Multiple-Wordlength Resource Binding

George A. Constantinides[1], Peter Y.K. Cheung[1], and Wayne Luk[2]

[1] Department of Electrical and Electronic Engineering,
Imperial College, London SW7 2BT, U.K.
{g.constantinides, p.cheung}@ic.ac.uk
[2] Department of Computing, Imperial College,
London SW7 2AZ, U.K.
wl@doc.ic.ac.uk

Abstract. This paper describes a novel resource binding technique for use in multiple-wordlength systems implemented in FPGAs. It is demonstrated that the multiple-wordlength binding problem is significantly different for addition and multiplication, and techniques to share resources between several operations are examined for FPGA architectures. A novel formulation of the resource binding problem is presented as an optimal colouring problem on a resource conflict graph, and several algorithms are developed to solve this problem. Results collected from many sequencing graphs illustrate the effectiveness of the heuristics developed in this paper, demonstrating significant area reductions over more traditional approaches.

1 Introduction

This paper describes a novel resource binding technique used by the Synoptix high-level synthesis system for FPGAs [1]. Synoptix synthesizes designs using multiple-wordlength resources, in an effort to minimize the total resource utilization required to implement a given Digital Signal Processing algorithm. The use of multiple-wordlength resources has an impact on the resource allocation and binding problem of high-level synthesis. This paper introduces algorithms for optimal and heuristic resource-binding for such multiple-wordlength systems. The nature of the binding problem is analyzed, and the suitability of current FPGA architectures is assessed.

Traditionally the wordlength problem for DSP applications has been to find a single uniform system wordlength, which satisfies both the conflicting requirements of design area/speed and acceptable rounding and truncation signal distortion. The idea of a single uniform wordlength is consistent with the DSP processor model of computation, where a single, or multiple, pre-designed fixed-wordlength computational units are responsible for all operations. When synthesizing a circuit for FPGA implementation, we are freed from such constraints. It is possible to use different wordlength functional units at different locations within the array, in order to minimize the overall logic utilization requirements. It is this approach that is taken by the Synoptix system, as described in [1].

R.W. Hartenstein and H. Grünbacher (Eds.) FPL 2000, LNCS 1896, pp. 646–655, 2000.

Of the several recent approaches to wordlength optimization [2,3,4], few have considered in detail the resource binding problem when wordlength information is known. One approach described in [2] is to modify a standard clique partitioning algorithm on the compatibility graph [5]. In the compatibility graph, nodes represent operations, and edges represent two operations that do not conflict, and so could possibly be implemented in the same resource. The standard high-level synthesis problem of resource allocation is to find the fewest resources to implement the graph. This is equivalent to partitioning the compatibility graph into the fewest cliques possible. In a standard approach described in [5], cliques are extracted by sorting nodes in descending order of degree. This is modified by the authors of [2] to sort nodes in descending order of wordlength.

In contrast, our approach operates by formulating a cost function on the conflict graph. In the conflict graph, nodes represent operations, and edges represent two operations that cannot be bound to the same resource. This paper demonstrates how an appropriate cost function can be used to achieve optimal colourings of the conflict graph, and therefore minimal area implementations of multiple-wordlength systems. We introduce optimal and heuristic approaches to the colouring problem, and demonstrate that a surprisingly simple heuristic can achieve highly successful results.

Section 2 of this paper introduces the problem of multiple-wordlength binding, and illustrates the effectiveness of current FPGAs for implementing such designs. Section 3 introduces optimal and heuristic solutions to the binding problem, the effectiveness of which is illustrated in Sect. 4. Finally some conclusions are drawn in Sect. 5.

2 Wordlength Binding

2.1 Addition and Multiplication

The multiple-wordlength resource binding problem is somewhat different for the two operators of multiplication and addition. In this discussion it is worth remembering that adders are typically small, and efficiently implemented in FPGA devices, while parallel multiplier resources are generally large and inefficiently implemented [6].

Consider the circuit shown in Fig. 1, where two 2-bit additions can be performed during a single clock cycle, and a single 4-bit addition can be performed during another clock cycle. By chaining together the carry-chains of two 2-bit adders, it is possible to produce a single ripple-carry computational unit, able to perform both operations. For adders, this is discussed in more detail in [7,8].

We now demonstrate that the same is not true for multiplication operations. Consider the 4×4-bit multiplication arrangement shown in Fig. 2(a) [9]. As shown by the multiplexers in Fig. 2(b), in order to perform a 4×3-bit and a 4×2-bit multiplication using a single 4×4-bit multiplier, it is necessary to break the sum/carry chaining and insert alternative operands at internal points within the multiplier array. It soon becomes clear that any general design will suffer

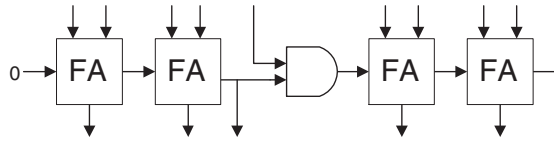

Fig. 1. Two adders chained to produce a single large adder

greatly from routing congestion, as all operands must be fed into their appropriate locations deep within the multiplier array, and there must be similar routing paths for the results. One possible solution to this problem is to limit the type of resource-sharing that can take place for multipliers. Figure 2(c) illustrates the use of a single 5×5-bit multiplier to implement a 2×3-bit and a 2×2-bit multiplier on one clock cycle, and a 5×5-bit multiplier on another cycle. Note that unlike the arrangement in Fig. 2(b), during a single cycle each row or column belongs to at most one multiplier. This allows existing vertical and horizontal-running wires to be used both for passing inputs to multiplier arrays, and for collecting results from them. Using this approach data need only be fed into, and collected from, the boundaries of the multiplier array.

If multiplier resources are to be reused in this way, an efficient implementation of the basic cell, shown in Fig 2(d) is necessary. For FPGA implementations, we propose three possible approaches to provide the necessary control signals to the multiplexers and gates inside the array: use of globally distributed signals,

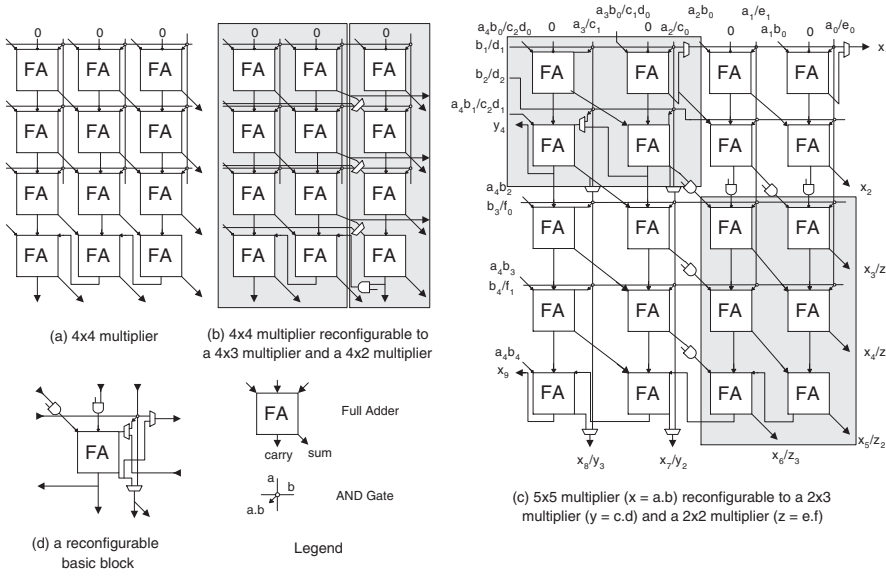

Fig. 2. Sharing a large multiplier resource

static 'scan-path' implementation, or dynamic reconfiguration. Many FPGAs have the ability to globally distribute a handful of signals. While this is certainly a possible solution, use of such signals is expensive, as it denies their use to any other circuits within the FPGA. Also for a multiplier array shared by several multipliers, several signals would be needed. An alternative method is to use a scan-path technique to reconfigure the multiplier array - each basic cell (Fig. 2(d)) can be in one of four states: 'outside multiplier', 'inside multiplier', 'right-edge', or 'bottom edge', indicating where in a multiplier the specific cell lies. Each cell therefore needs two state bits, which can be loaded using a scan-path mechanism [10] with two scan-paths, one horizontal and one vertical. Thus an entire n by m array can be reconfigured within $\max(n, m)$ cycles. Clearly the key problem of such an implementation is the significantly increased complexity of the basic cell over that of a simple multiplier array, which may outweigh the area gain from sharing the resource. Indeed the problem is compounded by the fact that each output of a basic cell is now driven by logic within that cell, whereas for the original multiplier array, several outputs were simply routing constructs on their respective inputs. This is confirmed by a Xilinx XC4k implementation of a 16×16-bit array of multiplier cells, which uses 938 CLBs for the cells in Fig. 2(d), but only 290 CLBs for a more traditional basic cell - the costs clearly outweighing the advantages.

We believe that the only feasible approach to sharing a single multiplier between several smaller multipliers within an FPGA is to use dynamic reconfiguration. However the Xilinx XC6200 and Virtex, two existing FPGA architectures offering partial dynamic reconfiguration, are clearly not suited to this task. The Virtex reconfiguration is too coarse grained - we estimate it would take approximately 400 cycles to completely reprogram a 16x16 multiplier array. This rules out Virtex for all but the longest of schedules. By contrast the XC6200 has the fine-grained architecture necessary, including features such as wild-carding. The problem is that by using the new basic cell, all routing must be local, unlike a more traditional design, where inputs can be fed through non-local routing to all participating cells. However the XC6200 has a relative paucity of local routing, leading in our estimation to an area growth of over two times.

It it therefore clear that for current FPGAs, while the use of multiple wordlength resources is a useful technique to minimize utilization [1], it is unlikely that sharing a single multiplier between two or more multipliers *on the same cycle* is an efficient approach. For adders, this is not necessarily true [8], and in any case adders consume significantly fewer FPGA resources than multipliers [6]. We therefore concentrate in the rest of this paper on formulating and solving a multiple-wordlength resource binding problem, wherein a large multiplier array can be used as a *single* multiplier on each clock cycle (though it may have different sizes at different cycles).

2.2 Problem Formulation

The *resource conflict graph* $G_{-}(V, E)$ is a graph whose vertex set $V = \{v_i,\ i = 1, 2, ..., n_{ops}\}$ is in one-to-one correspondence with the opera-

tions, and whose edge set $E = \{\{v_i, v_j\}\ i, j = 1, 2, ..., n_{ops}\}$ denotes conflicting operation pairs [5]. We define the *annotated multiplier conflict graph* to be the graph $G_-(V, E)$ together with an annotation (n_i, m_i) for each vertex v_i. Each annotation consists of an ordered pair of wordlengths, representing the two input wordlengths for the multiplier in question, ordered such that $n_i \geq m_i$ for all i. A *colouring* on the annotated multiplier conflict graph is a labelling $C : V \rightarrow L$ of the vertices by a set of labels L such that no edge in E connects two vertices with the same label. Approximating the area of an n-bit by m-bit parallel multiplier resource by nm, we may associate with each feasible colouring C a cost K_C defined in (1).

$$K_C = \sum_{\ell \in L} \left(\max_{i:C(v_i)=\ell} \{n_i\} \max_{i:C(v_i)=\ell} \{m_i\} \right) \quad . \tag{1}$$

The multiple-wordlength multiplier resource binding problem can then be defined as finding a feasible colouring C^* such that K_{C^*} is minimal. Note that if $n_i = n$ and $m_i = m$ for all i, this problem reduces to minimum colouring, since K_C becomes proportional to the number of colours. It is important to note that the *optimal* colouring may not always be a *minimal* colouring. Figure 3 illustrates a simple annotated multiplier conflict graph consisting of four multipliers. The graph can be coloured with two colours (Fig. 3(a)) with a cost of 1024. However, increasing the number of colours to three (Fig. 3(b)) allows a lower cost of 768 to be achieved. Three multipliers have been used (two 16×8-bit and one 32×16-bit) and yet the overall resource requirement is less than with two multipliers (two 32×16-bit). This a key difference between the multiple-wordlength binding problem and the more traditional binding problem, where a minimal colouring (minimum number of multipliers) is the aim.

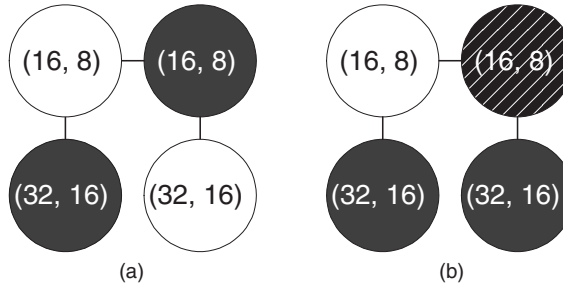

Fig. 3. (a) Minimal and (b) optimal colourings for an annotated conflict graph

3 Binding Algorithms

Having formulated the problem as a form of optimal colouring on an annotated graph, in this section of the paper we present several algorithms to perform the colouring. Three algorithms are presented: a simulated annealing heuristic, a fast effective application-specific heuristic 'swap and inherit', and a branch and bound algorithm, which produces a solution optimal with respect to the cost function. For comparative purposes, and to form a starting point for some of the algorithms, a standard left-edge colouring algorithm was also created, which colours the graph constructively by a sequential scan of the vertex set ordered by schedule start-time [5]. Also for comparative purposes, a simple 'sort by label' heuristic was implemented, which also colours the graph constructively by a sequential scan of the vertex set, but the ordering is by decreasing wordlength. The idea behind this is to 'hide' the cost of smaller nodes behind pre-coloured larger nodes.

3.1 Simulated Annealing

A simulated annealing based heuristic has been developed to solve the optimal colouring problem. Three annealing moves are provided: swap node colours, inherit node colour, and new node colour. Given two nodes of different colour, the 'swap node colours' move swaps the colours of the two nodes. The total number of colours in the graph is unchanged. The 'inherit node colour' also operates on a pair of nodes: node v_1 inherits the colour of node v_2, and the colour of node v_2 is unchanged. Thus if before the move there is only one node of colour $C(v_1)$, the overall number of colours in the graph will decrease. Otherwise it will remain unchanged. Finally, the 'new node colour' move operates on a single node, which must not be the only node of its colour. This node is re-coloured to a currently unused colour, increasing the total number of colours in the graph by one.

This framework of moves provides a structured way to move from one feasible graph colouring to another, by excluding infeasible moves. The starting point of the annealing is a left-edge colouring of the graph, as this provides a reasonable quality solution for the case when resources are limited (see Sect. 4 of this paper).

Our simulated annealing heuristic has several features. Due to the highly nonlinear nature of the cost function (1), if nodes were selected in a purely random manner for the moves described above, the vast majority of moves would have zero cost, since the maximum within each class would remain unchanged in these cases. The standard Metropolis algorithm for simulated annealing would always perform these moves [11]. This would lead to many moves giving little feedback to the cost-estimation process - it is hard to know whether these moves are 'good' or 'bad' until they become non-zero cost. If these moves were considered by the annealing algorithm it would therefore take many times more iterations to come to a result which cannot be known to be better than a result achieved by excluding these operations.

For pairwise operations, a necessary condition for a change in the cost function (either positive or negative) is that one of the nodes is has either the max-

imum n or maximum m value of its colour. By maintaining two lists of nodes, sorted in descending order of n and m respectively, our algorithm efficiently chooses only pairwise annealing moves where this condition is satisfied.

3.2 Branch and Bound

An exact method for finding an optimal solution to the colouring problem has also been implemented using a branch and bound technique, as illustrated in the pseudo-code below.

```
optimum_colouring( vertex, best_cost, num_colours ) {
  if( vertex = NULL ) {  // No more vertices to colour
    if( evaluate_cost() < best_cost )
      best_cost = evaluate_cost();
  } else {
    if( partial_cost() < best_cost ) { // Worth continuing (*)
      for c = 1 to num_colours {
        if( colouring vertex with c is feasible ) {
          colour( vertex ) := c
          optimum_colouring( vertex->next,best_cost,num_colours );
        }
      }
      colour( vertex ) = num_colours+1; // Always feasible
      optimum_colouring( vertex->next, best_cost, num_colours+1 );
    }
  }
}
```

Without the line marked `(*)` in the pseudo-code, this would provide a complete enumeration of all possible colourings. This line contains the bound, using a function `partial_cost()`, which evaluates the cost up to the colouring of the current vertex. The key point here is that the cost function (1) can never decrease by adding more vertices, only remain the same or increase. This forms the basis of the branch and bound.

3.3 Swap and Inherit

A fast and effective heuristic, 'swap and inherit', has also been developed to solve this colouring problem. From an analysis of the optimal colouring of realistic small graphs (fewer than 40 nodes can be coloured using the branch and bound algorithm in reasonable time), it was observed that non-minimal optimal colourings are, in reality, quite rare. In fact, never more than about 15% of those graphs generated by the TGFF algorithm [12]) required non-minimal colourings for optimality. If the range of possible graph colourings is restricted to minimal colourings, the search space is significantly reduced and other, faster algorithms become plausible candidates.

The swap and inherit heuristic operates as a steepest-descent local search, starting from a left-edge colouring. At each iteration, every feasible non-zero cost swap and inherit operation is considered in turn. At worst case this is a total of $n_{ops}(n_{ops}-1)$ ordered pairs of vertices. However, using the condition on non-zero cost operations described in Sect. 3.1 significantly reduces the number of vertex pairs that need to be considered. The operation providing the biggest pay-off in cost-function reduction is then performed, before proceeding to the next iteration. This loop is repeated until no pairwise operation will result in a cost-reduction, at which point the algorithm terminates. Note that the cost-function does not need to be recalculated from scratch at each operation, since the nature of (1) shows that post-operation cost can be easily derived from pre-operation cost using only local information about the annotation of each node taking part in the operation, together with the two largest n and m values for each of the twos colours.

The swap and inherit heuristic has the advantage over simulated annealing that it will quickly and efficiently find minima in the cost function. It clearly has the disadvantage that these minima may be local, due to both the greedy nature of the search performed, and the lack of an 'add new colour' move, which always has positive cost and would therefore be rejected anyway by a greedy algorithm. It is our contention that these disadvantages are relatively unimportant for realistic problems, as will be demonstrated in Sect. 4.

4 Results

The algorithms described in Sect. 3 have been implemented and tested on a set of randomly generated conflict graphs with varying properties. Firstly random sequencing graphs were generated using the method described in [12]. Nodes in these graphs were identified as adders or multipliers (the two types of operation node generated by our high level synthesis environment [1]). Each adder was assigned a latency of one clock cycle, and each multiplier a random latency uniformly distributed in the range 4–8 clock cycles. The wordlengths of each multiplier were then assigned in the range 16–32 bits, such that operations with a longer latency have larger wordlengths. Each sequencing graph was scheduled using the list-scheduling algorithm [5], with a constraint on the maximum number of multiplier and adder resources. From the scheduled sequencing graph, the conflict graph for each resource-type/latency was produced: two nodes are connected in the graph if their scheduling intervals intersect. Finally, these graphs were processed by the multiple-wordlength binding algorithms presented in this paper, in addition to the 'left-edge' and 'sort by label' algorithms mentioned in Sect. 3.

Shown in Fig. 4(a) is a plot of the difference between the cost using each algorithm and that using a random-order sequential scan of the vertex set [5], as a proportion of the random-order cost. Each point on the graph represents the average of 200 random sequencing graphs. The swap and inherit heuristic consistently out-performs all algorithms except simulated annealing, which marginally

outperforms the heuristic by at most 5% even for very large sequencing graphs. Interestingly, the left-edge algorithm outperforms sort-by-label for a large number of sequencing nodes. This is because as the graph complexity increases, it becomes more important to consider the internal structure of the graph, a factor ignored by the sort-by-label algorithm.

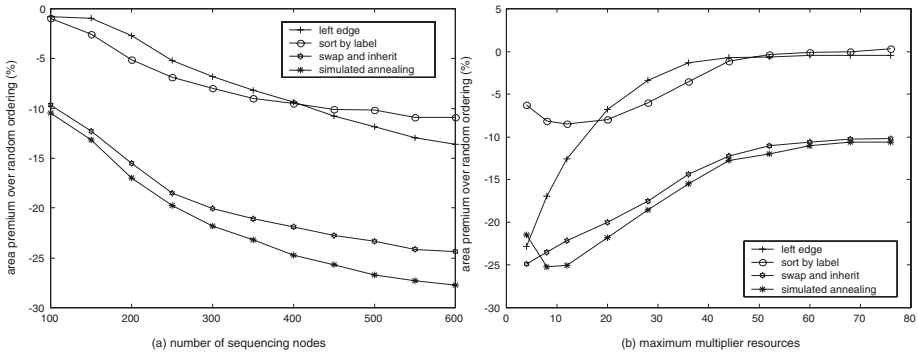

Fig. 4. Variation of multiplier area premiums (a) with number of nodes (for fixed resource constraints) (b) with resource constraints (for fixed number of nodes)

Figure 4(b) illustrates how the performance of the algorithms changes as resource constraints on the scheduling algorithm are relaxed. Left-edge shows a significant advantage for tight resource constraints, however this advantage is rapidly lost as the constraints are relaxed, as the importance of the internal structure of the graph is reduced. The swap and inherit algorithm lies consistently beneath all curves curves except simulated annealing, which marginally outperforms it, illustrating the robust performance of the heuristic. Indeed, for very tight resource constraints the swap and inherit outperforms simulated annealing, illustrating that the same annealing schedule and parameters are not optimal over the entire problem parameter space. By contrast, since the swap and inherit heuristic has left-edge as its starting point and is greedy, left-edge area gives a upper-bound on swap and inherit area.

5 Conclusions

After consideration of multiple-wordlength resource sharing, we conclude that for current FPGAs the multiple-wordlength multiplier resource binding problem should be defined such that a multiplier array is used to implement a single multiplier on each clock cycle. This problem has been formulated as an optimal colouring on an annotated conflict graph, and several algorithms have been developed to solve the colouring problem. It has been shown that while a simulated

annealing solution provides good quality solutions, for the price of a small area premium a fast steepest descent algorithm can be substituted. Both algorithms show significant area reductions over the use of more traditional approaches for the case of multiple-wordlength datapaths.

Our current work on this problem includes integrating the scheduling and binding problems to produce sharing between multiple-wordlength resources of different natural latency, and the investigation of the interaction between the wordlength optimization scheme described in [1] and the high-level synthesis techniques presented in this paper.

Acknowledgements

The authors would like to acknowledge the support of Hewlett-Packard Laboratories and the Engineering and Physical Sciences Research Council, U.K.

References

1. Constantinides, G.A., Cheung, P.Y.K., Luk, W.: Multiple precision for resource minimization. To appear in Proc. FCCM 2000
2. Kum, K., Sung, W.: Word-length optimization for high-level synthesis of digital signal processing systems. Proc. IEEE Int'l. Workshop on Signal Processing Systems SIPS'98 (1998) 569–678
3. Willems, M., Bürgsens, V., Keding, H., Grötker, T, Meyer, M.: System-level fixed-point design based on an interpolative approach. Proc. 34th Design Automation Conference (1997) 293–298
4. Cmar, R., Rijnders, L., Schaumont, P., Vernalde, S., Bolsens, I.: A methodology and design environment for DSP ASIC fixed point refinement. Proc. Design Automation and Test in Europe '99 (1999)
5. DeMicheli, G.: Synthesis and optimization of digital circuits. New York: McGraw-Hill (1994)
6. Haynes, S.D., Cheung, P.Y.K.: Configurable multiplier blocks for use within an FPGA. IEE Electronics Letters **34** (1998) 638–639
7. Landwehr, B., Marwedel, P., Dömer, R.: OSCAR: Optimum simultaneous scheduling, allocation and resource binding based on integer programming Proc. European Design Automation Conference (1994) 90–95
8. Ercegovac, M., Kirovski, D., Potkonjak, M.: Low-power behavioral synthesis optimization using multiple precision arithmetic. Proc. 36th Design Automation Conference (1999)
9. Hwang, K.: Computer arithmetic: principles, architecture and design. New York: Wiley and sons (1979)
10. Weste, N.H.E., Eshraghian, K.: Principles of CMOS VLSI design. Reading, MA: Addison-Wesley (1993)
11. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes in C. Cambridge, UK: Cambridge University Press (1988)
12. Dick, R.P., Rhodes, D.L., Wolf, W.: TGFF: Task graphs for free. Proc. CODES/CASHE'98 (1998) 97–101