

Partitioning of Register Level Designs for Multi-FPGA Synthesis.*

Madhavi Vootukuru, Ranga Vemuri and Nand Kumar †

Laboratory for Digital Design Environments

Department of Electrical and Computer Engineering and Computer Science

University Of Cincinnati, ML 30

Cincinnati, OH, 45221-0030.

mvootuku@ece.uc.edu, ranga.vemuri@ece.uc.edu

Ph : 513-556-4784

Abstract

In this paper we address the problem of partitioning register level designs for implementation on multiple FPGAs. The partitioner uses a modified multi-way Fiduccia-Mattheyses algorithm. Cost estimation functions needed by the partitioner to estimate the resources needed by the design on a FPGA have been developed. The methodology for estimation of resources on an FPGA (function generators, flip-flops and CLBs), and partitioning of the design are discussed in detail.

1 Introduction

A design that has to be implemented on a Field Programmable Gate Array (FPGA) needs certain resources on the FPGA device. These resources include configurable logic blocks (CLBs), each having a fixed number of function generators and flip-flops and user I/O pins. If the design which has to be downloaded onto an FPGA needs more resources than available on one FPGA, there is a need to partition the design into multiple segments such that each of the partition segments satisfies resource constraints on the devices available. To achieve this goal, we formulate the multi-FPGA partitioning problem as follows:

Given a register level design represented as a net-list of components and constraints in terms of maximum number of available CLBs, function generators, flip

*This work is being done at the University of Cincinnati and is supported in part by the ARPA RASSP program and is monitored by Wright Patterson Air Force base under contract no. F33615-93-C-1316 and by the Solid State Electronics Directorate of the Wright Laboratory of the US Air Force under contract no. F33615-91-C-1811.

†Nand Kumar is currently Vice-President of synthesis, at Triquest DA Inc.

flops and allowable user I/O pins on each chip, partition the design into a set of interconnected design segments, each of which satisfies the constraints. The system creates one or more bit map files depending on the specified constraints. The input to the system, is a register level design, instantiating components from a known parameterized component library. This library has various components such as adders, subtractors, multipliers, dividers, latches, multiplexers etc. These components are discussed in detail in later sections of the paper.

2 Synthesis and partitioning of FPGA designs

We use a high level synthesis system which takes behavioral descriptions as input and produces register transfer level descriptions of the same design. The high level synthesis system is called Distributed Synthesis System (DSS), developed at The University of Cincinnati [1], for producing RTL designs. The system produces a data path and a controller. The data path is represented as a net-list of register transfer level components and the controller is represented as a finite state machine.

The inputs to our system are the register level design (output of DSS) and the constraints are the FPGAs available, the library of RTL components, and the resource utilizations of all register level components for varying generic parameters. *Performance estimator* and *partitioner* form the central components of our system. Performance estimation involves accurate estimation of cost of the design and the partitioning involves the proper choice of design segments which satisfy user specified constraints. The cost here refers to the number of CLBs (packed CLBs), the number of function generators, and the number of flip-flops. Pin

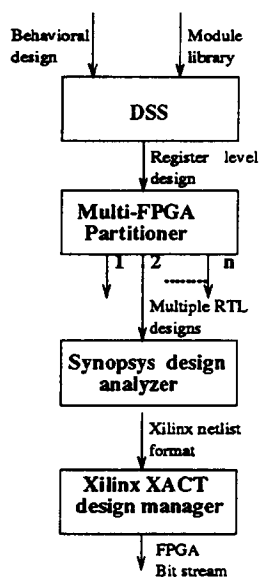


Figure 1: Multi-FPGA synthesis flow

constraints are also taken into account while fixing the partitions. If the design cannot be partitioned into the available number of chips, each with allowed number of I/O pins, the partitioner returns the best possible solution obtained during the specified iterations on the K-way FM algorithm.

The performance estimator works on the data path and the controller separately and gives estimations for the whole design using these individual estimates. Once the performance estimation is done, the partitioner is invoked. It uses a modified multi-way Fiduccia-Mattheyses algorithm [3] to produce partition segments which satisfy the constraints. These partitions are used as input to logic synthesis tools to generate bit map files. We chose Xilinx FPGAs as our target technology. The design flow for obtaining programmed bit map files for FPGAs is shown in Figure 1. We used Synopsys design analyzer for logic synthesis of partitioned RTL designs, and Xilinx XDM tools for generating layouts and producing bit-map files necessary to download the design onto an FPGA device.

3 RTL component library

The input register level design instantiates components from a RTL component library. These components use gate level components from Xilinx XC4000 family. The components are parameterized for bit-

Component	Xilinx Hard Macros used
Const_reg	Buf
Adder	Add1, Vcc, Inv
Subtractor	Add1, Gnd, Inv
Comparator	Nor2, And2b1, And2, Inv, And3b1, Xnor2
Latch	FDCE
Multiplexer	M2.1
Shift_reg	And2, Or2, Or3
Signal	And2, And3, And4, Inv, FDPE, Or2, Xnor2, FDCE
And	And2
Or	Or2
Nor	Nor2
Xor	Xor2
Xnor	Xnor2
Not	Inv

Table 1: RT level Components in component library instantiated in RT level code

width, number of inputs and number of select lines. Table 1 shows the components and the Xilinx hard macros used. The number of CLBs, function generators and flip-flops for different generic parameters of each of the components is made available to the system in the format shown in Table 2. This data was obtained experimentally by synthesizing each of the components with varying parametric values and generating the Logic Cell Array (LCA).

4 Performance Estimator

Estimation functions for estimating the number of function generators, flip-flops and CLBs needed for an input design have been developed. Estimation of resources needed by a design represented as a data path and a controller can be done by considering each of these entities separately.

Estimates for data path : We estimate the number of function generators and flip-flops needed by the data path and use this data in determining the number

S.No.	Module Name	(Bitwidth, Resource count)
1	Const_reg	FF : (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (16,0) FG : (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (16,0) CLB : (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (16,0)
2	Adder	FF : (1,0),(2,0),(4,0),(8,0),(16,0),(32,0),(64,0) FG : (1,1),(2,3),(4,6),(8,12),(12,18),(16,24),(20,30),(32,48),(64,64) CLB : (1,1),(2,1),(4,3),(8,6),(12,9),(16,12),(20,15),(32,24),(64,32)
3	Subtractor	FF : (1,0),(2,0),(4,0),(8,0),(16,0),(32,0),(64,0) FG : (1,1),(2,3),(4,6),(8,12),(12,18),(16,24),(32,48),(64,64) CLB : (1,1),(2,1),(4,3),(8,6),(12,9),(16,12),(20,15),(32,24),(64,32)
4	Comparator	FF : (1,0),(2,0),(4,0),(8,0),(16,0),(32,0) FG : (1,3),(2,8),(4,18),(8,37),(16,38),(24,60),(32,157) CLB : (1,1),(2,4),(4,9),(8,18),(16,19),(24,30),(32,78)
5	Shift_Reg	FF : (1,3),(2,4),(3,6),(4,8),(5,10),(6,12),(7,14),(8,16),(16,32),(32,40) FG : (1,1),(2,4),(3,6),(4,8),(5,10),(6,12),(7,14),(8,16) (16,32),(32,40) CLB : (1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,7) (16,16),(32,20)
6	Signal	FF : (1,7),(2,12),(4,19),(8,35),(16,67),(32,80) FG : (1,3),(2,5),(4,7),(8,13),(16,25),(32,40) CLB : (1,1),(2,2),(4,3),(8,6),(16,12),(32,20)
7	And	FF : (1,0),(2,0),(4,0),(8,0),(16,0) (32,0),(64,0) FG : (1,1), (2,2), (4,4), (8,8), (16,16) (32,32), (64,64) CLB : (1,1), (2,1), (4,2), (8,4), (16,8) (32,16), (64,32)
8	Or	FF : (1,0),(2,0),(4,0),(8,0),(16,0) (32,0),(64,0) FG : (1,1), (2,2), (4,4), (8,8), (16,16) (32,32), (64,64) CLB : (1,1), (2,1), (4,2), (8,4), (16,8) (32,16), (64,32)
9	Nor	FF : (1,0),(2,0),(4,0),(8,0),(16,0) (32,0),(64,0) FG : (1,1), (2,2), (4,4), (8,8), (16,16),(32,32), (64,64) CLB : (1,1), (2,1), (4,2), (8,4), (16,8) (32,16), (64,32)
10	Xor	FF : (1,0),(2,0),(4,0),(8,0),(16,0),(32,0),(64,0) FG : (1,1), (2,2), (4,4), (8,8), (16,16),(32,32), (64,64) CLB : (1,1), (2,1), (4,2), (8,4), (16,8),(32,16), (64,32)
11	Xnor	FF : (1,0),(2,0),(4,0),(8,0),(16,0),(32,0),(64,0) FG : (1,1), (2,2), (4,4), (8,8), (16,16) (32,32), (64,64) CLB : (1,1), (2,1), (4,2), (8,4), (16,8),(32,16), (64,32)
12	Not	FF : (1,0),(2,0), (3,0),(4,0), (5,0),(6,0),(7,0),(8,0), (16,0),(32,0) FG : (1,0), (2,0), (3,0),(4,0), (5,0),(6,0),(7,0),(8,0), (16,0), (32,0) CLB : (1,0), (2,0), (3,0),(4,0), (5,0),(6,0),(7,0),(8,0), (16,0), (32,0)
13	Latch	FF : (1,1),(2,2),(4,4),(8,8),(16,16) FG : (1,0),(2,0),(4,0),(8,0),(16,0) CLB : (1,1),(2,1),(4,2),(8,4),(16,8)
14	Multiplexer	FF : (1,0),(2,0),(4,0),(8,0),(16,0) FG : (1,1),(2,2),(4,4),(8,8),(16,32) CLB : (1,1),(2,1),(4,2),(8,4),(16,16)

Table 2: Data provided in Component-data file (input to estimator)

of CLBs needed by the whole design when it is taken to layout. To estimate the number of function generators and flip flops, we sum up these values for all the instantiated components. The generic values used to instantiate various register transfer level components, and their respective resource utilizations are obtained through table-lookup from the system database (Table 2). Logic trimming done at gate level is taken into account by reading the input net-list, and determining the signals not being used. In other words, we determine the load-less signals, if any, in the design: The flip-flops FDPE and FDCE are used by Xilinx tool to store the bits in the clocked components. Once the load-less signals are determined, the corresponding number of flip-flops used to store these signals is subtracted from the number obtained by summing up the individual component flip-flop counts in the design. This gives the number of flip-flops necessary by the data-path. A similar procedure is followed for obtaining an estimate of function generators used by the data-path.

No. of flip-flops needed by data-path =

$$\sum_{r \in dp} FF_count(r) - \sum_{s \in L} UFF_count(r)$$

where, $FF_count(r)$ is the number of Flip-flops of individual register level components, $UFF_count(r)$ is the unused flipflop count of component whose output signal is 's' and L is the set of load-less signals in the data-path.

No. of function generators needed by the data-path =

$$\sum_{r \in dp} FG_count(r) - \sum_{s \in L} UFG_count(r)$$

where, $FG_count(r)$ is the number of Function generators of individual register level components in data-path, $UFG_count(r)$ is the number of unused function generators of component whose output signal is 's' and L is the set of load-less signals in the data-path.

Since each CLB in XC4000 family of FPGAs has 2 function generators and 2 flip-flops, the number of packed CLBs needed is determined to be half the number of flip-flops (function generators) for designs with dominating sequential (combinational) logic, that is, dominating number of flip-flops (function generators).

Estimates for controller : The necessary function generators and flip-flops in the controller part of the design can be estimated from the finite state machine (FSM) description. The number of states in the controller is the main factor which determines the amount

of logic required on the chip. The number of state variables depends on the number of states in the FSM and is given by $\log_2(\text{number of states})$. A register whose bit-width is equal to the number of state variables is needed to store the present state and next state variables.

The elaborated FSM is represented as a set of multiplexers and gates by the synthesis tool. The size of inputs and select signals to the multiplexers was found to be proportional to the control word length, and number of function generators was found to be proportional to the number of states in the FSM. We conducted a series of experiments and found that the number of function generators needed by the FSM is lesser if the control word in the FSM depends only on the current state (Moore machine) rather than on the current state and the control inputs (Mealy machine). The number of gates (and hence the number of function generators) needed by the FSM depends on the number of nested 'case' statements in the FSM description in VHDL. This implies that every time the input flags or input state bits are checked for assigning a value to the output of FSM, the number of gates/function generators increase. Using a large number of designs, the increase was found to be 3 function generators for each nested 'case' statement. Hence the factor 3 in the equation below. The length of the control word, which is the output of the FSM does not significantly affect the amount of logic necessary for the controller. On the other hand, number of states in the controller has a major influence on the resources needed on an FPGA. We found that there is almost an exponential increase in the number of function generators necessary with increasing states in a controller. This is due to the extra logic that is needed for assigning values to the signals for each state that is included in the finite state machine. The exponent S was determined to be 2.5. This was found by varying the number of states in the controller, keeping all the other factors constant and producing LCA of the FSM.

No. of flip-flops needed by controller =

$$\text{No. of state variables in the FSM.}$$

No. of function generators needed by the controller =

$$\begin{aligned} &\text{No. of state variables} ** S + \\ &\text{No. of bits in control word} * C + \\ &\text{No. of nested 'case' statements} * F \end{aligned}$$

where, S = 2.5, C = 0.2 and F = 3.

Since the number of function generators in any FSM

is much larger number than the number of state variables (number of flip-flops) in the FSM, the number of packed CLBs needed by the controller is found to be half the number of function generators.

Estimates for the complete design : Estimates of resources needed by data-path and controller can be used in determining the number of function generators, flip-flops and packed CLBs needed by the complete design.

The number of function generators in the complete design = $(FG_{datapath} + FG_{controller}) * G$, where G represents the Global optimization factor and FG represents number of function generators.

The global optimization factor is found to lie between 0.6 and 0.8 depending on the amounts of combinational and sequential logic involved in the design.

Number of flip-flops in the complete design = $FF_{datapath} + FF_{controller}$

where FF represents the number of flip-flops.

Packed CLB count for the complete design = $0.5 * \text{Max}(\text{Function generators, Flip-flops})$

5 Partitioning Algorithm For Producing Multiple FPGAs

Partitioning of a design involves determining the cost constraints such as the overall resource utilizations on an FPGA, and constructing the partitions subject to these constraints.

The partitioner takes the input net-list and produces multiple segments. Each segment 's' is subject to the following constraints:

1. **Resource Constraint:** The resources required by any segment of the design should not exceed the maximum allowed values set by the user or the particular FPGA part number. The constraints here refer to number of CLBs, function generators and flip-flops, Let these constraints be denoted by CLB_s , FG_s , and FF_s , respectively for each segment 's' in the design.
2. **Pin Constraint:** Number of pins on any segment should not exceed the allowed number of user I/O pins P_s on each chip. This is checked for all the partitions of the design.

3. **Overall design constraint:** Number of segments after partitioning should not exceed the allowed number of FPGA devices.

We use the modified multi-way Fiduccia-Mattheyses algorithm [2] for partitioning an input design into multiple designs. The multi-way FM partitioning algorithm used is shown in Figure 2. The partitioner begins by invoking *initialize()* for initializing the values of total number of CLBs, function generators and flip-flops for the whole design, which are obtained as output of the cost estimation functions. It then takes the user specified constraints and determines the number of FPGAs needed by the design. This is calculated as,

Number of FPGAs = No. of packed CLBs in the complete design / CLB_s ,

Once the number of chips is found, a random initial partition of N partition segments is created by the K-Way FM algorithm, where N is the number of chips. As a result, the graph G of V vertices is partitioned into N segments, each with a fixed number of nodes. The initial partition is saved as *Best* partition. The pins on all partitions are calculated by *compute_pins_on_all_partitions()* and the value saved as *best_pins*. K-Way partitioning is carried out by repeatedly invoking the standard FM bi-partitioning algorithm [3] on pairs of partition segments. *two_way_fm()* tries to improve bi-partitions by moving one node at a time from one partition to the other. Each time a move is made, *check_chip_constraint(S)* is invoked to ensure that each partition segment satisfies the constraints. This function checks if the partition segment S satisfies the constraints such as CLBs, function generators, flip-flops and pins available on the chip and returns the *status* to K-Way FM. The status is true if the constraints are met by the partition segment and false otherwise.

The K-WAY FM algorithm is invoked repeatedly until (1) either a solution that satisfies the specified constraints is found, or (2) a user specified limit on number of iterations (MAX_ITER_CNT) is exceeded, or (3) a user specified limit on number of iterations over which the resultant partitions (MAX_IMP) do not show any improvement in cost is exceeded. The partitioner returns either a set of partition segments that satisfy the constraints or the best possible solution (if the constraints could not be met by all the partitions).

```

KWAY_(G)
begin
  Best ← initialize()
  N ← calculate_num_chips()
  Compute_pins_on_all_partitions()
  best_pins ← Pins(Best)
  S ← null
  continue_part ← TRUE
  iteration_cnt ← 1
  improve_cnt ← 1
  while continue_part = TRUE do
    for i = 1 to n-1 do
      for j = i+1 to n do
        two_way_fm( $s_i, s_j$ )
      end for
    end for
    curr_pins ← Pins(S)
    iteration_cnt ← iteration_cnt + 1
    status ← check_chip_constraint(S)
    if curr_pins < best_pins  $\vee$  status = TRUE then
      improve_cnt ← 1
      Best ← S
    else
      improve_cnt ← improve_cnt + 1
    end if
    if iteration_cnt = MAX_ITER_CNT  $\vee$ 
      improve_cnt = MAX_IMP
      then cont_part ← FALSE
    end if
  end while
  return(Best) /* retrieve best partition */
end

Check_chip_constraint(S)
begin
  status ← TRUE
  for all  $s_i \in S$  do /* segments in partition */
    if  $CLBs(s_i) > CLB_s \vee pins(s_i) > P_s \vee$ 
       $FG(s_i) > FG_s \vee FF(s_i) > FF_s$ 
      then status ← FALSE
    end if
  end for
  return(status)
end

```

Figure 2: Algorithm for kway partitioning

6 Implementation and results

We have developed a vertically integrated system for a top-down design flow for FPGA synthesis with DSS as the front end and multi-way Fiduccia-Mattheyses algorithm being used for producing partitions of the input design. We used Xilinx XC4000 as our target FPGA family and used Synopsys design analyzer and Xilinx XACT design manager tools, to produce bit map files for FPGA implementation. Tables 3, 4 and 5 show actual and estimated resources needed by the data-path, controller and the complete design respectively with $G = 0.8$ in most designs. Constraints and corresponding partitions obtained from the partitioner for a number of designs are shown in Table 6.

7 Future work

We are working on extending the present partitioning engine to partition an input design into multiple FPGA devices, each satisfying a speed constraint in addition to resource constraints. We are also working on extending this approach to support heterogenous FPGA targeting.

Acknowledgments

We thank Narendra Narasimhan, Vinoo Srinivasan and Sriram Govindarajan, University of Cincinnati for their help and advice. In particular, we thank Vinoo Srinivasan and Sriram Govindarajan for their contribution in the development of the RTL library targeted for FPGAs.

References

- [1] Jay Roy, Nand Kumar, Rajiv Dutta and Ranga Vemuri, "DSS: A Distributed High-Level Synthesis System", IEEE D&T Of Computers, vol. 9, No. 2, June 1992.
- [2] Nand Kumar, "High Level VLSI Synthesis For Multichip Designs", PhD Dissertation, University of Cincinnati. October 1994.
- [3] C.M.Fiduccia and R.M.Mattheyses, "A Linear-Time Heuristic for Improving Network partitions," Proc. 19th Design Automation Conference, pp. 175-181, June 1982.
- [4] Jay Roy, Rajiv Dutta and Ranga Vemuri, "Distributed Design Space exploration For High Level Synthesis Systems", 29 th ACM/IEEE Design Automation Conference, June 1992.

Design	No. of RT level components in the datapath	Function generators		Flipflops	
		actual	estimate	actual	estimate
TLC	33	40	44	48	48
SS-prod	34	400	407	367	367
DCT	23	148	155	204	204

Table 3: Estimated and actual values for data-path

Design	No. of states in the controller	Control word length (output)	Function generators		Flipflops	
			actual	estimate	actual	estimate
TLC	34	40	140	161	6	6
SS-prod	37	40	128	143	6	6
DCT	38	30	136	129	6	6

Table 4: Estimated and actual values for controller

Design	Function generators		Flipflops		Packed CLBs	
	Actual	Estimate	Actual	Estimate	Actual	Estimate
TLC	161	164	54	54	80	82
SS-prod	422	440	373	373	211	220
DCT	238	227	210	210	119	119

Table 5: Estimated and actual values for complete design

Design	Packed CLBs, FGs, FFs	FPGA available	Constraints			Partitions		
			CLBs	FGs	FFs	Chip1	Chip2	Chip3
TLC	80, 161, 54	XC4002	64	128	256	CLBs=67 FGs=133 FFs=42	CLBs=14 FGs=28 FFs=12	- - -
TLC	80, 161, 54	XC4003	100	200	360	CLBs=80 FGs=161 FFs=54	- - -	- - -
SS-Prod	211, 422, 373	XC4003	100	200	360	CLBs=100 FGs=190 FFs=172	CLBs=98 FGs=186 FFs=180	CLBs=13 FGs=46 FFs=21
SS-Prod	211, 422, 373	XC4005	196	392	616	CLBs=192 FGs=370 FFs=350	CLBs=19 FGs=22 FFs=23	- - -
DCT	119, 238, 210	XC4003	100	200	360	CLBs=81 FGs=168 FFs=192	CLBs=39 FGs=70 FFs=18	- - -
DCT	119, 238, 210	XC4005	196	392	616	CLBs=119 FGs=238 FFs=210	- - -	- - -

Table 6: Constraints and results of Partitioner

- [5] Daniel D. Gajski, Loganath, Ramachandran, "Introduction To High Level Synthesis", IEEE D&T Of Computers, 1994.
- [6] A. Sangiovani-Vincentelli, "Synthesis Methods For Field Programmable Gate Arrays", IEEE Proceedings, July 1993.
- [7] Nam-Sung-Woo, Jaesook Kim, "Efficient Method For Partitioning Circuits for Multiple FPGA Implementation", 30 th Design Automation Conference, June 1993.
- [8] R. Camposano, "From Behavior to Structure: High-Level Synthesis", IEEE Design & Test of Computers, pp. 8-19, Oct. 1990.
- [9] R. Camposano and W. Wolf, "High-Level VLSI Synthesis", Kluwer Academic Publishers, Boston, 1991.
- [10] XILINX XC4000 FPGA data Book, XILINX Inc., 1994.
- [11] XILINX Unified Libraries, XACT Libraries guide, April 1994.