International Journal of Natural Sciences: Current and Future Research Trends (IJNSCFRT)
ISSN: 2790-7929
https://ijnscfrtjournal.isrra.org/index.php/Natural_Sciences_Journal/index

Parallel Data Processing Solutions for Software Defined Radio

Daniel Shannon[a]*, Mo Abdallah[b]

[a,b]Syracuse University, Syracuse, NY
[a]Email: djshanno@syr.edu
[b]Email: maabdall@syr.edu

Abstract

Taking into consideration the decisions behind the current day processors used for Software Defined Radio (SDR) and digital signal processing, we will examine how current machine learning chips, such as Tensor Processing Units or their components including Systolic Arrays and Matrix Multiplier Units, can be incorporated into signal processing today. There are several architectural approaches to implementing digital signal processing with SDR, ranging from Field Programmable Gate Arrays to General Purpose Processors and Digital Signal Processors. SDR has historically been developed to move algorithms and signal processing away from hardware that is built for a specific purpose to software that can accommodate new algorithms and processes with ease. Recently, there have been advances in leveraging data level parallelism for SDR, including the use of Graphics Processor Units. After consideration of current and previous SDR technology, we propose a hypothetical Dynamic Band Processing Unit X (DBPUX) for processing multiple bands per cycle in a Digitize at Intermediate Frequency (DIF) SDR architecture using a Matrix Multiplier Unit (MMU) and a Systolic-Type Array for specific signal processing tasks. Further work is needed to simulate and apply performance metrics to the proposed processor.

Keywords: Software Defined Radio; Parallelism; Digital Signal Processing; Tensor Processing Unit; Systolic-Type Array.

1. Introduction

Traditional radios use fixed hardware components for specific tasks like modulation, filtering, and frequency tuning that come in the form of analog circuitry and Application Specific Integrated Circuits (ASIC) [1,2,3].

Received: 3/15/2024
Accepted: 5/12/2024
Published: 5/25/2024
----

* Corresponding author.

These hardware components are specialized for the task at hand, and updating the capabilities of the radio requires updating the hardware, which can be expensive. Software Defined Radio (SDR) can offload many of the hardware's functions to a processor for digital signal processing. Processors for SDR come in a variety of formats, depending on constraints of the problem at hand. SDR generally implies a need for real-time signal processing for monitoring or communication, so processors must be fast enough to handle the load of real time radio signals. Radio communication and SDR can be energy intensive, so the ability to process signals at low energy is always welcome.

SDR processors come in many forms with different motivations and compromises across flexibility and cost. Digital signals are generally processed on Field Programmable Gate Arrays (FPGA), Digital Signal Processors (DSP), General Purpose Processors (GPP), Graphics Processing Units (GPU), or some combination of processors. Analog hardware and ASIC solutions are the most efficient radios in terms of power consumption and processing speed, but they lack flexibility and are expensive to design. Each type of processor has strengths and weaknesses with respect to SDR, requiring a balance of power, size, price, and processing speed and function. Generally, SDR implies the need for real-time processing of signals for radio communication, so the processor must be fast enough to handle real time signals from the I/O [2,3].

1.1 Background and Previous Work

An examination of what SDR is and how it differs from Analogue Radio with dedicated circuits is introduced. After understanding what SDR is and what is relevant to both the software and radio aspects of SDR architecture, we examine different processors being used in SDR today, including FPGAs, DSPs, GPPs, and GPUs.

With a thorough understanding of the existing processors available for SDR, we begin to examine the Tensor Processing Unit (TPU) a highly parallel and energy efficient processor for solving Neural Networks in machine learning domains. We take a closer look at the MMU and the Systolic-Type array that enables much of the processor's energy efficiency and parallelism. As we examine the TPU and its approach to parallelism, we begin to consider how we can use design aspects of the TPU in signal processing for a hybrid DSP capable of processing multiple radio bands in one instruction cycle.

A careful searching for patterns and trends in current processors used in SDR is done and also identifying components and design aspects that can be used from each architecture is presented. while reviewing the processors, looking for highly parallel, low power, low latency solutions is the goal to accommodate the real-time processing of multiple bands in parallel.

1.2 Software Defined Radio Overview

The complexity of a radio depends on the expected functions of the radio itself, and where the software in SDR is implemented can vary depending on the resources and reasons for using SDR. SDR implementations can be grouped into 5 categories:

- DSP at Audio Frequencies (DAF) (Figure 1).
- Downconversion to Analog Baseband (DAB) (Figure 2).
- Digitizing at Intermediate Frequency (DIF) (Figure 3).
- Direct Radio Frequency Digitizing (DRF) (Figure 4).
- Hybrid solutions are combinations of the above.

DAF allows signals to be modulated and filtered in tone, which enables digital modulation using analog transceivers. DAB passes the signal through a Low Pass Filter (LPF) and Analog-to-Digital Converter (ADC) to a DSP, which can be a GPP with a sound card. Downconversion also enables a lower sample rate, allowing for low-pass filter decimation and for more CPU cycles to process each sample. These two SDR architectures rely heavily on analog radios and implement SDR at the start of transmitting or the end of receiving [3].

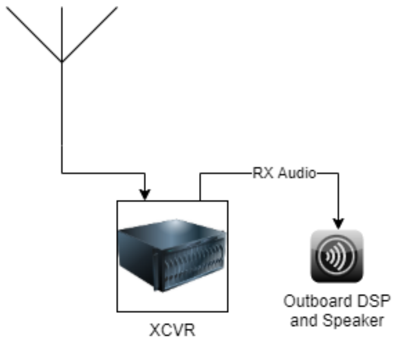


Figure 1: An outboard processor that outputs the processed radio as sound to a DSP [3].

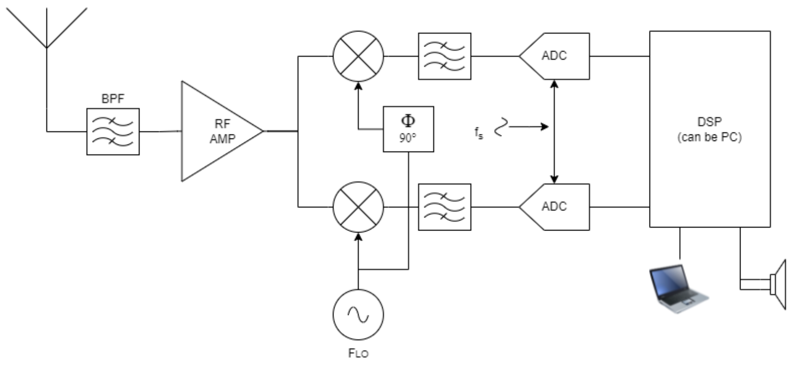


Figure 2: Direct-to-baseband SDR architecture does a downconversion to a baseband allowing for lower sample rate due to aliasing [3].

DIF signal processing architecture places the ADC after the signal has passed through a crystal IF filter whose purpose is to improve the range of blocked interfering signals. Figure 3 shows the block diagram of DIF signal processing. The IF crystal improves the blocking dynamic range, which enables a lower ADC sample rate as the crystal acts as a narrow bandwidth anti-aliasing filter. This narrow bandwidth signal is then sent to the DSP for further signal processing [3].

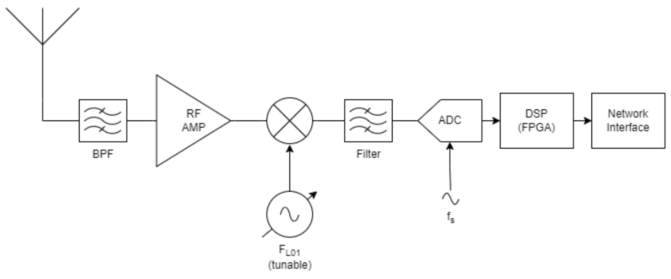


Figure 3: High-IF sampling allows frequency tuning before the ADC to the DSP [3].

The most complete SDR architecture is DRF, which passes the least processed radio signal to the processor when compared to DAF or DAB architecture. The DRF architecture passes the signal through a Band Pass Filter (BPF) and RF Amp before ADC and DSP. This shifts most of the radio work to the DSP. DRF and DIF DSPs can both be implemented as dedicated DSPs or FPGAs. In some applications, GPPs and GPUs can be used for processing signals in DIF and DRF SDR architectures [3].

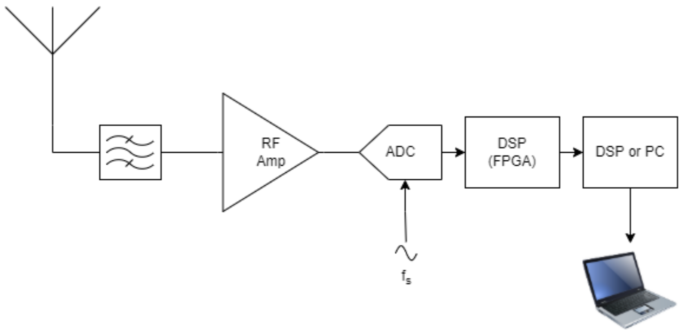


Figure 4: Direct RF-sampling applies an RF Amp and band pass filter before ADC [3].

1.2.1 Processor Overview

SDR can be defined as an implementation of the hardware components of analog radio systems in software that can be updated without updating the hardware. The primary advantage of SDR is flexibility. Given that there are several components of a radio, including the antennae, tuner, amplifier, filters, that can be implemented in software, SDR systems require compromise and balance to achieve the speed and efficiency needed for real-time radio communication. Size, weight, area, and power consumption are important factors to consider when designing SDE solutions [4].

1.2.2 Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) are widely used as the digital signal processing (DSP) core in SDR architectures such as Digitizing at Intermediate Frequency (DIF) and Direct RF Digitizing (DRFD). These reprogrammable chips consist of configurable logic blocks that can function as memory, arithmetic units, or control logic. FPGAs are particularly valuable in research and prototyping environments due to their reconfigurability, allowing engineers to rapidly iterate on signal processing algorithms and hardware architectures without fabricating custom chips.

While FPGAs typically consume more power and occupy more silicon area than analog radios or Application-Specific Integrated Circuits (ASICs), their flexibility often offsets these costs — particularly in early-stage or evolving SDR systems. As shown in Akeela and Dezfouli [2], FPGAs outperformed 16-core GPPs in double-precision General Matrix Multiply benchmarks. Their performance, measured by the aggregate of Look-Up Tables (LUTs), flip-flops, and DSP slices scaled by clock frequency, indicates their suitability for compute-intensive, parallelizable tasks such as modulation, demodulation, and filtering.

The performance for the GPPs $\mathbf{perf_{GPP}}$ was determined by multiplying the number of floating-point functional units $\mathbf{FU_{fp}}$ by the number of cores $\mathbf{n_{cores}}$ and the clock (frequency) $\mathbf{f_{clock}}$ (eq. 1). The performance of the FPGAs $\mathbf{perf_{FPGA}}$ is measured by summing the number of Look Up Tables (LUT), flip-flops, and DSP slices, and then multiplying that by the clock frequency (eq. 2).

$$perf_{GPP} = FU_{fp} \times n_{cores} \times f_{clock} \tag{1}$$

$$perf_{FPGA} = (LUT + flip - flops + DSPslices) \times fclock \tag{2}$$

Several papers discuss FPGAs and DSPs as interchangeable components. The beauty of FPGAs is that the hardware is reconfigurable, so everything discussed in the next section on dedicated DSPs can be implemented in FPGAs. This allows for the initial hardware development to be on FPGAs and then create a dedicated DSP chip for the final product if desired.

1.2.3 Digital Signal Processor

Digital Signal Processors (DSPs) are specialized microprocessors optimized for executing signal processing tasks efficiently. Like General Purpose Processors (GPPs), DSPs include core components such as registers, caches, memory, buses, Arithmetic Logic Units (ALUs), and program counters. However, DSPs are architected specifically to exploit the parallelism inherent in signal processing, particularly through features like the Parallel Logic Unit (PLU) and Multiplier-Accumulator (MAC), as seen in the TMS320 DSP line from Texas Instruments [3,5].

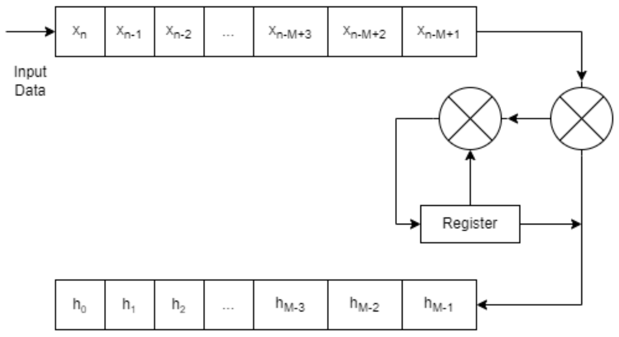


Figure 5: A Multiple and Accumulator convolution applied to a single register [5].

When we examine dedicated DSP systems, we can see several similarities and differences compared to GPPs. Modern GPPs store data and instructions together in memory, implementing the Von Neumann architecture. DSPs generally implement modified Harvard or Very Long Instruction Word (VLIW) architectures [5]. These architectures allow for the data and instructions to be accessed in parallel. If there are two separate buses between memory and data, the number of instructions per cycle is two. The number of instructions per cycle can be increased with additional buses. Harvard architecture separates the instruction and data in memory, while Von Neumann architecture combines data and instruction into one memory structure. VLIW architecture accesses the instruction from memory and specifies the data path for each available functional unit. VLIW can be categorized as a Single Instruction Multiple Data (SIMD) architecture [3]. This means that one instruction can be applied to multiple data streams at one time. This can increase performance and decrease power, but these improvements depend on the parallel nature of the task at hand. Some processes are still sequential and the gains from VLIW can be lost in those situations. The other type of architecture is Multiple Instructions Multiple Data (MIMD), which allows separate instructions to operate on each data stream. This enables a higher-level of parallelism across different memory blocks [3,5].

Several prior studies have explored the trade-offs of using DSPs in SDR systems. For example, Oppenheim and Schafer [5] highlight that the benefits of VLIW and SIMD architecture in DSPs can be lost when algorithms contain significant sequential dependencies. In contrast, our proposed DBPUX attempts to isolate sequential elements through a multiplexer while offloading parallel computation to an MMU, aiming to retain the benefits of SIMD without bottlenecks caused by instruction dependencies.

Chips such as the Texas Instruments TMS320C5x (Figure 6), TMS320C54x and Motorola DSP563x operate on fixed point arithmetic, which is useful for consistent, energy efficient processes that operate in a limited range. Texas Instruments TMS320C4x and TMS320C67xx processors use floating point arithmetic. Fixed point arithmetic makes sense for signal processing where we repeat operations often and know the range we are working in. Additional buses can be added to memory for increased parallelism. Multiple Access Memory through Dual-Access RAM (DARAM) and Multi-ported Memory increase the number of memory accesses/cycle. The memory access efficiency increases with the number of data buses connecting the memory to the processor for DARAM, but the hardware costs increase too.

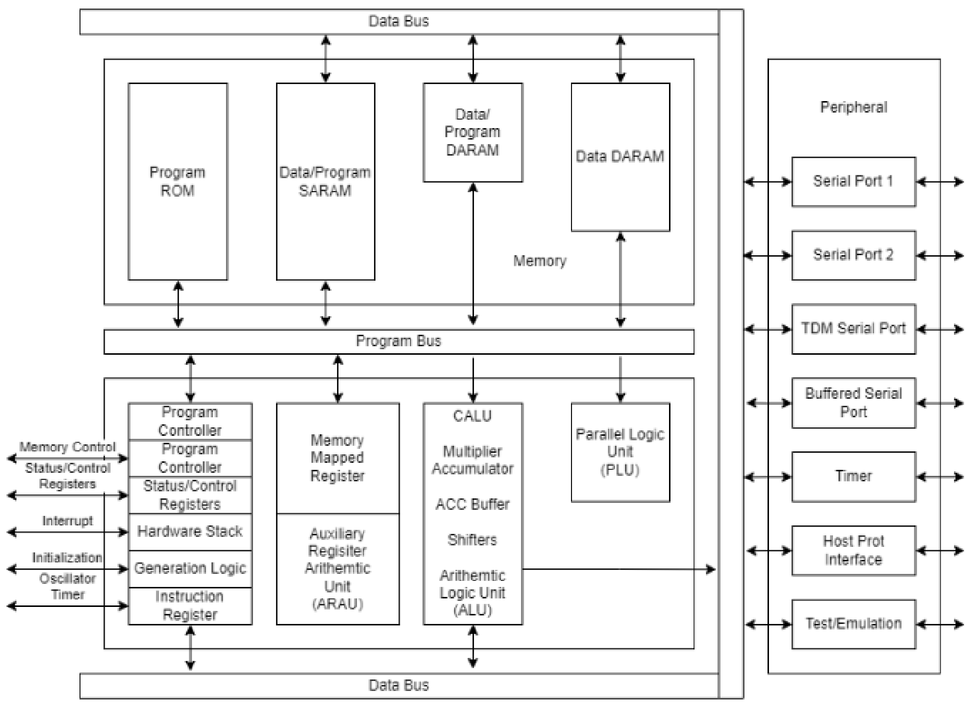


Figure 6: Simplified block diagram of the TMS320C5x architecture [5].

The Central Arithmetic Logic Unit (CALU) for the TMS320C5x consists of several smaller units, including a 16x16 bit parallel multiplier, Accumulator (ACC), ALU, Accumulator Buffer (ACCB), and a 0-16-bit barrel shifter. Several of these components can be found in GPPs, but the modified architecture and the parallel multiplier allow for increased performance in parallel signal processing. The Auxiliary Register ALU (ARAU), handles indirect addresses, which frees the CALU to perform parallel operations while the ARAU calculates addresses.

Memory architecture also plays an important role in DSP performance. Techniques like Dual-Access RAM (DARAM) and Multi-Ported Memory increase throughput by supporting more simultaneous memory accesses per cycle. We expand on this concept in DBPUX with dedicated coefficient caching (LC cache) to support efficient MMU operations, which addresses a limitation in many traditional DSPs where coefficient memory is often a shared resource.

DSP chips are designed to optimize for and take advantage of the parallel nature of signal processing algorithms and data. High efficiency is needed for power and real-time I/O considerations. DSPs use many components and concepts implemented in GPPs. Many of the components in DSPs are modified to accommodate parallelism in signal processing.

DSPs are a strong foundation for SDR due to their domain-specific optimizations and architectural efficiency. However, many conventional DSPs face limitations in flexibility, scalability across multiple bands, and real-time adaptability. The DBPUX builds on DSP principles while integrating design elements from machine learning hardware, aiming to combine the deterministic performance of DSPs with the parallel throughput of GPUs and TPUs.

1.2.4 Graphics Processing Unit

GPPs are generally insufficient for real-time SDR, but real-time SDR becomes possible when a GPU is used as a coprocessor with a CPU. CPUs are for solving general problems and are not optimized for real-time signal processing. CPUs deal in sequential instructions and some compilers implement out of order execution. This rearranging of sequential instructions makes parallel processing difficult. GPPs can be used in research and academia but are not suitable for use in critical real-time situations.

When GPUs are used in conjunction with CPUs, the parallel capabilities of the computer can become suitable for real-time SDR [2,6]. GPUs outperform parallel and serial CPU processing for signal detection performance, with GPUs being 2 orders of magnitude faster than serial CPUs [6]. For passive RADAR SDR applications, GPU outperformed CPUs in some cases, such as the implementation of Gradient Adaptive Lattice (GAL) and Normalized Least Mean Square (NLMS) algorithms. In instances of Range Doppler Elaboration and Cell-Average Constant False Alarm Rate (CA-CFAR), the multithreaded CPU+GPU outperformed a pure GPU. This is because the CPU is needed to control the memory and threads, which are optimized for sequential control in a CPU [7].

This work underscores the potential of leveraging data-level parallelism in SDR, particularly when algorithms exhibit high degrees of regularity or can be broken down into SIMD-like operations. However, GPU usage introduces latency from memory transfer overhead and lacks precise timing control required for deterministic radio protocols. These challenges are critical for real-time SDR applications in constrained environments, such as mobile or embedded systems.

Kim and his colleagues (2010) demonstrated that the NVIDIA GTX9800 GPU significantly outperformed the TMS320C6416 DSP across all major SDR modem tasks. While this highlights the GPU's immense parallel processing potential, the high-power consumption and limited suitability for embedded, low-power SDR environments remain concerns. Our work addresses this trade-off by exploring hybrid architectures like the DBPUX, which aims to retain GPU-like throughput while adopting the lower power characteristics of systolic array-based processing found in TPUs. The tests included derandomization, Viterbi decoder, demodulation, channel estimation, permutation, FFT, iFFT, permutation, CC encoder, and randomization [7]. We can compare the GPU and DSP throughput, tp_{GPU} and tp_{DSP}, with equations (3) and (4) for GPU and DSPs, respectively [7]. The GPU is approximately 90 times faster at Viterbi decoding than the DSP.

$$tp_{GPU} = \frac{\text{allocated data per frame } bit}{\text{processing time per frame } \mu s} \times \frac{1}{0.5(\text{code rate})} \tag{3}$$

$$tp_{DSP} = \frac{bit}{\mu s} \times \frac{1}{0.5}(\text{code rate}) \tag{4}$$

Although GPU performance outpaces that of DSPs in many benchmarked scenarios, the power requirements and lack of domain-specific instruction sets limit their effectiveness in energy-sensitive or highly time-constrained SDR contexts. Our design for DBPUX seeks to retain the advantages of parallel processing demonstrated by GPUs while addressing their power inefficiencies by adopting systolic-array style MMUs and signal-specific instruction sets.

Many GPUs are in PCs and Laptops, which require more power and area. The historically large power and area requirements of GPUs are improving and capable of being implemented in mobile devices. This makes GPU and CPU combinations more appealing as GPU technology improves. Right now, GPUs are not likely to be used for SDR systems in vehicles, satellites, and hard to reach places. A hybrid approach to SDR and signal processing that takes advantage of GPUs inherent parallelism with reduced power needs would be highly sought after in SDR.

GPUs demonstrate that parallelism can accelerate SDR signal processing tasks. However, to achieve lower power consumption, tighter hardware-software integration, and real-time signal control, our proposed DBPUX looks to adopt a hybrid approach. By integrating parallel design elements from both DSPs and GPUs, DBPUX aims to bridge the gap between high-throughput processing and real-time SDR constraints, while keeping energy requirements low.

1.2.5 Tensor Processing Unit

Tensor Processing Unit (TPU) is a processor that specializes in solving machine learning and neural network tasks [8,9]. We can explore the intrinsic parallelism of TPUs for machine learning and compare that to other chip designs that have high intrinsic parallelism, including GPUs and DSPs. Jouppi and his colleagues [8] and K. Sato and C. Young [9] demonstrated that this architecture achieved unprecedented energy efficiency in ML inference workloads. While the domain is different, we hypothesize that the same efficiency could apply to repetitive DSP functions such as FIR filtering, frequency shifting, or matrix transforms, provided the instruction set and dataflow model are adapted appropriately.

These chips are Complex Instruction Set Computers that specialize in machine learning. This is like Texas Instrument DSPs discussed earlier, but with specialized instruction sets for machine learning. Out of the processors discussed, there appears to be a direct relationship between throughput and power consumption, although current trends are indicating a reduction in the power and area requirements of GPUs. TPU chips are available in smartphones today to assist in machine learning tasks.

Figure 7 from Jouppi and his colleagues [8] visualizes the TPU architecture as a simplified block diagram. Two components of the TPU stand out, namely the Systolic Array (SA) and Matrix Multiplier Unit (MMU). SAs have been in use for several decades [10] and assist in matrix multiplication operations for domain specific applications. We can map the matrix to a specific algorithm, such as convolution, modulation, filtering, or transforming. Since processors can be domain specific, or general purpose with features optimized for specific domains, the method of mapping to the array's function units is determined by the needs of the processor. A function unit takes input data from the previous cell and applies a matrix operation on that output data. A cell consists of a memory register and an ALU. In DSPs, the data and memory flow in one diagonal direction, as seen in Figure 8.

In DSP processors, transmitted data and computed data often move orthogonally through the matrix array. In the TPU, the systolic array takes 256 inputs from the unified memory buffer at one time and operates on each cell at the same time. The array wave of computation through the MMU is coordinated through controls and pipelined data. We can measure the efficiency of the MMU by comparing the number of nodes to the number of executed instructions (eq. 5).

$$E(l,p) = \frac{\text{No. of nodes}}{\text{No. of instructions executes}} \qquad (5)$$

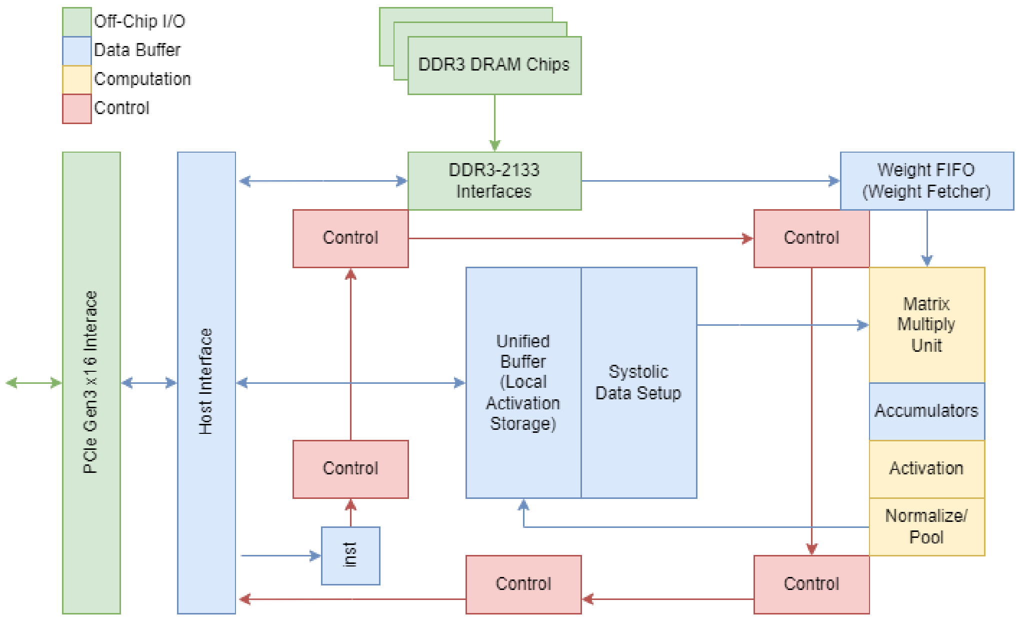


Figure 7: Coded and simplified TPU block diagram. The yellow components are where the primary computations occur for the machine learning algorithms. Systolic-type arrays and buffers transfer data from I/O to the computational components [8].

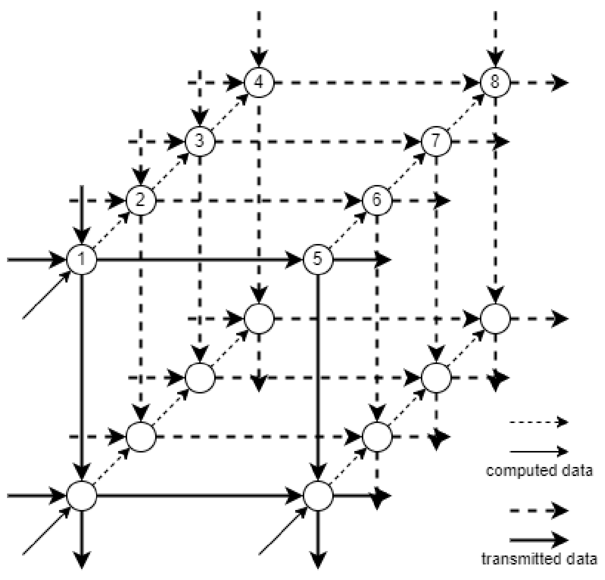


Figure 8: Example of systolic-type array computations for DSP [10].

While certain design aspects of TPUs are ideal for signal processing, like their impressive throughput and low power consumption per instruction, they may not be ideal for signal processing. This is because they are chips designed for machine learning and solving neural networks. The TPU has a CISC instruction set, which has instructions specifically for machine learning [8]. TPUs are efficient because they use 8-bit floating-point precision for matrix multiplication, which may not be sufficient for certain SDR and signal processing applications. For instance, the TMS320C5x discussed earlier uses 16-bit floating point precision.

TPUs are also proficient at batch processing huge amounts of data in parallel. The batching may be good for inspecting large amounts of collected radio signal data but may hinder real-time signal processing for critical communication protocols. TPUs in servers handle large training and inference sets, and TPUs on mobile phones still handle relatively large amounts of data, such as images and videos, Similar to graphics processors, using a tensor processor for signal processing may not initially make sense, but given the inherent parallelism with time to experiment and develop libraries, we may be able to develop ways to leverage TPUs for signal processing in mobile devices. An ideal solution is likely a hybrid solution, where we combine the strengths of TPUs such as the MMU and the SA with multiple channel antennas and a multiplexer [9].

Despite TPUs being ill-suited to real-time SDR due to their batch-oriented nature, their core architectural elements, the SA and MMU, offer a compelling model for low-power, high-throughput signal processing. Prior implementations assume static input dimensions, tolerance for latency, and pre-trained model. These parameters conflict with SDR’s need for real-time, streaming, and hardware-integrated processing. The TPU’s reliance on fused multiply-accumulate operations and quantized weights limits the dynamic range and flexibility required for certain radio waveforms.

1.2.6 SDR Processor Summary

DSP technology has been around since the 1970's [2] and has continued to improve since then. The digital processing of signals can take place any time after analog-dialog conversion, including the audio output of an analog radio. Several types of processors have been used for signal processing in academic and research environments, including FPGAs, DSPs, GPPs, GPUs, and combinations of the above listed.

The challenge with DSP is to balance power, size, and development time/cost. FPGAs can be used for rapid development of hardware, but they can be expensive and challenging to program. GPPs can execute any signal processing algorithm necessary, but they may not be ideal for real-time communication because they do not specialize in signal processing and lack the appropriate parallelism and speed for real-time processing. Dedicated DSP chips can take a lot of resources to develop, but they are specialized in signal processing. Some DSPs are specific to a set of functions, and others are like GPPs with specialized hardware or instructions for signal processing. GPUs are becoming more common in signal processing. Their parallelism makes them appeal to parts of signal processing that are repeatable to independent data, such as modulation, filtering, convolution, and transformation.

Another type of chip that is becoming popular today are machine learning/AI chips, including TPUs. The Tensor G2 chip we studied from Google is a CISC type architecture for machine learning, but certain aspects of it make it appealing to signal processing, such as its use of the systolic array and MMU to reduce the power per instruction to 2pJ [8]. The challenge with using a TPU for signal processing is that they are good for large batches of data, typically video, audio, or images. More challenging than using batched data for real-time communication is the complex instruction set designed for solving neural networks.

These findings suggest that there is room to improve upon existing processors to meet demands of modern SDR, the need for real-time, multi-band signal processing with low power consumption and scalable parallelism. Prior studies indicate that combining architectural elements from DSPs, GPUs, and TPUs may yield a more efficient and adaptable design.

This insight motivates the development of a hybrid solution, the Dynamic Band Processing Unit X (DBPUX), which integrates designs from previous processor architectures while addressing their limitations in the context of SDR. In the following sections, we propose DBPUX as a domain-specific processor that leverages systolic arrays, matrix multipliers, and a custom instruction set to support multi-band SDR processing with efficiency and scalability.

2. Materials and Methods

The initial question we asked ourselves was if we can take advantage of the machine learning chip on the Google Pixel 7 Pro for signal processing as part of an SDR architecture, or if there is a way to utilize aspects of the TPU to enhance SDR capabilities and pry new efficiency gains out of DSPs.

Tensor G2 has several processing units, which include a CPU, GPU, and TPU [11]. The CPU has 8 processors, including 2x Arm Cortex-X1 (2.85 GHz), 2x Arm Cortex-A78 (2.35 GHz), and 4x Arm Cortex-A55 (1.8 GHz). The GPU is an Arm Mali-G710 MP7. The TPU is the Next-gen TPU.

To understand how we can use the TPU for SDR, we examine the architecture of the TPU. We want the TPU to solve for coefficients in the FIR coefficient cache instead of using weights for neural networks. Additionally, the TPU's activation functions need to be overwritten with specific signal processing algorithms. Figure 9 shows how we can begin to repurpose the TPU for signal processing as part of SDR architecture. Doing so with an unrooted Pixel 7 Pro and using common libraries is not readily available. It may be possible to override the TPU's activation functions with a rooted phone and by creating novel low-level libraries.

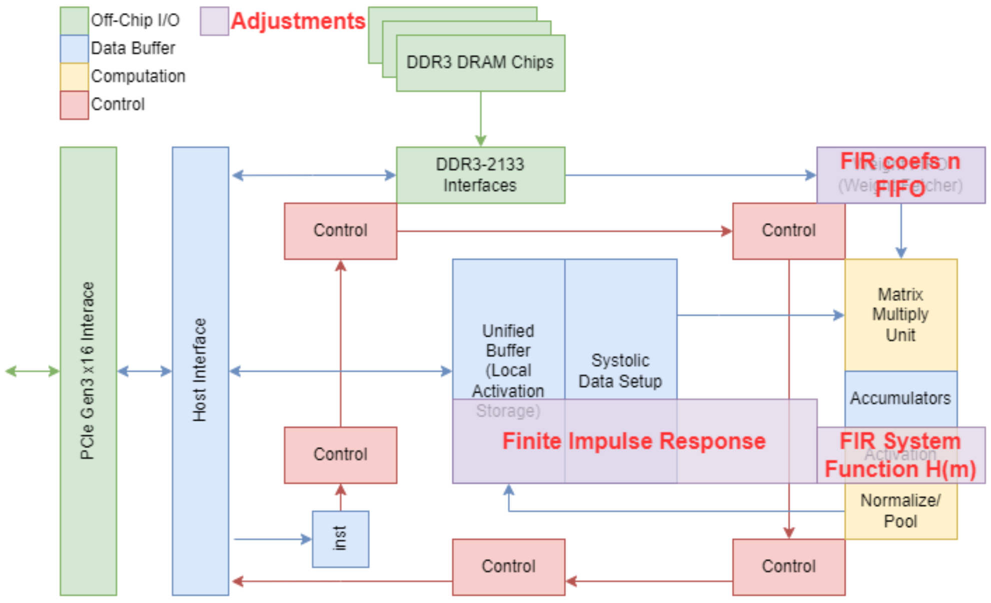
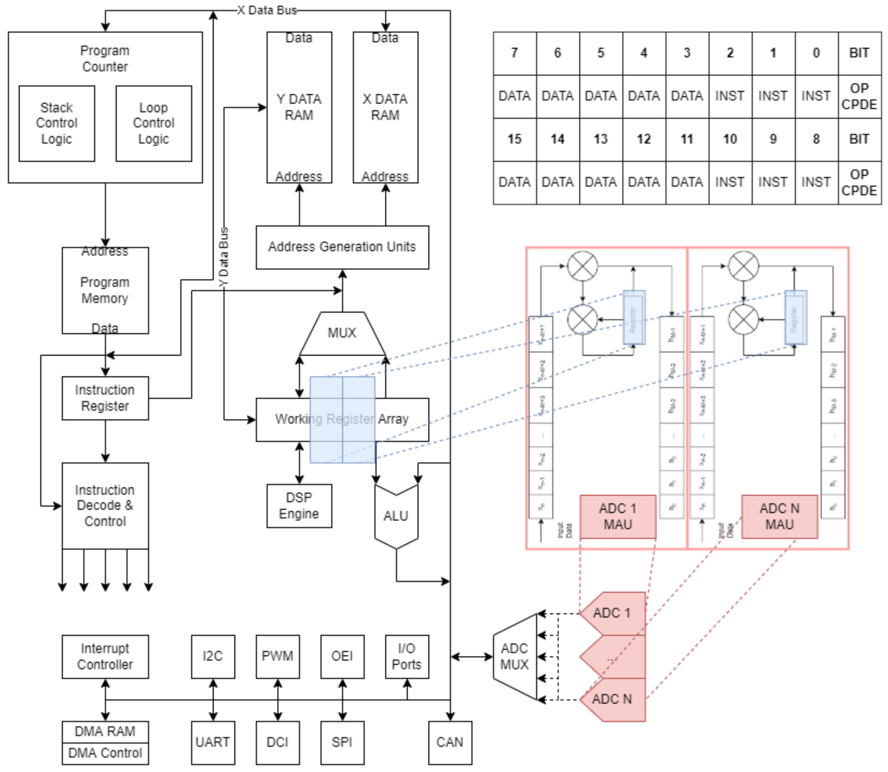


Figure 9: Modified TPU block diagram (Figure 7) showing how the Finite Impulse Response (FIR) could theoretically be mapped to a TPU. The FIR coefficients replace the model weights, and the Activation function is used as a FIR System Function instead.



7	6	5	4	3	2	1	0	BIT
DATA	DATA	DATA	DATA	DATA	INST	INST	INST	OP CPDE
15	14	13	12	11	10	9	8	BIT
DATA	DATA	DATA	DATA	DATA	INST	INST	INST	OP CPDE

Figure 10: A modified block diagram from ARRL [3] taking two ADC streams and multiplexing them into parallel MAC units [5], each using symmetric opcodes. The display opcode shows parallelism, but not the details of precision or address space. Blue highlights the overlap of registers for multiple MACs and red highlights the additional ADC buses necessary and the mapping to their respective MACs.

A promising approach is to design a hybrid processor using a multiplexer to take in multiple signals for digital signal processing and pipeline the bands into multiple instruction channels, offset by different sample steps. We can split a radio signal over two or more bands to process them in parallel on a single chip and then stream that to I/O. The instructions will be a memory-register architecture. Each band passes through the multiplexer, which will split the instruction and pass it to parallel processing as shown in Figure 10. For each clock cycle one bit from each band is processed in two parallel streams; they are merged at the end for further I/O to the PC. We can attempt to further refine the necessary instructions to achieve this and determine how to control the data between the ADCs and the multiplexer. For initial development, we can use simulation software and FPGAs to determine if the proof of concept works.

3. Results

The Dynamic Band Processing Unit X (DBPUX) is a specialized digital signal processor for processing multiple bands per cycle. X can be a variable number of bands. For instance, we can implement a DBTPU8, which processes 8 bands simultaneously on 8 different sample rates, or a DBTPU2 to process 2 bands simultaneously on 2 different sample rates.

The objective of this processor is to efficiently process multiple bands of signal for a software defined radio, while meeting latency and energy constraints needed for real-time SDR. The SDR architecture digitizes as an intermediate frequency and passes the initially processed signals to a PC or other DSP. The MUX moves the sampled bands through the MMU but also is available for the registers to access if any sequential operations are needed on the ADC stream. With this design most of the data and instructions will be around the algorithms of signal processing. The streamed and processed data is passed directly from the MUX to the MMU and back to the MUX.

3.1 Dynamic Band Processing Unit

The DBPUX is designed to take advantage of the anti-aliasing characteristics of DIF SDR architecture, where digitizing at an intermediate frequency enables anti-aliasing before processing, as seen in Figure 11. [3]. With the correct bandwidth and sample rate, we propose layering anti-aliased bands within a sample to efficiently pipeline signals to the processor. The lower sample rate of IF digitization allows sampling more bands in each period.

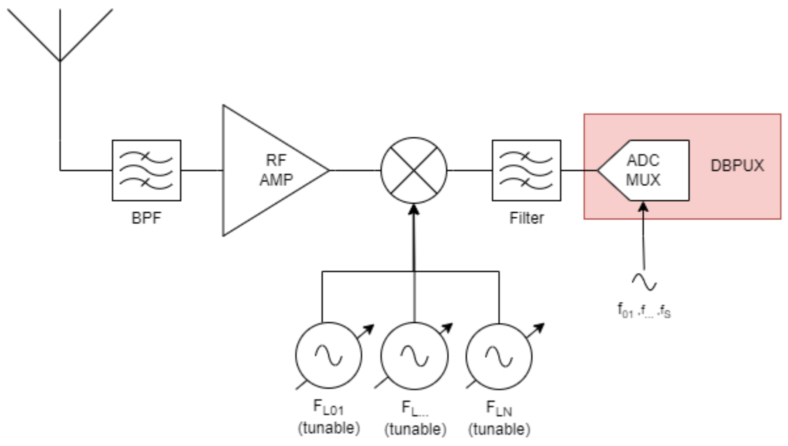


Figure 11: A DIF SDR architecture that demonstrates how digitizing at intermediate, anti-aliased, frequencies can allow for multiband processing in the DBPUX. The DBPUX is designated in red.

The objective of this processor is to efficiently process multiple bands of signals for a software-defined radio (SDR). The SDR architecture digitizes as an intermediate frequency and passes the initially processed signals to a PC or other DSP. The MUX moves the sampled bands through the MMU and is also available for the registers to access if any sequential operations are needed on the ADC stream. With this design, most of the data and instructions will focus on the algorithms of signal processing. The streamed and processed data is passed directly from the MUX to the MMU and back to the MUX depicted in Figure 12.

3.2 Processor Components and Characteristics

The **Program Counter (PC)** serves as the core component responsible for tracking and driving instructions and programs within the system. Complementing this, **Registers** provide fast, nearby memory access to ensure efficient computation. The **Arithmetic Logic Unit (ALU)** handles basic, non-signal processing arithmetic operations, while the **Register Arithmetic Logic Unit (RALU)** is specifically designed for address arithmetic calculations. Signal processing functions for different frequency bands are managed within **H(B)**, which works in conjunction with **Coefficient Memory** to store processing coefficients for each band, both of which play crucial roles in the **Matrix Multiplier Unit (MMU)**.

The **Systolic-Type Array (STA)** prepares the streamed signal data before it is processed by the MMU. The **MMU** itself is responsible for applying the stored functions to the respective frequency bands, operating as a continuous processing stream between the analog-to-digital converter (ADC) and input/output (I/O) interfaces. The system's **Memory Architecture** is categorized into **Data Memory, Instruction Memory, and Coefficient Memory**, supported by caching mechanisms such as the **L1 cache** for data and instruction memory and the **LC cache** for coefficient storage. Addressing beyond the standard instruction space is managed through the **Direct Memory Pointer (DMP)**, ensuring seamless access to extended instruction sets.

To facilitate seamless signal routing, the **Band Multiplexer (MUX)** acts as the bridge between the ADC, I/O, and the main processing unit. Data flow across the architecture is maintained through the **Address Bus**, which ensures accurate transportation of instruction and data memory addresses, and the **Data Bus**, which connects registers to the **H(B)** for efficient processing. Together, these components form a cohesive system designed for high-performance signal processing in a structured and efficient manner.

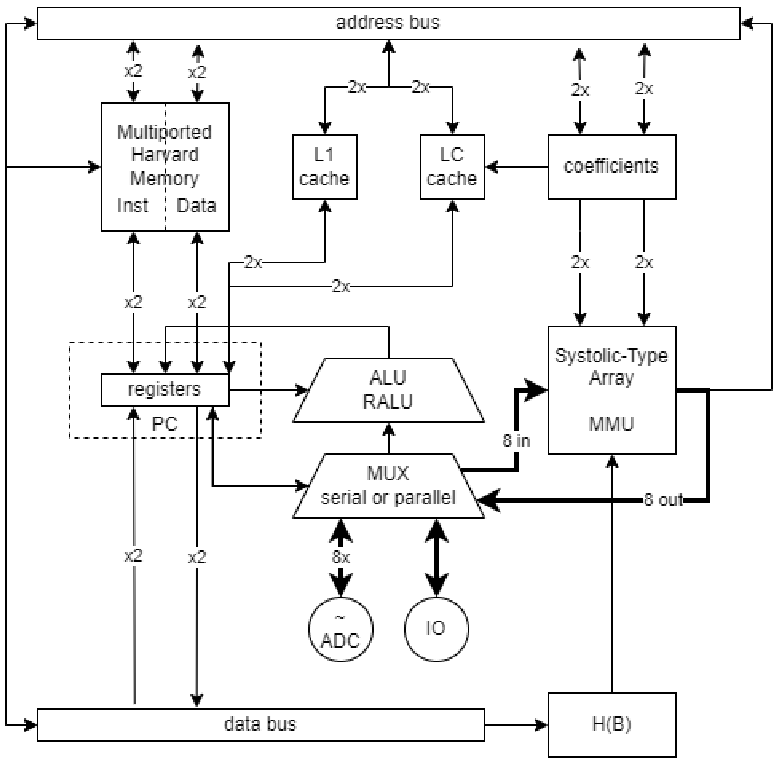


Figure 12: Simplified block diagram of the DBPU8 for DSP. The MMU processes signal processing algorithms on 8 bands at a time.

3.3 Registers

The system incorporates a set of General Registers, consisting of $2^5=32$ registers (r_0 to r_{31}), each with an 11-bit capacity. These registers serve as the primary storage locations for general-purpose data and intermediate computations, ensuring efficient processing and rapid access. The H Registers provide a specialized storage mechanism with $2^3=8$ registers (b_0 to b_7), each also 11 bits in size. These registers are designed to store hard coded and custom H functions to optimize the execution of designated signal processing functions.

3.4 Instruction Set

The system's instruction set architecture (ISA) is designed to support up to 32 distinct instructions, as determined by the first five bits of the opcode (Table 1). These instructions encompass arithmetic, logical, memory, and control operations. Fundamental arithmetic operations include addition (**add**, opcode 0), subtraction (**sub**, opcode 1), multiplication (**mul**, opcode 2), and division (**div**, opcode 3). Their immediate-value counterparts, **addi** (opcode 4), **subi** (opcode 5), **muli** (opcode 6), and **divi** (opcode 7), allow for direct computation with embedded constants.

Logical operations include **and** (opcode 8), **or** (opcode 9), **xor** (opcode 10), and **not** (opcode 11), enabling bitwise manipulations. Comparison instructions facilitate conditional execution, with less than (**lt**, opcode 12), greater than (**gt**, opcode 13), less than or equal (**lte**, opcode 14), and greater than or equal (**gte**, opcode 15).

The architecture also includes specialized operations such as equality check (**eqq**, opcode 16), load coefficient (**lc**, opcode 17), and store coefficient (**sc**, opcode 18). Data movement instructions include load immediate (**li**, opcode 19), store immediate (**si**, opcode 20), load data (**ld**, opcode 21), and store data (**sd**, opcode 22).

Memory management and matrix operations are handled by register matrix move (**rmm**, opcode 23), load matrix memory (**lmm**, opcode 24), and matrix multiplier unit operation (**mmu**, opcode 25). Control flow is managed with **jump** (opcode 26) and conditional branching, including branch if not equal (**bne**, opcode 27) and branch if equal (**beq**, opcode 28). Additionally, matrix-specific memory operations, such as store matrix memory high (**smmh**, opcode 29) and register matrix move high (**rmmh**, opcode 30), are available. Finally, the repeat (**rpt**, opcode 31) instruction facilitates loop execution for optimized performance.

Table 1: Instruction set for DBPUX.

Opcode	Operation	Opcode	Operation
0	add	16	eqq
1	sub	17	lc
2	mul	18	sc
3	div	19	li
4	addi	20	si
5	subi	21	ld
6	muli	22	sd
7	divi	23	rmm
8	and	24	lmm
9	or	25	mmu
10	xor	26	jump
11	not	27	bne
12	lt	28	beq
13	gt	29	smmh
14	lte	30	rmmh
15	gte	31	rpt

3.5 Instruction Size

The DBPUX uses a varying instruction set size to maintain a consistent address space. Each instruction begins with a 3-bit overhead to indicate its type, followed by a 5-bit opcode that defines the operation being performed. The instruction formats are categorized into five primary types: Harvard (H), Arithmetic (A), MMU (M), Move (O), and Jump (J), each tailored to specific processing requirements (Table 2).

The Harvard (H) format consists of 26 bits and is structured as op(5) r(5) c(11) r(5). This format is designed for operations that require access to both register and coefficient memory, enabling efficient data handling in the processing pipeline. It allows instructions to specify a source register, a coefficient value, and a destination register, facilitating optimized signal processing.

The Arithmetic (A) format varies between 20 and 26 bits, depending on whether an immediate constant is used. It follows the structure op(5) r(5) r(5) r/c(5/11), where the last field can either reference a third register (5 bits) or an immediate constant (11 bits). This flexibility allows the execution of standard arithmetic operations such as addition, subtraction, multiplication, and division while accommodating both register-based and immediate-value calculations.

The Matrix Multiplier Unit (MMU) format (M) ranges from 13 to 19 bits, formatted as op(5) b(3) r/addr/c(5/11/11). This format is specifically optimized for matrix operations, referencing b(3) to select an H register and using the final field to store either a register reference (5 bits), an address (11 bits), or a coefficient

(11 bits). This design ensures that matrix operations efficiently integrate with the broader computational workflow.

The Move (O) format is fixed at 26 bits, structured as op(5) r(5) r(5) addr(11). It facilitates data transfer between registers and memory, enabling efficient data manipulation while ensuring that register contents can be stored or loaded from memory locations with a direct address reference.

The Jump (J) format is 16 bits in length, following the structure op(5) addr(11). This format is designed for control flow operations, allowing the processor to alter execution paths based on specified conditions. The 11-bit address field ensures that jumps can target a wide range of instruction locations within the program space.

Table 2: Instruction formats for DBPUX.

Type	Symbol	Size (bits)	Format
Harvard	H	26	op(5) r(5) c(11) r(5)
Arithmetic	A	20-26	op(5) r(5) r(5) r/c(5/11)
MMU	M	13-19	op(5) b(3) r/addr/c(5/11/11)
Move	O	26	op(5) r(5) r(5) addr(11)
Jump	J	16	op(5) addr(11)

3.6 Instruction Pipeline

The DBPUX instruction pipeline follows a structured sequence of six stages, ensuring efficient execution of instructions while optimizing performance in a pipelined processor architecture. The stages Fetch, Decode, Execute, Memory, Write, and MMU Update operate concurrently in different pipeline stages to increase instruction throughput.

The Fetch (fi) stage retrieves the next instruction from Instruction Memory. The PC directs the processor to the correct memory location, and the instructions are loaded into the pipeline. If the instruction set supports variable-length encoding, the fetch stage may require additional cycles to retrieve longer instructions.

Once fetched, the instruction enters the Decode (di) stage, where it is interpreted, and its operands are identified. The opcode determines the operation type, and the relevant registers, immediate values, or memory addresses are extracted.

During the Execute (xi) stage, the actual computation takes place. Depending on the instruction type, different functional units may be utilized. The ALU performs operations like addition, subtraction, multiplication, and bitwise logic. If the instruction involves matrix operations, the MMU is activated. For branch/jump instructions, the PC is updated, and wrongly predicted branches may trigger pipeline flushing.

The Memory (mi) stage is responsible for accessing data from Data Memory. This stage is crucial for load (ld) and store (sd) instructions, where memory access times can impact performance.

In the Write (wo) stage, the results of the instruction are written back to the register file. If the instruction involves memory access, the retrieved data is stored in the destination register.

For matrix operations, the MMU Update (ui) stage finalizes computations by executing one cycle of the MMU and updating the H Registers and Coefficient Memory. This ensures that matrix transformations and multiplications are correctly stored and ready for subsequent operations.

3.7 Memory Structure

DBPUX employs a Harvard architecture system, which maintains separate memory spaces for data, instructions, and coefficients. This separation enables parallel access to different types of memory, significantly improving performance, especially in signal processing and high-throughput computation applications. This architecture ensures efficient execution by reducing memory access bottlenecks, a common issue in Von Neumann architecture where instructions and data share a common memory bus.

The three memory types, Data, Instruction, and Coefficient each have 256 words per block with a varying number of blocks. The Coefficient memory contains the least blocks and total bits as it is generally more stable than the data and instruction memory.

Table 3: Memory structure for DBPUX.

Memory Type	Blocks	Words/Block	Bits/Word	Total Bits
Data	1000	256	16	4,096,000
Instruction	1000	256	29	7,424,000
Coefficient	32	256	11	90,112
Total	**2032**	**256**		**11,610,112**

3.8 Cache Levels

The DBPUX processor has a dual-cache system to optimize memory access and improve processing efficiency. This system consists of a unified L1 cache for both data and instructions and a dedicated LC cache for coefficient memory. These caches help reduce memory bottlenecks, enhance execution speed, and maintain efficient data flow within the processor.

The L1 cache is a unified cache with 128 blocks, designed to store both instruction and data memory. This architecture enables the processor to dynamically allocate cache space between data and instructions based on workload demands. Frequently accessed instructions and data are stored within the L1 cache, reducing memory access latency. By enabling simultaneous instruction to fetch and access data, the L1 cache minimizes stalls and maximizes computational throughput. The replacement policy for this cache is Least Recently Used (LRU).

The DBPUX also incorporates an LC cache, a specialized 8-block cache dedicated to coefficient memory. This cache is optimized for signal processing workloads, where matrix operations and precomputed coefficients play a vital role in accelerating computations. The LC cache operates using an LRU replacement policy, ensuring that frequently accessed coefficients are retained while less-used values are evicted when new data is needed. By reducing the need for repeated memory accesses, the LC cache enhances the efficiency of the Matrix Multiplier Unit (MMU).

3.9 DBPUX Constraints, Limitations, and Future Work

While this study proposes a novel multi-band digital signal processing architecture, several limitations must be acknowledged. First, the DBPUX is a conceptual processor and has not been implemented in hardware or simulated in software. Thus, performance metrics such as instruction latency, power consumption, and thermal efficiency are speculative and extrapolated from related architectures like the TPU and TMS DSP families. The proposed instruction set has not yet been tested in FPGA or compiler toolchains, making real-time processing assumptions unverified. Additionally, the use of mobile TPUs for SDR remains constrained by limited developer access and the absence of low-level control on most consumer devices. Future work will involve building a DBPUX simulation environment to validate the instruction pipeline, memory architecture, and multi-band processing performance.

There are several constraints that need to be taken into consideration, such as energy usage, size, and cost. Connecting 8 antennas to one processor may prove impractical, though we may use one antenna with a sample rate synchronized tuner. We need to determine benchmark tests that cover efficiency and performance to compare our processor against.

There are several next steps we can take to continue studying and prototyping DBPUX. We can virtualize the prescribed ISA. Having a virtual ISA to simulate processing digital signals will allow us to build tests and highlight areas of the instruction set that need to be refined. We can simulate multi-band signals using tools like GNU Radio [12] and then use the virtual ISA to process the simulated signals. If the ISA can successfully process the simulated signals, then we can begin to estimate the efficiency (5) of the processor and the implementation of the systolic-type array in the MMU that does the multi-band processing. After developing a virtual processor, we can implement the solution in an FPGA or a protype chip.

4. Conclusion

SDR technology has come a long way since its inception in the 1970s. Today, there are several options for SDR depending on your size, energy, cost, and speed of development requirements. SDR's main advantage is the ability to update algorithms and signal processing systems by updating the software and not the hardware. Today, GPPs, GPUs, FPGAs, and DSPs can all be used for SDR in some capacity. FPGAs and DSPs are the best suited for real-time critical radio communication, but GPPs and GPUs perform better each day as GPUs become more affordable and accessible. TPUs share many similar traits to GPUs and are being added to personal mobile devices.

We proposed DBPUX as the processor for a DIF SDR architecture, enabling the processing of multiple bands per instruction after the intermediate frequencies have been digitized and anti-aliased. DBPUX takes advantage of a Harvard architecture to perform predefined DSP algorithms on signals passed through the MMU. This has the potential to reduce the energy per instruction to less than 2 pJ, as seen in the TPU's implementation of Systolic-Type Arrays and MMUs. There are several limitations to the hypothetical processor that need to be tested and verified, but we have outlined the steps needed to bring the idea to fruition and test its efficiency as an SDR processor.

With parallel processing technology improving the performance of GPUs, TPUs, and other hybrid solutions, more options for DSP are regularly becoming available to consumers and researchers. From here we can continue to search for low power, highly parallel digital signal processors for software defined radio.

References

[1] T. Ulversoy, "Software Defined Radio: Challenges and Opportunities," IEEE Communications Surveys & Tutorials, vol. 12, no. 4, pp. 531–550, 2010, doi: https://doi.org/10.1109/surv.2010.032910.00019.

[2] R. Akeela and B. Dezfouli, "Software-defined Radios: Architecture, state-of-the-art, and challenges," Computer Communications, vol. 128, pp. 106–125, Sep. 2018, doi: https://doi.org/10.1016/j.comcom.2018.07.012.

[3] H. Ward Silver, The ARRL Handbook for Radio Communications 2023, 100th ed., vol. 2. The American Radio Relay League, Inc., 2022.

[4] P. W. Garver, R. Abler, E. J. Coyle, and J. Narayan, "Comparisons of high performance software radios with size, weight, area and power constraints," Proceedings of the 9th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization, pp. 17–24, Sep. 2014, doi: https://doi.org/10.1145/2643230.2643238.

[5] A. V. Oppenheim and R. W. Schafer, Discrete-time Signal Processing, 3rd ed. Harlow: Pearson Education Limited, 2010.

[6] M. Bernaschi, A. D. Lallo, R. Fulcoli, E. Gallo, and L. Timmoneri, "Combined use of graphics processing unit (GPU) and Central Processing Unit (CPU) for passive radar signal & data elaboration," International Radar Symposium, pp. 315–320, Oct. 2011.

[7] J. Kim, S. Hyeon, and S. Choi, "Implementation of an SDR system using graphics processing unit," IEEE Communications Magazine, vol. 48, no. 3, pp. 156–162, Mar. 2010, doi: https://doi.org/10.1109/mcom.2010.5434388.

[8] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and Evaluation of the First Tensor Processing Unit," IEEE Micro, vol. 38, no. 3, pp. 10–19, May 2018, doi:

https://doi.org/10.1109/mm.2018.032271057.

[9] K. Sato and C. Young, “An in-depth look at Google’s first Tensor Processing Unit (TPU),” Google Cloud Blog, 2021. https://cloud.google.com/blog/products/ai-machine-learning/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

[10] J. Moreno and T. Lang, Matrix Computations on Systolic-Type Arrays. Springer Science & Business Media, 1992.

[11] R. Triggs, “Google Tensor G2 chip: Everything you need to know,” Android Authority, Aug. 16, 2023. https://www.androidauthority.com/google-tensor-g2-explained-3216087/

[12] “GNU Radio,” https://wiki.gnuradio.org/index.php/Main_Page