Computing and Informatics, Vol. 22, 2003, 427–438

HIGH-LEVEL TECHNOLOGY MAPPING FOR MEMORIES

Haifeng Zhou, Zhenghui Lin, Wei Cao

Department of Electronics Engineering
Shanghai Jiao Tong University
Shanghai, 200030, P. R. China
e-mail: heaven@sjtu.edu.cn

Manuscript received 7 January 2003; revised 19 May 2003
Communicated by Norbert Frištacký

Abstract. In this paper, we consider memory-mapping problems in High-Level Synthesis. We focus on the port mapping, bit-width mapping and word mapping, respectively. A 0-1 Integer Linear Programming (ILP) technique is used to solve the mapping problems, which synthesizes the source memory using one or more memory modules from a target memory library at a higher level. This method can not only perform bit-width mapping and word mapping, but it can also perform port mapping at the same time. Experimental results indicate that ILP approach is an effective method for memory reuse in high-level synthesis.

Keywords: Circuit CAD, digital circuit design, high-level synthesis, technology mapping

1 INTRODUCTION

Current VLSI designs have reached complexities of millions of gates and they are expected to grow even higher in the coming years. Systems of such complexity are very difficult to design by handcrafting each transistor or by defining each signal in terms of logic gates, since human designers cannot do an effective job for problems involving a large number of objects. For systems of such complexities, even the traditional design automation tools such as logic synthesis and physical design tools cannot provide good solutions in reasonable amounts of time. It is

necessary to develop design methodologies that can handle systems of higher complexity. Therefore, the idea of high-level design and design reuse [1] came into being.

Technology mapping is a very important step in digital system design, which transforms the technology independent structure description to the technology dependent library. Based on the complexity of the cells used in mapping, the library mapping could be categorized into different levels. At the logic level, library-mapping implements a logic level design by using logic level cells from a library [2, 3]. At the system level, mapping can be performed between system-level components such as processors, memories and interface units. MICON [4] is a system that tries to reuse the off-shelf system-level parts such as processors, memories and peripherals to build a single-board computer system. Smith [5] presented a polynomial algorithm for arithmetic component matching at an abstract level.

Memory modules are commonly used in data intensive applications in the fields of speech, image and video processing that require a significant amount of storage capability. Thus, the memory subsystems become an important focus of design. Hence, there is a need for efficient implementations of memory elements in these designs. Memory mapping has been reached in the case of ASICs where the mapping consists of selecting memory components from a library and/or selecting where the memory components are placed and the way in which they are connected to the hardware logic. Several studies were carried on utilizing memory modules in data path synthesis. FPGA on-chip memory banks are targeted in [6, 7]. Memory mapping for FPGAs with on-chip memories is addressed in [8]; however, only single-ported memory banks are assumed.

In this paper we present a 0-1 ILP approach for memory mapping, which implements a source memory module (s) from a set of memory modules from a target library (T). The approach is based on higher level of abstraction for memories: given the high-level specification of the source and the library modules in terms of word-count, bit-width and the port configurations, the approach implements the source memory module by using target memory modules in an efficient manner so as to optimize a user-given cost function.

2 PROBLEM DEFINITION

According to the definition in [9], at higher level of abstraction, a memory module m can be characterized by its size (number of words and bit-width for each word) and the amount of data access parallelism (port types and number of ports). Note that the three parameters (words, bit-width and ports) are orthogonal to each other, i.e. variation of one parameter is independent of others within a library of memory modules. Thus we need to consider all these parameters together in selecting a set of target memory modules to find a good implementation.

The memory-mapping problem is defined in terms of a source memory module s, a set of target memory modules T from a library and a user-given cost function C.

The memory mapping algorithm is a technique to implement a source memory module s, which has r_s read ports $\{r_l\}$ $(1 \leq l \leq r_s)$, w_s write ports $\{w_m\}$ $(1 \leq m \leq w_s)$ and rw_s read-write ports $\{rw_n\}$ $(1 \leq n \leq rw_s)$, from a set of memory modules in a target library $T = \{t_1, t_2, \ldots, t_i, \ldots, t_t\}$ $(1 \leq i \leq t)$ so as to optimize a user-given cost function C. The source memory module s has bit-width of B_s and has word count of N_s. Each t_i has r_i read ports $\{r_{ix}\}$ $(1 \leq x \leq r_i)$, w_i write ports $\{w_{iy}\}$ $(1 \leq y \leq w_i)$ and rw_i read-write ports $\{rw_{iz}\}$ $(1 \leq z \leq rw_i)$. Each t_i has the bit-width of B_i and has the word count of N_i. The implemented memory module should satisfy the source module requirements, i.e. the *ports*, *bit-width* and *word count* requirements.

Port Mapping. In order to meet the data access requirements of the source memory, the *port mapping* specifies the necessary and sufficient conditions that the target modules need to satisfy.

Bit-width Mapping. *Bit-width mapping* refers to the task of achieving the bit-width requirement of the source memory component s by using a set (one or more) of target memory modules from a library T. The whole bit-width of the implementation should be no less than that of the source memory component.

Word Mapping. *Word mapping* refers to the task of accomplishing the word-count requirement of the source memory components s by using a set (one or more) of target memory modules from a library T. The whole word-counts of the implementation should be no less than that of the source memory component.

Cost Function. The memory mapping algorithm is guided by the cost of the generated design. The cost of the synthesized source memory module is given by the combined cost of the various elements used in the design. These elements include address decode logic, target memory modules and output mux. For the sake of simplicity, we ignore the address decode logic and other costs. We refine these terms to generate a specific cost measure.

Area Measure: The area measure for the target memory modules is given by the sum of the area of all the target memory modules used in the designs; we take equivalent 2-input NAND gate counts as area cost function.

Delay Measure: The worst-case delay path for the synthesized memory goes through all the components in the design. The delay measure for the target modules is given by the maximum access delay for all the modules used in the design.

3 0-1 ILP MAPPING FORMULATION

Mapping Criteria. We present a 0-1 Integer Linear Programming (ILP) method to perform memory mapping in High-Level Synthesis. The mapping of the memory modules is based on the following observations:

1. For each read port of s, there should be either a target read port or a target read-write port of t_i to implement. For each write port of s, there should be either a target write port or a target read-write port of t_i to implement. For each read-write port of s, there should be either a target read-write port or a combination of a target read port and a target write port of t_i to implement.
2. To each port of t_i, there should be only one port of s to be mapped.
3. To the word mapping, the total words of the target memory should be no less than the word count of s.
4. To the bit-width mapping, the total bit-width of the target memory should be no less than the bit-width of s.
5. The cost of the implementation should be the least.

0-1 ILP Formulations. The following notations and variables are used in the formulation:

1. n_i is the number of t_i used to implement s;
2. C_i is the cost associated with t_i;
3. α_{lix} is a 0-1 integer variable such that $\alpha_{lix} = 1$, if port r_l of s is mapped to port r_{ix} of t_i, else $\alpha_{lix} = 0$;
4. β_{lkz} is a 0-1 integer variable such that $\beta_{lkz} = 1$, if port r_l of s is mapped to port rw_{kz} of t_k, else $\beta_{lkz} = 0$;
5. γ_{mjy} is a 0-1 integer variable such that $\gamma_{mjy} = 1$, if port w_m of s is mapped to port w_{jy} of t_j, else $\gamma_{mjy} = 0$;
6. ρ_{mkz} is a 0-1 integer variable such that $\rho_{mkz} = 1$, if port w_m of s is mapped to port rw_{kz} of t_k, else $\rho_{mkz} = 0$;
7. $\eta_{nix,jy}$ is a 0-1 integer variable such that $\eta_{nix,jy} = 1$, if port rw_n of s is mapped to port r_{ix} of t_i and port w_{jy} of t_j, under this circumstance, port r_{ix} is corresponding to port w_{jy}, they are combined to be mapped to the port r_{wn}; else $\eta_{nix,jy} = 0$;
8. λ_{nkz} is a 0-1 integer variable such that $\lambda_{nkz} = 1$, if port r_{wn} of s is mapped to port rw_{kz} of t_k, else $\lambda_{nkz} = 0$;

Now the problem of implementing s using T can be formulated as

Minimize

$$\sum_{i=1}^{k} n_i C_i \tag{1}$$

Subject to

$$r_s + rw_s - \sum_{i=1}^{k} n_i (r_i + rw_i) \leq 0 \tag{2}$$

$$w_s + rw_s - \sum_{i=1}^{k} n_i (w_i + rw_i) \leq 0 \tag{3}$$

$$r_s + w_s + rw_s - \sum_{i=1}^{k} n_i(r_i + w_i + rw_i) \leq 0 \tag{4}$$

$$\sum_{j=1}^{t} \sum_{y=1}^{n_i w_i} \alpha_{lix} + \sum_{k=1}^{t} \sum_{z=1}^{n_k rw_k} \beta_{lkz} = 1, \forall l \tag{5}$$

$$\sum_{j=1}^{t} \sum_{y=1}^{n_j w_j} \gamma_{mjy} + \sum_{k=1}^{t} \sum_{z=1}^{n_k rw_k} \rho_{mkz} = 1, \forall m \tag{6}$$

$$\sum_{i,j=1}^{t} \sum_{x=1}^{n_i r_i} \sum_{y=1}^{n_j w_j} \eta_{nix,jy} + \sum_{k=1}^{t} \sum_{z=1}^{n_k rw_k} \lambda_{nkz} = 1, \forall n \tag{7}$$

$$\sum_{l=1}^{r_s} \alpha_{lix} + \sum_{n=1}^{rw_s} \eta_{nix,jy} = 1 \tag{8}$$

$$\sum_{l=1}^{r_s} \beta_{lkz} + \sum_{n=1}^{rw_s} \lambda_{nkz} = 1 \tag{9}$$

$$\sum_{m=1}^{w_s} \gamma_{mjy} + \sum_{n=1}^{rw_s} \eta_{nix,jy} = 1 \tag{10}$$

$$B_s - \sum_{i=1}^{k} n_i B_i \leq 0 \tag{11}$$

$$N_s - \sum_{i=1}^{k} n_i N_i \leq 0 \tag{12}$$

The objection function (1) states that we want to minimize the total cost of the implementation. Constraints (2–4) guarantee that there are sufficient ports of t_i to be mapped to s. Constraints (5–7) map the ports of s to the ports of T, which is described in criterion 1. Constraints (8–10) guarantee that each port of T can only be mapped once, which is described in criterion 2. Constraints (11) and (12) make use of mapping criteria 3 and 4 respectively.

4 ALGORITHM IMPLEMENTATION

As the three sub-problems of memory mapping are orthogonal to each other, i.e., variation of one parameter is independent of another within a library of memory modules. We can consider these sub problems individually.

The bit-width mapping and word-mapping algorithm are given in Figure 1. Because they are similar, we integrated them into a concentrate function *sub_map*(s, T, *constrain*), where the constrain parameter is used to identify the mapping sub-problem. An enumeration scheme for all possible compositions of each bit-width or word-count was presented in a systematic fashion. In this algorithm the arrayed variable *part_map* keeps track of the best composition of target memory modules for each bit-width. For each target memory module t_i, we include $\lceil opt(s)/opt(t_i) \rceil$ instances in T_s, since a good mapping would require at most these many instances of t_i.

Finally, L_i enumerates all possible bit-widths or word-counts that can be composed using the first i memory modules from T_s.

The algorithm first initializes array variable *part_map*, T_s and L_0, and then successively enumerates all possible bit-widths or word-counts that can be composed using a subset of T_s. The function revise composes each element of L with a new target memory module and if the resulting composition is not a suboptimal one, the function stores the compositions in the new list and updates the local best solution *part_map*. The above algorithm uses the user-given cost function C to determine the quality of the mapping composition.

The algorithm described in Figure 2 is a scheme to perform the port-mapping task. The function tries to perform the mapping procedure in terms of mapping a source read port to a target read port or a target read-write port, a source write port to a target write port or a read-write port, and a source read-write port to a target read-write port or a target read port and a target write port.

The algorithm shown in Figure 3 describes the whole mapping algorithm. The algorithm generates the best bit-width mapping for each word-count *num* and creates a new memory module with the word-count of *num*. Then it performs the word mapping on the new memory modules. The result of word mapping is then passed to the port-mapping algorithm, which provides the final solution.

A Simple Example

Consider the example in Table 1, where the parameters of source memory and target memory modules are listed. The mapping results obtained by applying the approach presented above are shown in Figure 4. There are two possible solutions. Except for the glue logic, the former area cost is $11562 + 7564 = 19126$ gate counts and the latter one is $8680 \times 2 = 17360$ gate counts. Compared to the non-optimal solution in Figure 4 (a), the solution in Figure 4 (b) is preferred.

Parameters	Source(s)	Target Module				
		t_1	t_2	t_3	t_4	t_5
Word-count	256	256	256	128	128	128
Bit-width	12	8	4	12	8	4
R Ports	1	1	0	2	0	1
W Ports	1	0	0	2	1	0
RW Ports	2	1	2	1	2	2
Gate count		11562	7564	8680	6642	4636

Table 1. Parameters of the source and target memory modules

5 EXPERIMENTAL RESULTS

We applied our mapping algorithm to several examples. Our experiments use source memory modules derived from various memory-intensive designs reported in the

```
function sub_map(s, T, constrain)
begin
   if constrain=bit-width then
      opt(s) = B(s)
      opt(t_i) = B(t_i)
   else if constrain = word then
      opt(s) = N(s)
      opt(t_i) = N(t_i)
   end if
   for i = 1 to opt(s) do
      part_map(i) = Ø
   end for
      T_s = Ø
   for each t_i ∈ T do
      part_map[opt(t_i)] = t_i;
      for j = 1 to ⌈opt(s)/opt(t_i)⌉ do
         T_s = T_s + t_i;
      end for
   end for
   L_0 = Ø;
   for i = 1 to |T_s| do
      L_i = revise(L_{i-1}, T_si)
   end for
   return part_map[opt(s)]
end

function revise(L, t)
begin
   L_revise = L;
   for i = 1 to |L| do
      if opt(new_map) ≤ opt(s) or new_map does not have a t_j
            such that opt(new_map − t_j) ≥ opt(s) then
         insert new_map to L_revise
         update
      end if
   end for
   return L_revise
end
```

Fig. 1. Bit-width mapping and word mapping algorithm

function *port_map*(s, T)
begin
  for all read ports of s {
    if $\left(\sum_{i=1}^{k} n_i r_i \geq r_s\right)$ **then**
      map the read ports of s to the corresponding read ports of t_i;
    else
      map the read ports of s to the corresponding read ports of t_i;
      map the rest read ports of s to the corresponding read-write ports of t_i;
  }
  for all write ports of s {
    if $\left(\sum_{i=1}^{k} n_i w_i \geq w_s\right)$ **then**
      map the write ports of s to the corresponding write ports of t_i;
    else
      map the write ports of s to the corresponding write ports of t_i;
      map the rest write ports of s to the corresponding read-write ports of t_i;
  }
  for all read-write ports of s {
  if $\left(\sum_{i=1}^{k} n_i rw_i \geq rw_s\right)$ **then**
    map the read-write ports of s to the corresponding read-write ports of t_i;
  else
    map the read-write ports of s to the corresponding read-write ports of t_i;
    map the rest read-write ports of s to the corresponding rest read and write ports of t_i;
  }
end

Fig. 2. Bit-width mapping and word mapping algorithm

function *complete_map*(s, T)
begin
  $T_{word} = \emptyset$
  for each word-count num in T **do** {
    $T_{num} = \{t_i \mid W(t_i) = num\}$;
    $best\_bit = part\_map(s, T_{num}, bit\text{-}width)$;
    $T_{word} = T_{word} + T_{num}$;
  }
  $best\_map = part\_map(s, T_{word}, word)$;
  $solution = port\_map(s, t)$;
  return $solution$;
end

Fig. 3. Complete mapping algorithm

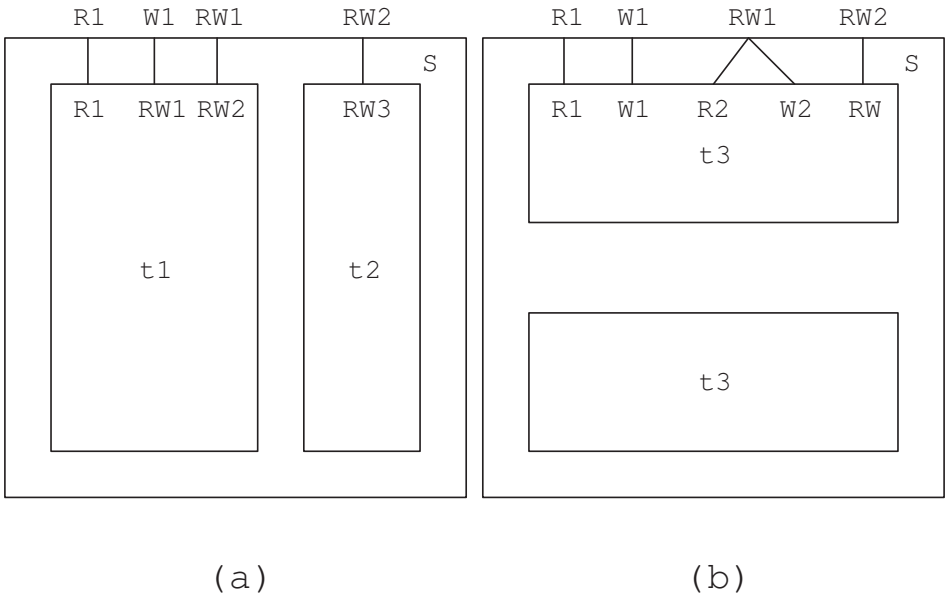

Fig. 4. Mapping result of the experiment

literature, as well as from industrial designs. Specifically, we report mapping results for four examples in Table 2. The first three columns in this table describe the source memory module. For each source module [10, 11, 12, 13], we list the name of the design from which the memory module was extracted, the source of the design and the size of the memory module. Columns 4 through 6 describe the mapping result. The fourth column lists the arrayed configuration of the target memory modules synthesized by our memory mapping algorithm; the fifth column displays the area cost function, i.e. the equivalent 2-input NAND gate counts of the implementation. The sixth column presents the run-time for our algorithm on SUN Sparc-5 Workstation. The seventh column lists the run-time in [9] and the eighth column lists the run-time improvement using our algorithm.

The table describes the memory mapping results generated by our mapping algorithm on a set of the source memory modules. The table shows the mapping results for area-efficient designs generated by our algorithm. These designs have been synthesized by using the memory modules taken from the library [14]. This library contains a set of RAM modules with the word-count equal to 16, 32, 64 or 128 and the bit-width equal to 2, 4 or 8. We report the area of the mapped designs in terms of equivalent 2-input NAND gates.

From the experiment, we can find that our approach for memory mapping is quite comprehensive. Our approach can synthesize source memory modules of varying complexity. These memories are of different word-count and bit-width. These

Example			Mapping Result				
#	Name	Size	Design	Cost (gates)	Run-time (s)	Run-time (s) [9]	Improvement
1	Differential Heat Release [10]	469×16	128×8, 128×8 128×8, 128×8 128×8, 128×8 64×8, 64×8 32×8, 32×8	68.4 K	1.4	1.8	22.2 %
2	Neural Network Chip [11]	160×8	128×8 32×8	12.1 K	0.59	0.9	34.4 %
3	Interface Scan [12]	360×16	128×8, 128×8 128×8, 128×8 64×8, 64×8 32×8, 32×8 16×8, 16×8	60.4 K	0.72	0.9	20.0 %
4	Image Processing [13]	384×12	128×4, 128×8 128×4, 128×8 128×4, 128×8	44.3 K	0.56	0.9	37.8 %

Table 2. Memory mapping result

memory modules come from designs of various complexity. Our memory mapping approach is quite general in the sense that it supports a wide variety of user selectable design parameters such as the source component, the target library and the optimizing cost function. If we analyze the run-time, we observe that we are able to generate these designs very quickly, in the order of a few seconds. Compared to the run-time in [9], we improved 20–40 % on this aspect.

6 CONCLUSION

In this paper, we presented a 0-1 integer linear programming technique to solve memory-mapping problems in high-level synthesis. The scheme implements a source memory module by using a set of target memory modules from a library. Our approach facilitates design reuse for memory subsystems. We identified and formulated the port-mapping subproblem into 0-1 ILP method. This method can not only perform bit-width mapping and word mapping, but it can also perform port mapping at the same time. Our experimental results that run on several industrial and literature-based examples demonstrate that our mapping algorithm can generate a variety of cost-effective memory designs based on the user-given cost function and the target library.

Acknowledgments

This work is supported by NSF, USA 5978 East Asia and Pacific Program (contract number: 9602485). We are grateful for their support. The authors would like to thank the anonymous referees for many helpful comments and suggestions in improving this paper.

REFERENCES

[1] Gajski, D.—Dutt, N.—Wu, A.—Lin, S.: High-Level Synthesis: Introduction to Chip and System Design. Kluwer Academic Publishers, 1992.

[2] Brayton, R.—Rudell, R.—Sangiovanni-Vincentelli, A.—Wang, A.: MIS: A Multiple-Level Logic Optimization system. IEEE Trans, On Computer-Aided Design, Vol. 6, 1987, pp. 1062–1081.

[3] Mailbot, F.—De Micheli, G.: Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations. IEEE Transactions On Computer-Aided Design, Vol. 12, 1993, pp. 599–620.

[4] Birmingham, W.—Gupta, A.—Siewiorek, D.: The MICON System for Computer Design. ACM/IEEE Design Automation Conference, 1989, pp. 135–139.

[5] Smith, J.—De Micheli, G.: Polynomial Circuit Models for Component Matching in High-Level Synthesis. IEEE Transactions, On Very Large Scale Integration (VLSI) Systems, Vol. 9, 2001, No. 6, pp. 783–800.

[6] Cong, J.—Yan, K.: Synthesis for FPGAs with Embedded Memory Blocks. Proceeding of International Symposium on Field Programmable Gate Arrays, pp. 75–81, ACM Press, 2000.

[7] Wilton, S.: Heterogeneous Technology Mapping for FPGAs with Dual-Port Embedded Memory Arrays. Proceeding of International Symposium on Field Programmable Gate Arrays, pp. 67–74, ACM Press, 2000.

[8] Ho, W.—Wilton, S.: Logical-to-Physical Memory Mapping for FPGAs with Dual-Port Embedded Arrays. Proceedings of International Workshop on Field-Programmable Logic and Applications, pp. 111–123, Springer, 1999.

[9] Jha, P.—Dutt, N: High-Level Library Mapping for Memories. ACM Transactions on Design Automation of Electronic Systems, pp. 566–603, July 2000.

[10] Ramashandran, L.—Chaiyakul: An Algorithm for Array Variable Clustering. European Design Automation Conference, Paris, France, pp. 262–266.

[11] Kachmer, D.—Rose J.: Definition and Solution of the Memory Packing Problem for Field-Programmable Systems. International Conference on Computer Aided Design, pp. 20–26.

[12] Lippens, P.: Memory Synthesis for High-Speed DSP Applications, Proceedings of the IEEE Custom Integrated Circuits Conference, 1991.

[13] Balasa, F.—Catthoor, F.—De Man, H.: Data-driven Memory Allocation for Multi-dimensional Signal Processing Systems. International Conference on Computer Aided Design, San Jose, CA 1994, pp. 31–34.

[14] Xilinx Development System Libraries Guide, San Jose, CA 1993.

Haifeng Zhou received his Master's degree from Dalian University of Technology in 1996. He is currently a Ph.D. candidate at the Department of Electronics Engineering of Shanghai Jiao Tong University. His main research interest is in high-level synthesis in electronic design automation.

Zhenghui Lin received his Ph.D. degree in electrical engineering from the University of Tokyo in 1981. He is currently a professor at the Department of Electronics Engineering in Shanghai Jiao Tong University. His research interests include theory of circuits and systems and electronic design automation.

Wei Cao received his Bachelor's, Master's and Ph.D. degrees in electronic engineering from Shanghai Jiao Tong University in 1995, 1998 and 2001, respectively. He is now with Aldec Corporation. His research interests include high-level synthesis and system design of VLSI systems.