Flexible Datapath Synthesis Through Arithmetically Optimized Operation Chaining

Sotirios Xydis, Ioannis Triantafyllou, George Economakos, Kiamal Pekmestzi
National Technical University of Athens - School of Electrical and Computer Engineering
Microprocessors and Digital Systems Laboratory
Email: {sxydis, ioannis, geconom, pekmes}@microlab.ntua.gr

Abstract
Datapath synthesis incorporating complex operation templates has been proven extremely efficient especially for the Digital Signal Processing (DSP) application domain. However, only architectural level optimizations have been reported for the specification and implementation of the operation templates. This paper introduces the consideration of arithmetic level optimizations for template based datapath synthesis. A high performance architecture for the implementation of DSP kernels is presented. It is based on flexible and arithmetically optimized components able to perform a large set of operation templates. A synthesis methodology for optimized mapping of DSP kernels onto the proposed architecture is also presented. Experimental results are reported showing significant gains in execution time, active chip area and power dissipation in comparison to previously published flexible template-based datapaths.

1 Introduction
Modern embedded systems are dominated by Digital Signal Processing (DSP) applications, i.e. telecom, wireless communications etc. The main characteristic of these applications is that they are extremely computation intensive with light-weight control structures. Their execution time is dominated by certain code segments, called kernels. In order to deliver high performance implementations, previous contributions [2], [6], [8], [11], [12], [14], [15], [16] have proposed the hardware mapping of these kernels. Research activities from the field of High Level Synthesis (HLS) [11], [14], [16], [18] and Application Specific Instruction Processors (ASIP) [2], [6], [12], [15] have presented promising results, concerning the performance metric, when complex hardware resources (custom instructions in the ASIP domain) are utilized instead of primitive ones i.e single ALUs, MULs etc. These complex hardware resources take advantage of chained operation templates found into the initial dataflow graph (DFG) of the DSP kernel. Operation chaining removes the intermediate registers between primitive hardware units, improving their overall delay.

Given that the template based HLS and ASIP methodologies work at the architectural level, most of the aforementioned techniques generate datapaths taking into account only observations on the structure of the DFGs. Thus, little or none awareness of techniques from lower design levels (i.e arithmetic level optimization) is encountered (Conventional Flow in Fig. 1a). The complex operation templates are either specified in a predefined behavioral template library [11], [14], [16] or extracted directly from the kernel’s DFG [2], [6], [12], [15]. Since flexibility has not been considered, a large number of different operation templates is generated reducing the physical homogeneity and the opportunities for efficient template sharing at register transfer level (RTL). Recently, in [8], the usage of flexible operation-templates has been proposed in order to address this type of inefficiencies. Their basic computational node,
called Flexible Computational Component (FCC), comprises of four ALU/MUL units. Their architecture is based on fully interconnecting such FCCs with an intermediate crossbar switch in order to exploit inter-template chaining as well. Although, significant performance improvements in execution time are reported by their approach, large hardware area overheads are imposed due to hardware redundancy. Additionally, the crossbar interconnection switch limits the scalability of their architecture.

The previously referenced contributions present the inherent limitation that the critical delay of the generated datapaths is determined of large carry-propagation chains which are found in the conventional binary arithmetic designs. In order to eliminate the large carry-propagation chains found into datapath components, the Carry-Save (C-S) arithmetic representation is used [4]. In C-S datapaths, chains of 3:2 (Fig. 1b) or 4:2 C-S (Fig. 1c) adders are formed in such a way that multi-input additions (present in multiplication, multiplication/accumulation (MAC) and accumulation/multiplication operations) reduce significantly faster than by doing serial addition of inputs. Many research activities [1], [3], [10], [17], [21] concerned datapath optimization using C-S arithmetic. In [1], [3], DFG transformation techniques are presented in order to either maximize the use of C-S arithmetic [3] or perform common subexpression elimination in C-S computations [1]. However, their studies took into account only ASIC like datapaths without concerning about flexibility of the final architecture. Timing driven synthesis utilizing C-S components is presented in [10], [17]. These techniques operate at the post RTL abstraction when the whole behavioral specification has been laid out as a single circuit. Thus, no flexibility and no hardware-sharing opportunities are supported. In [21] the problem of combined retiming, module selection and C-S arithmetic representation selection is formulated. They introduced Mixed representation Flow Graph (MFG) model which resolves the signal representation mismatch (C-S vs binary). The final datapath comprises a mixture of C-S or binary arithmetic operators. Again, no flexibility issues have been considered.

The main objective of this paper is to enable flexible and high performance datapath synthesis. In order to achieve this, we improve the state of the art solution on flexible datapath synthesis [8] by eliminating its two disadvantageous features, namely be 1) the unawareness of arithmetic level optimizations during architecture specification and 2) the unscalable and area-power hungry crossbar interconnection. Specifically, we introduce a flexible datapath architecture and a comprehensive methodology for synthesizing DSP kernels. The proposed datapath is formed combining concepts and optimization techniques both from the architectural and the arithmetic level of abstraction (Proposed Flow in Fig. 1a). It comprises of uniform and flexible computational units which enable the execution of a large set of operation templates found into DSP kernels. The overall architecture operates in C-S arithmetic delivering high speed implementations. A systematic synthesis methodology for optimized mapping of the DSP kernels onto the proposed architecture, is also presented. Experimental comparisons at the circuit level show that the proposed flexible computational units are able to operate on a significantly larger range of operating frequencies and with lower area overheads than the FCC units in [8]. Also, experiments on a representative set of DSP kernels, show that the proposed approach delivers an average improvement of 29.5% in execution latency together with 42.1% area reduction and 10.1% power gains, compared to the datapath solution in [8].

The rest of the paper is organized as follows. Section 2 presents the proposed architecture and the detailed design of its flexible computational units. In Section 3, the synthesis methodology for mapping DSP kernels onto instances of the proposed architecture is presented. Experimental results are reported in Section 4, while Section 5 concludes the paper.

2 Flexible Datapath Architecture

An abstract architectural model of the proposed datapath is illustrated in Fig. 2. It is composed by 1) the Flexible Computational Units (FCUs), 2) the register bank, 3) the data interconnection network, 4) a CS to binary arithmetic conversion module (CS2Bin) and 5) the control unit which drives the overall architecture (configuration words and multiplexor’s selection signals) in each control step. Each individual component has been designed to operate on C-S word operands of 16-bit, since such a word length is considered adequate for the majority of DSP datapaths [8].

The register bank is introduced for the storage of intermediate results and the sharing and the communication of variables among the FCUs. It is implemented by scratch registers according to the register allocation of each kernel. The data interconnection network handles the communication between the register bank and the FCUs. Different DSP kernels (different register allocation and data communication patterns per kernel) can be mapped onto the proposed architecture utilizing post RTL datapath sharing techniques [13].
The arithmetic conversion module (CS2Bin) performs the conversion from the C-S format to the conventional binary. It is implemented as a simple carry-propagate adder [4], since its critical path delay is overlapped with the critical path delay of the FCUs. The arithmetic conversion usually takes place at the end of each kernel execution in order to output the computed result in binary format.

The number of FCUs is determined at design time according to the available instruction-level parallelism (ILP) of each kernel. Fig. 3 depicts 1) the abstract model of the FCU, 2) the internal structure of the FCU node (detailed model) and 3) the way the abstract model maps onto the detailed one. The abstract model shows that the proposed FCU enables intra-template operation chaining by merging together the pre- and the post-multiplication addition. The alternative execution paths in each FCU are determined by properly setting the control signals of the two multiplexors. These execution paths define the template library in which each FCU can be configured (Fig. 4). Based on the template library of Fig. 4, the DSP kernels are mapped onto the proposed architecture (Section 3). Two level operation templates (of type Add-Mul, Mul-Add) dominates most of the DSP kernels [11]. However, three level operation templates are also occurred in many DSP kernels i.e. in the Symmetrical FIR filters. The proposed datapath supports also this type of templates (i.e T1 of Fig. 4). Inter-template (inter-FCU) chaining has not been considered in order to avoid the unscalable interconnection crossbar among FCUs, found in [8].

The internal structure of the FCU (Fig. 3) consists of 1) a 4:2 C-S adder of 2’s complement numbers, for the addition of the input data (X*, Y*), 2) a C-S to Modified Booth [4] recoding scheme, 3) a tree based adder for the addition of the partial products, 4) the final CS accumulation unit implemented also by a 4:2 C-S adder and 5) a configuration register. The superscript, *, denotes a C-S redundant representation composed of two numbers both in 2’s complement form. Each of the X*, Y*, K*, N*, Z* is in C-S format. The quantities X*C, X*S, Y*C, Y*S, K*S, K*C, N*C, N*S, Z*C, Z*S and A are all 2’s complement conventional binary numbers (Fig. 3). In general, the following relation stands for all C-S formatted data: X* = {X*C, X*S} = X*C + X*S. Thus, input data in conventional binary format, can be directly processed by a FCU without any conversion overhead.

The entire internal structure of each FCU operates directly on, and produces data in C-S arithmetic representation. This feature 1) enables the direct reusability of the unit’s output and 2) eliminates the latency intensive carry-chains found in the conversion from the C-S representation to the conventional binary form. The upper 4:2 C-S adder computes the N* = X* + Y* which can be used either in the pre- or the post-multiplication addition. Input A has to be always in 2’s complement binary numeric representation. At next, the 2’s complement C-S formed data, N* or K*, are driven either for multiplication or addition.

We analyze in more detail the multiplication path since it forms the critical path of the FCU. When the FCU is configured to perform multiplication, the recoding unit is activated. The recoding unit enables the multiplication operation to be conducted with one operand (N* or K*) in C-S format. It performs the conversion from C-S format to an intermediate format of Sign Digits (SDs) [4] and subsequently the conversion of intermediate SDs to the Modified Booth (MB) digit representation [4]. Further details on the implementation of the recoding unit can be found in [19]. It is worth to be noticed that the C-S to SD and the SD to MB conversion modules are carry-propagation-less circuits, contributing only slightly on the critical path delay. The MB digits are passed into the partial product generator and the partial products are reduced within a C-S Wallace tree addition scheme [4]. The multiplier’s output results in 2’s complement C-S format. Since the paper focuses on the overall architecture description and due to lack of space, we neglect further details about the core multiplier’s circuit.

The final accumulation unit is a 4:2 C-S adder with two fixed inputs (the multipliers C-S output) and two configurable inputs. All the inputs of the accumulation unit are

![Figure 3. The Flexible Computational Unit.](image)

![Figure 4. The FCU Template Library.](image)
in 2’s complement C-S format. The configured inputs are either the $N^*$ derived from the initial 4:2 C-S adder or an independent C-S input number ($K^*$). Finally, the operation mode of the FCU and the signs of the input numbers (determining addition or subtraction operations) are controlled through the configuration register by driving with proper control logic bits ($C_{Li}$) the multiplexors and the sign selection inside each 4:2 C-S adder module. The configuration register is loaded in a cycle by cycle basis with configuration words which are generated in the control unit (Fig. 2).

### 3 Synthesis Methodology

An HLS synthesis methodology has been developed in order to enable the efficient mapping of DSP kernels onto the FCU-based architecture. The overall flow of the proposed methodology is depicted in Fig. 5. It consists of 3 major phases. Each phase is reported with different color in Fig. 5, in order to be clearly distinguished.

In phase 1, the DFG of the kernel is extracted from its C code specification. Arrays’ address calculations are statically pre-computed and substituted in the original DFG by register to register transfers. By this way, the FCUs are concentrated on effective computations and not on simple index calculations. Additionally, the proposed datapath produces data in C-S arithmetic representation, which is inadequate format for arrays’ indexes. The DFG graph is unrolled according to the number of multiplication operations in the DFG and the allocated FCUs. By this way, the available ILP is balanced to the utilization of the allocated hardware, since the multiplier is the area-dominant component of the FCU.

The next phase of the proposed methodology is the C-S aware reduction of the original DFG. It is similar to applying aggressively the C-S module selection technique in [21]. The DFG reduction exploits the inherent feature of C-S addition/subtraction circuits to merge/compress multiple additions into a single one [17]. Thus, the size of the DFG shrinks, offering opportunities for faster schedules (number of cycles) than considering primitive resource operations. Conventional C-S aware DFG reduction techniques [1], [3], [10], [17] assumes only 3:2 C-S compressors. The proposed flexible architecture is able to handle 3:2 together with 4:2 C-S compression, since 4:2 C-S compressors are a superset of 3:2 compressors.

The pseudocode of the proposed C-S aware DFG reduction is shown in Fig. 6a. We used the notion of Boundary Operations (B_Ops) similar to [17]. Boundary Operations are the DFG’s nodes which set the boundaries of C-S aware reductions. Thus, C-S reductions are conducted in DFG nodes that lay between B_Ops nodes. In our case, assuming DFG graphs which comprise only addition/subtraction and multiplication operations, three types of B_Ops are encountered, 1) the DFG’s primary inputs, 2) the DFG’s primary outputs and 3) the multiplication nodes of the DFG. At first, the original DFG graph is ASAP scheduled, in order to topologically order the DFG’s nodes according to their timing dependencies. Next, the C-S addition trees are formed iteratively, by merging primitive DFG nodes. The B_Ops are excluded of the merging process, while the high fanout DFG nodes are merged only as roots of the C-S trees. At the end of each iteration, the formed C-S trees are substituted in the original DFG in order to be candidate nodes for extra merging in the next iteration. In each iteration, the 3:2 C-S reductions are considered first and a second pass forms the 4:2 C-S reductions wherever it is possible. The 3:2 compression nodes remain until the end of the C-S aware reduction process in order to include the case of a larger merging at the next steps. After, the completion of the C-S reduction process, the remained uncompressed addition nodes or 3:2 compression nodes are substituted by 4:2 compressor nodes, by adding one or two zero inputs, respectively, at the unbound ports. Fig. 6b illustrates the C-S aware reduction on a sample DFG.

The C-S aware reduction process transforms the original DFG in an intermediate representation which is compliant with the FCU’s resource model. A pattern generation procedure is applied onto the reduced DFG with respect to the FCU template library (Fig. 4). The pattern generation is a covering of the reduced DFG according to the FCU’s operation templates. It actually clusters the DFG’s C-S reduced nodes with the multiplication nodes, in order to perform maximal operation chaining. Currently, the covering

![Figure 5. The proposed Synthesis Flow.](image)
patterns are generated in an exhaustive manner. The designer is responsible to select those patterns that optimally cover the reduced DFG, as far as it concerns the minimization of DFG’s execution time. Automated library based pattern generation and selection [11] can also be incorporated, however these techniques are out of the scope of this paper.

In phase 3, the clustered DFG is scheduled in order to assign each FCU operation to a specific control step. Since the datapath is realized by a fixed number of FCUs, a resource-constrained scheduling problem with the goal of latency minimization is considered. The number of available FCUs and CS2Bin modules determines the resource constraint set. A list-based scheduler [7] has been developed that takes into account the mobility of FCU operations. The ASAP and ALAP time-stamps of the FCU operations are calculated and the mobility for each FCU operation is extracted as the difference of the corresponding ALAP-ASAP value. The FCU operations are prioritized based on the lower mobility value, since the lower the operation’s mobility the most critical the operation. Next, the scheduled FCU operations are bound onto specific FCU instances and the proper configuration bits are generated. After the completion of register allocation, a FSM description is extracted in order to implement the control unit of the overall architecture, and a FSMD [7] model of the FCU based architecture is generated in synthesizable Verilog.

4 Experimental Results

In this section, we provide experimental results showing the effectiveness of our approach. We have compared the proposed datapath with the one presented in [8], which forms the most recent and relevant work to ours.

A comparative area-timing exploration between the two basic computational components, the proposed FCU and the FCC in [8]) has been conducted. We have also included explorative results for the case of a conventional 16-bit multiplication unit (DW_Mul), in order to provide straightforward comparison with the area dominant component (multiplication unit) of non-flexible implementations. All the components were mapped onto the standard cells of TSMC 0.13um technology library, using the Synopsys Design Compiler version 2006 [20]. The arithmetic optimized ”parch” implementations from the Synopsys DesignWare library [20] were considered for both the conventional multiplication unit and the multiplication units found into FCC [8].

The lower limits of the area-timing values for the three implementations were exposed based on an iterative synthesis procedure which generated different delay-constrained versions of the two datapaths. The delay constraint was altered in each iteration considering a time interval of 0,1 ns, with an initial value of 1,0 ns and final value of 5,3 ns. Fig. 7 reports the comparative results. The proposed FCU is able to operate, without timing violations, in a time frame between [2,2ns, 5,3ns]. Respectively, the FCC unit [8] op-
erates without timing violations in a time frame of [3.9ns, 5.3ns]. Thus, the proposed FCU has a larger operative range of about $\Delta T_{FCU} > FCU = 1.7$ns than the FCC [8]. As expected, the DW_Mul unit has the largest violation free timing range of [1.6ns, 5.3ns], among the other two flexible components. The rather small $\Delta T_{FCU > MUL} = 0.8$ns between the proposed flexible computational unit and the optimized non-flexible DW_Mul unit shows the efficiency of our approach. Additionally, comparing the area of the two flexible components at 3.9 ns (the upper operative point of the FCC unit), the proposed FCU delivers approximately $\times 6$ smaller area than the FCC. The comparison of the proposed FCU and the DW_Mul at the upper operative point of FCU ($T = 2.2$ ns) shows that FCC occupies about $\times 3$ larger area than DW_Mul, at that specific point. However, Fig. 7 shows that for operative points larger than 4 ns the area of the FCU converge towards the optimized non-flexible DW_Mul.

A representative set of computationally intensive DSP kernels was formed in order to demonstrate the efficiency of the proposed solution. The benchmark suite consists of: 1) an 8-taps Symmetrical FIR filter (SymFir8), 2) a 16-taps FIR filter (Fir16), 3) a 6th order Elliptic filter (Elliptic), 4) a Volterra IIR filter, 5) the MESA Matrix Multiplication (Mesa Mat_Mul) kernel [5], 6) a straightforward 1-D DCT kernel with unrolled the column’s loop (U-R 1D-DCT), 7) the 2-D DCT (Jpeg DCT) used in JPEG [5] and 8) the 2-D Inverse DCT (Mpeg DCT) used in MPEG [5]. These kernels were mapped 1) onto a FCU-based architecture comprising 4 FCUs and 2) onto a FCC-based datapath with 2 FCCs (= 8 ALUs and 8 Muls) [8]. The SymFir8 and Fir16 have been unrolled 4 times, while each loop of the Volterra filter has been fully unrolled according to the synthesis methodology of Section 3. The Mesa Mat_Mul, U-R 1D-DCT and Jpeg DCT kernels were not unrolled since in each iteration more than 4 multiplication operations were available. For the mapping onto the FCC-based architecture, the above benchmarks were scheduled using the SPARK HLS tool [9] with the aggressive operation chaining option enabled and we manually optimized the resultant FCC-based datapath in order to take into consideration the inter-template chaining, which is not supported by SPARK. Due to the limited space we omitted comparative results between the proposed approach and datapaths composed only by primitive operators. However, such a comparison has been conducted in [8] for the case of FCC-based datapath. Since we are compared with the approach in [8], some straightforward qualitatively conclusions about the performance efficiency of our approach in comparison with primitive resource datapaths can be safely inferred.

The FCU-based and FCC-based FSMD models were synthesized with Synopsys Design Compiler and TSMC 0.13um technology library. For the proposed FCU-based datapath a timing constraint of $T_{clkFCU} = 3.8$ns was imposed while for the FCC-based datapath the timing-constraint was set to $T_{clkFC} = 4.8$ns (the middle values between the upper and lower operative points of Fig. 7, for each flexible component). Power analysis of the synthesized netlists performed with Synopsys PrimePower [20]. Worst case power analysis was considered by imposing $ToggleRate = 1$ to all the inputs and the internal nets of the synthesized netlists. Table 1 reports 1) the actual latency ($\#cycles \times T_{clk}$) in ns, 2) the active area in um² and 3) the power dissipation in mWatt for each DSP kernel.

The proposed datapath delivers faster implementations with smaller area complexity than the FCC-based datapaths in all cases. Specifically, the proposed datapath delivers average latency and area reductions of 29.5% and 42.1%, respectively. The average power consumption is 10.1% lower for FCU-based datapaths. In some kernels the power consumption of the proposed datapath is larger than the FCC-based datapath. This occurs due to the higher operating frequency of FCU datapath in comparison to the FCC datapath. However, the small area complexity (small load capacitance) of FCU-based datapaths amortize the power effect of high operating frequency in most DSP kernels.

We considered three design metrics (Fig. 8), namely 1) the Area-Delay (AD) product 2) the Power-Delay (PD)

<table>
<thead>
<tr>
<th>DSP Kernel</th>
<th>Proposed FCU</th>
<th>FCC [8]</th>
<th>Gain (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latency (ns)</td>
<td>Area (um²)</td>
<td>Power (mWatt)</td>
</tr>
<tr>
<td>SymFir8</td>
<td>15,2</td>
<td>75124,5</td>
<td>7,3</td>
</tr>
<tr>
<td>Fir16</td>
<td>22,8</td>
<td>94108,4</td>
<td>10,0</td>
</tr>
<tr>
<td>Elliptic</td>
<td>22,8</td>
<td>127900,4</td>
<td>11,2</td>
</tr>
<tr>
<td>Volterra</td>
<td>19</td>
<td>99462,8</td>
<td>8,8</td>
</tr>
<tr>
<td>Mesa Mat_Mul</td>
<td>79,8</td>
<td>103862,1</td>
<td>13,3</td>
</tr>
<tr>
<td>U-R 1D-DCT</td>
<td>505,4</td>
<td>198705,3</td>
<td>10,6</td>
</tr>
<tr>
<td>Jpeg DCT</td>
<td>497,8</td>
<td>199884,1</td>
<td>47,1</td>
</tr>
<tr>
<td>Mpeg IDCT</td>
<td>513</td>
<td>19957,3</td>
<td>50,5</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
product and 3) the Energy-Delay (ED) product, in order to evaluate the efficiency of the synthesized datapaths. The AD, PD and ED values (the lower the better) have been normalized according to the respective values of each benchmark for the FCC case [8]. Thus, the top dashed line (value 100%) represents the corresponding AD, PD or ED product of the FCC datapath. FCU datapath solution outperforms the FCC one in all cases and for all design metrics, except the PD value for the MPEG IDCT kernel. The AD values of FCU-based datapaths lay between 20.8%-58.4%. The PD and ED values range between 27.7%-111% and 14.6%-92%, respectively.

5 Conclusion

This paper presented a methodology for high-performance datapath synthesis based on flexible and arithmetically optimized architectural templates. Experimental results on several DSP kernels have shown average performance, area and power improvements of about 30%, 42% and 10% respectively, over previously published high performance and flexible datapath solution.

References


