Low-cost, High-speed Parallel FIR Filters for RFSoC Front-Ends Enabled by CλaSH

Craig Ramsay
University of Strathclyde
Glasgow, Scotland
craig.ramsay.100@strath.ac.uk

Louise H. Crockett
University of Strathclyde
Glasgow, Scotland
louise.crockett@strath.ac.uk

Robert W. Stewart
University of Strathclyde
Glasgow, Scotland
r.stewart@strath.ac.uk

***Abstract*—We present a new low-cost, high-speed parallel FIR filter generator targeting the Xilinx Radio Frequency System on Chip (RFSoC) and direct RF sampling applications. We compose two existing approaches in a novel hierarchy: efficient parallelism with Fast FIR Algorithm (FFA) structures, and efficient multiplierless FIR implementations with H_{cub}. The resource usage advantages (in both area and type) are compared with similar output from the traditional architecture, exemplified by vendor tools, as well as the H_{cub}-based filters without the FFA optimisation. Although these techniques are well studied individually in the literature, they have not enjoyed mainstream use as their structural complexity proves awkward to capture with traditional Hardware Description Languages (HDLs). This work continues a discussion of the use of functional programming techniques in hardware description, highlighting the benefits of having easily composable circuit generators.**

I. Introduction

We present a new family of low-cost, high-speed, parallel Finite Impulse Response (FIR) filters targeting direct Radio Frequency (RF) sampling applications with the Xilinx Zynq UltraScale+ RF System on Chip (RFSoC). These direct RF sampling devices demand fast filtering stages that operate over a number of samples in *parallel*, since the multi-GHz sampling clock is necessarily higher than the fabric clock of the internal Field Programmable Gate Array (FPGA). Figure 1 shows an overview of the XCZU28DR RFSoC's FPGA and RF capabilities, highlighting the relative scarcity of hardened multiply-accumulate resources (DSP48E2s) and the high data-rates produced by the multiple RF Analogue–Digital Converter (ADC) and Digital–Analogue Converter (DAC) channels.

Specific RFSoC use cases for parallel FIR architectures are plentiful. This is exemplified by the vendor support for a set of parallel, or "Super-Sample Rate" (SSR), circuit generators [1]. Example use cases include:

- Instrumentation applications that demand processing of the full available spectrum, including arrays for radio astronomy [2] and quantum computing readouts [3].
- Channelisation of millimetre wave Intermediate Frequency (IF) signals, including 5G NR (FR2) [4]. Such a signal can contain multiple baseband channels, occupying the whole 4 GHz spectrum provided by the RFSoC and demanding parallel filtering architectures for channel (de)multiplexing.

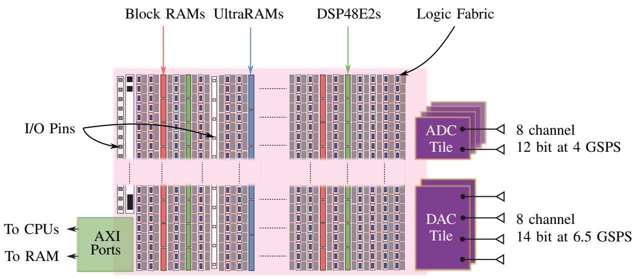


Fig. 1. Overview of RFSoC's FPGA and RF Data Converters

- Custom Digital Up/Down Conversion (DUC/DDC) as a front-end of *any* radio application. Especially useful when the characteristics of the available hardened DUC/DDCs [5] do not meet the application's requirements.

The demand for sample parallelism and the multi-channel nature of the RFSoC device amplifies the effects of filter resource usage, making optimal filter implementation a renewed battle in the current context of RFSoC systems. These optimisations are the concrete focus of our presented work. However, an equally important theme is the reflection on our practical implementation experience, casting a critical eye towards some traditional techniques. Instead of relying on software programming languages and ad hoc circuit generators, we encourage the use of modern functional Hardware Description Languages (HDLs) such as CλaSH [6] for describing families of complex, parameterised Digital Signal Processing (DSP) circuits. Our open source example in CλaSH highlights the realisable benefits of optimisations whose theory is well studied but whose implementations remain, for the most part, in academic folklore.

II. Proposed filter architecture

To improve upon the traditional parallel filter architecture, exemplified by the *System Generator* SSR blockset [1] and LogiCORE FIR Compiler [7], we employ two well studied but seldom implemented techniques. We compose these techniques into a novel hierarchy; one optimisation for the general structure of the filter's parallelism, and another optimisation for the multiplications required within each subfilter. These techniques have enjoyed wide discussion in the literature [8]–

[10] but are less often seen as practical implementations since they both prove to be extremely awkward to describe, at least in a general form, with traditional HDLs.

The following subsections describes the evolution of this structure from the traditional architecture, to a new multiplierless polyphase structure, and finally to our proposed multiplierless Fast FIR Algorithm (FFA) implementation.

A. Traditional Architecture

The traditional architecture for FPGA implementations of parallel FIR filters is a polyphase structure with systolic subfilters, mapped to specialised DSP48E2 multiply-accumulate resources. This approach is exemplified by the LogiCORE FIR Compiler [7] and is often used via the SSR blockset of the *System Generator* tool [1].

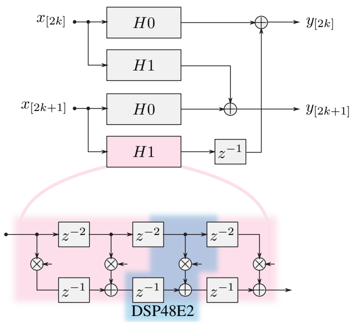


Fig. 2. Example SSR implementation (Polyphase with systolic subfilters) for 8 non-symmetric coefficients

These “SSR” structures can exploit coefficient symmetry, although we only visualise the non-symmetric subfilter architecture above. Figure 3 shows that the footprint scales approximately linearly with the level of parallelism (noting that each subfilter halves in length); with the exception of the extra adders and registers for phase recombination.

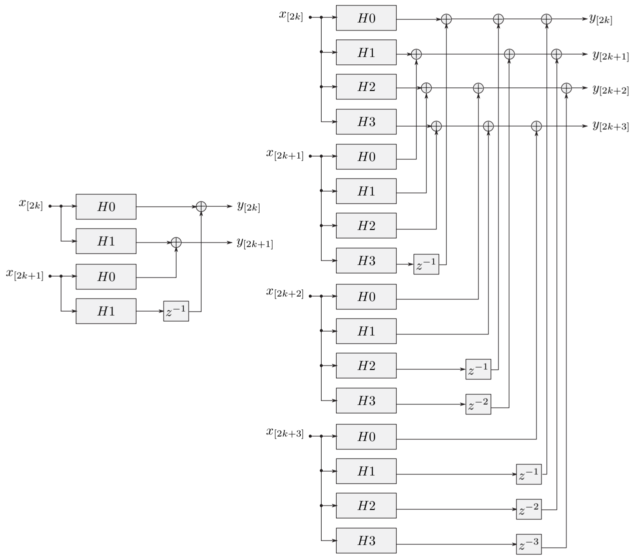


Fig. 3. Scaling of polyphase structures from x2 $\rightarrow$ x4 parallelism

The following section begins to extend this SSR structure, introducing multiplierless subfilters and exploiting resource sharing between subfilters.

B. Polyphase Filter with Shared MCM subfilters

As our first step, consider one of the subfilters in isolation. Figure 4 shows the systolic form being replaced with a transpose form. All of the multiplications now share the input as a common operand — a property which we will exploit despite the higher fan-out of the input signal. The shared input gives us an opportunity to share resources between each of our constant multiplications. The last step in Figure 4 shows the inclusion of a Multiple Constant Multiplication (MCM) block to perform this optimisation.

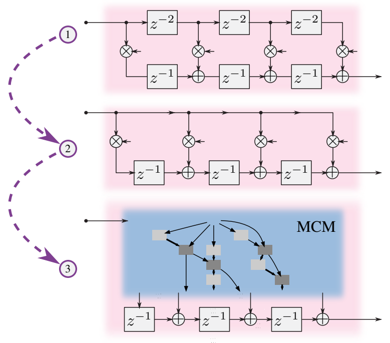


Fig. 4. From systolic FIR form to MCM-based transpose form

Many existing MCM algorithms have been presented in the literature, with the most general approaches being graph-based algorithms. These aim to decompose an expensive set of multiplications into a graph of inexpensive additions and bit shifts (free in the routing, excluding wordlength effects), precluding the need for any specialised DSP48E2s. The topology of these MCM circuits will change profoundly and quite unpredictably depending on the exact set of coefficient values. Most algorithms will try not only to minimise the graph for each multiplication in isolation, but also to optimise for the MCM block as a whole — although finding the optimal solution is known to be NP-complete.

We implement the H_{cub}, RSG, and RAG-n algorithms in [11] using CλaSH, recommending the use of an H_{cub} variant which limits the graph depth at the expense of the number of adders. This will generally result in smaller FPGA areas for fully pipelined MCM blocks due to the predetermined ratio of look-up tables to registers (1:2 for the RFSoC’s architecture); an effect explored further in [9].

Figure 5 shows an MCM graph generated using the H_{cub} variant. It realises an example coefficient set — the 15 coefficient half-band filter (`fir0`) present in each of the XCZU28DR ADC channel’s hardend DDCs. Here we can implement all 15 multiplications with only 5 pipelined adders and 7 pipeline registers, rather than 15 DSP48E2s. This also helps to demonstrate that patterns in the coefficient sets can be readily exploited. We only need to implement multiplications for *unique, odd, positive coefficients*; even-valued coefficients can be recovered through bit shifts, and negative coefficients can simply infer a subtractor in the filter's adder chain.

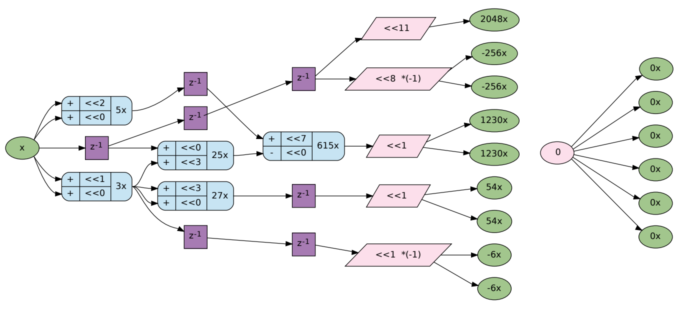


Fig. 5. MCM Graph for `fir0` using an H_{cub} variant

Due to the above effects, as well as more subtle commonality between coefficients identified by the MCM algorithm, implementing fewer but larger MCM blocks will always encourage more resource sharing, resulting in a more area efficient circuit. Figure 6 shows how we can apply this principle to polyphase filters, combining the MCM blocks common to each input sample. Since each shared MCM will implement the full impulse response, H, we will directly exploit both symmetry and antisymmetry in the coefficients.

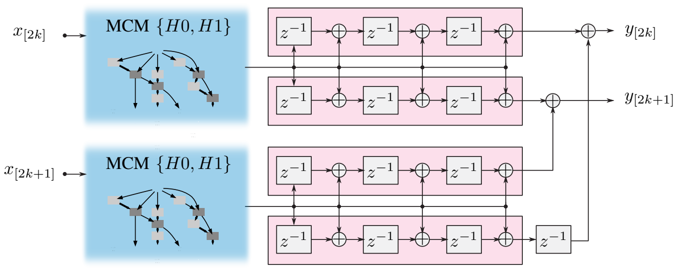


Fig. 6. Sharing MCMs in polyphase filters

The polyphase structure with shared MCM-based filters is one structure we will present implementation results for, but our final optimisation step considers a more complex parallel structure in place of polyphase.

C. FFA Filter with MCM subfilters

We propose the use of FFA for the overall parallel structure of the filter, as opposed to the more common polyphase decomposition. FFA identifies extra resource sharing opportunities and generally requires fewer subfilters, at the expense of extra pre/post adders and increased coefficient wordlengths in some subfilters. An example of a 2-parallel FFA structure is shown in Figure 7. Although further specialisation can be made for higher parallelisms, we will nest successive 2-parallel FFA structures in order to implement any required power-of-two level of parallelism. This form of FFA is recursive in nature and can be difficult to represent and parameterise with VHDL.

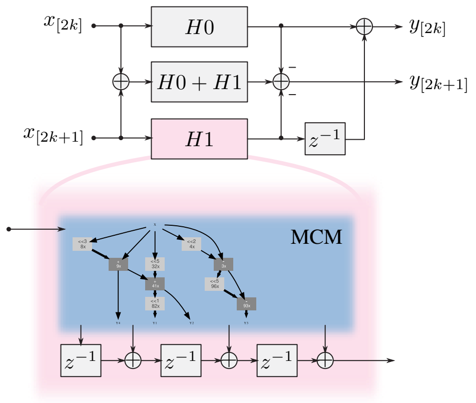


Fig. 7. Proposed 2-parallel filter with 8 coefficients

Although even the 2-parallel structure shows a multiplier saving of 25%, this increases with the level of parallelism. For a 2^p-parallel filter with constant subfilter length, and any natural number p, the polyphase structure needs a multiplier count proportional to 4^p. In contrast, our nested FFA structure needs a multiplier count proportional to only 3^p, although we acknowledge that the additional pre-adders and and their effect on wordlength will also contribute to the total circuit area. This scaling behaviour is shown in Figure 8 and the multiplier count is explored further in Section III. We consider the full circuit area (including pre-adders and their effect on wordlength) in Section IV.

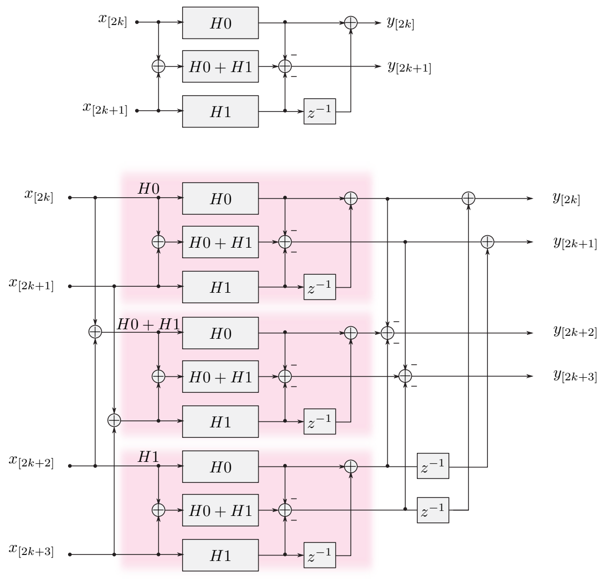


Fig. 8. Scaling of nested 2-parallel FFA filters for x2 → x4 parallelism

The main trade-off with this optimisation is FFA's ability to exploit coefficient symmetry. Each subfilter now has a unique input, precluding our shared MCM approach used with polyphase filters. So, although FFA appears extremely promising in the most general case, there is opportunity for a polyphase equivalent to perform favourably under real-world coefficient sets with symmetry or duplication. Section III offers an analysis of how each structure will perform under the six most common coefficient patterns.

III. Multiplier Counts Under Coefficient Symmetry

Although FFA reduces the number of multiplications in general, the pre-adders and $H_0 + H_1$ response can cause less favourable performance under coefficient symmetry. Since symmetry is prevalent in real-world impulse responses, we quantify the effect in Table I for 2^p-parallel filters with 2^pN coefficients, where N is the subfilter length. We consider common impulse response types, with zero padding to reach an integer multiple of 2^p, including:

Nonlinear phase:	No exploitable symmetry or antisymmetry in the coefficients
Type-II/IV:	Even coefficients with symmetry/antisymmetry
Padded Type-I/III:	Odd coefficients with symmetry/antisymmetry, then padded with one zero
Half-band:	Odd symmetric coefficients where every second value is zero except the centre value, padded with one zero

TABLE I
Multiplier count under symmetries for 2^p-parallelism and 2^pN coefficients

Structure	Impulse Response	Multiplications
Polyphase	*Nonlinear Phase*	4^pN
	Padded Type-I	$2^p\left\lceil\frac{2^pN}{2}\right\rceil$
	Padded Type-III	$2^p\left\lceil\frac{2^pN}{2}\right\rceil$
	Type-II	$2^p\left\lfloor\frac{2^pN}{2}\right\rfloor$
	Type-IV	$2^p\left\lfloor\frac{2^pN}{2}\right\rfloor$
	Half-band	$2^p\left\lceil\frac{2^pN}{4}\right\rceil$
FFA	*Nonlinear Phase*	3^pN
	Padded Type-I	$N(2+\sum_{k=1}^{p-1}3^k+2\sum_{i=0}^{p-2}\sum_{j=0}^{i}3^j)+(p-1)\left\lceil\frac{N}{2}\right\rceil$
	Padded Type-III	$N(2+\sum_{k=1}^{p-1}3^k+2\sum_{i=0}^{p-2}\sum_{j=0}^{i}3^j)+(p-1)\left\lfloor\frac{N}{2}\right\rfloor$
	Type-II	$\left\lceil\frac{N}{2}\right\rceil+2N\sum_{i=0}^{p-1}3^i$
	Type-IV	$\left\lfloor\frac{N}{2}\right\rfloor+2N\sum_{i=0}^{p-1}3^i$
	Half-band	$1+2^{p-1}+N+4N\sum_{i=0}^{p-2}3^i$

Although applications of nonlinear phase filters are uncommon, we include results for non-symmetric coefficients for two reasons. The FFA structure can exploit symmetry to a certain extent, but the analysis of this is non-trivial. We include nonlinear phase results to give a complete view of these patterns, since its multiplier cost is no longer just double that of types I→IV. Also, direct RF sampling *can* offer some demand for nonlinear phase filters; perhaps in digital predistortion for power amplifiers, or in instrumentation applications which are insensitive to phase distortion.

While the equations in Table I are useful for numerical evaluation of the algorithms, we appreciate that the visualisation in Figure 9 provides a clearer insight into the behaviour. The half-band analysis is for singe-rate filters only; the downsamling step as included in Section IV would introduce its own effects in FFA, varying with the level of parallelism.

In particular, note that the required number of multiplications for FFA is dramatically lower than polyphase for high levels of parallelism (x8 and x16). The extreme results for low levels of parallelism expose some subtleties in our consideration of real-world filter coefficients. For x2 parallel, type-II filters, FFA actually requires *more* multiplications than the simpler polyphase structure since the H_0 and H_1 responses break the symmetry in a worse-case manner. Here, the designer should opt to either adopt polyphase, or convert to a type-I impulse response.

These rules of thumb only regard the number of multiplications required in the filter structure and neglect any of the differences in additional adders, registers, and wordlengths which arise from the full filter implementation. The following section addresses these factors by presenting implementation results for each filter structure and impulse response type.

IV. Implementation Results

We present the implementation utilisation and timing results for a set of filters with 16 bit inputs and coefficients, using a set of realistic impulse responses. The implementation outputs and scripts to reproduce them are available at [11], with Vivado 2020.1 targeting the ZCU111 development board.

A. Utilisation Results

Figure 10 shows the Configurable Logic Block (CLB) and DSP48E2 usage for the FFA, polyphase with shared MCM-block subfilters, and SSR structures. Results are generated using out-of-context implementation. We sweep over parallelisms, number of coefficients, filter structures, and impulse response types. The SSR results are split into two resource types — one line for DSP48E2 usage (the systolic subfilter logic) and another for CLBs (likely for overheads in phase recombination). Our proposed MCM-based FFA and polyphase structures only use CLB resources, so the DSP lines are omitted.

As a general rule, our polyphase and FFA implementations have a percentage of CLB usage that is nearly bounded by the percentage of DSP usage of the traditional SSR implementation. From this, we can think of the proposed designs as a means of trading off a percentage of DSP usage for a similar or smaller percentage of the more general CLB fabric. A stronger assertion is that our total FFA CLB percentage tends towards approximately 50% of the traditional DSP percentage

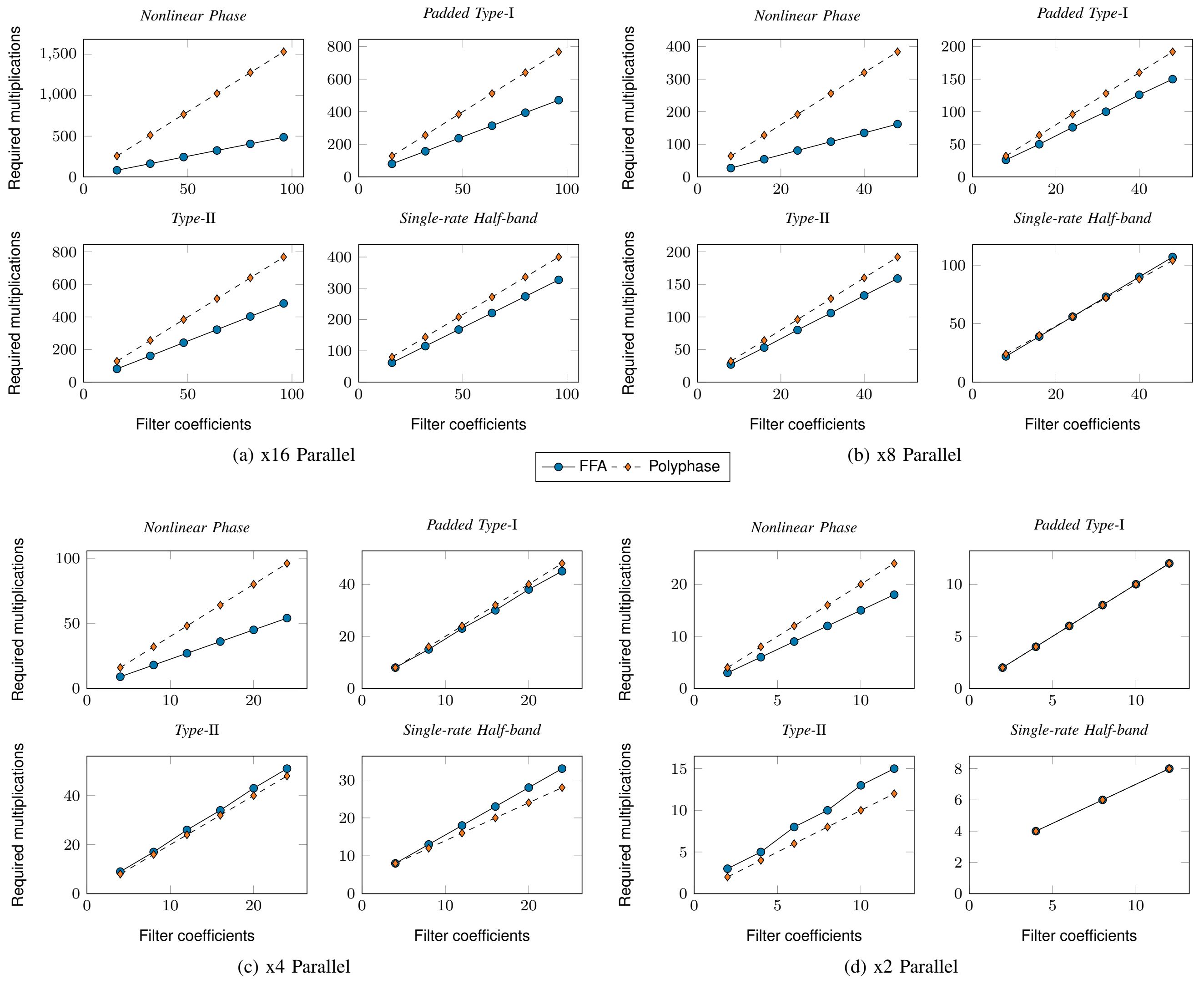


Fig. 9. Number of multiplications synthesised under symmetries

for nonlinear phase responses, 90% for type-I/II responses, and 80% for half-bands.

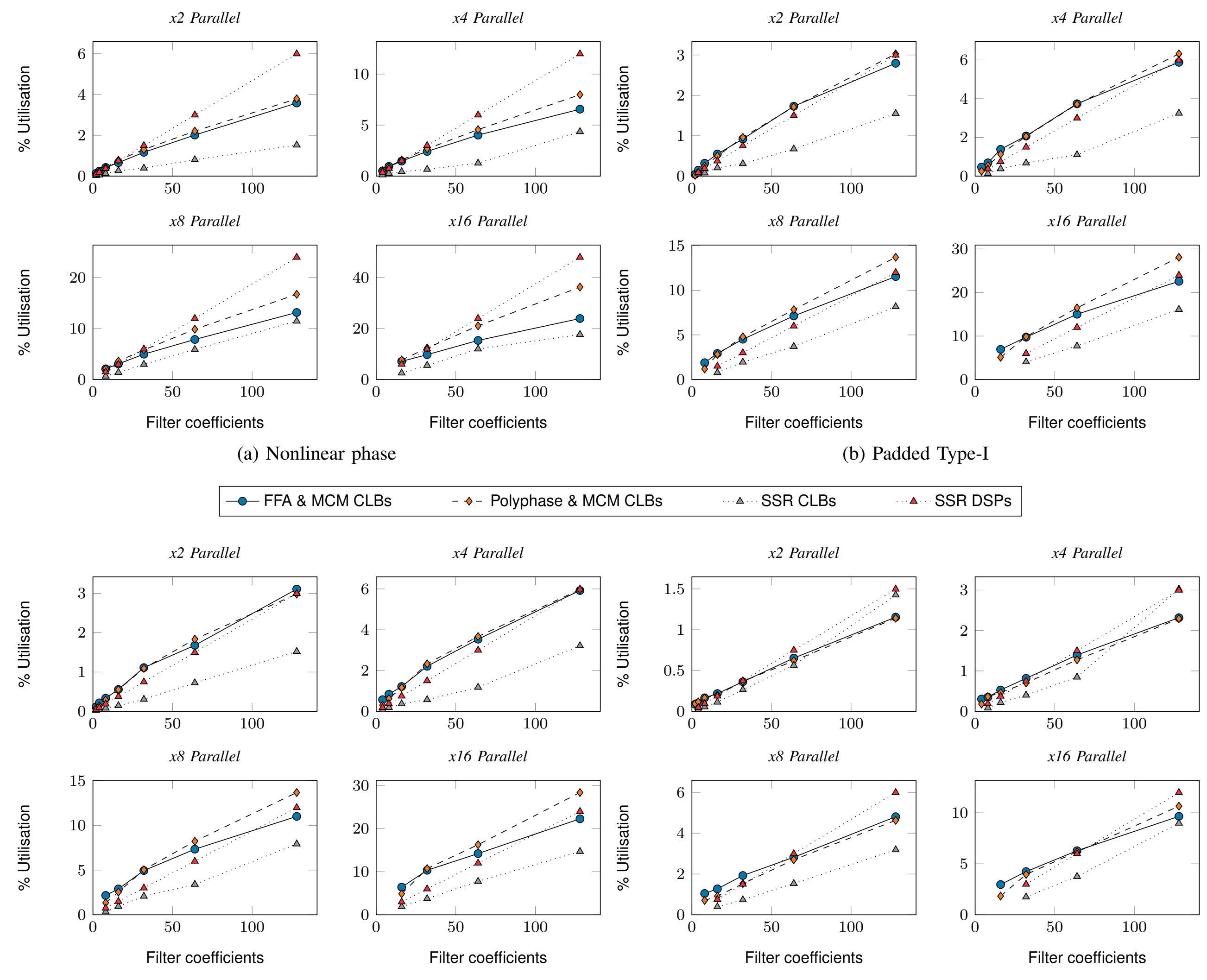
So far, we have neglected the CLB overhead incurred with the SSR implementation. The overhead is often comparable to the *total* CLB area required by our FFA implementation — especially for high levels of parallelism. Indeed, some of the extreme results for half-band filtering show that the FFA CLB area is actually smaller than just the CLB overhead incurred with SSR; not to mention the additional DSP usage!

The comparison between FFA and polyphase is more subtle. As predicted in Section III, FFA consistently outperforms polyphase for nonlinear phase impulse responses — trending towards 65% for x16 parallelism. For the remaining response types, the two architectures perform quite similarly for low levels of parallelism. For higher levels of parallelism, the FFA's advantages depend on the length of the subfilters. Small subfilters result in MCM blocks without much opportunity for resource sharing, limiting any optimisation. Since the polyphase structure shares larger MCM blocks between subfilters, the effects of resource sharing opportunity are less pronounced. In general, subfilter coefficient lengths of two or less are better suited to polyphase implementations, while FFA performs better for longer subfilters; tending towards 80% of the area for large x16 type-I/II filters.

B. Timing Results

We have estimated the maximum achievable clock frequency, $f_{\max}$, for each of the filter structures by implementing half-band decimators with $\times 8$ parallelism and various filter lengths. This is implemented as a small loopback design (no longer using out-of-context implementation) and $f_{\max}$ is taken as the fastest clock rate that meets timing over 6 iterations of a search, directed by the previous run's achieved timing estimate. Source code for this process is available at [11].

Figure 11 shows the results for our MCM-based FFA and polyphase filters, as well as the SSR architecture. This work



(a) Nonlinear phase

(b) Padded Type-I

(c) Type-II

(d) Half-band Decimator

Fig. 10. Implementation utilisation results

is conducted in the context of front-end digital filtering for the RFSoC and we should aim to support the ADC block's full data rate. Given the clocking resource's physical limit of 775 MHz [12] , the lowest possible level of parallelism we can use is $\times 8$ with a clock speed of ≥ 500 MHz. This target frequency is annotated as a red line in Figure 11 — any implementation above this line is sufficient for processing at the full ADC data rates.

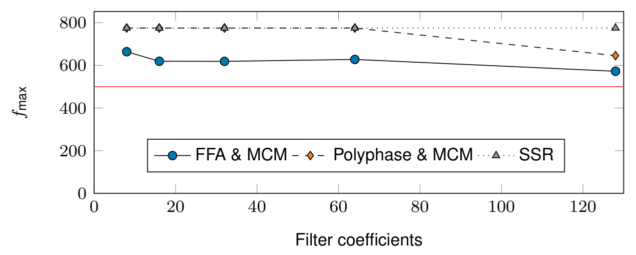


Fig. 11. Maximum frequency results for x8 half-band decimators

All three implementations remain above the 500 MHz target for every tested coefficient length. The SSR half-band decimators are the clear winner in terms of f_{max}, maintaining the physical maximum — but anything above our red line target is acceptable for all front-end applications. The MCM-based polyphase structure behaves comparably but does start to dip below the chip's maximum frequency between $64 \rightarrow 128$ coefficients, possibly due to high fan-out from our shared MCM blocks. Finally, the FFA architecture displays a similar trend with filter length but with smaller absolute frequencies. This reduction is due to our pipelining strategy attempting to better balance utilisation and f_{max}. The phase recombination step introduces critical paths including two adders between registers, but this can be easily further pipelined when timing closure is an issue.

V. Practical hardware description

This paper has, so far, only discussed the advantages offered by the FFA with MCM-based subfilters architecture. While we do offer some new analysis of how these two techniques interact in Section III, and publish open source implementations of both, the fundamental idea is simple — compose two existing, complementary algorithms from the literature. An interesting question to reflect on is: "Why has this (to the best of our knowledge) not been implemented already?".

We propose that the main factor is due to traditional hardware description languages and practices. Both FFA structures and MCM blocks have proven awkward to describe with traditional HDLs (such as VHDL and Verilog) for two different reasons:

- FFA lends itself to descriptions with structural recursion. This is not typically supported by traditional HDLs, perhaps because support for general recursion in a (structural) language would permit many non-synthesisable descriptions.
- MCM blocks can demand a huge amount of computation *at compile-time* in order to infer the circuit topology. The exact set of coefficient values will have a profound effect on the structure of the MCM block's shifts and adds — and this *must* be evaluated during compile-time, rather than during circuit run-time. This level of meta-programming is not found traditional HDLs beyond `for generate` statements directed by a range of scalars.

Historically, a designer can choose to either handcraft a circuit for one set of parameters, or turn to software programming in order to implement an ad hoc circuit generator for their algorithm of choice (see the implementation of H_{cub} [8] as a well regarded example). Implementations of the latter are used as atomic black boxes, producing specialised HDL output given a set of input parameters. This approach has known challenges, including being resistant to circuit verification techniques. These difficulties arise, in part, from the wide range of technologies at play with no unifying type checker or other assistance, as well as an effect similar to the required "semantic domain crossings" described in [13]. Our addition to this discussion is that ad hoc circuit generators also come with fundamental challenges for the *composition* of complementary techniques, discouraging demonstrably useful classes of circuit such as our FFA with MCM-based subfilters architecture.

Figure 12 visualises the main additional steps needed to compose two (hypothetical) ad hoc circuit generators. In prose, these steps include splicing new sections into each generator, working with the non-standard intermediate representation of each circuit, passing these representations between generators in a language agnostic way (shown in purple), and merging the HDL generation sections of the two codebases (shown in red).

We instead advocate for modern functional HDLs which are expressive enough to directly represent our algorithms and their staging — staging encodes *when* an expression should be evaluated (e.g. compile-time or circuit run-time).

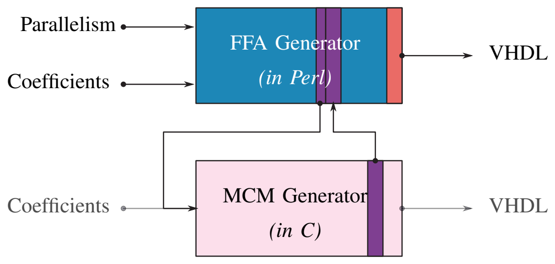


Fig. 12. Composing ad hoc circuit generators

Our implementation uses the language CλaSH, which offers these features under a single source language, type checker, simulator, and compiler. This massively simplifies verification efforts and enables the designer to have confidence in the final circuit output for *all* combinations of parameters, not just a select few test cases. Functional approaches like CλaSH directly encourage composition by allowing subcircuits to be passed to and from circuit generators. This allows exploration of the design space for many different algorithm combinations with little extra effort.

In reality, the implementation presented in this paper does lean heavily on Template Haskell [14] for meta-programming, including the staging and flattening of the structural recursion. This comes with a few practical difficulties such as a loss of type safety (historically, at least) and navigating the stage restrictions present in the Glasgow Haskell Compiler — the Haskell compiler on which CλaSH is based. CλaSH does facilitate our implementation with clear advantages over ad hoc circuit generators, but this work also has encouraged us to entertain two new language features for future work.

First of all, support for full spectrum dependent types can enable a host of quality of life improvements for the designer, including:

- Concise, type-safe minimum wordlength tracking, especially for DSP with constant coefficients. For readability, our CλaSH implementation manually resizes inputs to the worst-case wordlength and relies on EDA tools to prune uninhabited bits.
- Encoding proofs of circuit behaviour in the source language, wherever formal verification is demanded.

Our reliance on meta-programming techniques also highlights the need for multistage programming, preferably as a first class element of the source language. This enforces a clear split between compile-time computations and circuit run-time computation in a type safe way. Not only can this address some common criticism of Template Haskell, it may even alleviate the pressure on the compiler for unrolling primitive recursion and partial evaluation; we can write unrolling functions in the source language explicitly and evaluate them at compile-time. This is in opposition to CλaSH's set of templated VHDL code for recursively defined library functions. These ideas are explored further in [15]–[17] and could further encourage modern, verifiable DSP solutions for high-speed RFSoC applications and beyond.

VI. Related Work

Throughout this paper, the proposed MCM-based FFA filter architecture has been compared against the traditional polyphase structure with systolic subfilters exemplified by [7]. The comparison is particularly relevant since the proposed filter can be used as a direct, drop-in replacement for the traditional architecture. This compatibility is important due to the prevalence of System Generator/LogiCORE FIR compiler designs in the RFSoC's ecosystem; a designer can exploit the resource savings reported in this paper with minimal effort.

We acknowledge that filters, even for wide bandwidth signals, can be designed with a greatly reduced computational workload using Cascade Non-Maximally Decimated Filter Banks (NMDFBs) [18]. This employs multirate DSP techniques and time-sharing of DSP48E2s to reduce the workload, rather than optimising a direct FIR filter implementation. However, the NMDFB approach does burden the designer with the task of constructing a suitable bank of complementary filter responses. Our proposed designs instead offer resource savings via drop-in alternatives, without any extra design considerations.

VII. Conclusion

We have demonstrated that, in the context of RFSoC applications, a combination of FFA parallelism and MCM-based subfilters can generate area-efficient and high-speed parallel filters. These filters quite consistently exchange the traditional architecture's DSP usage for a similar percentage of the generic fabric resources (CLBs) — or for nonlinear phase filters, often approximately half of the equivalent DSP usage. This is ignoring the CLB "overhead" incurred by the traditional architecture as well — there are (somewhat extreme) scenarios where our *full* implementation has a smaller CLB area than the traditional implementation's *overhead* alone.

There are some interesting edge-cases for small filters with low parallelism where our polyphase structure with shared MCM blocks will often outperform an FFA equivalent, due to better exploitation of coefficient symmetry explored in Section III. Both implementations are presented here, and are available under open source licences.

Finally, we report on our practical experiences with the hardware description, identifying some limitations of traditional methods and how these discourage exploration of circuits with similarly complex structures. After identifying how modern functional HDLs, such as CλaSH, help alleviate some of these difficulties, we also propose language extensions which could improve the experience for future designers.

Acknowledgement

The authors would like to thank Xilinx for supplying EDA tools for this project and acknowledge funding for Craig Ramsay under EPSRC grant no. EP/N509760/1. All source code underpinning this publication is openly available at [11].

References

[1] Xilinx, Inc. (2018) UG897 — Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator (v2018.3). [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug897-vivado-sysgen-user.pdf

[2] P. Day, H. Leduc, B. Mazin, A. Vayonakis, and J. Zmuidzinas, "A broadband superconducting detector suitable for use in large arrays," *Nature*, vol. 425, pp. 817–21, October 2003.

[3] J. Pfau, S. P. D. Figuli, S. Bähr, and J. Becker, "Reconfigurable FPGA-based channelization using polyphase filter banks for quantum computing systems," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Cham: Springer International Publishing, 2018, pp. 615–626.

[4] 3rd Generation Partnership Project (3GPP). (2018) 5G; NR; Base Station (BS) radio transmission and reception, TS 38.104 version 15.2.0 Release 15. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138100_138199/138104/15.02.00_60/ts_138104v150200p.pdf

[5] Xilinx, Inc. (2020) PG 269 — Zynq UltraScale+ RFSoC RF Data Converter v2.3. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_3/pg269-rf-data-converter.pdf

[6] C. Baaij, "Digital circuit in cλash: functional specifications and type-directed synthesis," Ph.D. dissertation, University of Twente, Netherlands, 1 2015.

[7] Xilinx, Inc. (2015) PG149 — FIR Compiler v7.2 LogiCORE IP Product Guide. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/fir_compiler/v7_2/pg149-fir-compiler.pdf

[8] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, p. 11–es, May 2007. [Online]. Available: https://doi.org/10.1145/1240233.1240234

[9] K. N. MacPherson and R. W. Stewart, "Low FPGA area multiplier blocks for full parallel FIR filters," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, Dec 2004, pp. 247–254.

[10] D. Parker and K. Parhi, "Area-efficient parallel FIR digital filter implementations," in *Proceedings of International Conference on Application Specific Systems, Architectures and Processors: ASAP '96*, 1996, pp. 93–111.

[11] Craig Ramsay. (2021) Data for: "Low-cost, High-speed Parallel FIR Filters for RFSoC Front-Ends Enabled by CλaSH". [Online]. Available: https://doi.org/10.15129/a2c118f2-48a8-40d2-8896-89b9da71a4be

[12] Xilinx, Inc. (2021) DS 926 — Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds926-zynq-ultrascale-plus-rfsoc.pdf

[13] G. J. Smit, J. Kuper, and C. P. Baaij, "A mathematical approach towards hardware design," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

[14] T. Sheard and S. Peyton Jones, "Template meta-programming for haskell," in *Proceedings of the 2002 Haskell Workshop, Pittsburgh*, October 2002, pp. 1–16. [Online]. Available: https://www.microsoft.com/en-us/research/publication/template-meta-programming-for-haskell/

[15] E. Brady, J. McKinna, and K. Hammond, "Constructing correct circuits: Verification of functional aspects of hardware specifications with dependent types," in *Trends in Functional Programming*, vol. 8. United Kingdom: Intellect Books, 2008, pp. 159–176.

[16] C. Ramsay, L. H. Crockett, and R. W. Stewart, "On applications of dependent types to parameterised digital signal processing circuits," in *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2021, pp. 787–791.

[17] E. Brady and K. Hammond, "A verified staged interpreter is a verified compiler," in *Proceedings of the 5th International Conference on Generative Programming and Component Engineering*, ser. GPCE '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 111–120. [Online]. Available: https://doi.org/10.1145/1173706.1173724

[18] f. harris, E. Venosa, X. Chen, and C. Dick, "Cascade non-maximally decimated filter banks form efficient variable bandwidth filters for wideband digital transceivers," in *2017 22nd International Conference on Digital Signal Processing (DSP)*, 2017, pp. 1–5.