5.3 Architectural Design

5.3.1 Memory System Architecture

This section presents several memory system architectures used in FFT processors followed by a description of the one chosen for Spiffee. All example processors can calculate 1024-point complex FFTs.

Single-memory

The simplest memory system architecture is the single-memory architecture, in which a memory of at least $N$ words connects to a processor by a bi-directional data bus. In general, data are read from and written back to the memory once for each of the $\log_2 N$ stages of the FFT.

![Figure 5.1: Single-memory architecture block diagram](image)

Dual-memory

Processors using a dual-memory architecture connect to two memories of size $N$ using separate busses. Input data begin in one memory and “ping-pong” through the processor from memory to memory $\log_2 N$ times until the transform has been calculated. The Honeywell DASP processor (Magar et al., 1988), and the Sharp LH9124 processor (Sharp Electronics Corporation, 1992) use the dual-memory architecture.

![Figure 5.2: Dual-memory architecture block diagram](image)
CHAPTER 5. AN ENERGY-EFFICIENT, SINGLE-CHIP FFT PROCESSOR

Pipeline

For processors using a pipeline architecture, a series of memories, which generally range in size from $O(N)$ to a few words, replace $N$-word memories. Either physically or logically, there are $\log_r N$ stages. Figure 5.3 shows the flow of data through the pipeline structure and the interleaving of processors and buffer memories. Typically, an $O(r)$-word memory is on one end of the pipeline, and memory sizes increase by $r$ through subsequent stages, with the final memory of size $O(N)$. The LSI L64280 FFT processor (Ruetz and Cai, 1990; LSI Logic Corporation, 1990a; LSI Logic Corporation, 1990b; LSI Logic Corporation, 1990c) and the FFT processor designed by He and Torkelson (1998) use pipeline architectures.

![Pipeline architecture block diagram](image)

**Figure 5.3:** Pipeline architecture block diagram

Array

Processors using an array architecture comprise a number of independent processing elements with local buffers, interconnected through some type of network. The Cobra FFT processor (Sunada *et al.*, 1994) uses an array architecture and is composed of multiple chips which each contain one processor and one local buffer. The Plessey PDSP16510A FFT

![Array architecture block diagram](image)

**Figure 5.4:** Array architecture block diagram
processor (GEC Plessey Semiconductors, 1993; O’Brien et al., 1989) uses an array-style architecture with four datapaths and four memory banks on a single chip.

**Cached-memory**

The cached-memory architecture is similar to the single-memory architecture except that a small cache memory resides between the processor and main memory, as shown in Fig. 5.5. Spiffie uses the cached-memory architecture since a hierarchical memory system is necessary to realize the full benefits of the cached-FFT algorithm.

![Figure 5.5: Cached-FFT processor block diagram](image)

Performance of the memory system can be enhanced, as Fig. 5.6 illustrates, by adding a second cache *set*. In this configuration, the processor operates out of one cache set while the other set is being flushed and then loaded from memory. If the cache flush time plus load time is less than the time required to process data in the cache, which is easy to accomplish, then the processor need not wait for the cache between groups. The second cache set increases processor utilization and therefore overall performance, at the expense of some additional area and complexity.

![Figure 5.6: Block diagram of cached-memory architecture with two cache sets](image)

Performance of the memory system shown in Fig. 5.6 can be further enhanced by partitioning each of the cache’s two sets (0 and 1) into two *banks* (*A* and *B*), as shown in Fig. 5.7.
CHAPTER 5. AN ENERGY-EFFICIENT, SINGLE-CHIP FFT PROCESSOR

The double-banked arrangement increases throughput as it allows an increased number of cache accesses per cycle. Spiffee uses this double-set, double-bank architecture.

5.3.2 Pipeline Design

Because the state of an FFT processor is independent of datum values, a deeply-pipelined FFT processor is much less sensitive to pipeline hazards than is a deeply-pipelined general-purpose processor. Since clock speeds—and therefore throughput—can be dramatically increased with deeper pipelines that do not often stall, Spiffee has an aggressive cache→processor→cache pipeline. The cache→memory and memory→cache pipelines have somewhat-relaxed timings because the cached-FFT algorithm puts very light demands on the maximum cache flushing and loading latencies.

Datapath pipeline

Spiffee’s nine-stage cache→processor→cache datapath pipeline is shown in Fig. 5.8. In the first pipeline stage, the input operands \( A \) and \( B \) are read from the appropriate cache set and \( W \) is read from memory. In pipeline stage two, operands are routed through two \( 2 \times 2 \)
crossbars to the correct functional units. Four \( B_{\{\text{real, imag}\}} \times W_{\{\text{real, imag}\}} \) multiplications of the real and imaginary components of \( B \) and \( W \) are calculated in stages three through five. Stage six completes the complex multiplication by subtracting the real product \( B_{\text{imag}} \times W_{\text{imag}} \) from \( B_{\text{real}} \times W_{\text{real}} \) and adding the imaginary products \( B_{\text{real}} \times W_{\text{imag}} \) and \( B_{\text{imag}} \times W_{\text{real}} \). Stage seven performs the remaining additions or subtractions to calculate \( X \) and \( Y \), and pipeline stages eight and nine complete the routing and write-back of the results to the cache.

**Pipeline hazards**

While deep pipelines offer high peak performance, any of several types of pipeline conflicts, or “hazards” (Hennessy and Patterson, 1996) normally limit their throughput. Spiffee’s pipeline experiences a read-after-write data hazard once per *group*, which is once every 80 cycles. The hazard is handled by stalling the pipeline for one cycle to allow a previous write to complete before executing a read of the same word. This hazard also could have been handled by bypassing the cache and routing a copy of the result directly to pipeline stage two—negating the need to stall the pipeline—but this would necessitate the addition of another bus and another wide multiplexer into the datapath.

**Cache→memory pipelines**

As Eq. 5.4 shows, the cached-FFT algorithm significantly reduces the required movement of data to and from the main memory. The main memory arrays are accessed in two cycles in order to make the design of the main memory much easier and to reduce the power they consume. In the case of memory writes, only one cycle is required because it is unnecessary to precharge the bitlines or use the sense amplifiers.

Figure 5.9 shows the cache→memory pipeline diagram. A cache is read in the first stage, the data are driven onto the memory bus through a \( 2 \times 2 \) crossbar in stage two, and data are written to main memory in stage three.

![Figure 5.9: Cache→memory pipeline diagram](image)
The memory→cache pipeline diagram is shown in Fig. 5.10. In the first stage, the selected memory array is activated, bitlines and sense amplifiers are precharged, and the array address is predecoded. In the second pipeline stage, the wordline is asserted; and the data are read, amplified, and latched. In stages three and four, data are driven onto the memory bus, go through a crossbar, and are written to the cache.

5.3.3 Datapath Design

In this section, “full” and “partial” blocks are discussed. By “full” we mean the block is “fully parallel,” or able to calculate one result per clock cycle, even though the latency may be more than one cycle. By “partial” we mean the complement of full, implying that at least some part of the block is iterative, and thus, new results are not available every cycle.

A “block” can be a functional unit (e.g., an adder or multiplier), or a larger computational unit such as a complete datapath (e.g., an FFT butterfly). A partial functional unit contains iteration within the functional unit, and so data must flow through the same circuits several times before completion. A partial datapath contains iteration at the functional unit level, and so a single functional unit is used more than once for each calculation the computational unit performs.

A full non-iterative radix-2 datapath has approximately the right area and performance for a single-chip processor using 0.7 µm technology. Spiffie’s datapath calculates one complex radix-2 DIT butterfly per cycle. This fully-parallel non-iterative butterfly processor has high hardware utilization—100% not including system-wide stalls.

Some alternatives considered for the datapath design include:

- A higher-radix “full” datapath—unfortunately, this is too large to fit onto a single die
- Higher-radix “partial” datapaths (e.g., one multiplier, one adder,…)
• Higher-radix datapath with “partial” functional units \((e.g., \text{radix-4 with multiple small iterative multipliers and adders})\)

• Multiple “partial” radix-2 butterfly datapaths—in this style, multiple self-contained units calculate butterflies without communicating with other butterfly units. Iteration can be performed either at the datapath level or at the functional unit level.

The primary reasons a “full” radix-2 datapath was chosen are because of its efficiency, ease of design, and because it does not require any local control, that is, control circuits other than the global controller.

5.3.4 Required Functional Units

The functional units required for the Spiffee FFT processor include:

- **Main memory** — an \(N\)-word \(\times 36\)-bit memory for data
- **Cache memories** — 32-word \(\times 40\)-bit caches
- **Multipliers** — 20-bit \(\times 20\)-bit multipliers
- **\(W_N\) generation/storage** — coefficients generated or stored in memories
- **Adders/subtracters** — 24-bit adders and subracters
- **Controller** — chip controller
- **Clock** — clock generation and distribution circuits

5.3.5 Chip-Level Block Diagram

Once the required functional units are selected, they can be arranged into a block diagram showing the chip’s floorplan, as shown in Fig. 5.11. Figure 6.1 on page 129 shows the corresponding die microphotograph of Spiffee1.

5.3.6 Fixed-Point Data Word Format

Spiffee uses a signed 2’s-complement notation that varies in length from 18+18 bits to 24+24 bits. Table 5.1 gives more details of the format. Sign bits are indicated with an “\(S\)” and general data bits with an “\(X\)”
Figure 5.11: Chip block diagram

<table>
<thead>
<tr>
<th>Format</th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>General format</td>
<td>SXXXXXXXXXXXXXXXXXXXX</td>
<td></td>
</tr>
<tr>
<td>Minimum value</td>
<td>10000000000000000000</td>
<td>−1.0</td>
</tr>
<tr>
<td>Maximum value</td>
<td>01111111111111111111</td>
<td>+0.9999981</td>
</tr>
<tr>
<td>Minimum step size</td>
<td>00000000000000000000</td>
<td>+0.0000019</td>
</tr>
</tbody>
</table>

Table 5.1: Spiffie’s 20-bit, 2’s-complement, fixed-point, binary format
Figure 6.1: Microphotograph of the Spiffee1 processor