Doctoral Thesis

High precision 1024-point FFT processor for 2D object detection

Author(s):
Wosnitza, Matthias Werner

Publication Date:
1999

Permanent Link:
https://doi.org/10.3929/ethz-a-002064853

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
High Precision 1024-point FFT Processor for 2D Object Detection

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by
MATTHIAS WERNER WOSNITZA
Dipl.-Ing., University of Hannover
born 11 April 1968

GERMAN CITIZEN

accepted on the recommendation of
Prof. Dr. Gerhard Tröster, examiner
Prof. Dr. Peter Pirsch, co-examiner

1999
Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Dr. Gerhard Tröster for his support and confidence in me and my work.

I also would like to thank my associate advisor Prof. Dr. Peter Pirsch for his suggestions and for the meticulous reading of my dissertation.

My special thanks go to my colleague Marco Cavadini for many fruitful discussions and valuable support during the processor’s final layout phase. Without his assistance, punctual ‘tape-out’ for the scheduled MPW-run would hardly have been possible.

Further thanks go to my colleagues Thomas Degen, Markus Uster and Teddy Loeliger, who all enriched my life at the ETH.

I would also like to thank Richard Annen, Patrick Zenklusen and Bruno Widmer for their contributions during their graduate work.

Zurich, March 1999

Matthias Wosnitza
Contents

Kurzfassung iii
Abstract v

1. Introduction 1
   1.1. A Compact and Versatile Object Detection System . . . 1
       1.1.1. General Problems Related to Object Detection in
              Industrial Environments 1
       1.1.2. Additional Requirements by Versatility 3
       1.1.3. System Specifications 4
   1.2. Assessment of Existing Methods for Object Detection 5
       1.2.1. Rotation Invariant Detection Techniques 5
       1.2.2. Frequently Employed Methods for Industrial Ob-
              ject Detection 7
       1.2.3. Conclusion 8
   1.3. Objectives of this Work 9
   1.4. Structure of the Thesis 10

2. The New BST Algorithm for Object Detection 11
   2.1. Derivation of the BST Algorithm . . . . . . . . . . . . 11
       2.1.1. The Normalized Cross-Correlation Function 11
       2.1.2. Impact of Template Shape and Background Texture 12
       2.1.3. Masking Irrelevant Pixel Information . . . . . . . 13
   2.2. Considerations on Real-Time Implementation . . . . . . 16
       2.2.1. Hierarchical Search . . . . . . . . . . . . . . 17
       2.2.2. Tracking . . . . . . . . . . . . . . . . . . 18
       2.2.3. Orthogonal Transforms . . . . . . . . . . . . 18
   2.3. Fourier Domain Computation of the BST Algorithm . 20
       2.3.1. Confronting FFT Algorithms . . . . . . . . . . . . 22
       2.3.2. Transformation of Real-Valued Data . . . . . . . 27
   2.4. Optimized BST Algorithm . . . . . . . . . . . . . . . 28
8. IC-Implementation and Measurements

8.1. Design Strategy ........................................ 127
8.1.1. Virtual Prototyping ................................. 127
8.1.2. Concurrent Multiprocessor Architecture and Pro-
cessor Design ........................................... 128
8.2. FFT Processor Implementation ....................... 130
8.2.1. Process Technology ................................ 130
8.2.2. Restrictions Due to Design Library and Tools ... 131
8.2.3. Signal Distribution ................................ 135
8.2.4. Minimization of Clock Skew ....................... 135
8.2.5. Peak Power Supply ................................ 136
8.2.6. Test Aids ........................................... 137
8.3. Measures of the New FFT Architecture ............. 138
8.3.1. Processor Performance ............................. 138

9. Comparison with Existing Processors .................. 141
9.1. Impact of Advanced IC Technologies ................. 141
9.1.1. Area Reduction by Full-Custom Layout .......... 142
9.1.2. Speed-Up and Area-Reduction by Employment of
       Deep Sub-Micron CMOS Technologies .......... 143
9.2. Interrelation between Die Size and Mantissa Wordlength 148
9.3. Confronting FFT Performance ....................... 150
9.3.1. General-Purpose CPUs ........................... 150
9.3.2. Digital Signal Processors ....................... 152
9.3.3. Dedicated FFT Architectures .................... 152
9.4. Conclusion ............................................ 158

10. Possible Extensions .................................. 161
10.1. Potential Improvements to Automated IC-Design ... 161
10.1.1. Timing-Driven Place and Route ................. 161
10.1.2. Considerations on 'VHDL Design' ............... 162
10.2. MCM Implementation ................................ 163
10.2.1. Off-Chip LM Implementation .................... 164
10.2.2. Multiprocessor MCM (System on an MCM) .... 165
10.3. Hardware Extensions ................................ 165
10.3.1. Programmable Processor Control ............... 166
10.3.2. Space Domain NCCF Computation ............... 166
10.3.3. Computation of Invariant Moments ............ 167
10.4. Additional Applications Requiring Fast FFT ...... 167
10.4.1. Rotation-Invariant Object Detection .......... 168
10.4.2. Synthetic Aperture Radar (SAR) ............... 169
Kurzfassung


In dieser Arbeit wird ein integrierter Prozessor vorgestellt, der als wesentliches Verarbeitungselement eines in einer parallelen Arbeit konzipierten Multiprozessorsystems die Detektion von Objekten innerhalb eines Bildstroms in Echtzeit ermöglicht.

Im ersten Teil der Arbeit werden die bestehenden Systemanforderungen analysiert und die normierte Kreuz-Korrelations-Funktion (NKKF) als stabiles und zuverlässiges Detektionsverfahren favorisiert. Zur Eröffnung eines weiten Anwendungsspektrums wird die NKKF dann zum neuen "BST"-Verfahren erweitert. Das "BST"-Verfahren ermöglicht die Detektion von beliebig geformten Objekten auch auf irregulärem Bildhintergrund.

Anschließend wird das dedizierte Multiprozessorsystem zur Echtzeitverarbeitung des "BST"-Verfahrens kurz motiviert. Da zur Reduktion der erforderlichen Rechenleistung das "BST"-Verfahren im Fourier-Bildbereich ausgeführt wird, besteht der eigentliche Mehrprozessorteil aus mehreren identischen FFT-Prozessoren.

Schwerpunkt dieser Arbeit ist der Datenpfad des Prozessors. Wesentliche Zielsetzungen für den Datenpfad-Entwurf bestehen in der Forderung nach hohem Datendurchsatz, präziser Arithmetik und effizienter Ausnutzung der implementierten Resourcen. Mittels Resourcenparalleler Verarbeitung, in Verbindung mit stark gepipelten Daten- und Kommunikationspfaden wird ein Datendurchsatz von 1.2 GByte/sec erzielt. Eine effiziente Ausnutzung der arithmetischen Resourcen wird durch dedizierte Registerfiles gewährleistet, die die Daten innerhalb des Datenpfades reorganisieren. Um eine hohe Rechengenauigkeit mit einem im Hinblick auf eine VLSI-Integration vertretbarem Aufwand zu erreichen, ist eine spezielle Hybrid Block-Exponent (HBE) Arithmetik implementiert. Der durchschnittliche relative Quantisierungsfehler liegt damit deutlich unter 0.01%. Verglichen mit einer gleichgenauen fixed-point Arithmetik ermöglicht die neue HBE Arithmetik eine Flächenreduktion des Prozessor-Layouts um 71%.


Da die FFT sehr häufig in der Signalverarbeitung eingesetzt wird, eröffnen sich für den Prozessor zahlreiche weitere Einsatzgebiete. Die hohe Verarbeitungsgeschwindigkeit, die beschleunigte Transformation reellwertiger Daten, die hohe Rechengenauigkeit sowie die Multiindexesorfähigkeit zeichnen den Prozessor insbesondere für Applikationen in den Bereichen Radar, Echo-Cancellation, Image-Restoration, medizinische Diagnostik sowie andere Echtzeitanwendungen aus.
Abstract

Driven by the steady increase in automation, demand for efficient systems of object recognition and localization keeps growing. Automated manufacturing and quality controls (visual inspection) and autonomous robots (robot-guidance) are some representative examples of machine vision applications. For each of these, the objective is to enable machines to visually recognize their environment.

An integrated processor is presented in this work. The processor represents the major processing unit of a multiprocessor system for real-time object detection within an image stream.

In the first part of this work, existing requirements on the object detection system are analyzed and the normalized cross-correlation function (NCCF) is proposed as a stable and reliable similarity measure. To cover a large field of applications, the NCCF is then extended to the new “BST” technique. The “BST” technique allows for the detection of arbitrarily shaped free-form templates, even on irregular backgrounds.

Next, the motivations behind the development of the dedicated multiprocessor system for real-time processing of the “BST” technique are given briefly. Since the underlying computational requirements are reduced by computing the “BST” technique in the Fourier domain, the system’s actual multiprocessor part consists of multiple identical FFT processors.

As the multiprocessor system is outlined, further requirements on the embedded FFT processor are investigated and a new processor architecture, which is dedicated to Fourier domain correlation of 2D data sets with sizes up to 1024 × 1024 is proposed. In contrast to existing approaches, modes for elementwise matrix multiplication as well as real-valued FFT pre- and post-processing tasks are implemented in addition to the actual FFT functionalism. With this, the processor supports all the computationally intensive low-level tasks required for Fourier domain correlation.

The main focus of this work is on the processor’s data path. Major objectives on data path design are high throughput, very precise
FFT results and efficient utilization of implemented resources. By means of resource-parallel computation, in combination with aggressively pipelined arithmetic units and communication paths, a data throughput of 1.2 GByte/second is achieved. Efficient resource utilization is realized by means of special purpose registerfiles reorganizing selected data within the processor's data path. To achieve high precision at moderate costs, a dedicated hybrid block exponent (HBE) arithmetic is implemented. As a result, the average quantization error is clearly lower than 0.01%. Compared to fixed-point arithmetic with this degree of accuracy, the processor's die size is reduced by 71% through implementation of HBE arithmetic.

Feasibility and operationalism of the proposed processor architecture was proven by a corresponding prototype design based on commercially available 0.5μm three metal-layer CMOS technology. The design contains 2.1 million transistors filling an area of 167mm². Operating at 66MHz clock frequency the processor computes a complex 1024-point FFT within 80μs. At 3.3V power-supply, the processor consumes 1.9W. The proposed architecture thus achieves a clear improvement compared with previous FFT processor designs in regard to the precision-to-computation-time ratio.

Since the FFT is commonly employed in signal processing, the proposed FFT processor can be used for a vast range of additional applications. Features such as large processing power, support for accelerated transformation of real-valued data, high arithmetic precision and multiprocessor support make the processor the ideal choice for applications as diverse as radar, echo-cancelation, image restoration, medical diagnostics as well as other real-time applications.
1. Introduction

The availability of powerful systems for object detection* has become extremely important due to the emerging significance of machine vision in general robotic applications. In particular, object detection systems are increasingly entering the domains of product inspection, process surveillance and quality control. Optical parts inspection and machine parts identification are some representative examples. However, up to now, only a few systems for object detection have existed and, even so, are either restricted to a certain target application (little versatility) or are voluminous and hence expensive.

1.1 A Compact and Versatile Object Detection System

Due to the emerging need for object detection systems in industrial environments, this work concerns the realization of a scalable and very versatile system fitting the demands of today’s inspection and surveillance applications.

1.1.1 General Problems Related to Object Detection in Industrial Environments

In practical situations, the visual appearance of an object embedded into the camera scene may differ from the corresponding template representation due to different types of distortion. If such a difference exists, actual object 'recognition' becomes significantly more difficult. To allow for reliable object detection, we must employ a detection procedure which is invariant to these distortions. Alternatively, we might

*Object detection is concerned with the determination of the presence or absence of an object supposed to be in an image.
try to avoid them. In the following, we give a brief overview of the repre-
sentative types of distortion which are most frequently encountered
in industrial and general image processing.

**Variation in Lighting Conditions**

Variation in lighting conditions can be caused by diffuse light and de-
fective or exchanged lighting fixtures, respectively. In addition to these
environmental influences, components of the object detection system
itself (e.g., thermal noise of the camera and frame-grabber amplifiers)
also contribute to a fluctuation in subsequently processed scene illu-
mination. Since both contrast and brightness are affected, most image
processing, and particularly object detection, procedures are very prone
to changes in scene illumination.

According to the underlying nature, variations in intensity are di-
vided into *additive* and *multiplicative* illumination changes. Referring to
gray-level images, additive changes to image intensity correspond to a
displacement in gray-level representation between template and scene.
Hence, to allow for reliable object detection, variations in scene illumi-
nation must be compensated for, either by appropriate pre-processing
or by choosing an 'intensity-invariant' detection algorithm.

**Noise**

The image quality of gray-level scene representation photographed by a
video camera is distorted by two types of noise.

According to peculiarities inherent to the video camera (e.g., shot
noise of the CCD-sensor and quantization error of the A/D-converter),
the original scene representation is usually overlaid by additive *Gaussian
noise*. In addition, 'salt and pepper' noise may occur due to defects in
data transmission or electronic components.

**Blur and Geometric Distortion**

*Blur* is most commonly due to inappropriate focus settings and hence
can be eliminated for most cases.

However, imperfections in the design and assembly of cameras cause
a distorted reproduction of the photographed scene. For this, the most
relevant types of distortion are *radial lens distortion*, *decentering dis-
tortion* and *thin prism distortion* [Rechsteiner 97]. Since each camera
system will show an individual combination of these distortion types,
proper compensation must be based on corresponding calibration.
In conclusion, when severe geometric distortions prevent reliable object localization, these distortions must be compensated for by dedicated pre-processing prior to the actual object detection procedure.

**Scaling, Rotation and Perspective Distortions**

For most industrial applications, we can assume constant object size throughout all localization operations. Based on this assumption, a variation in scale can only occur due to different distances between the camera and the object under detection. Consequently, the requirement for a scale-invariant detection technique can be dropped by fixing the relative positions of the camera and the monitored search area and also by photographing the reference template under the same conditions. By doing this and by choosing an appropriate angle between the scene and the camera axis, the impact of perspective distortion on detection performance is also eliminated.

However, the problem of potentially rotated objects is much more difficult to solve. While some applications naturally guarantee constant object orientation (e.g., chip positioning), extra effort must be made for others. For these, one can choose for either additional mechanical positioning systems or rotation-invariant object detection algorithms.

**1.1.2 Additional Requirements by Versatility**

Object detection in industrial environments (e.g., optical parts inspection at assembly lines) places a variety of requirements on corresponding system realizations.

Often, the object of interest is characterized by pronounced structural information which might be the only measure by which to indicate the object's exact position and orientation. This is particularly the case for turnery parts. For those situations, a maximum of the object's structural information must be preserved in order to allow for a reliable and exact localization. As a result, processing of high-resolution gray-scale image material is a must.

In reference to automated manufacturing, the majority of workpieces (e.g., tools, electronic components, etc.) is of arbitrary shape or even with holes in the structure. These objects are commonly referred to as 'free-form templates'. In regard to free-form templates, the main problem is that the actual object representation is incidentally distorted by the current base (e.g., assembly line). Targeting reliable object localization, this effect must be properly compensated for.
Further, referring to 'assembly line' production, corresponding manufacturing processes are often characterized by short production times per item or by moving objects. These circumstances require the object detection system to process a continuous image stream, i.e., to operate in 'real-time'. And finally, the object detection system must be compactly and economically realized in order to allow for relevant sales and efficient application.

1.1.3 System Specifications

Recalling the essential application requirements given in section 1.1.2, the new object detection system must be compact and versatile, and must allow for the reliable detection of arbitrarily-shaped objects (free-form templates) even on very irregular backgrounds.

Addressing the required gray-scale image representation, we choose an underlying 'color-depth' of 8b in order to differentiate between 256 gray levels. To combine the 'high resolution' requirement with economical state-of-the-art sensor technology, we specify the system's input image geometry to 1024 × 1024 pixels. This resolution, in combination with 8b gray-level representation, preserves all structural and textural information for almost all templates. However, targeting maximal versatility, the system must also be adaptable to lower sensor resolutions such as 512×512 and 256×256, respectively. Apart from this, in order to maintain the usual template to search area ratio, we fix the corresponding template size to 128×128 pixels. However, template dimensions may vary with the actual application. Hence, the object detection system must show sufficient flexibility to support different template sizes.

To meet the 'real-time' requirement for a large variety of applications, the object detection system must show a computational power allowing for sustained processing of 1024 × 1024 8b-gray-scale camera images at a rate of at least 10 frames/second. Taking the continuous advance in sensor technology into account, the system architecture must be scalable in order to allow for the accommodation of the larger computational rates that will be required to process high definition images generated by forthcoming sensor resolutions.
1.2 Assessment of Existing Methods for Object Detection

A variety of methods for object detection currently exist: they can be distinguished mainly in terms of computational complexity and robustness in regard to image distortions such as varying illumination conditions, signal noise and blur and geometrical lens distortions (cf. section 1.1.1). These 'stability characteristics', together with the algorithm's sensitivity to uniform scaling and the rotation of objects can be used to describe the detection performance of a given method. As a further distinguishing feature, some algorithms allow different sizes of template and search area while others (e.g., correlation) can only compare regions of equal size, thus requiring the isolation of search windows from the search area.

A detailed overview of image detection and registration techniques is given in [Pratt 91]. In the following, we give a brief survey of existing object detection methods. For this, we classify the algorithms into two main groups: algorithms for rotation-invariant localization of templates and methods that are sensitive to rotation. While algorithms of the first class allow for maximal versatility, they also impose very large computational requirements. Consequently, rotation-variant techniques can be implemented more reasonably and are hence more frequently employed in industrial object detection applications.

1.2.1 Rotation Invariant Detection Techniques

Invariant Transforms

Several algorithms used in rotation-invariant object detection are based on circular-harmonic functions ([Elizur 91]) or phase-only correlation ([Leclerc 89]). In addition, Fourier-Mellin (FM) descriptors are invariant to translation, rotation and changes of scale and intensity [Lejeune 89]. A further invariant method dealing with gray-level images is given by the intensity to phase coding [Sadovnik 93]. However, cost-efficient hardware realization of these methods is prevented by their computational requirements. In addition, the FM transform is not shift-invariant and depends on the origin of its polar coordinate system. To get a centroid, an object of interest must first be isolated from the surrounding background, hence requiring an additional segmentation algorithm. This is also required for circular-harmonic expansion, since it
sacrifices translational invariance [Wang 94, Zetzsche 89]. While time-consuming operations can be addressed by optical\(^1\) implementation, a second point related to the algorithms referred to above is lack of stability [Tang 91]. A detailed view of the prospects and problems of optical correlators is given in [Washwell 94].

Since 'optical' methods are not suitable for real-time VLSI implementation, other algorithms for rotation-invariant feature extraction have evolved.

**Ring Projection Algorithm**

A ring-projection algorithm dedicated to rotation-invariant pattern matching is presented in [Cheng 90, Tang 91]. The proposed algorithm basically computes histograms based on ring-shaped trajectories and achieves good results for binary images. Since background pixels will distort the similarity measure, the algorithm will be less successful for gray-scale images. Implemented on a multiprocessor system, the algorithm induces large communication overhead and hence requires dedicated array architectures ([Cheng 89, Cheng 91]) with little flexibility.

**Moment Invariants**

Alternatively, the method of invariant moments is a classical approach to extracting scale-, translation- and rotation-invariant features from 2D images [Hu 62, Khotanzad 87]. Here, the similarity measure is constructed on a set of non-linear functions which are defined upon the moments. Though moment invariants have successfully been employed for aircraft identification [Dudani 77], the computational requirements are still too large when targeting a compact real-time system. An efficient reduction method is achieved by performing gray-scale to binary conversion. This reduces computational and storage requirements and allows the use of logical rather than arithmetical operations. In this case, treating gray-level images as bit planes might be advantageous.

\(^1\)Due to the large computational requirements, invariant transformations (e.g., Fourier-Mellin) are preferably realized by means of dedicated transform lenses, i.e., optically.
1.2. Frequently Employed Methods for Industrial Object Detection

For non-rotation-invariant pattern matching, the most frequently used methods are the correlation algorithm and the matched filter. While these methods are reliable, they are also characterized by a comparatively large operation count. One approach to lowering the computational requirements is to reduce actual image information to a certain abstraction level. Similarity measures are then defined through low-level features such as gradient magnitudes or edges [Chou 90].

Edge Oriented Procedures

In industrial environments, a relevant portion of detection tasks concerns workpiece localization. In cases where workpieces are mainly characterized by shape rather than by texture, edge-oriented detection procedures are a reasonable choice in terms of computational requirements. By 'edge extraction', actual object representation is reduced to one of its characteristic features – its outline.

Abstraction to edge information is usually obtained by filtering, e.g., Sobel-kernel [Spirig 96]. The filter outputs are then compared by means of a cross-correlation or a matched filter.

The achieved reduction in object information allows for reasonable compensation of possible scaling and rotation effects. Multiple filters for representative directions can be applied to allow for rotation-invariant object detection [Zetzsche 89]. Alternative methods of obtaining the desired invariance property are given by affine transformations (e.g., the Hough transform) or other appropriate 'coding techniques' (e.g., the normalized histogram of edge orientations).

Although computational requirements can be drastically reduced by abstraction to edge level, a lot of image information is discarded. Hence, in general, methods based on abstracted features (e.g., edges) almost always target one specific application which permits this information loss. Through simulations we have proven that those methods often fail when operating on complex gray-scale templates. This effect is also confirmed by [Aschwanden 93], who observed a dramatic decrease in detection performance when image quality is reduced by blur or geometric distortion.

In addition, a significant portion of objects cannot be characterized by a definable contour. Although these objects do not have a clearly defined boundary (whether due to intrinsic fuzziness or to lighting con-
ditions), the majority of them can be identified according to their characteristic color and texture distribution, respectively.

**Area-Based Correlation Techniques**

To locate objects that are more likely to be identified by their structural and textural information, processing of actual image information is required. As a result, image contents cannot be reduced by binarization or filtering; processing of the pixel's intensity information (e.g., brightness and color) is required. In these cases, area-based similarity measures such as the matched filter or correlation techniques must be used.

A common characteristic of correlation techniques in image processing is that they compute a measure of similarity by comparing two 2D data sets of equal size. The actual similarity measure is then computed by arithmetic combination of corresponding pixels.

A comprehensive comparison of correlation techniques is given in [Aschwanden 93]. Referring to stereo-vision and image compression applications (i.e., block matching), two correlation techniques are most commonly employed: 'Sum of Absolute Differences' (SAD) and 'Sum of Squared Differences' (SSD). Both methods are marked by their simplicity in computation. In particular, the SAD can be computed exclusively by means of addition and subtraction. However, as proven in [Aschwanden 93], these techniques are very sensitive to variations in illumination and fail even under minor changes of environmental lighting conditions.

Considerably better invariance concerning variations in intensity can be achieved by employing the normalized cross-correlation function (NCCF). According to the underlying arithmetic procedure, the NCCF is invariant against multiplicative variations in image amplitude. However, it is sensitive to additive changes in illumination. Nevertheless, invariance in regard to multiplicative and additive variations in lighting conditions can be achieved by means of the variance normalized correlation. In this procedure, the mean values of both local search window and template are eliminated prior to the actual NCCF.

**1.2.3 Conclusion**

In spite of the large number of detection methods that have evolved, there is still no generally applicable procedure covering all situations encountered in industrial environments [Aschwanden 93]. As a rule of
thumb, the better the detection performance of a certain algorithm, the larger the corresponding computational requirements. In particular, as discussed in section 1.2.1, reliable rotation-invariant methods applicable to a broad application field are characterized by large computational requirements and are therefore unsuited to real-time hardware implementation. On the other hand, methods with reduced operation counts are more or less fitted to one specific application.

However, in almost all cases, computational power is limited by constraints on cost and system size. A trade-off between computational rates and detection performance is thus required in order to select the most appropriate localization technique for a given application.

As a result, rating the characteristics of the algorithms discussed so far with regard to the requirements given in sections 1.1.1 and 1.1.2, we acknowledge the NCCF as most suited to the required object detection system. The NCCF qualifies as a very reliable measure that is robust against surface reflections, noise and various other image distortions [Silver 87, Rothacher 91, Burt 82, Aschwanden 93]. In addition, the NCCF makes few requirements on the underlying gray-scale image material.

The NCCF is however sensitive to rotation. Even small rotations of a few degrees can reduce the peak of the cross-correlation function to noise level. Nevertheless, we have shown by extensive simulations that angle deviations up to ±5 degrees are tolerable, depending on the template and background pattern. However, if full rotation-invariance is required, a library of stepwise rotated templates must be employed. We also have to modify the original NCCF in order to allow for the reliable detection of free-form templates.

1.3 Objectives of this Work

This work is marked by two major goals in reference to the realization of the targeted object detection system. The first is to construct a robust detection technique from the favored NCCF algorithm which allows for both reliable free-form object detection and efficient hardware implementation.

Concerning actual system implementation, the demands for real-time processing of high resolution images and for scalable system performance almost naturally lead to a multiprocessor architecture. In order to achieve the most compact implementation, the multiprocessor system must be powered by dedicated high-performance processor...
elements. With this, the second major goal of this work is to realize a dedicated processor element, powering a compact object detection system allowing for precise localization of arbitrarily sized\(^1\) free-form templates in 1024 × 1024 8-bit-gray-scale camera images at a rate of at least 10 frames/second.

To achieve maximum system performance, massive research effort must be put into the design and coordination of system and processor architecture. To minimize design time, system architecture and processor elements are advantageously designed in parallel. Accordingly, the main focus of this thesis is on the design and VLSI implementation of the dedicated processor element, whereas concepts for relevant multiprocessor system architectures are comprehensively discussed in [Cavadini 99].

1.4 Structure of the Thesis

In Chapter 2 we extend the NCCF to a new detection technique which allows for the detection of arbitrarily-shaped objects, even against irregular backgrounds, with high stability and reliability.

Targeting real-time implementation of this new algorithm, we derive a scalable SIMD multiprocessor architecture powered by dedicated processor elements in Chapter 3. The architecture of the processor element allows for fast and efficient computation of all the low-level tasks of the new detection algorithm and is discussed in Chapter 4.

In Chapter 5, we concentrate on the processor’s processing part – the constant throughput data path with native support for complex arithmetic. In Chapter 6, we discuss relevant arithmetic concepts allowing for both high-precision computation and area-efficient VLSI implementation. In the course of the underlying discussion, we derive a statistical model for measuring the impact of finite wordlength effects on the precision of computed data path results. To achieve maximum precision at moderate hardware costs, we implement a dedicated hybrid arithmetic into the data path architecture in Chapter 7.

In Chapter 8, we give a brief overview of the actual processor-implementation and present the corresponding processor measures. A comparison with existing processors is presented in Chapter 9. Suggestions for future work on the processor design are discussed in Chapter 10.

\(^{1}\)However, at least 128 × 128 templates must be processable!
2. The New BST Algorithm for Object Detection

In this chapter, we extend the reliable normalized cross-correlation function (NCCF) to the new "blue-screen" correlation technique (BST). The proposed BST technique allows for the use of variably sized free-form gray-level templates (including holes), thus broadening the field of potential applications. In addition, compared to the original NCCF, detection performance and reliability are significantly improved.

By computing the BST algorithm in the Fourier domain, the computational requirements are drastically reduced and data dependencies within the processing flow are completely eliminated. Concerning hardware implementation, we discuss a selected set of suitable Fourier transform techniques. Finally, we present the optimized computation flow for the BST algorithm.

2.1 Derivation of the BST Algorithm

2.1.1 The Normalized Cross-Correlation Function

A mathematically-derived algorithm for computing a similarity measure between two signals is the (unnormalized) cross-correlation function (UCCF):

\[ r_u(x, y) = \sum_{v=0}^{v_{Max}} \sum_{u=0}^{u_{Max}} t(u, v) \cdot s(x + u, y + v), \quad (2.1) \]

where \( t \) and \( s \) denote template and corresponding search window, respectively.

To locate an object, a search window of template size (e.g., \( u_{Max} \times v_{Max} \)) must be isolated from the search area and correlated with the template for every possible object position. The object’s position is
then identified by appropriately thresholding the \(r_u(x, y)\), i.e., by finding their maxima.

However, the application of the UCCF defined by Eq. (2.1) is not suitable for mean-value afflicted images since the best match is assumed to be at the location of the brightest spot [Aschwanden 93]. Nevertheless, the correlation result can be improved by normalizing the UCCF by the geometric mean of the energies of the search window \(s\) and template \(t\). By doing so, the normalized cross-correlation function (NCCF) is given as

\[
r(x, y) = \frac{r_u(x, y)}{\sqrt{C \cdot \sum_{v=0}^{U_{max}} \sum_{u=0}^{U_{max}} s^2(x + u, y + v)}},
\]

with \(C = \sum_{v=0}^{U_{max}} \sum_{u=0}^{U_{max}} t^2(u, v)\).

According to its structure, the NCCF is simple to implement and allows for efficient parallel computation.

### 2.1.2 Impact of Template Shape and Background Texture

Although the NCCF shows good results for many applications, the detection of non-rectangularly shaped templates on irregular backgrounds is almost impossible. This can easily be shown by looking at the definition of the NCCF (Eq. 2.2). Here, the computation of both the UCCF (nominator) and the normalizing term (denominator) is based on 2D sums, each covering a rectangular area.

Often, however, the object to be found only fills a small part of the rectangular template area (cf. Fig. 2.1). In this case, many pixels of the rectangularly photographed template do not represent any relevant object information. In general, the arrangement of these "irrelevant pixels" depends on the underlying template geometry.

Referring to the free-form template "padlock" shown in Fig. 2.1, the set of irrelevant pixels contained within the rectangular template area is marked by a chequered pattern. In practice, however, these irrelevant pixels will represent the background on which the template (e.g., padlock) was photographed on. This background information is arbitrary and mostly does not match the background of the corresponding search area. However, template "background" will be included in the NCCF result and, depending on its texture, distorts the correlation peak. Under these circumstances, the NCCF will give a false object position.
To overcome this problem, a new algorithm based on the reliable NCCF is developed. The new algorithm masks (i.e., sifts out) irrelevant pixels, thus leading to correct correlation results independent of template shape and background pattern.

Extensive simulations based on representative image material have clearly shown that by sifting out background template pixels during the correlation process, the precision and reliability of the correlation result can be improved significantly. Since this procedure has an analogy with the "blue-screen technique" (BST) used in television, we have adopted this notation.

2.1.3 Masking Irrelevant Pixel Information

This section describes the process of masking irrelevant pixel information. First, we concentrate on space domain computation of the BST algorithm applied to gray-level images.

Considering Eq. (2.2), the exact value of \( r(x, y) \) is often of minor interest but the position of the (absolute) maxima over all \( r(x, y) \) is required. Hence, Eq. (2.2) can be squared to avoid the costly square-root operation. Furthermore, we can eliminate the term \( C \) since it is independent of the position variables \( x \) and \( y \) and thus is only a scaling factor varying with the template. We can thus simplify Eq. 2.2 to

\[
r'(x, y)^2 = \frac{\left( \sum_{u=0}^{u_{\text{Max}}} \sum_{v=0}^{v_{\text{Max}}} t(u, v) \cdot s(x + u, y + v) \right)^2}{\sum_{u=0}^{u_{\text{Max}}} \sum_{v=0}^{v_{\text{Max}}} s^2(x + u, y + v)}.
\]

(2.3)
Here, masking can be separately performed on both the nominator and denominator.

**Masked UCCF Computation**

The nominator represents the UCCF of template \( t \) and search area \( s \). Regarding space domain processing of the UCCF on gray-level images, irrelevant template pixels can be ignored by setting their gray-level to \( '0' \), thereby eliminating their influence on the correlation result. This can be achieved by either choosing a suitable background when sampling (e.g., photographing) the template or by using adequate pre-processing techniques. Thus, suppression of irrelevant template pixels within the computation of the UCCF can be realized quite easily. However, simulations have clearly shown that masking of irrelevant template pixels must also be performed on the denominator of Eq. (2.3) when high-precision object detection is concerned. This complicates the situation as is shown in the following.

**Masked 'Energy Term' Computation**

The denominator of Eq. (2.3) represents the 'image energy' of the rectangular search window at position \((x, y)\). For space domain computation of the energy term, only those pixels \( s(x+u, y+v) \) whose corresponding template pixels \( t(u,v) \) contain relevant object information have to be squared and accumulated. This requires a data-dependent processing scheme since prior to the accumulation step, pixels have to be classified as “template” or “irrelevant background” pixels. This increases the already large computational requirements of the “blue-screen technique”, since efficient techniques for computing the energy term as reported in [Aschwanden 93] cannot be applied.

As a result, we must derive an alternative method for the computation of the energy terms if we are aiming for efficient hardware implementation.

In reference to this computation method, we achieve large efficiency in terms of system modularity and complexity, if both UCCF and energy terms can be computed on the same hardware. In this case, we can either subsequently compute the nominator and denominator of Eq. 2.3 on one 'system' or in parallel by two identical 'systems'. As a result, the alternative method must compute the energy terms by means of a “correlational” procedure.
The computation of masked energy terms is based on summing up the squares of selected search area pixel quantities. Consequently, we can perform the required “correlation” by means of a binary* replica of the template ($t_{BIN}$) and a squared copy of the search area ($s_{SQR}$). For this, we compute $t_{BIN}$ and $s_{SQR}$ as follows:

$$t_{BIN}(u,v) = \begin{cases} 
1 & \text{if } t(u,v) \text{ contains relevant object information} \\
0 & \text{otherwise}
\end{cases}$$

$$s_{SQR}(x,y) = s(x,y)^2$$

In the presence of sufficient gray-level differences between ‘irrelevant background’ and actual ‘object describing’ pixels, $t_{BIN}$ can be computed automatically (e.g., by appropriate thresholding). However, in

*The pixels of this binary template are either ‘0’ or ‘1’

(a) Binarized template.  
(b) Detail of squared search area.

(c) "Correlation track": Similar to 2D correlation, the binarized template (“kernel”) is moved over the squared copy of the search area.

Figure 2.2: Space domain computation of masked energy terms by correlating binarized template and squared search area.
cases where a binary template representation cannot be automatically obtained, manual processing of the template’s photograph is required (e.g., drawing tool). An example of corresponding template and ‘binary template’ representations is given by Figs. 2.1 and 2.2(a), respectively.

The actual ‘correlation’ between the binarized template and the squared search area is illustrated in Fig. 2.2. As shown, the binary replica of the template is shifted across the squared search area. Thus, the UCCF can also be used to compute energy terms.

### 2.2 Considerations on Real-Time Implementation

As discussed in the previous section, straightforward space domain computation of the BST algorithm is very costly. Repetitive extraction of search windows, which are then correlated with the template, induce computational requirements of order $O(N^2 \cdot n^2)$, with $N$ and $n$ giving the border length of search area and template. As shown in Table 2.1, locating a $128 \times 128$ free-form template in a $1024 \times 1024$ search area by the BST algorithm requires $52.7 \cdot 10^9$ arithmetic operations for a single frame. Targeting a ‘real-time’ rate of 10 frames/second, the object detection system must sustain a constant computational rate of 527 Giga-operations per second.

Facing today’s technology, these high computational rates can only be achieved with large, expensive computing systems. Thus, substantial reduction is necessary to allow for an efficient hardware realization.

In this section, three methods for reducing the computational re-

<table>
<thead>
<tr>
<th>Task</th>
<th>Subtask</th>
<th>Operation Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCCF</td>
<td>Addition</td>
<td>$(N - n + 1)^2 \cdot n^2$</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
<td>$(N - n + 1)^2 \cdot n^2$</td>
</tr>
<tr>
<td>Energy Terms</td>
<td>Addition</td>
<td>$(N - n + 1)^2 \cdot n^2$</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
<td>$(N - n + 1)^2 \cdot n^2$</td>
</tr>
<tr>
<td>UCCF/Energy Terms</td>
<td>Division</td>
<td>$(N - n + 1)^2$</td>
</tr>
</tbody>
</table>

**Table 2.1:** Computational requirements for space domain BST computation.
requirements are proposed: hierarchical-search, tracking and the indirect computation of the UCCF by means of orthogonal transforms. For the first two approaches, the strategy is to determine rough object positions first with minimal computational effort. In a second step, a more precise search is then performed in the vicinity of these approximate positions. A large portion of computational power is thus saved; this power would otherwise be spent on pedantically verifying 'no-object' locations.

2.2.1 Hierarchical Search

According to the quadratic dependency of geometric size, a first option for decreasing computational requirements is to reduce the spatial resolution of the search area and template by applying a resolution pyramid [Gonzalez 83]. Typical scales for these ‘reduced’ copies are $1:2^k$, hence 1:4, 1:16, etc. (cf. Fig. 2.3).

In general, several dedicated filters, in addition to simple sub-sampling, exist for scaling down resolution. Hierarchical search was successfully implemented in the correlation system reported in [Rothacher 93], where a region of $8 \times 8$ pixels is reduced to a single element referred to as a “moxel”.

However, stability and reliability might be negatively affected if this approach were applied to the BST algorithm. This is so because minimizing computational requirements involves reducing the size of the search area and the template as much as possible during the first approximate search. This massive down-scaling (as $8 \times 8 \rightarrow 1 \times 1$) results in significant information loss. This is particularly the case for large

Figure 2.3: Reduction of spatial search area resolution by means of a resolution pyramid.
high-resolution templates of complex texture and little contrast. However, large information loss implies a large number of potential object positions resulting from the first approximating search process. In the case of real-time processing, computational resources are limited and hence not all of those positions can be precisely verified during the second step. Thus, false positions might be favored and followed further, while actual object locations might by “dropped”.

Related investigations based on representative image material have clearly shown that the maximum possible reduction strongly depends on image structure and thus cannot be fixed with general validity. Potential savings in computational requirements thus become unpredictable.

2.2.2 Tracking

A further approach to determining an approximated object position is given by tracking. To do this, we assume that only a limited number of objects exist within the search area. We also presuppose continuous movement, i.e., objects will enter and leave the image pane only through its borders.

Then, if object velocity is not too high, each object should be located in the vicinity of its corresponding position within the previous frame. In addition, based on its previous trajectory, the position of an object within the next frame can be predicted with sufficient probability. As for hierarchical search, a precise correlation is then performed around the estimated locations.

Although computational requirements can be drastically reduced by tracking, the spectrum of possible applications is quite delimited. A fuzzy computing method for rotation-invariant image tracking based on correlation is proposed in [Wang 94].

2.2.3 Orthogonal Transforms

As discussed, computing the BST algorithm in space domain involves shifting the template over the search area and measuring the similarity between the template and the corresponding search windows. This procedure may include thousands of computationally-expensive window comparisons. Hence, a further very efficient way of reducing computational requirements is to indirectly compute the UCCF and energy terms by means of an orthogonal transform possessing the convolution property. By doing this, the cyclic convolution turns into simple element-wise multiplications of two spectra.
The most popular orthogonal transforms having the convolution property are the Hartley transform, cosine/sine transforms, the Walsh-Hadamard transform, the Fourier transform and several number theoretic transforms. A detailed overview is given in [Ahmed 75, Mitra 93].

**Hartley Transform, Sine/Cosine Transform**

The discrete Hartley transform (DHT) [Bracewell 83] is very similar to the discrete Fourier transform but is completely real-valued according to its kernel $\cos \theta + \sin \theta$. Since the real-valued fast Fourier transform always requires slightly fewer operations than and almost the same overhead as the DHT, the DHT should only be used in special cases [Mitra 93]. Further, the sine-cosine transforms always require more operations than the real-valued FFT counterparts.

**Walsh-Hadamard Transform**

The Walsh-Hadamard transform (WHT) [Ahmed 71] is perhaps the most well-known non-sinusoidal orthogonal transform. Rather than sine and cosine, the WHT kernel consists of square waves and can essentially be computed using addition and subtraction. Though the WHT is very simple to compute, its relevance for transform-based correlation is very limited. According to its properties, the WHT can directly be used only for dyadic correlation [Ahmed 75]. To perform the required cyclic correlation, the transform matrix must be multiplied by a corresponding correction matrix [Zarowski 85]. The additional computational requirements imposed hereby strongly increase with transform length. Because of this, use of the WHT is worthwhile only for lengths up to just 64 (128 under certain conditions). Thus, the WHT is not suited to indirectly computing the BST algorithm on $1024 \times 1024$ images.

In conclusion, we rate the Fourier transform as the orthogonal transform most suitable in terms of computational requirements and versatility to computing the BST algorithm in the transform domain.

---

‡Number theoretic transforms (NTTs) are discussed in connection with FFT computation in section 6.3.6.
2.3 Fourier Domain Computation of the BST Algorithm

The best-known method for efficiently computing correlation is using the convolution theorem and a fast Fourier transform algorithm [Blahut 84]. The discrete Fourier transform (DFT) has been the subject of much investigation and we can therefore refer to numerous results [Cooley 65, Rabiner 75, Mitra 93, Burrus 85].

For Fourier domain correlation, the spectra of search area and template must first be computed. The corresponding Fourier transformation of 2D data is performed according to

\[ V(u, k) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} v(i, j) W_N^{iu} W_N^{jk}, \quad W_N = e^{-j\frac{2\pi}{N}} \]  

(2.5)

where \( v(i, j) \) denotes space domain data and \( V(u, k) \) corresponds to spectrum elements.

**Computation of the 2D-FFT by the Separation Approach**

As an alternative to Eq. (2.5), a very efficient 2D transformation method is given by the separation approach. Looking at Eq. (2.5), we can substitute the ’inner’ sum by a temporary result \( V' \):

\[ V'(u, k) = \sum_{i=0}^{N-1} v(i, j) W_N^{iu}. \]

We can thus formulate the computation procedure for Eq. (2.5) as follows:

\[
\text{FOR } u=0 \text{ TO } N-1 \text{ DO} \newline
\quad \text{FOR } k=0 \text{ TO } N-1 \text{ DO} \newline
\quad \quad V'(u, k) = \sum_{i=0}^{N-1} v(i, j) W_N^{iu} \newline
\quad \quad V(u, k) = \sum_{j=0}^{N-1} V'(u, k) W_N^{jk} \newline
\quad \text{END} \newline
\text{END}
\]
2.3. Fourier Domain Computation of the BST Algorithm

![Diagram of Initial Data Set, Temporary Matrix, and Final Spectrum](image)

Figure 2.4: Computation of the 2D-FFT by means of the separation approach: First, all rows are successively 1D-transformed, resulting in the temporary matrix $V'(u, k)$. Transforming all columns of this temporary matrix yields the required 2D spectrum.

Here, $V'(u, k)$ and $V(u, k)$ are subsequently computed within the two nested loops. However, $V'(u, k)$ and $V(u, k)$ can also be separately computed by a pair of nested loops:

```c
/* Rows */
FOR k=0 TO N-1 DO
    FOR u=0 TO N-1 DO
        $V'(u, k) = \sum_{i=0}^{N-1} v(i,j) W_N^{iu}$
    END
END

/* Columns */
FOR u=0 TO N-1 DO
    FOR k=0 TO N-1 DO
        $V(u, k) = \sum_{j=0}^{N-1} V'(u, k) W_N^{jk}$
    END
END
```

The 2D data set is sectioned into rows and columns by this procedure. In a first step, all rows are successively 1D-transformed, resulting in the temporary matrix $V'(u, k)$. Subsequently, all columns of this temporary matrix are 1D-transformed, yielding the required 2D spectrum. As a result, the 2D-FFT is computed by means of 1D-FFTs (cf.
When both the padded template and the search area are transformed, Fourier domain correlation is performed by elementwise multiplication of the spectra of the template and the search area. Finally, the UCCF results \( r_u(x, y) \) are obtained by inverse transformation of this product matrix. Data flow for Fourier domain correlation is shown in Fig. 2.5.

Compared to the space domain, computational requirements are thus significantly reduced from order \( O(N^2 \cdot n^2) \) to \( O(N^2 \log_2 N) \). In addition, the dependency between computational requirements and template size is completely eliminated. This is because for elementwise multiplication, both spectra must be of equal size and thus the smaller template is padded to fit the search area [Rabiner 75]. Hence, a variation in template size only manifests itself in the number of padded data.

While computational requirements are drastically reduced by employing the Fourier transform, we must note that the underlying numeric set is extended to complex numbers. Since complex arithmetic is needed requirements on storage and bus-width increase, thus affecting hardware implementation.

### 2.3.1 Confronting FFT Algorithms

Algorithms for efficiently computing the DFT are named fast Fourier transform (FFT) algorithms. Many techniques for computing the discrete Fourier transform have evolved, each with different advantages and each to be preferred above others in different situations. Basically, there are two strategies. One is to turn the DFT into a convolution,
2.3. Fourier Domain Computation of the BST Algorithm

which is then computed by means of efficient convolution algorithms. The second strategy is to turn a 1D transform into a 2D transform, which is easier to compute [Blahut 84]. A detailed overview on FFT algorithms is presented in [Mitra 93, Blahut 84].

Cooley-Tukey FFT

For cases where transform length \( N \) can be expressed as a power of \( r \), a very efficient FFT algorithm is developed by Cooley and Tukey [Cooley 65], which turns a 1D into a 2D transform. According to the chosen \( r \), the resulting techniques are named radix-\( r \) algorithms. In terms of hardware realization, \( r \) is usually a power of two and hence radix-2, -4 and -8 algorithms are of increased significance. However, considerations in choosing the right radix are fairly complex and depend on the current implementation.

Operation counts for the radix-2 Cooley-Tukey algorithm are given in Table 2.2. Besides straightforward computation, figures are also quoted in case of optimized computation. Here, complex multiplications are assumed to be performed using three real multiplications and three real additions. Trivial multiplications by \( \pm 1 \) and \( \pm j \) are not counted and symmetries of trigonometric functions are fully used.

Targeting high speed FFT computation, major design aspects are to achieve a FFT architecture suited for VLSI implementation as well as to realize fast external data transfer. In addition, reduction of off-chip bandwidth will simplify the overall communication concept and board design.

For butterfly-oriented architectures with radix \( r \), \( \frac{N}{r} \cdot \log_r N \) butterflies must be computed, each requiring \( 2r \) memory accesses. Hence, to

<table>
<thead>
<tr>
<th>Blocklength ( n )</th>
<th>Native</th>
<th>Fully Optimized(^a)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multiplications</td>
<td>Additions</td>
</tr>
<tr>
<td>256</td>
<td>4096</td>
<td>6144</td>
</tr>
<tr>
<td>512</td>
<td>9216</td>
<td>13824</td>
</tr>
<tr>
<td>1024</td>
<td>20480</td>
<td>30720</td>
</tr>
</tbody>
</table>

\(^a\)Complex multiplications are assumed to be performed using three real multiplications and three real additions.

Table 2.2: Number of real operations for Cooley-Tukey radix-2 FFT [Blahut 84].
minimize bandwidth between arithmetic unit and external memory, the
closed $r$ must be large. In addition, the number of required multiplica-
tions are reduced with higher radices [Blahut 84]. On the other hand,
a large radix restricts the set of composable transform lengths and the
computation of single butterflies becomes more complex. Whereas a
single radix-4 butterfly can be exclusively computed by 16 additions,
non-trivial complex multiplications are required to calculate a radix-8
butterfly. Furthermore, there are actually no computational savings in
higher radix algorithms. Lower radix graphs can be modified to obtain
the same number and type of additions, subtractions and multiplica-
tions as in any higher radix graph [McGee 98]. Because of this, the
optimal trade-off is often seen as deciding for the radix-4 algorithm
[Bidet 95, Mitra 93]. The flow-graph of a single radix-4 butterfly is
shown in Fig. 2.6. Transformation is performed in-place, i.e., the re-
sults $X(i)$ are stored in memory locations of the input values $x(i)$ and
thus no extra memory for storing intermediate results is required.

**Split-Radix FFT**

The split-radix algorithm is a hybrid between radix-2 and radix-4
[Duhamel 84]: the even-indexed elements are indexed as radix-2 while
the odd-indexed elements are indexed as radix-4. With this, indexing is
complicated, but significant savings in computational requirements are
achieved. Actually, operation count is even lower than for any other
known constant-radix FFT, and the split-radix FFT is known to be
 optimum for transform lengths up to and including 16. In addition,
considering the total amount of operations, it is assumed to be optimal
for all transform lengths that are powers of two [Mitra 93].

Although very efficient, we consider the split-radix algorithm to be unsuited for hardware implementation, because complex address-
generation schemes are required and data-dependency is disproportion-
ately larger compared to radix-4.

**Prime-Factor DFT**

For blocklengths $N = N_1 \cdot N_2 \cdot \ldots \cdot N_j$, transformation can be performed by means of mixed-radix algorithms. Generally, the mixed-radix FFT is not very attractive since radix -2, -4, -8 and -16 algorithms have a higher relative efficiency. However, for the special case that the $N_i$ are relatively prime, application of the very efficient prime-factor algorithm (PFA) is feasible.

By applying the PFA, the 1D-DFT is converted into a multi-dimensional DFT. The underlying mapping is based on the Chinese reminder theorem (section 6.3.5). As its particular characteristic, there are no twiddle factors between the dimensions. In addition, concerning small blocklengths, each dimension is transformed using very efficient short-length modules (e.g., Winograd). Because of this, the PFA has a very low operation count compared to other FFT algorithms while retaining a flexibility that allows a wide variety of transform lengths rather than just those of powers of two. More details on the PFA algorithm can be found in [Mitra 93, Blahut 84].

Although an efficient pipeline architecture for computing the prime factor FFT is proposed in [bun 92], multiple butterfly structures are required for VLSI implementation of the PFA. However, since its low operation count is very close to the split-radix FFT, we give an example for efficient PFA computation in the course of discussing finite field arithmetic (section 6.3.6).

**Winograd Short-Length Modules**

Winograd short-length modules can be used for efficiently computing the DFT in cases where transform length is a small prime or a power of it. The most popular blocklengths are 2, 3, 4, 5, 7, 8, 9 and 16. Winograd short-length modules are based on Rader’s permutation, which turns a DFT into a cyclic convolution and then employs Winograd’s efficient short-length cyclic convolution algorithms [Mitra 93]. Winograd algorithms minimize the number of required multiplications, usually at the expense of an increased number of additions. Furthermore, all dimensions are simultaneously processed.

The Winograd large FFT is a combination of Winograd short-length modules and can be used whenever transform length is composed of relatively prime factors (cf. PFA). The Winograd large FFT is a very
efficient method, presupposed that efficient Winograd short-length modules exist for all dimensions. On the other hand, it is also more intricate to implement since the repetitive loops are much larger. In addition, the Winograd FFT must be implemented with careful attention to scaling noise [Blahut 84].

Sectioned Convolutions

In practice, Fourier domain convolution is often used for filtering an input data sequence. Concerning speech and radar signals, the input sequence $x(n)$ is much longer than the employed filter kernel $h(n)$, i.e., $N \gg M$, and may even be of infinite duration. In such cases where the resulting sequence length $N + M - 1$ is too large or infinite, it is impractical to compute the DFT directly because large amounts of memory are required and large latency is imposed. To overcome these problems, two techniques evolved to section the larger sequence and compute partial results that can be pieced together to form the desired output sequence. Both methods, the "Overlap-Add" and "Overlap-Save" technique, allow for the starting of the convolution before the entire longer sequence is available: the methods are discussed in [Rabiner 75, Jackson 89].

On the other hand and in reference to the underlying correlation problem, we have clearly shown that computational requirements are larger compared to direct computation for both techniques. This is mainly due to the fact that elements located in overlapping areas are transformed twice since they are part of two independent sections. In the case of correlating a 1024-point sequence with a 128-point kernel, direct computation by a single radix-2 1024-point FFT/iFFT requires approximately 31,744 complex operations. On the other hand, to employ a 512-point FFT, the input sequence must be broken up into at least three sections requiring a total of 43,008 complex operations. Finally, eight sections are required to allow for the use of 256-point FFTs, imposing a computational load of 51,200 complex operations. Because of the increase in computational requirements, sectioned convolutions are unsuited to our target application.

To recap, we prefer the Cooley-Tukey FFT with regard to VLSI implementation because of its versatility and regular structure.
2.3. Fourier Domain Computation of the BST Algorithm

2.3.2 Transformation of Real-Valued Data

Although computational requirements are drastically reduced by performing the correlation in the Fourier domain, further savings can be achieved. Recalling that both the search area and the template are real-valued (8b gray-scale), we know the symmetry properties of the corresponding spectra. Hence, only half of the spectrum information needs to be computed; the remaining coefficients can be completed according to the complex-conjugated symmetry property.

Real-Valued FFT

Since this is a very common situation, real-valued FFT algorithms that reduce memory and computational requirements by about a factor of two for real-valued sequences exist. Basically, there are two approaches to realizing a real-valued FFT. The first is to use a complex FFT and to modify its in- and output. The other is to modify the algorithm such that it suits the properties of real-valued input data. However, only real-valued FFTs can be transformed with the latter approach.

Since we intend to compute the 2D-FFT by the separation approach, the input data to all row FFTs is real – however, their outputs are complex. Thus, the following column FFTs are supplied with complex input data. Therefore we have to implement a “general” complex FFT. As a result, we have to choose the first real-valued FFT implementation form.

The basic idea of this real-valued FFT technique is to simultaneously transform two real-valued sequences with one complex FFT. To do this, the two real-valued sequences \( x(n) \) and \( y(n) \) are combined to one complex sequence \( z(n) \) (cf. Fig. 2.7). This combination can be easily implemented on the system-level.

FFT transformation of this “merged” sequence \( z(n) \) yields a temporary spectrum \( Z(k) \). To obtain the required spectral coefficients \( X(k) \) and \( Y(k) \), \( Z(k) \) must be unscrambled [Sorensen 87]:

\[
X[k] = \frac{1}{2}(Z_r[k] + Z_r[N - k]) + \frac{i}{2}(Z_i[k] - Z_i[N - k]),
\]

\[
Y[k] = \frac{1}{2}(Z_i[N - k] + Z_r[k]) + \frac{i}{2}(Z_r[N - k] - Z_r[k]),
\]  

(2.6)

where \( k = 1, \ldots, \frac{N}{2} \), \( X[0] = \{Z_r[0], 0\} \) and \( Y[0] = \{Z_i[0], 0\} \).

Execution of Eq. (2.6) requires \( 2N - 4 \) real additions. The subscripts \( r \) and \( i \) in Eq. (2.6) denote real and imaginary parts, respectively. The corresponding data-flow is depicted in Fig. 2.7.
Figure 2.7: Simultaneous transformation of two real-valued rows by one complex FFT: The two real-valued sequences $x(n)$ and $y(n)$ are merged into one complex sequence $z(n)$. After FFT transformation, the two (half) spectras $X(k)$ and $Y(k)$ are obtained by unscrambling $Z(k)$ (cf. Eq. 2.6).

**Inverse Real-Valued FFT**

In the special case of 2D correlation we know that the UCCF-result matrix is real. Thus, when performing the final 2D-iFFT, the row iFFTs will compute real-valued output data. Based on this knowledge, we can apply an inverse real-valued FFT on these rows. To do this, we reverse the “unscramble”-process defined by Eq. (2.6) as follows: first, the first halves of the two symmetric sequences $X(k)$ and $Y(k)$ must be “scrambled”, yielding the complex sequence $Z(k)$ (cf. Fig. 2.7). Performing an iFFT on $Z(k)$ will then compute the complex sequence $z(n) = x(n) + jy(n)$ whose real and imaginary parts correspond to real-valued iFFTs of $X(k)$ and $Y(k)$. We construct the real-valued iFFT by solving Eq. (2.6) to $Z$:

$$
Z_r[k] = X_r[k] - Y_i[k], \quad Z_r[N-k] = X_r[k] + Y_i[k], \\
Z_i[k] = X_i[k] + Y_r[k], \quad Z_i[N-k] = Y_r[k] - X_i[k],
$$

where $k = 1, \ldots, \frac{N}{2}$, and $Z[0] = \{X_r[0], -Y_r[0]\}$.

**2.4 Optimized BST Algorithm**

As discussed in section 2.1.3, computation of the energy terms can be described by a correlational process. Thus, they can also be computed via the FFT. Not only are computational requirements thereby significantly reduced, but data dependent processing is also eliminated since only the spectra of the binarized template and the squared search area have to
2.4. Optimized BST Algorithm

Figure 2.8: Flow-chart of Fourier domain BST computation. The low-level processing part consists of two "correlation" paths (cf. Fig. 2.5) for computing the UCCF and energy terms, respectively.

be multiplied. Implementation of the BST-algorithm in hardware is drastically simplified because of this. Furthermore, since frequency domain processing of the energy terms is similar to UCCF computation, the nominator and denominator of Eq. (2.3) can be processed on identical hardware [Wosnitza 96]. The resulting algorithm flow for Fourier domain BST computation is shown in Fig. 2.8.

Since both "correlation paths" (UCCF and energy terms computation) are totally independent, we can achieve a further increase in speed by computing them in parallel. Computational requirements for non-optimized Fourier domain BST computation are listed in Table 2.3.

2.4.1 Pre-Computation of Template Spectra

Regarding object detection on a continuous image data stream, both the template and the binarized template can be transformed in advance, thus yielding additional savings (cf. the 'OFFLINE' comment in Fig. 2.8). These pre-computed spectra can also be stored in conjugate complex form. The required iFFT can thus be realized with the FFT ([Rabiner 75]) and hence no implementation of the iFFT is required.
Chapter 2. The New BST Algorithm for Object Detection

<table>
<thead>
<tr>
<th>Task/Sub-Task</th>
<th>ADD</th>
<th>MULT</th>
<th>DIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Correlation&lt;sup&gt;a&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D-FFT Search Area</td>
<td>—</td>
<td>2N 1D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td>—</td>
</tr>
<tr>
<td>2D-FFT Template</td>
<td>—</td>
<td>2N 1D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td>—</td>
</tr>
<tr>
<td>Elementwise Mult.</td>
<td>3 \cdot N^2</td>
<td>3 \cdot N^2</td>
<td>0</td>
</tr>
<tr>
<td>2D-iFFT</td>
<td>—</td>
<td>2N 1D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td>—</td>
</tr>
<tr>
<td>Normalization</td>
<td>0</td>
<td>(N - n + 1)^2</td>
<td>(N - n + 1)^2</td>
</tr>
<tr>
<td>Total</td>
<td>\sum 2 \times 2D Correlation + Normalization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup> e.g., computation of UCCF/energy terms.

<sup>b</sup> For computational requirements of an N-point 1D-FFT, please refer to Table 2.2 (optimized case).

Table 2.3: Number of real operations required for non-optimized Fourier domain BST computation (N × N search area; n × n template).

### 2.4.2 Real-Valued FFT-Based UCCF Computation

Looking at the data-flow of Fourier domain BST computation (Fig. 2.8), we see that in- and output data of each “correlation” path (e.g., UCCF and energy terms) is completely real-valued, allowing us to employ the real-valued FFT techniques discussed in section 2.3.2.

By computing the 2D-FFT by first transforming all rows and then all columns (separation approach, section 2.3) and doing the opposite to compute the 2D-iFFT, we can divide Fourier domain correlation into three major phases:

1. **Real-valued row FFTs**: All pairs of neighboring rows are simultaneously transformed by means of real-valued FFTs resulting in a \((N/2 + 1) \times N\) temporary matrix.

2. **Next**, all columns of this matrix are transformed using conventional complex FFTs. After transformation, all column elements are multiplied with corresponding template coefficients before inverse column transformation is performed.

3. **Finally**, for the row-wise iFFT, two rows are “scrambled” (Eq. 2.7) each time before being processed by the inverse real-valued FFT.

The resulting data-flow for the real-valued FFT-based UCCF computation is shown in Fig. 2.9. Computational rates required for optimized Fourier domain BST computation are listed in Table 2.4.
2.4. Optimized BST Algorithm

(a) Phase 1: real-valued row FFTs. Each pair of two neighboring rows of the real-valued search area are merged into one complex sequence which is subsequently processed by the processor element (1D-FFT, UNSCRAMBLE (Eq. 2.6)). Real-valued row transformation of the $N \times N$ search area results in an $(\frac{N}{2} + 1) \times N$ temporary matrix.

(b) Phase 2: complex column FFTs – elementwise multiplication of transformed column and corresponding template column – inverse FFT on complex product column.

(c) Phase 3: real-valued row-iFFTs. Each pair of two neighboring rows of the 'phase 2' temporary matrix are merged into a length $N + 2$ complex sequence. Processing this row by a sequence of SCRAMBLING (Eq. 2.7) and 1D-iFFT yields the two corresponding real-valued space domain representations. After completion of phase three, the required $N \times N$ UCCF matrix is computed.

Figure 2.9: Optimized Fourier domain UCCF computation employing real-valued FFT techniques.
<table>
<thead>
<tr>
<th>Task/Sub-Task</th>
<th>ADD</th>
<th>MULT</th>
<th>DIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Correlation&lt;sup&gt;a&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D-FFT Search Area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-Valued Row FFTs</td>
<td>$\frac{N}{2} \times 1$D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unscramble</td>
<td>$\frac{N}{2} \cdot (2N - 4)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Complex Column FFTs</td>
<td>$(\frac{N}{2} + 1) \times 1$D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elementw. Mult.</td>
<td>$3 \cdot (N \times (\frac{N}{2} + 1))$</td>
<td>$3 \cdot (N \times (\frac{N}{2} + 1))$</td>
<td>0</td>
</tr>
<tr>
<td>2D-iFFT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex Column iFFTs</td>
<td>$(\frac{N}{2} + 1) \times 1$D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scramble</td>
<td>$\frac{N}{2} \cdot (2N - 4)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Real-Valued Row iFFTs</td>
<td>$\frac{N}{2} \times 1$D-FFTs&lt;sup&gt;b&lt;/sup&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normalization</td>
<td>0</td>
<td>$(N - n + 1)^2$</td>
<td>$(N - n + 1)^2$</td>
</tr>
<tr>
<td>Total</td>
<td>$\sum 2 \times 2D$ Correlation + Normalization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup> e.g., computation of UCCF/energy terms.

<sup>b</sup> For computational requirements of an $N$-point 1D-FFT, please refer to Table 2.2 (optimized case).

Table 2.4: Number of real operations required for optimized Fourier domain BST computation ($N \times N$ search area; $n \times n$ template).
Compared to conventional FFT techniques, operation count and memory requirements are reduced by approximately 50% through the application of real-valued FFT and iFFT. In addition, considering the column transformations, we can achieve additional savings in computational requirements. Since the left-most (DC-values) and the right-most columns are also completely real-valued, again, a second application of a real-valued FFT becomes feasible. In this case, two multiplication procedures would however be required for the elementwise multiplication with template coefficients.

2.5 Results

The resulting computational requirements for optimized Fourier domain BST computation are listed in Table 2.5. Compared to space domain computation, a reduction of the computational requirements up to a factor of 270 can be achieved, depending on the resolution of the search area.

Figure 2.10 demonstrates the improved detection performance of the BST algorithm. In this example, a padlock serves as complex template with free-form shape plus additional holes in its center. While locating this object on an irregular background is absolutely impossible when using the traditional NCCF, as shown in Fig. 2.10(b), the BST algorithm delivers the correct object positions with absolute precision and selectiveness as indicated by the sharp peaks (white dots) in Fig. 2.10(c).

Experiments with various template shapes and background patterns have clearly shown that the robustness of the proposed BST algorithm

<table>
<thead>
<tr>
<th>Search Area Size</th>
<th>Space Domain Ops$^1$</th>
<th>Frequency Domain Ops$^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rad.-4 FFT</td>
<td>Optimized</td>
</tr>
<tr>
<td>256x256</td>
<td>1.09·10⁹</td>
<td>27.6·10⁶</td>
</tr>
<tr>
<td>512x512</td>
<td>9.7·10⁹</td>
<td>126·10⁶</td>
</tr>
<tr>
<td>1024x1024</td>
<td>52.7·10⁹</td>
<td>549·10⁶</td>
</tr>
</tbody>
</table>

$^a$ Each real-valued arithmetic operation (e.g., addition, subtraction, multiplication and division) is considered as one operation.

Table 2.5: Computational requirements for a single frame (template size 128x128, "Blue-Screen Technique", cf. Tables 2.1, 2.4).
against geometric distortion and noise is of the same order as for the NCCF unless the number of relevant pixels in the rectangular template area enters a range where the majority of template pixels do not carry any substantial template information. Since the template's bounding box is "automatically" adapted to its minimal size by the BST algorithm, in this case a template with a more rectangular shape must be chosen in order to reduce the number of irrelevant template pixels.

To sum up, the BST algorithm offers the same advantages as the NCCF (reliability, stability, etc.) but extends the capability to detect arbitrarily-shaped free-form templates of variable size on an irregular background.
3. Implementation

In the preceding chapters, we have been concerned primarily with the algorithmic realization of free-form object detection. The current and the following chapters will focus on the special-purpose hardware concept implementing the BST algorithm.

To conclude, we propose a scalable multiprocessor system performing all computationally-intensive tasks of the BST algorithm in combination with high performance DSPs for high-level processing and controlling. To achieve maximal performance, the multiprocessor system is based on dedicated processor elements (PEs). Design and VLSI implementation of this PE is the central element of this work and is discussed in the following chapters.

3.1 Requirements

One of the largest requirements concerning hardware implementation of the BST algorithm is given by the real-time constraint of 10 complete correlations per second (cf. system specifications, section 1.1.3). At the present level of available technology, a typical state-of-the-art DSP (e.g., ADSP-21062) requires more than two seconds to process a 1024 x 1024 search area by the BST algorithm. As a result, the large computational requirements imposed by the real-time constraint (cf. Table 2.5) can only be met by means of massive parallel and thus comparatively expensive multiprocessor systems. On the other hand, due to the expected demand for real-time object detection systems, compactness and a reasonable price are essential preconditions to achieving good sales potentials. Hence, the number of required components must be reduced as much as possible.

Also, in order to efficiently support multiple image resolutions, system performance must be scalable to accommodate the variance in computational requirements. This is also required to make the overall system extendable to further, larger computational demands which will be
imposed by foreseeable quickly growing sensor technology.

In addition, to open a large application field, the system must be as versatile as possible. Referring to the BST technique, this means providing robustness against varying illumination conditions and irregular backgrounds, and hence is basically a matter of accuracy provided throughout computation. On the other hand, thinking of a more generally applicable system, other algorithms like the native 2D-FFT or additional pre-/post-processing steps (even in the course of the BST algorithm) must be executable on the resulting system. Thus, a trade-off between flexibility and compactness must be found.

To sum up all requirements, we have to design a compact but versatile multiprocessor system based upon an extendable and scalable architecture.

## 3.2 System Concept

Although extremely desirable, design of a very versatile system supporting multiple algorithms for object detection is difficult. Since very different kinds of algorithms had to be supported, this approach definitely does not allow for the most compact implementation. As a consequence, the major objective for system design concentrates on the efficient implementation of Fourier domain BST computation and its related tasks.

Concerning the cost-to-performance ratio, special-purpose hardware is always more efficient compared to general-purpose computers. Because of this, the constraints concerning real-time processing, compactness and scalability almost naturally lead to a problem-oriented multiprocessor system.

### 3.2.1 Classification of BST-Related Tasks

In a first step to achieving an optimal hardware mapping of the BST algorithm, we classify all tasks involved in Fourier domain BST computation according to their number and complexity. For convenience, following traditional classification in signal processing, we divide them into high-level and low-level tasks.

**Low-Level Tasks**

*Low-level tasks* are characterized as regular arithmetic operations imposing the majority of computational requirements due to their large
In the case of the BST algorithm, we attach the 1D-FFT, the elementwise complex multiplication and the real-valued FFT’s pre-/post-processing steps to the class of low-level tasks*.

**High-Level Tasks**

In addition to low-level operations, there are also *high-level tasks*, which are of increased complexity, particularly to the extend that they comprise data-dependent operations and global controlling tasks. The final normalization of UCCF results by corresponding energy terms (cf. Fig. 2.8) and the isolation of local maxima within the resulting matrix of similarity measures are typical representatives.

### 3.2.2 Parallelisms in Low-Level BST Computation

According to the dominating computational requirements imposed by low-level tasks, we have to place a special effort on their implementation. To achieve compactness and cost-efficiency our basic strategy is to map all computationally extensive low-level tasks onto a multiprocessor system. To do so, we have to exploit data-parallelism within Fourier domain low-level BST computation.

It is obvious that both multiprocessor architecture and the PE concept strongly depend on the level on which the BST algorithm is parallelized. Looking at the optimized BST algorithm (Fig. 2.8), we see that it consists of two symmetrical UCCF computations in which several levels of parallelism exist:

**Arithmetic Operation**

At the arithmetic operation level (i.e., single additions, multiplications, etc.) we have multi-tasking-like conditions, which allow us to fall back onto established multiprocessing techniques. Thus, on this level, we can achieve parallel BST computation by means of *super-scalar, fine-grained multi-threaded* or *simultaneous multi-threaded* processing.

**Butterfly**

This level is appropriate only for FFT computation and is justified by the large number of existing butterfly processors.

---

*Hence all operations required to compute a Fourier domain correlation as described by Fig. 2.9 are considered low-level tasks.*
Row
Computing the 2D-FFT by the separation approach and transposing the data set between the column- and row- 1D-FFTs naturally leads to image rows being the fundamental data element [Cavadini 97b]. As shown in Fig. 2.8, all remaining low-level tasks can also be parallelized on the row-level.

Row-Fragments
In the course of exploiting row-parallelism, the required blocklength can further be partitioned into multiple segments (cf. sectioned convolution, section 2.3.1). The major advantage of this level of parallelism is that only one transform length must be supported by the PEs. Larger blocklengths are composable. However, as already discussed, system performance decreases according to repeated computation of overlapping sequence elements.

While the arithmetic structure of corresponding PEs is simplified for lower levels of parallelism, communication requirements between the PEs and the image memories strongly increase. Because of this, controlling and, in particular, efficient scheduling of available resources become significantly difficult. To simplify the aspects of controlling and resource scheduling and also to minimize bandwidth between system memory and the PEs, we choose parallel BST computation on the row-level, as already indicated by Fig. 2.9. As we will see in the following, the resulting PE thus becomes reusable for a variety of related 1D applications.

3.2.3 Multiprocessor Architectures
The decision to exploit parallelism on the row-level is also motivated by considerations concerning partitioning and the distribution of search area and template within the multiprocessor system and, in this course, also by taking into account the number of repeated computations [Cavadini 99].

By making assumptions about data distributions we also found the basis for corresponding multiprocessor architectures. The variety of parallel computer architectures can be categorized into different groups [Flynn 66, Flynn 72, Rosenthal 97]. Referring to this classification, the three most important approaches suitable for low-level BST computation are single instruction – multiple data (SIMD), multiple instruction
3.2. System Concept

- multiple data (MIMD), and multiple instruction – single data (MISD) architectures.

MISD systems are pipeline-architectures; systolic arrays are a typical representative. Usually, all data is clocked through a one- or more-dimensional arrangement of PEs. The functional scope of each PE in the formation may vary according to the tasks they are performing.

However, data-flow is more or less fixed according to the underlying arrangement of PEs. In addition, the system's data throughput is limited by that of the "slowest" PE. Hence, overall throughput can only be increased by replacing the "weakest" PE by a more powerful one which shows the same functionality. In general, since the number and type of implemented PEs are given by the system's functionality, implementation of additional PEs will increase the system's computational performance but will also modify the system's functionality. As a result, an MISD real-time object detection system is not easy to scale.

Better scalability can be achieved by means of SIMD and MIMD architectures. SIMD systems perform identical instructions on all PEs simultaneously. In contrast, MIMD architectures concurrently compute different tasks on the set of available PEs. MIMD computers can be further differentiated into SPMD (single program – multiple data) and MPMD (multiple program – multiple data) architectures. For implementation and taking the specified real-time constraints into account, the SPMD architecture qualified as the most suitable option because of its scalability in regard to computational performance and its flexibility in supporting the required set of low-level BST tasks [Cavadini 97b].

3.2.4 Concepts for System Memory

The requirements on system memory are very manifold. First of all, it must be large enough to hold the complete search area, the transformed template as well as several temporary results.

A second major design aspect concentrates on optimal data transfer between all computation units to achieve optimal performance. Continuously providing new data to all PEs is rendered even more difficult according to the increasing discrepancy between processor cycle times and access times to available memory. In this context, an extensive comparison and assessment of physical memory types is given in [Cavadini 97b, Cavadini 99], which also includes the requirements

\[†\] Referring to the system architecture under discussion, a PE is considered to be a distinct electronic component, e.g., ASIC.
imposed by the variety of access patterns as well as estimations of corresponding external data bandwidth.

Furthermore, by exploiting BST parallelism on the row-level, several problems arise concerning the distribution of data onto the PEs: system memory must first be addressed by rows and then by columns. Thus, distributed memory and shared memory concepts are architectural options for system memory [Cavadini 97a].

The distributed memory architecture opens complete decoupling of data access for each PE during row processing, thus making it ideal for all row-oriented operations of the BST algorithm. However, performance breaks down for all column-related tasks. This is because the required data is distributed over all memory partitions assigned to different PEs. Thus, column-related tasks will profit from a shared memory implementation since it allows for global access to system memory: rows and columns can be addressed without any particular communication procedures. However, this approach leads to massive conflicts between PEs that concurrently request data from the same memory module.

Combining the advantages of both concepts, a mixed shared distributed memory architecture is most suited to implementing system memory [Cavadini 97a, Cavadini 99].

### 3.3 Resulting System Architecture

The resulting system architecture, which implements the mixed shared distributed memory architecture, is shown in Fig. 3.1. Since questions regarding the design of this system architecture are comprehensively covered in [Cavadini 99], we will confine ourselves to a brief system description in the following.

According to our classification of BST-related tasks (section 3.2.1), the system consists of two main parts: the multiprocessor system (Frequency Domain Engine) (FDE), that computes the low-level tasks (e.g., UCCF and energy terms) and the high-level computational part. While all PEs of the FDE are realized in VLSI for performance reasons, high-level tasks usually complicate VLSI implementation due to their complexity (e.g., arithmetic division) and data dependency. They are also represented by a comparatively low operation count and hence preferably mapped on available general purpose computing devices. Keeping this and a short time to market in mind, an appropriate digital signal processor (DSP) is a good choice for high-level processing. At the present level of existing DSPs, the ADSP 2106X (SHARC)
3.3. Resulting System Architecture

Seems to be most suited to taking on the required tasks since it disposes of a high floating-point performance as well as good multiprocessing and communication capabilities. Hence, the high-level computational part consists of two SHARCs, which also take on controlling and external system interfacing (Video I/O).

The mixed shared-distributed memory architecture for the FFT processor is basically implemented by two bus systems and several crossbars. The *Horizontal Bus System* (HBS) allows each PE to efficiently access all rows within its exclusively-assigned memory partition. In addition, the *Vertical Bus System* (VBS) enables global data access for all PEs. While the HBS is traditionally implemented by address and control signals, VBS architecture is specifically optimized to implement transparent matrix transposition during data transfer. More details on this feature are given in [Cavadini 97b, Cavadini 99]. Due to its shared memory characteristic, the VBS is also employed for data exchange between the low-level FFT processor part and the high-level processing part as well as for I/O purposes.

With this concept, system performance almost linearly scales with the number of implemented PEs and supported communication bandwidth. Referring to the latter aspect, we want to point out that VBS-
bandwidth can be increased by providing more than one data bus. Detailed performance data of the proposed multiprocessor system are listed in Table 3.1. The figures given here are based on "best case" assumptions, e.g., complete utilization of all embedded PEs and fastest access to system memory. The size of system memory required for the processing of \(256 \times 256\), \(512 \times 512\) and \(1024 \times 1024\) search areas amounts to 1.92 MByte, 7.7 MByte and 30.8 MByte, respectively [Cavadini 99].

<table>
<thead>
<tr>
<th>Search Area</th>
<th>(P^a)</th>
<th>(SH^b)</th>
<th>(VBS_{PAR}^c)</th>
<th>Frame Rate [Frames/sec.]</th>
<th>Period [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 \times 1024</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4.35</td>
<td>229.89</td>
</tr>
<tr>
<td>1024 \times 1024</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>7.59</td>
<td>131.75</td>
</tr>
<tr>
<td>1024 \times 1024</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>7.59</td>
<td>131.75</td>
</tr>
<tr>
<td>512 \times 512</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>16.96</td>
<td>58.96</td>
</tr>
<tr>
<td>512 \times 512</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>29.09</td>
<td>34.38</td>
</tr>
<tr>
<td>512 \times 512</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>30.35</td>
<td>32.95</td>
</tr>
<tr>
<td>256 \times 256</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>61.80</td>
<td>16.18</td>
</tr>
<tr>
<td>256 \times 256</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>99.67</td>
<td>10.03</td>
</tr>
<tr>
<td>256 \times 256</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>121.39</td>
<td>8.24</td>
</tr>
<tr>
<td>1024 \times 1024</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4.35</td>
<td>229.89</td>
</tr>
<tr>
<td>1024 \times 1024</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>7.59</td>
<td>131.75</td>
</tr>
<tr>
<td>1024 \times 1024</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>12.11</td>
<td>82.58</td>
</tr>
<tr>
<td>512 \times 512</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>16.96</td>
<td>58.96</td>
</tr>
<tr>
<td>512 \times 512</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>29.09</td>
<td>34.38</td>
</tr>
<tr>
<td>512 \times 512</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>45.29</td>
<td>22.08</td>
</tr>
<tr>
<td>256 \times 256</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>61.80</td>
<td>16.18</td>
</tr>
<tr>
<td>256 \times 256</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>99.67</td>
<td>10.03</td>
</tr>
<tr>
<td>256 \times 256</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>143.69</td>
<td>6.96</td>
</tr>
</tbody>
</table>

\(^a\)Number of embedded PEs  
\(^b\)Number of implemented DSPs (e.g., SHARC)  
\(^c\)Number of implemented VBS bus systems

Table 3.1: BST performance of scalable multiprocessor system according to [Cavadini 99]. Best case (e.g., system memory realized by SDRAM, optimal utilization of embedded processor elements).
4. Processor Design

Given the overall structure of the real-time system, we can now concentrate on the product profile of the embedded processor element. Hence, in this section, we walk through the decision-making process used to specify and design the actual FFT processor.

In conclusion, the specific requirements imposed by the set of major system constraints lead us to design a dedicated processor element in order to achieve maximal system performance.

Figure 4.1: FFT processor housed in a PGA-180 ceramic package. Die size: $11.6 \times 14.4 \text{ mm}^2$. 
4.1 Requirements on the Ideal Processor Element

According to the characteristics of the multiprocessor system outlined in section 3.3, the best overall performance can only be achieved by populating it with 'ideal' processor elements, the characteristics of which are optimally designed to support the peculiarities of the underlying multiprocessor system. In the following, we will discuss the requirements which are made on this ideal PE.

First of all, the ideal PE must provide high computational performance in order to meet the system's real-time specification of 10 frames per second and to minimize the number of required system components. In particular, quick FFT computation must be guaranteed. As discussed in [Cavadini 99], managing the real-time constraint by a 4-PE multiprocessor system means that the ideal PE has to compute a complex 1024-point 1D-FFT within $80\mu s$. Hence, the PE's arithmetic unit must provide native support for complex operations and parallel resources to allow for high data throughput.

Enhanced I/O-characteristics are also required. While this demand requires the availability of high-performance communication interfaces first and foremost, the bandwidth capacity of the external VBS/HBS bus-systems must not be exceeded. In terms of economy, access requirements on system memory must also be reduced as much as possible. The ideal PE must then have a local memory that is large enough to hold at least one complex 1024-point sequence (parallelization on row-level, cf. section 3.2.2).

In addition, local memory is the basic precondition for simultaneous data processing and I/O. To support this feature, double-sized memory (e.g., two complex rows) is required. In addition, to achieve optimal capacity utilization of the given multiprocessor architecture, the ideal PE must show a balanced time-ratio for computation and I/O phases. That is, within the time span required for actual data processing (e.g., FFT), previously-computed results must be exchanged with new data. For other ratios, system performance is either limited by the PE's computational power or by I/O transfer rate, respectively.

Although the ideal PE must provide excellent FFT performance, it must also show sufficient flexibility to support all low-level tasks required for Fourier domain BST computation, e.g., elementwise multiplication and real-valued FFT pre-/post-processing tasks.

Furthermore, the ideal PE must allow for high-precision FFT com-
putation in order to guarantee a maximal inaccuracy of 0.4% for the final NCCF matrix (cf. section 6.1). Only then can precise and reliable object detection over a large variety of image materials and lighting conditions be achieved.

### 4.2 Rating of Existing Processor Classes

In this section, we rate existing processor classes by the extend to which they match the characteristics of the ideal processor element.

#### 4.2.1 Dedicated FFT Processors

According to their problem-specific architecture, the best FFT performance is obviously achieved by dedicated FFT processors. However, as can be seen from Table 4.1, commercially-available processors dispose of fixed-point arithmetic showing wordlengths between 16 and 24\(b\). While these wordlengths might be sufficient for 1D applications employing short or medium blocklengths, they do not match the numeric range associated with 2D FFT computation. As a result, a 2D \(1024 \times 1024\) spectrum computed by 24\(b\) fixed-point arithmetic is significantly distorted by underlying quantization errors (cf. section 6.3.1). However, as discussed in Chapter 6, BST computation on high-resolution images requires high precision.

<table>
<thead>
<tr>
<th>Wordlength</th>
<th>FFT Processor</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16(b)</td>
<td>PDSP16510A (Plessey)</td>
<td>98(\mu)s</td>
</tr>
<tr>
<td></td>
<td>SNC960A (Sicom)</td>
<td>20(\mu)s</td>
</tr>
<tr>
<td>18(b)</td>
<td>PDSP16515A (Plessey)</td>
<td>98(\mu)s</td>
</tr>
<tr>
<td>20(b)</td>
<td>L64280 (LSI)</td>
<td>26(\mu)s</td>
</tr>
<tr>
<td>24(b)</td>
<td>BDSP91V24 (Butterfly DSP)</td>
<td>54(\mu)s</td>
</tr>
<tr>
<td></td>
<td>DSP-24 (DSP Architectures)</td>
<td>21(\mu)s</td>
</tr>
</tbody>
</table>

Table 4.1: Fixed-point wordlength and execution time for 1024-point complex FFT of today’s commercially-available FFT processors.
Furthermore, dedicated FFT designs are characterized by poor flexibility. In particular, the set of low-level tasks required for Fourier domain BST computation is not completely supported (e.g., elementwise complex multiplication and real-valued FFT). The resulting real-time system is thus significantly enlarged since additional computational elements must be implemented to perform remaining non-FFT tasks.

In reference to processor interfacing, corresponding I/O-characteristics vary with different FFT processor types. However, in most cases, the underlying interfacing is non-optimal with regard to the dedicated multiprocessor system discussed in section 3.3. In particular, simultaneous computation and I/O is prevented due to under-dimensioned or missing on-chip memory, resulting in non-optimal task scheduling on system-level.

### 4.2.2 Digital Signal Processors (DSPs)

One major advantage of DSPs is given by their flexibility. Being programmable, all BST-related low-level tasks can be implemented on this processor type. Furthermore, some DSPs are equipped with floating-point arithmetic units allowing FFT results to be computed with excellent accuracy. In addition, referring to I/O-characteristics, DSPs' external memory interfaces operate concurrently with accesses to on-chip memory and always transfer one word per memory transaction.

On the other hand, the maximum clock speed of the currently available DSPs is very moderate, i.e., clearly lower than 100MHz. This is particularly the case for high-precision DSPs providing native floating-point arithmetic*. Furthermore, up to now, no DSP provides native support for complex arithmetic. As a result, multiple clock cycles are required to accomplish complex multiplications and additions, hence significantly reducing overall FFT performance.

At present, the ADSP-2106X (SHARC) is the most powerful floating-point DSP that is commercially available. However, as shown in [Cavadini 99], approximately 30 SHARC:s are required to allow for real-time computation of the BST algorithm on 1024 × 1024 images (10 frames/second). The targeted real-time system then becomes too complex and voluminous.

*This limit is to be broken by the 167MHz TMS-320C6701 floating-point DSP which has been announced for Q02/99.
4.2.3 General Purpose CPUs

The most flexible processor designs are obviously represented by the family of general-purpose (GP) CPUs. However, GP processors are highly sequential machines and hence much less efficient in FFT computation than dedicated FFT architectures or DSPs. Targeting a maximum of flexibility, GP architectures are not optimized for FFT computation. In particular, as for the DSP, they do not natively support complex data formats and do not have a coefficient ROM.

In addition, GP designs are characterized by a very limited amount of parallel resources. Also, efficient utilization of potentially provided parallelism is usually difficult due to highly pipelined data paths.

Concerning I/O-characteristics, some high-performance GP processors (e.g., ALPHA) transfer multiple words per transaction. However, general-purpose processors' external memory interfaces generally do not operate in parallel with on-chip memory [BDTI 97]. Concurrent computation and I/O cannot be realized because of this. Also, only a few GP CPUs provide multiprocessor capability and if so, resulting scalability is usually very limited (e.g., support for two or four processors).

In light of these facts it is clear that the present level of GP processor technology cannot be employed economically for Fourier domain BST computation. However, GP CPUs provide the best precision and flexibility. Also, as clock speed exceeds 400MHz, many real-time operations can now be performed on GP CPUs without hardware acceleration. The term 'native signal processing' (NSP) is sometimes used to describe this situation [Lapsley 97]. And since the clock speed of GP CPUs keeps increasing with each forthcoming generation, they could be an interesting alternative for real-time BST computation in a couple of years.

4.3 Processor Concept

Looking at the processor classes discussed so far, it is clear that no satisfying solution exists. Consequently, we designed a dedicated processor element in order to achieve maximum system efficiency and performance.

To recap, major requirements made on the targeted object detection system are given by the demands for "real-time", compactness, cost-efficiency and scalability. To derive the basic requirements on the processor architecture from these system constraints, we employ a deductive strategy as illustrated by Fig. 4.2.
Figure 4.2: Interrelation of primary system constraints and requirements on PE architecture.
As a result, the processor element is mainly characterized by a fast arithmetic unit natively supporting complex operations. This arithmetic unit must allow for high data throughput in FFT computation and must also support all low-level tasks required for Fourier domain BST computation.

In addition, the PE has to provide I/O- and control-characteristics allowing for fast external data exchange and convenient build-up of multiprocessor systems. A problem-specific I/O interface must therefore be implemented.

Also, in order to reduce bandwidth requirements on system buses as well as to enable concurrent computation and I/O, a local memory must be implemented. In order to achieve economic provision of twiddle-factors, FFT coefficients have to be generated on-chip.

## 4.4 Resulting Processor Structure

The resulting PE structure corresponding to the requirements derived in the previous section is depicted in Fig. 4.3. The main components are represented by the configurable complex arithmetic unit (APU)\(^1\), the local memory, the coefficient-ROM and the I/O interface.

Global PE controlling is basically partitioned into APU and I/O controlling. For each of these control sub-systems, hierarchical controlling is implemented in order to reduce wiring overhead and potential race

\[^1\text{A}rithmetic \text{P}rocessing \text{U}nit\]

![Figure 4.3: Processor block-diagram.](image-url)
problems. To achieve maximum performance, three address-generation
units (AGU) allow for independent and simultaneous generation of ad-
dress sequences for the APU read port, the APU write port and the
coefficient ROM.

Looking at the basic elements of our processor concept, special ef-
fort was placed on the design of the arithmetic unit to achieve optimal
system performance. In particular, to meet the precision constraints
imposed by high-resolution object detection via the BST algorithm (cf.
section 6.1), we spent much time on an appropriate arithmetic concept
allowing for both computation of precise object positions and efficient
VLSI implementation. According to the diversity and significance of
corresponding design aspects, a detailed discussion on the design of the
arithmetic unit is given in the coming chapters.

In the following, we will give a brief overview of major problems
and features associated with the remaining major building blocks of the
processor concept described above.

4.4.1 Local Memory (LM)

As discussed, implementation of on-chip memory is primarily motivated
by reduction of system bus-bandwidth. However, by locally storing
intermediate results, we also decrease access-requirements on system-
memory. This is of great benefit since when targeting high-speed com-
putation, it might not be too expensive to make the arithmetic unit
fast, but it could turn out to be very expensive to use comparably fast
system memory.

Anticipating the actual structure of the arithmetic unit (APU) de-
veloped in Chapter 5, the APU has two read ports (coefficient and LM
data) and one write port (LM data) (cf. Fig. 4.3). The LM must
therefore allow for simultaneous read- and write-accesses and hence is
preferably realized by means of dual-ported memory elements.

When targeting maximal data throughput, simultaneous computa-
tion and I/O and hence implementation of two dual-ported on-chip
memories is a must. However, as illustrated in Fig. 4.3, three single-
ported memories are implemented.

This sub-optimal implementation comes from the peculiarities of the
CMOS technology used: dual-ported memory elements were not avail-
able (see section 8.2.2). Usually, one would assume that implementation
of four single-ported memories is consequently required to compensate
for the missing multi-port feature. However, since the APU’s computa-
tion time is always at least twice as long as the time span associated
with bidirectional I/O, we can serialize in- and output operations without any decrease in performance. Only three memory elements must then actually be implemented to achieve the desired functionality.

As a result, the LM is implemented by three arbitrarily-configurable on-chip memories. All three memories are connected to the APU’s read and write ports and to the I/O port by means of the dedicated data and address routing network (DARN) (cf. Fig. 4.3). Each of the three memories can thus be configured to be read or written by the APU or to be externally accessible via the I/O interface. This approach guarantees concurrent computation and exchange of I/O data.

Considerations of Efficient LM Utilization

Concerning the size of each local memory, storage depth must be suited to allow for the processing of 1024-point data sequences. However, recalling system specifications, the processor must also handle sequences of length 256 and 512, respectively.

Referring to the most obvious solution, we could simply down-load those shorter sequences into the processor for further processing. This would mean, however, that 50 to 75% of the implemented LM resources would remain unused. Furthermore, according to the decreased processing time, overall task scheduling on the system-level must be accordingly modified to meet these varying timing constraints.

Our strategy for overcoming all these problems is to simultaneously process multiple sequences whenever the corresponding sequence length is smaller than 1024. Hence, we down-load two 512-point sequences or four 256-point sequences into the processor, as illustrated in Fig. 4.4 for the case of FFT computation.

We can thus achieve efficient LM utilization while maintaining constant system timing at the same time. Also, the initialization overhead imposed by controller peculiarities and pipeline latency is shared by all 'sub-sequences', resulting in shorter processing times for a single 'sub-sequence'.

4.4.2 Provision of Transform Coefficients

Anticipating the constant throughput data path unit (cf. section 5.3.5), new twiddle coefficients $W$ must be provided with each clock cycle. According to this, the fastest and most economical way is to generate the twiddle-coefficients locally within the PE. Then, no signal distribution
Figure 4.4: Efficient LM utilization by means of simultaneous processing of multiple sequences: For block lengths 256 and 512, four and two sequences, respectively, are merged into a length 1024 “meta-sequence” and simultaneously processed by the processor, as shown in subfigures b) and c).

from external system memory to the PEs is required, thus allowing for fast access and a corresponding reduction in pin count.

Several options exist for the on-chip generation of transform coefficients. Since the involved coefficients are constant, the obvious way is to implement a corresponding coefficient ROM. Alternatively, in cases where a ROM implementation is not available or would consume too much area, coefficients might also be generated by an appropriate combinational logic. Referring to the CMOS technology used, we found the ROM implementation variant to be more efficient in terms of compactness.

In regard to area requirements, implementation effort could be further reduced since only a sub-set of the required twiddle coefficients must be actually generated. This is because, referring to the complex plane, the coefficients are uniformly distributed over the unit-circle. Thus, according to this point symmetry, we would have to generate twiddle coefficients only for one of its quadrants. The ‘missing’ coef-
ficients could then be restored according to the underlying symmetry property.

While this approach would allow for smaller ROM(s), additional arithmetic components must be provided to perform the required restoration. To conclude, we found the additional area-overhead imposed by these restoring measures to be much larger than the achieved savings in the ROM area.

4.4.3 System Interfacing

A problem-specific I/O interface has been implemented to achieve fast data exchange with external system components. The I/O interface transfers one real data-word with each clock cycle. Corresponding real and imaginary parts are thus transferred in sequence and external bus bandwidth as well as pin count are reduced.

In addition, a look-up table has been implemented into the I/O path to allow for transparent pre-/post-processing of I/O data (cf. Fig. 4.3). The efficient squaring of 'energy terms' as well as thresholding (binarization) can thus be realized (cf. Eq. 2.1).

The I/O interface operates on a separate clock and is thus able to tightly interact with external system components, offering additional flexibility to integrate the processor into a given system environment. Attaching the PE as a coprocessor to existing general-purpose computing facilities is thereby significantly simplified.
5. High-Throughput Data Path
FFT Architecture

In this chapter we derive a new data path architecture for efficient high-throughput FFT computation. Configuring the data path architecture, we propose a new hardware mapping for radix-4-based FFT computation and specify the set of required arithmetic resources. To allow for efficient resource utilization, we develop a dedicated task-switching concept based on pipeline interleaving. Finally, to allow for execution of further BST-related low-level tasks, we add minimal modifications to the derived FFT architecture.

5.1 Architectural Requirements

The goal of achieving the best performance places several requirements on the design of the processor’s arithmetic unit (APU).

To recap, one major objective of APU design is to allow for high data throughput. According to the interface of the local memory, the arithmetic unit must continuously compute one result per clock cycle in order to sustain constant data throughput.

In order to realize a compact object detection system, the arithmetic unit must support all computationally-intensive tasks required for Fourier domain BST computation. Hence, the APU architecture must have sufficient flexibility at its disposal to support 'elementwise-multiplications' and real-valued FFT pre-/post-processing tasks in addition to actual FFT computation. Furthermore, referring to different image sizes, handling of multiple sequence lengths must be considered.

In addition, the resulting APU architecture must allow for monolithic VLSI implementation. Corresponding constraints on interconnect and silicon area impose clear restrictions on appropriate data-flow and hence architecture types.
Looking into the future, the APU architecture should also allow for simple mapping to advanced process technologies. Compared to today’s standard cell processes, considerable domination of interconnect delay over gate delay must be assumed for these technologies. Planning of appropriate signal distribution is thus an important part of APU design in order to achieve technological scalability.

Finally, targeting high performance at a large operations per area ratio, efficient utilization of implemented arithmetic resources is a must.

5.2 Confronting Previous FFT Designs

Fourier domain-based computation of the BST algorithm is clearly dominated by the FFT. APU architecture is therefore preferably optimized for FFT computation.

Up to now, several dedicated architectures have evolved and have achieved an acceleration in FFT computation by exploiting algorithm-specific features, above all parallelism. Looking at the variety of FFT designs, best data throughput is achieved by array architectures. Various designs of this architecture type exist. According to underlying data-flow, they can be classified into 1D and 2D architectures, respectively.

5.2.1 1D Array Architectures

A typical representative of 1D array architectures is given by the pipeline FFT. As the major characteristic of pipeline FFT architectures, parallel radix-r-based FFT processing is realized by means of \( \log_r N \) butterfly units. The resulting high data throughput makes this architecture type an appropriate option for high-speed systems. A review of pipeline FFT architectures is given in [He 98].

The principles of the pipeline FFT are discussed in [Rabiner 75]. As mentioned above, each \( \log_r N \) stage of a radix-r FFT is associated with a corresponding butterfly unit. Each butterfly unit consists of a commutator, delay lines and an arithmetic element (AE) computing single radix-r butterflies. The delay lines perform the required reorganization of the data stream. According to the data access scheme of a radix-4 FFT, their length is reduced by a factor of four with each succeeding stage. The block diagram of a 64-point radix-4 pipeline FFT is shown in Fig. 5.1.
5.2. Confronting Previous FFT Designs

In normal operation, the input sequence is alternately distributed on the four inputs of the first commutator. Hence, a new quartet of input data is available every four clock cycles. As a consequence, overall system efficiency is merely 25%, making the native radix-4 pipelined FFT very inefficient.

However, several approaches to achieving 100% efficiency exist. The simplest way to yield 100% resource utilization is to reduce computation clock of the AEs to one fourth of the sampling clock rate. On the other hand, this approach will result in the inefficient use of silicon area. Alternatively, the radix-4 pipeline FFT can be made 100% efficient by simultaneously transforming four distinct sequences \( x_1(n), x_2(n), x_3(n) \) and \( x_4(n) \) (Fig. 5.2(a)). As shown in Fig. 5.2(b), each successive sequence is delayed by \( N/4 \) samples. In a sense, processing four independent signals is the most natural mode for the radix-4 pipeline FFT because the switching patterns of all commutators become the same, with each successive commutator switching at four times the rate of its predecessor. Recalling sequenced convolutions (section 2.3.1),

Figure 5.1: 64-point radix-4 pipeline FFT [Rabiner 75].

Figure 5.2: Pipeline FFT computation of four independent input sequences.
a radix-4 system thus also accommodates a 4:1 overlapped FFT with 100% efficiency.

5.2.2 2D Array Architectures

Compared to their 1D counterparts, considerably higher data throughput can be achieved by 2D array architectures.

A 2D array architecture performing 64-point FFT computation is presented in [Hui 96]. It is implemented in 0.6\(\mu\)m CMOS on a 7.8\(\times\)8mm\(^2\) die and computes a 64-point complex FFT within 3.6\(\mu\)s (18 Mega-samples per second). In contrast to most other approaches, the proposed architecture is based on dedicated algorithm-to-architecture mapping in order to achieve efficient VLSI implementation. In particular, the regularity inherent to the underlying DFT matrix has been exploited, resulting in a corresponding radix-4 matrix factorization. The original DFT matrix is thereby partitioned into smaller sub-matrices which are then mapped onto corresponding radix-4 computation arrays. Additional major building blocks of the proposed FFT architecture are in- and output formatters as well as commutator units.

According to the resulting regularity in data-flow, VLSI implementation of this array architecture is characterized by little wiring overhead.

On the other hand, referring to the underlying algorithm-to-architecture mapping, this architecture is very dedicated and does not support any of the remaining BST-related low-level tasks. But even for FFT computation, a strong limitation of supported transform lengths exist since transform length must be a power of four. The proposed architecture is therefore not suitable for, e.g., a 512-point FFT.

5.2.3 Rating of Previous Array Architectures

The pipeline FFT concept discussed so far allows for continuous transformation of an incoming data stream. However, to make the system 100% efficient in the case of non-overlapping FFTs, i.e., to completely utilize the implemented AEs, we must either slow down the computation clock of the AEs or decide on the simultaneous transformation of four different sequences. While for the first option the performance of a single AE is incompletely utilized, the second option demands a quadrupling of the size of local memory to maintain concurrent computation and I/O. In addition, the delay-lines used for data reorganization imply a considerable overhead of additional memory. In the case of a 1024-point radix-4 pipeline FFT, a total of 2,556 delay elements is required.
Also, the computation of tasks other than FFT processing, such as elementwise computation or real-valued FFT pre-/post-processing, is not possible with this inflexible architecture type.

In reference to the 2D architecture discussed, extension to larger transform lengths requires increased silicon area since a complete ensemble of commutator unit and computation array must be provided for each corresponding radix-4 stage. While a corresponding 64-point transform architecture can be implemented on a single chip, area requirements will be twice as high when targeting a 1024-point FFT.

5.3 The New Data Path FFT Architecture

Facing these limitations, existing array-concepts have not met the requirements for efficient Fourier domain BST computation. As a result, design of a dedicated data path architecture is required to combine high throughput with efficient resource utilization, to minimize the effort in data shuffling and to achieve the desired flexibility in both variable transform lengths and functional scope by a single arithmetic unit.

The new FFT data path architecture is developed in this section. We will focus on FFT processing first in regard to the derivation of the new architecture. Later, we will extend the architecture to fit the demands of all BST-related low-level tasks.

5.3.1 APU Concept

As given in section 5.1, basic requirements on APU design are expressed by demands for high throughput, high performance, technological scalability and qualification for VLSI implementation.

We have therefore chosen for pipelined processing in order to ensure high data throughput. This approach also allows for the complete encapsulation of functional units by pipeline registers as well as the implementation of pipelined interconnect to cope with problems associated with large interconnect delays. Thus, implementation of a pipelined data path architecture allows for the required technological scalability.

To reduce access requirements on the local memory, we have to minimize the complexity of I/O operations and minimize data transfer between APU and local memory. Therefore, in reference to the intended pipeline processing, the arithmetic unit preferably operates on a continuous data stream.
To achieve this computational performance, we have to exploit the parallelism inherent to the set of low-level tasks as much as possible. Furthermore, addressing efficiency of the architecture, the economic use of required silicon area must be achieved. As a consequence, utilization of implemented arithmetic resources must be maximized. In view of the APU's overall efficiency, the flexibility required to support all computationally-extensive low-level tasks must be achieved by a minimum of arithmetic resources.

As a result, we have to design a dedicated data path, allowing for highly pipelined and highly parallel data processing. To decide on an actual ensemble of arithmetic resources, we have to focus on the most 'critical' task that must be supported by the arithmetic unit. Since the largest requirements on data management and computational power are imposed by the FFT, optimization is done for the radix-4 algorithm.

5.3.2 Hardware Mapping for Radix-4-based FFT Computation

To achieve high FFT performance, and hence high data throughput for the FFT mode, the parallelism inherent to radix-4-based FFT computation must be exploited.

Parallelisms in Radix-4-Based FFT

Any FFT process is usually a mix of several types of parallelism. The symmetry of the FFT makes it possible to invent a host of structures containing almost the whole spectrum of possible parallelisms. As for any other algorithm, time overlap of control, arithmetic operations and memory access exist.

The most important type of parallelism inherent to the radix-4-based FFT is data parallelism. Due to its regularity, as many as $\frac{N}{4} \log_4 N$ parallel butterfly units (BU) can be employed, allowing for the extreme case of building the entire FFT as a single butterfly array, as shown in Fig. 5.3.

To reduce the number of required BUs, we can perform orthogonal "projections" on the butterfly array. By the first "projection", we perform parallelization on the stage level. The other projection yields the already known 'pipeline FFT' (section 5.2.1).

Although very advantageous for any kind of high-speed applications, exploitation of these degrees of data parallelism by a single-chip imple-
Figure 5.3: Potential approaches to exploit the data parallelism inherent to radix-4-based FFT computation, visualized at the example of a radix-4-based 16-point FFT. The “Stage” and “Pipeline FFT” approaches represent the orthogonal projections of the “Butterfly Array”.

<table>
<thead>
<tr>
<th>Degree of Parallelism</th>
<th>Corresponding Architecture</th>
<th>Number of APUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butterfly Array</td>
<td>![Butterfly Array Diagram]</td>
<td>$\frac{N}{4} \times \log_4 N$</td>
</tr>
<tr>
<td>Stage</td>
<td>![Stage Diagram]</td>
<td>$\frac{N}{4}$</td>
</tr>
<tr>
<td>Pipeline FFT</td>
<td>![Pipeline FFT Diagram]</td>
<td>$\log_4 N$</td>
</tr>
<tr>
<td>Single Butterfly</td>
<td>![Single Butterfly Diagram]</td>
<td>$1$</td>
</tr>
</tbody>
</table>
mentation is made more difficult due to underlying data dependencies. Furthermore, for actual realization of data-parallel computation, multiple butterfly units are required, leading to a voluminous implementation. Monolithic IC implementation of the APU is thus rather improbable, and hence no 'one-processor-system' can be realized. In addition, great communication is imposed among the set of butterfly units.

Hence, targeting a single-chip processor, we acknowledge processing all radix-4 butterflies by a single butterfly unit as the most convenient approach (cf. Fig. 5.3). The parallel computation of a single radix-4 butterfly is then required in order to achieve maximum data throughput.

Parallelization of Radix-4 Butterfly Computation

Looking at the structure of a single radix-4 butterfly (Fig. 5.4), parallelization of butterfly computation can also be achieved by either exploiting data or function parallelism. Concerning data parallelism, multiple radix-2 sub-butterflies can be computed in parallel. However, actual implementation is made more complicated by underlying data dependencies. As a result, on a 'single butterfly' level, exploitation of function parallelism is more convenient.

Arranging Arithmetic Resources

In order to allow for function-parallel butterfly computation, we must provide an appropriate set of arithmetic resources. Aiming at the most efficient architecture, the complete utilization of these arithmetic resources must thereby be achieved.

In conjunction with efficiency the term “matched system” is commonly referred to for an optimally-balanced architecture. To realize a matched APU we must basically ensure that butterfly computation time is equal to memory cycle time. Employing two-port memory, two complex words can be simultaneously read-out and written back into local memory with each clock cycle. Consequently, one piece of output data must be produced for each piece of input data clocked into the APU. It is fair to consider this concept to be optimum since a decrease in only either computation or memory access time will not result in any improvement of overall performance.

The goal of computing one butterfly result with each clock cycle allows us to narrow down combinations of number and type of arithmetic resources allowing for efficient utilization. To achieve maximum utiliza-
5.3. The New Data Path FFT Architecture

Figure 5.4: Decoupling and mapping of a radix-4 butterfly on corresponding arithmetic resources. The two complex adder/subtractors (CADD) compute the temporary results \(a, c, b, d\) and \(e, f, g, h\) in sequence, respectively, while the complex multiplier (CMUL) computes the products of 'final-stage' temporary results and corresponding twiddle coefficients (also in sequence). According to this mapping, each of the arithmetic resources performs four "operations" until a complete radix-4 butterfly is computed.

Our approach to exploiting function parallelism for radix-4 computation is thus to simultaneously compute results of the 'entry-stage' \(\{a, b, c, d\}\), the 'final-stage' \(\{e, f, g, h\}\) and overall radix-4 butterfly results \(X[0\ldots3]\). To do so, we assign a complex adder (CADD) to each of the radix-2 sub-stages and one complex multiplier (CMUL) to the twiddle multiplication part. The resulting mapping is shown in Fig. 5.4. According to this mapping, one of the radix-4 butterfly's results \(x[0], x[1], x[2], x[3]\) can be computed with each clock cycle.
5.3.3 Considerations on Efficient Resource Utilization

After determining the strategy for hardware mapping, we must now ask how the new butterfly unit can be most efficiently utilized.

Efficiency can be quantitatively described as the percentage of time that the arithmetic resources are kept busy during butterfly computation. Referring to our efficiency definition, we have to distinguish between two major 'time' scales:

Clock Cycle: Given the goal of maximal exploitation of the pipeline concept, pipeline registers must be strategically placed, i.e., the transmission delay of combinational circuit fragments framed by two pipeline registers should be approximately the same for the entire data path. Otherwise, performance surplus of the 'faster' circuit fragments remains unutilized. As a result, obtaining efficiency on the clock cycle level is mainly a matter of appropriate placement of pipeline registers and not an architectural concern.

"Operation" Cycle: The second time scale refers to the timely distribution of arithmetic operations on available resources. To make the butterfly architecture 100% efficient, matching data must be available for all arithmetic resources in each clock cycle.

Efficiency on "operation" cycle level is basically determined by the underlying operation scheduling and hence is also a question of required data accesses.

Requirements on Local Data Access

Looking at the chosen hardware mapping (Fig. 5.4), we see that two complex adders and one complex multiplier are implemented. As a result, a total of six complex operands must be provided with each clock cycle in order to allow for efficient utilization of these three arithmetic resources.

However, recalling the requirement for high data throughput, pipelined computation is the most promising option. For this, decoupling of 'entry-stage', 'final-stage' and 'twiddle multiplications' is realized by means of pipelining. On the other hand, due to the data dependencies inherent to the radix-4 butterfly structure, efficient resource utilization is difficult for pipelined data stream processing. In contrast to known approaches to achieving 100% efficiency (e.g., over-
5.3. The New Data Path FFT Architecture

lapped FFT, multiple clock systems), we decide on another technique for efficient utilization of arithmetic resources: pipeline interleaving.

Pipeline Interleaving

The basic idea of pipeline interleaving is to achieve efficient utilization of arithmetic resources by means of appropriate switching pipeline tasks which are assigned to this resource. In the ideal case, the resource is constantly kept busy over time by bridging the wait states of "idle" tasks. The employment of pipeline interleaving techniques for efficient digital filtering is covered in [Jiang 97]. Basic concepts of pipeline interleaving design for FIR, IIR and FFT array processors are discussed in [Chen 95].

5.3.4 Interleaved Computation of Radix-2 Sub-Butterflies

According to the structure of our butterfly unit (Fig. 5.4), we most efficiently perform pipeline-interleaving on the radix-2 sub-butterfly level. To explain the underlying principle, we focus on the computation of ‘entry-stage’ results first.

'Entry-Stage' Computation

Concerning the 'entry-stage', the underlying principle of “task-switching” is illustrated in Fig. 5.5. The top row in this figure shows the order of input data clocked into the arithmetic unit for pipeline processing (x[1], x[3], x[0], x[2]). This data represents the input to the bd- and ac-radix-2 butterfly of the ‘entry-stage’. Until not both x[1] and

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
<th>#10</th>
<th>#11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butterfly &quot;b/d&quot;</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Butterfly &quot;a/c&quot;</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CADD #1</td>
<td>NaN</td>
<td>b</td>
<td>d</td>
<td>c</td>
<td>a</td>
<td>b'</td>
<td>d'</td>
<td>c'</td>
<td>a'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.5: Detailed scheduling of 'entry-stage' radix-2 sub-butterflies on CADD#1.
Figure 5.6: Entry-stage commutator buffer: a) Detailed structure: 1:4 commutator, size-4 buffer, 4:2 commutator. b) Icon for indicating internal data-flow selection.

$x[3]$ are available, the “bd-butterfly queue” assigned to CADD#1 is pending (indicated by a “P”), i.e., the processing of this task is disabled due to the lack of required data. With cycle number three, the bd-butterfly becomes runnable (“R”) and the corresponding temporary results $b$ and $d$ can be computed during two successive clock cycles. The corresponding status of the ac-butterfly is also depicted in Fig. 5.5. Each time a butterfly queue is runnable, this task is “switched” to the assigned arithmetic resource, i.e., CADD#1.

Looking at the bd- and ac-butterfly-queue (Fig. 5.5), corresponding ‘runnable-states’ are alternating every two clock cycles. To implement pipeline-interleaving on radix-2 level, we thus buffer all incoming data that cannot immediately be used for processing. Then, as soon as sufficient data is available in the buffer, it will be provided to the assigned arithmetic resource. By using an appropriate buffer strategy, this approach allows for the continuous utilization of CADD#1.

Referring to the ‘buffering’, two basic requirements exist: first, the buffer must provide two outputs in order to feed the following CADD. Second, to allow for efficient utilization of this CADD, random access on buffered data must be supported. Hence, our approach is to implement a dedicated commutator buffer for transparent reorganization of data-flow. The scheduling pictured in Fig. 5.5 also helps to determine the minimal buffer size. Since $x[1]$ and $x[3]$ are required to compute the bd-butterfly, two buffer elements are needed to ensure their being provided to CADD#1 during cycles number three and four. To avoid the loss of $x[0]$ and $x[2]$, which are clocked in within this time, two additional buffer elements are required, resulting in a total of four buffer elements.

Our approach to realizing the commutator buffer is thus to combine
5.3. The New Data Path FFT Architecture

Figure 5.7: Switching patterns for 'entry-stage' commutator buffer (cf. Fig 5.6) according to the scheduling given by Fig. 5.5. For each commutator state, the corresponding input data is listed above the commutator icon.

A 1:4 commutator (ring-buffer pointer), a size-4 buffer and a 4:2 commutator (actual “switch” performing the pipeline-interleaving). The resulting structure of the commutator buffer is shown in Fig. 5.6. By means of appropriate switching, we line up the partly processed samples in exactly the way they are needed.

Continuing pipelined data stream processing by clocking in the succeeding input quartet, denoted by \( x[i] \)’ in Fig. 5.5, the old buffer contents can be overwritten. To do so, we simply maintain the order of switching the 1:4 and 4:2 commutators.

The switching patterns are shown in Fig. 5.7. After a short lead-in phase, all commutator states follow a prescribed repetitive pattern.

'Final-Stage' Computation

While a size-4 commutator buffer allowed for efficient utilization of CADD#1, the situation becomes more complex for the 'final-stage'. This is basically due to the different address offsets required to access data associated with 'entry-' and 'final-stage' radix-2 sub-butterflies, respectively. As a result, we have to modify our buffer strategy developed for the 'entry-stage'.

Referring to Fig. 5.4, possible sequences of 'entry-stage' sub-butterfly results are given by \{a-c/c-a\} and \{b-d/d-b\}. Concerning the decision for a specific order, our objective is to enable the most efficient option for processing the 'final-stage', i.e., minimization of required buffer-size and latency. Looking at the 'final-stage', required data pairs for corresponding radix-2 butterflies are \{a-b/b-a\} and \{c-d/d-c\}. One preferred output order of the 'entry-stage' is thus \{bd ca\}. 
Figure 5.8: Detailed scheduling of 'final-stage' radix-2 sub-butterflies on CADD#2 and corresponding twiddle multiplications.

Assuming this output order, the $gh$-butterfly can be computed after the third clock cycle, i.e., when both $d$ and $c$ data are available (cf. Fig. 5.8). Proceeding the sequential input to the 'final-stage' by supplying the $a$ value in the next cycle, the $ef$-butterfly becomes "runnable". At this time, a complete quartet of related temporary data is stored in a corresponding size-4 buffer. However, in contrast to the 'entry-stage', feeding the following complex adder has been performed for just one clock cycle. Consequently, we cannot overwrite the $c$ and $d$ storage places since this data is still needed for one additional clock period. Since the $a$ and $b$ values have not been used at all, we need an additional buffer element to store the forthcoming $b'$-value of the following quartet.

In consequence of this fifth storage element, we have to modify the 'final-stage' commutator buffer in that we have to use a 1:5 commutator,
5.3. The New Data Path FFT Architecture

Figure 5.10: Switching patterns for 'final-stage' commutator buffer (repetitive part) according to the scheduling given by Fig. 5.8.

a size-5 buffer and a 5:2 commutator (Fig. 5.9). As for the 'entry-stage', the switching patterns related to these commutators follow a regular pattern after a dedicated lead-in phase. The repetitive part of these patterns is depicted in Fig. 5.10.

Referring to the implemented pipelined processing, the order of processing input and intermediate data was deliberately chosen to simplify the requirements concerning address generation. As a result, input data clocked into the first commutator buffer (cf. Fig. 5.5) and data clocked out of the CMUL (cf. Fig. 5.8) are addressed in the same order. Addresses for processed samples \( X[j] \) are thus easily obtained by appropriately delaying the address-sequence of the corresponding \( x[j] \)-values. Thus, only one address generator is required.

### 5.3.5 Resulting FFT Architecture

The resulting FFT architecture, consisting of two complex adders, one complex multiplier and two commutator buffers is shown in Fig. 5.11.

With each clock cycle, two temporary results for 'entry-stage' and 'final-stage' are computed in addition to the actual butterfly result. The

![Figure 5.11: Pipelined FFT data path architecture.](image)
computation of a single radix-4 butterfly is thereby accomplished every four cycles. For convenience, the formation of registerfiles and CADDs is referred to as ‘adder part’ in the following discussions.

5.4 Design of Complex Multiplier

Referring to our concept of continuous data stream processing, the complex multiplier (CMUL) must compute one result with each clock cycle. In addition, referring to overall chip size, the complex multiplier must allow for area-efficient implementation. Furthermore, low power consumption is a desirable feature. Targeting a corresponding implementation, we will discuss relevant concepts for real arithmetic multipliers first.

5.4.1 Relevant Multiplier Concepts

During the last decades, several concepts evolved, each aiming at the most efficient multiplier design. Despite this diversity, we can divide multipliers into two major categories: clocked multipliers and array multipliers. However, in both types, the product is obtained by summing up partial products in sequence.

One representative of clocked multipliers is the add-shift multiplier discussed in [Rabiner 75]. Due to the inherent sequential computation, clocked multipliers can be implemented with comparatively small area but with the disadvantage of limited data throughput.

5.4.2 Array Multipliers

The fastest multipliers are realized by a 2D array of one-bit adders and are referred to as array multipliers. According to the parallelism inherent to array multipliers, the final result can be obtained without registering intermediate results. There are many array configurations which can be categorized according to the method of underlying adder-connection and according to the way negative numbers are handled. Usually, carry-save techniques are applied to achieve faster propagation of carry bits.

According to the structure of an array multiplier, two methods exist to achieve a speed-up of the underlying multiplication. First of all, the array multiplier is characterized by a very regular structure and hence allows for “convenient” implementation of pipeline stages (in particular,
when the addition of generated partial products is realized by means of carry-save arithmetic). The second approach is to accelerate the accumulation of all partial products. This can be achieved by either reducing the number of partial products or by using fast addition schemes.

### Optimized Partial Product Generation

**Booth-Encoding.** Booth-encoding was first proposed in [Booth 51]. The original purpose of this encoding technique was to allow for convenient multiplication of two's complement numbers.

In contrast to original Booth-encoding, the modified Booth technique developed by Mac Sorley operates on sequences of three neighboring bits [Mac Sorley 61]. The number of resulting partial products is thus divided in half. Extending this concept, the total number of partial products can be reduced further by enlarging the number of bits supplied to actual encoding. Area requirements of the corresponding encoder stage are then significantly increased and are not compensated for by the reduced adder-matrix.

### Efficient Accumulation of Partial Products

**Wallace Adder Trees.** Wallace proposed a dedicated carry-save adder tree for the fast accumulation of partial products [Wallace 64]. The Wallace adder tree basically uses full-adder cells to logarithmically reduce the set of input bits to two final operands. Recently, 4:2 compressor cells have been used instead of their full-adder counterparts to realize the carry-save adder tree. Architectures for pipelined Wallace tree multipliers are discussed in [Pang 90].

**Dadda Addition Scheme.** A further addition scheme was proposed by Dadda [Dadda 65, Dadda 76]. Starting with a matrix arrangement of bits representing the partial products, the underlying strategy is to reduce the number of bits by means of ‘counters,* in a stepwise fashion. As for the Wallace tree, two partial sums which are to be added by means of fast CLA adders will remain at the end. Although both Wallace tree and Dadda addition scheme show the same number of ‘stages’ for a given number of partial products, transistor count of the Dadda addition scheme is slightly lower.

*A counter is considered an arithmetic element, the binary output of which indicates how many of its \( n \) inputs are "1" (\( n=3 \): full-adder, \( n=2 \): half-adder).
5.4.3 Implementation

Looking at the multiplication of two complex numbers, $\zeta = a + jb$ and $\eta = c + jd$, four real multiplications and two real additions are required, as stated by Eq. (5.1).

$$\zeta \cdot \eta = (a + jb) \cdot (c + jd) = (ac - bd) + j(bc + ad)$$

However, targeting high data throughput, limitations imposed by the design library required the implementation of a comparatively large array multiplier (cf. section 8.2.2). Hence, to achieve a compact VLSI implementation, we had to reduce the number of real multipliers as much as possible.

Employed Algorithm for Complex Multiplication

One approach to reducing the number of required real multiplications is to use a dedicated computation procedure, as described by Burrus ([Burrus 85]). Referring to this procedure, underlying operands $\zeta$ and $\eta$ are appropriately pre-processed giving three different input quantities

$$Z = a(c - d), \quad D = a + b, \quad E = a - b, \quad (5.2)$$

which are then computed to yield real and imaginary parts of the final product:

$$\Re = Ed + Z, \quad \Im = Dc - Z.$$  \hspace{1cm} (5.3)

As a result, complex multiplication is realized with only three real multiplications$^\dagger$. However, the price for the saved multiplication is paid by three further additions.

On the other hand, an addition is much more compact to implement in silicon than a multiplier. Furthermore, assuming the case that one multiplicand is known in advance (e.g., twiddle coefficient), the new input quantities $D$ and $E$ can be pre-computed. The number of required additions is thus reduced to three. However, the need to support 'elementwise multiplications' prevents application of this pre-computation technique.

$^\dagger$In fact, we can derive a total of 16 different but equivalent procedures to reduce the number of required multiplications from four to three. However, since they all require the same number of additions/subtractions and multiplications, in our case, no preferable procedure exists.
5.4. Design of Complex Multiplier

**Resulting CMUL Structure**

The resulting structure of the complex multiplier is shown in Fig. 5.12. It directly transposes the 'Burrus algorithm' described above.

Compared to a straightforward implementation of the complex multiplication, area requirements are reduced by approximately 20% due to the saved real-arithmetic multiplier. On the other hand, referring to the final layout of the CMUL (cf. Fig. 8.4), compact arrangement of the three multipliers imposes considerable requirements on data routing. However, employing a three metal layer CMOS-process, we proved actual routing-overhead to be tolerable.

To achieve high data throughput, the CMUL structure depicted in Fig. 5.12 is heavily pipelined. At least each of the real-multipiers contains 16 pipeline stages. The CMUL can then sustain computation of one complex product with each clock cycle.

![Complex-multiplier structure diagram](image)

**Figure 5.12:** Complex-multiplier structure according to the complex multiplication technique described by Eqs. 5.2 and 5.3.
5.5 Implementation of Additional BST Low-Level Tasks

The APU architecture derived in the preceding sections allows only for radix-4-based FFT computation. However, recalling our original processor concept, all BST-related low-level tasks have to be supported. As a result, we have to implement additional functionality into the butterfly unit. For this, referring to area requirements, the major objective is to only perform minimal modifications to the derived FFT architecture.

To minimize the need for additional functional blocks, our approach is to determine synergies between radix-1 butterfly computation and remaining low-level tasks first. Then, based on these synergies, we map all additional low-level tasks on the existing FFT architecture as completely as possible, and in doing so look for missing features. Finally, we implement the as yet non-supported functionality and add modifications to data-flow.

5.5.1 Fixing Required Functionality

Referring to the algorithm flow of Fourier domain BST computation (Fig. 2.8), computation of both forward and inverse FFT is required. However, as discussed in section 2.4, we can compute the inverse FFT by a combination of forward FFT and pre-processing on the template’s spectrum. To do so, we have to modify the tasks for 'elementwise multiplication' and 'scramble'. As a result, actual computation of the inverse FFT must not be implemented into the arithmetic unit, allowing for lean controlling.

Concerning the set of image sizes that must be supported, corresponding 256-point and 1024-point FFTs can be computed by the radix-4 algorithm. However, this is not the case for a 512-point FFT. Since 512 is not a power of four, the length 512 sequence must be split into two 256-point partitions. The results of corresponding 256-point sub-transforms must then be combined by a final radix-2 stage to obtain the desired length 512 spectrum. The APU must then also support computation of radix-2 butterflies. In addition, we have to map 'elementwise multiplication' and the real-valued FFT’s pre- and post-processing tasks (cf. section 2.3.2) onto the APU architecture.

Looking at the arithmetic requirements of these additional tasks, the computation of radix-2 butterflies can be achieved by employing the 'final-stage' CADD and the CMUL. Also, 'elementwise multiplication' can
basically be accomplished by means of the complex multiplier. Finally, computation of the RFFT's pre- and post-processing tasks (scramble, unscramble) can be decomposed into additions, subtractions and divisions by two (e.g., shifts). Hence, they are supported by the CADDs for the most part.

The elementary function set is then given by addition/subtraction, multiplication and shift. Since the first two operations are supported by the CADDs and the CMUL, a large synergy exists among the set of BST-related low-level tasks. In fact, additional functionality is required merely for the RFFT's pre- and post-processing tasks.

### 5.5.2 Extended FFT Architecture

As discussed before, all low-level tasks can mainly be computed by the CADDs and the CMUL. However, in order to allow for complete execution of these tasks on the APU architecture, additional functionality must be implemented. Furthermore, increased flexibility in data-flow is required to allow the data to pass through the arithmetic units in the necessary order. Two main modifications must then be applied to the FFT architecture shown in Fig. 5.11.

#### Application Specific Registerfiles

According to the algorithmic requirements of 'elementwise-multiplication' and RFFT pre-/post-processing tasks, additional support for arithmetic shifts (ASR) as well as for swap and copy of real and imaginary parts must be implemented. Our approach to providing this additional functionality is to extend the commutator buffers since they are optimally located before each of the two CADDs. To do so, we implement an additional functional-block along with the actual buffer elements.

Concerning concrete realization of these extended commutator buffers, we decide on application-specific registerfiles in order to guarantee fast write- and read-access. The underlying structure of a registerfile block is depicted in Fig. 5.13. As can be seen, a registerfile block comprises a total address-space of seven storage places.

Recalling the requirements on the commutator buffer, two output operands must be provided to feed the following CADD. For this, according to the required random access, corresponding multiplexers have to be connected to each of the buffer elements. Since inherent multiplexer
delay increases with the number of connected ports, we divide total address space into three sub-blocks. As a result, assuming appropriate data distribution, the two multiplexers can be separately connected to the sub-blocks, as shown in Fig. 5.13. The maximum number of multiplexer ports is thus reduced to four, allowing for tolerable switching-delays.

Two of the three sub-blocks are for data-shuffling and the storage of intermediate results. The additional functional block comprises two storage places of the registerfile's address-space. Data written into each of its corresponding addresses is subject to fixed arithmetic modification. Supported operations are swapping, shifting and copying of real and imaginary parts.

**Reconfigurable Data-Flow**

The second design modification is imposed by the requirement for flexible data flow. While for the computation of radix-4 butterflies the underlying data is shifted through the established arrangement of two CADDs and one CMUL in sequence, a different sequence of these arithmetic resources is required for the other BST low-level tasks.

To achieve the required reconfigurability, we realize flexible data-flow by means of multiplexers. In addition, each of the two registerfile blocks can alternatively take its input data from a feedback path connected to the corresponding CADD. This modification allows for sequential additions and subtractions, as required by the RFFT's pre- and post-processing tasks.

Figure 5.13: Extended commutator buffer (registerfile).
5.6 Resulting APU Architecture for FD BST Computation

The structure of the extended APU architecture is depicted in Fig. 5.14. The basic components are two register files storing intermediate results and performing data reorganization as well as two complex adders (CADD) and one highly pipelined complex multiplier (CMUL).

In contrast to the native FFT architecture shown in Fig. 5.11, four additional multiplexers are implemented, allowing for individual data-flow configuration for each of the supported BST low-level tasks.

The architecture disposes of two input-ports (connected to local memory and coefficient ROM) and one output-port (local memory). Based on pipelined data processing, the architecture thus allows for sequential computation of results, thereby achieving a constant data throughput of 1.2 GByte/second at 66MHz.

Although originally derived for the efficient computation of radix-4 butterflies (cf. Fig. 5.4), the implemented arithmetic resources are also efficiently utilized for the remaining tasks, as shown in Table 5.1.

In addition, according to the highly pipelined data processing, the resulting APU architecture allows for simple porting to advanced process technologies.

Figure 5.14: Block diagram of the extended arithmetic data path unit (APU).
## Table 5.1: Subdivided utilization of APU resources

<table>
<thead>
<tr>
<th>Mode</th>
<th>Regfile#1</th>
<th>CADD#1</th>
<th>Regfile#2</th>
<th>CADD#2</th>
<th>CMUL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT-R4</td>
<td>4/7 = 57%</td>
<td>100%</td>
<td>7/7 = 100%</td>
<td>100%</td>
<td>75%(^a)</td>
</tr>
<tr>
<td>FFT-R2</td>
<td>0/7 = 0%</td>
<td>0%</td>
<td>4/7 = 57%</td>
<td>100%</td>
<td>75%</td>
</tr>
<tr>
<td>MUL</td>
<td>3/7 = 43%</td>
<td>50%(^b)</td>
<td>3/7 = 43%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>UNSCR</td>
<td>7/7 = 100%</td>
<td>50%</td>
<td>7/7 = 100%</td>
<td>50%</td>
<td>0%</td>
</tr>
<tr>
<td>SCR</td>
<td>5/7 = 71%</td>
<td>50%</td>
<td>7/7 = 100%</td>
<td>50%</td>
<td>0%</td>
</tr>
</tbody>
</table>

\(^a\)Although the CMUL computes one result with each clock cycle for the two FFT modes, multiplications with "1" are considered unnecessary, hence reducing the CMUL's efficiency from 100 to 75%.

\(^b\)The corresponding arithmetic resource computes a practical result with every second clock cycle.

Table 5.1: Subdivided utilization of APU resources. "Register-Usage" specifies the number of registers that are employed for the corresponding mode (each registerfile holds a total of seven registers). "Time-Usage" specifies the percentage of time that the corresponding resource is utilized over the entire processing period.
6. High Precision Arithmetic Concept

In this chapter, arithmetic concepts commonly employed for FFT computation are rated according to corresponding accuracy and implementation costs. During this discussion, we put the main focus on fixed-point arithmetic and its related problems. In particular, referring to finite wordlength effects, we develop a statistical model to analyze the impact of quantization errors on the precision of corresponding FFT results.

To reduce the required fixed-point wordlength, we propose block exponent (BE) arithmetic. As a result, we achieve excellent detection performance over a large variety of image materials by implementing block floating-point arithmetic with 64\textit{b} mantissa (real and imaginary part) associated with one corresponding 8\textit{b} exponent. Compared to an equivalent 2\times47\textit{b} fixed-point implementation, we thus achieve a considerable area reduction of 71%.

6.1 Precision Requirements

A minimum precision must be warranted for a particular object detection application. In the case of FFT-based correlation, the accuracy required to compute the desired object location(s) strongly depends on the underlying image material. Image size, image background and lighting conditions play a very important role in reference to the detection sensitivity of the BST algorithm. The situation becomes even more difficult when using templates of arbitrary shape (e.g., free-form templates); this reduces the number of pixels that can be used to compute the BST similarity measure.

To visualize the detection performance of the BST algorithm, Fig. 6.1(a) depicts an example search area of size 256 \times 256 (face) containing a 128 \times 128 template (padlock). The search area has an irregular
background and offers weak contrast. In addition, the 'padlock' template is of non-rectangular shape.

Figure 6.1(b) shows the results of the BST algorithm represented by a 2D gray-scale intensity matrix. Coordinates with light colors indicate high correlation measures. The distinct white point in the upper-left area indicates the position with the highest similarity measure, thus giving the position of the template. A 3D representation of the BST correlation measures is depicted in Fig. 6.2.

In the above example, the template’s reference point (upper-left corner) is located at position \( \{ x = 40, y = 40 \} \). As expected, the BST algorithm identifies this position as the location of the object by the absolute maximum of the 2D result matrix. This maximum is associated with a measure of 6674.6. The highest result next to the absolute maximum is located at position \( \{ x = 34, y = 38 \} \) with a measure of 6391.2. Thus, in this example, the measure for the correct object position differs from the largest 'false' peak by 4.4%, corresponding to a signal-to-noise ratio of 0.38 dB. Thus, in this example, the object detection system must provide a resolution of at least 2% in order to allow for determination of the desired position.

Unfortunately, with this accuracy, distinct peak identification cannot be guaranteed for other correlation cases. Depending on template shape,
6.1. Precision Requirements

Figure 6.2: 3D relief of correlation results. The correlation peak is indicated by the arrow.

template texture and background pattern*, height-dominance of the object-identifying peak can be very small. Even worse, though rarely, actual object positions can show a smaller similarity measure than other, no-object positions†. As a result, while an accuracy of 2% is sufficient for the example shown in Fig. 6.1, considerably higher precision may be required in other cases.

However, since we have to choose for the most appropriate arith-

---
*These characteristics determine the spectral representation of the original search area, the search area with embedded template(s) and the isolated template, respectively. In cases where the spectrum of the isolated template is not too much distorted by the original search area’s spectral components, less accuracy in BST computation is tolerable.
†This problem also exists for phase correlation-based 'motion estimation'. Here, to overcome this problem, potential object positions are additionally verified by means of space domain correlation. However, for the targeted real-time system, this is not an appropriate option since it involves the same problems as for hierarchical search (cf. section 2.2.1).
metric type, we have to fix the required system precision. To do so, we empirically determined a 'generic accuracy', i.e., the accuracy fitting the precision requirements of almost every correlation case. Hence, we performed several correlations based on representative image material. During the evaluation process, lighting conditions and background texture have been varied. In particular, the impact of non-rectangular templates on the detection sensitivity has been investigated.

As a result, aiming at a general, flexible system and thinking of processing $1024 \times 1024$ search areas, a maximal computation error of 0.4% must be guaranteed for the BST similarity measures. However, as discussed above, there might be a small number of correlation cases requiring higher accuracy. For those cases, we cannot guarantee a precise location of object positions.

Facing the accuracy requirement mentioned above, the processor arithmetic must be chosen by carefully screening potential concepts. We will therefore discuss relevant arithmetic types in the following.

### 6.2 Floating-Point Arithmetic

According to the properties of the 2D-FFT, the corresponding spectrum is characterized by large numeric dynamic. Since floating-point arithmetic allows for the largest dynamic range, the highest precision is obviously achieved by this arithmetic type.

A floating-point entity is represented by two fixed-point numbers, the mantissa $m$ and the exponent $a$. A floating-point number is then expressed as the product of $m$ and a base (in our case two) raised to the power of $a$:

$$f = m \cdot 2^a. \; ^\dagger$$

The major disadvantage of floating-point arithmetic is its large expense in regard to VLSI implementation. Both floating-point multiplication and addition are more complex to implement than their fixed-point counterparts.

In particular, a floating-point adder has a much more complex structure compared to a fixed-point adder, resulting in an area overhead of about factor 10 (cf. [Quach 91]). This area overhead comes from the fact that appropriate mantissa pre-processing is required prior to the actual addition. First, the mantissa of the smaller number must be aligned

\(^\dagger\text{Usually, } m \text{ is normalized, i.e., } \frac{1}{2} \leq |m| < 1.\)
according to the exponent of the larger operand. Then, the resulting mantissas are added using regular fixed-point arithmetic. Finally, this sum must eventually be renormalized, requiring the corresponding exponent to be readjusted.

Referring to total chip size, implementation of single precision floating-point arithmetic implicates an increase in required silicon area of approximately 20% as compared to 32b fixed-point arithmetic [Wosnitza 98].

6.3 Fixed-Point Arithmetic

Hence, since we target a highly compact VLSI implementation, fixed-point arithmetic is our choice.

As indicated by nomenclature, in fixed-point representation the position of the binary point is fixed. The integer part is represented by the bits to the left of the binary point while the fractional part is given by the bits to the right of the binary point.

Although fixed-point arithmetic allows for compact implementation, it inherently imposes a dynamic range limitation on the corresponding FFT computation. This limitation is given by the possibility of overflows during fixed-point additions. Hence, we have to employ a minimum wordlength to achieve our target precision. To determine required wordlength, we must also consider finite wordlength effects which induce further inaccuracies throughout the correlation process.

6.3.1 Finite Wordlength Effects in FFT Computation

An important aspect of fixed-point arithmetic is the impact of finite wordlength effects on the precision of computed FFT results.

Based on a limited wordlength, both sampled input data and coefficients are represented with finite precision only. This quantization of coefficients results in a difference between the FFT’s transfer function and the ideal one. For a general fixed filter function this effect is known as the coefficient sensitivity of the filter. Coefficient sensitivity is a deterministic type of error that can be judged ahead of time. However, general evaluation of a certain filter class, as for the FFT, calls for statistical measures of coefficient sensitivity and is discussed in section 6.3.3.
Noise Induced During FFT Computation

Looking at the FFT algorithm proposed by [Cooley 65], computation is basically achieved by accumulation of products resulting from twiddle multiplications:

\[ X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk} \quad k = 0, 1, \ldots, N - 1 \]  \hspace{1cm} (6.1)

where \( W_N = e^{-j \frac{2\pi}{N}} \). Referring to Eq. (6.1), in the case of a \( b \)-bit fixed-point arithmetic, the result of each arithmetic operation must be scaled if it cannot be represented by the given \( b \) bits. By this scaling, additional noise which can be categorized into two main classes is introduced within the computation [Rabiner 75]:

**Round-off noise.** For fixed-point arithmetic, multiplication of two \( b \)-bit operands results in a \( 2b \)-bit product. To avoid a continuous increase of required wordlength, it is necessary to quantize the multiplication results to the available accuracy. Two standard methods of eliminating the lower bits of the product exist: truncation and rounding. For rounding, the result is replaced by the \( b \)-bit number closest to the original unrounded quantity. By truncation, bits of lower order are simply discarded. A round-off error is thus induced when computing the product \( x[n] \cdot W_N \) (cf. Eq. 6.1). This error is uniformly distributed between \(-\frac{1}{2} \cdot 2^{-b} \) and \( \frac{1}{2} \cdot 2^{-b} \), with zero mean and variance \( \sigma^2 = 2^{-2b}/12 \).

**Scaling noise.** For fixed-point addition, the required wordlength will increase by one bit per addition only in the case of overflow. In this case, scaling of the result is required. The error induced by this right-shifting will be either 0 or \( \pm 2^{-b} \), depending on the contents of the out-shifted bit. Assuming equal probability for a shifted '1' and '0', in the case of two's complement representation, the variance of this error is \( 2^{-2b}/2 \).

However, in applications where an overflow is rather unlikely and/or doesn't affect the result in a relevant way (e.g., a small 2D-filter kernel for image thresholding), scaling can be avoided by using dedicated saturation arithmetic, limiting the output to the most positive or negative representable number.

Round-off noise and scaling noise will continuously increase inaccuracy during computation. The choice of an appropriate rounding mechanism (e.g., **ROUND** or **TRUNC**) and the time when a required scaling (e.g.,
6.3. Fixed-Point Arithmetic

SHIFT) of an operand is performed have a large impact on the precision of the result. As shown in section 6.3.3 in terms of FFT computation, a proportional dependency between the output noise and the transform length \( N \) can be generally assumed.

6.3.2 Precision-Reduction Induced by Overflows

As stated by Parseval's theorem (Eq. (6.2)),

\[
\sum_{n=0}^{N-1} x^2(n) = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2, \quad (6.2)
\]

the mean square value of the transform \( X \) is \( N \) times the mean square value of the input sequence \( x \). Thus, the DFT of a sequence tends to be significantly larger in magnitude than the sequence itself.

The same effect is also observed for stage-oriented FFT processing. Each radix-4 butterfly shown in Fig. 2.6 operates on four complex input numbers and computes four new complex results given as

\[
\begin{align*}
X_{m+1}(i) &= (X_m(i) + X_m(k) + X_m(j) + X_m(l)) \cdot W \\
X_{m+1}(j) &= (X_m(i) - X_m(k) - jX_m(j) + jX_m(l)) \cdot W \\
X_{m+1}(k) &= (X_m(i) + X_m(k) - X_m(j) - X_m(l)) \cdot W \\
X_{m+1}(l) &= (X_m(i) - X_m(k) + jX_m(j) - jX_m(l)) \cdot W, \quad (6.3)
\end{align*}
\]

where \( W \) is a complex root of unity (cf. Eq. (6.1)). Looking at Eq. (6.3), we conclude that the root-mean-square increases by a factor of two at each radix-4 stage:

\[
\sqrt{\frac{|X_{m+1}(i)|^2 + |X_{m+1}(j)|^2 + |X_{m+1}(k)|^2 + |X_{m+1}(l)|^2}{4}} = 2 \cdot \sqrt{\frac{|X_m(i)|^2 + |X_m(j)|^2 + |X_m(k)|^2 + |X_m(l)|^2}{4}}. \quad (6.4)
\]

However, to minimize initial quantization errors (cf. section 6.3.1), we have to completely exploit mantissa capacity from the start of the FFT computation. Hence, when using fixed-point arithmetic we must employ scaling procedures to prevent overflow. On the other hand, with each scaling operation we discard mantissa information, thereby reducing accuracy of FFT results.
Strategies for Sequence Rescaling

When targeting a maximum shift of one bit per scaling iteration, the sequence must be checked for overflow after each radix-2 sub-stage. That is, any potential overflow must be eventually corrected after both the radix-4 butterfly’s entry and final radix-2 sub-stage.

To correct overflows occurring during FFT computation, one of the following scaling methods can be applied (cf. [Welch 69, Rabiner 75]):

Right-Shift at Every Radix-2 Sub-Stage. If for all elements of the first stage \(X_0(i)\) the relation \(|X_0(i)| < \frac{1}{2}\) holds, an overflow is prevented by right-shifting all results after each computed radix-2 sub-stage by one bit (except the last sub-stage). This scaling method is particularly characterized by its simplicity concerning implementation and data processing. On the other hand, because of many needlessly performed shift-operations, this method will also compute the most inaccurate results. Implementations of this scaling technique are realized in the architectures proposed in [Hui 96, Cetin 97].

Sequence Controlling. The main objective of this scaling method is to normalize all elements of sub-stage \(m\) to the condition \(|X_m(i)| < \frac{1}{2}\), \(\forall i\). If values larger than \(\frac{1}{2}\) are computed in a current sub-stage, all elements of this sub-stage are right-shifted by one bit. Sequence controlling is very time-consuming, since for all stage elements the corresponding magnitude has to be computed. In addition, by normalization to \(\frac{1}{2}\), one mantissa bit is basically unused.

Test for Overflow. In this case, the starting sequence is scaled according to the constraints \(|\Re\{X_0(i)\}| < 1\) and \(|\Im\{X_0(i)\}| < 1\), where \(\Re\) and \(\Im\) denote real and imaginary parts, respectively. In contrast to the two scaling methods described above, the normalization level is thus one. If an overflow is detected during the computation of a single radix-2 sub-butterfly, the whole sequence (including the results of successfully computed butterflies in this stage) is right-shifted by one bit. Computation continues after the radix-2 sub-stage in which the overflow occurred. The highest accuracy is achieved with this scaling method. However, a major drawback is given by the fact that in the case of an overflow, the entire sequence has to be re-processed.

To summarize, finite-arithmetic effects are manifested in coefficient quantization, rounding or truncation of intermediate results and correc-
tion of arithmetic overflows. Each of these effects results in an inaccurate transformation.

### 6.3.3 Analytical Model for Estimating Quantization Errors

Looking at VLSI implementation, the area required by arithmetic components, such as adders and multipliers, generally increases with wordlength. Targeting small-area implementation, wordlength must be kept as short as possible. Thus, the trade-off between area requirements and precision must be carefully evaluated for each application.

To do so, the objective of this section is to compute boundaries for the noise-to-signal ratio as a function of wordlength $b$. As shown by Welch ([Welch 69]), boundaries on the round-off noise generated in the FFT can be computed. Based on this approach, we develop an improved relation for the root-mean-square ratio on a statistical basis. To do this, we consider two sources of noise within the computation: round-off noise and scaling noise (cf. section 6.3.1).

We start our discussion by assuming worst-case conditions. Throughout this discussion we assume that all errors are independent, and hence the variance of the sum is the sum of variances. Furthermore, we assume initial quantization errors stemming from analog to digital conversion and propagating through the entire computation to be white with variance $\delta^2$. Also, for convenience, we will set the variance of the round-off noise $\sigma_1^2 = 2^{-2b}/12$ to $\sigma^2$. With this, scaling noise variance is given by $\sigma_2^2 = 2^{-2b}/2 = 6\sigma^2$.

### Worst-Case Consideration

For convenience, we decompose a single radix-4 butterfly into 'entry-' and 'final-' radix-2 sub-stages followed by a sequence of complex twiddle multiplications (cf. Fig. 2.6). Arithmetic additions have to be performed in each of the 'sub-operations'. Assuming worst-case conditions, at least one overflow will occur in each sub-operation, thereby requiring the sequence to be rescaled and thus introducing scaling noise.

According to our decomposition, a concrete measure for the output variance of the $k$-th radix-4 stage can be developed in a stepwise fashion.

\[ \frac{\text{rms} \text{(error)}}{\text{rms} \text{(result)}} \]
Entry-stage. According to the addition performed within a single radix-2 butterfly, the output variance $V'_k$ of the first radix-2 sub-stage basically computes to the sum of its input variances $V_{k-1}$:

$$V'_k = 2 \cdot V_{k-1} + 2^{2(\Psi - (k - 1)) + \Delta} \cdot 6\sigma^2. \quad (6.5)$$

The second term in Eq. (6.5) considers the scaling noise introduced due to the assumed overflow. Since scaling noise depends on the position where bit-truncation is performed, all prior scalings on the sequence have to be taken into account. At stage $k$, a total of $\Psi \cdot (k - 1)$ scalings have been applied, with $\Psi$ denoting the number of overflows occurring within a single radix-4 stage. We can thus correct the bare scaling noise of $6\sigma^2$ by a corresponding weight-factor. In this factor, an offset $\Delta$ is additionally introduced, allowing for convenient verification of the theoretical model developed in this section against real simulation results (see section 6.3.4). This offset considers initial scalings on the sequence to be transformed due to peculiar characteristics inherent to the simulation.

Final-stage. The output variance $V''_k$ of the second radix-2 sub-stage is computed in the same way. Taking the input variance $V'_k$ and considering the scaling operation performed during the first radix-2 sub-stage yields

$$V''_k = 2 \cdot V'_k + 2^{2(\Psi - (k - 1)) + 1 + \Delta} \cdot 6\sigma^2. \quad (6.6)$$

Twiddle Multiplications. In the following twiddle multiplications, the output values $X''_k$ of the second radix-2 sub-stage are multiplied with corresponding coefficients $W$. The multiplication result of $X''_k \cdot W$ thus becomes

$$(\Re\{X''_k\} + j\Im\{X''_k\}) \cdot (\Re\{W\} + j\Im\{W\}) = (\Re\{X''_k\}\Re\{W\} - \Im\{X''_k\}\Im\{W\})$$

$$+ j (\Re\{X''_k\}\Im\{W\} + \Im\{X''_k\}\Re\{W\}), \quad (6.7)$$

where $\Re$ and $\Im$ denote real and imaginary parts, respectively. To be consistent with the variance estimations derived in Eqs. 6.5 and 6.6, we must only consider the real or imaginary part of the product. Focusing on the real part’s variance gives

$$V\{X''_k \cdot W\} = |\Re\{X''_k\}|^2 \cdot V\{W\} + V\{X''_k\} \cdot \Re\{W\}^2 \quad (6.8)$$
According to the properties of the coefficients, \( \Re\{W\}^2 + \Im\{W\}^2 \) is unity. Furthermore, representing \( W \) with \( b \)-bit precision results in rounding noise. Thus, \( V\{W\} \) is \( \sigma^2 \) and we simplify Eq. (6.8) to

\[
V\{X''_k \cdot W\} = |X''_k|^2 \cdot \sigma^2 + V\{X''_k\}. \tag{6.9}
\]

For further simplification we recall that the average modulus squared (AMS) of the sequence under transform increases by a factor of four with every stage (Eq. 6.4). Furthermore, we proved the AMS of the complex multiplication results to be equal to the AMS of the \( X''_k \). By setting \( K \) to the AMS of the initial array \( (X_0) \),

\[
K = |X_0|^2 = \frac{1}{N} \sum_{i=0}^{N-1} |X_0(i)|^2, \tag{6.10}
\]

and also considering the round-off noise introduced by truncating \( b \) bits of the multiplication result, we rewrite Equation (6.9) as

\[
V_k = V''_k + 4^k K \sigma^2 + \left( 2^{2(\Psi \cdot (k-1)+2+\Delta)} + 6 \cdot 2^{2(\Psi \cdot (k-1)+2+\Delta)} \right) \sigma^2. \tag{6.11}
\]

In contrast to [Welch 69], Eq. (6.11) also considers scaling noise in addition to round-off noise, thereby taking into account the additions performed during each complex multiplication. By substituting \( V''_k \) and \( V'_k \) and by further simplification, we finally obtain the output variance of the \( k \)-th radix-4 stage:

\[
V_k = 4 \cdot V_{k-1} + 12 \cdot 2^{2(\Psi \cdot (k-1)+\Delta)} \cdot \sigma^2 + 6 \cdot 2^{2(\Psi \cdot (k-1)+\Delta+2)} \cdot \sigma^2 + K \cdot 4^k \cdot \sigma^2 \tag{6.12}
\]

Equation (6.12) represents a recursive expression. We obtain the total variance at stage \( M \) by explicitly solving this expression by setting \( V_0 := \delta^2 \):

\[
V_M = \delta^2 \cdot 4^M - \frac{(K \cdot 4^\Psi - 4 \cdot K + 148 \cdot 4^\Delta)}{4^\Psi - 4} \cdot \sigma^2 \cdot 4^M + K \cdot \sigma^2 \cdot (M+1) \cdot 4^M + 148 \cdot \frac{\sigma^2 \cdot 4^\Delta \cdot (4^M)^\Psi}{4^\Psi - 4}. \tag{6.13}
\]
Looking at the last term of Eq. (6.13), we see that output variance has an exponential dependency in $\Psi$. Assuming worst-case conditions, a total of three overflows occur for each radix-4 butterfly ($\Psi = 3$). Thus, output variance has a cubic dependency concerning noise introduced by finite wordlength effects. Focusing on transform length as a second important parameter, Fig. 6.3 illustrates the proportional dependency of output variance and transform length $N$. To verify this result of Eq. (6.13), we derive a simple estimation of the output variance by Parseval's theorem (Eq. 6.2):

$$|X(i)|^2 = N \cdot |x(i)|^2.$$ 

Assuming uniformly distributed sequences $\{x(i), X(i)\}$ with zero mean, $|X(i)|^2$ and $|x(i)|^2$ approximate the empirical variance $\bar{s}^2$ of $x(i)$ and $X(i)$, respectively, thus yielding

$$V\{X(i)\} \approx N \cdot V\{x(i)\},$$  

which confirms the proportional dependency in $N$ depicted in Fig. 6.3.

Figure 6.5(a) depicts the output variance $V_M$ for different transform and wordlengths.

**Typical-Case Consideration**

Contrary to worst-case considerations, we verified an average of only two overflows per stage by simulations. These two overflows will have max-
imum impact on error variance when occurring during the two radix-2 sub-stages of the radix-4 butterfly. Thus, for a typical-case analysis, we assume overflow-free complex twiddle multiplications. Equations (6.5) and (6.6) thus remain valid for typical-case analysis. However, we have to omit the scaling noise part of Eq. (6.11) to obtain $V_k$:

$$V_k = (V_k'' + K \cdot 4^k \cdot \sigma^2) + \left(2^{2\cdot\psi\cdot(k-1)+2+\Delta}\right) \cdot \sigma^2. \quad (6.15)$$

Further substitution and simplification yields the recursive definition for the complete variance at the output of the $k$-th butterfly:

$$V_k = 4 \cdot V_{k-1} + 12 \cdot 2^{2\cdot\psi\cdot(k-1)+1+\Delta} \cdot \sigma^2 + 6 \cdot 2^{2\cdot\psi\cdot(k-1)+1+\Delta} \cdot \sigma^2 + K \cdot 4^k \cdot \sigma^2 + 2^{2\cdot\psi\cdot(k-1)+\Delta+2} \cdot \sigma^2. \quad (6.16)$$

Again, we get total output variance for a $N$-point FFT by explicitly solving Eq. (6.16) and by substituting $M = \log(N)/\log(4)$ and $\sigma^2 = 2^{-2b}/12$:

$$V_N = \delta^2 \cdot N - \frac{2^{-2b}}{12} \left(\frac{K \cdot 4^\psi - 4 \cdot K + 52 \cdot 4^\Delta}{4^\psi - 4}\right) \cdot N. \quad (6.17)$$

$$+ \frac{2^{-2b}}{12} \cdot K \cdot \left(\frac{\ln(N)}{\ln(4)} + 1\right) \cdot N + \frac{13}{3} \cdot \frac{2^{-2b} \cdot 4^\Delta}{4^\psi - 4} \cdot N^\psi.$$

**Error variance and RMS**

The error related to fixed-point computation is given as the difference between the computed results and corresponding exact results $X_M(i)$ at the stage $M$:

$$E(i) = X_M(i) - \hat{X}_M(i). \quad (6.18)$$

We obtain the error variance $\sigma_E^2$ of FFT results from Eq. (6.17) by setting the initial quantization errors ($\delta^2$) to zero, thus only considering the contribution of scaling and rounding noise. On the other hand, the empirical error variance $\sigma_E^2$ and the RMS error are defined by

$$\sigma_E^2 = \frac{1}{N-1} \sum_{i=0}^{N-1} \left(E(i) - \overline{E(i)}\right)^2, \quad \text{rms}_E = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} |E(i)|^2}. \quad (6.19)$$
Assuming a uniform error distribution, \( \bar{E}(i) \) is nullified and hence

\[
\sigma_E^2 = \frac{1}{N-1} \cdot \sum_{i=0}^{N-1} E(i)^2 \approx \frac{1}{N} \cdot \sum_{i=0}^{N-1} |E(i)|^2 = \text{rms}_E^2. \tag{6.20}
\]

With this, we compute \( \text{rms}_E \) as

\[
\text{rms}_E \approx \sqrt{\sigma_E^2} = \sqrt{\frac{V}{N}}|\delta_0|. \tag{6.21}
\]

The root-mean-square of the FFT results (\( \text{rms}_R \)) can be approximated by using the average modulus squared defined in Eq. (6.10) and also by assuming a uniform distribution of the \( X_M(i) \):

\[
\text{rms}_R = \sqrt{\frac{1}{N} \cdot \sum_{i=0}^{N-1} |X_M(i)|^2} \approx \sqrt{4^M \cdot K}. \tag{6.22}
\]

To be consistent with the computation of \( \text{rms}_E \), only the real part's contribution to \( \text{rms}_R \) must be considered when computing the RMS ratio \( \text{rms}_E/\text{rms}_R \). We thus finally compute the RMS ratio as

\[
\frac{\text{rms}_E}{\text{rms}_R} = \frac{\sqrt{V_N}}{|\delta_0|} \frac{\sqrt{N \cdot K/2}}{\sqrt{N \cdot K}}
= \sqrt{\frac{(4^k-b-4^{1-b})K N \ln(N)+104 \ln(2) (4^{-b} N ^{b} -4^{1-b} N)}{6(4^k-4) \ln(2) \cdot N \cdot K}}. \tag{6.23}
\]

The RMS ratio derived in Eq. (6.23) is shown in Fig. 6.6(a) for both worst- and typical-case analysis.

### 6.3.4 Investigations on Required Fixed-Point Wordlength

To verify the analytical results derived in section 6.3.3, we implemented a fixed-point arithmetic model in C. This model describes all arithmetic components on a very low level, thus reflecting actual hardware structure.

To determine the impact of finite wordlength effects on the precision of our butterfly architecture, we measured corresponding error variance throughout a series of 1000 FFT simulations based on random input data (white noise). Measured error variance is depicted in Fig. 6.4 as a function of accumulator and coefficient wordlength for both a 256-point and 1024-point radix-4 FFT.
6.3. Fixed-Point Arithmetic

Figure 6.4: Simulated error variance parameterized in coefficient and accumulator wordlength.
Looking at the graphs depicted in Fig. 6.4, we see that error variance is decreasing when coefficients are represented with increased accuracy. However, this decrease becomes extremely flat as soon as more bits are employed for accumulator than for coefficient representation. It is then clear that an asymmetric proportion of accumulator and coefficient wordlengths is dominated by the more inaccurate representation.

Nevertheless, most fixed-point FFT processors are characterized by different accumulator and coefficient wordlengths (e.g., the FFT designs proposed in [Cetin 97] and [Baas 98]). Looking at Fig. 6.4 it is clear that these asymmetric wordlength implementations (e.g., 24b accumulator, 16b coefficients) are only justified in terms of the simplest overflow handling and do not contribute to any increase in accuracy.

In addition, with regard to VLSI implementation, using different wordlengths for representing coefficients and temporary results implies dedicated data-distribution and dedicated arithmetic resources. In particular, an asymmetric implementation narrows down the generality of the complex multiplier which cannot be re-used for complex element-wise multiplication. As a result, equal wordlengths have to be used for both accumulator and coefficients. The following discussions are thus based on the assumption of equally-sized accumulator and coefficient wordlengths.

Characteristics of error variance as a function of word and transform length are shown in Fig. 6.5. To allow for a comparison between our analytical model and C-simulation, corresponding results are given for both. While the analytical results given by Equations (6.17) and (6.13) are represented as solid graphs, the results obtained by simulations are depicted by distinct points. Considering different overflow characteristics, analytical results are given for both worst- and typical-case scenarios. The good correspondence of analytically predicted and measured results depicted in Fig. 6.5(b) proves the typical-case error model derived in section 6.3.3.

The RMS ratio derived in Eq. (6.23) is given in Fig. 6.6(b), together with corresponding simulation results. As for the error variance, predicted and measured data correspond very well. Figure 6.6(a) compares the RMS ratio for worst- and typical-case considerations.

Relative Error Induced By Quantization Effects

Both error measures, error variance and RMS ratio, are based on statistics. They are thus always referring to a set of multiple values. Consequently, there is no guarantee that each particular element of the
Figure 6.5: Error variance for 1D-FFT. Simulated and analytical results ($\delta = 0, \Delta = 1.5, K = 0.666$).
RMS ratio (Typical Case vs. Worst Case)

(a) Worst- and typical-case analysis for 1024-point FFT.

(b) Simulated vs. analytically computed results (typical-case).

Figure 6.6: RMS ratio.
underlying set will meet the predicted measure. As a result, in most situations, error variance and RMS ratios are only giving the trend when looking for the number of bits required to meet a given target precision.

Depending on the application, a relative error measure will be more significant for fixing the required wordlength. Hence, we define a relative error $E_R$ by comparing $b$-bit fixed-point results $x(i)$ to the corresponding double-precision floating-point results $\hat{x}(i)$:

$$E_R = \frac{|x(i) - \hat{x}(i)|}{|\hat{x}(i)|}.$$  

Referring to the C simulations, the maximum and the average of the relative error defined by Eq. (6.21) are given in Fig. 6.7. A maximum $E_R$ might be required for applications where a particular upper error-boundary must be met.

For the 1024-point FFT, Fig. 6.8 indicates $E_R$ as a function of fixed-point wordlength $b$.

According to the similarity in graphs for relative error (Fig. 6.7(a)) and error variance (Fig. 6.4), we can also assume exponential dependency between $E_R$ and corresponding fixed-point wordlength $b$. We can thus extrapolate our simulation results using exponential fitting techniques.

### Required Fixed-Point Wordlength

As a result, we can estimate required wordlength to achieve a specific precision. For instance, to achieve an overall precision of 0.4% for all correlation results (cf. section 6.1), underlying UCCF accuracy must be twice as high. As verified by simulations, achieving a relative error $E_R$ smaller than 0.2% for a single UCCF means that the corresponding 1D-FFTs must show an average precision of approximately 0.001%. To do this, we see from Fig. 6.8 that we must provide a fixed-point wordlength of 47$b$ for a single mantissa or 94$b$ for a complex word.

### 6.3.5 Approaches to Reducing Required Fixed-Point Wordlength

Referring to the required fixed-point wordlength of 94$b$ for a complex-mantissa, corresponding arithmetic resources and buses will impose sig-
Figure 6.7: 1024-point 1D-FFT. Relative quantization error $E_R$ in %.
nificant area overhead. Hence, methods to reduce required wordlength must be found.

**DC Masking**

Looking at the FFT spectrum corresponding to an image with nonzero mean, we find that its maximum component is almost always associated with its DC component.

Accordingly, one promising approach to reducing the dynamic range of the 2D spectrum is to eliminate the DC component of both search area and template. To do so, we have to subtract the DC component from all pixels prior to FFT transformation. As a result, due to the reduction in dynamic-range, the required target precision can be achieved by smaller wordlengths.

On the other hand, removing the DC component is a global operation that simultaneously affects all pixels of the underlying image. This fact might be a problem in cases where search area and template are characterized by different DC values. In this case, DC-free gray-level representation of the object to be located differs for search area and template. In the worst-case, this difference in representation will cause the BST algorithm to compute inaccurate object positions. Hence, the impact of DC masking on the object detection process varies with underlying image contents and can not be precisely predicted.
Nonlinear Number-Representation

Concerning two’s complement representation, the most negative representable number is larger in magnitude than the most positive representable number. Thus, when covering the full numeric range of \([-1 \ldots +1]\) of FFT coefficients, almost 50\% of the provided negative value space remains unused.

An approach to efficiently utilizing the representable value range is to limit the maximum representable positive number to the largest quantity smaller than one. We thus basically substitute each '1' by '0.999...', hence introducing a nonlinear number representation. Since we only affect a total of four coefficients, this measure will only result in slightly increased inaccuracy. However, required wordlength can only be reduced by one bit by this approach.

Alternatively, as realized for representation of DCT coefficients in image compression algorithms (e.g., H.261 ([H261 90]), etc.), we can apply a flexible quantization envelope on 1D-FFT results. Large values are thereby compressed and thus represented with lower resolution. However, this approach requires appropriate 'decompression' during the succeeding column FFT and will induce further distortion into the spectrum in addition to enlarging and making VLSI implementation more complicated.

Residue Number System (RNS) Arithmetic

In cases where wordlength becomes too large, it can be broken down by use of RNS arithmetic, thereby allowing for smaller arithmetic units [Blahut 84]. Based on the Chinese remainder theorem (CRT), a non-negative integer \(x\) is then represented by giving only its moduli \(x_i\) with respect to each of several integers: \(x \approx (x_1, x_2, \ldots x_M)\). Then, an arithmetic operation \(z = x \circ y\) (with "\(\circ\" denoting addition, subtraction or multiplication) can be independently performed on this multiple-moduli residue representation as \(z_i = (x_i \circ y_i)_m\). Due to this modular structure, RNS arithmetic naturally leads to modularity and parallelism in digital hardware. Its further characteristic of performing carry-free additions, subtractions and multiplications is commonly exploited when high precision over large wordlength is required.

Since FFT processing is dominated by a large number of additions, subtractions and multiplications, RNS arithmetic could be an appropriate way to break down wordlength. However, while employing RNS
6.3. Fixed-Point Arithmetic

Fixed-point arithmetic allows for parallel high-speed computation, it also imposes the problem of converting to and from residue number representation\(^1\), thereby imposing additional area overhead. Monolithic VLSI implementation therefore becomes more difficult.

In addition, integer-based computation of the Fourier transform is required to apply RNS arithmetic. To do this, we must either decide on dedicated scaling of all data (as discussed in [Alia 84]) or, alternatively, we must compute the correlation by means of finite field arithmetic.

6.3.6 Finite-Field Integer Arithmetic

In this section, we discuss integer-based computation of the Fourier transform as, for example, required for employment of RNS arithmetic.

In finite-field arithmetic, data can only take on integer values. Therefore, finite-field computations are exact; there are no errors due to rounding. A field is, loosely speaking, an abstract algebraic structure where addition, subtraction and multiplication are defined on a set of integers \(x \in \{0, \ldots, p - 1\}\). Furthermore, all operations in the field are performed modulo \(p\). A detailed view of finite fields is given in [Rabiner 75, Blahut 84, Mitra 93].

For the special case where \(p\) is prime, a Galois field \(GF(p)\) is obtained in which for all elements a multiplicative inverse element exists. As in our “usual” arithmetic, several number theoretic transforms (NTTs) which possess a convolution property have evolved for Galois fields. In contrast to the DFT, NTTs employ integers instead of \(W_N\), making complex arithmetic no longer necessary. Impressive computational savings are thus achieved in a limited number of cases without any loss of accuracy, making NTTs useful for special purpose hardware implementations [McClellan 79].

Correlation Algorithms for Finite-Fields

In a Galois field, the correlation can be performed by using any appropriate method in that field. Hence, existing algorithms for short

\(^1\)Targeting the moduli sets \(\{2^n, 2^n - 1, 2^n - 1 - 1, 2^n + 1\}\), efficient conversion algorithms are proposed in [Hiasat 98, Gallaher 97, Vinnakota 94]. Concerning hardware implementation, one option for performing the required modulo division is to implement a look-up table for the corresponding conversion scheme in a ROM. Alternatively, a high-speed ROM-less residue to binary converter for the three moduli residue number system \(\{2^n - 1, 2^n, 2^n + 1\}\) is presented in [Piestrac 95]. In addition, concerning the modulo-set \(\{2n + 2, 2n + 1, 2n\}\), a hardware implementation for residue number to binary conversion is introduced in [Premkumar 95].
convolutions (the Cook-Toom algorithm, Winograd short convolution algorithms, etc.) can be employed to design a direct convolution technique. However, the irregular structure of most short-length modules will lead to more voluminous address generation units and increased demand in buffer memory.

Using the Fourier transform in the Galois field and the convolution theorem is an alternative option. The 2D Fourier transform

$$V(u, k) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} v(i, j) \cdot \omega^{ik} \cdot \mu^{ju}, \quad k = 0, \ldots, n - 1 \quad u = 0, \ldots, m - 1$$ (6.25)

exists with all elementary properties in $GF(p)$ whenever $n$ and $m$ are divisors of $p - 1$, and $\omega$ and $\mu$ are elements of order $n$ and $m$, respectively.

To choose $p$ appropriately, we have to consider the corresponding set of existing transform lengths as well as the resulting value range. No correlation coefficient may be larger than $p - 1$. Recalling $8b$ resolution $128 \times 128$ templates, the smallest usable prime $p$ is 1,065,369,611. Also, depending on the chosen $p$, the resulting modulo-arithmetic is more or less difficult to realize. Modulo-division for diminished-radix-2 $(2^n - 1)$ and augmented-radix-2 $(2^n + 1)$ can be performed by carry-add and carry-subtract techniques, respectively. More details on hardware realizations performing modulo-operations are given in [Curiger 93]. For simple computations of modulo-operations, Mersenne primes are preferred.

**Mersenne Number Transforms**

Mersenne number transforms in $GF(p)$ are achieved for $p = 2^m - 1$, where $p$ is a prime. The first seven Mersenne primes are 7, 31, 127, 8191, 131,071, 524,287 and 2,147,483,647. Mersenne number transforms are particularly characterized by their simple modulo-arithmetic. Resembling 1's-complement arithmetic, overflow bits are simply added into the low order bits. On the other hand, since usable transform block-lengths $n$ are those dividing $2^m - 2$, application of the Cooley-Tukey FFT algorithm is ruled out because $(2^{2n} - 1) - 1$ is not a power of two.

However, Mersenne number transforms can be computed by any mixed-radix FFT or Winograd small FFT algorithm by which a FFT of desired blocklength can be constructed. Targeting a transformation of $1024 \times 1024$ images, we could choose the transform length $N = 1023 = 3 \cdot 11 \cdot 31$. To do so, one row and one column of the image remain
6.3. Fixed-Point Arithmetic

unused. However, the chosen transform length allows us to apply the
prime factor FFT for efficient computation:

\[
X[k_1, k_2, k_3] = \frac{1}{N} \sum_{n_1=0}^{2} \sum_{n_2=0}^{10} \sum_{n_3=0}^{30} x[n_1, n_2, n_3] W_3^{n_1 k_1} W_{11}^{n_2 k_2} W_{31}^{n_3 k_3}. \tag{6.26}
\]

According to the required value range, we would have to perform
Eq. (6.26) in \(GF(2^{31} - 1)\), where we set the coefficients to \(W_3 = 634,005,911, W_{11} = 100,973,744\) and \(W_{31} = 2\).

**Fermat Number Transforms**

Fourier transforms are simplest in \(GF(2^m + 1)\) [Agarwal 74], which is
a field whenever the Fermat number \(p = 2^m + 1\) is a prime. For such
primes, \(p - 1\) and any other factor of \(p - 1\) is a power of two, hence
allowing the application of the Cooley-Tukey FFT.

However, by verification of suitable Fermat-numbers \(2^{2^m}\) up to
\(m = 7\), we have proven 65,537 to be the largest usable prime. Ac¬
cording to this limited value range, Fermat number transforms seem to
be unsuitable with regard to the target application where correlation
coefficients are much larger than 65,536.

A promising solution to this problem is given by the use of RNS
arithmetic (see section 6.3.5), which can be used to break the required
wordlength into a number of short wordlengths [Blahut 84]. For this,
we recommend \(\{127, 129, 256, 257\}\) as an appropriate set of relatively
prime integers and break down the correlation:

\[
r_{x,y}^{(l)} = \sum_{i=0}^{M-1} \sum_{k=0}^{M-1} s_{x+i,y+k}^{(l)} l_{i,k}^{(l)} \pmod{m_l}, \quad x, y = 0, \ldots, N - M \tag{6.27}
\]

where \(r_{x,y}^{(l)}\) denotes the residue \(r_{x,y} \pmod{m_l}\). The major disadvantage
of this approach is the need to perform two successive modulo-divisions
after each arithmetic operation: the first is required due to computa¬
tion in \(GF(p)\), the second corresponds to the actual RNS arithmetic.
In terms of hardware implementation, area overhead significantly in¬
creases. Because of this, on the other hand, the required correlation
can be computed in the field \(GF(2^{16} + 1)\) where Fourier transforms of
size \(2^{16}, 2^{15}, \ldots, 2^2\) and 2 exist. A very efficient transform scheme for
a 1024-point FFT is proposed in [Blahut 84]. It consists of two nested
“multiply-free” 32-point Cooley-Tukey FFTs which can be computed only by shifts and adds. Details on the arithmetic for Fermat number transforms are given in [Leibowitz 76]. As an alternative to RNS arithmetic, an integer ring transform can be used to broaden the number of usable bits for a given $p$.

**Integer Ring Transforms**

If $p$ is a composite integer, the resulting algebraic body is a finite ring in which certain elements do not have multiplicative inverses. Nevertheless, it is often possible to define transforms in a ring $\mathbb{Z}/(p)$, but the situation here is not as straightforward. For $p = \hat{p}^m$, where $\hat{p}$ is an odd prime, the blocklengths $n$ that can be chosen in $\mathbb{Z}/(\hat{p}^m)$ are the same blocklengths that can be chosen in $GF(\hat{p})$. The major gain is the increase in wordlength from about $\log_2 \hat{p}$ bits to about $m \log_2 \hat{p}$ bits. Apart from that, very little value is added by working with a composite $p$ [Blahut 84].

**Conclusion on Finite Field Arithmetic**

In summary, FFT transformation can be accomplished without using complex numbers by employing finite-field arithmetic. This simplification results in decreased memory requirements as well as in decreased overall operation count. In particular, avoiding complex arithmetic allows for more simple and compact arithmetic units and allows for the design of efficient ‘shift-add’ FFTs. Furthermore, according to the underlying integer computation, RNS arithmetic can be employed to break down overall wordlength, resulting in even smaller arithmetic units.

On the other hand, the decrease in memory requirements and operation count is almost the same as that achieved by means of the real-valued FFT. In addition, by performing the 2D correlation by means of finite-field arithmetic, a corresponding system implementation cannot be employed for general Fourier transformation as required by radar- and speech-processing applications, for example (cf. section 10.4). The system’s application-field would thus be significantly narrowed down. In addition, performing modulo-operations requires dedicated arithmetic units which are significantly larger than their native arithmetic counterparts. We also found the ‘add-shift’ FFTs to be less convenient than suggested in the literature. This is mainly because not all multiplications with powers of two can be efficiently realized by means of shifting in $GF(p)$. For some, the corresponding shift would simply be too large.
to be efficiently handled by existing modulo-units. Facing these drawbacks, we acknowledge 'traditional arithmetic' as the more appropriate option for the targeted object detection system.

6.4 Block Floating-Point Arithmetic

Coming back to 'traditional arithmetic', block floating-point (BFP) arithmetic realizes a compromise between fixed- and floating-point representation. The basic concept of BFP-arithmetic is to represent the place-values of mantissas corresponding to a block of associated real numbers by one common exponent (=block exponent, BE).

The block exponent is obtained by examining all the numbers in the set and representing the largest number as an ordinary floating-point number with a normalized mantissa. Appropriately dividing the entire data set into corresponding blocks thus results in a significant enlarged numerical dynamic comparison to fixed-point computation. In addition, the typical circuit-overhead imposed by floating-point arithmetic can be avoided.

According to the BFP principle, each block exponent must be appropriately chosen to allow the representation of the largest magnitude contained in the underlying data block with the most accuracy. Since the remaining elements of the data block are scaled against the largest number, the differences in magnitude must not be too large. Otherwise, a considerable amount of information is discarded due to the scaling of mantissas corresponding to lower magnitudes, which is equivalent to a loss in accuracy. In this case, BFP arithmetic converts to native fixed-point computation.

6.4.1 Employing BFP Arithmetic for 2D Cross-Correlation

According to [Rabiner 75], BFP number-representation is most suitable for FFT implementations. To achieve maximum precision at moderate costs, our approach is thus to implement BFP arithmetic on the system-level. Down-sizing of required system memory is thereby achieved along with the reduction of imposed bus-bandwidth.

As discussed in section 3.2.2, 1D data rows are established as the system's fundamental data elements. Consequently, for BFP-based cross-correlation, row-specific block exponents are preferably chosen on the system-level [Cavadini 99].
6.4.2 Impact on APU Arithmetic

According to the fact that one BE is assigned to each row of data, corresponding 1D-processing can basically be accomplished by means of fixed-point arithmetic. However, aiming at higher precision, implementing a further, dedicated BFP arithmetic into the processor’s butterfly unit seems to be an appropriate measure. To do this, we had to support different BE scopes on the system- and processor-levels in order to realize a cascaded BFP arithmetic implementation.

Referring to the concrete implementation of efficient BFP arithmetic into the butterfly architecture developed in Chapter 5, the major objective is to achieve maximum precision at a minimum of imposed area overhead. To achieve this, and to further minimize controlling effort, we must carefully adjust the scope of block exponents employed during FFT processing. Hence, in the following discussion we focus on the properties corresponding to relevant BE scopes.

6.4.3 Confronting Relevant BE Scopes for 1D-FFT Computation

To find out if a meaningful BE scope exits for 1D-FFT computation, we rate potential “block-sizes” according to corresponding overall precision and implementation overhead.

According to the properties of radix-4-based 1D-FFT computation, relevant BE scopes are ‘mantissa’, ‘complex-mantissa’, ‘quartet’ and ‘stage/row’. The basic impact of “block-size” on the covered numerical range is qualitatively depicted in Fig. 6.9: by continuously enlarging BE scope from ‘single mantissa’ to complete ‘data row’, characteristics of the corresponding numerical dynamic varies from native floating-point computation to native fixed-point arithmetic.

In the following, the set of relevant BE scopes for 1D-FFT computation is comprehensively discussed.
Mantissa

Assigning a dedicated exponent to a single mantissa entirely corresponds to native floating-point arithmetic. Concerning VLSI implementation, the major disadvantage of this arithmetic-type is given by its area-consumptive adders. Also, recalling that row-specific block exponents are established on the system-level, input data to the butterfly unit is consequently characterized by little variation in the numerical dynamic. Compared to fixed-point arithmetic, floating-point computation of the first stage will thus not result in any additional accuracy, but will impose significant area overhead.

Although more accuracy is achieved by completing 1D-FFT computation by means of floating-point arithmetic, the use of sequential floating-point additions leads to an overall alignment of all exponents contained in the underlying data row. This effect comes from the principle of floating-point addition, since the involved mantissas are aligned to each other before the actual arithmetic operation is performed. Whenever large dynamic input data is supplied to the butterfly unit, the alignment units of the ‘entry-stage’ floating-point adders will thus perform a pairwise mantissa normalization. Proceeding with data-flow computation, complete normalization of each input quartet is done by the ‘final-stage’ alignment units, thereby smoothing the increased numerical dynamic probably induced by floating-point twiddle multiplications.

Complex Mantissa

This approach is basically equivalent to the floating-point concept, except that one exponent is assigned to both real and imaginary parts of each complex data. In the case that all real and imaginary parts are covering piece of the same numerical range, this approach will not lead to a decrease in achieved precision but will result in slightly reduced memory requirements. Furthermore, by integrating corresponding real and imaginary part additions into one single unit, one exponent-compare unit can be saved, resulting in additional area reduction.

Apart from that, the main statements given in the “mantissa-scope” section can be adopted for the “complex-mantissa” scope because of the relationship to native floating-point arithmetic.

Data-Quartet

The basic idea of this approach is to assign one exponent to each quartet of complex data that is grouped for computing a single radix-4 butterfly.
Mantissas are thereby inherently aligned within each quartet and actual butterfly computation can consequently be performed by means of fixed-point arithmetic, resulting in significant simplification and area savings.

However, during butterfly computation we must take care of overflows. If an overflow occurs, the remaining mantissas must also be scaled to maintain consistency regarding the underlying quartet BE. To avoid a mix-up between quartets showing different block exponents, we have to tag all quartets. In addition, in case of an overflow, we must selectively scale only that data which is associated with the overflow-generating addition. As a result, considerable controlling is required.

Furthermore, the data set associated with each quartet varies from stage to stage. Hence, with each proceeding stage, we have to align the mantissas for each of the new arranged quartets. As a result, the variation between all quartet-specific BEs is increasingly reduced.

**Row/Stage**

Enlarging the BE scope to a complete data row corresponds to fixed-point computation. As already discussed for the "quartet" scope, row-level BE computation requires special care regarding overflows.

**Conclusion on Potential BE Scopes**

Looking at the variety of possible BE scopes discussed so far, we find the slightly increased precision achieved by a fine-grained scope not proportionally correlated to the effort required to support the corresponding BFP arithmetic. In particular, although floating-point arithmetic is assumed to provide best numerical dynamic, after successively processing all stages, all BEs are more or less adjusted to each other. This adjustment effect is also observed for 'complex-mantissa' and 'quartet' BE scopes and almost corresponds to a complete normalization on the row-level.

According to this alignment effect, we will not sacrifice significant accuracy when choosing a 'stage-level' BE. However, with this, 1D-FFT computation can basically be performed by means of fixed-point arithmetic. As a result, VLSI implementation of the CADDs is significantly simplified.

**6.4.4 Fixing Required Mantissa Wordlength**

To model the impact of finite wordlength effects on the precision of correlation results, we implemented a C model of BFP arithmetic allowing
6.4. Block Floating-Point Arithmetic

<table>
<thead>
<tr>
<th>Word Length</th>
<th>Error Variance</th>
<th>$\frac{rms_{E}}{rms_{R}}$</th>
<th>Relative Error Max.</th>
<th>Relative Error Av.</th>
<th>Peak-Pos. $x, y$ (Mag.$\times 10^7$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20b</td>
<td>$1.21 \cdot 10^{14}$</td>
<td>1</td>
<td>100%</td>
<td>100%</td>
<td>-,-</td>
</tr>
<tr>
<td>24b</td>
<td>$2.81 \cdot 10^{13}$</td>
<td>0.126203</td>
<td>49.8%</td>
<td>9.9%</td>
<td>526,151 (8.4)</td>
</tr>
<tr>
<td>28b</td>
<td>$1.29 \cdot 10^{11}$</td>
<td>0.008329</td>
<td>3.76%</td>
<td>0.66%</td>
<td>520,147 (8.5)</td>
</tr>
<tr>
<td>32b</td>
<td>$5.13 \cdot 10^{8}$</td>
<td>0.000524</td>
<td>0.26%</td>
<td>0.041%</td>
<td>916,658 (8.6)</td>
</tr>
</tbody>
</table>

Table 6.1: Error measures for $1024 \times 1024$ cross-correlation (UCCF) computed by BFP arithmetic.

for simple variation of relevant parameters.

As already stated in section 6.3.4, we have to meet a maximum relative error of 0.2% for each UCCF computation to guarantee excellent detection quality over a broad application field. Thus, to determine the required mantissa wordlength, a number of representative search areas and templates have been transformed by means of the C model. To obtain meaningful results, we put special focus on irregular background and free-form templates.

Corresponding error measurements for a $1024 \times 1024$ UCCF are given in Table 6.1. Looking at the RMS ratio, it is clear that object detection is impossible for wordlengths smaller than 20b since noise and signal are of the same order of magnitude. As proved by simulations, this holds true even for a search area of size $256 \times 256$.

A graphical representation for $E_R$ is shown in Fig. 6.10. To achieve error bounds acceptable for object detection, a wordlength of at least 28b is required corresponding to a maximum relative error of 3.76%. Assuming the same inaccuracy for the normalizing energy term, and further assuming that the division of UCCF and corresponding normalizing term is carried out by floating-point arithmetic, the maximum relative error doubles to 7.52% for the NCCF. Depending on the application, this accuracy could be sufficient.

However, targeting high-precision object detection (section 6.1), a mantissa wordlength of at least 32b is absolutely necessary to guarantee a maximum relative error of approximately 0.2% (cf. Table 6.1).
Figure 6.10: $1024 \times 1024$ UCCF: Extrapolation of average relative error $E_R$.

### 6.4.5 Achieved Area Reduction by Means of BFP Arithmetic

Recalling our investigations in native fixed-point arithmetic, a fixed-point wordlength of $47\text{b}$ is required to achieve the same accuracy (cf. section 6.3.4). As a result, compared to a $47\text{b}$ native fixed-point arithmetic, implementation of $32\text{b}$ BFP arithmetic allows for an area reduction of a very considerable $71\%$.

---

**Referring to the processor model discussed in section 9.2, $95\text{mm}^2$ are required to implement $32\text{b}$ BFP arithmetic by means of full-custom layout techniques, whereas approximately $337\text{mm}^2$ are required for an equivalent $47\text{b}$ fixed-point implementation.**
7. Block Floating-Point Implementation

In this chapter, we focus on the implementation of BFP-arithmetic into the butterfly architecture proposed in section 5.3.5. In particular, starting with regular row-level BE-computation, we develop a dedicated hybrid block exponent (HBE) arithmetic concept combining multiple BE scopes to achieve maximal precision at moderate hardware costs. As a result, compared to native fixed-point implementation, precision is increased by one order of magnitude while area requirements are reduced by 71%.

7.1 Fixed-Point Sub-Butterfly Computation

To realize row-level BE arithmetic, all input data has to be aligned according to the underlying block exponent. As mentioned earlier, we must pay special attention to overflows throughout the following fixed-point computation.

7.1.1 Concepts for Overflow-Handling

Due to the two radix-2 sub-stages embedded into the butterfly unit, up to two overflows might occur during the two succeeding CADD operations. To ensure correct processing after such an overflow event, the concerned data must be corrected, i.e., shifted to the right. However, by doing this we modify the 'individual' exponent of the scaled mantissa, thereby provoking an inconsistency to the underlying principle of row-level BE representation. As a consequence, the entire data set must be re-scaled.

Scaling Strategy

A set of potential scaling strategies is given in section 6.3.2. Since our intention is to achieve maximum precision, we have to ensure maximum
utilization of mantissa capacity. Consequently, the test for overflow strategy is a favorite choice. In the case of an overflow however the entire sequence must then be right-shifted by one bit (cf. section 6.3.2).

Consequently, concerning direct realization of test for overflow, a dedicated task has to be assigned to test scaling necessity and to eventually perform a required alignment. As a result, this most accurate scaling strategy is extremely time-consuming and makes actual computation time unpredictable.

Concerning simple radix-2 computation, a first step to avoiding a dedicated scaling task is to transparently scale those results that are computed after the overflow occurred 'on the fly'. We still, however, have to align that data that was processed prior to the overflow event. To allow for the distinction between already scaled data and data still requiring an adjustment, our approach is to extend each piece of data by a corresponding tag carrying scaling information. By verifying these tags in conjunction with the underlying overflow status, "deferred" scaling of yet non-corrected data can be performed when the sequence is re-clocked into the butterfly unit to compute the subsequent stage.

We can thus preserve continuous data stream processing. We combine the "speed" advantage of 'sequence controlling' (no re-processing required) and the maximum utilization of mantissa capacity.

**Corresponding Overflow-Concept for R4-Butterfly**

Looking at our radix-4 butterfly unit (Fig. 5.14), we see that the two radix-2 sub-stages are concatenately computed by means of dedicated pipeline-interleaving. As a result, we cannot perform an "isolated" scaling after the 'entry-stage' since some of its data has already run through the succeeding 'final-stage' adders and hence was probably subject to an additional overflow. Therefore, we have to employ an adapted concept to directly realize our test for overflow-principle in combination with "tagging".

First, we will focus on the 'entry-stage'. As for the native radix-2 butterfly discussed before, all results subsequently computed to an 'entry-stage' overflow have to be scaled, regardless of whether or not the underlying arithmetic operation has caused an overflow. Furthermore, we have to additionally scale the set of results being computed prior to the overflow event. According to the pipelined APU structure, some of those results are still within the butterfly unit and hence can still be transparently scaled on the fly. Concerning results that have already
left the 'adder part', deferred scaling can be performed by means of
tagging.

In a similar way, alignment of 'final-stage' results have to be per-
formed in the presence of a corresponding overflow event (right-shifting
of second sub-stage). However, transparent scaling of results that are
computed prior to the actual overflow can only be performed during the
subsequent stage's computation start-up.

Concerning the 'final-stage', as mentioned before, we could addition¬
ally perform an alignment corresponding to an 'entry-stage' overflow.
By doing so, we would exploit the butterfly unit's pipeline structure to
anticipate a limited amount of the "deferred" scalings. However, since
we have to align all mantissas prior to the processing of the subsequent
stage anyway, it should not matter whether we scale these results at this
time or during computation start-up of the succeeding stage. However,
we will achieve higher precision for the latter.

In conclusion, with the overflow concept described so far, we achieve
a "guaranteed" FFT computation time, independent of the number of
overflows that have occurred. Referring to the targeted real-time com¬
putation, this is of great benefit compared with other, time-dependent
scaling techniques, as, for example, the one employed by Trivedi
([Trivedi 81]).

**High-Precision Overflow Concept (HPOC)**

For the following discussion we will assume that an 'entry-stage' overflow
occurred during the computation of the $d$-value. In this case, according
to our original scaling strategy, we have to right-shift all results cur¬
rently computed in the 'entry-stage'. However, if there was no overflow
during computation of the $a$ and $b$ values (cf. Fig. 5.4), we can gain
additional accuracy of $0.5b$ on average if we scale the $ef$-results of the
corresponding butterfly instead of $a$ and $b$.

Thus, to achieve maximum precision, we must scale as few operands
as possible prior to the actual arithmetic operations. However, since we
are basically employing fixed-point arithmetic within the 'adder part',
we must at least align those operands that are to be combined with the
overflow-corrected result during further processing. Thus, as a minimum
alignment requirement, we have to ensure that corresponding input data
to each radix-2 sub-butterfly show equal "individual" exponents.

While this requirement is automatically achieved for the 'entry-
stage' due to transparent row-normalization at computation start-up,
we must place additional effort on establishing data consistency for the 'final-stage'. In the case of an 'entry-stage' overflow we must also scale the corresponding "partner element". On the other hand, in the case of an overflow occurring in the 'final-stage', only the concerned result has to be scaled. These results are solely subject to the subsequent twiddle multiplication and hence no exponent consistency must be maintained, as was required for fixed-point CADD operations.

As a consequence, we must implement "selective" scaling. As a result, the HPOC approach naturally leads to elements differing in their "individual" exponents. Hence, to allow for a subsequent alignment on the row-level, we have to keep track of the number of performed shifts for each individual data element.

**Individual recording of overflow-history.** Several options for recording the number of individually performed shifts exist. The most obvious solution is to use the tag concept mentioned above. Whenever a shift operation is performed on a distinct element, its corresponding tag has to be accordingly incremented by one.

Another approach related to tagging is to realize an overflow count memory. This dedicated memory type must be appropriately sized to hold the individual shift-history for the entire data row. The basic functionality is as follows: to update the number of shifts performed on a specific data element, its corresponding address must be supplied to the memory which then automatically increments the storage place associated to that address.

A further interesting alternative is given by associative memories (AMs). The basic idea of this approach is to provide multiple AMs, each associated to a distinct number of performed shift operations. During computation start-up, the addresses of all data elements contained in a complete sequence are associated to the AM representing 0-shifts (AM0). When data alignment is performed, the corresponding element-address is "moved" from its current AMn to AMn+1. After processing is completed, the actual number of individually performed shifts is obtained by supplying the concerning address to all AMs, with only one AM giving a positive answer.

**Conclusion.** In conclusion, the advantage of the HPOC scaling concept is clearly given by its high precision. However, the requirement for selective scaling imposes an increased effort on the underlying overflow controlling. In addition, each value leaves the 'adder part' with an
individual BE. Individual recording of shift-history is thus required for each piece of data.

Small-Area Overflow Concept (SAOC)

Selective scaling and tracing of individual shift history requires appropriate resources and controlling. Hence, to reduce silicon area we have to look for a more convenient overflow concept.

To avoid expensive recording of individual alignment history our approach is to initiate simultaneous scaling of all data that is currently processed in the 'adder part' whenever an overflow is generated by one of the CADDs. By this concept, we basically equalize overflows that occur in 'entry-stage' and 'final-stage', and hence achieve an essential simplification of overflow recording. Since simultaneous shifts are performed upon each overflow, tags corresponding to 'adder part' output data are given by the total number of overflows that have occurred so far. By employing pseudo code, we can give an 'algorithmic' description of the SAOC:

```
FOREACH clock-cycle DO {
  /* read new input data */
  read(LocalMemory, element); /* complex-mantissa +tag */

  /* align complex mantissa to be consistent with 'adder part' data */
  element =asr(element, (OverflowCounter -element.tag));
  buffer(element, Registerfile_1);

  /* parallel computation of two radix-2 sub-butterflies */
  process_in_parallel {
    CADD1_res =exec(CADD1, take_data_from(Registerfile_1));
    buffer (CADD1_res, Registerfile_2);
    CADD2_res =exec(CADD2, take_data_from(Registerfile_2));
    IF (overflow_occurred) call(overflow-event);
  }

  /* merge processed data with corresponding tag */
  output(CADD2_res.OverflowCounter);
}

procedure(overflow-event){
  OverflowCounter +=1;
  exec(Scale_all_adder-part_data);
}
```
As expressed by the pseudo code, tracing of overflow history reduces to counting overflow events assigned to the 'adder part'. As a result, no sophisticated management is required to cope with selective scaling or different individual exponents.

On the other hand, to maintain exponent consistency within the 'adder part', we have to scale new input data according to the number of counted overflows. As discussed above, scaling input operands instead of the corresponding results leads to slightly reduced precision.

**Conclusion on Overflow-Handling Concepts**

By now, we have presented different concepts for overflow handling, all based on a modified test for overflow scaling strategy. The concepts differ in precision and in implementation overhead. Referring to our cross-correlation application, we proved the precision-loss associated with the 'small-area overflow concept' to be acceptable. Hence, to reduce implementation costs, we implemented this concept to process overflows occurring during APU computation.

### 7.1.2 Resulting Sub-Butterfly Implementation

To implement the 'small-area overflow concept' into the butterfly unit shown in Fig. 5.14, we have to integrate ASR* capability into each of the registerfile's storage elements as well as into all pipeline registers assigned to the 'adder part'. In addition, we have to consider that dedicated overflow correction must be provided for the shifting units following a potentially overflow-generating adder.

Simultaneous data scaling is coordinated by an Overflow Controller (OC). For this, the OC is connected to each CADD module by sense-lines (cf. Fig. 7.1). Actual scaling is initiated by means of the OC's control lines. Furthermore, the OC contains a counter that records the total number of shift operations (Overflow Counter).

To ensure exponent consistency throughout FFT processing, new data clocked into the 'adder part' must be aligned to "match" the data currently contained in the registerfiles and pipeline registers. This alignment is performed by the APU Entry Shifter (cf. Fig. 7.1) and represents the "deferred" scaling mentioned above. Recalling our overflow concept, actual alignment is performed according to the difference of individual shifts performed on each data element (tag) and the actual number of occurred overflows (OC).

---

*Arithmetic Shift Right*
To recap, using fixed-point arithmetic within the 'adder part' allows for area-saving implementation of the CADDs (about 90% area reduction compared to the floating-point solution), buses and pipeline registers. By implementing the SA0C, compact overflow handling is realized, leading to a further reduction of overall size.

7.2 Pseudo Floating-Point Twiddle Multiplications

At the output of the 'adder part' the current contents of the overflow counter are attached to each piece of data "departing", representing its individual exponent.

The tagged data format obtained from the 'adder part' is shown in Fig. 7.2 and entirely corresponds to BE representation on the 'complex-mantissa' level (section 6.1.3). We can thus perform *pseudo floating-point* (PFP) twiddle multiplications to increase the numeric range of butterfly results. Compared to complex fixed-point multiplication, the effort required to implement a complex PFP multiplier does not significantly increase. This is because real and imaginary mantissa are aligned according to one exponent for each operand. Hence, similar to the transition from real fixed-point to floating-point multiplication, we merely

---

\[\text{Contrary to native floating-point arithmetic, we have only one "exponent" for both real and imaginary parts.}\]
have to extend complex fixed-point multiplication by additionally computing the sum of corresponding BEs. As a result, we gain increased numerical dynamic at reasonable cost. Although this improvement to numerical dynamic is to be smoothed at computation start-up of each sequential stage (alignment at row-level), it is preserved for the last stage.

7.2.1 Local Overflow Correction

Referring to the actual complex multiplication we use the "Burrus-technique" discussed in section 5.4.3. Therefore, we have to consider overflows occurring during computation of the complex product.

Looking at the structure of the complex multiplier (cf. Fig. 5.12), real adders, real subtractors and real multipliers are employed to compute a complex product. Since we are using 32\text{b} two's complement for number representation, no overflow might occur during a single 32\text{b}×32\text{b} multiplication. This can be easily proved by comparing the maximum magnitude of the product (|−2^{31}||−2^{31}| = 2^{62}) and the numeric range covered by a 64\text{b} two's complement representation (|−2^{63}, 2^{63} − 1|).

However, overflows might occur during the additions and subtractions located before and after the single multiplications. To maintain data consistency at each point of the complex multiplication process, simultaneous scaling is again required. Hence, we implemented two scaling zones (cf. Fig. 7.3). We assigned the three additions/subtractions required prior to the 32×32 multiplications to the first zone. The addition and subtraction following these multiplications form the second scaling zone.

7.2.2 Exponent Logic

As mentioned before, to realize complex PFP multiplication we have to additionally compute a product BE by summing up the BEs attached to each of the two input operands. The computation of the product BE is performed by the 'exponent logic' (cf. Fig. 7.3).
7.2. Pseudo Floating-Point Twiddle Multiplications

Like the 'adder part', the complex multiplier disposes of a pipelined structure to allow for high data throughput. In particular, each $32 \times 32$-multiplier contains 16 pipeline stages to sustain clock rates beyond 100MHz. According to this pipeline-structure, multiple "products" are contained in the complex multiplier, each in a different temporary state. Consequently, we have to store the resulting BE for each product until it can be attached to the definitive multiplication result.

Referring to the product BE's computation procedure denoted above, it is presumed that no overflow occurs during the corresponding complex fixed-point multiplication. However, we must correct the product BE whenever an overflow is encountered in at least one of the two scaling zones. Concerning this correction, we must solely increase the product BE corresponding to the overflow-afflicted "product", without affecting other product BEs that are also stored within the exponent logic. To do so, and also to allow for the storage of multiple product BEs, our approach is to implement a 'BE queue'. By adapting the queue length to the complex multiplier's pipeline depth, each queue element

\[\text{Figure 7.3: Overflow handling for complex multiplier.}\]

\[\text{Although the multiplier's layout is comparably compact (cf. Fig. 8.5), the underlying standard cell placement is rather sub-optimal, hence increasing parasitic interconnect delays.}\]
can be assigned to one particular pipeline stage. We can thus implement incrementers into those queue elements that correspond to each of the two scaling zones. The overflow logic then processes possible overflows and initiates correction of corresponding product BEs by means of these incrementer elements.

As a result, we guarantee consistency between corresponding product BEs and maturing "products" throughout the complex multiplication process.

### 7.3 Resulting Hybrid-BE Arithmetic

The resulting arithmetic concept employed for radix-4-based FFT computation is shown in Fig. 7.4.

The implemented combination of fixed-point 'adder part' computation and PFP twiddle multiplications results in a hybrid BE (HBE) arithmetic concept. According to this concept, $M - 1$ of $M$ stages are basically computed by means of fixed-point arithmetic, i.e., by using a row-level BE. However, after finishing computation of the last stage no further row-level alignment is required. The large numerical dynamic resulting from PFP twiddle multiplications is thus preserved for the $M$-th stage. Or, from a different point of view, the BE scope changes from 'row-level' to 'complex-mantissa' for the last stage.

Since the last stage is computed by PFP arithmetic, a 1024-point

![Figure 7.4: Resulting HBE concept for radix-4 FFT computation.](image)
FFT is basically computed with a precision of a native 256-point fixed-point FFT. Hence, compared to native fixed-point arithmetic, significant improvement of accuracy by one order of magnitude is achieved (see section 7.5).

### 7.3.1 HBE-Based Computation of Non-FFT Tasks

Although we derived the HBE concept by specifically focusing on FFT computation, elementwise multiplication and the real-valued FFT pre- and post-processing tasks (e.g., scramble, unscramble) are completely supported.

Since each of these tasks can be computed by only one “stage”, the individual tags are preserved for each data element. Since each tag corresponds to a BE on the 'complex-mantissa' level, all non-FFT tasks are consequently computed by means of “native” PFP arithmetic.

### 7.4 Embedding HBE Arithmetic into System Concept

Due to the HBE arithmetic implemented into the butterfly unit, sequences coming from APU processing are characterized by individual exponents (PFP format). However, since row-level BE arithmetic is used on the system-level, we must provide the option to convert a 'PFP sequence' into a row-level BE representation.

To do so, we implement a 'normalizer' into the processor's I/O interface. The basic principle of the underlying normalization procedure is illustrated in Fig. 7.5. Here, the individual exponents of PFP input sequences are indicated by different shadings. According to the implemented HBE arithmetic, each PFP input sequence must be converted into row-level BE representation, i.e., all sequence elements must be represented by one common exponent. Starting with a sequence characterized by individual exponents, we must obviously choose the largest exponent for subsequent scaling. In the next step, we then shift the mantissas of all sequence elements according to the difference of their corresponding individual exponent and our chosen 'scaling reference' (e.g., the largest of all individual exponents).

As depicted in Fig. 7.5, this procedure can be also performed for the processor's output data. Alternatively, in order to maintain its PFP format, the output sequence can be written back into system memory by bypassing the normalizer unit (Fig. 7.5).
Figure 7.5: Embedding of HBE Arithmetic into System Concept. Input sequences represented in PFP format (individual exponents are indicated by different shades) must be converted into row-level BE presentation prior to actual processing. This is realized by means of a 'normalizer'. Due to the HBE arithmetic implemented in the arithmetic data path unit, results coming from APU processing are also represented in PFP format. Depending on the succeeding processing task, these results can also be either left in PFP format or converted into row-level BE representation.

### 7.4.1 HBE-Based 2D-Correlation Flow

Recalling the system's target algorithm, 2D Fourier domain-based correlation is accomplished by an appropriate sequence of 1D-FFTs/iFFTs, elementwise multiplications and real-valued FFT pre-/post-processing steps with which underlying UCCFs are computed (cf. Fig. 2.9).

Referring to HBE arithmetic UCCF computation, mutation of individual exponents (e.g., 'complex mantissa' scope) during the different processing steps is depicted in Fig. 7.6. In this figure, each individual exponent is represented by an individually shaded square.

#### Row-RFFT

Starting with a completely real-valued search area (all individual exponents are aligned, i.e., identically shaded), we first compute real-valued FFTs for each pair of neighboring rows. According to our HBE arithmetic, computed spectra are represented in PFP format. This individuality in 'complex-mantissa' BEs is indicated by differently shaded squares that form the \( N \times \frac{N}{2} + 1 \) temporary matrix in Fig. 7.6(a).
(a) **Row-RFFTs**: Starting with the completely real-valued search area (characterized by identical exponents), real-valued FFTs are performed on each pair of neighboring rows. The corresponding FFT results are represented in PFP format, as indicated by the differently "shaded" exponents.

(b) **Complex column FFTs - Elementwise Multiplications - Column iFFTs**: Prior to actual processing, each 'PFP-column' is converted into row-level BE representation. After the 'normalized' column is processed, it is written back to system memory maintaining its PFP format.

(c) **Real-valued row-iFFTs**: Each pair of two neighboring rows is normalized and processed in sequence (iRFFT). By maintaining the PFP format of the iRFFT results, the final UCCF matrix is also represented in PFP format. Assuming that UCCF normalization (e.g., computation of the NCCF) is performed by means of floating-point arithmetic, significant increase in accuracy is thus achieved compared to native fixed-point computation.

Figure 7.6: HBE arithmetic-based UCCF computation (cf. Fig. 2.9). The sub-figures a), b) and c) depict the mutation of individual exponents, i.e., exponents assigned to single complex mantissas. Different exponents are indicated by differently shaded squares.
Column FFT, Elementwise Multiplications, Column iFFT

The temporary matrix is transposed to allow for succeeding column transformation. This is achieved on the system-level by addressing image memory via the VBS (cf. section 3.3). After transposition, all columns are reloaded into the PEs for further processing.

Strategies for row-level alignment prior to column FFTs. Recalling the implemented HBE concept, each data sequence is to be normalized when clocked into the butterfly unit. We have two choices for the scaling-reference required for this row-alignment:

1. The first option is to scale all columns according to the maximum block exponent $B_T$ contained in the temporary matrix. The normalization thus realized is equivalent to global matrix normalization. To compute $B_T$, the maximum of all BEs has to be determined; this can be done efficiently during the previous write-out of all row-elements.

2. The second option is to normalize each column according to its own maximum block exponent $B_C$. In this case not the maximum of all BEs must be determined but the $B_C$s of all rows instead. This makes the situation here more sophisticated compared to global normalization, while on the other hand, no crucial advantage seem to be obtained at first glance.

Nevertheless, increased accuracy is achieved by the second normalization strategy. Since the largest magnitudes are usually assigned to the DC coefficients of the row-spectra, we can expect the largest BEs to be associated with elements of the leftmost column (column #0) with sufficient probability. However, a large BE does not necessarily imply largely utilized mantissas and hence we might further suppose situations where no additional overflow will occur during further processing of column #0. On the other hand, by individual normalization of each column, corresponding mantissas will be represented with better (or at least equal) accuracy compared to global matrix normalization. Overflows are thus to be expected for the majority of these columns. Finally, after all, the difference between the new individual $B_C$ and the maximum BE of column #0 will be smaller compared to its value before column processing for several columns. For these, fewer bits must then be discarded during the final normalization step (row-iRFFT) resulting in increased overall precision.
As shown in Fig. 7.6 after column alignment, FFT, elementwise multiplications and iFFT are performed in sequence. By processing all columns in this manner a new temporary matrix, also populated with varying BEs (i.e., PFP format), is achieved.

**Row-iRFFT**

The final row-iFFTs are processed very similarly to the row FFTs.

Looking at the resulting UCCF algorithm flow (Fig. 7.6), data computed by the FFT processor is always given in PFP format. On the other hand, due to row-level BE computation in the 'adder part', this increase in numerical dynamic is smoothed during the start-up of each of the three UCCF processing steps. However, individual BEs are preserved for the final result (cf. Fig. 7.6(c)).

Compared to native fixed-point arithmetic, we thus preserve a significant increase in numeric dynamic.

### 7.5 Results

#### Cost Efficient Implementation

By restricting 'adder part’ computation to fixed-point arithmetic and further implementing a dedicated *small-area overflow concept* we allow for simple adder structures and 'lean' controlling. We thus achieve savings in chip area and can therefore reduce overall implementation costs.

#### Precision Improvement Compared with Fixed-Point Arithmetic

In addition, by combining fixed-point 'adder part’ computation with PFP twiddle multiplications we realize new HBE arithmetic. By employing HBE arithmetic, we basically compute the 1D-FFT by two underlying BE scopes. First, we compute $M - 1$ stages by row-level BE arithmetic while changing the BE scope to 'complex-mantissa' for the $M$-th stage. By maintaining the “individual” exponents of the last stage, the overall precision of an $N$-point FFT computed with this concept is comparable to a $N/4$-point FFT computed by native fixed-point computation. Thus, the error variance of a 1024-point 1D-FFT is reduced to a quantity corresponding to a 256-point FFT resulting in a precision improvement of one order of magnitude (cf. Fig. 6.5). Hence,
compared to native fixed-point computation, we achieve a significant increase in precision and numeric dynamic at very moderate costs.

As already mentioned in section 6.4.5, area requirements are reduced by a considerable 71% in comparison to a fixed-point implementation of equivalent accuracy by means of our dedicated HBE arithmetic concept.

**Target Precision Met for all Operation Modes**

Actual precision obtained by HBE arithmetic has been measured in over 1200 experiments using randomly generated input data. Achieved accuracy is depicted in Fig. 7.7 for all APU operation modes, e.g., FFT, elementwise multiplications (MUL), scramble (SCR) and unscramble (UNSCR). The average as well as the maximum and minimum of the maximum relative error of underlying simulation runs are specified. For convenience, supported sequence lengths 256, 512 and 1024 are abbreviated as “S”, “M”, and “L”, respectively.

Referring to the FFT precision depicted in Fig. 7.7, the average of maximal relative errors $^3$ is very close to our target specification of 0.001% (cf. section 6.3.4). Since the other operation modes (e.g., MUL, SCR, UNSCR) are computed by one 'stage', the corresponding accuracy is even better. As a result, our target precision is clearly met for all operation modes.

$^3 E_{R_{256}} = 0.00052674\%, E_{R_{512}} = 0.00117596\%, E_{R_{1024}} = 0.00143539\%$
8. IC-Implementation and Measurements

This chapter focuses on the actual implementation of the FFT processor. After a short discussion of the underlying design strategy, we go into the details of the CMOS technology used and major layout characteristics. In particular, we discuss restrictions imposed by design tool and library on the examples of local memory and multiplier design as well as measures taken to cope with clock skew and peak power supply. Finally, we give detailed performance measures on the realized FFT processor.

8.1 Design Strategy

As mentioned before, one major constraint to system design was given by the demand for a short time to market. To meet this goal, our strategy was to employ 'virtual prototyping' techniques in order to allow for premature ratings of the real-time system’s efficiency.

8.1.1 Virtual Prototyping

To realize the real-time object detection system, we must mutually coordinate the high-level control program (SHARCs), the multiprocessor-system and the interface layer which allows us to control and to configure the embedded PEs. While each of these can, to a certain degree, be tested in isolation from the rest, the potential sources for errors and problems multiply rapidly when these pieces are brought together. Testing for bugs throughout all design phases is therefore a must. As a result, our approach was to establish a virtual system prototype for each relevant design phase.

Starting with the derivation of the new BST algorithm, we modeled
the underlying task-flow by employing the KHOROS* image-processing

The KHOROS software allows for a very modular algorithm description and can be conveniently extended by customer-specific modules. We thus performed a stepwise refinement of the system's prototype model by additionally implementing C models for relevant processor components, e.g., I/O interfacing.

In addition, C models for fixed-point and HBE arithmetic have been established and implemented into the virtual prototype system in order to get first estimations on the impact of finite wordlength effects on the reliability of the object detection procedure.

During the transition to actual processor implementation, the entire processor architecture as well as all major system components were modeled by means of VHDL. By embedding the processor's VHDL description into the virtual multiprocessor system, final algorithm partitioning and mapping of corresponding algorithm-parts onto hardware components originated from stepwise refinement of underlying model descriptions.

8.1.2 Concurrent Multiprocessor Architecture and Processor Design

The multiprocessor architecture dedicated to real-time BST computation (cf. Fig. 3.1) consists of multiple components, e.g., multiple instances of our dedicated FFT processor, SRAM, FPGAs, DSPs, etc., that were to be assembled on a PCB. Concerning the FFT processor itself, it was to be implemented in VLSI in order to achieve maximal performance. As a result, we had two different design levels (e.g., PCB and VLSI design) that could be processed separately. Targeting a short time to market, we decided on the concurrent design of multiprocessor architecture and FFT processor.

Apart from that time issue, our drive for concurrent system and processor design was also motivated by the goal of maximal performance. Recalling the computational requirements imposed on the real-time system, employed design techniques for the board- and silicon-levels must be completely exhausted in order to achieve the most compact system implementation.


†Printed Circuit Board
Each of the two design spheres (e.g., PCB and VLSI) has its distinct limitations. Bus-systems routed on a PCB can be merely clocked up to approximately 100MHz, and on-board connectors, such as headers, impose further limitations on 'system clock'. On the other hand, only certain tasks can be efficiently implemented in VLSI. Implementation of sophisticated control features into the FFT processor would needlessly complicate and enlarge the corresponding VLSI design.

Our approach was thus to exploit synergies on system- and
processor-levels in order to achieve an optimally balanced overall system implementation. As a result of our concurrent design approach, system partitioning was fixed by means of an iterative process (cf. Fig. 8.1). Function set and interfacing of the FFT processor is thus shaped to perfectly fit into the overall system concept.

8.2 FFT Processor Implementation

This section concentrates on the actual implementation of the processor architecture developed in Chapters 1 through 7. Following a common practice, we have to provide the new architecture with a representative name. Combining striking processor features such as pipeline-interleaved butterfly computation and high FFT performance, we composed the name "SPIFFIRE".

The micrograph of the SPIFFIRE layout is shown in Fig. 8.4. To allow for a comparison with processor and APU structure (cf. Figs. 8.2 and 5.11), all major layout components are identified by corresponding labels. In addition, the underlying floor plan is shown in Fig. 8.3.

8.2.1 Process Technology

Concerning processor implementation, the major constraint on the choice of an appropriate technology was imposed by the demand for

\*\*Smart Pipeline Interleaved Fast Fourier Transform Processing Engine\*\*

Figure 8.2: Processor block-diagram (cf. Fig. 4.3).
8.2. FFT Processor Implementation

![Processor floor plan diagram]

Figure 8.3: Processor floor plan.

cost-efficient prototyping. As a result, process technologies offered within MPW\(^8\) runs were preferred.

On the other hand, deciding on MPW implementation generally reduces the options for potential technologies\(^1\). This restriction particularly holds in reference to COMPASS design-automation tools. Even before COMPASS was taken over by AVANTI, no advanced technologies (i.e., showing feature sizes smaller than 0.8\(\mu\)m and offering more than two layers of metal interconnect) were offered within any MPW run.

However, a solution was found according to a large compatibility between two 0.5\(\mu\)m three metal-layer CMOS technologies. While corresponding COMPASS design libraries exist for the HCMOS5 technology (SGS Thomson) the compatible CO5M-D process (Alcatel Mietec) was offered within the Europractice MPW program.

As a result, we performed the entire SPITFFIRE design employing HCMOS5 libraries and realized actual production at Alcatel-Mietec.

8.2.2 Restrictions Due to Design Library and Tools

Although we found a practical way to implement the processor in 0.5\(\mu\)m CMOS technology, we were confronted with several restrictions imposed

---

\(^8\)Multi Project Wafer

\(^1\)Only a restricted number of technologies are offered by MPW runs.
Figure 8.4: FFT processor micro-photograph. Die size: 167mm$^2$ (11.6mm x 14.4mm).
by the HCMOS5 library as well as by the design tool. These restrictions mainly manifested themselves in the unavailability of the required components as well as in the design tool's inability to control the complexity of our processor design. We were thus forced to diverge from our ideal design approach. As a result, final processor implementation was shaped by several restrictions imposed by the HCMOS5 library as well as by the design tool (COMPASS vS8r4.10).

In order to maintain the targeted scheduling, our approach was to locate practicable, though possibly suboptimal, alternatives. The two examples most involved are the design of the local memory and the complex multiplier.

**Impact on Local Memory**

As discussed in section 4.4.1, implementation of two memory banks is required to allow for simultaneous computation and I/O. For this, however, employment of dual-ported memory is assumed since the APU concurrently reads and writes one piece of complex data with each clock cycle.

However, in the special case of the HCMOS5 library, only single-ported memory is available. As a result, three memory banks must be implemented to allow for concurrent computation and I/O along with a dedicated module for flexible data and address routing (cf. Fig. 8.2).

Processor performance is thus significantly reduced since area requirements and memory access time considerably increase.

**Impact on Multiplier Design**

To achieve maximum performance, our objective was to build the CMUL (cf. Fig. 5.12) by implementing fast and compact real-arithmetic multipliers.

Concerning multiplier realization, the HCMOS5 data path library provides an $N \times M$ bit array multiplier. However, propagation delay of a corresponding $32 \times 32$ multiplier amounts to 44.73ns (22MHz). Pipelining would be an appropriate measure for speeding up the multiplication process, but implementation of pipeline stages into the array multiplier is not a feature supported by the HCMOS5 data path compiler.

As a result, other implementation alternatives had to be investigated if we were to meet our target of a multiplier capable of sustaining clock rates of at least 66MHz.
In order to achieve a short design time, full-custom layout could not be considered. Targeting a widely automated design, we first focused on data path compilers of different technologies. Looking at a 0.8μm CMOS technology, the corresponding data path compiler allowed for generation of array multipliers using an arbitrary number of implemented pipeline stages, thus enabling adjustment of multiplier throughput over a wide range of clock rates. After retargeting the generated CMN8-netlist to HCMOS5 (TCSC773), we thus obtained a relatively compact layout (Fig. 8.5) showing a worst-case operation frequency of more than 100MHz.

According to the underlying netlist retarget, we had to realize this array multiplier by means of standard cells. Compared to equivalent macro cell multipliers, area requirements and performance are clearly worse, hence implementing multipliers by means of standard cells is usually avoided.

However, since we had to construct the multiplier by means of standard cells anyway, our second approach was to implement a dedicated multiplier architecture (non array) to achieve better area and speed characteristics.

\[\text{CMN8 by VLSI Technology}\]
Concerning these objectives, a very area-efficient multiplier concept is proposed in [Millar 92]. The proposed multiplier basically employs modified Booth-encoding to reduce the number of partial products which are then efficiently accumulated by means of a Dadda addition scheme (cf. section 5.4.2).

Although very efficient, actual implementation of this multiplier concept failed due to immense wiring overhead. The main reason for this was ineligible cell placement based on the routing tool's inability to notice the regularity of 'encoding part' and 'addition tree'.

The only way to improve cell placement was to break down the layout into fragments, i.e., to create sub-layouts for single encoding cells and single stages of the adder tree. Although area requirements could have been reduced by this approach, speed characteristics also decreased due to some 'long block-interconnects' while total area was still larger than for the array multiplier described above. As a result, we implemented the CMUL by employing the retargeted array multiplier, the layout of which is shown in Fig. 8.5.

8.2.3 Signal Distribution

Facing the resulting chip dimensions of 11.6 x 14.4 mm² (cf. Fig. 8.4), considerable effort was spent on global signal distribution in order to prevent the expected, substantial interconnect delays. Thus, in implementing the APU architecture, all functional blocks were completely encapsulated by pipeline registers and all block interconnects as well as most of the buses were partitioned by a convenient number of pipeline stages.

8.2.4 Minimization of Clock Skew

Regarding the highly pipelined data path architecture, we focused additional effort on global clock distribution and switching noise.

In the absence of a central clock driver ("elephant clock"), we implemented externally-driven buffer trees supplying the APU and I/O clock systems. Using a buffer tree for clock distribution, clock skew has been reduced to an upper bound of 0.8ns by readjusting selected drivers' strength at layout level. To do so, we localized unfavorably-dimensioned clock drivers by means of a graphical delay chart (cf. Fig. 8.6).
8.2.5 Peak Power Supply

In static CMOS circuits, the most important switching currents typically occur in pad-drivers.

Considering a single pad-driver, the series impedance $Z_g$ formed by the package pin, bonding wire and on-chip line includes a resistive part $R_g$ and an inductive part $L_g$ [Kaeslin 96]. The ground bounce voltage is thus obtained as

$$u_g = R_g i_g + L_g \frac{d i_g}{dt},$$

where $i_g$ denotes the sum of the output driver's output current. As a result, ground bounce can be reduced by connecting multiple power-/ground-pads in parallel, thereby reducing $R_g$ and $L_g$. Hence, 50 ground- and 50 power-pads have been implemented to supply the chip core. The driver stages of output- and bidirectional-pads are separately supplied by dedicated power/ground pads.

Concerning peak power supply, two overlapping rings for $V_{DD}$ and $V_{SS}$ were implemented surrounding the chip core, forming an extra capacitance (cf. Fig. 8.4).

![Figure 8.6: Graphical representation of clock tree delay shown, as example, for the 'LM interface' module. The stepwise increase of total delay between the 'driving clock' and the 'local clock' of each individual pipeline flip-flop is shown. As can be seen, all leaves of the delay tree are located between 9.1 and 9.95ns. Hence, the timely difference between all leaves (e.g., clock skew) is smaller than 0.8ns. In cases where actual clock skew is larger than 0.8ns, non-optimally delayed clock tree branches can be located by this type of graphical representation and manually optimized by either speed-up or slow-down techniques.](image-url)
8.2.6 Test Aids

Following our maxim of 'design for testability', all registers of the design are assigned to one of five scan paths. Although overall area is increased by this measure (about 6%), possible design problems can be revealed after chip fabrication.

Furthermore, each of the three memory banks is externally accessible by means of the data and address routing network (DARN). Therefore, no block-isolation techniques have to be implemented to allow for an individual memory test.
8.3 Measures of the New FFT Architecture

This section focuses on actual measurements associated with the SPITFFIRE processor.

After implementation, several tests were performed to determine the processor’s actual power-dissipation and maximum clock speed. The most important processor details are summarized in Table 8.1.

### 8.3.1 Processor Performance

Concerning the real-time computation of the BST algorithm, the relevant performance measures are given by execution time and precision.

Detailed information on corresponding processor characteristics are listed in Table 8.2. Actual figures on accuracy were obtained by a variety of simulations. Referring to the modes MUL, UNSCR and SCR, corresponding results are generally computed within one “stage” and should actually show identical precision. However, as can be seen from Table 8.2, varying accuracies have been obtained due to different addressing schemes applied on the underlying input data.

Recalling that the processor allows for concurrent computation and I/O, specified clock cycles in Table 8.2 represent sustained performance measures. A complex 1024-point FFT can be computed every 5.337 clock cycles. As specified in Table 8.3, a single SPITFFIRE processor

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>FFT, elementwise matrix-multiplication, pre-/post-processing for real-valued FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence lengths</td>
<td>256, 512, 1024-point</td>
</tr>
<tr>
<td>Computing power</td>
<td>924 MOps</td>
</tr>
<tr>
<td>Technology</td>
<td>0.5µm three metal-layer CMOS</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3V</td>
</tr>
<tr>
<td>Die size</td>
<td>11.6mm×14.4mm (167mm²)</td>
</tr>
<tr>
<td>Transistors</td>
<td>2.3M</td>
</tr>
<tr>
<td>Clock rate</td>
<td>66MHz</td>
</tr>
<tr>
<td>Power consumption</td>
<td>1.9W at 66MHz</td>
</tr>
<tr>
<td>On-chip memory</td>
<td>3 × 9kByte</td>
</tr>
<tr>
<td>I/O bandwidth</td>
<td>230 MByte/sec. @50MHz</td>
</tr>
<tr>
<td>Package</td>
<td>PGA 180, ceramic</td>
</tr>
</tbody>
</table>

Table 8.1: Processor specifications.
8.3. Measures of the New FFT Architecture

<table>
<thead>
<tr>
<th>Mode</th>
<th>total clock cycles</th>
<th>execution time (66MHz)</th>
<th>quantization error in % (Av.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024-pt FFT</td>
<td>5337</td>
<td>80.9μs</td>
<td>1.44 \cdot 10^{-3}</td>
</tr>
<tr>
<td>512-pt FFT</td>
<td>2665</td>
<td>40.4μs</td>
<td>1.18 \cdot 10^{-3}</td>
</tr>
<tr>
<td>256-pt FFT</td>
<td>1068</td>
<td>16.2μs</td>
<td>5.27 \cdot 10^{-4}</td>
</tr>
<tr>
<td>1024-pt MUL</td>
<td>2095</td>
<td>31.7μs</td>
<td>8.72 \cdot 10^{-6}</td>
</tr>
<tr>
<td>512-pt MUL</td>
<td>1047</td>
<td>15.9μs</td>
<td>8.87 \cdot 10^{-6}</td>
</tr>
<tr>
<td>256-pt MUL</td>
<td>524</td>
<td>7.9μs</td>
<td>1.42 \cdot 10^{-5}</td>
</tr>
<tr>
<td>1024-pt UNSCR</td>
<td>1050</td>
<td>15.9μs</td>
<td>7.57 \cdot 10^{-6}</td>
</tr>
<tr>
<td>512-pt UNSCR</td>
<td>532</td>
<td>8.1μs</td>
<td>7.53 \cdot 10^{-6}</td>
</tr>
<tr>
<td>256-pt UNSCR</td>
<td>273</td>
<td>4.1μs</td>
<td>9.21 \cdot 10^{-6}</td>
</tr>
<tr>
<td>1024-pt SCR</td>
<td>1051</td>
<td>15.9μs</td>
<td>7.96 \cdot 10^{-6}</td>
</tr>
<tr>
<td>512-pt SCR</td>
<td>533</td>
<td>8.1μs</td>
<td>8.24 \cdot 10^{-6}</td>
</tr>
<tr>
<td>256-pt SCR</td>
<td>274</td>
<td>4.2μs</td>
<td>9.00 \cdot 10^{-6}</td>
</tr>
</tbody>
</table>

Table 8.2: Processor performance @ 66MHz.

<table>
<thead>
<tr>
<th>image size</th>
<th>clock cycles</th>
<th>exec.-time</th>
<th>frames/sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 x 1024</td>
<td>26 182 594</td>
<td>0.40 sec.</td>
<td>2.52</td>
</tr>
<tr>
<td>512 x 512</td>
<td>6 552 018</td>
<td>0.10 sec.</td>
<td>10.0</td>
</tr>
<tr>
<td>256 x 256</td>
<td>1 373 128</td>
<td>0.02 sec.</td>
<td>48.07</td>
</tr>
</tbody>
</table>

Table 8.3: BST-performance of a single SPITFFIRE processor (66MHz).

thus completes BST computation of a 1024 × 1024 frame within 0.4 seconds. In particular for image sizes smaller than 512 × 512, real-time computation of the BST algorithm can be realized employing only a single SPITFFIRE processor.
9. Comparison with Existing Processors

In this chapter, we compare the SPITFFIRE architecture to other, existing processor designs. Since almost every one of the confronted designs is realized by a different design technique (e.g., standard cell or full-custom design) and technology (e.g., feature size, number of routing-layers, etc.), we have to compensate for these effects in order to allow for an objective rating. In addition, previous FFT designs show typical fixed-point wordlengths of 16 to 24 bits, allowing for comparably compact VLSI implementation. Since silicon area can also be employed as an 'efficiency measure', we develop a model to estimate the SPITFFIRE’s die size for arbitrary wordlengths, allowing for a corresponding 'area comparison'.

Finally, based on our models, we compare our SPITFFIRE architecture to other processor designs. To conclude, under compensatory consideration of execution time, die size and accuracy, our SPITFFIRE design achieves a rating about four orders of magnitude better than any other fixed-point FFT processor.

9.1 Impact of Advanced IC Technologies

As discussed, several limitations that prevented ideal processor implementation existed. Most of these restrictions were imposed by the design-automation tool (COMPASS V8r4.10). The complexity of the SPITFFIRE-processor was definitely too large to be efficiently handled by the design tool. As a result and as indicated by the processor layout (Fig. 8.4), overall wiring overhead (about 70%) is too large. One major reason for this is based on unfavorable cell placement. Also, referring to hierarchical layout, the routing tool was incapable of utilizing free routing-space of hierarchically lower layout-levels.
A further performance limitation is imposed by technology. Looking at today's 0.18\(\mu\)m CMOS technologies, performance of a new architecture implemented by an 'obsolete' 0.5\(\mu\)m CMOS process cannot be exploited to its (architectural) limit.

This effect is further enforced by employing standard cell design techniques. In particular, referring to the HCMOS5 library with which the processor is realized, standard cells are optimized for low power dissipation rather than for speed. Also, further performance limitations are imposed by the lack of fast dual-ported SRAM, fast ROMs and fast multipliers (cf. section 8.2.2).

Hence, to allow for a relevant performance rating of our FFT architecture, we have to model the impact of advanced IC-technologies (e.g., full-custom layout techniques and employment of deep sub-micron CMOS processes) on dedicated processor characteristics (e.g., die size and clock rate).

### 9.1.1 Area Reduction by Full-Custom Layout

Looking at the processor layout (Fig. 8.4), we see a considerable amount of available silicon area that remains unused due to the peculiarities of the employed routing tool (see above). We can thus expect significant area reduction when implementing the SPITFFIRE processor by means of full-custom layout techniques.

To obtain concrete figures, we performed an actual 'full-custom' area estimation in three steps:

1. First, we determined the local wiring-factor (WF) for each of the layout's major building blocks. To do this, we compared the total area of each of these blocks against the actual amount of underlying standard cell area. Looking at the different layout blocks, corresponding WFs vary between 1.2 and 5.43 (cf. Table 9.1). Since WF is usually between 1.2 and 3 for standard cell designs, we assume a total wiring overhead of 20\% for the case of full-custom layout (WF=1.2). This is a conservative estimation since for most full-custom designs WF is smaller than 1.2.

2. In the second step, according to our assumption, we reduced the wiring overhead for standard cell routing to 20\%. Corresponding area specifications are listed in Table 9.1. In addition, the general interrelation between wiring overhead and total chip area is graphically depicted in Fig. 9.1.
3. Finally, to obtain the desired area estimation for full-custom layout, we further assumed the availability of dual-ported SRAM*, more efficient block-interconnect routing as well as increased compactness for generated macro cells. Concrete area figures determined by this approach are reflected in the rightmost column of Table 9.1.

As a result, by implementing the SPITFFIRE FFT processor using full-custom design techniques, area requirements can be reduced from 167mm² to 96mm², i.e., by 42%. Although quite impressive, this estimation of area reduction is rather pessimistic since we assumed the pad-frame to be unaffected by full-custom layout. However, by reducing active cell area we also decrease overall power dissipation. Less power/ground pins are thus needed, allowing for a more compact pad-frame.

9.1.2 Speed-Up and Area-Reduction by Employment of Deep Sub-Micron CMOS Technologies

As discussed in Chapter 8, we employed-commercially available 0.5μm CMOS technology to implement the SPITFFIRE processor. In reference to standard cell design techniques, clock rates of about 100MHz can

*Only two memory banks are thus required to allow for simultaneous computation and I/O.
## Table 9.1: Area requirements for major layout blocks

<table>
<thead>
<tr>
<th>Layout-Block</th>
<th>Total Area</th>
<th>Standard Cell Design</th>
<th>Full-Custom (WF=1.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cell Area</td>
<td>Wiring Factor</td>
<td>A [mm²]</td>
</tr>
<tr>
<td>Registerfile 1</td>
<td>8.781</td>
<td>1.856</td>
<td>4.73</td>
</tr>
<tr>
<td>Registerfile 2</td>
<td>10.12</td>
<td>1.863</td>
<td>5.43</td>
</tr>
<tr>
<td>CADD (2x)</td>
<td>1.163</td>
<td>0.973</td>
<td>1.20</td>
</tr>
<tr>
<td>CMUL</td>
<td>22.04</td>
<td>8.696</td>
<td>2.53</td>
</tr>
<tr>
<td>Controlling</td>
<td>4.622</td>
<td>1.977</td>
<td>2.34</td>
</tr>
<tr>
<td>LM-Bank</td>
<td>15.12</td>
<td>12.041</td>
<td>1.26</td>
</tr>
<tr>
<td>Coeff. ROM</td>
<td>3.745</td>
<td>1.720</td>
<td>2.18</td>
</tr>
<tr>
<td>I/O Interface</td>
<td>2.015</td>
<td>0.968</td>
<td>2.08</td>
</tr>
<tr>
<td>Pad Frame</td>
<td>15.61</td>
<td></td>
<td>&lt;15.61</td>
</tr>
<tr>
<td>TOTAL</td>
<td>167</td>
<td>114.6</td>
<td>1.46</td>
</tr>
</tbody>
</table>

thus be achieved. On the other hand, today's state-of-the-art DSPs and general purpose CPUs are generally based on advanced 0.25/μm and 0.35/μm technologies, thereby achieving clock rates of above 200MHz (cf. Table 9.3).

Looking at the next few years, considerable shrinkage of feature size can be expected. According to current road-maps, 0.07/μm CMOS technology is scheduled for the year 2010 [Geppert 96]. Furthermore, along with decreasing feature sizes, the number of wiring-layers will increase from 4-5 (0.35/μm) to 7-8 (0.07/μm). Additional reduction in wiring overhead and die size is thus achieved and will involve a further down-sizing of interconnect delay and higher clock rates.

### Copper Interconnect

The continuing decrease of interconnect delay is strongly supported by the growing application of copper for wiring purposes. Compared to aluminum, copper is characterized by a considerably lower resistance.
Signal delays are thus reduced while, at the same time, the maximal wire-length is increased.

Today, copper interconnect is available from IBM (CMOS-7S), Motorola and VLSI Technology (VSC11). Corresponding processes show a typical feature size of 0.15µm and employ copper metal for the fourth and fifth metalization layer. Looking at forthcoming 'copper technologies', i.e., feature sizes smaller than 0.15µm, the number of copper layers will increase. As a result, interconnect-delay will exponentially decrease for advanced deep sub-micron CMOS processes, thus allowing for clock rates above 1000MHz.

Scaling of Processor Characteristics

Looking at the possibilities presented by advanced technologies, we must now ask to what extent the SPITFFIRE architecture can profit. To do so, our approach is to scale relevant processor-properties, i.e., processor area and clock rate, according to the characteristics of deep sub-micron CMOS technologies.

To obtain reliable scaling estimations, we performed a mapping between relevant road-map information† ([Geppert 96]) and corresponding characteristics of the SPITFFIRE architecture. Resulting scaling-estimations obtained by this method are quantitatively listed in Table 9.2. The figures reflect the impact of CMOS technology on processor area, clock rate, power dissipation and BST performance, assuming the employment of copper interconnect for feature sizes smaller than 0.15µm. A graphical representation of the interrelationship between feature size, processor area and clock rate is shown in Fig. 9.2.

Due to the highly pipelined APU architecture, we can completely exploit the speed-up enabled by advanced copper technologies. We can thus achieve a progressive scaling of operating clock rate, as indicated by Fig. 9.2(a).

The area-estimations given in Fig. 9.2(b) are computed according to typical transistor densities associated with the respective technology. To conclude, the proposed FFT architecture ideally scales with the increased performance offered by advanced deep sub-micron CMOS technologies.

†transistor densities, supply voltages, gate delays
<table>
<thead>
<tr>
<th>Layout Technique</th>
<th>Std. Cell</th>
<th>Full-Custom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Size</td>
<td>0.5 (\mu)</td>
<td>0.35 (\mu)</td>
</tr>
<tr>
<td>Clock Rate [MHz]</td>
<td>66</td>
<td>100</td>
</tr>
<tr>
<td>Supply Voltage [V]</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>Power Dissipation [W]</td>
<td>1.9</td>
<td>2.5</td>
</tr>
<tr>
<td>Core Size [mm(^2)]</td>
<td>150</td>
<td>80</td>
</tr>
<tr>
<td>Pad Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin Count [#]</td>
<td>180</td>
<td>120</td>
</tr>
<tr>
<td>Wire-bond Pitch [(\mu)m]</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>Pad Height [(\mu)m]</td>
<td>667</td>
<td>667</td>
</tr>
<tr>
<td>Die Size [mm(^2)]</td>
<td>167</td>
<td>95.8</td>
</tr>
<tr>
<td>BST Performance (\frac{frames}{sec.})</td>
<td>2.7</td>
<td>4.0</td>
</tr>
</tbody>
</table>

\(^a\)BST algorithm with 1024 \(\times\) 1024 search area

Table 9.2: Scaling of processor-characteristics by advanced CMOS technologies.
9.1. Impact of Advanced IC Technologies

Interrelation of Feature Size and Clock Rate

(a) Clock Rate.

Interrelation of Feature Size and Full-Custom Area

(b) Area.

Figure 9.2: Interrelation of SPITFFIRE processor characteristics and feature size (full-custom layout).
9.2 Interrelation between Die Size and Mantissa Wordlength

In addition to FFT performance, the silicon area required for implementation is another important characteristic that concerns the rating of processor efficiency. However, in order to allow for a relevant area comparison of different architecture types, underlying candidates must have the same wordlength.

Since existing fixed-point FFT processors dispose of wordlengths between 16 and 24 bit (cf. Table 4.1), we must provide a method of estimating an area measure of the 32-bit SPITFFIRE processor for a given wordlength \( b \).

To do so, we derive a simple model reflecting the impact of a variation in wordlength \( b \) on the area of discrete layout components. Concerning the interrelation between area and wordlength, we can classify the layout components into three categories:

1. First, there are some layout blocks, e.g., controllers, the area of which is completely unaffected by variation of wordlength \( b \).

2. Next, elements such as adders, registerfiles and memories show a linear dependency between area and \( b \).

3. Finally, we have to assume a quadratic area-dependency for all multipliers.

![Full-Custom Layout Wiring Factor vs Wordlength](image)

**Figure 9.3**: Observed interrelation between corrected wiring-factor \( w' \) and mantissa wordlength \( b \) (cf. Eq. 9.2).
Given an arbitrary wordlength $b$, we thus compute the corresponding total area $A_B$ of a single layout block as

$$A_B = A_C \cdot \left( \frac{b}{b_0} \right)^k \cdot w', \quad k \in \{0, 1, 2\}$$  \hspace{1cm} (9.1)$$

where $b_0$ represents our standard wordlength (e.g., 32b), $A_C$ and $w'$ denote the underlying cell-area and corrected wiring-factor, respectively.

As indicated by the term "corrected", we have to adapt the actual wiring factor to underlying mantissa wordlength. This is because we observed a reduction of the wiring overhead $w$ along with a decrement in wordlength $b$ (cf. Fig. 9.3). To determine the dependency between wiring overhead $w$ and wordlength $b$, we generated standard cell layouts for representative layout-components (e.g., registerfile sub-blocks, CADD, ASR pipeline stages) for the selected wordlengths $b=20, 24, 28$ and 32 bit. Averaging the empirical observations for each of our test-components, this dependency can be approximated by

$$w' = w_0 \cdot (0.87875 + 0.00375 b), \quad \text{with } b \in [10..32].$$ \hspace{1cm} (9.2)$$

Based on this model, we can compute the corresponding $A_B$ for all layout components for a given wordlength $b$. An estimation of the total chip area is then obtained by accumulation of all $A_B$ ($\Rightarrow$core size) and by adding a corresponding pad-frame geometry.

Figure 9.4: Full-Custom design (cf. section 9.1.1): Interrelation of mantissa wordlength $b$ and total chip area. To complete the picture, figures are also given for actual 'core size'.
Actual area-estimations obtained by this model are depicted in Fig. 9.4 for both core size and total die size (e.g., inclusive pad-frame). Although the figures given in this diagram refer to full-custom layout (cf. section 9.1.1), they also reflect the qualitative tendency for the standard cell case.

As extracted from Fig. 9.4, area requirements are reduced from 95.8mm$^2$ to 52.7mm$^2$ by shrinking mantissa wordlength from 32 to 24b (full-custom case). This considerable reduction of 45% is mainly caused by the quadratic area dependency of the three real multipliers.

### 9.3 Confronting FFT Performance

Explicitly developed to match the requirements of Fourier domain correlation, the SPITFFIRE processor is consequently the best choice concerning efficient computation of the BST algorithm. However, regarding its optimized butterfly unit, it also shows excellent FFT performance and hence outstrips almost every one of its competitors.

For a selected set of available processors, an actual rating of corresponding FFT performance is listed in Table 9.3. Here, the SPITFFIRE processor is compared to dedicated FFT architectures as well as to today’s high-end general-purpose CPUs and DSPs.

As can be extracted from Table 9.3, the listed competitors are realized in different technologies. To allow for a meaningful rating of FFT performance, we have to compensate for speed-up effects provided by advanced technologies. To do so, we scale corresponding FFT performance measures according to a fictive 0.18µm$^2$ CMOS implementation for each processor. The corresponding speed-up factors required for this scaling are obtained from the clock rate ratio indirectly given by Table 9.2.

### 9.3.1 General-Purpose CPUs

Looking at the “normalized” FFT execution times, worst FFT performance is provided by general-purpose processors. Although operated at comparatively high clock rates, their flexible ALU architecture does not allow for any meaningful parallelization of the FFT algorithm as indicated by the large cycle counts (source: [BDTI 97]).

---

This is the smallest feature size employed by the designs listed in Table 9.3.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GP CPUs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium 200</td>
<td>0.35</td>
<td>float</td>
<td>float</td>
<td>200</td>
<td>158750</td>
<td>793.75</td>
<td>1.71</td>
<td>464.18</td>
</tr>
<tr>
<td>PPC604e</td>
<td>0.25</td>
<td>float</td>
<td>float</td>
<td>200</td>
<td>110000</td>
<td>550.00</td>
<td>1.33</td>
<td>413.53</td>
</tr>
<tr>
<td><strong>DSPs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADSP-21062</td>
<td>0.25</td>
<td>32</td>
<td>float</td>
<td>40</td>
<td>18221</td>
<td>455.53</td>
<td>1.33</td>
<td>342.5</td>
</tr>
<tr>
<td>TMS-320C6701</td>
<td>0.18</td>
<td>32</td>
<td>float</td>
<td>167</td>
<td>20880</td>
<td>125.03</td>
<td>1</td>
<td>125.03</td>
</tr>
<tr>
<td>TMS-320C6202</td>
<td>0.25</td>
<td>32</td>
<td>fixed</td>
<td>250</td>
<td>15048</td>
<td>704</td>
<td>1.33</td>
<td>529</td>
</tr>
<tr>
<td><strong>FFT Archs.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BDSP91V24</td>
<td>0.5</td>
<td>24</td>
<td>fixed</td>
<td>80</td>
<td>3224</td>
<td>40.30</td>
<td>2.18</td>
<td>18.47</td>
</tr>
<tr>
<td>DSP-24</td>
<td>0.5</td>
<td>24</td>
<td>fixed</td>
<td>100</td>
<td>2100</td>
<td>21.00</td>
<td>2.18</td>
<td>9.63</td>
</tr>
<tr>
<td>SPIFEE</td>
<td>0.6</td>
<td>20</td>
<td>fixed</td>
<td>85</td>
<td>5185</td>
<td>30.00</td>
<td>2.30</td>
<td>13.04</td>
</tr>
<tr>
<td>PDSP16510A</td>
<td>1.4</td>
<td>16</td>
<td>fixed</td>
<td>40</td>
<td>3920</td>
<td>98.00</td>
<td>≈8</td>
<td>12.25</td>
</tr>
<tr>
<td>SNC960A</td>
<td>0.6</td>
<td>16</td>
<td>fixed</td>
<td>65</td>
<td>1300</td>
<td>20.00</td>
<td>2.30</td>
<td>8.70</td>
</tr>
<tr>
<td>CNET</td>
<td>0.5</td>
<td>10</td>
<td>fixed</td>
<td>20</td>
<td>1020</td>
<td>51.00</td>
<td>2.18</td>
<td>23.38</td>
</tr>
<tr>
<td>SPITFFIRE</td>
<td>0.5</td>
<td>2 × 32+8</td>
<td>block-exp.</td>
<td>66</td>
<td>5337</td>
<td>80.86</td>
<td>3.26&lt;sup&gt;a&lt;/sup&gt;</td>
<td>24.8</td>
</tr>
</tbody>
</table>

<sup>a</sup>215MHz (0.18µm full-custom, cf. Table 9.2) /66MHz (0.5µm standard cell)

Table 9.3: Performance of state-of-the-art FFT processors (1024-point FFT). The speed-up factors required for normalization to 0.18µm feature size are obtained by means of Fig. 9.2(a).
9.3.2 Digital Signal Processors

Looking at the FFT performance of today's commercially-available DSP processors (cf. Table 9.3), the shortest computation times are achieved by the TMS-320C67x and C62x, respectively.

The C6202 is a fixed-point processor. TI claims that a 1024-point complex radix-4 FFT can be performed in 60μs [TI 98a]. To achieve this specified transform time, the two on-chip MAC units must be completely utilized. Very highly optimized assembler code is required for this. Furthermore, it is assumed that the processor is running at 250MHz and does not block for instruction fetches. In a real environment, however, we have to expect lower FFT performance due to I/O limitations of the C6x family. In addition, the above FFT specification refers to 16b precision. As the C62xx family disposes of two 16b multipliers only, computation time significantly increases as more precision is required. The total computation time for a 32b radix-4 1024-point FFT thus amounts to 704μs [Matusiak 98].

The latest model of the C67x floating-point branch is represented by the C6701. According to TI, it will be available as of Q02/99 and will compute a 1024-point complex FFT within 125μs [TI 98b]. On the other hand and in terms of the fixed-point versions, this processor will show many of the same speed limitations as those stemming from I/O.

9.3.3 Dedicated FFT Architectures

Corresponding execution times are considerably better for dedicated FFT architectures (cf. Table 9.3). However, up to now, high precision was obviously not a targeted design objective since only a maximum wordlength of 24b is provided. In addition, the majority of these architectures do not allow for computation of non-FFT tasks, and therefore are unsuited to real-time BST computation.

Considerations on Required Clock Cycle Count

A second relevant issue to FFT performance is the number of required clock cycles. Referring to the radix-4-based 1024-point FFT, approximately 15,360 complex operations are needed, assuming a single radix-4 butterfly to be computed by eight complex additions and four complex multiplications. Referring to the set of dedicated FFT architectures listed in Table 9.3, five processors require less than 5000 cycles to compute a complex 1024-point FFT. While this seems to be a very advan-
tageous characteristic at first glance, several efficiency-related problems can be derived from this property.

As discussed in section 5.3.2, radix-4 computation is efficiently performed by two complex adders and one complex multiplier. Implementing these three arithmetic resources, the total number of clock cycles is divided by three, resulting in "efficient" 5,120 cycles, which is also confirmed by the SPIFEE architecture (cf. Table 9.3). Consequently, further reduction of required clock cycles by provision of additional parallelism obviously imposes difficulties according to efficient resource utilization.

However, assuming maximal parallelization at complex operation level and further assuming that each complex operation can be performed within one clock cycle, the minimal cycle count for a 1024-point FFT computes to \( \approx 15,360/12 = 1,280 \) cycles. To further reduce the required cycle count beyond this limit, parallelization by means of multiple butterfly units is required. This is obviously the case for the CNET FFT design (10b fixed-point wordlength) which shows a total cycle count of merely 1,020 cycles. However, this approach is definitely unsuited to large wordlengths (e.g., 32b) because of its large area requirements.

"AT-product"

By looking at actual execution times for a complex 1024-point FFT, we rate underlying architectures according to their performance only.

However, concerning actual efficiency rating of different FFT architectures, a more relevant set of criteria is given by costs and performance. Hence, our next approach to establishing a corresponding efficiency measure is to register costs and performance by means of silicon area \( (A) \) and computation time \( (T) \), respectively. Targeting a compensatory combination of \( A \) and \( T \) (i.e., a small die size can "compensate" for a longer execution time), our approach is to employ the "AT-product" \( (A \times T) \) in order to compute actual efficiency measures. Architectures characterized by a lower AT-product are thus considered to be more efficient than architectures with a higher one.

An overview of state-of-the-art FFT processors is given in Table 9.4. Corresponding AT-products of a selected\(^8\) set of FFT processors are listed in Table 9.5. To allow for a reasonable comparison of different

---

\(^8\) Each radix-4 butterfly requires 12 complex operations.
\(^9\) e.g., for which both numbers on silicon area and execution time for a complex 1024-point FFT are accessible.
<table>
<thead>
<tr>
<th>Processor</th>
<th>$T^a$ [μs]</th>
<th>Wordlength [bit]</th>
<th>rel. Quant. Err. (av.) [%]$^d$</th>
<th>$A_{0.5}^b$ [mm$^2$]</th>
<th>$T_{0.5}^c$ [μs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIFFIRE, 100MHz $^e$</td>
<td>52.8</td>
<td>$2 \times 32 + 8$</td>
<td>0.0014</td>
<td>95.8</td>
<td>52.8</td>
</tr>
<tr>
<td>SPIFEE 1 [Baas 98]</td>
<td>30</td>
<td>20</td>
<td>61.5</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>CNET [Bidet 95]</td>
<td>51</td>
<td>10</td>
<td>1530</td>
<td>100</td>
<td>51</td>
</tr>
<tr>
<td>Commercial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP-24 (DSP Architectures)</td>
<td>21</td>
<td>24</td>
<td>10.2</td>
<td>217</td>
<td>21</td>
</tr>
<tr>
<td>L64280 (LSI) $^f$</td>
<td>26</td>
<td>20</td>
<td>61.5</td>
<td>233</td>
<td>10</td>
</tr>
<tr>
<td>PDSP16510A (Plessey)</td>
<td>98</td>
<td>16</td>
<td>134</td>
<td>22</td>
<td>35</td>
</tr>
<tr>
<td>BSDP9124 (Butterfly DSP)</td>
<td>54</td>
<td>24</td>
<td>10.2</td>
<td>???</td>
<td>47</td>
</tr>
<tr>
<td>SNC960A (Sicom)</td>
<td>20</td>
<td>16</td>
<td>134</td>
<td>???</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 9.4: Overview of state-of-the-art FFT processors.

$^a$Execution time for complex 1024-point FFT
$^b$A, normalized to 0.5μm feature size
$^c$T, normalized to 0.5μm feature size
$^d$cf. Fig. 6.8
$^e$full-custom design, cf. Table 9.2
$^f$chip-set
AT-products, underlying $A$ and $T$ criteria are scaled to 0.5$\mu$m feature size. In addition, computation of the AT-product is classified according to different wordlengths. For each of these, the die size of the SPITFFIRE processor is adjusted accordingly (cf. Fig. 9.4).

As expressed by the AT-products listed in Table 9.5, the SPITFFIRE implementation can be considered to be more “efficient” than the CNET architecture ([Bidet 95]) and the DSP-24. However, the ‘SPIFEE 1’ and the PDSP16510A have a lower AT-product. In particular, the ‘SPIFEE 1’ architecture seems to be considerably more efficient compared to the SPITFFIRE design. As discussed in [Baas 98], the fast clock rates of the ‘SPIFEE 1’ architecture (173MHz) are enabled by a combination of dedicated caching strategy and dedicated CMOS technology (Stanford proprietary). For comparison, performance measures for the SPIFEE architecture implemented on 0.5$\mu$m ultra-low-power (ULP) CMOS are also given (SPIFEE ULP). In addition, the SPIFEE and SPITFFIRE designs differ in their memory characteristics. While two memory banks are implemented on the SPITFFIRE architecture, SPIFEE contains only one local memory. Hence, to obtain a relevant comparison, we modeled the SPIFEE area for the case of a two LM implementation (SPIFEE 2LM). As a result, the large difference between AT-products that corresponds to the two processor designs diminishes.

Although we adjusted the SPITFFIRE’s area according to the wordlength of each of the other designs listed in Table 9.5, we did not consider the potential speed-up corresponding to a reduction in wordlength. However, for shorter wordlengths, arithmetic resources can be more compactly realized, resulting in reduced block-interconnect delays and hence higher clock rates.

**PT and PTA Products**

Hence, to obtain a meaningful comparison, wordlength must be considered. Instead of adapting the SPITFFIRE’s area and clock rate characteristics for each individual comparison with a different design, our approach to establishing a more reasonable rating is to directly include “wordlength” into the efficiency-measure.

**PT-product.** To do so, our third approach is to rate precision ($P$) (thereby considering wordlength) versus performance ($T$) (e.g., corresponding FFT throughput).\footnote{This is because area information is not available for all FFT designs listed in Table 9.4} Corresponding $P \times T$ results for the
### Table 9.5: Performance comparison of selected FFT processors by means of the “AT-product”. Comparison is classified according to underlying wordlengths. For each, the SPITFFIRE’s die size is adapted according to Fig. 9.4. To allow for a reasonable comparison, criteria (e.g., $A, T$) are normalized to 0.5\(\mu\)m feature size.

<table>
<thead>
<tr>
<th>Processor</th>
<th>ft. size $[\mu m]$</th>
<th>$T$ $^a$ [$\mu s$]</th>
<th>$A$ $^b$ [$mm^2$]</th>
<th>$T'$ $^c$ [$\mu s$]</th>
<th>$A'$ $^d$ [$mm^2$]</th>
<th>$A' \cdot T'$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>10b mantissa wordlength</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPITFFIRE$_{10b}$</td>
<td>0.5</td>
<td>52.8</td>
<td>95.8</td>
<td>52.8</td>
<td>30.19</td>
<td>1594</td>
</tr>
<tr>
<td>CNET</td>
<td>0.5</td>
<td>51</td>
<td>100</td>
<td>51</td>
<td>100</td>
<td>5100</td>
</tr>
<tr>
<td><strong>16b mantissa wordlength</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPITFFIRE$_{16b}$</td>
<td>0.5</td>
<td>52.8</td>
<td>95.8</td>
<td>52.8</td>
<td>45.34</td>
<td>2394</td>
</tr>
<tr>
<td>PDSP 16510A</td>
<td>1.4</td>
<td>98</td>
<td>172</td>
<td>35</td>
<td>22</td>
<td>770</td>
</tr>
<tr>
<td><strong>20b mantissa wordlength</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPITFFIRE$_{20b}$</td>
<td>0.5</td>
<td>52.8</td>
<td>95.8</td>
<td>52.8</td>
<td>56.47</td>
<td>2982</td>
</tr>
<tr>
<td>SPIFEE 1</td>
<td>0.7 $^e$</td>
<td>30</td>
<td>49</td>
<td>25</td>
<td>25</td>
<td>625</td>
</tr>
<tr>
<td>SPIFEE ULP</td>
<td>0.5</td>
<td>61</td>
<td>25</td>
<td>61</td>
<td>25</td>
<td>1525</td>
</tr>
<tr>
<td>SPIFEE 2LM$^f$</td>
<td>0.5</td>
<td>61</td>
<td>33.3</td>
<td>61</td>
<td>33.3</td>
<td>2031</td>
</tr>
<tr>
<td><strong>24b mantissa wordlength</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPITFFIRE$_{24b}$</td>
<td>0.5</td>
<td>52.8</td>
<td>95.8</td>
<td>52.8</td>
<td>68.54</td>
<td>3619</td>
</tr>
<tr>
<td>DSP-24</td>
<td>0.5</td>
<td>21</td>
<td>217</td>
<td>21</td>
<td>217</td>
<td>4557</td>
</tr>
</tbody>
</table>

---

$^a$Execution time for complex 1024-point FFT

$^b$Die size

$^c$ $T$ normalized to 0.5\(\mu\)m feature size

$^d$ $A$ normalized to 0.5\(\mu\)m feature size

$^e$0.6$_{eff}$

$^f$To achieve identical memory characteristics for both the SPIFEE and the SPITFFIRE processor, we modeled the total area corresponding to a 2LM SPIFEE implementation.
9.3. Confronting FFT Performance

Figure 9.5: “Performance versus Precision” rating of various FFT processors ($PT$-product). The number of 1024-point 1D-FFTs computed per second versus underlying relative quantization error (average, cf. Fig. 6.8) is shown.

Figure 9.6: “Execution Time versus Precision and Silicon Area” ($PTA$-product): rating of various FFT processors. The execution time for a complex 1024-point 1D-FFT versus the product of corresponding chip-size and relative quantization error (average) is shown. 'Area' and 'Execution Time' are normalized for 0.5µm feature size (cf. Table 9.4).
designs listed in Table 9.4 are given in Fig. 9.5. As a result, the SPITFFIRE's $P \times T$ measure (26.5) is about factor 7,123 lower compared to the second 'best' design (BDSP9124, $P \times T = 188,889$).

**PTA-product.** Further, to additionally reconsider the area cost criterion, we extend our $P \times T$ measure by silicon area $A$. We thus achieve a PTA efficiency measure ($P \times T \times A$). Corresponding measures are shown in Fig. 9.6. As we can see from this figure, compared to the 'SPIFEE 1' ($PTA = 38,438$), the PTA measure of the SPITFFIRE design is about a factor of 5.491 lower ($PTA = 7$). Referring to 'SPIFEE 2LM' ($PTA = 124,925$), 'efficiency-advantage' is even larger (factor 17,846).

Hence, considering the three criteria silicon area, execution time and wordlength (i.e., precision), the SPITFFIRE architecture is about four orders of magnitude more efficient than the second best architecture (SPIFEE).

### 9.4 Conclusion

Targeting high-precision FFT computation, the SPITFFIRE processor is the only alternative beside high-end DSPs and general-purpose CPUs since previous fixed-point FFT processors have a maximal wordlength of 24b. However, as opposed to its high-precision competitors, the SPITFFIRE architecture allows for concurrent computation and I/O, thereby avoiding computation interruptions due to external data exchange. A complex 1024-point FFT can thus be computed every 5,337 clock cycles.

As a result, the SPITFFIRE processor is about six times faster than the fastest DSP available (SHARC). However, the performance of high-precision DSPs is growing rapidly. The TMS320C6702 floating-point DSP announced for Q02/99 is supposed to compute a complex 1024-point-FFT within 125μs. The timely development of the performance of high-precision DSPs is shown in Fig. 9.7. As expressed by this figure, high-end floating-point DSPs will overtake the FFT performance of today's SPITFFIRE processor in 2004. However, assuming a mapping of the SPITFFIRE architecture to equivalent future technologies, it will maintain its performance advantage compared to the more generic DSP architectures.

Compared with dedicated FFT processors, the SPITFFIRE's PTA rating is about four orders of magnitude better than that of previous
9.4. Conclusion

Figure 9.7: FFT performance of state-of-the-art high-precision DSPs (e.g., floating-point and 32-bit fixed-point). As expressed by the diagram, high-end floating-point DSPs will overtake the FFT performance of today's SPITFFIRE processor in 2004.

designs. Also, referring to its extended functional scope, real-valued FFT techniques are directly supported by hardware. An average of 3,194 clock cycles is thus required to transform a real-valued 1024-point sequence (48.4 μs @66MHz).

As a result, the SPITFFIRE processor is the best choice when high-performance and high-precision FFT computation is required.
10. Possible Extensions

In this chapter, we focus on potential extensions and modifications to the processor design as well as to the design-flow.

Referring to the implementation of the SPITFFIRE processor, we propose the use of MCM assembly techniques in order to increase yield and to reduce system size. In addition, we discuss methods of increasing the processor's versatility, allowing us to process other object detection algorithms by means of the SPITFFIRE architecture.

Finally, referring to the excellent speed and precision characteristics of our SPITFFIRE design, we focus on additional applications requiring fast FFT.

10.1 Potential Improvements to Automated IC-Design

Looking back at the problems and restrictions encountered during the entire design phase, several improvements to automated IC design are desirable.

The first demand addresses more powerful design tools requiring significantly less hierarchical partitioning to efficiently control design processes of large complexity. In particular, the provision of reliable timing-driven place and route is a must. Aside from tool-specific problems, a general improvement of 'VHDL design' would offer great advantages concerning automated design and portability. In the following, we will discuss the latter two items in more detail.

10.1.1 Timing-Driven Place and Route

Throughout the layout phase of the FFT processor we noted considerable discrepancies between predicted delay information obtained at pre-layout state and corresponding post-layout measurements. This
divergence was basically founded on two tool limitations: misleading pre-layout performance estimations and unfavorable cell placement. As a result, several iterations were required to achieve the targeted “speed” characteristic.

However, in order to enable higher clock rates and shorter design cycles, provision of reliable timing-driven place and route is a must. This requirement particularly applies to clock trees. As discussed in section 8.2.4, we had to manually replace clock drivers at layout-level. This is not only a very time-consuming task, but is also prone to newly-induced errors. As a result, automatic “balancing” of the clock tree must be definitively supported by the design-tool. Appropriate balancing techniques are discussed in [Téllez 97].

10.1.2 Considerations on 'VHDL Design'

Originally developed for design specification, VHDL* allows for hierarchical and modular description of arbitrary architecture types.

In this decade, several VHDL simulators enabling the designer to test design functionality from a beginning behavioral- to a final gate-level description evolved. In addition, 'design synthesizers' emerged, generating gate-level netlists from a given VHDL description. VHDL design thus allows for convenient porting of a given design-description to a new target technology.

Hence, in order to achieve a portable processor design, we described all processor components by means of VHDL. However, even in consideration of the immense advantages brought by VHDL design, several further improvements are desirable.

Large Semantical Overhead

Concerning the actual VHDL language, underlying grammar allows for large flexibility in design description. On the other hand, this flexibility is paid for by a considerable semantical overhead. Particularly for small design components (e.g., pipeline stage), a corresponding VHDL description is mostly dominated by the required grammatical skeleton rather than by actual design information.

*VHSIC Hardware Description Language
10.2. MCM Implementation

“Unreliable” Timing Information

In addition, for most CMOS technologies no self-contained VHDL design-flow exists. This is particularly the case for standard cell technologies for which no corresponding VHDL library is available. But even if available, VHDL libraries are often characterized by imprecise timing specifications and generally do not consider any kind of post-layout information, e.g., interconnect delays. No reliable timing information can then be obtained by means of VHDL simulations. As a result, further timing simulations have to be performed based on a “recommended” simulator, thereby doubling the simulation effort.

“Macro Cells”

A further problem is given by the fact that some components, such as memories or multipliers, still do not apply to VHDL-based synthesis. This is partly due to the fact that only a restricted sub-set of the VHDL language is supported by existing synthesizers. Apart from that, efficient synthesis of macro cells, such as memory elements or arithmetic components (adders, multipliers) is still a challenging problem. As a result, several design steps must be “manually” performed outside the desired VHDL design-flow, thus leading to a restricted portability of the underlying design.

To conclude, reliable timing characteristics obtained by VHDL simulations are a basic requirement to simplifying and shortening the overall design process. Also, in order to allow for a self-contained and completely automated design-flow, features for efficient macro cell generation must be provided.

10.2 MCM Implementation

A further interesting realization alternative is given by the emerging field of MCM\(^\dag\) implementation techniques. With this method, multiple dies are directly mounted on a common substrate, thereby avoiding area consumptive packaging as well as corresponding pin- and PCB wire-delays.

\(^\dag\) Multi-Chip Module
10.2.1 Off-Chip LM Implementation

Concerning the SPITFFIRE processor, a first potential application of MCM design is motivated by off-chip implementation of the local memory. A considerable reduction of die size is thus achieved (at least 40%), allowing for higher yield rates. As demonstrated by Analog-Devices, production costs can thus be significantly reduced. In addition, referring to external LM implementation, memory-size can be more conveniently increased (i.e., without any decrease in expected processor yield) to allow for processing of larger sequence lengths (e.g., 2048, 4096, etc.). Looking at other processor designs, the concept of MCM implementation of processor core and cache-memory was also applied by Intel for the Pentium-Pro processor.

Hence we must now address the problems concerned with external LM implementation. The most obvious problems associated with an external LM realization are 'pin count' and 'access time'.

Pin Count

Due to the fact that the LM is connected to the APU by a write- and read-port, corresponding signals must be transferred via I/O pins in the case of an external LM implementation. Referring to the PE's arithmetic concept \((2 \times 32b + 8b)\), the total number of pins associated with the LM thus increases from 51 to 166 for the off-chip case. Facing this increased pin count, the resulting layout is clearly pad-limited and hence we cannot expect any significant area reduction. However, referring to MCM implementation, we can overcome this problem by employing enhanced I/O techniques, e.g., multiple concentric pad-frames or area I/O.

Access Time

The second problem associated with external LM implementation is given by a considerable increase in access time. This is due to additional capacitive loads imposed by the involved I/O pins and corresponding wiring and assembly techniques.

One commonly applied approach to speeding-up access time is to employ 'memory interleaving'. With this method, total memory space is divided into multiple partitions which are addressed in an interleaved

\(^4\)For the SHARC-processor, unit price dropped from over $300 to merely $10.
manner. In the ideal case, all access requests are overlapping and overall access time for successive data accesses is reduced to $T_{AP}/N_P$, with $T_{AP}$ and $N_P$ denoting the access time of a single partition and total number of partitions, respectively. For this, however, an appropriate data-distribution on all memory partitions must exist. If the underlying address sequence requires some data to be redundantly kept in multiple memory partitions to allow for fast data access, area requirements increase and mechanisms must be provided to maintain data consistency. Referring to the FFT algorithm, we have proven that an appropriate data-distribution scheme exists in the case of four-partition memory interleaving. Any decline in memory access characteristics can thus be prevented for the case of off-chip LM implementation.

In addition, we can employ dedicated caching-strategies (cf. [Baas 98]) to improve access characteristics for the on-chip arithmetic data path architecture.

### 10.2.2 Multiprocessor MCM (System on an MCM)

In addition, targeting a very compact multiprocessor system, a set of two, four or more SPITFFIRE processor cores can be implemented on one MCM together with corresponding system memory. Such a “system on an MCM” approach has been successfully realized for a complete Pentium-based PC motherboard [Hirt 97].

Because of its complexity in terms of system aspects, MCM implementation of the real-time object detection system proposed in section 3.3 is comprehensively discussed in [Cavadini 99].

To conclude, referring to the continuing trend to further miniaturization, improvements to MCM and monolithic VLSI implementation will allow us to realize the object detection system on an area of $10 \times 10 \text{mm}^2$ within the next decade. This would allow for a very close implementation of image sensor (e.g., CCD) and object localizing system (cf. “smart-camera”).

### 10.3 Hardware Extensions

Concerning the realized processor design, several extensions can be made in order to increase flexibility and versatility.
10.3.1 Programmable Processor Control

At present, the command set of the SPITFFIRE processor is fixed, i.e., the sequence of control events required to perform each of the implemented BST low-level tasks is predefined by corresponding state-machines.

While this approach allowed for 'lean-controlling', reduced test-efforts and shorter design times, versatility is clearly restricted. Unfortunately, the processor's high-performance complex arithmetic unit can then not be employed for other demanding algorithms, such as computation of complex scalar products or matrix-matrix multiplication.

Hence, in order to increase the processor's versatility, we have to implement a programmable control unit. Along with this, we must also provide on-chip instruction memory and corresponding down-load mechanisms.

10.3.2 Space Domain NCCF Computation

Although operation count is significantly reduced by performing the cross-correlation in the Fourier domain, the option of employing local pre-processing steps (e.g., filtering, elimination of local mean value, etc.) on an isolated search window is prevented. Each filter operation performed on the search area's spectrum is of global effect. The same is true of any brightness correction, e.g., elimination of the inherent DC component.

If such pre-processing steps are required, space domain NCCF computation is inevitable. Since underlying UCCF algorithm flow only consists of accumulations and multiplications, it can basically be computed on our dedicated data path architecture. For this, we merely have to implement an additional feed-back path from the multiplier to one of the two registerfile blocks.

To perform the required multiplications, we can utilize one of the CMUL's real-arithmetic multipliers. However, referring to the 8b data-format of the search area, the 32×32 multiplier would be poorly utilized. To overcome this problem, we can "split" one 32 × 32 multiplier into four 8×8 partitions hereby performing four "smaller" multiplications at the same time. The design of the corresponding 32 × 32 multiplier must be appropriately modified for this. This partition approach is commonly referred to as 'split ALU' design and can be found in several designs dealing with multiple data formats (cf. the MMX technology embedded into INTEL Pentium processors [Peleg 96]).
Finally, concerning the required masking of irrelevant pixel-information (cf. section 2.1.3), the option of data-dependent accumulation must be provided.

10.3.3 Computation of Invariant Moments

As discussed in section 1.2, invariant moments (MI) are a promising approach to realizing rotation-invariant object detection [Belkasim 91]. For this method, 2D moments $m_{pq}$ are basically computed to a similarity measure.

Looking at the computation procedure of underlying $m_{pq}$ (Eq. 10.1), pixel information $f(x, y)$ is multiplied by corresponding weights $x^p$ and $y^q$. Pixel coordinates $x$ and $y$ must thus be raised to the power of $p$ and $q$, respectively. While this seems to be a very demanding arithmetic operation at first glance, we can conveniently compute the required weights by means of cascaded accumulations. Presupposing appropriate flexibility in data-flow, invariant moments can thus be generally computed on our data path architecture by means of the CADDs.

$$m_{pq} = \sum_{x=0}^{N} \sum_{y=0}^{N} x^p y^q f(x, y), \quad p + q \in \{2, 3\} \quad (10.1)$$

Although corresponding operation count is comparably large, approximately $(N - n)^2 \cdot (9n^2 + 50)$ operations are required to locate an $n \times n$-template in an $N \times N$ search area, the spectrum of potential object detection applications is significantly enlarged by additional implementation of the MI algorithm.

10.4 Additional Applications Requiring Fast FFT

The SPITFFIRE processor is basically optimized to efficiently support Fourier domain correlation of real-valued data. According to the similarity between correlation and convolution, the new processor thus also offers a strong speed-up potential for all kinds of related FIR-filter implementations.

Apart from this, a large variety of additional algorithms also requiring efficient FFT computation exist. Since our FFT architecture
disposes of excellent speed and precision characteristics, it can be ba-
sically employed for each kind of FFT-based application. Typical rep-
resentatives are given by the emerging fields of medical imaging, image
restoration, spread spectrum communications, speech and audio pro-
cessing, echo cancelation as well as radar and sonar applications.

To state relevant examples, we give a brief overview of FFT-requiring
methods for rotation-invariant object detection and on synthetic aper-
ture radar.

10.4.1 Rotation-Invariant Object Detection

Falling back on rotation-invariant object detection (section 1.2), several
algorithms founded on the Fourier transform exist.

Log-Polar Transform

A promising FFT-based approach allowing for invariant object detection
is given by the log-polar algorithm proposed in [Cideciyan 95]. Compu-
ting the Fourier magnitude of an image removes the effect of translation
and converts rotation and uniform scalings into independent shifts in
orthogonal directions. Hence, by further mapping the Fourier magni-
tudes into polar coordinates \((r, \theta)\), decoupling of the rotation and scale
factors is achieved. Rotation angle and scale factor can then be obtained
by computing the (circular) cross-correlation function of the log-polar
transformed Fourier magnitudes of template and isolated search win-
dow.

Radon Transform

Secondly, as stated in [Al-Shaykh 96], the Radon transform can be uti-
lized to achieve features invariant to translation, rotation and scaling.
The Radon transform can be optically implemented, but optionally,
FFT-based computation of the Radon transform is possible [Shukla 92].

Invariants Defined by Higher Order Spectra

Another approach to obtaining invariant features is proposed in
[Chandran 97]. Here, functions are defined on higher order spectra serv-
ing for pattern recognition. Higher order spectra can be computed by
the FFT as products of Fourier coefficients and provide high noise im-
munity.
10.4.2 Synthetic Aperture Radar (SAR)

Today, synthetic aperture radar is employed for a variety of monitoring and surveillance tasks. Referring to the underlying task-flow, the main problem envisioned by a corresponding real-time system is the requirement for fast computation of the 2D-FFT. However, up to now, no appropriate processor allowing for precise and fast FFT transformation has existed. As a result, several approaches have evolved to avoid the original need for fast Fourier transformation [Wang 97, Webb 98].

Looking at the architectural concept of the SPITFFIRE processor, it ideally matches the FFT requirements imposed by SAR. A compact realization of the "native" SAR algorithm can thus be achieved by means of our new FFT architecture.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Hiasat 98]</td>
<td>Ahmad A. Hiasat &amp; Hoda S. Abdel-Aty-Zohdy. <em>Residue-to-Binary Arithmetic Converter for the Moduli Set ( (2^k, 2^k - 1, 2^{k-1} - 1) ).</em> IEEE Trans. on Circuits...</td>
</tr>
</tbody>
</table>


[Sadovnik 93] L. Sadovnik, A. Sawchuk, T. Lu & A. Kostrzewski. *Scale, Translation and Rotation Invariant All-Optical Processor Based on Intensity-to-Phase Coding*. SPIE,


[Trivedi 81] M. Trivedi, K. V. Lever & C. Eng. Measured Performance of Block-Floating-Point Hardware FFT Proces-


Curriculum Vitae

Date of Birth: 11 April 1968
Place of Birth: Alfeld (Leine), Germany

EDUCATION

1974-1978  Primary School, Duingen.
1978-1987  'Gymnasium Alfeld'.
June 1987  "Abitur" (German A-level diploma for general university qualification). Majors in mathematics and physics.
1987-1988  National service (German Airforce).
1994      "Dipl.-Ing." degree (M.S.) from the University of Han¬nover.
1994-1999  Electronics Laboratory, ETH Zurich.
  Teaching and research assistant, 'Image Processing Group'.