerable reduction in the interconnection complexity. In order to avoid significant degradation in the performance only small size memories may be used, say upto eight cells. Owing to the large number of redundant registers in the intermediate level of the design, a large number of memories would be required. A proper optimization strategy would need to perform both register optimization and optmization using memories.

The presentation continues with studies of the register-interconnect optimization problem, the memory allocation problem and finally a scheme for overall optimization. Some results about the performance of the various strategies are also presented.

2 Register-Interconnect Optimization

2.1 Prologue

Register-interconnect optimization (RIO) is an intermediate stage in the synthesis process. The status of the design at the time of starting this step is described below. The initial description is transformed into one or more directed acyclic graphs (DAG) [1] to represent the data flow and the control flow, as well. Next, the DAGs are scheduled, possibly on the basis of a given operator set. Alternatively, the operator set may be fixed after scheduling is over. Finally the operations in the DAG are bound to specific operators. These scheduled DAGs where operations are bound to specific operators serve as the input to the RIO module.

2.2 Formulation

The starting point of the formulation is a suitable representation of the design space. Call this space D_R and the initial partial design d_{R_0}. The terms variables and registers are used interchangeably here. First a live variable analysis [1] is performed on the scheduled DAGs. This yields the sharability information between the variables and is stored in a matrix. The

0-8186-2465-5/91 $1.00 © 1991 IEEE

initial sharability matrix is called s_0. A sharability matrix s is symmetric and $s_{ij} = 1$ only if register r_i is sharable with register r_j. A cost function f_{C_R} is defined over D_R. This cost is a measure of the expected area of the data path of the system. A transformation function f_{T_R} defined over D_R maps the current design onto a new design as a result of merging two registers r_i and r_j. Registers r_i and r_j may be merged only if $s_{ij} = 1$. Another transformation function f_{T_S} is defined over all square matrices. The function f_{T_S} maps s to s' as a result of merging registers r_i and r_j to form a new sharability matrix.

The problem may now be stated as, starting with d_{R_0} and s_0, obtain a design d_{R_f} as a result of applying f_{T_R} on d_{R_0} and f_{T_S} on s_0 and then successively on the intermediate d_{R_i}'s and s_i's such that $f_{C_R}(d_{R_f})$ is minimum. Each application of f_{T_R} corresponds to the updation of the current partial design and each application of f_{T_S} corresponds to the updation of the current sharability matrix as a result of merging the two variables selected in the current step. This formulation suggests the use of a branch and bound strategy for obtaining the optimal solution with respect to the chosen f_{C_R}. It can be easily shown that this problem is NP-hard by reducing the clique cover problem to this problem.

2.3 Representation

Two aspects of the design are relevant at this stage, namely, the sharability information and the interconnection information. The representation of the former has already been described. Memories are not considered in the context of RIO. Ports, in this context, only refer to the ports at the interface of the module under consideration. The interconnection information may be summarized as follows:

- links from registers to operators
- links from operators to registers
- links from registers to registers
- links from operators to operators
- links from ports to operators
- links from registers to ports
- links from operators to ports
- links from ports to registers
- links from ports to ports

It is, however, not necessary to separately classify the inputs and outputs of the operators and the various ports. As the assignment of operators is already completed, these may be stored as fixed connection points. The left input, the right input and the output of each operator are considered to be distinct and fixed connection points. Registers are stored as variable connection points. Some of the variable connection points may be merged. Each connection point is assigned a distinct number. The design is now represented as an incidence matrix I. $I_{kl} = 1$ if component point l is incident on component point k. More than one connection point may be incident at a single connection point. In such a case a multiplexer is needed. Multiplexers are not explicitly represented.

2.4 Transformation

As a result of merging registers r_i and r_j, $i < j$, the incidence matrix is updated as follows. All points receiving from r_j now receive from r_i. All points feeding r_i and r_j now together feed into r_i. r_j is deleted from the incidence matrix. The sharability matrix s is updated to s' (say). In s' the row and column corresponding to r_j in s are absent. $s'_{ik} = 1$ iff s_{ik} and $s_{jk} = 1$. $s'_{pq} = s_{pq}$, $p, q \neq i$ and $p, q \neq j$.

2.5 Cost Function

The cost function f_{C_R} is a measure of the area occupied by the registers and the busses. This function should be easily computable as the time complexity of the algorithm would be unacceptably high if the time complexity of computing f_{C_R} is high. A measure derived by actually performing the routing and layout is very costly in terms of computation time. Even area estimators such as Plest [2] that avoid routing are not fast enough for our purpose.

The cost function used here is based on the register and multiplexer usage. The register cost is proportional to the width of the register and the multiplexer cost is proportional to the number of input channels and the width of the output. A linear combination of the register and multiplexer cost is taken as the total cost. This cost function is being called RMC. Each register or operator itself takes into account one bus at every input point. Multiple connections which enter a point require a multiplexer whose size varies depending on the number of lines it must multiplex. So it may be assumed that multiplexer and register cost are reasonable estimators of register-interconnect complexity. The minimization of this function would result in a reduction of the number of registers and multiplexers in the design. However, the interconnect complexity is tightly coupled with this cost function. Therefore, the bus usage is simultaneously reduced along with the registers and the multiplexers.

3 RIO Algorithm

3.1 Algorithm

Three factors have been kept in mind while designing the algorithm.

1. Merging of two registers in general produces a saving in area. Therefore the greedy algorithm must attempt to find register pairs that yield maximum reduction in the interconnect complexity. However, this approach could choose pairs that rapidly deplete the edges in the sharability graph and generate a design with too many registers.

2. By concentrating on register minimization alone, a design with high interconnection complexity could result.

3. It is therefore necessary to have a tradeoff between the two optimization criteria.

The following definitions enable a concise presentation of the algorithm.

Common vertex A vertex v is said to be common to vertices v_1 and v_2 if $s_{vv_1} = 1$ and $s_{vv_2} = 1$.

Deleted edge An edge (v, v_1) is said to be deleted on merging v_1 and v_2 if $s_{vv_1} = 1$ and $s_{vv_2} = 0$. The function $de(v_1, v_2)$ returns the number of edges deleted on merging v_1 and v_2.

Clique factor Let P_i be the vertex pairs in s having precisely i common vertices. Let $Q_i = \bigcup_{j=i}^{j=\infty} P_j$. The clique factor associated with Q_i is $cf(Q, i) = \frac{2|Q_i|}{i(i-1)}$.

The algorithm is outlined in figure 1.

3.2 Performance

The above algorithm has been tested extensively on randomly generated behaviours. The behaviours are generated on the basis of two parameters, viz. the number of variables and the number of arithmetic logic units (alu). Here alu's are used to denote hardware operators capable of performing some predetermined functions. For each behaviour the corresponding DAG is constructed and scheduled on the basis of the alu's generated along with the behaviour. Following a live variable analysis, register optimization is performed both with and without interconnect cost lookahead. The cost per bit of a register or a multiplexer is taken as one unit. A register is given twice the weight of a multiplexer. It is found that while the number of registers needed in both the cases is almost the same, the multiplexer cost in the first case is much less. The comparitive results are tabulated in figure 2. The saving is seen to increase with the number of alu's.

4 Memory Allocation

4.1 Prologue

Given a set of registers to be placed in a memory, it is necessary to distribute the references to the constituent members over its ports. A single port memory might suffice, or a multiport memory might be called for, depending on the distribution of the references. The port assignment should be such that the resulting interconnection is minimal. Sometimes the precise number of ports to be used is pre-specified and an optimal port assignment is to be determined, if one exists. In this work, the attempt is to determine the minimum number of ports that will result in an optimal interconnection. The optimality in the above case is defined with respect to the cost function f_{C_R} applied on a transformed design. The details of this transformation and its use to perform port assignment follow later. The same criteria which are used to judge the interconnection complexity of a design are used to measure the quality of a port assignment. It is then necessary to determine which registers are to be grouped together for placement in a particular memory. Some of the registers might even be left as they are. As in case of the register-interconnect minimization problem, it is possible to formulate a branch and bound algorithm for performing register allocation.

A simple greedy algorithm has been used to determine the register groups. At each step a port assignment is performed to determine the quality of the assignment and the best candidate is chosen. This approach is made possible by the polynomial time characteristics of the RI with RMC strategy.

4.2 Memory Allocation Algorithm

The algorithm described in figure 3 takes the initial design d_{M_0} and produces d_{M_f} as a result of grouping registers into memories, when possible. The register minimization aspect has not been touched upon in this algorithm. It is easily incorporated by performing a clique partitioning of R_0 after exiting from the loop statement. M records the final register groups.

4.3 Port Assignment

The design space is now called D_M. The design instance is represented as d_M and the cost function as f_{C_M}. The initial design d_{M_0} is in fact what would have been d_{R_0} if only register compaction were to be performed. The same methods are used to represent a design and to evaluate the cost function. Only the memories and their ports come in as new entities. They are, as usual, handled as connection points. The transformation function is f_{T_M}. It maps the current design to a new design as a result of grouping a set of registers into a memory. The mapping is guided by the port assignment. The details of f_{T_M} follow later.

4.4 Transformation to RI with RMC

Given any design d_M and a set of registers R_0 to be placed in a memory, an instance of RI with RMC is created for performing the port assignment. The transformation is now described. A variable connection point is defined in d_{R_0} corresponding to each reference to a member of R_0. Fixed connection points are defined for all other connection points in d_M. To construct the initial sharability matrix, two variables are marked sharable in s_0 if their corresponding references occur in different time steps. However, variables corresponding to multiple references to the same member in the same time step are marked sharable. This enables reads to the same member to be mapped on the same port, if needed.

Suppose that a reference to a member of R_0 is a data transfer to some connection point in d_M, ie. a read. The corresponding connection point in d_{R_0} is marked as the destination of the variable connection point in d_{R_0} corresponding to that particular reference. In case of a write the marking used is that of a source. This defines the transformation of the port assignment problem to RI with RMC.

4.5 Interpretation of Results

Each variable cluster obtained after solving the RI with RMC problem corresponds to a port. Each port is a new connection point in d'_M obtained from d_M. These connection points are essentially independent.

procedure reg_min

1. $d_R = d_{R_0}$
2. $s = s_0$
3. while (the sharability matrix is not exhausted) do
4. { determine the maximum x and the corresponding Q such that $cf(Q, x) \geq 1$; in the absence of such an x, let Q include all the edges in s.
5. determine the member (v_1, v_2) of Q which has the minimum value of $\frac{de(v_1, v_2)}{f_{C_R}(d_R) - f_{C_R}(f_{T_R}(d_R, v_i, v_j))}$
   /* the denominator represents the saving as a result of merging v_1 and v_2 */
   /* a trade–off is made between the deleted edges and the saving */
6. $d_R = f_{T_R}(d_R, v_1, v_2)$
7. $s = f_{T_S}(s, v_1, v_2)$
8. }
9. $d_{R_f} = d_R$

Figure 1: Register minimization algorithm.

no. var	no. alu	Registers				Multiplexers			Cost
		DAG[a]	NZ[b]	Z[c]	NZ/Z	NZ	Z	NZ/Z	NZ/Z
5	1	10.84	38.75	38.50	1.01	65.75	69.00	0.96	0.98
5	2	20.69	47.25	46.75	1.01	145.75	183.50	0.80	0.84
5	3	30.62	51.25	49.00	1.05	250.25	317.00	0.79	0.83
10	1	20.84	77.25	76.75	1.01	125.75	138.75	0.91	0.94
10	2	41.37	94.50	91.75	1.03	280.75	375.75	0.75	0.81
10	3	61.22	100.75	95.25	1.06	471.25	623.00	0.76	0.80
15	1	32.47	115.25	113.75	1.01	183.75	200.00	0.92	0.95
15	2	61.28	143.50	135.75	1.06	410.25	553.75	0.74	0.81
15	3	92.22	154.50	144.75	1.07	674.50	919.00	0.73	0.78
20	1	42.19	154.00	151.00	1.02	251.25	271.75	0.92	0.96
20	2	82.44	188.00	180.00	1.05	526.25	722.00	0.73	0.79
20	3	122.67	194.91	182.30	1.07	849.70	1206.79	0.71	0.75
25	1	53.41	193.75	190.75	1.02	312.75	340.50	0.92	0.95
25	2	103.28	229.50	219.00	1.05	636.00	886.75	0.72	0.78
25	3	153.03	249.50	234.00	1.07	1045.50	1523.00	0.69	0.74

[a] unoptimized number of registers
[b] cost when using RMC with non–zero mux. cost
[c] cost when using RMC with zero mux. cost

Figure 2: Performance of register minimization algorithm.

```
procedure mem_alloc
1.    d_M = d_{M_0}
2.    R = {r | r is a register in d_{M_0}}
3.    M = ∅
4.    loop
5.    {     if R = ∅ then break /* all registers already placed */
6.          r ← any register of R
7.          R = R − {r}
8.          R_0 = {r} /* the first register in the current memory */
9.          loop
10.         {     if R = ∅ or | R_0 | ≥ mem_size then break /* no further placement possible */
11.               find r ∈ R that maximizes
                  p = f_{C_M}(d_M) − f_{C_M}(f_{T_M}(d_M, R_0 ∪ {r})) /* the interconnect saving */
                  and leads to the usage of no more than max_ports ports
                  let p_max be this value of p
12.               if p_max ≥ 0 /* some saving has actually been made */
13.               {     R_0 = R_0 ∪ {r} /* include this register */
14.                     R = R − {r}
15.               }     else
16.                     break
17.         }
18.         if | R_0 | > 1 /* a memory must have atleast two members */
19.         {     M = M ∪ R_0 /* include this memory */
20.               d_M = f_{T_M}(d_M, R_0)
21.         }
22.   }
23. d_{M_f} = d_M
```

Figure 3: Memory allocation algorithm.

Comparison of results						
max. size[a]	max. port	memory usage	no. of mem.	multiplexer usage	no. of mux.	no. of lines multiplexed
6	3	3 port mem; 1 no. 2 port mem; 2 nos.	3	3 inp. mux.; 2 nos. 2 inp. mux.; 1 no.	3	8
8	3	3 port mem; 2 nos.	2	3 inp. mux.; 1 no. 4 inp. mux.; 1 no.	2	7
8	2	2 port mem; 3 nos.	3	3 inp. mux.; 2 nos. 2 inp. mux.; 1 no.	3	8
6	2	2 port mem; 3 nos.	3	3 inp. mux.; 1 no. 2 inp. mux.; 2 nos.	3	7
-	-	2 port mem[b] 3 nos.	3	2 inp. mux.; 1 no. 3 inp. mux.; 3 nos.	4	11
-	-	3 port mem[c]; 2 nos.	2	2 inp. mux.; 4 nos.	4	8

[a]Size of the memory
[b]Details of design in [5]
[c]Details of design in [5]

Figure 4: Performance of memory allocation algorithm.

As mentioned earlier each variable corresponds to a memory reference. If it is a read then the connection point corresponding to the source is incident on the port. On the other hand if it is a write then that connection point receives from the port. This also defines the transformation function f_{T_M}.

4.6 Performance

The memory allocation strategy using the aforementioned port assignment method has been tested against a design reported in [5]. The results are tabulated in figure 4. Memory allocation was performed on the reported design for memory sizes of six and eigth, and port sizes of two and three. The last two rows of the table indicate statistics of the reported design in [5] when only two port memories are permitted and when only three port memories are permitted. It is seen to produce marginally better results.

5 Overall Strategy

5.1 Method

In order to obtain feasible designs both register optimization and placement of registers on memories is required. The basic source of optimization through the use of memories comes from our ability to distribute the accesses to the constituent registers to reduce multiplexer usage. Two registers writing to the same terminal, at different times, placed in different memories entail the use of a multiplexer. It is therefore desirable to place registers having common targets in the same memory. In order to reduce the number of memories required such registers also need to be suitably merged, subject to their compatibility. But this is just the direction in which register-interconnect optimization works.

In view of this observation, first a RIO is run on the intermediate design available after scheduling is performed. Subsequently the registers are grouped into memories using the memory allocation algorithm.

5.2 Performance

The performance of the above strategy was tested on randomly generated behaviours, as in case of RIO, but on a much smaller set. For each behaviour and the corresponding set of alu's two sets of experiments were performed. On set involved performing RIO with the interconnect cost lookahead forced to zero and then doing memory allocation. The second set involved RIO with proper interconnect cost lookahead and then doing memory allocation. In both the cases memory allocation produced significant cost reduction. However, the second experiment seems to produce better results than the first only when RIO in the second case produces a design with much lower cost than the first. Otherwise the experiments produce similar results.

References

[1] Ullman J. D. Aho A. V., Sethi R. ' *COMPILERS Principles, Techniques and Tools.* Addison-Wesley Publishing Company, June 1987.

[2] Parker A. C. Kurdahi F. J. 'Plest: A program for area estimation of VLSI integrated circuits. *Proceedings of the 23rd Design Automation Conference*, 1986.

[3] Parker A. C. Kurdahi F. J. 'Real: A program for register allocation. *Proceedings of the 24th Design Automation Conference*, 1987.

[4] Siewiorek D. P. Tseng C. J. 'Automated synthesis of data paths in digital systems. *IEEE Transaction on Computer Aided Design*, CAD-5(3), July 1986.

[5] Majithia J. C. Majumdar A. K. Wilson T. C. Banerjee D. K.' Optimal allocation of multiport memories in datapath synthesis. *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, 1989.