Register Binding for FPGAs with Embedded Memory

Hassan Al Atat and Iyad Ouaiss
iyad.ouaiss@lau.edu.lb
Department of Computer Engineering
Lebanese American University
Byblos, Lebanon

Abstract*

The trend in new state-of-the-art FPGAs is to have large amounts of on-chip embedded memory blocks. These memory blocks are used to hold the input/output data for various applications. Existing register binding techniques in high-level synthesis aim at minimizing the storage requirements of circuits by sharing variables among registers and thus minimizing the required number of registers for a specific design. In this paper, a new technique is proposed that makes use of the existing embedded memory blocks and maps variables to these blocks. The proposed memory binding approach gives considerable performance increase over the existing register binding techniques. The memory binding technique resulted in up to 57% savings in the total chip area (number of Logic Cells/Elements occupied on the FPGA) over the old register binding techniques for a small resource bag and up to 6% savings for a large resource bag.

Introduction

In the earlier stages of FPGAs development, the trend was to have as many logic cells as possible to perform the specified design implementation. Each logic cell in the FPGA had some internal memory element in the form of a latch or a flip-flop used to store the result of the logic cell. Since this internal memory was not sufficient for large logic models where a data structure is needed to be stored in hardware, FPGAs were usually attached with external memory banks in order to hold the data structures of input, output, and temporary variables. Since the external communication with such banks is slow compared with internal FPGA communication, and since many of the designs targeted to FPGAs are data-oriented, the external FPGA communication becomes a major bottleneck in the design's performance. Hence, the need for FPGAs with embedded memory became apparent. Due to the scarcity of internal memory in old FPGAs, the techniques used in High-level Synthesis (HLS) aim at minimizing the storage requirements of the input designs so that the available memory is sufficient for holding the data structures of the design. Data path allocation, part of the High-Level Synthesis flow, deals with the allocation and binding of functional, storage, and interconnection units to operations, variables, and connections in a design respectively ([1], [2], [3] and [4]). The register binding and allocation techniques aim at allocating a minimal number of registers to hold the design's variables. Some of these techniques, such as Clique Partitioning [5] and the Left-Edge Algorithm [6], ignore the costs of register binding and the implied interconnection. Although other techniques, such as the extended Clique Partitioning [7] and the Weighted-Bipartite [8] take interconnection costs into consideration while performing register binding. In either case, the obtained results always tend to minimize the number of registers enduring interconnection costs. Since the trend in new FPGAs is to contain embedded memory banks within the FPGA (up to 10 Mbits of embedded memory in the Stratix [11] and Virtex II Pro [12] FPGAs), register binding and allocation is not necessarily confined to finding the minimal number of registers but aims at a more complex mapping of variables into registers as well as memory banks. This paper presents a new framework for register binding by first motivating the need for a new technique; second, by performing a literature survey on related work; third, by introducing the concept of embedded *memory binding*; fourth, by presenting the implementation of memory binding approach illustrated with an example; fifth, by discussing the results obtained followed by a conclusion.

Motivation

An extensive literature survey was conducted on the Altera and Xilinx families of FPGAs. The survey clearly reveals the new trend in FPGAs to have large amounts of embedded (on-chip) memory banks. A summary of the literature survey showing this trend for both Altera and

* This work is sponsored in part by the Research Council of the Lebanese American University.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

Xilinx devices is presented in Table 1 - Table 4. In the tables, the row "Block Size" specifies the size of each memory block. For instance, the Altera Stratix device has three different block sizes 512-bits, 4-Kbits and 512-Kbits, whereas the Altera Cyclone has memory blocks that are all 4-Kbits. The row "Max. Logic Cells" specifies the maximum number of Logic Elements in the Altera devices and the maximum number of Logic Cells in the Xilinx devices. The row "Max RAM Bits" specifies the maximum number of embedded memory bits. The row "Ratio" is the ratio of embedded memory bits to logic cells. Finally, the row "Max Distributed Memory" specifies the number of distributed memory bits in the FPGAs (as specified for the Xilinx devices only), where the function generators in the logic cells are used as storage devices.

Table 1: Altera Devices

Features v/s Family	Stratix [11]	Cyclone [15]	Apex II [14]	Apex 20K [20]
Block Size	512, 4K, 512K	4K	4K	2K
Max. Logic Cells	114,110	20,060	67,200	16,640
Max. RAM Bits	10,118,016	294,912	1,146,880	212,992
Ratio	88.67	14.70	17.07	12.80

Table 2: Altera Devices (continued)

Features v/s Family	Apex 20KE [20]	Apex 20KC [21]	Mercury [13]
Block Size	2K	2K	4K
Max. Logic Cells	51,840	38,400	14,400
Max. RAM Bits	442,368	327,680	114,688
Ratio	8.53	8.53	7.96

Table 3: Xilinx Devices

Features v/s Family	Virtex II Pro [12]	Virtex II [18]
Block Size	18K	18K
Max. Logic Cells	125,136	104,832
Max. RAM Bits	10,248,192	3,096,576
Ratio	81.90	29.54
Max. Distributed Memory	1,779,712	1,490,944

Table 4: Xilinx Devices (continued)

Features v/s Family	Virtex E Extended Memory [17]	Virtex E [16]	Virtex [19]
Block Size	4K	4K	4K
Max. Logic Cells	21,168	73,008	27,648
Max. RAM Bits	1,146,880	851,968	131,072
Ratio	54.18	11.67	4.74
Max. Distributed Memory	301,056	1,038,836	393,216

As shown in Table 1 - Table 4, the new trend is to have large amounts of embedded memories in FPGAs. Although new FPGAs have large amounts of logic cells, the ratio of embedded memory bits to logic cells has tremendously increased from about 5% to 82% for the Xilinx family of devices and from about 8% to 88% for the Altera family of devices.

Related Work

Several studies have been developed for using memory blocks in High-Level Synthesis for ASICs. Stock [25] proposed a method for HLS, where variables are grouped into memory models before binding them into registers; the method uses single-ported memory blocks. After grouping variables to registers, Simulated Annealing is used for minimizing interconnections. Tseng and Siewiorek [3] proposed a technique for using memories; the technique binds variables into registers then groups registers into memory pads. However, the technique does not take interconnection costs into account. Another approach that uses multi-port memories is presented in [26], the technique maps all variables first into one big memory model, and then it partitions that memory into several small modules. This technique takes interconnection costs into account before performing binding. In [27], a technique is presented that uses multi-port memories for register allocation; the technique uses 0-1 Integer Linear Programming in order to minimize interconnection costs. The technique in [28] extends that of [27] to minimize the total number of multi-port memories. The above mentioned techniques aim at using memories in datapath allocation, and minimizing both the number of used memory blocks and the interconnections of the designs. However, for FPGAs with

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

embedded memory blocks the problem is different: the number of memory blocks on the FPGAs is fixed and minimizing their number used is not necessarily a goal; on the contrary, an approach to minimize these blocks could lead to increasing interconnection costs.

In compilers research ([29] and [30]) and code generation for embedded systems, an approach known as register spilling is used. When the number of registers found in hardware is smaller than the number of variables alive, the variables are stored temporarily in memory and then loaded into empty registers when they are needed. Thus, in order to access the variables stored in memory, an expensive memory-to-register move is first performed. However, in memory binding, variables bound to memories can be accessed directly without the need to load them in registers. As will be seen in the following section, the concepts of register spilling however can still be applied to memory binding in selecting which variables map onto flip-flops and which variables map onto memory banks when examining critical paths of the design.

Memory Binding

Since all new FPGAs contain a large amount of embedded memory, this paper presents a new High-Level Synthesis technique that takes advantage of these memories. In FPGAs, registers are traditionally implemented using logic resources, for example by using the flip-flops of the Logic Elements in the Altera devices or of the Logic Cells of the Xilinx devices. Although flip-flops are present along with function generators within logic cells, the utilization ratio of the function generators decreases when the flip-flops are used due to placement and routing congestion. Thus, in traditional synthesis systems, even if the FPGAs contain embedded memory banks, registers in the design will be implemented using the scarce storage devices in the logic cells; hence decreasing the overall area utilization of the FPGA.

More specifically, in the register binding techniques, variables are bound to registers if their lifetimes (i.e. the time interval from which the variable is first written to the time it is last used) do not overlap. Some variables have long lifetimes although they span many control steps without being used and thus they conflict with other variables and must be bound to separate registers. Having a large number of registers in a design does not only induce extra logic consumption due to the registers, but also increases the interconnections and may lead to a consumption increase of logic resources required by extra multiplexers introduced in the design. Thus, the number of multiplexers separating the registers from the functional units increases as the number of registers increases. In other words, if the number of variable holders is decreased (registers or memories), the connections in the circuit and the number of synthesized multiplexers decrease too.

The developed technique in this section is referred to as *Memory Binding*. This technique assigns variables of a design to memory banks if the variables in the same memory bank do not conflict with each other. Variables conflict with each other if they are read in the same time step or if they are written in the same time step. If the memory bank has *N* write and *M* read ports, this condition changes (*N* and *M* being greater than one). In the developed technique, all memory banks are assumed to have exactly one write port and one read port. This assumption simplifies the solution, but is justified by the fact that almost all FPGAs' memory banks are dual read/write ports and by the fact that a memory bank with *N* read and *N* write ports is similar to N memory banks with exactly 1 read and 1 write port each. The memory binding technique has two main advantages:

- First, if the FPGAs contain embedded memories that are not completely used by data, variables can be bound to memories and not to registers. Therefore, the logic resources for implementing registers are saved.
- Second, conflicts between variables are no longer related to their lifetimes but solely to the times when the variables are read or written. Thus, memories will hold more variables and the interconnection cost as well as the number of multiplexers decrease.

The main disadvantage of the memory binding technique is the complexity introduced in the design in terms of control generation and the resulting delays. Knowing that High-Level Synthesis (HLS) takes as input an unscheduled Control-Data Flow Graph (CDFG) and the output is a synthesized data path with a Finite State Machine (FSM) controller, each state in the controller corresponds to one control step of the design, and the output of the controller is a sequence of signals that drives the datapath. The main disadvantage for using memory binding is the extra complexity needed for addressing the

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

memory banks; these complexities are added to the controller of the obtained design. Thus, for every port in the memory banks, an address and a read/write signal are added to the output signals of each state of the FSM. The generation of these extra signals will consume extra logic resources of the FPGAs. The size of extra logic resources consumed depends on three factors: the number of control steps (scheduled time steps), the number of memories used, and the number of variables bound to each memory[†]. The number of control steps determines the number of inputs to each logic cell, the number of memories determines how many addresses must be generated (one address per port), and the number of variables bound to each memory determines the size of each address and thus the number of logic cells required to produce every address[‡].

The memory binding technique induces three types of delay. The first type of delay is due to variables being bound to memories rather than registers and thus the variables will be in memories that are within a greater distance from the registers to their operators. This will introduce more congestion and more delay to the circuit. However, this fact is only true for older FPGAs where the memory banks are located at the boundaries of the FPGA or where there are no embedded memory blocks on the FPGA and there is a need for exterior memory banks. However, newer FPGAs have on-chip memory banks that are evenly distributed on the die. By decreasing the distance between the memory banks and the operators of the circuit, variables bound to the memory banks become closer to their sinking and sourcing operators.

The second type of delay is due to the usage of memories which have larger access delays when compared to registers. In this case, the technique can be altered by first, calculating the critical paths of the circuit before performing binding, and then binding some or all variables on the critical paths to registers and all other variables to memories. This alteration of the technique will only map variables onto memory banks when the delay of memory accesses does not affect the critical paths of the design, i.e. the overall design speed.

The third type of delay is due to the time needed to calculate the memory addresses of the variables. These address calculations are part of the controller operation and can happen in parallel to operations performed in the datapath.

Algorithm

The memory binding technique takes a scheduled CDFG as input and outputs a list that contains the list of variables bound to each memory bank. The technique works as follows: first, a list is obtained for every variable of the design; the list contains the name, written time and all read times of the variable. This list is obtained by going through the scheduled CDFG and finding for each variable the control steps in which it was an input for an operation.

```
While (U ≠ Ø) {
 V= next variable in U;

 If (V is an output of an operation Op) {
   Write_time of V= Scheduled_time of Op;
 }// end if
 Else {
  Write_time of V= 0;
 }// end else

 For (all scheduled operations in SCDFG) {
  O = next operation in SCDFG;
  If (V is an input to O) {
   Add Scheduled_time of O to Read_times of V;
  }//end if
 }// end for loop

Add V to VarList;
Add Write_time of V to VarList;
Add Read_times of V to VarList;

Remove V from U;
}// end while loop
```

Figure 1: Algorithm for Finding Variable List

The algorithm for finding the variables' list is shown in Figure 1. Note that U is the set containing all variables' names; SCDFG is the scheduled CDFG that contains information about all operations such as the name of inputs and the scheduled time for each operation; VarList is a list of all variables, containing the name, write time, and all read times of each variable.

[†] The number of memories used and the variables bound to each memory are inversely proportional. I.e. as the number of used memories increases, fewer variables are bound to each memory.

[‡] Note that although the depth of the memory is known and thus the size of its address is also known, the controller generates a smaller address to only match the number of variables. For example, if the depth of a memory is 1K and only six variables are bound onto the memory, the controller generates an address of three bits only while the other seven bits of the address are hardwired.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

```
While (VarList ≠ Ø) {
List BeginList, EndList, Mem;

Mem is an empty list;
BeginList is an empty list;
EndList is an empty list;

For (all variables in VarList) {
 V= next variable of VarList;
 If(Write_time of V not found in BeginList){
  If(all Read_times of V not found in EndList){
     Add Write_time of V to BeginList;
     Add all Read_times of V to EndList;
     Add V to Mem;
     Remove V from VarList;
  }//end if
 }//end if

 Add Mem to list of all Memories;
 }//end for loop

}// end while loop
```

Figure 2: Memory Binding Algorithm.

The memory binding algorithm is shown in Figure 2. The algorithm starts by looping until all variables are assigned to memories. In each iteration an unassigned variable is taken and assigned to a new memory bank; then two lists are created for this memory bank, beginList and endList; the beginList contains all the writing times of all the variables assigned to the memory bank, and the endList contains all the reading times of all the variables assigned to the memory bank. Any unassigned variable whose written time is not found in the beginList and whose all read times are not found in the endList is assigned to the memory bank and its written time is added to the beginList and all its read times are added to the endList. Also, in order to determine the number of multiplexers between the functional units and the memories and the number of multiplexers before the functional units, functional unit binding must be known.

Consider the scheduled CDFG example shown in Figure 3; the dashed lines refer to the scheduled time steps. The CDFG of Figure 3 is given as an input to the memory binding technique. The result of memory binding is shown in Figure 4. For comparison, if traditional register binding techniques using Clique Partitioning or Left-Edge Algorithm are performed, the outputs for the same design are shown in Figure 5 and Figure 6 respectively.

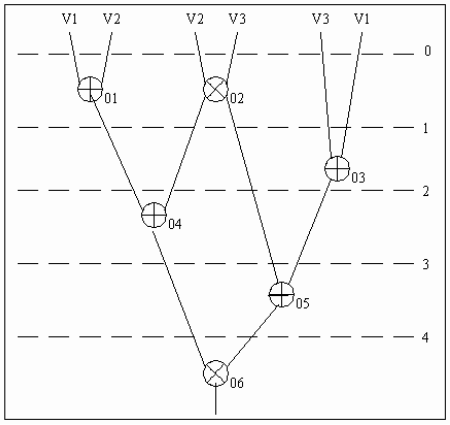

Figure 3: Scheduled CDFG Example

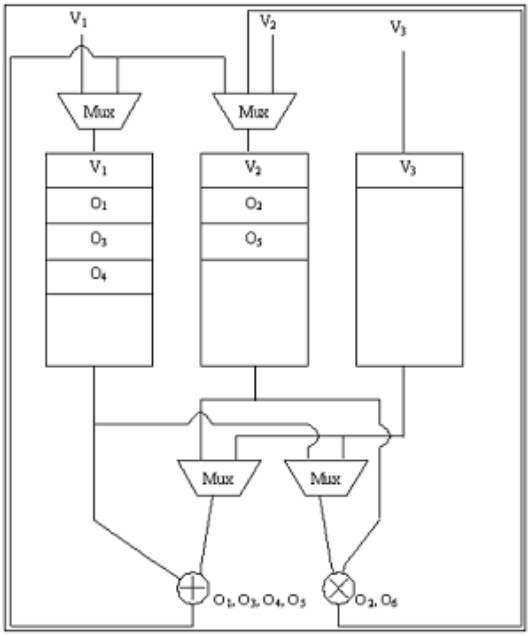

Figure 4: Memory Binding Example

Figure 4, Figure 5, and Figure 6 show that the complexity of the design is decreased when memory binding is used. The decrease in complexity is due to the decreased number of multiplexers, which in turn is due to the fact that the technique binds a larger number of variables to the same memory when compared to Clique Partitioning and Left-Edge Algorithm. So, for the above example, the memory binding approach is better than the other two techniques. The small resource bag makes variables span several control steps and thus their lifetime will overlap and they cannot be bound to the same register, but could be bound to the same memory bank.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

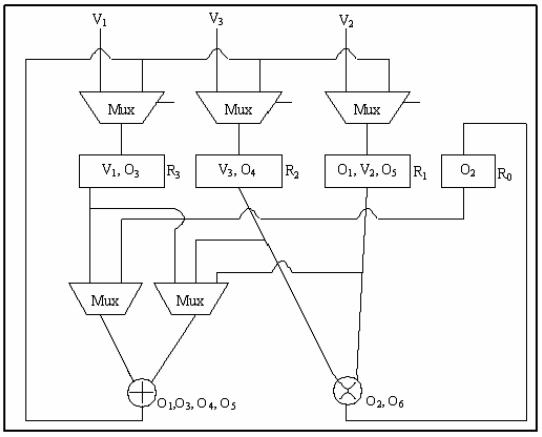

Figure 5: Clique Partitioning Example

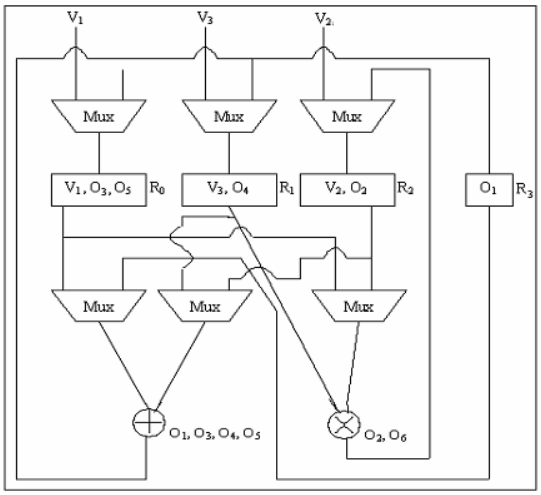

Figure 6: Left-Edge Algorithm Example

Results

To test the performance of the memory binding technique three benchmarks were used and results for the memory binding technique were compared with those for the Clique Partitioning and the Left-Edge Algorithm. The used benchmarks are: Discrete Cosine Transform (DCT), Differential Equations (DiffEq2 and DiffEq3). The DCT benchmark is a 35 node CDFG, it was used in [24]; DiffEq2 was first used in [6] and is an application of Euler's Method for solving second order differential equations [23]. DiffEq3 is an application of Euler's Method for solving third order differential equations [23]. All the benchmarks were scheduled using different resource bags in order to have several CDFGs on which to execute the register and memory binding techniques.

The results of the three techniques are compared according to the area occupied by each synthesized output. The area is in terms of logic cells: the number of logic cells occupied by each component in the circuit is shown in Table 5. These numbers where obtained using Max Plus II software Version 10.2 [22]. The components of the used benchmarks were input, compiled, and simulated for the Altera ACEX1K device family using the EP1K100FI484-2 FPGA.

Table 5: Number of Logic Cells per Component

Component Name	# of Logic Cells
MUX 2 to 1_32bit	32
MUX 3 to 1_32bit	64
MUX 4 to 1_32bit	96
MUX 5 to 1_32bit	132
MUX 6 to 1_32bit	165
MUX 7 to 1_32bit	198
MUX 8 to 1_32bit	231
MUX 9 to 1_32bit	264
MUX 10 to 1_32bit	297
REGISTER 32-bit	32
SUBTRACTOR 16-bit	31
ADDER 16-bit	31
LPM_MULT 16-bit	730

The multipliers, adders and subtractors are assumed to be 16-bit operations, and the multiplexers and registers are assumed to be 32-bits long.

After running the memory binding technique over the benchmarks, the clique partitioning and left edge algorithm were also implemented and executed on the benchmarks. The results of all three techniques were compared for all the benchmarks. Only the results for the Discrete Cosine Transform benchmark are reported in this paper, however they concur closely to the results obtained for the other benchmarks.

The memory binding technique reduces the area of the chip by two main factors: registers and multiplexers. The technique eliminates the area occupied by registers, since all variables are bound to memories. In addition, the number of multiplexers is reduced since memory binding assigns more variables to the same memory; thus reducing the number of memories used.

In order to illustrate the difference between memory binding and the other register binding techniques, Table 6 and Table 7 show a comparison between the results for Memory Binding, Left-Edge Algorithm, and Clique

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

partitioning respectively. In each table, the columns under the name Resource Bag shows the number of each functional unit in the used resource bag. Note that other types of functional units, such as subtractors, were needed in the designs; however, a single instance of each was used. The column named "MUX" shows the number of logic cells occupied by the multiplexers. This number denotes the number of logic cells occupied by all multiplexers; i.e. the multiplexers between the functional units and the memories, and the multiplexers before the functional units. Finally, the column named "Total" denotes the total number of logic cells occupied by the synthesized circuit in each case.

Table 6: Left-Edge and Memory-Binding Comparison for the DCT Benchmark

Resource Bag		Left Edge Algorithm Area		Memory Binding Area	
+	x	MUX	Total	MUX	Total
1	1	1876	3245	647	1408
1	16	1815	14166	1262	12973
2	1	1876	3276	967	1759
2	16	1846	14228	1587	13329
3	1	1876	3307	967	1790
3	16	1836	14249	1774	13547

Table 7: Clique Partitioning and Memory-Binding Comparison for the DCT Benchmark

Resource Bag		Clique Partitioning Area		Memory Binding Area	
+	X	MUX	Total	MUX	Total
1	1	1322	2691	647	1408
1	16	1526	13973	1262	12973
2	1	1322	2722	967	1759
2	16	1715	14161	1587	13329
3	1	1322	2753	967	1790
3	16	1837	14282	1774	13547

Table 8 and Table 9 show the gain of the memory binding technique over Clique Partitioning and the Left-Edge Algorithm respectively in terms of logic cells. The column named "MUX" is the gain of multiplexers' area; the column "% MUXes" is the percentage gain in multiplexers area; the column named "Total" is the total gain in logic cells used for multiplexers and registers; and the column named "% Total" is the percentage of total gain (the total logic cells obtained by the memory binding minus the total number of logic cells obtained by the other technique divided by the total number of logic cells obtained by the other technique).

Table 8: Gain over Clique for the DCT Benchmark

Resource Bag		Gain over Clique			
+	x	MUX	% MUXes	Total	% Total
1	1	675	51.06	1283	47.68
1	16	264	17.30	1000	7.16
2	1	355	26.85	963	35.38
2	16	128	7.46	832	5.88
3	1	355	26.85	963	34.98
3	16	63	3.43	735	5.15

Table 9: Gain over Left-Edge Algorithm for the DCT Benchmark

Resource Bag		Gain Over Left-Edge Algorithm			
+	x	MUX	% MUXes	Total	% Total
1	1	1229	65.51	1837	56.61
1	16	553	30.47	1193	8.42
2	1	909	48.45	1517	46.31
2	16	259	14.03	899	6.32
3	1	909	48.45	1517	45.87
3	16	62	3.38	702	4.93

As it is seen from the tables, the memory binding technique gives better results compared to the Clique Partitioning and Left-Edge Algorithm techniques, especially for smaller resource bags where the savings in number of multiplexers is larger. Since the memory binding technique assigns variables, which are not written and read in the same control step, to the same memory, while the register binding techniques assign variables to the same register if the lifetimes of these variables do not coincide. Thus, for a small resource bag, most of the variables will scan many control steps although they may be read once and written once, hence they will be bound to different registers while they could share the same memory.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

Conclusion

Since existing register binding techniques do not take embedded memories into account, the *Memory Binding* technique was developed in this paper. This new technique maps variables to memories rather than to registers; the mapping is done on the basis that variables mapped to the same memory do not coincide with each other. The technique eliminates all registers from the design, which reduces the placement and routing congestion in and around logic cells. In addition, the technique helps to reduce the number of multiplexers in the design since memories hold larger amounts of variables than registers; thus minimizing the connections in the design. The memory binding technique resulted in up to 65% savings in multiplexers' area (number of Logic Cells/Elements occupied on the FPGA) and up to 57% in the total chip area (number of Logic Cells/Elements occupied on the FPGA) over the old register binding techniques for a small resource bag. However, for larger resource bags the memory binding technique resulted in up to 7% savings in multiplexers' areas and up to 6% savings in the total chip area over the old register binding techniques. Since implemented designs are usually constrained by FPGA area, resource bags typically tend to be small. With small resource bags, the new memory binding technique provides substantial area savings.

The analysis of the critical paths in the designs is currently being investigated and an adapted memory binding algorithm is being developed that analyzes the critical paths with respect to the delay introduced by mapping the variables to memory banks. The goal of this adapted technique is to optimize the assignment of variables onto registers and/or memory banks while minimizing the overall delay of the design and the overall area utilization of the FPGA.

Furthermore, the memory binding technique assumes that all memory banks have exactly one write port and one read port. This simplification of the problem is not a good assumption if the memory ports are dual read/write ports. In this case, the problem is harder and the technique must be further developed to cater to the flexibility of the ports.

References

[1] D. Gajski, N. Dutt, A. Wu, and S. Lin, "*High-Level Synthesis*," Kluwer Academic Publishers, Boston, 1992.

[2] G. De Micheli, "*Synthesis and Optimization of Digital Circuits*," McGraw-Hill, NewYork, 1994.

[3] Y-L. Lin, "*Survey Paper: Recent developments in high-level synthesis,*" ACM Transactions on Design Automation of Electronic Systems, Vol. 2, No. 1, January, 1997, pp 2 - 21.

[4] M. C. McFarland, A. C. Parker, and R. Camposano, "*Tutorial on high-level synthesis,*" Proceedings of the 25th Design Automation Conference, 1988, pp. 330 - 336.

[5] C.J. Tseng and D.P. Siewiorek, "*Automated Synthesis of Data Paths on Digital Systems*," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. CAD-5, no. 3, pp. 379-395, July 1986.

[6] F.J. Kurdahi and A.C. Parker, "*REAL: A Program for Register Allocation*," Proceedings of the 24th Design Automation Conference, pp. 210-215, 1987.

[7] C-Y. Huang, Y-S. Chen, Y-L. Lin and Y-C. Hsu, "*Data Path Allocation Based on Bipartite Weighted Matching,*" Proceedings of the 27th Design Automation Conference, pp. 499 - 504, 1990.

[8] P.G. Paulin and J.P. Knight, "*Force-Directed Scheduling for the behavioral Synthesis of ASICs*," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 8, no. 6, pp. 661-679, June 1989.

[9] Hashimoto and J. Stevens, "*Wire Routing by Optimizing Channel Assignment within Large Apertures*," The 8th Design Automation Conference, pp. 155-169, 1971.

[10] P. G. Paulin and J. P. Knight, "*Scheduling and binding algorithms for high-level synthesis*," Proceedings of the 26th Design Automation Conference, pp. 1 – 6, 1989.

[11] Altera Corporation, "*Stratix Programmable Logic Device Family Data Sheet*," Version 2.1, August 2002.

[12] Xilinx Corporation, "*Virtex II-Pro Platform FPGAs: Introduction and Overview*," Version 2.2, September 2002.

[13] Altera Corporation, "*Mercury Programmable Logic Device Family Data Sheet*," Version 2.1, March 2002.

[14] Altera Corporation, "*APEX II Programmable Logic Device Family Data Sheet*," Version 3.0, August 2002.

[15] Altera Corporation, "*Cyclone FPGA Family Data Sheet*," Version 1.0, September 2002.

[16] Xilinx Corporation, "*Virtex E 1.8V Field-programmable Gate Arrays*," Version 2.3, July 2002.

[17] Xilinx Corporation, "*Virtex E 1.8V Extended Memory Field-programmable Gate Arrays*," Version 1.5, July 2002.

[18] Xilinx Corporation, "*Virtex II 1.5V Field-Programmable Gate Arrays*," Version 1.9, September 2002.

[19] Xilinx Corporation, "*Virtex 2.5 VField Programmable Gate Arrays*," Version 2.5, April 2, 2002.

[20] Altera Corporation, "*APEX 20K Programmable Logic Device Family Data Sheet*," Version 4.3, February 2002.

[21] Altera Corporation, "*APEX 20KC Programmable Logic Device Family Data Sheet*," Version 2.1, April 2002.

[22] Altera Corporation*,"http://www.altera.com/products/software/pld/products/max/sofemax_baseline.html"*.

[23] E. Kreyszig, "*Advanced Engineering Mathematics*," (8th ed.), John Wiley & Sons, Inc., New York, 1999.

[24] S. Govindarajan, I. Ouaiss, et al., "*An Effective Design System for Dynamically Reconfigurable Architectures*," Proceedings of Sixth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 312-313, IEEE Computer Society, April 1998.

[25] L. Stok, "*Interconnect Optimisation During Data Path Allocation*." In Proceedings of the European Design Automation Conference (Edinburgh, Scotland), pp. 141 - 145, March 1990.

[26] T. Kim and C. L. Liu. "*Utilization of Multiport Memories in Data Path Synthesis*". Proceedings of the 30th Design Automation Conference, pages 298–302. ACM Press, June 1993.

[27] M. Balakrishnan, et al. "*Allocation of Multiport Memories in Data Path Synthesis*". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, volume 7, pages 536–540, April 1988.

[28] I. Ahmad and C. Y. Chen. "*Post-Process for Data Path Synthesis*". Proceedings of International conference on Computer Aided Design, pages 276–279. ACM Press, 1991.

[29] P. Briggs, *"Register coloring via graph coloring"*. Rice University, Ph.D. Thesis, April 1992

[30] G. Chaitin, M. Auslander, A. Chandra, J. Coockie, M. Hopkins, and P. Markstein, "*Register Allocation via coloring."* Comput. Lang. 6, pp.47 - 57, Jan. 1981.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE