Integrating HW/SW Functionality for Flexible Wireless Radio

Alexander Strachan and Nigel Topham

School of Informatics, University of Edinburgh,
Edinburgh, Scotland, EH8 9AB

Abstract. Current methods of implementing wireless radio typically take one of two forms; either dedicated fixed-function hardware, or pure Software Defined Radio (SDR). Fixed function hardware is efficient, but being specific to each radio standard it lacks flexibility, whereas Software Defined Radio is highly flexible but requires powerful processors to meet real-time performance constraints.
This paper presents a hybrid hardware/software approach that aims to combine the flexibility of SDR with the efficiency of dedicated hardware solutions.
We evaluate this approach by simulating five variants of the IEEE 802.15.4 protocol, commonly known as Zigbee, and demonstrate the range of performance and power consumption characteristics for different accelerator and software configurations. Across the spectrum of configurations we see power consumption varies from 8% to 38% of a dedicated hardware implementation, and show how the hybrid approach allows a new modulation standard to be retrofitted to an existing design, with only a modest increase in power consumption.

Keywords: Wireless Radio, Digital Signal Processing, Embedded Systems, Computer Architecture, Accelerators.

1 Introduction

Wireless radio is a pervasive technology, which relies heavily on real-time data streaming and digital signal processors. However, architectures for wireless radio tend to be specialized, fixed-function devices designed to handle high data rates and complex mathematical processing.

Current embedded wireless radio designs operate across a spectrum and can be summarized at three extremes; fixed-function radio PHYs where all processing is done in dedicated hardware; software defined radio where all processing is done in software; and offload engines where a proportion of processing is performed in software before passing it to fixed-function hardware. The above methods all share the trait that once data is passed to fixed function hardware, that data is never seen again by the CPU. This has disadvantages; if the hardware does not perfectly fit the required algorithm it cannot be used, and if it cannot be disabled, then processing would need to be moved to software, decreasing efficiency.

The system architecture presented in this paper splits a hardware accelerator into a pipeline of computational blocks, each representing one stage of the radio processing pipeline, but allows the CPU to optionally intercept and manipulate data as it moves between pipeline stages. This provides flexibility for each part of the pipeline to be implemented in hardware or software, and for additional pipeline stages to be inserted as software components, as required by the modulation scheme. The IEEE 802.15.4 [1] standard, commonly referred to as Zigbee, is used in this paper as an example of a simple, industry standard radio specification used in IoT applications such as Thread [2], that contains several different modulation schemes.

David C. Wyld et al. (Eds): IoTE, CNDC, DSA, AIAA, NLPTA, DPPR - 2024
pp. 21-23, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.142302

1.1 Research Aims and Challenges

The primary motivation of this research is to explore how radio-processing systems can be made more flexible by sharing the workload of PHY processing between accelerator hardware and a general-purpose processor.

Key questions include; Can radio hardware designs be fixed/enhanced later on using software/CPUs? How does performance vary across the spectrum between pure hardware and software? What is the performance/power consumption impact on the system? Is there a cutoff point beyond which software cannot keep up? What phases of radio processing demand acceleration, and what phases can be adequately handled by software? How should data be streamed between hardware accelerator and software running on a CPU?

To answer these questions requires a simulation system capable of exploring the design space of hybrid software-hardware architecture solutions. This simulation system should accurately model real-world CPU and radio designs, by generating bit-accurate radio data streams, providing representative data on the performance of the system, and demonstrating the relative impact on power consumption, when hardware functions are handled in software.

2 Accelerator Architecture

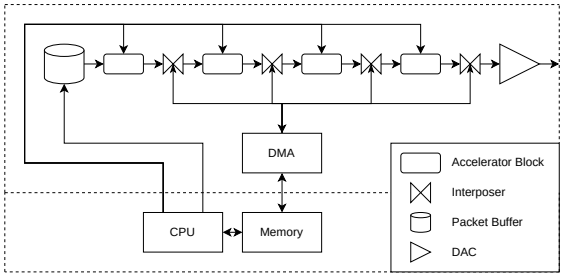

Fig. 1. Overall System Diagram showing accelerator and interposer connections to CPU and memory hierarchy.

Figure 1 shows the overall architecture of a system combining a general-purpose CPU with a radio-processing accelerator pipeline. This shows only the transmitter side, although the same approach can be applied to the receiver. The design comprises a packet buffer, where the data packets to be modulated are stored, a number of partially reconfigurable accelerator blocks that perform the radio signal processing, and a Digital to Analog Converter (DAC). However, such a design, while partially reconfigurable, is not flexible enough to support a hybrid system, that allows software to assist the processing. To allow this, devices called interposers are inserted between each stage of the accelerator pipeline. These interposers interface to the CPU via a DMA device, allowing the CPU to extract or insert data at any point in the accelerator pipeline. The CPU is then able to create flexibility by adding or replacing stages in the pipeline using software components.

2.1 Radio Accelerator Pipeline Design

The IEEE 802.15.4 standard [1] specifies many different Physical Layer (PHY) designs intended for different applications. For this work, six variations of this standard were

Table 1. IEEE 802.15.4 modulation types, data/symbol/sample rates (at four samples per symbol), for various specified frequency bands.

No.	Frequency Band	Modulation	Data Rate	Symbol Rate	Sample Rate[1]
1	2450 MHz	OQPSK	31250 B/s	62500 sym/s	4 MHz
2	915 MHz	OQPSK	31250 B/s	62500 sym/s	2 MHz
3	780 MHz	OQPSK	31250 B/s	62500 sym/s	2 MHz
4	868 MHz	BPSK	2500 B/s	20000 sym/s	1.2 MHz
5	915 MHz	BPSK	5000 B/s	40000 sym/s	2.4 MHz
6	920.8–928 MHz	GFSK	12500 B/s	100000 sym/s	400 kHz

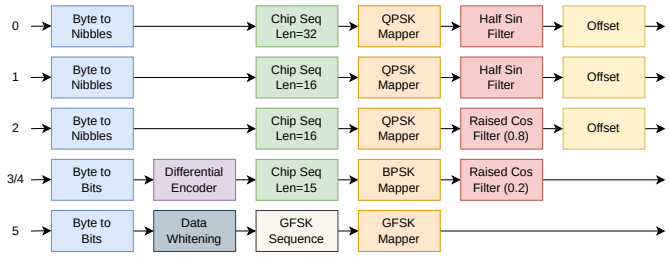

Fig. 2. Radio Accelerator Pipelines for The Chosen 802.15.4 Standards.

chosen that are similar in complexity and processing, and share mathematically common features. These can be seen in Table 1.

In Figure 2, block diagrams for each of the six standards are shown. It can be seen that these six standards share common operations or have operations that are a superset or a subset of operations in other standards. Therefore, a unified set of partially reconfigurable blocks was designed and then combined to form a single processing pipeline. This pipeline can be seen in Figure 3 and can be reconfigured to support any one of the six standards, either by disabling blocks that are not required, or by reconfiguring blocks to support a given radio standard. Descriptions explaining the operation of each block are shown in Table 2.

2.2 Interposer Design

To support a hybrid design, an interposer was designed which is inserted between blocks in the accelerator pipeline. A diagram showing the interposer design can be seen in Figure 4. It consists of; a set of double buffers, one set for each direction, allowing data to be filled and drained from the accelerator pipeline simultaneously with data transfer to the CPU; a pair of (de)multiplexers, allowing the interposer to be enabled or disabled, and logic to interrupt the CPU when there is a certain amount of data to be read or written to the interposer buffer. Data transfer to and from the interposers utilizes DMA, to handle the high data rates required.

Initially, the accelerator was connected to the CPU as an I/O device, but this led to unacceptable performance because the accelerator interface registers were treated as

[1] Using four samples per symbol

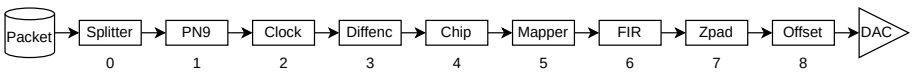

Fig. 3. Unified Reconfigurable Radio Accelerator Pipeline.

Table 2. Unified Accelerator Block Descriptions.

Block	Description
Splitter	Splits bytes into symbols (nibbles or bits).
PN9	XORs a data whitening sequence with symbols.
Clock	Generates (counter)clockwise series of symbols.
Diffenc	Differential encoder.
Chip	Maps symbols to orthogonal chip sequences.
Mapper	Maps symbols to complex IQ samples.
FIR	41-tap digital filter, used to shape samples.
Zpad	Inserts N zeros every M samples.
Offset	Delays imaginary (Q) component by N samples.

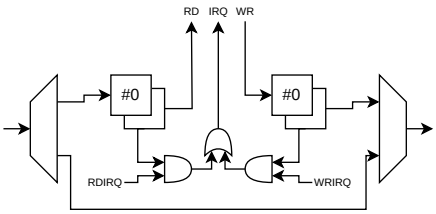

Fig. 4. Block Diagram Showing Interposer Design.

uncachable locations. This led to significant cache coherency overheads when moving data from cached data structures created by the CPU to the uncached accelerator I/O registers (and *vice versa*).

The introduction of DMA allowed burst transfers to be used to transfer data to and from the accelerator, but this still had unacceptable performance, as the mapped DMA memory still needed to be marked as uncachable. By using manual cache flush/invalidate operations on the DMA memory, these costs were reduced significantly, allowing slow memory transfer operations to be performed in longer bursts. This system allows a general purpose CPU to intervene and manipulate data from the accelerator pipeline, without unduly affecting the real-time characteristics of the system. Additionally, a ring buffer was added between the last block and the DAC to smooth out any non-determinism in the CPU response time.

2.3 Software Design

A software implementation of the Zigbee radio standards was written in `C++` to run under Linux as a userspace application. The Linux UIO system [3] is used to allow a userspace application to respond to CPU interrupts by blocking on a `read(2)` system call, allowing the CPU to sleep when no data transfer is required. Additionally, the u-dma-buf kernel module [4] allows a userspace application to `mmap(2)` memory suitable for DMA opera-

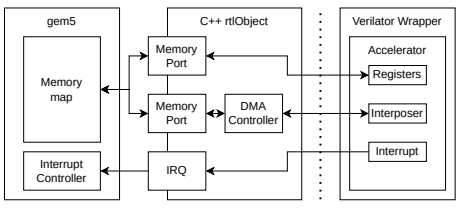

Fig. 5. Gem5 Simulation Architecture Overview.

tions. It also allows the program to manage cache coherency on this memory, allowing the userspace application to invalidate and flush the CPU cache.

Operation First, the CPU fills the packet buffer, configures each pipeline stage via memory mapped I/O and starts the accelerator. If the interposers are not in operation, there is nothing more for the CPU to do. The CPU then reads the accelerator status registers to determine if the enabled interposer is ready for reads or writes. If no interposer is ready, the `read(2)` system call is used on the UIO device, which causes the process to block, until the next interrupt occurs. When the CPU is unblocked, it again checks to see if the accelerator is waiting on a read or write. A read is accepted if the CPU currently has no pending data to write, then the size of the data is checked and the cache backing the DMA buffer is invalidated, by writing to a file exposed by the u-dma-buf module. If data is to be processed by software, this is done now, leaving an output buffer ready to be written when the interposer is ready. A write is accepted if the CPU has pending data to write. If the data to be written is larger than the currently set buffer size, it is broken up into chunks and the size of this data is then written to the accelerator. The data is then written to the mapped DMA buffer, and the cache is flushed, by writing to a file exposed by u-dma-buf. Then the DMA transfer is started, and various counters are updated, depending on how much data is left to be transferred. This whole process repeats, until a packet is read with the "last" flag set, indicating that this is the last data packet to be transferred. Once the last data packet is processed, the process sleeps until another interrupt occurs which signifies the accelerator as finished.

3 Results

To obtain experimental results for the above design, a simulator was designed based on gem5+rtl [5]. This approach allows cycle accurate hardware RTL designs to be simulated in conjunction with a simulated CPU model by integrating gem5 [6] and Verilator [7]. gem5 is used here as a generic model of an out-of-order CPU and its configuration is not changed between simulation runs. Therefore, any potential inaccuracies in gem5's micro-architectural model will apply equally to all results.

A block diagram of the simulator architecture can be seen in Figure 5. It consists of two modules, the accelerator design written in SystemVerilog, which is connected to a gem5 device written in C++. The Verilog module contains the accelerator blocks and interposers which are connected together via AXI-Stream [8] interfaces. The C++ module forwards memory and interrupt requests to and from the Verilog simulation and the CPU. As the RTL system does not provide a way to directly connect Verilog code to the gem5 memory hierarchy, the C++ module contains a simple DMA controller. The accelerator peripheral is connected to the CPU via the off-chip peripheral bus.

To evaluate the design, tests were constructed that simulated the transmission of a number of packets using the hybrid CPU/accelerator architecture. These tests allowed any contiguous segment of the nine accelerator blocks to be replaced with a software equivalent, enabling the behavior of the system to be analyzed when different parts of the system are replaced with software. After each accelerator run, the generated IQ data was written to a file to allow the accelerator output to be checked for correctness. This output was compared against reference models in MATLAB [9] and GNURadio [10].

3.1 CPU Power Consumption

To evaluate the impact on power consumption of using software to assist a hardware accelerator, simulations were run with each block in the hardware pipeline in turn being replaced with an equivalent software module running on the CPU. In these tests the CPU is idle when gem5 reports the CPU as clock-gated, and therefore the proportion of all cycles that are clock-gated is used as an indicator of the relative dynamic power consumption of the CPU. These experiments focused on the OQPSK modulation scheme operating at 2450MHz, as this has the highest data rate and thus presents a worse case scenario.

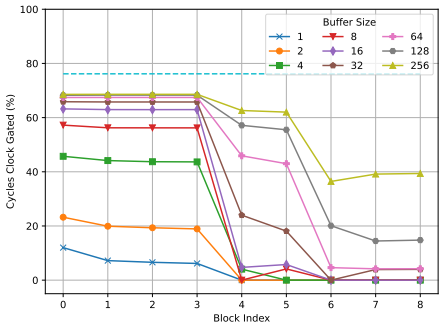

Fig. 6. Cycles Clock Gated Across Each Block Replaced with Software for Various Buffer Sizes

This test was performed for a range of different buffer sizes and results were compared against a full hardware implementation, as shown in Figure 6. It can be seen that for blocks appearing earlier in the pipeline, and with a large enough buffer size, the performance of this architecture can reach to within 10% of a full hardware implementation. This gap widens when blocks from further down the accelerator pipeline are replaced by software. This test was repeated with the software processing disabled and the hardware left enabled, to measure how much overhead the software signal processing has over the cost of data movement. It was seen that the software processing overhead was between ±6%. This shows that the primary cost of this system is the movement of data, not the complexity of the signal processing.

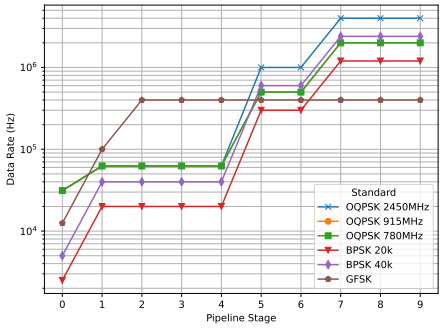

Fig. 7. Data Rate Across Accelerator Pipeline per Radio Standard

The above tests show the primary indicator of performance is the data rate that a block is operating at. Figure 7 shows the data rate increase across the pipeline for each

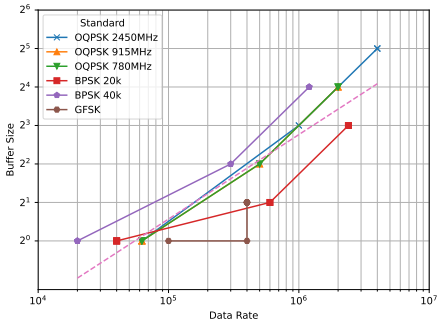

Fig. 8. Minimum Buffer Size for Data Rate Relation

Table 3. Block Usage per Modulation

	Splitter	PN9	Clock	Diffenc	chip mapper	FIR	zpad	offset	
OQPSK	✓	—	—	—	✓	✓	✓	✓	✓
BPSK	✓	—	—	✓	✓	✓	✓	—	—
GFSK	✓	✓	✓	—	—	✓	—	—	—

standard, it can be seen that this plots shape is similar to that of Figure 6. However, it was noted that certain combinations of block index and buffer size would cause the DAC to underrun, meaning that this combination is unsuitable for real-time applications. Figure 8 shows the minimum buffer size to not underrun, against data rate. Noting the log-log scale, a power relation can be seen, $Size \approx k \cdot rate^m$ where $m \approx 0.66, k \approx 0.0007$. This is a minimum bound, in a real system, a higher buffer size may be required.

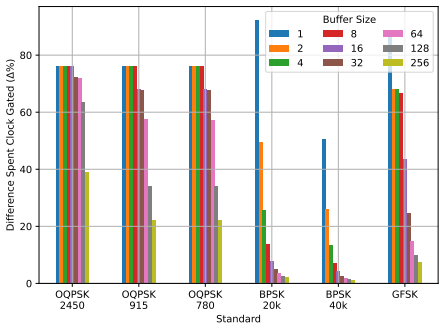

Fig. 9. Cycles Spent Clock Gated Across Each Standard with Unique Blocks Removed for Various Buffer Sizes

It can be seen in Table 3, that certain blocks in the accelerator pipeline are only used by one standard. This opens up the possibility of simulating a system where these blocks were not part of the accelerator when it was manufactured. These "missing" blocks could then be added using software instead, allowing a radio standard to be added "after the fact". Therefore, an experiment was run where these blocks were disabled and replaced with software and these results were compared with a full hardware implementation of that standard, for a range of buffer sizes. It can be seen in Figure 9 that in the best case (BPSK), with a large enough buffer size, power consumption can be within 3% of a full hardware implementation, but in the worst case (OQPSK 2450) power consumption

is 40% worse than hardware. This difference is due to OQPSKs unique blocks occurring later in the pipeline, and thus they operate at a higher data rate.

3.2 CPU Behaviour

In the prior section it was seen that the performance impact of this architecture is strongly linked with the movement of data to and from the accelerator. To further explore the CPU behaviour during these periods, the test program was modified to add marks at each phase of the programs execution. These phases are: `Init` where the system is configured; `Loop` where decisions are made on what operation to perform; `IRQ` where if the CPU has no work to do, it sleeps; `Read` is where data is read from the accelerator; `DSP` where data processing is performed; `Write` is where data is written to the accelerator; and `End` where the CPU is finished processing and is sleeping waiting for the accelerator to complete. The tests were run for each standard at a buffer size of 256.

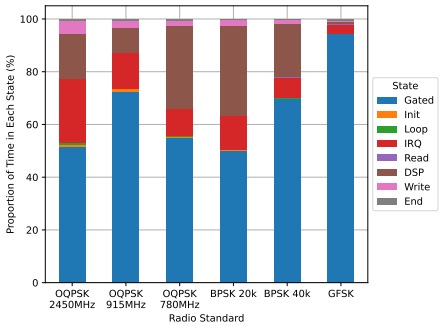

Fig. 10. Proportion of CPU Cycles Spent in Each State per Modulation Type

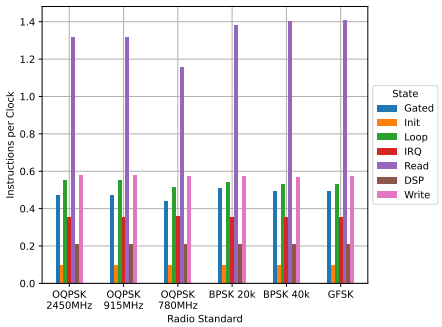

Fig. 11. Instructions per Clock in Each State per Modulation Type

In Figure 10 the time spent in each phase is shown. It can be seen that the CPU spends the majority of its time clock gated, leading to reduced power consumption. The time the CPU spends active is primarily spent in the DSP, IRQ, Read and Write stages. Across all standards the average IPC is 0.25, showing poor instruction level parallelism. In Figure 11, the IPC in each phase is shown, showing that while the DSP takes up the majority of the time, it is by far the most efficient phase. Whereby the other phases show IPC values well below 0.6. These results show that the time spent in the IRQ, read and write stages are

inefficient enough to bring down the overall IPC by a factor of 5, despite them taking less overall time and being less mathematically complex than the signal processing.

To investigate potential causes of the low IPC, the cycles spent and state of the CPU pipeline was measured for each phase of the program. In Figure 12 a breakdown of the cycles spent in the Read Write and DSP phases can be seen, averaged across all standards. It can be seen that the most computationally intensive signal processing block is the FIR filter and that most of the time spent reading and writing is on cache management operations, relating to the setup of the DMA unit.

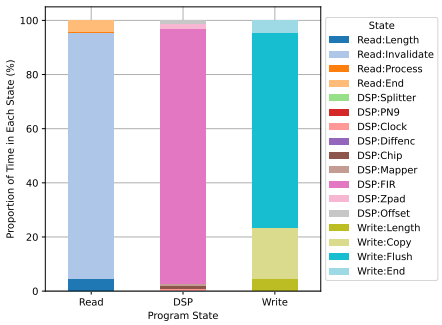

Fig. 12. Breakdown of CPU Cycles Spent in Read DSP and Write States

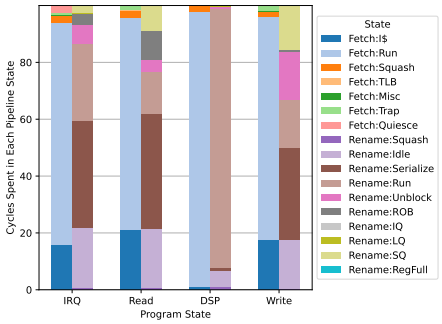

Fig. 13. Breakdown of Cycles Spent in Fetch and Rename Stages in Read DSP and Write States

In Figure 13 a breakdown of the CPU fetch and rename pipeline states is shown for the IRQ, Read, Write and DSP stages, averaged across each standard. The IRQ state spends a significant amount of time stalled in the fetch state, due to instruction cache misses, caused by the system call overhead, as the kernel manages the interrupts. On average the interrupt latency is 7494 cycles. The read and write stages spend a similar proportion of time also stalled on the instruction cache, also due to the cache management operations requiring system calls. Additionally, the cache management operations, and memory reads/writes stall the CPU in the rename stage, due to the need to serialize data and the load/store queues and ROB becoming full. Therefore, it can be seen that the cause of the low IPC is the large overhead of moving memory between the CPU and the accelerator and management of the CPU caches.

4 Conclusion

In conclusion, this work shows that software can be used to enhance a hardware accelerator pipeline, and can work effectively with a real-world modulation standard, in real-time. A relation was discovered that can be used to calculate the minimum buffer size required for software to intervene at a given data rate. The extra power consumed by this design varies considerably depending on the data rate at the which the CPU is required to intervene. However, is it hoped that this overhead can be reduced in future designs. Finally, a co-simulation approach was used that allowed cycle-accurate simulation of the hardware accelerator, so it could be integrated in a SoC design with minimal effort.

However, communication between the CPU and interposers has a large overhead due to the interposer and CPU having opposing views on memory layout. The CPU requires a cache hierarchy, where data is stored at various non-local levels to hide DRAM latency; while the accelerator requires contiguous data in a single location to take advantage of DMA burst transfers. This results in a significant amount of time being spent of cache management operations, that cause the CPU to stall, due to lack of space in the ROB and load/store queues. Cache management and interrupt handling also require system calls that cause high instruction cache miss rates, also stalling the CPU.

A limitation with this approach, is that certain radio standards may have areas in their processing pipeline which operate at too high a data rate for a CPU to replace. Additionally, a potential drawback of this system is that the overall performance of the SoC would be reduced because one of the CPU cores would be tied up performing signal processing. If this system can be used to add or enhance a standard, it will come at the cost of increased power consumption, with the tradeoff being that, the SoC will not need to be redesigned or replaced, reducing development time and waste. A more detailed comparison of this system, against existing hardware, is planned using a future improved version of the system described in this paper.

This overhead results in the CPU reporting low IPC values, due to poor instruction-level parallelism, leading to more clock cycles spent active, increasing power consumption. While the proportion of CPU time spent on digital signal processing is large, the IPC value during this period is high, showing that the CPU is operating efficiently during this time. However, it is unclear if the low efficiency of the cache management operations are due to the operation itself, or the cost of the system calls required to perform these operations.

4.1 Related Work

Research This work builds upon the gem5+RTL framework [5], which allows collection of detailed CPU behavior information, while using an RTL model for the accelerator design, combining the advantages of both types of simulation.

Similar work in reconfigurable radios for Zigbee have been considered in [11, 12] but neither of these papers combine similar blocks for different modulation types. In papers [13–15] this combining of similar processing blocks is performed for the Wi-Fi and 3G protocols.

The RFNoC [16] project looks into taking advantage of the FPGAs that are present in many SDR interfaces, in the GNURadio framework, to alleviate the CPU from the most demanding processing. However, the use of high-level software and FPGAs preclude use of this in embedded systems.

Additionally, while pure software solutions may work for simpler standards such as Zigbee, other more complicated standards like Wi-Fi require parts of the PHY and MAC to have hardware support to meet timing deadlines [17–22]. This shows that some sort

of hardware acceleration can be required, even if power consumption is not taken into account.

However, none of this work considers a combined hybrid approach of these ideas, placing the CPU "in the loop" between accelerator blocks, in a way that could work in an embedded system context.

Industry The ADSP-SC589 from Analog Devices [23] is a processor, consisting of a standard ARM Cortex-A5 core and two SHARC Digital Signal Processors. These three processors are connected over a shared memory bus which is shared with various fixed-function DSP accelerators. Therefore, this system is very flexible with both the CPU and DSP being fully independent and programmable, allowing for data to be passed between the CPU and DSPs in various ways. However, this device is not designed for radio applications, it is designed for automotive and entertainment applications. While Analog Devices do manufacture components designed for radio applications, they are primarily front-ends and offer little flexibility.

The Qualcomm Hexagon DSP architecture [24], used in their Snapdragon processors, are split into two types; the aDSP, used for multimedia applications and the mDSP, used for the modem. The Hexagon is a DSP that has been extended to be more general purpose, and can run a hypervisor that can run a full RTOS or Linux kernel. This allows significant flexibility, as DSPs usually do not permit running of high level operating systems. The Hexagon is connected to the memory bus to communicate with the processor. This system works "the other way" compared to the hybrid system described in this paper. It adds general purpose execution to a DSP, as opposed to adding DSP acceleration to a general purpose processor.

4.2 Further Work

In this work only a single core CPU was used, so the CPU can only take part in one contiguous section of signal processing. So, could multithreading be utilized to replace multiple accelerator blocks, at different points in the pipeline, or used for multiple parallel processing pipelines? Additionally, the accelerator is connected to the CPU via a standard off-chip I/O bus. So would moving the accelerator closer to the CPU improve the data movement overhead, reducing latency? For example in this paper [25] a neural network accelerator was connected directly to an x86 CPUs ring bus, improving latency.

In the system described in this paper, when the CPU is woken up by an interrupt for reads, the cache has to be invalidated before processing can be started. Could this work be performed before the CPU is woken up?, reducing the time the CPU is active. An example of this is Intel's Direct Cache Access technology [26–28] which allows for a PCIe device to DMA data into the CPU cache, reducing cache misses, when the CPU is woken up. Furthermore, cache and interrupt management operations require system calls, which cause instruction cache misses that stall the CPU. Can this overhead be reduced by exposing these operations more effectively to userspace? For example, in this paper [29] instructions were added to reduce the communication overhead between the CPU and an external accelerator. Finally, while the CPU works efficiently during signal processing, it still takes up a large proportion of time. So could new ISA extensions be used to reduce the time spent on signal processing? For example in this paper [30] complex number DSP instructions were added to a RISC-V processor.

Finally, can this work be applied to other data streaming applications outside wireless radio that have real-time requirements?

Acknowledgment

The authors would like to thank UKRI/EPSRC and Keysight Technologies for their financial support. UKRI project reference: 2590731

References

1. IEEE, "Ieee standard for low-rate wireless networks," *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pp. 1–800, 2020.
2. T. Group, "Thread," https://www.threadgroup.org/, 2023, [Online; accessed 16-October-2023].
3. H.-J. Koch, "The userspace i/o howto," https://www.kernel.org/doc/html/v4.12/driver-api/uio-howto.html, 2006, [Online; accessed 16-October-2023].
4. K. Ichiro, "u-dma-buf(user space mappable dma buffer," https://github.com/ikwzm/udmabuf, 2023, [Online; accessed 16-October-2023].
5. G. López-Paradís, A. Armejach, and M. Moretó, "Gem5 + rtl: A framework to enable rtl models inside a full-system simulator," in *Proceedings of the 50th International Conference on Parallel Processing*, ser. ICPP '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3472456.3472461
6. N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, aug 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718
7. Veripool, "Verilator," https://www.veripool.org/verilator/, 2023, [Online; accessed 16-October-2023].
8. ARM, "Amba 4 axi4-stream protocol specification," https://developer.arm.com/documentation/ihi0051/a/, 2023, [Online; accessed 16-October-2023].
9. MathWorks, "Zigbee toolbox," https://uk.mathworks.com/help/comm/zigbee.html, 2023, [Online; accessed 16-October-2023].
10. B. Bloessl, "Ieee 802.15.4 zigbee transceiver," https://github.com/bastibl/gr-ieee802-15-4, 2023, [Online; accessed 16-October-2023].
11. M. A. Zubair, A. K. Nain, J. Bandaru, P. Rajalakshmi, and U. Desai, "Reconfigurable dual mode ieee 802.15.4 digital baseband receiver for diverse iot applications," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, pp. 389–394.
12. A. Massouri and T. Risset, "FPGA-based Implementation of Multiple PHY Layers of IEEE 802.15.4 Targeting SDR Platform," in *SDR-WInnComm*. Schaumburg, Illinois, United States: Wireless Innovation Forum, Apr. 2014. [Online]. Available: https://inria.hal.science/hal-01005317
13. A. Sadek, H. Mostafa, A. Nassar, and Y. Ismail, "Towards the implementation of multi-band multi-standard software-defined radio using dynamic partial reconfiguration," *International Journal of Communication Systems*, vol. 30, no. 17, p. e3342, 2017, e3342 IJCS-16-0762.R1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3342
14. A. Kamaleldin, S. Hosny, K. Mohamed, M. Gamal, A. Hussien, E. Elnader, A. Shalash, A. M. Obeid, Y. Ismail, and H. Mostafa, "A reconfigurable hardware platform implementation for software defined radio using dynamic partial reconfiguration on xilinx zynq fpga," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 1540–1543.
15. S. Hosny, E. Elnader, M. Gamal, A. Hussien, and H. Mostafa, "Multi-partitioned software defined radio transceiver based on dynamic partial reconfiguration," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–4.
16. M. Braun, J. Pendlum, and M. Ettus, "Rfnoc: Rf network-on-chip," *Proceedings of the GNU Radio Conference*, vol. 1, no. 1, 2016. [Online]. Available: https://pubs.gnuradio.org/index.php/grcon/article/view/3
17. K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, "Sora: High-performance software radio using general-purpose multi-core processors," *Commun. ACM*, vol. 54, no. 1, p. 99–107, jan 2011. [Online]. Available: https://doi.org/10.1145/1866739.1866760
18. G. Stewart, M. Gowda, G. Mainland, B. Radunovic, and D. Vytiniotis, "Ziria: Wireless programming for hardware dummies," Microsoft Research, Tech. Rep. MSR-TR-2013-135, November 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/ziria-wireless-programming-for-hardware-dummies/
19. G. Stewart, M. Gowda, G. Mainland, B. Radunovic, D. Vytiniotis, and C. L. Agulló, "Ziria: A dsl for wireless systems programming," in *ASPLOS*. ACM - Association for Computing Machinery, March 2015. [Online]. Available: https://www.microsoft.com/en-us/research/publication/ziria-a-dsl-for-wireless-systems-programming/

20. H. Wu, T. Wang, Z. Yuan, C. Peng, Z. Li, Z. Tan, B. Ding, X. Li, Y. Li, J. Liu, and S. Lu, "The tick programmable low-latency sdr system," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 101–113. [Online]. Available: https://doi.org/10.1145/3117811.3117834
21. K. Kang, Z. Zhu, D. Liu, W. Zhang, and H. Qian, "A software defined open wi-fi platform," *China Communications*, vol. 14, no. 7, pp. 1–15, 2017.
22. B. Ding, J. Liu, H. Wu, and T. Wang, "Gplm: An 802.11ac-capable low-mac architecture for fpga-based sdr systems," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–7.
23. A. Devices, "Sharc+ dual-core dsp with arm cortex-a5," https://www.analog.com/media/en/technical-documentation/data-sheets/adsp-sc582_583_584_587_589_adsp-21583_584_587.pdf, 2023, [Online; accessed 9-Mar-2023].
24. L. Codrescu, W. Anderson, S. Venkumanhanti, M. Zeng, E. Plondke, C. Koob, A. Ingle, C. Tabony, and R. Maule, "Hexagon dsp: An architecture optimized for mobile multimedia and communications," *IEEE Micro*, vol. 34, no. 2, pp. 34–43, 2014.
25. G. Henry, P. Palangpour, M. Thomson, J. S. Gardner, B. Arden, J. Donahue, K. Houck, J. Johnson, K. O'Brien, S. Petersen, B. Seroussi, and T. Walker, "High-performance deep-learning coprocessor integrated into x86 soc with server-class cpus industrial product," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 15–26.
26. A. Farshin, A. Roozbeh, G. Q. M. Jr., and D. Kostić, "Reexamining direct cache access to optimize I/O intensive applications for multi-hundred-gigabit networks," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 673–689. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/farshin
27. R. Huggahalli, R. Iyer, and S. Tetrick, "Direct cache access for high bandwidth network i/o," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 50–59.
28. M. Wang, M. Xu, and J. Wu, "Understanding i/o direct cache access performance for end host networking," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 1, feb 2022. [Online]. Available: https://doi.org/10.1145/3508042
29. L. Morais, V. Silva, A. Goldman, C. Alvarez, J. Bosch, M. Frank, and G. Araujo, "Adding tightly-integrated task scheduling acceleration to a risc-v multi-core processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 861–872. [Online]. Available: https://doi.org/10.1145/3352460.3358271
30. H. B. Amor, C. Bernier, and Z. Přikryl, "A risc-v isa extension for ultra-low power iot wireless signal processing," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 766–778, 2022.