Synthesis of Signal Processing Structured Datapaths for FPGAs Supporting RAMs and Busses

Baher Haroun and Behzad Sajjadi

Department of Electrical and Computer Engineering, Concordia University
1455 de Maisonneuve Blvd. W., Montreal, Quebec H3G 1M8
e-mail: [haroun,behzad]@ece.concordia.ca

ABSTRACT

A novel approach is presented for transforming a given scheduled and bound signal processing algorithm for a multiplexer based datapath to a BUS/RAM based FPGA datapath. A datapath model is introduced that allows *maximum flexibility in scheduling bus transfers independent of operation scheduling*. A novel integer linear programming (ILP) formulation that optimally selects and assigns data-transfers to busses *while scheduling* the bus transfers to minimize a linear combination of the number of busses, bus loading in terms of tristate drivers and fanout, registers and register file storage (RAM) locations. We demonstrate that our resulting optimal datapaths compare favorably to others for signal processing synthesis benchmarks such as: single and multiple elliptic filter and fast discrete-cosine-transform (FDCT).

1 Introduction

FPGAs are being used in signal processing applications as re-configurable computation accelerators. Two of the families of FPGAs that is specifically suitable for such applications are the Xilinx XC4000 series and the ORCA ATT2C series, which support fast arithmetic functions as well as efficient storage of variables in SRAMs [2]. Moreover, these FPGAs have global wires that can be driven by tri-state drivers to implement either wide muxes or chip wide busses. Other structural characteristics that influence the structure of a datapath are:

(a) The *limited interconnection resources* for FPGAs. This makes interconnection a primary resource for optimization especially for large bit-width datapaths. Otherwise, larger (more costly) FPGAs may be required. Also for FPGAs, the interconnection delay is large due to the configurable series switch resistance and capacitive effects. Hence, limiting interconnection loading is critical in improving the cycle time of the architecture. ***(b)*** The abundance of registers and of storage RAM bits (in relation to the number of logic gates that implement the computational functional units (FUs)). Two hardware mechanisms can be used to store data in an XC4000 FPGA: 1) Using registers at the output of each combinational logic block (CLB) (two look-up tables (LUTs) and two registers per CLB), and 2) Using RAMs by re-configuring the LUTs. One CLB can store up to 32 bits of data. Therefore, *1 CLB can potentially store 16 times as much if it is configured as a RAM instead of as a register. The use of SRAM instead of registers in variable storage can result in significant savings in storage area.* ***(c)*** CLBs are composed of programmable logic (LUT) followed by a register. Hence, datapaths which utilize this property of having registers in front of logic, stand to maximize the usage of the FPGA resources.

There are a number of synthesis approaches targeting a *random topology* multiplexer based datapaths, i.e., a point to point connectivity model using muxes and registers. Examples are: Cathedral-III[8] which uses heuristics, or using optimal approaches (ILP), e.g. [1], [5], [9], [3] and [6], these approaches can be suitable for *multiplexer based* FPGA architectures but do not explicitly address the above three concerns for FPGAs supporting busses and RAMs. While others targeting bus based architectures, (e.g. Cathedral-II [7], HYPER[16], SPAID-X[13], STAR[4], and [15]) use an architecture model that has bus/RAM access time in series with functional unit delay in one cycle. For such architectures cycle times are long and may be more suited for ASIC like implementations were special techniques can be used to reduce the effects of bus /RAM delay[12] but not in the case of FPGAs.

2 Synthesis System Overview

The following aspects synthesizing FPGA datapaths are the contributions of this paper:

1- To present an architectural model that can be used for bus/RAM based FPGA data-paths that allows for efficient interconnection and data storage with an architecture commensurate with the FPGA logic blocks.

2- To show that the complexity of architectural search for FPGA bus/RAM based architectures can be effectively handled as follows; a first step for *scheduling and operation binding* as done with multiplexer based architectures, followed by a second step for *bus transfer scheduling and assignment* and then the last and third step for performing *interconnection and storage binding*. This is performed *provided that:* a “structural complexity” [1] is minimized in the first step, register and RAM cost together with bus loading are minimized in the second step and physical floorplanning, routing and delay modeling (system clock cycle duration) is optimized in the last step [10]. An overview of the synthesis system is shown in Figure 1.

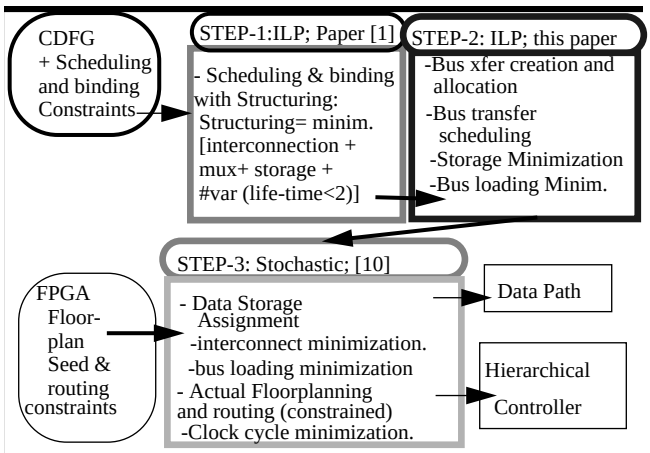

Figure 1: *Our Tool,* OSTA, *uses an ILP approach for Scheduling and bus assignment followed by a stochastic storage and interconnection binding. P*aper [1] describes the scheduling and binding for structured multiplexer based architectures for FPGAs. Structuring increases the existence of data transfers with life time > 2 that are specifically suitable to be assigned to busses hence reducing the number of local interconnections by merging them with busses in step-2 of the synthesis. In [10], the interconnection and storage binding is done in conjunction with floor planning to ensure true physical constraints of the implementation technology are taken into account

3- To present an ILP formulation, to **concurrently** minimize number of busses, bus loading and data storage while optimally assigning bus transfers to busses, scheduling these transfers and minimally allocating busses. Our proposed architectural model (see 1 above and Section 3) allows this flexible bus transfer scheduling.

4- To show that ILP handles *small/medium size* applications and is a viable approach to architecture synthesis of bus/RAM based architectures. On the other hand, *for large problems*, heuristic/stochastic approaches may be suitable. In this case, this paper provides proof that our flexible bus transfer datapath and synthesis model producing structured architectures, is an effective approach. Hence our introduced model, the constraints and the optimization criteria can be suitable for other search (e.g. stochastic) techniques.

Section 3 presents the architectural model and the synthesis data transfer model used in our optimization of bus transfers. Section 4 presents the ILP formulation and Section 5 shows the effectiveness of our bus/RAM based datapath synthesis approach.

3 Modeling and Optimization Criteria

3.1 Underlying Architectural Model Supporting RAMs and Busses

The model of the architecture that is used in our synthesis technique follows the general structure of Figure 2. For an example of a full data path see Figure 6. Different types of modules used are: register module, register-file module (implemented as a RAM), functional unit (FU) module and FU multiplexer sub-module. If the number of mux inputs is small, the FU multiplexer module can be part of the function of the CLBs implementing the FU module. On the other hand, when the number of mux inputs is large, extra mux sub-modules are implemented using extra CLBs.

*A two phase clock (*Φ1 *and* Φ2*) can be used* to define the data transfers between the register-file (RAM) and the registers of the data path. For data transfers between the registers of the data path only one clock phase (ϕ1) is used. The data moves from a data path register to the FU input through a FU output, then the data moves from the FU output to one or *more* of the data path registers.

For the transfers between the registers and the register file,Φ1 *and* Φ2, are used. The data path registers are considered as "master" registers and the slaves are the storage locations inside the register-file (RAM). This clocking is explained in Figure 3 (b). Note that the RAM is written in the beginning of the cycle (Φ1) and read at the end (Φ2 (or $\overline{\Phi 1}$)). Hence, the cycle time is determined by either the critical path between any of the data path registers or the read and write time plus the bus delays of a register file. Input /output ports to the system can be considered as registers (R).

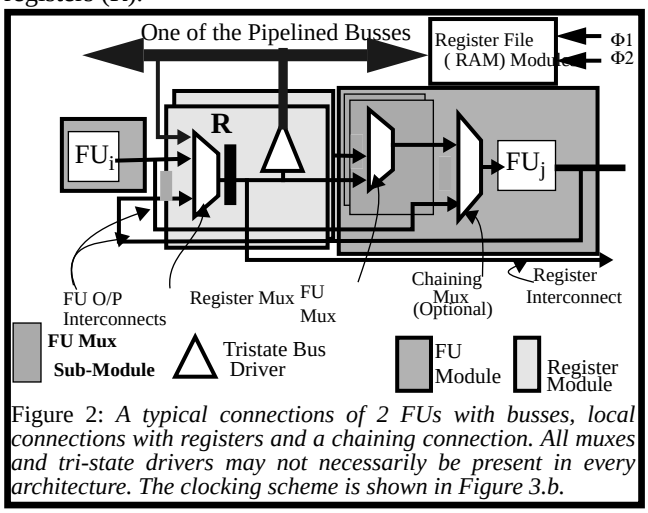

Figure 2: *A typical connections of 2 FUs with busses, local connections with registers and a chaining connection. All muxes and tri-state drivers may not necessarily be present in every architecture. The clocking scheme is shown in Figure 3.b.*

3.2 RAM and Bus Support

Our architecture model allows both individual registers and grouped storage in register files which are implemented as RAMs, hence reducing storage area. Support of RAMs is essential, especially in the case of handling signal processing algorithms requiring large storage (e.g. multi-channel filters). Figure 3 (a) shows the conventional register file model (a) where the register file is accessed by a non-pipelined bus(e.g in [3][12][13] and [15]). Figure 3 (b) shows our proposed clocking scheme where a register file based on RAM implementation is accessed by a pipelined bus. In case of the non-pipelined bus access (conventional case), the read and write time of the register file array are part of the system cycle. Chaining the read and write time in one cycle increases the cycle time which reduces the clock rate and hence may result in a large performance reduction.

By pipelining the bus, the read and writes are scheduled in separate cycles preceding and following the operation

respectively. The decision then to store the data in a large RAM can only be done if the life time of a variable is at least two cycles.

By allowing any variable to exist in more than one storage location, that is in a register and in a register file location, and by supporting flexible bus transfers, our approach has removed the bus delay from the critical path. The busses and their associated register files are, hence, treated as functional units and their influence on the clock cycle duration is independent of other functional unit delays. Therefore, the clock cycle of the datapath is controlled by either a critical path through a FU or through a register file and a bus but not by adding both, as is the case in conventional bus based datapaths. Hence, our architecture model has faster clock rates than conventional approaches and also uses less busses and resources as will be shown.

3.3 The Register and Bus Binding Model

Given the above architectural model, there are a number of different cases for which an edge in the Data Flow Graph (a data transfer) can be bound. These cases are encompassed by the two generic transfers shown in Figure 4. Notice that the bus in Figure 3. (b) (see also Figure 6.b) communicates only with registers. In Figure 4, a bubble indicates the clock cycles at which the data is transferred from a register (R_n) to the bus and the square indicates the cycle where the data is read from the register file through a bus to a register (R_p). These cycles are determined in step-2 of Figure 1, together with bus allocation. In a sense, a scheduling of bus transfers is performed at step-2. *This flexibility of scheduling bus transfer as shown in Figure 4 is very different from all previous architecture models in the literature using busses.* All other models have to transfer data immediately at the end of an operation, or upon its start. Our model allows relaxing this constraint which is one of the contributions of this paper. This relaxation in using busses can result in significant reduction in the number of busses required (see Section 5).

4 STEP-2: ILP Search for an Optimal Bus Based Architecture

4.1 The ILP Bus Assignment and Scheduling

We assume that operation scheduling and binding have been performed (step-1, Figure 1) while minimizing inter-connections and maximizing the number of transfers with life-time > 1 using the structural complexity measure [1].

The objective of step-2 is to minimize: (a) the number of parallel bus transfers which reduces the number of busses allocated, (b) the maximum overlap in registers which reduces the number of registers allocated, (c) the maximum overlap in life-times of variables assigned to busses which reduces register file storage locations, (d) the number of registers having tri-state access to the busses hence minimizing tri-state output loading on each bus and (e) the number of destination registers that a bus is connected to, this also minimizes bus loading. Note that in step-2, no register binding is made, only the scheduling of the bus transfers and selecting which transfer goes on a bus and which bus that transfer uses..

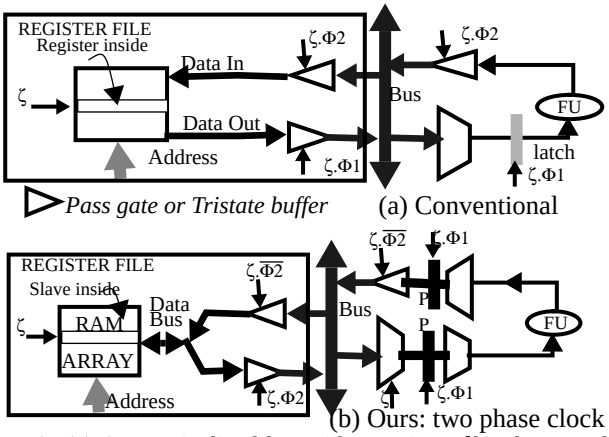

Figure 3: *(a) A non-pipelined bus with a register file where read and write access delays are chained in one cycle. (b)Our Clocking scheme: A pipelined bus and register file (note the two phase clocking) where a write is followed by a read in one cycle. The bits ζ's generated from the controller are "AND"ed with the clocks (*i.e. *ζ.Φ1, ζ.Φ2) to indicate the latching time. Our scheme requires that data stored in a register file to have a life time greater than or equal to 2 cycles.*

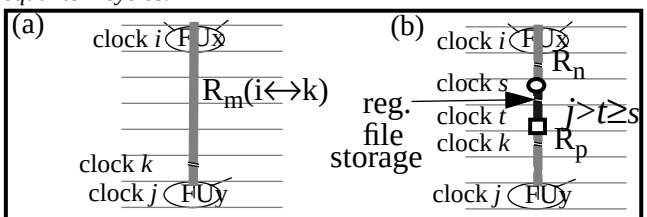

Figure 4: (a) Shows a data transfer using local interconnections through individual register Rn. (b) Shows a data transfer using a pipelined bus. Rn and Rp do not have to be different, but have to have at least a life time of one cycle. The variable is stored in a register file between clock cycle s and k through Bus_q.

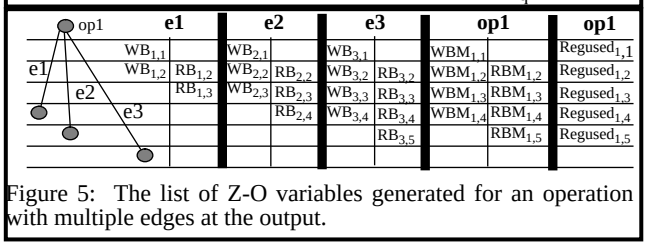

Figure 5: The list of Z-O variables generated for an operation with multiple edges at the output.

. The following are the notations and variables used in our ILP formulation given in Table 1 for the bus scheduling and binding. **E:** set of edges with lifetime >= 2cycles. $\mathbf{N_B}$**:** Max. number of buses allowed. **D(Bus):** Represents the delay of the bus. **Multoutput:** Set of operations with multiple edges at their output. **Single:** Set of edges that do not belong to Multiple output operations. **EdgeOverlap(s):** Set of edges that overlap at each cycle. $\mathbf{WB_{(e,s,n)}}$**:** =1 only if clock cycle "s" is used to write the variable (edge) "e" in the register file through bus "n", otherwise = 0. $\mathbf{RB_{(e,s,n)}}$**:** =1 only if clock cycle "s" is used to read the variable (edge) "e" from the register file through bus "n", otherwise = 0. $\mathbf{WBM_{(op,s,n)}}$**:** =1 only if clock cycle "s" is used to write any of the variables (edges) at the output of operation "op" in the register file through bus "n", otherwise = 0. $\mathbf{RBM_{(op,s,n)}}$**:** =1 only if clock cycle "s" is used to read any of the variables (edges) at the

output of operation "op" from the register file through bus "n", otherwise = 0. **Range(e_X):** The cycles in which the Z-O variables "x" exists ("x" could be RB or WB or...). **Regused$_{(op,s)}$:** =1 only if any of the edges at the output of operation "op" is alive at clock cycle "s", otherwise = 0. **MinReg:** An integer variable used to count the lower bound on the number of registers needed to implement the architecture. **TotReg:** An integer variable used to count the maximum lifetimes of all the registers. **MaxRegFile$_n$:** an integer variable used to count the maximum required number of RAM locations per register file. **Tristate:** This number of tristate drivers for every bus is calculated and the maximum

. .

Table 1

$$\sum_{n=1}^{N_B} \sum_{s \in Range(e_{WB})} WB_{e,s,n} \le 1 \qquad \forall e \in E \qquad \textbf{(1)}$$

$$\sum_{n=1}^{N_B} \sum_{s \in Range(e_{RB})} RB_{e,s,n} \le 1 \qquad \forall e \in E \qquad \textbf{(2)}$$

$$\sum_{s \in Range(e_{WB})} WB_{e,s,n} - \sum_{s \in Range(e_{RB})} RB_{e,s,n} = 0 \qquad \forall e \in E \qquad \forall n = 1 \ldots N_B \qquad \textbf{(3)}$$

$$\sum_{p=s-D(Bus)+1}^{ALAP(e_{WB})} \sum_{n=1}^{N_B} WB_{e,p,n} + \sum_{p=ASAP(e_{RB})}^{s} \sum_{n=1}^{N_B} RB_{e,p,n} \le 1 \qquad ALAP(e_{WB}) + D(Bus) - 1 \ge s \ge ASAP(e_{RB}) \quad \forall s \quad \forall e \in E \qquad \textbf{(4)}$$

$$WB_{e,s,n} - WBM_{op,s,n} \le 0 \qquad \forall op \in Multoutput \quad \forall e \in op \quad \forall n \quad \forall s \in Range(e_{WB}) \qquad \textbf{(5)}$$

$$RB_{e,s,n} - RBM_{op,s,n} \le 0 \qquad \forall op \in Multoutput \quad \forall e \in op \quad \forall n \quad \forall s \in Range(e_{RB}) \qquad \textbf{(6)}$$

$$\sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} WB_{e,s,n} + \sum_{op \in Multoutput} WBM_{op,s,n} \le 1 \qquad \forall n \qquad \forall s \qquad \textbf{(7)}$$

$$\sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} RB_{e,s,n} + \sum_{op \in Multoutput} RBM_{op,s,n} \le 1 \qquad \forall n \qquad \forall s \qquad \textbf{(8)}$$

$$1 - \sum_{n=1}^{N_B} \sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n} + \sum_{n=1}^{N_B} \sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n} - Regused_{op,s} \le 0 \qquad \forall op \in Multoutput \quad \forall e \in op \quad \forall s \in (Range(e_{WB}) \cup Range(e_{RB})) \qquad \textbf{(9)}$$

$$\sum_{\substack{e \in \\ Single \\ \wedge EdgeOverlap(s)}} \left(1 - \sum_{n=1}^{N_B} \sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n} + \sum_{n=1}^{N_B} \sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n} \right) + \sum_{\substack{op \in \\ Multoutput}} Regused_{op,s} + reg_s - MinReg \le 1 \quad \forall s \qquad \textbf{(10)}$$

$$\sum_{e} \left(1 - \sum_{n=1}^{N_B} \sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n} + \sum_{n=1}^{N_B} \sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n} \right) - TotReg \le 1 \qquad \textbf{(11)}$$

$$\sum_{e \in EdgeOverlap(s)} \left(\sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n} - \sum_{p=Asap(e_{RB})}^{s-1} RB_{e,p,n} \right) - MaxRegFile_n \le 0 \qquad \forall s \quad \forall n \qquad \textbf{(12)}$$

$$\sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} \sum_{p=s}^{Alap(e_{RB})} RB_{e,p,n} + \sum_{op \in Multoutput} \sum_{p=s}^{Alap(op_{RBM})} RBM_{op,s,n} \le Tristate \qquad \forall n \qquad \forall s \qquad \textbf{(13)}$$

$$Objective Function \leftarrow Minimize\ (C1 \times MinReg) + (C2 \times TotReg) + \left(C3 \times \sum_{n=1}^{N_B} MaxRegFile_n \right) + (C4 \times Tristate) + \left(C5 \times \sum_{op \in Multoutput} \sum_{s} \sum_{n=1}^{N_B} WBM_{op,s,n} \right) + \left(C6 \times \sum_{op \in Multoutput} \sum_{s} \sum_{n=1}^{N_B} RBM_{op,s,n} \right) + \left(C7 \times \sum_{op \in Multoutput} \sum_{s} Regused_{op,s} \right)$$

of them is assigned to this integer variable.

The first two constraints of Table 1 ensure that only one clock cycle is used to write the variable in the register file through one bus and one cycle to read it back from the register file through one (same or another) bus.

The third constraint ensures that every variable assigned to a bus transfer is written in and read from the same register file through one bus. Without constraint 3, use of multi-ported register files are enabled where one variable is written through one port (from one bus) and read from another port (and another bus).

Inequality 4 ensures that a read occurs after a write to a register file for any variable. Constraint 5 and 6 are intended for operations with multiple edges at its output. We assign a new set of zero-one (0-1) variables (WBM, RBM) for these output edges (as shown in Figure 5). These 0-1 variables are forced to "1" if there is a read from a register file or a write to a register file at any cycle for any of the multiple output edges. For instance in Figure 5, $WBM_{1,2}$ is set to "1" when at least one of the variables $WB_{1,2}$, $WB_{2,2}$, $WB_{2,2}$ is equal "1". The same argument applies to the RB and RBM variables. To ensure that at every cycle only one data variable can be read and only one data variable written through any one specific bus, we enforce constraints 7 and 8. In these constraints we use WB and RB as 0-1 variables for edges that do not belong to multiple output operations and WBM and RBM as 0-1 variables for the edges of the multiple output operations.

To account for the register cost in the cost function, edges of multiple output operations are assigned a set of 0-1 variables ($Regused_{op,s}$) per cycle "s". These 0-1 variables are set to "1" using constraint 9, when any of the multiple edges for an operation is alive in cycle "s". Constraint 10 determines a lower bound (integer variable MinReg) on the number of registers that can be assigned in step-3. Constraint 11 indirectly ensures that the total register life-time is reduced. Constraint 12, counts the maximum required number of RAM locations per register file. For all the busses, constraint 13 counts the maximum bus loading due to tri-state drivers per bus.

The objective function to be minimized has seven components. The first four terms contribute directly to the final datapath structure. The last three terms are essential for the correct assignments of the Z-O variables in the ILP formulation.

4.2 Register and Multiplexer Binding with Datapath Generation

In this paper, we report the register bindings obtained by using the tool in [10]. In summary, the tool used performs the register binding while at the same time performing a floorplanning of the datapath to be able to compute the routing delay and area cost and its effect on the clock cycle duration. Such a tool produces better results than independently determining a register binding followed by a floorplanning. It uses a stochastic (simulated annealing) search while continuously minimizing the critical paths that determine the clock cycle for the datapath together with layout area.

5 Results

Elliptic Filter: We use the 5th order elliptic filter to demonstrate a number of points regarding our architecture features of pipelined busses, the operation binding using the structuring approach and our bus transfer scheduling. Figure 6 (a) shows a 2 adder one multiplier schedule with 17 cycles. The *operation* schedule is very similar to the one obtained by ALPS [3] (this filter is retimed). Our ILP tool, OSTA running on a SPARC10, produced a scheduling and operation binding (step-1) in 21.3 CPU seconds. The bus scheduling and binding (step-2) took 1.7 seconds. The detailed register binding was done in conjunction with floorplanning (step-3) and took 210 CPU seconds[10] (and 1700secs for ILP[17]). All results proven optimal. Note that only **one pipelined bus** is used compared to an optimal of **7 busses** for the OASIC[5] architectural model, and **4 busses** of the SPAID-X style architecture used in [13][15] which is equivalent of a 400% saving in the number of busses. Because these busses require global wires which are not abundant in FPGAs, such a saving which is a direct consequence of our approach is very significant in the routability of a datapath. .

	#cycles	#TS (#BC)	#mxi/ps	#Bus (nets)	#Regs (#lc)	#CLB / bit
OASIC[12]	18	na	na	7(na)	9	4.5
InSyn [6]	19	na	na	4(na)	8+(5)	6.5
SPAIDX[13]	19	19	18	5(3)	-(21)	5
IP [15]	19	12(23)	11	4(10)	(10)	4
[17]-lm1	19	6(12)	19	3 (8)	11	5.5
[17]- LM2	19	na	25	- (11)	11	5.5
STAR[4]	19	16(28)	17	5 (na)	13	6.5
OSTA (ours)	17	2 (6)	22	1 (7)	7(6)	4

Table 2: Architecture for EWF, 2 adders 1 pipelined multiplier. Only nets with fanout > 1 are counted. Our estimate for the number of CLBs for storage includes recursive storage. If recursive edges are added to the others solutions, up to an extra four registers or 2 CLB/bit of word length are needed. #TS= #tristate drivers, #BC= # of Bus Connections as in [15]. #CLB/bit is the # of CLBs used per bit of the word width of the datapath. #lc is the number of latches or RAM locations.

To show how our model and synthesis compares with others, we highlight a number of measures: number of busses, bus loading, number of networks with fanout >1 (indicates complex interconnections), number of tristates, fanout of networks, and components of delay on the critical path.

Table 2 compares different synthesized datapaths for the EWF.

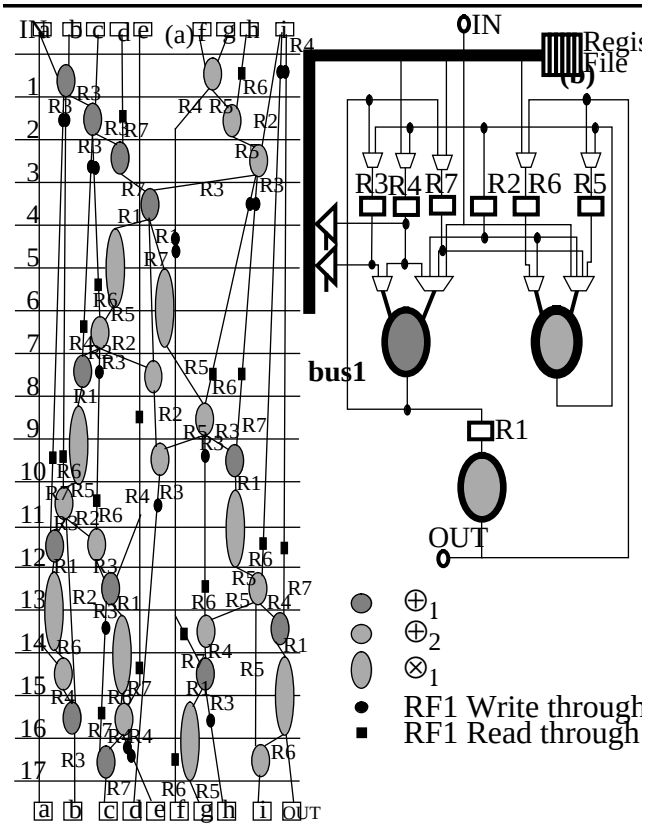

Figure 6: (a) Details of the scheduling and bindings for the elliptic filter example. (b) The resulting architecture schematic. This architecture accounts for all recursive variable storage.

For the EWF architecture shown in [15], the maximum bus loading is 3 inputs and three tri-state drivers, which is the same for the architecture in Figure 6. Note that, in the architecture in [15] bus delays and RAM access time are added to the delay of a functional unit and registers to obtain the critical path determining the cycle time. While for the architecture in Figure 6, the RAM access and FU delay are independent as discussed before. It is evident that our architecture is simpler and uses less interconnections.

Regarding storage cost, our architecture uses a total of 7 registers and 6 register file locations (4 CLBs per bit of datapath width), compared to an optimum of 9 registers for a mux based (without busses[1]) datapath which is equivalent to using 4.5 CLBs per bit of data path width for storage. It is important to note that we include the cost of storage of all recursive variables (z^{-1} delays in the EWF filter specification) unlike almost all other published solutions (the 8 recursive variables i-b, Figure 6 are not bound to storage in other references). The result by Li & Mowchenko [18] accounts for the recursive edges and uses 11 registers or 5.5 CLBs per bit of data path width for storage. This data path can either be viewed as a bus based architecture (LM-1) or a multiplexer based architecture (LM-2) in Table 3. We use less multiplexer inputs (> 12%), less CLBs for storage (>40%) and 300% less for the number of busses.

Our architecture has a maximum loading of three drivers on the bus and 3 outputs (mux inputs). The maximum fanout of any register output is 4. The maximum number of mux inputs is 4. These values are the best values for any published EWF architecture.

Cascaded-Elliptic Filter: An alternative example that requires significantly more storage and is suitable to highlight the advantages of RAM/bus based architectures is a filter composed of two elliptic filters connected in cascade (output of first is directly the input of the second). The operation scheduling and binding (solution time 58 sec.) as well as the bus scheduling (solution time 33 sec.) are shown in Figure 7. For storing all variables, i*ncluding the recursive edges*, our resulting datapath uses 11 registers at a cost of --(1/2 CLB/bit/register) and 11 register file (RAM) locations at a cost of (1/2 CLB/bit for all 11 locations). In comparison, a multiplexer based solution would use 18 registers at a cost of (1/2 CLB /bit/register). Hence, the bus based solution saves an equivalent of 3 CLBs per bit of the word length.

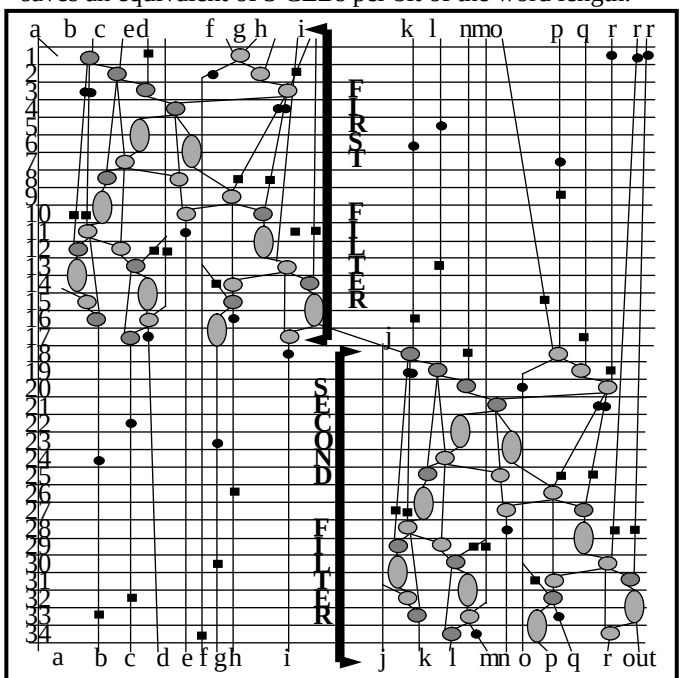

Figure 7: This is the scheduling and binding of two elliptic filters connected in series (output of the first is the input of the second). The architecture uses two adders and one pipelined multiplier. The operation schedule and binding was done using our ILP formulation for step-1. The bus transfer scheduling was done using our step-2 ILP formulation for a one bus solution.

For a 32 bit width data path our estimate for our datapath is: (64 CLBs for the adders and their multiplexers, 206 CLB for a 12x16 booth re-coded multiplier (synthesized), and 192 CLBs for storage, total of 462 CLBs). An extra 3x32=96 CLBs are required for a multiplexer based solution with a total of 558 CLBs. A saving of 20% in terms of the total number of CLBs required for our datapath. Since our architecture uses less interconnections and only one bus, the efficiency due to routing is higher.

For a two bus solution (not shown) the number of registers used in our solution was 8 and the number of total register

files locations are 14. This results in a saving of 4 CLBs per bit of the word length. For a 32 bit width datapath our estimate is 430 CLB and the saving is 30% in terms of the total number of CLBs required for this datapath. Note that if a more efficient multiplier is used, these percentage gains can be increased.

Fast Discrete Cosine Transform: This example is used to demonstrate that ILP can handle medium size graphs that have high parallelism (parallelism is limited in EWF example). Step-1 took 33 seconds. For a mux based solution the number of registers used is 11, while for a step-2 bus binding using one bus which took 1.3 seconds), the architecture uses 8 registers and 4 register file locations. This results in a saving of 1 CLB per bit of the data path width over a multiplexer based datapath that does not use busses or RAMs. More details about this and the values of the cost function coeficcients used can be found in [17].

It is to be noted that the running time of the ILP solution for the bus scheduling and assignment is very small (compared to operation scheduling and binding). This is due to 3 factors; 1- The range of each bus transfer is less than the original graph since all operations are already scheduled. 2- All edges with life time <2 are eliminated from the bus search, 3- The ILP formulation used is tight. This is evident from the number of branch and bound trials did not exceed tens of branches taken in all preceding examples. In some instances where the register costs are not added to the cost function, no branching was observed and the integer optimal solution is obtained directly from the linear program solution.

6 Conclusion

Our synthesis approach when applied to benchmark examples produces architectural solutions that outperforms all other published architectures regarding both its structure, loading (hence, performance) as well as resources used. *This result is due to flexible data transfers on pipelined busses in our architectural model, and also due to the synthesis approach that "structures" the architecture and efficiently schedules the bus transfers while minimizing bus loading.* We have demonstrated that any schedule and binding for a multiplexor based data path can be transformed to support a bus/RAM based datapath by properly scheduling the bus-transfers. In our methodology, by using structural complexity reduction during scheduling and binding, we have shown that bus binding and storage allocation can be delayed to later steps of synthesis and produces good results. Hence, we have proposed an efficient split of the architectural synthesis procedures. We have also shown that ILP techniques can be effectively used in conjunction with complex cost functions, for small to medium size DFGs, to tie together different levels and various tasks in architectural design. We have also shown by our low running times the tightness of the formulation we have presented. Moreover, our constraints and cost functions can be extended to other heuristic architecture search techniques that may have better running times for larger problems. Future research will focus on addressing very large memory required for loop execution and multiple FPGA computation accelerator systems as well as methods of relegating some of the bus binding decision to step-3 (the register binding and floorplanning) of our synthesis approach.

References

[1] B. Haroun, B. Sajjadi, "ILP Synthesis of Signal Processing Architectures with minimum Structural Complexity", CICC, May 1994, pp. 237-240.

[2] J. Rose, A. ElGamal, A.Sangiovani-Vincentelli, "Architecture of Field Programmable Gate Arrays", Proc. IEEE, Vol. 81, No. 7, July 1993.

[3] C.T. Hwang, J.H. Lee, Y.C. Hsu, "A Formal Approach to the Scheduling Problem in HLS",IEEE Tran. CAD, Vol-10, No.4, April 1991, pp464 475.

[4] F.S. Tsai, Y.C. Hsu: "STAR: An Automatic Data Path Allocator". IEEE Tr. CAD. Vol11, No9, September 1992.

[5] C. Gebotys, M.I. Elmasry, "Optimal Synthesis of High Performance Architecture:" JSSC, March 1992, pp 389-397.

[6] M. Rim, R. Jain, R. Deleone, "Optimal Allocation & Binding in HLS", DAC-92, pp.120-123.

[7] G. Goosens, J. Rabaey, J. Vandewalle, H, De Man: "An Efficient Microcode Compiler for Application Specific DSP Processors", IEEE Transaction on CAD, Vol 9, No. 9, pp.925-937,1990.

[8] S. Note, W. Geurts, F. Catthoor, H. De Man, "Cathedral-III: Architecture-Driven High-Level Synthesis for High Throughput DSP Applications", 28th Design Automation Conference, 1991, pp.597-602.

[9] C.Gebotys, M. I. Elmasry, "Global Optimization Approach for Architectural Synthesis", IEEE Tr. on CAD, Vol-12, No.9, Sept. 1993, pp.1266-1278.

[10] A. Safir, B. Haroun, "A Floorplanner for Datapath Optimization" submitted to Tr. VLSI Systems.

[11] B. Haroun et. al. "VLSI Architecture Synthesis and Implementation of HiFI Digital Filters" Proc. Canadian Conference on VLSI, pp.107-114, Oct. 1989.

[12] C. Gebotys "Synthesizing Optimal Application Specific DSP Architectures", in: "VLSI Design Methodologies for DSP Architectures" ed. M. Bayoumi, pp.43-92, Kluwer Academic Publishers, Boston, MA, 1994.

[13] B. Haroun et.al "Synthesis of Multiple Bus Architectures For DSP Applications", in: "VLSI Design Methodologies for DSP Architectures" ed. M. Bayoumi, pp.93-130, Kluwer Academic Publishers, Boston, MA, 1994.

[14] C. Gebotys, "Synthesizing Optimal Register file Architectures for FPGA Technology", CICC, May 1994, pp, 233-236.

[15] J. J. Rabaey, et.al. "Fast Proto-typing of data path Intensive Architecture." IEEE Design & Test, Vol.8, No.2, pp.4051, 1991.

[16] B. Sajjadi, "Architectural Synthesis for FPGA Based Signal Processing Systems", M.Sc. Thesis in preparation, Concordia University, 1995.

[17] T. Li, J. Mowchenko, "Applying Simulated Evolution to High Level Synthesis", IEEE Tr. CAD, March 1993, pp.389-409.