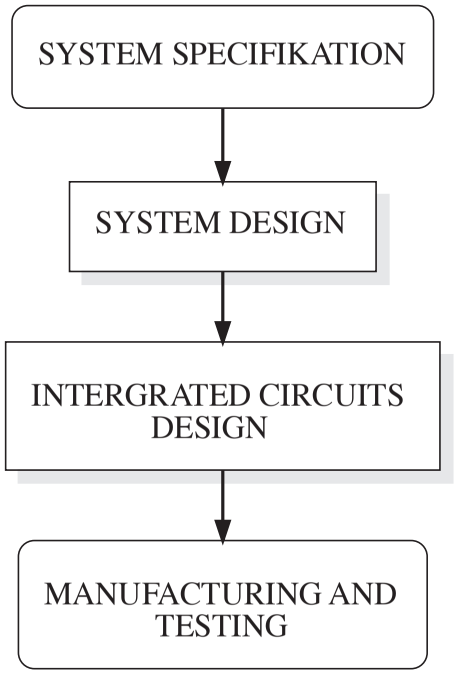
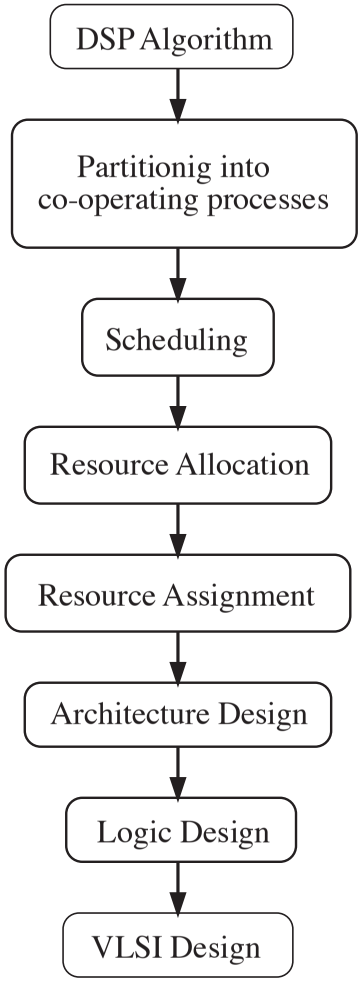
DSP SYSTEM DESIGN STRATEGY





DIRECT MAPPING TECHNIQUE

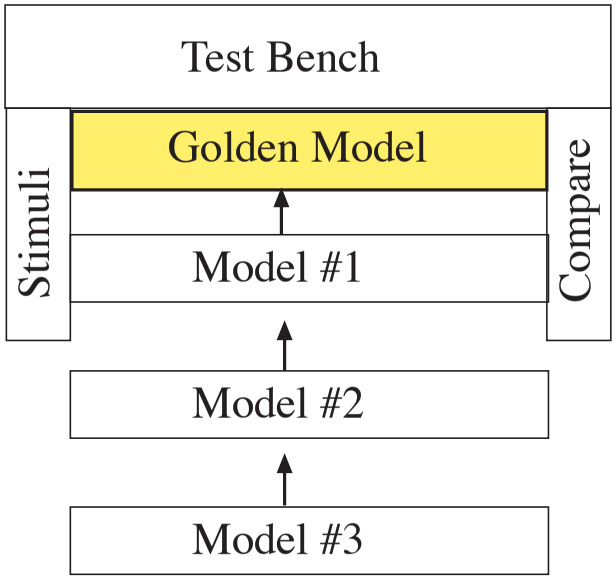
PROBLEM UNDERSTANDING

We advocate that a sequence of models with increasing degrees of details and functionality i built in order to

better understand the “problem”;

explore the design space;

obtain a specification



FFT PROCESSOR, CONT.

In this chapter we will perform the main steps in the system design phase of the FFT processor.

This means that we will synthesize the architecture with its hardware modules step by step, starting from the algorithm.

We begin by partitioning the algorithm, followed by scheduling, resource allocation, resource assignment, and finally, synthesis of a matching architecture.

We will first perform a crude design, then subsequently improve it until a satisfactory solution is reached.

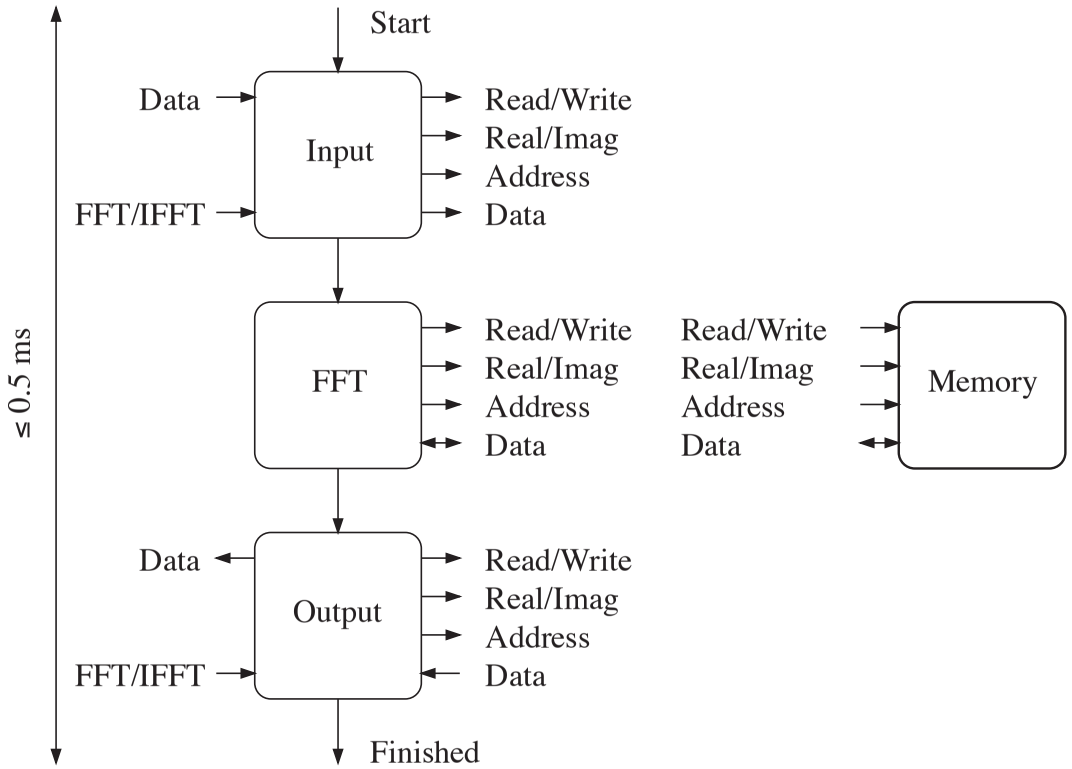
First Design Iteration

In a top-down design approach we start by identifying the major tasks (processes) to be executed.

At the top level we partition the FFT processor into four communicating processes:

input, FFT, output, and memory process.

Generally, it is useful to introduce a memory process at this level explicitly. The input process is responsible for reading data into the FFT processor from the outside world.



Input process: Interchanges the real and imaginary parts if needed.

Memory process: The input sequence and intermediate results are stored by the memory process.

FFT process: The FFT process only computes the FFT since the IFFT is computed by interchanging the real and imaginary parts of the data in the input and output processes.

Output process: Writes data from the memory to the outside world, performs unscrambling of the data array in the last stage of the FFT. Interchanges the real and imaginary parts if needed.

These processes are then partitioned into a hierarchy of simpler processes.

Partitioning should be performed such that the processes at the lowest level in the hierarchy can be easily mapped onto hardware resources.

The partitioning should be done in steps where each step is sufficiently small as to make the transformation obvious, thereby avoiding design errors.

Resource Allocation

Next we determine the resources required.

The partitioning assumes that the input, FFT, and output processes are executed sequentially.

It is possible to overlap their execution and speed up the FFT processor slightly, but the complexity of the design would increase significantly. The number of butterfly operations for each FFT is

$$\frac{N}{2}\log_2(N) = 5120$$

A bit-serial butterfly PE can be implemented using $W_d = 24$ clock cycles and we assume that the usable clock frequency, $f_{CLPEmax}$, is at least 220 MHz. The time available for the FFT is $t_{FFT} = 0.372$ ms. The number of butterfly PEs is

$$N_{PEb} = \frac{W_d \cdot \frac{N}{2}\log_2(N)}{t_{FFT} f_{CLPEmax}} = \frac{24 \cdot 5120}{0.372 \cdot 10^{-3} \cdot 220 \cdot 10^6} \approx 1.5$$

Thus, we need only two butterfly PEs to meet the required throughput. The minimum required clock frequency for the PEs is

$$f_{CLPE} = \frac{W_d \cdot \frac{N}{2}\log_2(N)}{N_{PE} \cdot t_{FFT}} = \frac{24 \cdot 5120}{2 \cdot 0.372 \cdot 10^{-3}} = 165.2 \text{ MHz}$$

We can also estimate the required data rate for the memory process.

For each butterfly operation we must read and write two complex values. Hence, the data rate will be

$$f_{memory} = \frac{(2+2)\frac{N}{2}\log_2(N)}{t_{FFT}} = 55 \cdot 10^6 \text{ complex words/s}$$

In principle, it is possible to use only one logical memory.

However, we choose to use two logical memories since this will simplify the implementation of the memories and increase the memory bandwidth, which is a critical factor.

Also, it is desirable that the memory clock frequency is a multiple of the I/O frequency.

This will make it possible to use the same clock frequency for the memories throughout the input, FFT, and output processes.

We therefore decide to use a memory clock frequency of 32 MHz, i.e. twice the I/O data rate. The time required for the FFT then becomes

$$t_{FFT} = \frac{(2+2)\frac{N}{2}\log_2(N)}{2 \cdot 32 \cdot 10^6} = 0.32 \text{ ms}$$

and the minimum required clock frequency for the PEs is

$$f_{CLPE} = \frac{24 \cdot 5120}{2 \cdot 0.32 \cdot 10^{-3}} = 192 \text{ MHz}$$

The throughput becomes

$$\frac{1}{t_{I/O} + t_{FTT}} = \frac{1}{0.128 \cdot 10^{-3} + 0.32 \cdot 10^{-3}} = \frac{1}{0.488 \cdot 10^{-3}} = 2232 \text{ FFTs/s}$$

Second Design Iteration

Note that both the original FFT program and the description using communicating processes are sequential descriptions.

The following design iteration therefore aims to modify the original sequential algorithm into a parallel description that can be mapped efficiently onto the hardware resources.

We must modify the original program so that two butterfly operations can be executed concurrently.

We must also compute two coefficients W^p concurrently. There are many possible ways to do this.

Let us therefore first consider the signal-flow graph for a 16-point Sande–Tukey's FFT where the butterfly operation has been chosen as the basic operation.

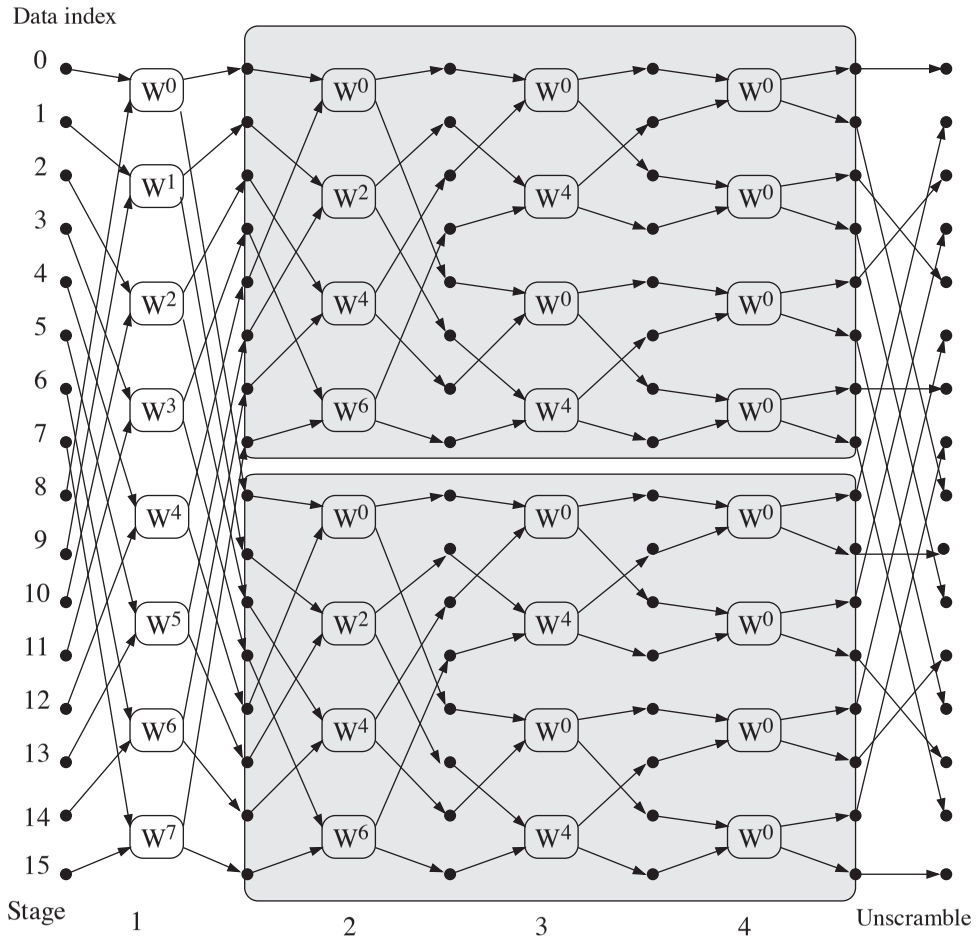
There are $M = \log_2(N)$ stages with $N/2$ butterfly operations each.

The coefficients (i.e., the twiddle factors) in the two parts are the same.

The relation between the coefficients for the butterflies in rows p and $p + N/4$ in the first stage of the FFT is

$$W^{p+N/4} = W^{N/4}W^{p} = -jW^{p}$$

Thus, the twiddle factor for one of the butterfly PEs can be obtained from the twiddle factor of the corresponding butterfly PE by simply changing the sign and interchanging the real and imaginary parts.



Fast Fourier Transform – Sande-Tukey

```
for Stage = 1 to M
    for q = 1 to (N div(2*Ns)
        for n = 1 to Ns
            Butterfly
        next n
    next q
next Stage
```

```
for Stage = 1 to M
    for ???
        for ??
            Butterfly_1
            Butterfly_2
        next ??
    next ??
next Stage
```

First Alternative

We will investigate two different alternative choices of order for the computations of the butterflies in the inner loop.

Having two concurrent butterflies means that we must generate two indices k in parallel: k_1 and k_2.

First alternative: execute butterflies p and $p+N/4$ concurrently.

Second alternative: execute p and $p+N_s/2$ concurrently in the first stage while p and $p+N_s$ are executed concurrently in the following stages.

Third Design Iteration

In this step we collect index computations into a process, *Addresses*.

The behavior of this process depends on the chosen index generation scheme. We also specify the behavior of the input and output processes in more detail by partitioning them into the processes *Memory*, *Memory_Read*, and *Memory_Write*.