# Software-Hardware Co-Design of Multi-Standard Digital Baseband Processor for IoT

Hela Amor, Carolynn Bernier

# Software-Hardware Co-Design of Multi-Standard Digital Baseband Processor for IoT

Hela Belhadj Amor and Carolynn Bernier
*IC Design, Architectures and Embedded Software Department*
*Univ. Grenoble Alpes, CEA, LETI*
38000 Grenoble, France
carolynn.bernier@cea.fr

***Abstract*—This work demonstrates an ultra-low power, software-defined wireless transceiver designed for IoT applications using an open-source 32-bit RISC-V core. The key driver behind this success is an optimized hardware/software partitioning of the receiver's digital signal processing operators. We benchmarked our architecture on an algorithm for the detection of FSK-modulated frames using a RISC-V compatible core and ARM Cortex-M series processors. We use only standard compilation tools and no assembly-level optimizations. Our results show that Bluetooth LE frames can be detected with an estimated peak core power consumption of 1.6 mW on a 28 nm FDSOI technology, and falling to less than 0.6 mW (on average) during symbol demodulation. This is achieved at nominal voltage. Compared to state of the art, our work offers a power efficient alternative to the design of dedicated baseband processors for ultra-low power software-defined radios with a low software complexity.**

***Index Terms*—IoT, LPWA, software-defined radio, RISC-V**

## I. Introduction

To enable massive Internet of Things (IoT) applications, many new wireless signaling schemes, and especially those targeting long-range, wide area (LPWA) networks, have recently been proposed. While focusing on similar applications, these new signaling schemes are considerably different. For example, ultra-narrow band signaling based on differential binary phase shift keying (DBPSK) and Gaussian frequency shift keying (GFSK) modulations is exploited in the Sigfox [1] protocol. Alternatively, chirp sequence spread spectrum (CSSS) modulation is employed by the LoRa physical layer [2]. These new systems however share similar characteristics from an RF/analog perspective. For instance, many LPWA networks operate below 1 GHz and require a relatively small analog bandwidth (less than 2 MHz). Thus, a common RF/analog front-end could easily be used to build a multi-standard transceiver, provided that the digital front-end (DFE) and digital baseband (DBB) elements of the transceiver are sufficiently versatile to address the specific requirements of each signaling scheme. Faced with the uncertain evolution of LPWA networks and standards, such a multi-standard solution could minimize development cost for multiple solutions, enable multi-mode applications and future-proof designs.

A software-defined LPWA transceiver is therefore highly desirable. But developing a programmable architecture that is compatible with the ultra-low power (ULP) consumption required by LPWA applications is a major challenge. Luckily, both CMOS scaling and the current trend towards heterogeneous, multi-core IoT platforms can help to address this challenge [3]–[5]. In addition, recent research has proposed specialized architectures to address this challenge [6], [7]. But since developing a dedicated processor architecture is costly and resource-intensive, the present work instead focuses on a different approach based on existing and widely available 32-bit scalar processor architectures with single-cycle multiplication but no floating-point unit (FPU). While not originally designed with wireless digital signal processing (DSP) in mind, we show that these processors can provide an attractive solution to this end. We start by proposing an architecture that partitions the hardware and software digital processing elements of a wireless receiver. Then we compare the performance of a benchmark application on two ARM-based processors and an open-source RISC-V compatible core (RI5CY) [9]. Finally, a power estimation based on a Bluetooth LE application is used to prove the feasibility of our approach.

## II. Related Work

For the first time, an SDR processor architecture that achieves mW-level power consumption was presented in [6]. To achieve this result, the authors claim that the most computationally intense kernels in a digital receiver are vectorial in nature and that the datapath of an IoT receiver can be reduced to very small bit widths (4 or 8 bits) without noticeably sacrificing performance. The authors therefore propose an architecture based on a custom Single Instruction Multiple Data (SIMD) unit associated with a scalar unit. The focus is on reducing the SDR processor's working frequency in order to employ deep voltage scaling. Unfortunately, the choice of vectorial processing for wireless DSP is questionable for two reasons. While many of the required algorithms are indeed vectorial in nature (e.g. FIR filters), those that operate on undecimated sample streams are clearly better implemented using reconfigurable hardware in the DFE. Secondly, limiting the signal stream to 8 bits results in quantification errors that can impact sensitivity and limit the tolerance to other imperfections of the wireless transceiver.

A different approach is proposed in [7] which presents a baseband processor architecture based on 32-bit scalar processing. Application-specific instructions are introduced to

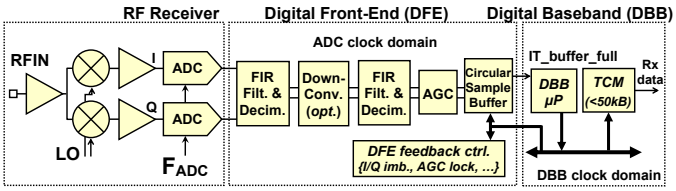


Fig. 1. Simplified architecture of software-defined receiver

reduce power in register files through instruction chaining. While running a frame detection algorithm, the authors state that these instructions reduce the number of required instructions from 10 to 4. We observe that, again, the focus is on reducing the processor clock frequency. Finally, in both of these works, we note that the choice of designing a dedicated machine implies that algorithms must be hand-coded in machine specific assembly code.

## III. Software-defined ULP Receiver Architecture

A key question to answer when building a programmable platform is which parts of the transceiver's DSP chain should be performed in hardware (HW) versus software (SW). Indeed, practical power consumption considerations limit the use of software DSP to sample streams that have been sufficiently decimated from the ADCs' high sampling rate. Fortunately, narrow-band wireless transmission schemes typically have similar DSP requirements in the *digital front-end* (DFE). Limiting the discussion to the receiver, which is computationally more demanding, the DFE is in charge of removing the intermediate frequency (IF) if any, and performing partial or complete channel filtering and automatic gain control (AGC). The logical choice therefore consists in implementing the DFE DSP in configurable hardware which can easily accommodate the differing requirements of IoT wireless standards.

A simplified software-defined receiver architecture is shown in Fig. 1. Complex samples are produced at a constant sample rate by the DFE and are temporarily stored in a circular sample buffer. Once this buffer contains at least BLOCK_SIZE samples, it triggers a "buffer full" interrupt. This wakes up the DBB processor which then reads BLOCK_SIZE samples starting from its last read address and processes them. During this time, the DFE is still continuously producing and storing samples in the buffer which must be sized to prevent over-writing of unread samples. Real-time processing of the incoming samples is necessary since the exact arrival time of a frame is unknown. Also, it is common for the DBB algorithm to send control signals to the DFE which requires a quick response time from the DBB.

Figure 2 shows a common characteristic of wireless frames which are composed of at least three distinct parts. The *Preamble* contains symbols that allow the receiver to detect the frame's presence and recover the symbol timing information


Fig. 2. DBB Frame processing steps

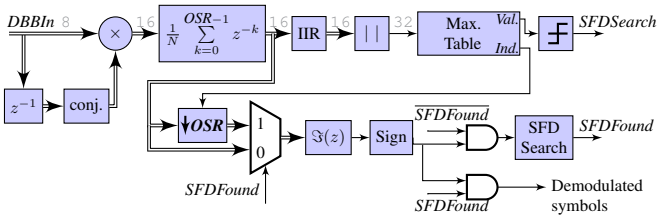


Fig. 3. Simplified view of FSK demodulation algorithm. Double arrows indicate complex signals. The signal dynamic range is shown in lighter type.

from the noisy signal. If a preamble is detected, the receiver starts looking for the *Start of Frame Delimitor (SFD)* which contains symbols that allow the receiver to synchronize to the beginning of the frame's useful data. Once it is found, the receiver can start demodulating the data symbols. As a rule, the computational complexity is highest in the "SFD Search" phase since, in order to avoid false detection, most receivers continue scanning for a preamble while searching for the SFD. The complexity of the "Demodulation" phase is the lowest since the recovered symbol clock can be used to further decimate the sample stream. To take advantage of the variations of complexity over time, the DBB processor must be able to put itself in deep sleep, ideally in a single cycle, while waiting for the next block of samples to be available in the sample buffer. To further minimize power, the ROM section of the DBB processor's TCM (Fig. 1) should be implemented using either low-retention-current SRAM or using a high-speed embedded non volatile technology. In this way, the transceiver's voltage supply can be switched off between successive frame transmissions.

## IV. Software FSK Demodulation

In this section, we focus on a specific example of a DBB receiver algorithm for an FSK-based signaling scheme. Because FSK is very common in IoT applications, this algorithm will serve as a good benchmark for the complexity comparison presented in section V. The detection and demodulation of FSK-modulated frames can be achieved by an algorithm similar to that presented in Fig. 3 [8]. While a dynamic range of only a few bits for $DBBIn[k]$ may be sufficient in ideal conditions, in practice, a few more bits are typically required, with 8 bits a safe maximum. We also assume that the signal is over-sampled at an over-sampling ratio (*OSR*) of 6 times the symbol rate (*OSR*'s between 2 and 8 are common, with higher *OSR* values leading to greater detection performance).

The DBB algorithm starts with a Hermitian product, defined as $DBBIn[k] \times DBBIn[k-n]$ with $n$ the delay between two samples. With a delay of 1, the imaginary part of the Hermitian product is approximately proportional to the instantaneous frequency. To eliminate a bit of noise, the signal is then averaged over *OSR* consecutive samples and then scaled down by 3 bits ($N = 8$) to conserve a 16-bit dynamic range. Next, the IIR filter $y[k] = x[k] - 0.875\,y[k - OSR]$ is used to correlate the signal with the known preamble made up of alternating symbols (e.g. "01010101..."). The absolute value is then calculated and the greatest of these values over *OSR* consecutive samples is stored, along with its index, in a table.

If one of these stored values exceeds a user-defined threshold, the *SFDSearch* signal is set to TRUE, with TRUE=1 and FALSE=0. If this occurs (before a timeout), symbols are demodulated by taking the sign of the imaginary part of the signal and the search for the pre-defined SFD bit sequence is enabled. If found, the *SFDFound* signal is set to TRUE. This shuts off both the preamble processing steps and the SFD search and activates the decimation of the input signal in time-alignment with the recovered symbol clock. A simplified version of the implemented algorithm is given in Algorithm 1.

A bare-metal C version of this algorithm is implemented in order to extract the computational complexity of the three phases of frame reception, noted as *Synch*, *SFD* and *Demod* in Table I. Other phases of frame reception, such as de-interleaving, decoding, CRC computation, etc. are not implemented since they do not have real-time constraints. In order to execute the code on available hardware, we first load the samples corresponding to a noisy frame signal into the processor's memory and increment a pointer to this array by BLOCK_SIZE to emulate the reading of the sample buffer. In this study, we target 32-bit machines with single-cycle multiplication but no FPU. In order to avoid wasting precious CPU cycles to check for overflows, preliminary Matlab simulations of the fixed-point DBB algorithm were used to evaluate if bit scaling is necessary at the output of mathematical operators. The result of this study can be seen in Fig. 3 where the dynamic range increases less quickly than could be expected. In general, we observe that, with an 8-bit input, the algorithm, with its succession of multiplications, fits well on a 32-bit machine. This is a general characteristic of DBB processing algorithms, where complex multiplications are very common. Our implementation uses no assembly level optimization (except when investigating the RI5CY DSP built-in extensions). We have deliberately chosen not to optimize the C-source code and to let the compilers perform the optimizations they are capable of. Optimizing the source code, would, of course, further improve the results. The only software optimization that was implemented consisted in choosing BLOCK_SIZE $= 6$ and manually unrolling the program's main "for" loop which, of course, increases the image size slightly but reduces complexity by avoiding the modulo operator as well as certain "if-then-else" constructs.

## V. Complexity Comparison Results

In this section, an identical version of the software-loop-unrolled DBB application code is compiled and executed on several widely-available platforms: ARM Cortex M0+, M3, and RI5CY. RI5CY is an open-source RISC-V compatible core with hardware and DSP extensions [9]. The ARMCLANG compiler and KEIL uVision5 simulator were used to simulate the ARM cores. RTL-level simulations where used to simulate the RI5CY core. The average number of CPU cycles required to process a single complex sample, which is obtained by dividing the number of cycles required to process a block of complex samples by BLOCK_SIZE, is presented for the three phases of frame detection and for each core (Table I).

TABLE I
Per Target Complexity of FSK DBB Algorithm, BLOCK_SIZE=6

| Cpu Target | Average # of CPU cycles per complex sample | | | Memory (Bytes) | |
|---|---|---|---|---|---|
| | *Synch* | *SFD* | *Demod* | *ROM* | *Stack* |
| M0+[a] | 70 | 93 | 41 | 2468 | 376 |
| M3[a] | 69 | 89 | 42 | 2096 | 308 |
| RI5CY Basic[b] | 45 | 57 | 21 | 3052 | 287 |
| RI5CY + Ext.[c] | 45 | 56 | 19 | 2816 | 271 |
| RI5CY + built-ins.[c] | 50 | 61 | 27 | 2868 | 239 |

[a]ARMCLANG V6.7, Flags: "-Ofast"
[b]GCC-5.3.0, Flags: "-Ofast -funroll-all-loops"
[c]GCC-5.2.0, Flags: "-march=IMXpulpv2 -Ofast -funroll-all-loops"

As expected, the algorithm complexity is greatest in the SFD search phase.

First, we observe that with this code, the performance of the M0+ and M3, other than image size, is similar. To explain this unexpected result, we investigated the memory accesses for these target and observed that the M0+ performs an average of 50% more accesses compared to the M3. We then studied the IPC (instructions per clock) of both processors during the *Demod* phase and find that the M3 IPC is 0.65 (due to a larger number of multi-cycle instructions) while the M0+ IPC is equal to 1. This explains the similarity of M3 and M0+ results in table I in spite of considerably different memory accesses. Knowing that the M3 comes with a factor of 3 power consumption overhead with respect to the M0+ [10], we conclude that the M0+ is a better option for our application.

Next, the code was compiled for the RI5CY core first using the generic RISC-V GCC compiler and then using the "-march" option that is specific to this core. In the first two cases, results are similar, only two cycles being saved in the *Demod* phase due to the reduction in the number of branches. We observe that, while RI5CY's HW loop and post increment extensions cannot be used by our algorithm, RI5CY's p.mac, p.msu and immediate branching instructions help to decrease image size slightly. Finally, the application code is

**Algorithm 1** FSK DBB implementation pseudocode

```
OSR = 6;
for ( index = 0 ; index < BLOCK_SIZE ; index++ ) do
  Calculate Hermitian product and average
  if SFDFound == TRUE then
    if index % OSR == maxindex then
      Demodulate
    end if
  else
    IIR filter and Max. of Magnitude over OSR samples
    if MaxValue > Threshold then
      SFDSearch = TRUE;
      maxindex = index;
      if index % OSR == maxindex then
        Demodulate and search for SFD
      end if
    end if
  end if
end for
```

modified to exploit RI5CY's built-in vectorial instructions. The dotsp4 built-in function is used to calculate the 8-bit complex multiplication of the Hermitian product. Recall that $z3 = z1 \times z2^*$ is found using: $Re(z3) = ac + bd$ and $Im(z3) = bc - ad$, where $z1 = a + ib$ and $z2 = c + id$. To find $Re(z3)$, we load vector V1 (resp. vector V2) with the real and the imaginary parts of $z1$ (resp. $z2$) and then use the built-in function (_builtin_pulp_dotsp4(V1,V2)), which executes simultaneously four byte operations on a 32-bit word. Next, to find $Im(z3)$, V2 must be re-ordered and a sign change applied before the dotp operation can be executed. These data manipulations explain the poor results obtained (Table I). In conclusion, RI5CY's hardware extensions bring little benefit in our application.

Generaly, we observe that the RISC-V core (basic) improves performance by more than 35% with respect to ARM-based architectures. To understand the reasons behind this performance gain, we carefully analyzed specific code segments and found that, while the M3 uses a more complex DSP-related instruction set, since these instructions require more than one execution cycle, in the end, the same number of cycles are required for mathematical calculations as on the RI5CY core. The analysis also revealed the benefit of the BNE instruction (branch on comparison of two register values) that is available in the RISC-V ISA. Finally, because of its limited number of registers, the M3 needs to store a greater number of intermediate results whereas the RI5CY core, with 32 registers, performs typically 50% fewer memory accesses compared to the M3. In turn, this will have a proportionally positive impact on the TCDM power consumption.

## VI. DSP Processor Power Consumption Estimation

Using the results presented in Table I and the power consumption of the RI5CY core [9], it is possible to estimate the power consumption of the core that would execute our DBB algorithm for the Bluetooth Low Energy (LE) physical layer employing the GFSK modulation at a 1 Msymbols/s signaling rate [11]. Extracting the worse-case complexity of 57 cycles/sample from Table I (RI5CY basic configuration) and assuming that the *OSR* can be reduced to 2 samples/symbol (the resulting impact on sensitivity being less of an issue in short-range IoT applications), we find that a minimum clock frequency of 114 MHz is required.

The basic RI5CY core presented in [9] is designed for a 2.8-ns cycle time (357 MHz) and drains 26.28 $\mu$W/MHz at 1.08 V in 65 nm CMOS. If we apply the 1.9x expected power gain observed when moving from 65 nm to 28 nm UTBB FD-SOI (as in [9]), we estimate that the core's power consumption, while running at 114 MHz, would be on the order of 1.6 mW during the *SFD Search* phase, i.e. the shortest phase of frame detection, and falling to less than half of this (on average) during symbol demodulation. Since these estimations are given for a supply voltage of 1.08 V, we observe that further power savings could be obtained using voltage scaling. Of course, the above power estimation is incomplete without the added power drain of memory and interconnect.

TABLE II
Comparison of present work with State of the Art

| | [6] | [7] | This work |
|---|---|---|---|
| Physical layer | Bluetooth-LE GFSK 1-Ms/s | OFDMA & CSSS | Bluetooth-LE GFSK 1-Ms/s |
| Processor type | Dedicated | Dedicated | Generic |
| Architecture | Vectorial | Scalar | Scalar |
| Compilation Tools | no | no | standard open-source |
| Technology | CMOS 28nm | CMOS 65nm | FDSOI 28nm |
| Dyn. Power [mW] | 1.372 (peak) | 7.2 (ave.) | 1.6[a] (peak) |
| Frequency [MHz] | 20 | 3 | 114 |
| OSR | 4 | - | 2 |
| Core area [mm$^2$] | 0.074 [b] | 0.204 | 0.068[c] [9] |
| Voltage scaling | yes (0.45 V) | no | no |

[a] Estimated.
[b] The complete system requires an additional MCU.
[c] This is for the RISC-V basic design in 65 nm CMOS. According to the authors, the design occupies 46.9 kGE [9].

The comparison with prior art, Table II, shows that our work offers a power and area efficient alternative to the design of dedicated baseband processors. Indeed, by choosing approaches that lower frequency and therefore voltage, authors in [6] and [7] accept surface overheads which, in the end, result in increased power consumption.

## Conclusion

In this work, we show that, thanks to an optimized hardware/software partitioning of the receiver's digital signal processing operators, an ultra-low power software-defined radio for IoT applications can be implemented on generic 32-bit architectures. Bluetooth LE frame detection can be achieved at a peak power consumption of 1.6 mW assuming a basic RISC-V compatible core in 28 nm FDSOI technology, and falling to less than 0.6 mW (on average) during symbol demodulation. Our work offers a power and area efficient alternative to the design of dedicated baseband processors.

## References


[1] www.sigfox.com
[2] SEMTECH, SX1272/73 Datasheet. [Online] Available: http://www.semtech.com/images/datasheet/sx1272.pdf
[3] Texas Instruments, CC1310 Datasheet. [Online] http://www.ti.com/product/cc1310/description
[4] F. Conti, D. Palossi, A. Marongiu, D. Rossi and L. Benini, "Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms," in DATE Conference Proceedings, pp. 1201-1206, 2016.
[5] Y. Pu, et al., "A 9-mm$^2$ Ultra-Low-Power Highly Integrated 28-nm CMOS SoC for Internet of Things," IEEE J. of Solid-State Circuits, vol. 53, no. 3, pp. 936-948, March 2018.
[6] Y. Chen, et al., "A low power software-defined-radio baseband processor for the Internet of Things," in Proceedings of the IEEE International Symposium on HPCA, pp. 40-51, 2016.
[7] S. Wu, S. Kang, C. Chakrabarti and H. Lee,"Low power baseband processor for IoT terminals with long range wireless communications," in Proceedings of the IEEE GlobalSIP Conference, pp. 728-732, 2016.
[8] D. Lachartre, et al., "7.5 A TCXO-less 100Hz-minimum-bandwidth transceiver for ultra-narrow-band sub-GHz IoT cellular networks," in Proceedings of the IEEE ISSCC Conference, pp. 134-135, 2017.
[9] M. Gautschi, et al., "Near-Threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 25, No. 10, pp. 2700-2713, 2017.
[10] [Online] https://developer.arm.com/products/processors/cortex-m
[11] Bluetooth Core Specification version 5.0, Bluetooth SIG, 2016, [Online] https://www.bluetooth.com/specifications/bluetooth-core-specification