High-Level Synthesis with Reconfigurable Datapath Components

George Economakos

National Technical University of Athens
School of Electrical and Computer Engineering
Microprocessors and Digital Systems Laboratory
Iroon Polytexneiou 9, GR-15780 Athens, Greece
geconom@microlab.ntua.gr

Abstract

High-level synthesis is becoming more popular as design densities keep increasing, especially in the ASIC design world. Although FPGA design follows ASIC design methodologies and FPGA densities are increasing too, programmable devices also offer the advantage of partial reconfiguration, which allows an algorithm to be partially mapped into a small and fixed FPGA device that can be reconfigured at run time, as the mapped application changes its requirements. This paper presents a novel resource constrained high-level synthesis scheduling heuristic, which utilizes reconfigurable datapath components. The resulting schedule can be shortened so as the gain in clock cycles can overcome the timing overhead of reconfiguration. The main advantage of the proposed methodology is that through run time reconfiguration, more complicated algorithms can be mapped into smaller devices without speed degradation.

1. Introduction

Modern consumer digital devices are built using either application specific hardware modules (ASICs) or general purpose software programmed microprocessors, or a combination of them. Hardware implementations offer high speed and efficiency but they are tailored for a specific set of computations. On the contrary, software implementations can be modified freely during their life-cycle but they are much more inefficient in terms of speed and area. Reconfigurable computing [3] is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Reconfigurable devices, including *Field-Programmable Gate Arrays* (FPGAs), contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, usually called logic blocks, are connected using a set of programmable routing resources. Custom digital circuits can be mapped to the reconfigurable hardware by computing the logic functions of the circuit within the logic blocks, and using the configurable routing to connect them. Currently, the most common configuration technique is to use *Look-Up Tables* (LUTs), implemented with *Random Access Memory* (RAM).

The areas of a program that can be accelerated through the use of reconfigurable hardware may be too numerous or complex to be loaded simultaneously onto the available hardware. In such cases, it is beneficial to be able to swap different configurations in and out of the reconfigurable hardware as they are needed during program execution. This concept is known as *Run-Time Reconfiguration* (RTR). Through RTR, more sections of an application can be mapped into hardware and thus, despite reconfiguration time overhead, a potential for an overall performance improvement is provided. RTR can be applied on different phases of the design process, according to the granularity of the reconfigurable blocks, which may be complex functions [9], simple RTL components [1] or LUTs [11]. The reconfiguration data can be stored inside the reconfigurable device [10] or transfered from an embedded or host processor [9]. The underlying architecture can be traditional FPGAs or special purpose architectures [6, 11, 12], supporting very fast reconfiguration.

High-Level Synthesis (HLS) [7], where a behavior is mapped into an RTL architecture, has a great impact on circuit implementation because each HLS transformation acts on large portions of the design. Reconfiguration in HLS can be applied in the construction of the RTL architecture considering that each RTL component is not active in every control step. Partially inactive components can be merged into a reconfigurable component. This paper presents a novel resource constrained HLS scheduling heuristic, which utilizes reconfigurable datapath components covering different function types in different control steps through RTR. Since RTR in modern architectures can take as little as 10ns [12], a part of every control step may be used for calculations and the other part for RTR. With the proposed heuristic, the resulting schedule can be shortened enough to overcome the timing overhead of reconfiguration. Experimental results show an average 50% reduction in control steps that compensates for the worst case of 50% increase in clock period, with better hardware utilization.

2. Related Research

RTR is a leading technology improvement of reconfigurable computing. A key point for its broad acceptance is how to conduct reconfiguration quickly and flexibly. Conventional FPGAs have not focused on RTR much, because they have been used mainly for emulation and prototyping purposes. This has started to change and new architectures are proposed. In [6, 12], the proposed architecture can store up to 8 different contexts to configure LUTs with a fast hardware context switch. In [11] a similar architecture with 8 different contexts is presented along with a scheduling algorithm to partition a technology mapped design. However, scheduling after technology mapping is less effective than the proposed scheduling at the RT level. A hardware context switching in a conventional FPGA architecture is proposed in [10], but it requires much more hardware resources than a non-RTR approach. In [8] RTR efficiency is considered by partitioning a dataflow graph into a minimal set of reconfigurations applied through the whole run time of the corresponding application. Reconfigurable components are presented in [2] and [4]. In [2] the *morphable multiplier* is presented, which is an array multiplier that can be configured through multiplexers to work as either an adder or a multiplier. In [4] morphable multipliers are used for the design of a graphics processor. Reconfigurable computing for HLS is reported in [1] where register binding is handled by reconfiguring on-chip embedded memory.

3. The Proposed Solution

This paper considers RTR during HLS proposing a novel resource constrained scheduling heuristic utilizing RTR arithmetic units. Specifically, after experimentation with different FPGA architectures, it has been found that a binary multiplier takes 3 to 4 times the LUTs required for an adder of the same input bit width. So, we can assume that we have an arithmetic component that can be used as a multiplier in some control steps and as 3 adders (at least) in all the others. If we perform resource constrained scheduling with such reconfigurable components we can reduce the latency, in terms of control steps, of our circuit.

For example, consider a digital filter with two inputs x and y and two outputs z_1 and z_2, where $z_1 = a_0x_0 + x_1 + x_2 + a_3x_3 + x_4 + a_5x_5$ and $z_2 = b_0y_0 + b_1y_1 + y_2 + y_3 + b_4y_4 + y_5$. If we want to build a circuit for this system, using two multipliers and two adders in every control step, we will come out with the schedule of figure 1. If one of the multipliers is reconfigurable, and as in the above discussion can be used as either a multiplier or 3 adders, we can reduce the latency by one control step, as shown in figure 2.

Such a result is promising but to apply RTR a reconfiguration delay is required at the beginning of appropriate control steps and thus, the control step period must be extended. So, in the above example, the one control step gain will be outperformed by the increase in clock period. However, if we want to implement the system using two multipliers and one adder we will come out with a large schedule, shown in figure 3. In that case, making one multiplier reconfigurable will result in a more drastic latency improvement, as shown in figure 4. If RTR can reach 10ns, as reported in modern architectures, the latency reduction is almost 50%.

4. Scheduling with RTR Logic

Practical problems in hardware scheduling are modeled by generic sequencing graphs, with possibly multiple-cycle operations of different types. With this model, the scheduling problem is known to be intractable. Therefore, heuristic algorithms have been researched and used. For resource constrained scheduling, that is when the number of available hardware resources is determined, a very efficient and widely used algorithm is list scheduling [7]. In its general form, list scheduling is the following algorithm.

```
INSERT_READY_OPS($V$,$PList_{t_1}$,$PList_{t_2}$,...,$PList_{t_m}$);
Cstep=0;
while (($PList_{t_1} \neq \emptyset$) or ... or ($PList_{t_1} \neq \emptyset$)) do
  Cstep=Cstep+1;
  for k=1 to m do
    for funit=1 to $N_k$ do
      if ($PList_{t_k} \neq \emptyset$) then
        $S_{current}$=SC_OP($S_{current}$,FIRST($PList_{t_k}$,Cstep));
        $PList_{t_k}$=DELETE($PList_{t_k}$,FIRST($PList_{t_k}$);
      endif
    endfor
  endfor
  INSERT_READY_OPS($V$,$PList_{t_1}$,$PList_{t_2}$,...,$PList_{t_m}$);
endwhile
```

The algorithm uses a priority list $PList$ for each operation type $t_k \in T$. Each operation's priority is defined by its *mobility*, that is the difference between its ALAP scheduling value and its ASAP scheduling value. The operations in all priority lists are scheduled into control steps based on N_k which is the number of functional units performing operation of type t_k. The function INSERT_READY_OPS scans

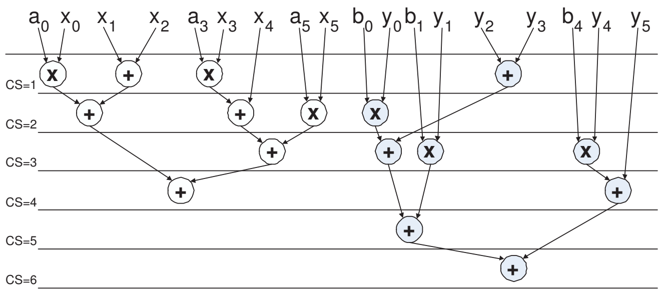
a_0 x_0 x_1 x_2 a_3 x_3 x_4 a_5 x_5 b_0 y_0 b_1 y_1 y_2 y_3 b_4 y_4 y_5
CS=1
CS=2
CS=3
CS=4
CS=5
CS=6

Figure 1. Schedule with 2 mult. - 2 add.

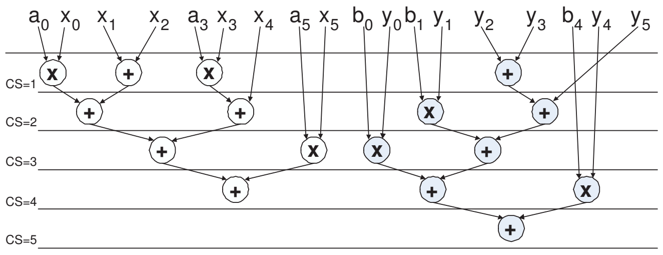
a_0 x_0 x_1 x_2 a_3 x_3 x_4 a_5 x_5 b_0 y_0 b_1 y_1 y_2 y_3 b_4 y_4 y_5
CS=1
CS=2
CS=3
CS=4
CS=5

Figure 2. Schedule with 1+1 mult. - 2 add.

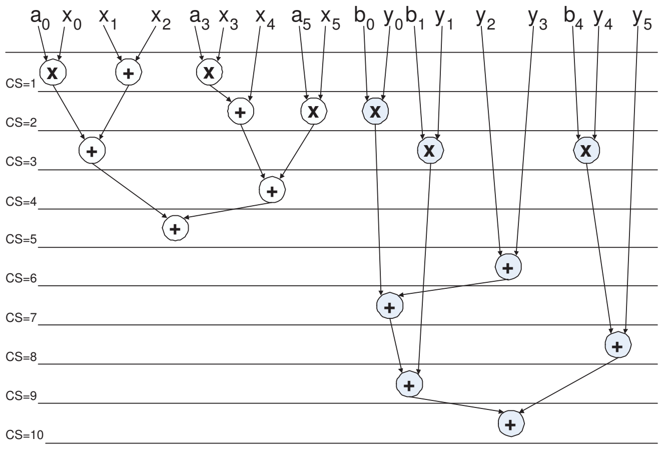
a_0 x_0 x_1 x_2 a_3 x_3 x_4 a_5 x_5 b_0 y_0 b_1 y_1 y_2 y_3 b_4 y_4 y_5
CS=1
CS=2
CS=3
CS=4
CS=5
CS=6
CS=7
CS=8
CS=9
CS=10

Figure 3. Schedule with 2 mult. - 1 add.

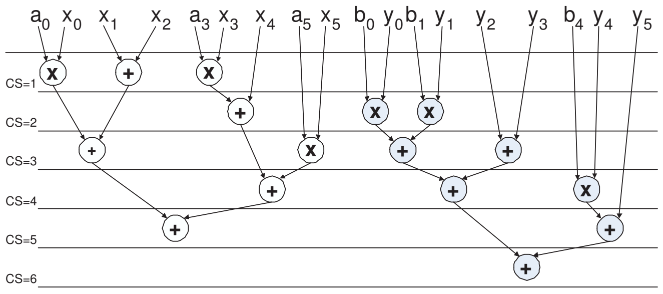
a_0 x_0 x_1 x_2 a_3 x_3 x_4 a_5 x_5 b_0 y_0 b_1 y_1 y_2 y_3 b_4 y_4 y_5
CS=1
CS=2
CS=3
CS=4
CS=5
CS=6

Figure 4. Schedule with 1+1 mult. - 1 add.

the set of nodes V and moves ready operations (with all predecessors scheduled) to the appropriate priority list. The function SC_OP($S_{current}$,o_i,s_j) schedules operation o_i in control step s_j. The function DELETE($PList_{t_k}$,o_i) deletes operation o_i from the specified list. Operations with low mobility are put first in the list. In other words, operations that do not have many opportunities to be scheduled in subsequent control steps are preferred for the current. As the algorithm moves on, the ASAP and ALAP values change and thus mobilities are dynamically re-calculated.

The same algorithm can be used when a subset of the resources are reconfigurable and through RTR can be used in some control steps as one type and in all the rest as another type. For example, a reconfigurable binary multiplier can be used as either a multiplier or (at least) three additions. The required modifications are the following:

- The priority lists corresponding to types covered by a reconfigurable operator are merged into a common list.
- The numbers of available resources of each type, including RTR resources, are kept in separate variables.
- When both reconfigurable and non-reconfigurable components can perform an operation, the latter takes precedence. So priority lists are merged after all non-reconfigurable components have been used.
- At each control step a reconfigurable component can cover only one operation type.
- The number of available reconfigurable components is not decreased with each operation scheduled but only when full coverage is reached.

With the above modifications, the new resource-constrained scheduling heuristic with reconfigurable components can be realized. The circuits designed using this heuristic may have to pass through RTR at every control step. This can double the control step period or lower the operating frequency. However, frequency degradation can be outperformed by the latency improvements achieved (more on that in the following section). Considering applications that are bound to a specific I/O protocol, like the 33 MHz PCI bus, our approach can dramatically improve the effectiveness of small partially reconfigurable FPGA devices. Moreover, better hardware utilization is supported because idle components may be reconfigured to perform other tasks. Finally, the proposed reconfiguration is kept minimum by utilizing very few (less than five) reconfigurable components. Small scale reconfiguration can be performed in a flexible and fast way both in modern architectures but also in future proposals, which will try to take full advantage of RTR.

Application	Number of nodes	Number of cycles		
		0 RTR	1 RTR	2 RTR
Fircls	63	24	18	10
Firls	64	32	25	17
Firrcos	79	42	30	18
Invfreqz	41	25	18	10
Maxflat	115	51	38	22
Remez	55	28	20	17

Table 1. DSP schedules with RTR

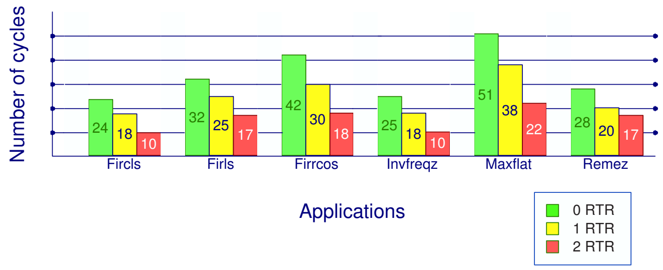

Figure 5. Graph of DSP schedules with RTR

5. Experimental Results

The scheduling algorithm of the previous section has been implemented on top of a C-to-RTL HLS synthesis environment [5]. In order to evaluate the proposed methodology, six different DSP applications (from MATLAB's DSP tool box) have been used as testbenches: Fircls (Constrained least square FIR filter), Firls (Least square linear-phase FIR filter), Firrcos (Raised cosine FIR filter), Invfreqz (Discrete-time filter from frequency data) Maxflat (Generalized digital Butterworth filter) and Remez (Parks-McClellan optimal FIR filter). Table 1 shows three implementations for each application, one with no reconfigurable components (the reconfigurable multipliers discussed above), one with one reconfigurable component and one with two reconfigurable components. The implementations with two reconfigurable components have an average latency improvement of 53% that can overcome even a doubling in control step period due to RTR. Figure 5 summarizes the results of table 1 and gives another view of the improvements achieved.

6. Conclusions

A novel resource constrained HLS scheduling heuristic, which utilizes reconfigurable datapath components has been presented in this work. Using reconfigurable multipliers, the resulting schedule can be shortened so as the gain in clock cycles can overcome the timing overhead of reconfiguration. Since RTR in modern reconfigurable architectures has been reported to take as little as 10ns, an approach where half of the control step may be used for calculations and the other half for RTR is feasible. The main advantage of this solution is that through RTR, more complicated algorithms can be mapped into smaller devices without speed degradation.

References

[1] H. Al Atat and I. Quaiss. Register binding for FPGAs with embedded memory. In *12th Annual Syumposium on Field-Programmable Custom Computing Machines*, pages 167–175. IEEE, 2004.

[2] S. Chiricescu, M. Schuette, R. Glinton, and H. Schmit. Morphable multipliers. In *12th International Conference on Field Programmable Logic and Applications*, pages 647–656. IEEE, 2002.

[3] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, 2002.

[4] K. Dale, J. W. Sheaffer, V. V. Kumar, and D. P. Luebke. Applications of small scale reconfigurability to graphics processors. Technical Report CS-2005-11, University of Virginia, 2005.

[5] G. Economakos, P. Oikonomakos, I. Panagopoulos, I. Poulakis, and G. Papakonstantinou. Behavioral synthesis with SystemC. In *Design Automation and Test in Europe Conference and Exhibition*, pages 21–25. ACM/IEEE, 2001.

[6] T. Fujii, K. Furuta, M. Motomura, M. Nomura, M. Mizuno, K. Anjo, K. Wakabayashi, Y. Hirota, Y. Nakazawa, H. Ito, and M. Yamashina. A dynamically reconfigurable logic engine with a multi-context/multi-mode unified-cell architecture. In *International Solid-State Circuits Conference, Digest of Technical Papers*, pages 364–365. IEEE, 1999.

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

[8] S. Ghiasi, A. Nahapetian, and M. Sarrafzadeh. An optimal algorithm for minimizing run-time reconfiguration delay. *ACM Transactions on Embedded Computing Systems*, 3(2):237–256, 2004.

[9] C. Patterson. High performance DES encryption in virtex FPGAs using JBits. In *Symposium on Field-Programmable Custom Computing Machines*, pages 113–121. IEEE, 2000.

[10] J. Torresen and K. A. Vinger. High performance computing by context switching reconfigurable logic. In *16th European Simulation Multiconference*, pages 207–210, 2002.

[11] S. Trimberger. Scheduling designs into a time-multiplexed FPGA. In *6th International Symposium on Field Programmable Gate Arrays*, pages 153–160. ACM, 1998.

[12] M. Yamashina and M. Motomura. Reconfigurable computing: Its concept and a practical embodiment using newly developed dynamically reconfigurable logic (DRL) LSI. In *5th Asia and South Pacific Design Automation Conference*, pages 329–332. ACM/IEEE, 2000.