International Journal of Electronics
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tetn20

Implementation of comprehensive address generator for digital signal processor

Ramesh M. Kini [a] & Sumam S. David [a]
[a] Department of ECE, National Institute of Technology Karnataka, Surathkal, Karnataka, India
Version of record first published: 26 Sep 2012.

To cite this article: Ramesh M. Kini & Sumam S. David (2012): Implementation of comprehensive address generator for digital signal processor, International Journal of Electronics, DOI:10.1080/00207217.2012.713009

To link to this article: http://dx.doi.org/10.1080/00207217.2012.713009

International Journal of Electronics
2012, 1–18, iFirst

Implementation of comprehensive address generator for digital signal processor

Ramesh M. Kini* and Sumam S. David

Department of ECE, National Institute of Technology Karnataka, Surathkal, Karnataka, India

(*Received 7 June 2010; final version received 1 May 2012*)

The performance of signal-processing algorithms implemented in hardware depends on the efficiency of datapath, memory speed and address computation. Pattern of data access in signal-processing applications is complex and it is desirable to execute the innermost loop of a kernel in a single-clock cycle. This necessitates the generation of typically three addresses per clock: two addresses for data sample/coefficient and one for the storage of processed data. Most of the Reconfigurable Processors, designed for multimedia, focus on mapping the multimedia applications written in a high-level language directly on to the reconfigurable fabric, implying the use of same datapath resources for kernel processing and address generation. This results in inconsistent and non-optimal use of finite datapath resources. Presence of a set of dedicated, efficient Address Generator Units (AGUs) helps in better utilisation of the datapath elements by using them only for *kernel operations*; and will certainly enhance the performance. This article focuses on the design and application-specific integrated circuit implementation of address generators for complex addressing modes required by multimedia signal-processing kernels. A novel algorithm and hardware for AGU is developed for accessing data and coefficients in a bit-reversed order for fast Fourier transform kernel spanning over $\log_2 N$ stages, AGUs for zig-zag-ordered data access for entropy coding after Discrete Cosine Transform (DCT), convolution kernels with stored/streaming data, accessing data for motion estimation using the block-matching technique and other conventional addressing modes. When mapped to hardware, they scale linearly in gate complexity with increase in the size.

Keywords: address generation; bit-reversed address; convolution; digital signal-processing kernel; dynamically reconfigurable datapath; fast Fourier transform; finite impulse response filters; infinite impulse response filters; motion estimation; zig-zag address generation

1. Introduction

Offline data processing or stream data processing of a large number of data points, as in the case of multimedia applications, requires data to be accessed from memory at high rates. The sequence of this data access is non-linear and complex, varying with the type of signal-processing kernel that is being executed. It is not possible to generate these sequences of addresses with simple Address Generator Units (AGUs) at 3 addresses per clock. Use of kernel execution datapath for address computation leads to ineffective utilisation of resources. Hence, it is desirable to have a dedicated

*Corresponding author. Email: rameshkinim@gmail.com

ISSN 0020–7217 print/ISSN 1362–3060 online

http://dx.doi.org/10.1080/00207217.2012.713009
http://www.tandfonline.com

Comprehensive AGU (CAGU). This article deals with the design and Application-Specific Integrated Circuit (ASIC) implementation of such a CAGU.

The three major goals that have driven the Digital Signal Processsor (DSP) implementation are data parallelism, application-specific specialisation and functional flexibility. These have led to a variety of hardware implementations (Tessier and Burleson 2000). Performance of a signal-processing system can be measured using parameters like speed of execution or throughput, energy consumed in performing the task, flexibility in terms of programmability or modifying the system to perform any other function and cost of ownership.

Signal-processing kernels like Fast Fourier Transform (FFT), DCT, Finite Impulse Response (FIR) filters are good examples for functions. A transform or an object primitive can be thought of as a function with a state associated with it, for example an Infinite Impulse Response (IIR) filter. If the transform like IIR filter operation is performed on a block of data and this operation on each block of data is made independent of the other block, i.e. the initial state required for the processing of each data block is generated within itself or can be provided as a part of the input, then each of these transforms can be considered as a function and the sequential or parallel invocation of these objects do not effect each other. Compute models, selection of a suitable computing model, development of implementation strategy, functions, transforms, various kinds of dataflow and system architectures are described by DeHon (2008).

A lot of research has taken place to study the various aspects of reconfigurable devices, like reconfigurability, synthesis, placement, routing strategies, configuration loading and management. These concepts have matured to a great extent and in turn reconfigurable devices have improved by incorporating this knowledge (DeHon 2008). Various computing methods, their comparison, hardware implementation issues, types of coupling in a reconfigurable system, software issues, partitioning between hardware and software, context memories and reconfiguration issues have been dealt in Compton and Hauck (2002).

Computational datapaths have regular structures. Coarse-grained architectures can provide configurable algebraic operators with word lengths equal to data word. Such structures are very area efficient (Hartenstein 2001). In coarse-grained reconfigurable structures, the datapath is created using programmable interconnects between hardwired datapath elements and operators.

The concept of Reconfigurable Processor (RP) comes from the idea of having a General-Purpose Processor (GPP) the coupled with some reconfigurable resources that allow the execution of custom application-specific instructions. Dynamically RP achieves higher speed of computation with lower cost of silicon area for computation intensive applications in domains like multimedia.

The mapping of an entire application on to a reconfigurable platform may result in the sub-optimal use of reconfigurable resources as large amount of infrequently used code will occupy the precious reconfigurable resource. Hence, mapping of only the most often used kernels to a reconfigurable resource and executing the non-kernel code part of the application on a GPP will be effective.

Section 2 deals with the design issues that highlights the need for a CAGU and provides an overview of the setup in which the designed CAGU can be used. Section 3 describes the algorithms and hardware developed for AGUs supporting data/coefficient fetch and result update. Details of integration of the individual AGUs into a CAGU is also discussed. Section 4 gives the details of the ASIC implementation of the CAGU and the

Dynamically Reconfigurable Data-Path (DRDP). It also explains the testing procedure of the fabricated CAGU chip. Section 5 discusses the timing details of the execution of the three DSP kernels, namely FFT, convolution and Sum of Absolute Difference (SAD) computation for Motion Estimation (ME).

2. Overview of the proposed dynamically RP

RPs find extensive applications in networking and multimedia domains. Multimedia applications, like Audio/Video Encoders/Decoders, Transcoders, Fast Fourier Transform (FFT) in Orthogonal Frequency Division Multiplexing (OFDM) used in Software-Defined-Radios, can exploit the features of an RP to minimise the hardware requirements and at the same time improve the throughput.

The proposed processor has an array of four dynamically Reconfigurable Functional Units (RFU) sharing a common memory. The array of RFUs and the memory are controlled by an Application-Specific Instruction set Processor (ASIP). The RFU array with a control processor can also be mapped onto an FPGA-based embedded-system for multimedia applications.

The overall operation of the proposed processor can be summarised as follows.

Upon receiving a service request for processing/executing a kernel operation, the ASIP controller facilitates the storage of data to be processed in the appropriate memory block, initialises a free RFU with appropriate control settings, schedules the kernel on that RFU. Upon completion of the kernel operation, it frees the RFU. Basically, the ASIP coordinates the job of kernel execution.

Perceived advantages of the architecture are:

(1) The innermost loop of a kernel can be executed in a one-clock cycle, and hence executes faster than a GPP or a DSP processor.
(2) Since the kernel itself is implemented as a micro-program, there is no need for the code (program) memory and hence no need for the instruction fetch and decode. Therefore, there will be apparently savings in memory requirement and power.
(3) The application can be written in terms of function calls. These function calls can be mapped to kernels executed on the RFU by the ASIP. The application program tends to be compact.

The Dynamically Reconfigurable Data Path (DRDP) under discussion supports signed integer and fixed point arithmetic. The output of any of these functional units can be routed to at least one input of all the functional units. The DRDP unit will have three input ports and an output port, other than two memory read ports and a memory write port. The schematic representation of a DRDP is shown in Figure 1(a). The configuration of a datapath is defined by a control word that is stored in a configuration memory.

The DRDPs can be cascaded by interconnecting the Input/Outputs (IO) appropriately to form a complex datapath with kernel operations spread over multiple DRDPs in a chained or pipelined fashion. The processor design is targeted at the acceleration of execution of these often used kernels under the control of the ASIP. DRDP consists of an optimal number of adders (adder/subtractor), multipliers, comparators, barrel-shifters for normalisation/de-normalisation, register-files for temporary storage, as shown in Table 1. The RFU contains a DRDP, 3 AGUs for the generation of memory addresses and a configuration memory with multiple contexts, as shown in Figure 1(b).

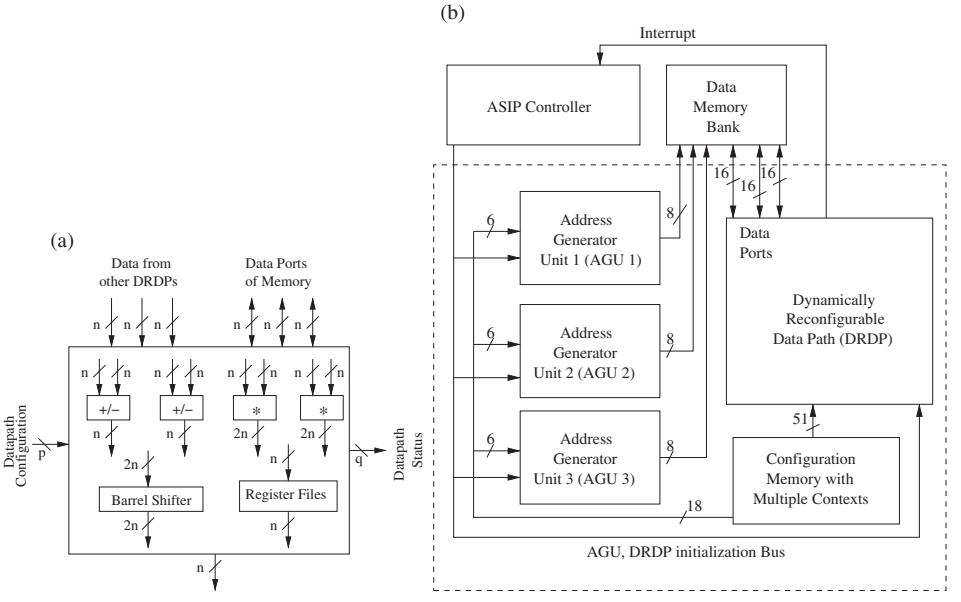

Figure 1. Block schematic diagram of reconfigurable functional unit with its components: (a) proposed dynamically reconfigurable datapath unit (RDPU); (b) reconfigurable functional unit.

Table 1. Components used in the RFU.

Name of the component used	Number of instances
Adder/subtractor	2
Multiplier	2
Barrel shifter	2
Comparator	1
Register files (each having 4 registers)	4
AGU	3

Given a function (written using a subset of 'C' language) and a set of resource constraints, 'SPARK' – a high-level synthesis tool (Gupta et al. 2004a,b) generates control/data flow graphs of the function, Very-high-speed-integrated-circuit Hardware Description Language (VHDL) code for the synthesis and equivalent 'C' code for the verification purpose among others. The type and the number of functional units to be used in the datapath have been optimised for the selected kernels. The goal of executing the innermost loop of any of the selected kernel in the one-clock cycle with few finite arithmetic units is satisfied if address update units are separated from datapath of the kernel. This leads to the use of dedicated AGUs. This observation is also supported by Lysecky, Stitt, and Vahid (2006) that, in the case of application implementation through the high-level synthesis, the inclusion of AGU minimises the circuit complexity and helps achieving higher speeds of execution.

HDL code in the structured style has been developed and tested for each of the address update functions required by data/coefficient fetch and result-write for each kernel,

such that one address in the required sequence is generated in every clock cycle. Common functional units used in the address generation process for these kernels were identified. A CAGU was then developed, which can support various addressing modes with a shared datapath elements and control structure.

Though coarse-grain reconfigurable architectures provide significant speedups, the ability to compile from imperative high-level languages like 'C', Matlab® and Java® to achieve noticeable speedup is not proved due to reasons like reduced focus on compilation phase and mapping computational structures effectively on reconfigurable architectures (Cardoso 2004). Mapping of the innermost loop to a DRDP coprocessor is discussed in Huang and Malik (2002). The system described in this article can map a high-level DSP kernel with recurring loops (like *N*-point FFT, FIR filter implementations) onto the DRDP for execution. The CAGU supports address generation across multiple levels of recursive loops for these kernels without changing the datapath.

Most of the RP architectures developed till now need a complex specialised compiler to compile the application or high-level synthesis of the application to map the application on to the respective processor. This research targets at increasing the speedup by designing a coarse-grained architecture that aids in mapping the multimedia applications more effectively to the proposed architecture. The multimedia applications can be written in any imperative high-level language using two kinds of function calls. The first type is the basic multimedia file header processing and system IO handling, which are processed by the controller processor. The second type being the DSP kernels that can be mapped directly to the RFUs and can be executed in an accelerated fashion. Thus eliminating the need for resynthesis, special efforts required in writing the compiler and effective exploration of design space to suit the reconfigurable hardware. The interaction between the controller–processor and the RFUs are described elsewhere in this article.

The Montium Tile Processor has an AGU associated with each memory block in it (Smit et al. 2007). This AGU can only generate memory access patterns like increment, decrement and bit-reversed as available in typical DSP processors. For other complicated functions it depends on the Look-Up-Table approach. It is important to note that Montium Tile Processor's AGU supports only one loop and between loops user code intervention is needed. In short, the addressing modes do not support the address generation for the execution of the entire kernel without any coded manipulation in-between loops.

A set of dedicated, efficient AGUs will definitely enhance the performance. Next section discusses the AGU in detail.

3. Comprehensive address generator for dynamically RPs for DSP kernels

In signal-processing applications, data access can be characterised as indexed access of vectors stored in memory, where indices are referred to as pointers in higher level languages. These indices are data dependent and the sequence of access depends on the kernel operation being performed on the data. *Stride* can be defined as the distance between the addresses of consecutively accessed vectors in the memory. The strides could be linear or non-linear, and can be characterised by an algebraic equation. Thus, the address of the next location to be accessed can be expressed as an algebraic expression in terms of the current address and can be viewed as an addition of a modifier to the current address. So, the address generation process can be seen as a sequence of algebraic

operations performed on the current address. This algebraic expression can be synthesised into an AGU using arithmetic operators and a controller.

Extracting the Address Expression (AE), applying high-level optimising methods like AE splitting/clustering, induction variable analysis, target architecture selection and global-scope algebraic optimisation are explored in Miranda, Catthoor, Janssen, and De Man (1998). It also aims at the reduction of cost of time-multiplexed address unit at the system level. This approach is more suited for the high-level synthesis of typical multimedia applications.

In the case of a signal-processing ASIC, AGU caters to the generation of a pre-determined type of the sequence of addresses. In the case of GPP, the AGU may support a few simple sequence types or addressing modes. A Programmable Digital Signal Processor (PDSP) will have dedicated AGUs that support few additional addressing modes like bit-reversed and circular. Also, AGUs do not use the datapath resources leading to the concurrency of address generation and datapath operations. If any other sequence is required, then it needs to be generated by the execution of processor code and the use of datapath computational units, resulting in a non-optimal use of resources like datapath elements and time. Signal-processing applications require varied types of addressing modes. For a given application, each type of data access may need a different type of addressing mode and this depends on the architecture of the processing system, datapath, type of memory used and the way data are stored in the memory. For example, FFT computation needs the data to be fetched in a bit-reversed order and the coefficients in a linear fashion, as well as the result to be written back in a bit-reversed order. FIR filter implementation of stored data requires the data to be fetched in a linear fashion, but the same filter-processing streaming data needs to fetch the data from a circular buffer and needs circular addressing mode.

Most of the signal-processing applications can be written in terms of DSP kernels and the kernels themselves could be function calls. To achieve speedy execution, the execution of kernels may be offloaded to a hardware with a fixed or reconfigurable datapath under the control of a supervisor processor. The job of the supervisor processor is to schedule the kernels on the hardware, initialise the hardware with data pointers and other kernel control variables. The interaction between the hardware and the controller could be through interrupts, DMA and the setting of control words. Such a hardware could be a coprocessor or a reconfigurable datapath hardware capable of executing various kernels, having the ability to switch between specified kernel operations by reconfiguring its datapath with a simple switching of microprogram memory banks. This hardware accelerator unit may be capable of accessing the data from a virtual memory for processing and result-storing purposes. The datapath may be pipelined and the memory being accessed could be multiported. High throughput demands high memory bandwidth and puts very high constraints on timing budget for address generation. Hence, it is desirable to have multiple reconfigurable AGUs capable of concurrent generation of various special address sequences required by a variety of DSP kernels, with each of the reconfigurable datapath. For example, one AGU for each data fetch, coefficient fetch and result write operations. The reconfigurable AGU may have its own local microprogram or a finite state machine for generating sequences of addresses and may function under the supervisory control of a microprogram for that kernel.

Some of the most often used addressing sequences are *Sequential*, *Sequential with offset*, *Shuffled*, *Bit-reversed*, *Reflected*. Hulina, Coraor, Kurian, and John (1995) discusses the implementation of a coprocessor for the generation of these address sequences and

provides the host processor with a few additional special addressing modes defined by signal-processing algorithms, without any change in the host processor's instruction set architecture or the external memory.

Efficient generation of address sequences like zig-zag for DCT, sequence of addresses to fetch the twiddle-factors in FFT operation and sequence of addresses to fetch the data in convolution operation are essential for multimedia processing. Address sequence generation for kernel operations on both stored data and streaming data are also necessary.

This article describes an address generation unit suitable for a DRDP processor, capable of generating one address per clock cycle in a required sequence and can be synchronised with the datapath operations using the *Address Generate Enable* signal. The following address sequences suitable for multimedia applications are supported:

- Bit-Reversed – for data fetch and data store in the case of a complete N-point FFT kernel.
- Fetching twiddle-factors for a complete N-point FFT kernel.
- Data fetch operation for a convolution kernel (stored data – any values of N and M).
- Data fetch operation for a convolution kernel (streaming data – any values of N and M).
- Impulse response coefficient fetch operation for a convolution kernel (stored/streaming data) – *Modulo M* (circular) addressing mode.
- Result store operation for a convolution kernel (stored/streaming data) – Divide by N addressing mode.
- Data fetch operation for a linear-phase FIR filter (streaming data).
- Data fetch operation from macro-blocks for ME kernel.
- Zig-zag – suitable for fetching data for entropy coding.
- Other modes like increment and decrement.

Hardware for each of these addressing modes was developed separately and later a CAGU that can generate all the address sequences listed above was designed. The AGU developed is tailored to work with a DRDP, though the concept can be used with any datapath unit with appropriate synchronisation. As an example of application of this CAGU, consider an array of RP datapaths that support the execution of Single Kernel Multiple Data (SKMD), implying concurrent fetch of data from multiple databanks/data-streams for executing the same kernel operation on these data. A set of CAGUs pertaining to data access for a specific kernel can be configured by selecting suitable addressing modes and may be used to generate address sequences required by multiple datapaths.

AGU and DRDP are designed as a parameterisable word length and a address size using a hardware description language in fully structured style of coding. Thus the hardware synthesised will be identical to the description in the code. AGU and the DRDP provide the necessary status signals back to the controller and are controllable by a micro-programmed controller. Hence, reconfigurability is guaranteed with just a change of the micro-program.

Our earlier paper (Kini and Sumam 2009) reports AGUs developed for data access, twiddle-factor fetch suitable for a complete N-point FFT kernel and implementation of N-point FFT kernel using Dynamically Reconfigurable Data Path (DRDP). Design and implementation of AGUs for data and coefficient fetch for a complete convolution kernel

and AGU for accessing data from $N \times N$ pixel array in a zig-zag order used in entropy coding after DCT are reported in Kini and Sumam (2009).

3.1. *Address generation for N-point FFT kernel*

FFT kernel can be executed using a single DRDP in a folded manner or can be executed using four DRDPs in a chained manner. A typical datapath that implements a single *Butterfly* operation in Decimation-In-Frequency (DIF) is given in Kini and Sumam (2009) and on the same lines datapath for a Decimation-In-Time (DIT) scheme is also implemented.

In each of these datapaths four DRDP units have been cascaded and configured to form a single datapath capable of performing one butterfly operation for every two clock cycles. So, an N-point FFT operation is completed in $(N\log_2(N) + 4)$ clock cycles, where a constant of four clock cycles corresponds to initialisation of the DRDP and write back latency of the last butterfly result. The setup assumes that the twiddle-factors are precomputed and saved in memory.

3.1.1. *Bit-reversed address generation for N-point FFT*

Many algorithms to compute bit-reversed address are available in the literature. Many of them are best suited for coding using high-level languages on microprocessor or DSP (Evans 1989; Rodriguez 1989; Walker 1990; Yong 1991). These algorithms can be classified as those based on heuristics (Yong 1991) and algorithms using Seed-Table (Walker 1990). Address generation using these methods have a long delay as compared to the datapath latency and the memory access delay.

Hardware AGUs have been developed for array processors (Nwachukwu 1985). Nwachukwu (1985) and Hulina (1995) implement the bit-reversed address generation using counter-multiplexer method. The counter-multiplexer method can generate a variety of patterns but as the number of addresses increases, the area increases exponentially, also resulting in the increase in power dissipation and leakage.

The method proposed in our paper (Kini and Sumam 2009) can generate a sequence of addresses suitable for many multimedia algorithms and uses adders, shifters, counters in the datapath and very few gates for the simple control logic. This translates to a linear increase in transistor count with the increase in the number of address bits unlike the counter-multiplexer method. Banerjee, Dhar, and Banerjee (2001) describe an algorithm for address generation for data access for an N-point FFT. The hardware developed for implementing the algorithm uses three loadable down counters and allied control circuit. Hardware developed by us for the same functionality needs only two shift registers and allied control circuit, as depicted in the Figure 2(a). The shifters hold data patterns like '11..1100..00' and '00..00100..00', and are shifted once after the completion of each stage of FFT. Only two bits toggle in each of the shifters as compared to multiple bits toggling in each of the counters after every address generation as in Banerjee et al. (2001).

The input or the output samples need to be re-ordered in a bit-reverse fashion a depending on whether DIT or DIF approach is employed. This reordering is done by exchanging data in memory locations pointed by pairs of addresses generated by bit-reversed address generator for the first stage. For the actual exchange, the data elements of the pair are fetched from memory, stored in registers of the DRDP and written

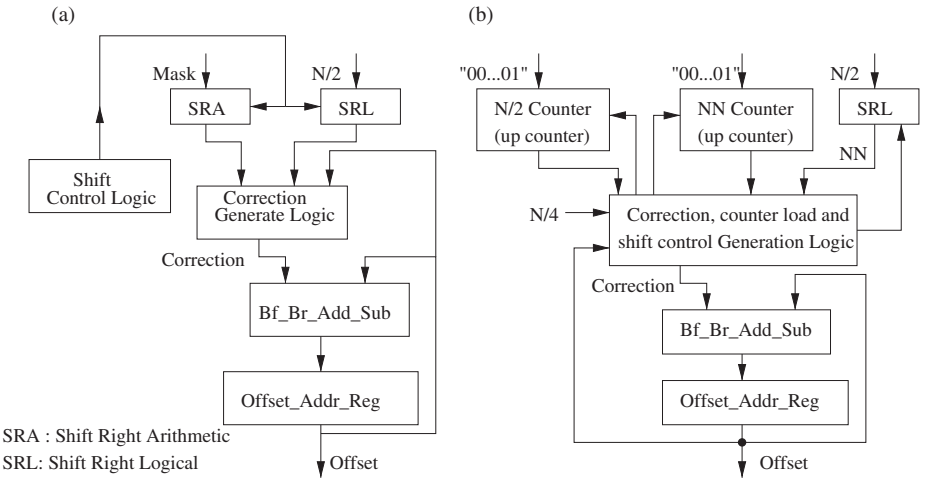

Figure 2. Hardware Schematic of AGUs used by FFT kernel: (a) Hardware schematic representation of bit-reversed address generator; (b) Hardware schematic representation of FFT twiddle-factor address generator.

back in exchanged order from registers. For fetching the data and writing it back, we use two AGUs appropriately synchronised.

Discussion on reducing the number of memory accesses for mapping the data array in the bit-reversed order, avoiding self-reversed binary address patterns is given in Harley and Maheshwaramurthy (2004). This concept is useful in reducing the memory accesses required during pre- or post-processing of data in terms of reordering the data samples while computing FFT using DIT or DIF schemes. This is realised by comparing the bit-reversed address generated by AGU with an internal linear up counter. If the counters match, exchange is not required. Hence, no memory read or write operations are performed and the AGU will continue to generate the next address in the sequence.

3.1.2. *Address generation for accessing twiddle-factors for N-point FFT*

For an FFT butterfly operation, a pair of data operands are needed – one with a bit-reverse order address and the other is a twiddle-factor. An algorithm for generating the sequence of addresses for fetching twiddle-factors for any N-point FFT with $\log_2 N$ stages has been developed. The respective hardware has been designed, simulated, tested and the block schematic diagram is shown in Figure 2(b).

3.2. *Address generators for convolution kernel*

Convolution approach is used in implementing digital filters like FIR filter. When an input sequence of length N is convolved with an impulse response of length M, the output sequence is of length $N+M-1$. The address generation scheme assumes that the given data is padded with $M-1$ zeroes at both ends. The sample points of impulse response are stored in a reverse order in the memory.

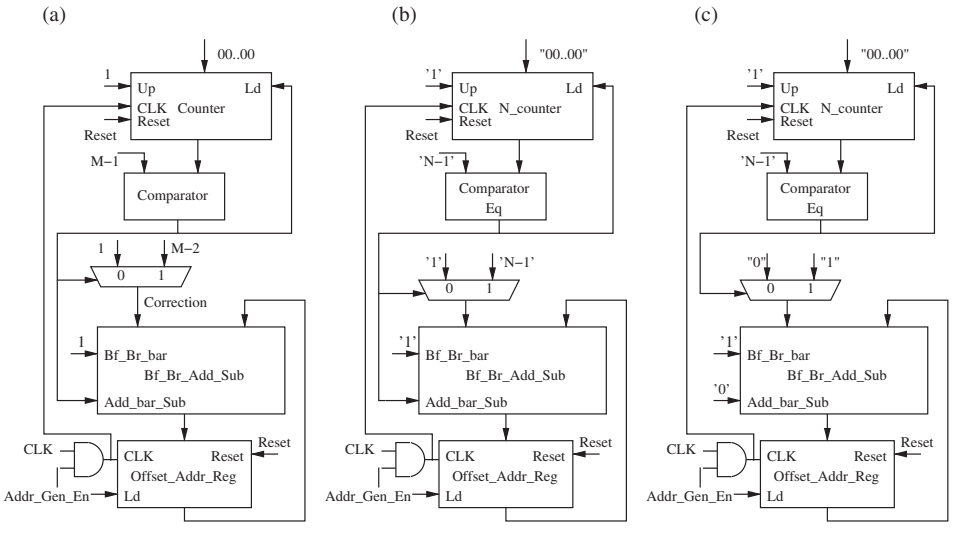

Figure 3. Hardware schematic representation of AGU used by convolution kernel: (a) Data fetch – stored data, (b) coefficient fetch, (c) storage of result.

The sequence of addresses for fetching coefficients follows a *Modulo M* pattern and that for writing the convolution result *Divide by M*. The convolution kernel with N data points and M impulse response points is executed in $((N+M-1)\times N+3)$ clock cycles, where three clock cycles correspond to the initialisation and write latency of the last result.

3.2.1. *Address generator for fetching data for convolution – stored data and streaming data*

The convolution operation may be performed on stored data or streaming data. Each of these operations need a different type of AGU. The AGU suitable for both applications have been developed. The algorithm for fetching the coefficients will remain the same whether the kernel is working on stored or streaming data. The AGU used for storing the convolved data will use linear or circular addressing mode depending on whether the kernel is working on the stored or streaming input.

The hardware schematic diagram of the AGU for data fetch in the case of stored data is shown in Figure 3(a). In the case of streaming data, the data samples are stored in a circular buffer. Schematic representation of the AGU hardware is as shown in Figure 4(a) and the algorithm for generating the address for fetching streaming data can be summarised as shown in Figure 4(b).

3.2.2. *Address generator for accessing filter coefficients in convolution*

The algorithm for generating the addresses for fetching coefficients is of *Modulo N* type and is similar to that of data fetch for convolution (stored data). Whenever the counter reaches a value of $N-1$, the correction is $-(N-1)$, as shown in Figure 3(b).

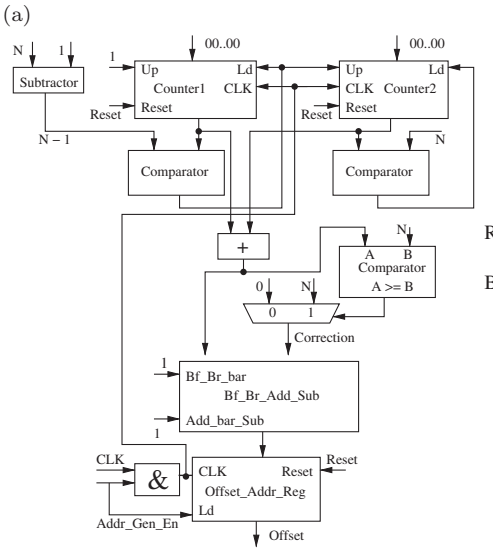

Figure 4. (a) Hardware schematic representation and (b) algorithm for data fetch AGU for convolution kernel – streaming data.

3.2.3. *Address generator for accessing convolved data writeback*

The algorithm for generating the addresses for result storage is of Divide by 'N' type and is similar to that of coefficient fetch for convolution. When the counter value is less than $N-1$, the correction is 0 and when the counter is equal to $N-1$, the correction is 1 as shown in Figure 3(c).

3.3. *Address generator for accessing data for a linear-phase FIR kernel*

Consider a case of linear-phase FIR filter with even number of coefficients $h(n)$ that are symmetric. For example, let $N=6$, then:

$$h(0) = h(5), h(1) = h(4) \text{ and } h(2) = h(3)$$

Hence, $y(n)$ can be written as

$$y(n) = [x(n) + x(n-5)]h(0) + [x(n-1) + x(n-4)]h(1) + [x(n-2) + x(n-3)]h(2)$$

A novel algorithm for generating the appropriate sequence of addresses to fetch the data has been developed. The corresponding hardware has been implemented and tested. The schematic representation of the hardware is shown in Figure 5(a) and the corresponding algorithm can be summarised as shown in Figure 5(b).

3.4. *Address generators for ME using block-matching technique kernel*

In the case of video there will be little movement of objects (including the background) between adjacent frames. An object in the previous frame tends to get displaced/rotated in

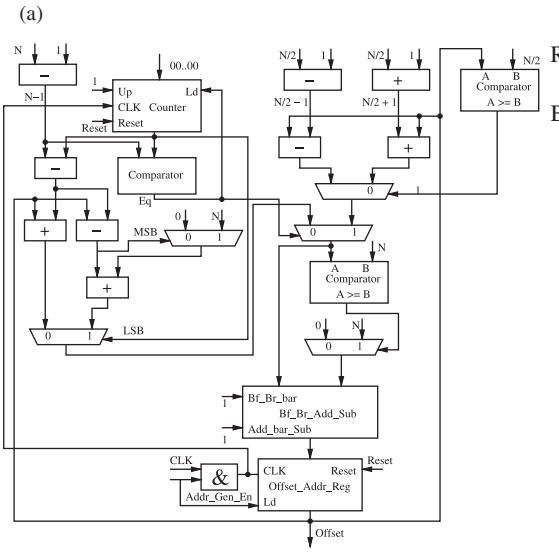

Figure 5. (a) Hardware schematic representation and (b) algorithm for data fetch AGU for linear-phase FIR filter kernel AGU – streaming data.

the next frame by a small amount. Practically there may not be much of a difference between the frames. Frames may contain multiple slices and the slices can be divided into smaller blocks called Macro-Blocks (MB). If a macro-block in the given slice in the current frame is compared with the corresponding macro-block or in the neighbourhood of the corresponding macro-block in the prior frame, the difference will normally be very small. The search area for macro-block match is restricted to the search-parameter p pixels around the macro-block in a slice in the previous frame. Searching for such a block for which the cost function is the least and within a prescribed limit and computing the displacement of the block from the previous to the current frame is the goal of ME. This operation needs to be done for all blocks in the frame and is a computationally expensive operation. Motion detection over longer distance needs larger value of the search-parameter and results in an exponential increase of computational complexity. Popular cost functions are Sum of Absolute Difference (SAD), Mean Absolute Difference (MAD) and Mean Square Error (MSE). The MSE as a cost function is computationally more intensive compared to MAD. The datapath for computing SAD is shown in Figure 6(a).

The SAD computation kernel for an $N \times M$-sized macro-block is executed in $(N \times M + 2)$ clock cycles, where the additional two clock cycles correspond to the initialisation and write latency of the last result.

3.4.1. *Address generator for fetching data for ME using block-matching technique*

The AGU assumes that the data of the current and the reference slices are kept in the memory as rows. The width and the height of the macro-block (mb_wd, mb_ht) and the width of slice (sl_wd) are given. The AGU uses two up-counters to maintain the row and

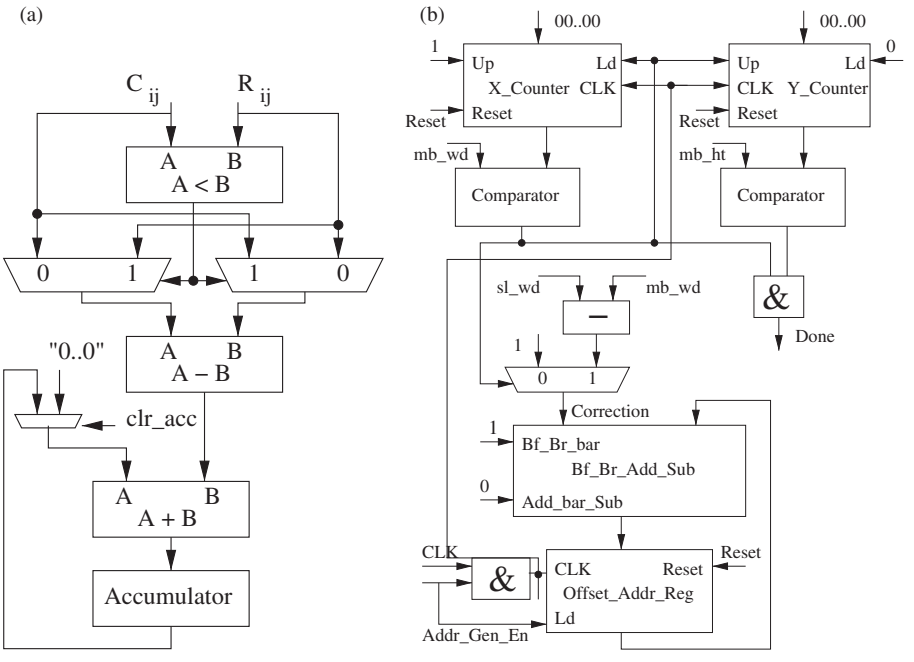

Figure 6. ME using block-matching technique: (a) data-path for SAD computation; (b) hardware schematic representation of AGU for data fetch.

column counts. The block schematic representation of the hardware implementation is shown in Figure 6(b).

3.5. *Zig-zag address generation for accessing $N \times N$ pixel array*

JPEG uses entropy coding for compressing the data after performing DCT and quantisation. After computing the 2D – DCT of an $N \times N$ image, it can be seen that the significant coefficients are present in the top-left corner of the 2D matrix. For compressing the coefficients further, it is necessary to process only these coefficients.

Entropy coding requires the quantised data of $N \times N$ pixel array to be read in zig-zag fashion as shown by the sequence of arrows in Figure 7(a). An algorithm has been developed to generate this address sequence for any value of $N \times N$ (N being even). The hardware has been developed, simulated, tested and the block schematic is shown in Figure 7(b). Cond1 to Cond7 check if the counter value pair has reached the boundary of the pixel array map, and hence a change of direction in scanning is required.

The algorithms have been implemented in VHDL in a fully structured coding style. The data width and the address width are parameterisable. The coding completely adheres to structural style and the algorithm is using components that scale linearly in terms of the complexity of the number of transistors in the hardware. All the algorithms have been simulated and tested. Structured approach of the coding helps in identifying common components used across various addressing modes, like counters, shifters, adder

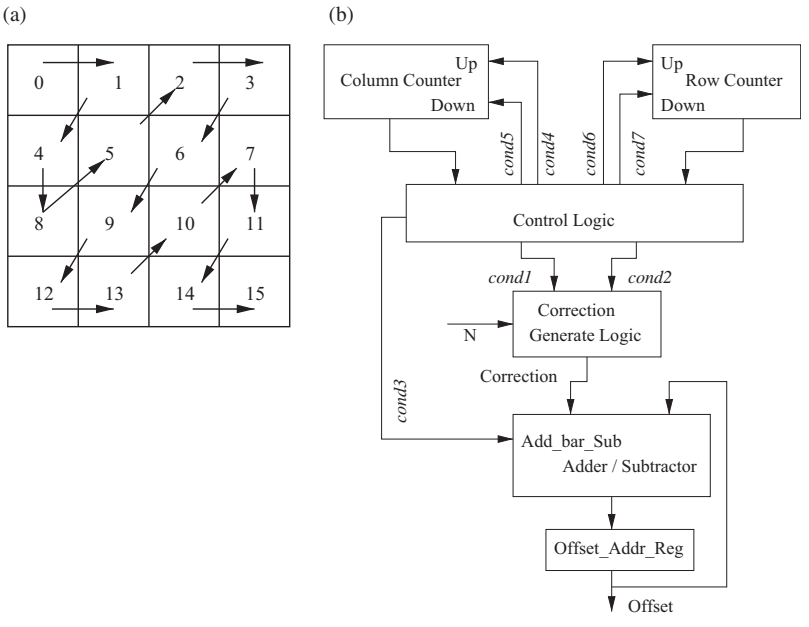

Figure 7. (a) Sequence of addresses and (b) hardware schematic representation of AGU for zig-zag addressing mode.

and comparators. A CAGU that can support 11 addressing modes listed earlier has been designed using up/down counters cum shifters, accumulator, comparators and little glue logic.

4. ASIC implementation

The CAGU and the DRDP have been implemented as ASICs. Standard cell library from *Faraday* based on *UMC* 0.18 μm with six metal layers process was used for implementation. The synthesis of CAGU and DRDP were done with the following constraints:

set_clock_uncertainty 1.0 ns
set_output_delay -max 1.0 ns
set_input_delay -max 1.0 ns
Synthesis effort/optimisation : Medium

4.1. *ASIC Implementation of the CAGU*

Since most of the kernels datapath would involve at least one MAC unit whose datapath delay is going to be considerable, the timing of the CAGU need not be very aggressive and a clock of 10 ns period was assumed. Keeping this in mind, ripple-carry adders were used in the CAGU. They are slower but occupy less space.

The CAGU was synthesised, placed and routed. The post-synthesis report is summarised in Table 2. The snapshot of the layout is shown in Figure 8(a). It was observed that if the clock period for a given address width is increased from the specified value in Table 2, the positive slack of the design would increase. The number of cells used,

Table 2. Synthesis report of the CAGU for various address widths.

Criteria/address width (clk period)	8 bit (6ns)	16 bit (8ns)	24 bit (10ns)
Cells used	1016	1975	2948
Area in μm^2	21,802	43,019	64,202
Leakage power in nW	39.46	77.84	117.46
Dynamic power in mW	2.95	5.43	8.65
Slack in ps	25	296	238

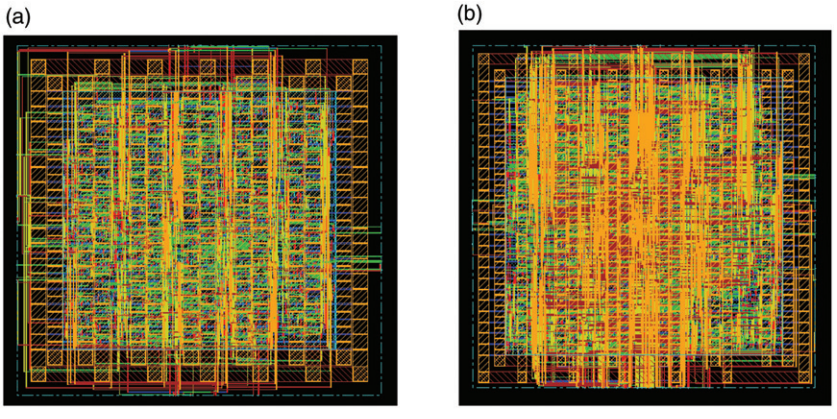

Figure 8. Snapshots of the layouts: (a) 8 bit wide CAGU; (b) 8 bit wide DRDP.

area occupied and power would reduce indicating that at lower clock frequencies we can have a much compact CAGU that consumes less power. The functional verification was carried out on the extracted netlist and the CAGU is functioning as desired.

The CAGU chip was fabricated at IMEC Belgium through Europractice. The CAGU is one of the three designs co-located on a multi-design chip (QFN48 package). There were two other designs (one digital and the other mixed signal) that shared the silicon and the pins of the chip. The total die size was 1.5 mm × 1.5 mm and the chip had 48 pins. There were constraints on the number of pins available for each design. The two digital designs shared some of the pins between them and the data-pins were de-multiplexed at the inputs and multiplexed at the outputs.

A common PCB with the required power supply was designed for testing purposes and the pins were brought out on connectors. Xilinx Virtex-5 FPGA kit was used to generate the control signals required by the CAGU (which otherwise would have been generated by the RFU under the control of the embedded controller processor). The output of the CAGU were observed on a Mixed Signal Oscilloscope as well as routed to the host computer. The fabricated chip was tested and the results are promising.

4.2. *ASIC implementation of the DRDP*

A DRDP suitable for an effective execution of a set of selected DSP kernels has been developed. The DRDP consists of two 16-bit adder/subtractors, two 8 × 8 multiplier units,

Table 3. Synthesis report of the DRDP.

Parameter	Clock period = 24 ns
Cells used	2379
Area in μm^2	61,937
Leakage power in nW	103.60
Dynamic power in mW	14.65
Slack in ns	1.7

Table 4. Timing details of various Kernel execution.

Kernel	Number of clocks	Overhead (Initialisation and latency)
FFT N-point	$N\log_2 N$	4
Convolution: (i) Stored data	$(N+M-1)M$	3
(ii) Streaming data	$(N+M-1)M$	3
ME: SAD Computation $N \times M$ macro-block	$N \times M$	2

one accumulator register, one 16×16 barrel shifter, one magnitude comparator and four register files each with four registers. The synthesis was performed with a clock period constraint of 24 ns. The post-synthesis observations are tabulated in Table 3. Back-end design of the DRDP has been done and Figure 8(b) shows the layout of the DRDP. The post-layout extracted netlist has been verified and simulated.

The implementation process was carried out using Cadence Tool Suite. Functional simulation and post-extraction behavioural simulation were carried out with the ModelSim tool.

5. Results and conclusion

Efficient address generation algorithms and hardware suitable for speeding up the execution of DSP kernels using DRDPs have been developed, implemented and tested. Timing details of the various kernel execution simulations is shown in Table 4. These results prove the efficacy of the AGUs developed, the ability to synchronise the data access and computation of result using the DRDPs as in the case of an 8-point DIF FFT kernel, convolution kernel and SAD computation in ME kernel.

5.1. Conclusion

The concept proposed demonstrates the utility of dedicated AGUs and proves the following:

- Computation of one address per clock cycle per AGU.
- CAGU can be configured to generate address sequence over the entire DSP kernel without any intervention of any kind during the execution of the kernel.

- With the help of these AGUs and suitable reconfigurable datapath the innermost loop of most of the chosen DSP kernels can be executed in one clock period. A FFT butterfly operation is completed in two clock cycles.

Acknowledgements

The development, fabrication and testing of the chip was supported by the Ministry of Communication and Information Technology, Government of India, under Special Man-power Development Program in VLSI (SMDP – Phase II).

References

Banerjee, A., Dhar, A.S., and Banerjee, S. (2001), 'FPGA Realization of a CORDIC-based FFT Processor for Biomedical Signal-processing', *Microprocessors and Microsystems*, 25, 131–142.

Cardoso, J.M.P. (2004), 'Self-loop Pipelining and Reconfigurable Dataflow Arrays', in *International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS IV)*, Samos: Springer Verlag, pp. 234–243.

Compton, K., and Hauck, S. (2002), 'Reconfigurable Computing: A Survey of Systems and Software', *ACM Computing Surveys*, 34, 171–210.

DeHon, A. (2008), 'Compute Models and System Architectures', in *Reconfigurable Computing – The Theory and Practice of FPGA-based Computation*, eds. S. Hauck, and A. DeHon, Burlington, MA 01803-4255: Morgan Kauffmann, Chap. 5, pp. 91–127.

Evans, D. (1989), 'A Second Improved Digit-reversal Permutation Algorithm for Fast Transforms', *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37, 1288–1291.

Gupta, S., Gupta, R.K., Dutt, N.D., and Nicolau, A. (2004a), 'Coordinated Parallelizing Compiler Optimizations and High-level Synthesis', *ACM Transaction on Design Automomatic Electronic Systems*, 9, 441–470.

Gupta, S., Gupta, R.K., Dutt, N.D., and Nicolau, A. (2004b), *SPARK – A Parallelizing Approach to The High-level Synthesis of Digital Circuits*, Boston: Kluwer Academic Publishers.

Harley, T., and Maheshwaramurthy, G. (2004), 'Address Generators for Mapping Arrays in Bit-reversed Order', *IEEE Transactions on Signal Processing*, 52, 1693–1703.

Hartenstein, R. (2001), 'Coarse Grain Reconfigurable Architecture (embedded tutorial)', in *ASP-DAC'01: Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, Yokohama, Japan, New York, ACM, pp. 564–570.

Huang, Z., and Malik, S. (2002), 'Exploiting Operation Level Parallelism Through Dynamically Reconfigurable Datapaths', in *DAC'02: Proceedings of the 39th Annual Design Automation Conference*, New Orleans, Louisiana, USA, New York, ACM, pp. 337–342.

Hulina, P., Coraor, L., Kurian, L., and John, E. (1995), 'Design and VLSI Implementation of an Address Generation Coprocessor', *IEE Proceedings – Computers and Digital Techniques*, 142, 145–151.

Kini, R.M., and Sumam, D.S. (2009), *Comprehensive Address Generator for Digital Signal Processing, Fourth International Conference on Indusrial and Information Systems*, ICIIS 2009, 28–31 Dec, pp. 325–330.

Lysecky, R., Stitt, G., and Vahid, F. (2006), 'Warp Processors', *ACM Transactions on Design Automation of Electronic Systems*, 11, 659–681.

Miranda, M.A., Catthoor, F.V.M., Janssen, M., and De Man, H.J. (1998), 'Highlevel Address Optimization and Synthesis Techniques for Data-transfer-intensive Applications', *IEEE Transactions on Very Large Scale Integration Systems*, 6, 677–686.

Nwachukwu, E. (1985), 'Address Generation in an Array Processor', *IEEE Transactions on Computers*, C-34, 170–173.

Rodriguez, J. (1989), 'An Improved FFT Digit-reversal Algorithm', *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37, 1298–1300.

Smit, G.J.M., Kokkeler, A.B.J., Wolkotte, P.T., Hölzenspies, P.K.F., van de Burgwal, M.D., and Heysters, P.M. (2007), 'The Chameleon Architecture for Streaming DSP Applications', *EURASIP Journal of Embedded Systems*, 2007, 11–20.

Tessier, R., and Burleson, W. (2000), 'Reconfigurable Computing for Digital Signal Processing: A Survey', *Journal of VLSI Signal Processing*, 28, 7–27.

Walker, J. (1990), 'A New Bit Reversal Algorithm', *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38, 1472–1473.

Yong, A. (1991), 'A Better FFT Bit-reversal Algorithm Without Tables', *IEEE Transactions on Signal Processing*, 39, 2365–2367.